Algorithms CS 5312

Principle Component Analysis

Imdad Ullah Khan

Contents

1 Dimensionality Reduction **2**
1.1 Aims of PCA . . . 2

2 Differences between PCA and JL-Transform **3**
2.1 Data-driven vs. Data Oblivious dimensionality reduction . . . 3
2.2 Randomized vs Deterministic . . . 3
2.3 Preservation of pairwise distances . . . 3
2.4 Meaningful coordinates in the compressed space . . . 3
2.5 Dimensions of reduced space . . . 3

3 Principal Component Analysis **4**
3.1 Zero-Centering the data and uniform scaling all coordinates . . . 4
3.1.1 Covariance, Correlation and the Variance-Covariance matrix . . . 4
3.2 PCA: The Idea . . . 5
3.2.1 Projections . . . 5
3.2.2 A bigger example . . . 8
3.2.3 Choosing the best direction for projecting: : Minimum Reconstruction Error 11
3.2.4 Direction with Min Reconstruction Error := Direction with Max Variance . 12
3.2.5 Subsequent projections: Larger k . . . 14
3.3 PCA: Linear Algebraic Formulation . . . 15
3.4 Diagonal Covariance matrix . . . 15
3.4.1 Eigenbases, Diagonalization, Eigen Decomposition . . . 16
3.4.2 Eigen Decomposition . . . 17
3.4.3 Real Symmetric Matrices . . . 18
3.4.4 Orthogonal Matrices . . . 18
3.4.5 Eigendecomposition of covariance matrix . . . 19
3.5 PCA: The Algorithm . . . 19
3.5.1 Complexity of the Algorithm . . . 20
3.5.2 Number of principal components needed: Scree Plot . . . 21
3.5.3 The Power Iteration method to compute eigen vectors . . . 21

4 PCA: Case Studies and Examples **24**
4.1 Images as Vectors . . . 27

5 Limitations of PCA and Other feature extraction and dimensionality reduction methods **31**

1 Dimensionality Reduction

Given a dataset described by a set of (generally many) features, we aim to generate another set of (generally much fewer) features that best describes the data (with as low distortion as possible). As we discussed in the curse of dimensionality lecture, that too many features generally degrade the quality of analytics and at least the computational cost of working with high dimensional data is very high.

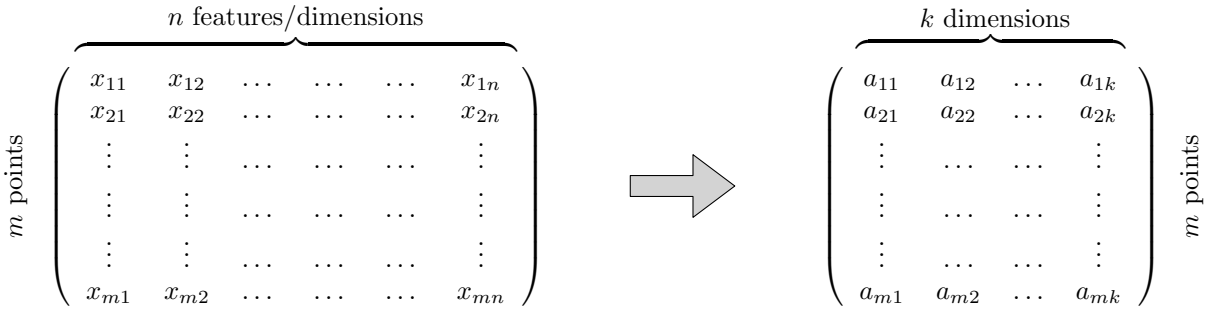


Figure 1: Schematic view of dimensionality reduction

In this course we will study many techniques for dimensionality reduction, namely, the Johnson-Lindenstrauss transform and (it's variations), the AMS transform (that is originally meant for something different), Locality Sensitive Hashing, and Principal Component Analysis.

The specific definition and measure of distortion depends on the given data set and the target application. The Johnson-Lindenstrauss lemma was meant to (almost) preserve the Euclidean distance between every pair

In this lecture, we study another widely used method called principal component analysis for dimensionality reduction.

1.1 Aims of PCA

- Low dimensional data visualization
- Get a sense of source of variations in data
- Understand pairwise correlation between attributes of data
- Reduce dimensions with little *'distortion'*

Visualizing high dimensional data (using 1 axis per attribute) cannot be performed for more than 3 dimensions. PCA brings to forth a low dimensional picture, that shows the data from a

"more informative" view point. The operations of PCA reveal the internal structure of the data in a way that best explains the variance in the data.

2 Differences between PCA and JL-Transform

2.1 Data-driven vs. Data Oblivious dimensionality reduction

Recall that the Johnson-Lindenstrauss transform is oblivious to data, i.e. it compresses each data point without looking at data at all. The main advantage of this approach is its computational efficiency. However, for the same reason it has a disadvantage since it does not exploit the structure of data.

PCA, on the hand, is a data-driven method, it exploits the structure of data to reduce its dimensionality. This improves the quality of the data, in the sense that it preserve the structure of data better, but it has higher computational cost.

2.2 Randomized vs Deterministic

Recall that the JL transform a data point $x = (x_1, x_2, \ldots, x_n)$ to the compressed version $y = (\langle x, r_1\rangle, \langle x, r_2\rangle, \ldots, \langle x, r_k\rangle)$, where for $1 \leq i \leq k$, r_i is a random unit vector in $\mathbb{R}^n$ (i.e. n independent random numbers from $\mathcal{N}(0, 1)$ (normalized to have length 1)).

PCA also projects each data points onto k vectors. But these k vectors are chosen based on the given dataset. Hence, this is a deterministic process, will always produce the same results for the same dataset and the number of reduced dimensions k.

2.3 Preservation of pairwise distances

JL transforms (probabilistically) guarantees that pairwise Euclidean distances between compressed data points are (approximately) preserved. PCA offers no such guarantee about pairwise distances between the compressed data points

2.4 Meaningful coordinates in the compressed space

The dimensions of the reduced space (the k unit vectors) in the case of JL-method have no intrinsic meaning, while those used in PCA are often very meaningful as they are derived for data. Consider the eigen-faces example

2.5 Dimensions of reduced space

Recall that the guarantees of the JL-method required that $k = \Omega(\log N)$, where N is the number of data points. Since in big data era, this N is generally very large, we conclude that JL generally gives good results (pairwise distance preservation) only when k is large (usually in low hundreds).

PCA on the other hand, **can** give very "good" results even when k is 1, 2, or 3. Indeed, if the purpose of PCA is to meaningfully visualize data, then the dimensions have to be at most 2 or maximum 3.

3 Principal Component Analysis

In this section, we will discuss the overall idea of PCA and its etymology. Later we will discuss the linear algebra underlying PCA followed by the algorithmic details and some case studies. We will also discuss some cases where PCA does not work as well.

Let our dataset be a set of m points $X \subset \mathbb{R}^n$ (each point is a n-dimensional vector).

3.1 Zero-Centering the data and uniform scaling all coordinates

Throughout we will assume that the dataset is zero-centered, i.e. it's mean (centroid), $\sum_{i=1}^{m} x_i$ is $\mathbf{0}$. This can be enforced by subtracting the mean vector $\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$ from every data point in X. This shifting the data to center around origin does not change the positions of the point relative to each other, in the sense that ranks of points in every dimension remain the same. For example, in 2d the highest and lowest point, leftmost and rightmost point, middle point vertically and horizontally remain the same. This is very important as it keeps the required linear algebra very simple and much cleaner.

PCA like many other data analytics processes is very sensitive to units or scales of coordinates. Therefore, we will assume that all coordinates of X are in some uniform scale, if not we can enforce it by scaling it down (the best in this case will be z-score normalization). Recall that z-score normalization or standardization of a variable u_i is given by

$$u'_{ji} = \frac{u_{ji} - \bar{u_i}}{\sigma_i},$$

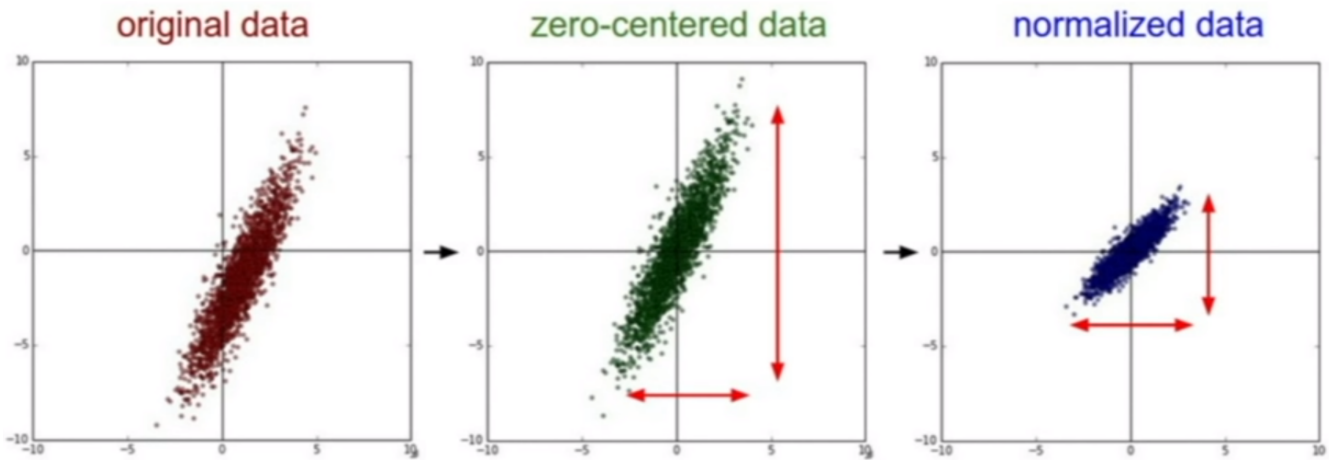
where $\bar{u_i}$ is the mean and σ_i is the standard deviation of the variable u_i. It is easy to see that the resultant variable u_i has mean 0 and variance and standard deviation equal to 1.

3.1.1 Covariance, Correlation and the Variance-Covariance matrix

Recall that the variance in a set of data points or variance of a variable (in our case a column of the matrix) is the most commonly used measure of spread in the data. It is the average squared distance from the mean, formally for a variable u_i (the ith column), its variance is given by $\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}(x_{ji} - \bar{u_i})^2$, where $\bar{u_i}$ is the mean of variable u_i.

The covariance between two variables u_i and u_j is a measure of how the two variables vary with each other. It is the degree to which u_i and u_j are linearly correlated. Formally, it is given by

$$Cov(u_i, u_j) = Cov(i, j) = C_{ij} = \frac{1}{m}\sum_{l=1}^{m}(x_{li} - \bar{u_i})(x_{lj} - \bar{u_j})$$



Using pairwise covariance among many variables, makes sense only if they are in the same scale (and units). The correlation (aka Pearson Correlation) is essentially a normalized version of covariance. Correlation between two variables u_i and u_j is given by

$$r_{ij} = \frac{Cov(u_i, u_j)}{\sigma_i \sigma_j}$$

i.e. it is the covariance divided by the product of standard deviations. It is easy to see that for all i, j we have $-1 \leq r_{ij} \leq 1$.

Notice that when the variables u_i and u_j are standardized, then since the standard deviations of both are 1, covariance is equal to the correlation.

The **Variance-Covariance Matrix** (aka covariance matrix) of a n-dimensional data sets is a $n \times n$ matrix C, where the (i, j)th entry equal to $Cov(i, j)$. For standardized dataset, all entries on the principal diagonal (the variances) are 1, while the off diagonal entries are the pairwise correlations. Note that this is a symmetric matrix. Suppose each $x_i \in X$ is a document in the bag of words model (each column represents a word in vocabulary and the (i, j)th entry of X is a the frequency of jth word in x_i). Then the dot-product of column j and column l, i.e. value of $X^T X(j, l)$ is a measure of how frequently the jth and lth words co-occur in the documents.

For a standardized dataset the variance-covariance matrix C is given by $C = X^T X$.

3.2 PCA: The Idea

3.2.1 Projections

The overall idea of PCA is projecting the dataset onto a line or a lower dimensional subspace, and recording the projection vectors.

Assume that dataset is a set of m vectors of dimensions n, i.e. $X \subset \mathbb{R}^n$ and $|X| = m$. We store the data and pictures it as a $m \times n$ matrix (each row is a data point) and each column is a dimension (attribute/variable). Suppose names of the original variables are $\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n}$. Note that we think of them as standard bases vectors of $\mathbb{R}^n$. In other words, each data point is

$$X = \begin{array}{c} \overbrace{\phantom{\mathbf{u_1}\ \mathbf{u_2}\ \cdots\ \mathbf{u_m}}}^{m \text{ features/dimensions}} \\ \mathbf{u_1}\ \ \mathbf{u_2}\ \ \cdots\ \ \mathbf{u_m} \\ \begin{pmatrix} x_{11} & x_{12} & \dots & \dots & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & \dots & \dots & x_{2m} \\ \vdots & \vdots & \dots & \dots & \dots & \vdots \\ \vdots & \vdots & \dots & \dots & \dots & \vdots \\ \vdots & \vdots & \dots & \dots & \dots & \vdots \\ x_{n1} & x_{n2} & \dots & \dots & \dots & x_{nm} \end{pmatrix} \end{array} \begin{array}{l} \\ \\ n \text{ points} \end{array}$$

$$C \ \ or \ \ \Sigma = \begin{array}{c} \overbrace{}^{\text{m}} \\ \begin{pmatrix} Cov(u_1,u_1) & Cov(u_1,u_2) & \dots & Cov(u_1,u_m) \\ Cov(u_2,u_1) & Cov(u_2,u_2) & \dots & Cov(u_2,u_m) \\ \\ \vdots & \vdots & \dots & \vdots \\ \\ \vdots & \vdots & \dots & \vdots \\ \\ \vdots & \vdots & \dots & \vdots \\ \\ \vdots & \vdots & \dots & \vdots \\ \\ Cov(u_m,u_1) & Cov(u_m,u_2) & \dots & Cov(u_m,u_m) \end{pmatrix} \end{array} \begin{array}{l} \\ \\ m \end{array}$$

a linear combination of $\mathbf{u_1}, \dots, \mathbf{u_n}$ and the coordinate $(x_{i1}, x_{i2}, \dots, x_{in})$ of x_i are the coefficients of that linear combination, i.e.

$$x_i = x_{i1}\mathbf{u_1} + x_{i2}\mathbf{u_2} + \dots + x_{in}\mathbf{u_n}$$

We would like to project X onto a k-dimenisonal subspace spanned by (bases) vectors $\mathbf{v_1}, \dots, \mathbf{v_k} \subset \mathbb{R}^n$. In other words we would like to express each point x_i in reduced dimensional space as linear combination of $\mathbf{v_1}, \dots, \mathbf{v_k}$ Let x_i' be the reduced dimensional version of x_i, then we want

$$x_i' = a_{i1}\mathbf{v_1} + a_{i2}\mathbf{v_2} + \dots + a_{ik}\mathbf{v_k},$$

where for $j = 1$ to k $a_{ij} = \langle x_i, \mathbf{v_j} \rangle = x_i \cdot \mathbf{v_j}$

$$\begin{array}{c} \overbrace{\phantom{\mathbf{u_1}\ \mathbf{u_2}\ \cdots\ \mathbf{u_m}}}^{m \text{ dimensions}} \\ \mathbf{u_1}\ \ \mathbf{u_2}\ \ \cdots\ \ \mathbf{u_m} \\ \begin{pmatrix} x_{11} & x_{12} & \dots & \dots & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & \dots & \dots & x_{2m} \\ \vdots & \vdots & \dots & \dots & \dots & \vdots \\ \vdots & \vdots & \dots & \dots & \dots & \vdots \\ \vdots & \vdots & \dots & \dots & \dots & \vdots \\ x_{n1} & x_{n2} & \dots & \dots & \dots & x_{nm} \end{pmatrix} \end{array} \begin{array}{l} \\ \\ n \text{ points} \end{array} \quad \Rightarrow \quad \begin{array}{c} \overbrace{\phantom{\mathbf{v_1}\ \mathbf{v_2}\ \cdots\ \mathbf{v_k}}}^{k \text{ dimensions}} \\ \mathbf{v_1}\ \ \mathbf{v_2}\ \ \cdots\ \ \mathbf{v_k} \\ \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \vdots & \dots & \dots & \vdots \\ \vdots & \dots & \dots & \vdots \\ \vdots & \dots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nk} \end{pmatrix} \end{array} \begin{array}{l} \\ \\ n \text{ points} \end{array}$$

The goal of PCA is to find these vectors $\mathbf{v_1}, \dots, \mathbf{v_k} \subset \mathbb{R}^n$. Since we want them to span a k-dimensional subspace, we would require them to be linearly independent. Actually, they should be mutually orthogonal and of unit lengths (so orthonormal). We want to lose "as little information" as possible, we will formulate the objective (goodness measure) for finding $\mathbf{v_1}, \dots, \mathbf{v_k}$.

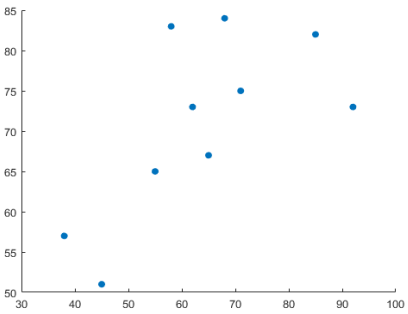
Space Saving Keep in mind the following made-up example to quantify the space saving by reducing dimensions. Suppose we have $1m$ vectors in dimensions $1k$. If storing a real number takes 4 bytes, then the space required to store matrix X is $4b$ bytes. If $k = 10$, then the space

required to store the matrix X' is $1m \times 10 \times 4$ bytes (for the values $a_{..}$) plus the $10 \times n \times 4$ bytes (for the vectors $\mathbf{v_1}, \ldots, \mathbf{v_k}$).

For now let's assume that the value $1 \leq k \leq n$ is given as input. Later we will discuss some rule of thumbs for what would be the best k.

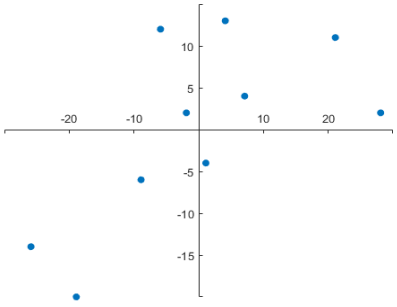
Example with $k = 1$: Suppose we have records of 10 students' performance in two courses. We want the resultant space to be only one-dimensional (just one number to represent each student or one vector $\mathbf{v_1}$). In other words, we want projection onto a line.

	$\mathbf{c}_1$	$\mathbf{c}_2$
s_1	92	73
s_2	85	82
s_3	71	75
s_4	68	84
s_5	65	67
s_6	62	73
s_7	58	83
s_8	45	51
s_9	38	57
s_{10}	55	65



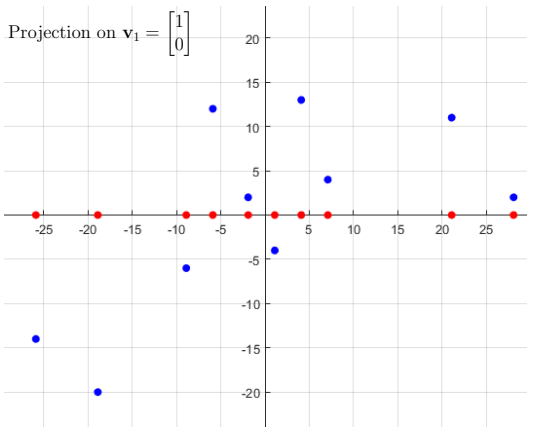
Raw Data

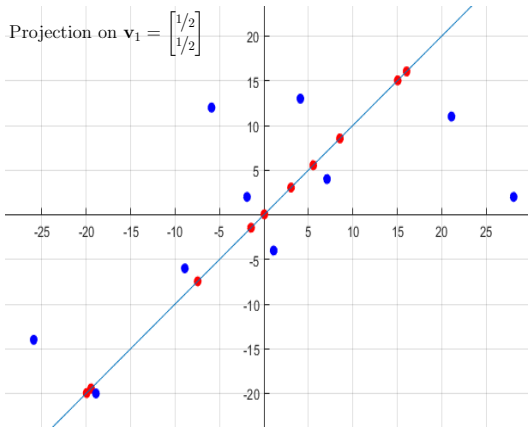
	$\mathbf{c}_1$	$\mathbf{c}_2$
s_1	28.1	2
s_2	21.1	11
s_3	7.1	4
s_4	4.1	13
s_5	1.1	−4
s_6	−1.9	2
s_7	−5.9	12
s_8	−18.9	−20
s_9	−25.9	−14
s_{10}	−8.9	−6



Zero-Centered Data

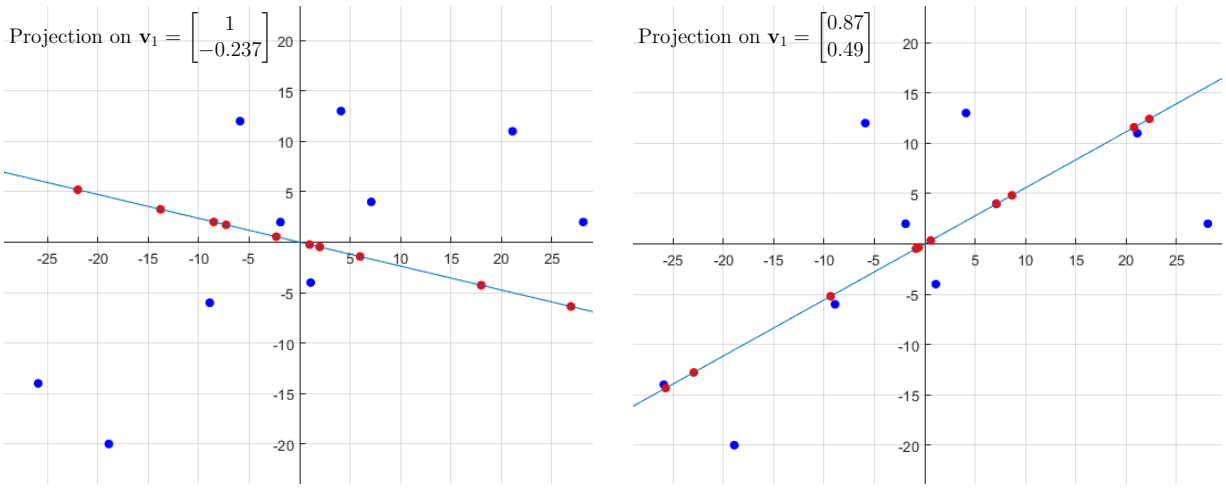
Feature Selection: We can just pick one vector from $\mathbf{u_1}, \ldots, \mathbf{u_n}$ as the basis. This could the vector $c_1 = \begin{bmatrix}1 & 0\end{bmatrix}^T$ or $c_2 = \begin{bmatrix}0 & 1\end{bmatrix}^T$. Projecting onto these any of these vectors means just keeping the corresponding component and dropping the remaining coordinate(s). In the following figure we show the projection on the vector $\begin{bmatrix}1 & 0\end{bmatrix}^T$. Each blue point original is now represented by its projection onto the horizontal axis (red points).





Average: We can project onto the vector $\mathbf{v_1} = \begin{bmatrix}1/2 & 1/2\end{bmatrix}^T$. Hence for instance, the coordinates of student s_1 (the value of a_{11}) will be $\langle s_1, \mathbf{v_1} \rangle = 92/2 + 73/2 = 82.5$ (the average value). It should be clear that we can achieve weighted averages via projecting onto appropriately chosen vectors.

Random Projection: We can also project the data onto a random direction, as we did in the Johnson-Lindenstrauss transform. The following is an example projection onto the random vector $\begin{bmatrix}1 & -0.2369\end{bmatrix}^T$.



Optimal Projection: PCA projects the data onto the so-called principal components, that in this case would the "best vector" for projection. We will define the goodness measure and discuss how to find them. It turns out that the best vector for projection onto is $\begin{bmatrix}0.8736 & 0.4867\end{bmatrix}^T$

3.2.2 A bigger example

Consider the following data

	c_1	c_2	c_3	c_4
s_1	95	89	70	64
s_2	91	91	71	70
s_3	79	77	65	58
s_4	76	74	68	69
s_5	76	69	65	64
s_6	78	68	65	64
s_7	79	70	47	42
s_8	62	61	47	46
s_9	68	63	88	88
s_{10}	68	67	90	89
s_{11}	66	63	82	75
s_{12}	66	67	78	70
s_{13}	68	63	75	72
s_{14}	64	63	76	70
s_{15}	53	46	79	72
s_{16}	43	42	61	60

	c_1	c_2	c_3	c_4
s_1	24.3	21.9	-0.4	-3.1
s_2	20.3	23.9	0.6	2.9
s_3	8.3	9.9	-5.4	-9.1
s_4	5.3	6.9	-2.4	1.9
s_5	5.3	1.9	-5.4	-3.1
s_6	7.3	0.9	-5.4	-3.1
s_7	8.3	2.9	-23.4	-25.1
s_8	-8.8	-6.1	-23.4	-21.1
s_9	-2.8	-4.1	17.6	20.9
s_{10}	-2.8	-0.1	19.6	21.9
s_{11}	-4.8	-4.1	11.6	7.9
s_{12}	-4.8	-0.1	7.6	2.9
s_{13}	-2.8	-4.1	4.6	4.9
s_{14}	-6.8	-4.1	5.6	2.9
s_{15}	-17.8	-21.1	8.6	4.9
s_{16}	-27.8	-25.1	-9.4	-7.1

Suppose we project this data onto the vector $\mathbf{v_1} = \begin{bmatrix}0.6 & 0.6 & -0.4 & -0.4\end{bmatrix}^T$. For instance for s_1 as the one coordinate (in the basis $\mathbf{v_1}$), we can save $\langle s_1, \mathbf{v_1}\rangle = 24.3 * .6 + 21.9 * .6 + -0.4 * -.4 + -3.1 * -.4 = 28.7$. Similarly, the vector for s_1 in the bases system ($\mathbf{v_1}$) is given by

$$s_1' = \langle s_1, \mathbf{v_1}\rangle \mathbf{v_1} = 28.7 * \begin{bmatrix}0.6 & 0.6 & -0.4 & -.0.4\end{bmatrix}^T = \begin{bmatrix}17.5 & 16.5 & -11.2 & -11.1\end{bmatrix}^T$$

For all vectors there representation is given in the following table.

	$a_{i1} = \langle s_\cdot, \mathbf{v_1}\rangle$
s_1	28.7
s_2	24.7
s_3	16.3
s_4	7.4
s_5	7.6
s_6	8.2
s_7	25.5
s_8	8.4
s_9	-18.9
s_{10}	-17.8
s_{11}	-12.8
s_{12}	-7
s_{13}	-7.7
s_{14}	-9.7
s_{15}	-28.1
s_{16}	-24.9

	$\langle s_\cdot, \mathbf{v_1}\rangle \mathbf{v_1}$			
s_1'	17.5	16.5	-11.2	-11.1
s_2'	15	14.2	-9.6	-9.5
s_3'	9.9	9.4	-6.3	-6.3
s_4'	4.5	4.2	-2.9	-2.8
s_5'	4.6	4.4	-3	-2.9
s_6'	5	4.7	-3.2	-3.2
s_7'	15.5	14.6	-9.9	-9.9
s_8'	5.1	4.8	-3.3	-3.3
s_9'	-11.5	-10.8	7.4	7.3
s_{10}'	-10.8	-10.2	6.9	6.9
s_{11}'	-7.8	-7.3	5	4.9
s_{12}'	-4.3	-4	2.7	2.7
s_{13}'	-4.7	-4.4	3	3
s_{14}'	-5.9	-5.6	3.8	3.8
s_{15}'	-17.1	-16.1	10.9	10.9
s_{16}'	-15.1	-14.3	9.7	9.6

	c_1	c_2	c_3	c_4
s_1	24.3	21.9	-0.4	-3.1
s_2	20.3	23.9	0.6	2.9
s_3	8.3	9.9	-5.4	-9.1
s_4	5.3	6.9	-2.4	1.9
s_5	5.3	1.9	-5.4	-3.1
s_6	7.3	0.9	-5.4	-3.1
s_7	8.3	2.9	-23.4	-25.1
s_8	-8.8	-6.1	-23.4	-21.1
s_9	-2.8	-4.1	17.6	20.9
s_{10}	-2.8	-0.1	19.6	21.9
s_{11}	-4.8	-4.1	11.6	7.9
s_{12}	-4.8	-0.1	7.6	2.9
s_{13}	-2.8	-4.1	4.6	4.9
s_{14}	-6.8	-4.1	5.6	2.9
s_{15}	-17.8	-21.1	8.6	4.9
s_{16}	-27.8	-25.1	-9.4	-7.1

If we compare each s_i' with the original s_i, we see that there is some sense in the data, coordinates with big values are still bigger, and the coordinates with smaller values are somewhat smaller. But this we saved 75% of storage size. Recall that we only have to store the 4 coordinates for $\mathbf{v_1}$, which does not add much when there are millions of data points.

We take it one step further, and project the data onto two dimensional subspace spanned by $\mathbf{v_1} = \begin{bmatrix}0.6 & 0.6 & -0.4 & -0.4\end{bmatrix}^T$ and $\mathbf{v_2} = \begin{bmatrix}0.4 & 0.4 & 0.6 & 0.6\end{bmatrix}^T$

	a_{i1}	a_{i2}
s_1	28.7	15.8
s_2	24.7	19.3
s_3	16.3	−1.5
s_4	7.4	4.5
s_5	7.6	−2.3
s_6	8.2	−1.9
s_7	25.5	−24.4
s_8	8.4	−32
s_9	−18.9	20.1
s_{10}	−17.8	23.5
s_{11}	−12.8	8.1
s_{12}	−7	4.4
s_{13}	−7.7	3
s_{14}	−9.7	0.9
s_{15}	−28.1	−7.1
s_{16}	−24.9	−30.2

	$a_{i1}\mathbf{v_1} + a_{i2}\mathbf{v_2}$			
s_1'	23.4	22.8	−1.8	−1.7
s_2'	22.3	21.9	1.8	1.9
s_3'	9.4	8.8	−7.2	−7.2
s_4'	6.2	6	−0.2	−0.2
s_5'	3.8	3.4	−4.3	−4.3
s_6'	4.3	4	−4.3	−4.3
s_7'	6.4	4.9	−24.3	−24.3
s_8'	−6.9	−8	−22.2	−22.2
s_9'	−4	−2.8	19.2	19.2
s_{10}'	−2	−0.8	20.8	20.8
s_{11}'	−4.7	−4.1	9.8	9.7
s_{12}'	−2.6	−2.3	5.3	5.3
s_{13}'	−3.6	−3.2	4.7	4.7
s_{14}'	−5.6	−5.2	4.3	4.3
s_{15}'	−19.8	−19	6.7	6.7
s_{16}'	−26.5	−26.4	−8.2	−8.3

	c_1	c_2	c_3	c_4
s_1	24.3	21.9	−0.4	−3.1
s_2	20.3	23.9	0.6	2.9
s_3	8.3	9.9	−5.4	−9.1
s_4	5.3	6.9	−2.4	1.9
s_5	5.3	1.9	−5.4	−3.1
s_6	7.3	0.9	−5.4	−3.1
s_7	8.3	2.9	−23.4	−25.1
s_8	−8.8	−6.1	−23.4	−21.1
s_9	−2.8	−4.1	17.6	20.9
s_{10}	−2.8	−0.1	19.6	21.9
s_{11}	−4.8	−4.1	11.6	7.9
s_{12}	−4.8	−0.1	7.6	2.9
s_{13}	−2.8	−4.1	4.6	4.9
s_{14}	−6.8	−4.1	5.6	2.9
s_{15}	−17.8	−21.1	8.6	4.9
s_{16}	−27.8	−25.1	−9.4	−7.1

Our approximations s_i's for s_i's are almost exact, we project onto a 3d space spanned by $\mathbf{v_1} = \begin{bmatrix}0.6 & 0.6 & -0.4 & -0.4\end{bmatrix}^T$, $\mathbf{v_2} = \begin{bmatrix}0.4 & 0.4 & 0.6 & 0.6\end{bmatrix}^T$, and $\mathbf{v_3} = \begin{bmatrix}-0.7 & 0.7 & 0.1 & -0.1\end{bmatrix}^T$.

	a_{i1}	a_{i2}	a_{i3}
s_1	28.7	15.8	−0.9
s_2	24.7	19.3	2.6
s_3	16.3	−1.5	2
s_4	7.4	4.5	0.8
s_5	7.6	−2.3	−2.3
s_6	8.2	−1.9	−4.4
s_7	25.5	−24.4	−2.5
s_8	8.4	−32	2.4
s_9	−18.9	20.1	−2.1
s_{10}	−17.8	23.5	0.8
s_{11}	−12.8	8.1	0.4
s_{12}	−7	4.4	3.5
s_{13}	−7.7	3	−1.2
s_{14}	−9.7	0.9	1.9
s_{15}	−28.1	−7.1	−2.5
s_{16}	−24.9	−30.2	1.5

	$a_{i1}\mathbf{v_1} + a_{i2}\mathbf{v_2} + a_{i3}\mathbf{v_3}$			
s'_1	24	22.2	−1.9	−1.6
s'_2	20.5	23.7	2	1.5
s'_3	8	10.2	−7.1	−7.5
s'_4	5.6	6.6	−0.2	−0.3
s'_5	5.4	1.8	−4.5	−4
s'_6	7.4	0.8	−4.7	−3.8
s'_7	8.1	3.1	−24.5	−24
s'_8	−8.5	−6.3	−22	−22.5
s'_9	−2.5	−4.3	19	19.5
s'_{10}	−2.6	−0.3	20.8	20.7
s'_{11}	−5	−3.8	9.8	9.7
s'_{12}	−5	0.2	5.6	4.9
s'_{13}	−2.7	−4.1	4.6	4.9
s'_{14}	−6.9	−3.9	4.5	4
s'_{15}	−18.1	−20.7	6.5	7
s'_{16}	−27.5	−25.3	−8	−8.5

3.2.3 Choosing the best direction for projecting: : Minimum Reconstruction Error

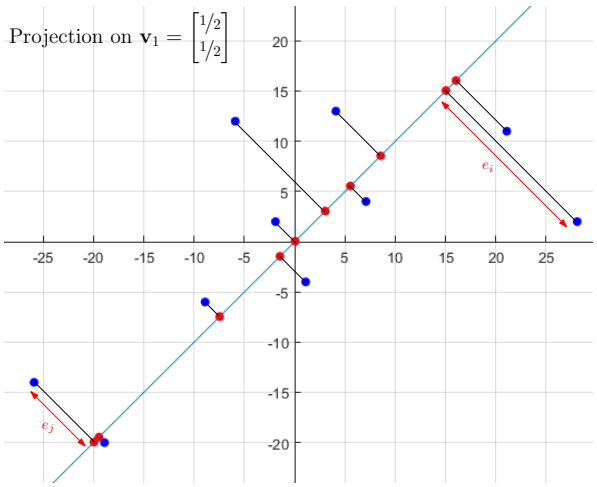
Again, for now we focus on $k = 1$, suppose we project 2d data onto a line, what is the error that we get? Suppose the 2d dataset of 10 students were projected onto the vector $\begin{bmatrix}0.5 & 0.5\end{bmatrix}^T$. Let e_i be the **perpendicular distance** between s_i and its projection s'_i, then the errors over all points is summarized in the following tables.

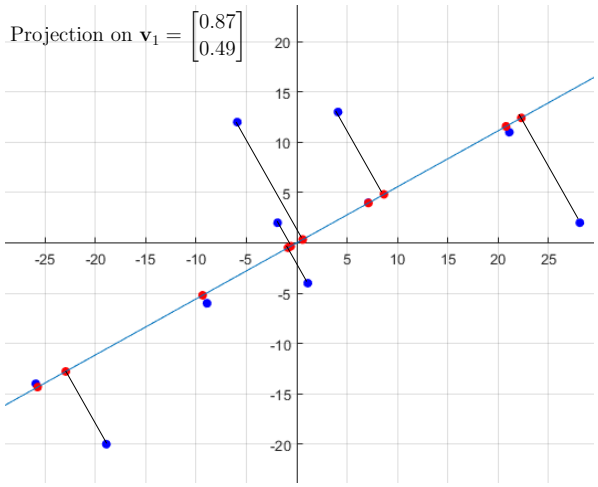
	c_1	c_2
s_1	28.1	2
s_2	21.1	11
s_3	7.1	4
s_4	4.1	13
s_5	1.1	−4
s_6	−1.9	2
s_7	−5.9	12
s_8	−18.9	−20
s_9	−25.9	−14
s_{10}	−8.9	−6

	$a_{i1}\mathbf{v_1}$	
s'_1	15	15
s'_2	16	16
s'_3	5.5	5.5
s'_4	8.5	8.5
s'_5	−1.4	−1.4
s'_6	0	0
s'_7	3	3
s'_8	−19.4	−19.4
s'_9	−19.9	−19.9
s'_{10}	−7.4	−7.4

	$\|s_i - s'_i\|$
e_1	18.5
e_2	7.1
e_3	2.2
e_4	6.3
e_5	3.6
e_6	2.8
e_7	12.7
e_8	0.8
e_9	8.4
e_{10}	2.1

Similarly, if the dataset was projected onto the vector $\begin{bmatrix}0.8736 & 0.4867\end{bmatrix}^T$. The resultant points and correposponding errors are given in the following table and also depicted in the figure above. The total (sum of) error in all points as a result of the first projection is 64.3467, while that in the second projection is 51.6030.





	c_1	c_2
s_1	28.1	2
s_2	21.1	11
s_3	7.1	4
s_4	4.1	13
s_5	1.1	-4
s_6	-1.9	2
s_7	-5.9	12
s_8	-18.9	-20
s_9	-25.9	-14
s_{10}	-8.9	-6

	$a_{i1}\mathbf{v_1}$	
s'_1	22.3	12.4
s'_2	20.8	11.6
s'_3	7.1	4
s'_4	8.7	4.8
s'_5	-0.9	-0.5
s'_6	-0.6	-0.3
s'_7	0.6	0.3
s'_8	-22.9	-12.8
s'_9	-25.7	-14.3
s'_{10}	-9.3	-5.2

	$\|s_i - s'_i\|$
e_1	11.9
e_2	0.7
e_3	0
e_4	9.4
e_5	4
e_6	2.7
e_7	13.4
e_8	8.3
e_9	0.4
e_{10}	0.9

3.2.4 Direction with Min Reconstruction Error := Direction with Max Variance

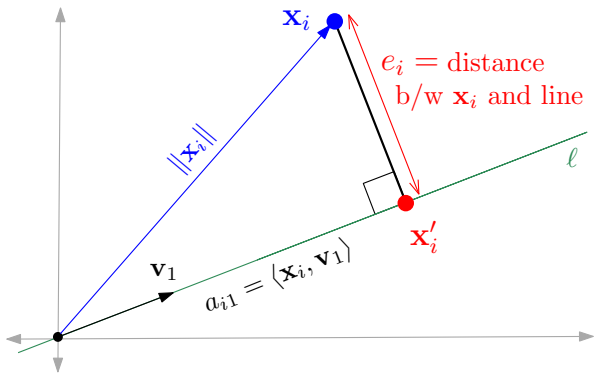
In general, our criteria for selecting the vector for projection is to find such a vector that results in the minimum reconstruction error or minimum information loss. Let X be the dataset and let $\mathbf{v_1}$ be the unit vector on which we project X, i.e. $x'_i = \langle x_i, \mathbf{v_1}\rangle \mathbf{v_1}$. Then the total reconstruction error is given by

$$\sum_{x_i \in X} e_i =: \sum_{x_i \in X} \|x_i - x'_i\| = \sum_{x_i \in X} (\text{distance b/w } x_i \text{ and line spanned by } \mathbf{v_1})$$

Note that this $(x_i - x'_i)$ is the perpendicular distance between x_i and the line spanned by $\mathbf{v_1}$ or between x_i and its projection on $\mathbf{v_1}$.

For technical reasons, we will consider the sum of squared reconstruction error, as it is easy to deal with and it has a nice connection with the variance in the data. Therefore, the optimization problem of finding the optimal vector $\mathbf{v_1}$ has the following objection function

$$\arg\min_{\mathbf{v_1}, \|\mathbf{v}\|=1} \sum_{x_i \in X} \|x_i - x'_i\|^2 := \arg\min_{\mathbf{v_1}, \|\mathbf{v}\|=1} \sum_{x_i \in X} (\text{distance b/w } x_i \text{ and line spanned by } \mathbf{v_1})^2 \qquad (1)$$



Consider the above diagram with a typical data point x_i and it's projection on the line spanned by the vector $\mathbf{v_1}$. By the Phythogorus theorem, we have

$$(\text{distance b/w } x_i \text{ and line spanned by } \mathbf{v_1})^2 = \|x_i\|^2 - \langle x_i, \mathbf{v_1} \rangle^2$$

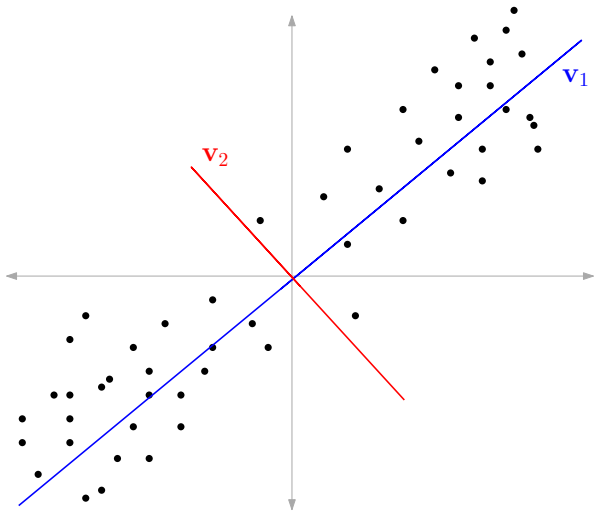
Since for a fixed x_i, $\|x_i\|^2$ is constant. Minimizing the left hand side is equivalent to maximizing $\langle x_i, \mathbf{v_1} \rangle^2$. Thus the objective function of PCA (with $k = 1$) can be equivalently stated as

$$\arg\max_{\mathbf{v_1}, \|\mathbf{v}\|=1} \sum_{x_i \in X} \langle x_i, \mathbf{v_1} \rangle^2 =: \arg\max_{\mathbf{v_1}, \|\mathbf{v}\|=1} \sum_{x_i \in X} a_{i1}^2 \tag{2}$$

Note that since our data is zero-centered, for $1 \leq j \leq n$, $\sum_{i=1}^m x_{ij}^2 = \sigma_j^2$, the variance in the jth coordinate of X. By linearity of dot-product ($\mathbf{p} \cdot (\mathbf{q} + \mathbf{r}) = \mathbf{p} \cdot \mathbf{q} + \mathbf{p} \cdot \mathbf{r}$), we have that $\sum_{i=1}^m \langle x_i, \mathbf{v_1} \rangle = \langle \sum_{i=1}^m x_i, \mathbf{v_1} \rangle$. Since the first term in the dot product is 0, we get that $\sum_{i=1}^m \langle x_i, \mathbf{v_1} \rangle = \sum_{i=1}^m a_{i1} = 0$. Hence the projections a_{i1}'s are zero-centered and $\sum_{x_i \in X} a_{i1}^2$ is the variance in a_{i1}'s.

We conclude that the objective of PCA (with $k = 1$) is (2) is finding a (direction) $\mathbf{v_1}$ in which there is the most variance in the data X.

To see a general example (in addition to the specific examples above), observe in the following diagram, the data has much higher variance in the direction of $\mathbf{v_1}$ and much smaller variance in the direction of $\mathbf{v_2}$. Thus PCA will seek to find $\mathbf{v_1}$. Observe that the vector $\mathbf{v_1}$ is the optimal vector satisfying both (1) and (2) (as they are both the same).



3.2.5 Subsequent projections: Larger k

PCA reduces dimensionality from n to k by projecting each point into a k-dimensional subspace such that the sum of squared distances of points to this k-dimensional subspace is minimum. That is the objective of PCA with $k \geq 2$ is

$$\underset{k\text{-dim subspace } S}{\arg\min} \sum_{x_i \in X} \|x_i - x_i'\|^2 := \underset{k\text{-dim subspace } S}{\arg\min} \sum_{x_i \in X} (\text{distance b/w } x_i \text{ and } S)^2, \tag{3}$$

where x_i' is the projection of $x_i \in X$ on the subspace S. As above this is equivalent to

$$\underset{k\text{-dim subspace } S}{\arg\max} \sum_{x_i \in X} (\text{length of projection of } x_i \text{ on } S)^2 \tag{4}$$

Just as we did for $k = 1$ (that we represented the line by its unit basis vector), here also we represent the desired k-dimensional subspace S by its k (linear independent) bases vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}$ above. Note that if $\mathbf{v_1}, \ldots, \mathbf{v_k}$ are not linearly independent, then they will not span a k-dim subspace.

Furthermore, to keep the algebra simple, we require that $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}$ make orthonormal bases for S. This makes life very easy. Because computing length of projections on a space S given by its orthonormal bases is just the sum of projections on each basis vectors, i.e. This simple fact follows from the Pythogorus theorem.

$$(\text{length of projection of } x_i \text{ on } S = span(\mathbf{v_1}, \ldots, \mathbf{v_k}))^2 = \sum_{j=1}^{k} \langle x_i, \mathbf{v_j} \rangle^2 \tag{5}$$

$$\underset{\substack{\mathbf{v_1},\ldots,\mathbf{v_k} \\ \|\mathbf{v_p}\|=1 \\ \mathbf{v_p} \perp \mathbf{v_q}}}{\arg\max} \sum_{i=1}^{m} \underbrace{\sum_{j=1}^{k} \langle x_i, \mathbf{v_j} \rangle^2}_{\text{squared projection length on } span(\mathbf{v_1}, \ldots, \mathbf{v_k})} \tag{6}$$

The vectors $\mathbf{v_1}, \ldots, \mathbf{v_k}$ maximizing the objective function (6) are called the *top k principal components of X*. In summary, the general problem of principal component analysis is given as

Problem 1 (Principal Component Analysis)**.** *Given $X \subset \mathbb{R}^n$, $|X| = m$ and an integer $k \geq 1$, find vectors $\mathbf{v_1}, \ldots, \mathbf{v_k}$ to maximize* (6) *and projection of X onto $\mathbf{v_1}, \ldots, \mathbf{v_k}$.*

3.3 PCA: Linear Algebraic Formulation

In matrix notation, the projection of X onto a vector $\mathbf{v_1}$ can be rewritten as

$$X\mathbf{v_1} = \begin{pmatrix} x_{11} & x_{12} & \ldots & \ldots & \ldots & x_{1m} \\ x_{21} & x_{22} & \ldots & \ldots & \ldots & x_{2m} \\ \vdots & \vdots & \ldots & \ldots & \ldots & \vdots \\ \vdots & \vdots & \ldots & \ldots & \ldots & \vdots \\ \vdots & \vdots & \ldots & \ldots & \ldots & \vdots \\ x_{m1} & x_{m2} & \ldots & \ldots & \ldots & x_{nm} \end{pmatrix} \begin{pmatrix} v_{11} \\ v_{12} \\ \vdots \\ \vdots \\ \vdots \\ v_{1m} \end{pmatrix} = \begin{pmatrix} \langle \mathbf{x}_1, \mathbf{v_1} \rangle \\ \langle \mathbf{x}_2, \mathbf{v_1} \rangle \\ \vdots \\ \vdots \\ \vdots \\ \langle \mathbf{x}_n, \mathbf{v_1} \rangle \end{pmatrix}$$

Since we are interested about squares of projections, we take the dot-product of $X\mathbf{v_1}$ with itself.

$$X\mathbf{v_1} \cdot X\mathbf{v_1} = (X\mathbf{v_1})^T X\mathbf{v_1} = \mathbf{v_1}^T X^T X\mathbf{v_1} = \sum_{i=1}^{n} \langle x_i, \mathbf{v_1} \rangle^2$$

Let $C = X^T X$ be the covariance matrix, we restate the objective function (2) (for $k = 1$) as

$$\underset{\mathbf{v_1}, \|\mathbf{v}\|=1}{\arg\max} \sum_{x_i \in X} \langle x_i, \mathbf{v_1} \rangle^2 := \underset{\mathbf{v_1}, \|\mathbf{v}\|=1}{\arg\max}\ \mathbf{v_1}^T X^T X\mathbf{v_1} := \underset{\mathbf{v_1}, \|\mathbf{v}\|=1}{\arg\max}\ \mathbf{v_1}^T C\mathbf{v_1} \tag{7}$$

3.4 Diagonal Covariance matrix

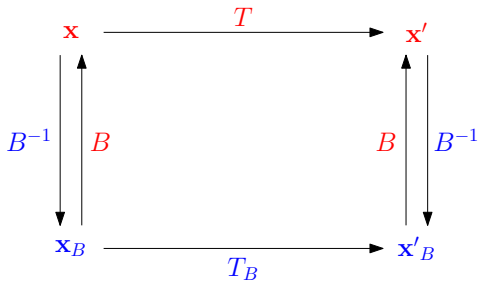
In this special not very realistic case assume $k = 1$ and the covariance matrix C is a diagonal matrix, i.e. $C = \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_n \end{bmatrix}$. Furthermore, assume that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$.

Note that the latter assumptions are not very strict as the covariance matrix only contain non-negative entries and the order can be achieved by re-arranging rows of any matrix. The only restricting assumption is C being diagonal, which correspond to all correlations being 0. We make this assumption for developing an understanding and will remove it soon.

We want to find a vector $\mathbf{v}$ such that

$$\underset{\mathbf{v}, \|\mathbf{v}\|=1}{\arg\max} \sum_{x_i \in X} \langle x_i, \mathbf{v} \rangle^2 := \underset{\mathbf{v}, \|\mathbf{v}\|=1}{\arg\max}\ \mathbf{v}^T X^T X\mathbf{v} := \underset{\mathbf{v}, \|\mathbf{v}\|=1}{\arg\max}\ \mathbf{v}^T C\mathbf{v} = \underset{\mathbf{v}, \|\mathbf{v}\|=1}{\arg\max} \sum_{i=1}^{m} \lambda_i v_i^2 \tag{8}$$

$$B = \begin{bmatrix} | & | & & | \\ \mathbf{b_1} & \mathbf{b_2} & \dots & \mathbf{b_n} \\ | & | & & | \end{bmatrix}$$



Recall that C is just a scaling linear transformation and we want to find a unit vector $\mathbf{v}$ for which $\sum_{i=1}^{m} \lambda_i v_i^2$ is maximum.

Lemma 1. *The standard basis vector* $\mathbf{v} = \mathbf{e}_1 = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}^T$ *is the optimal solution to* (9)

Proof. We will show that for any other unit vector $\mathbf{u}$ we have that

$$\sum_{i=1}^{m} \lambda_i v_i^2 \geq \sum_{i=1}^{m} \lambda_i u_i^2$$

Since $\|\mathbf{u}\| = 1$ i.e. $\sum_{i=1}^{m} u_i^2 = 1$, we get that $\sum_{i=1}^{m} \lambda_i u_i^2$ is just weighted average of λ_i's weighted by u_i's. For $\mathbf{v} = \mathbf{e}_1$ we have $\sum_{i=1}^{m} \lambda_i v_i^2 = \lambda_1 = \max_i \lambda_i$. We know that any average is always at most the maximum, we get that $\mathbf{v} = \mathbf{e}_1$ maximizes (9) □

Thus for a dataset whose covariance matrix C is a diagonal matrix its top principal component to project the data on is $\mathbf{e}_1$. For general covariance matrix, (still $k = 1$) we extend the above idea.

3.4.1 Eigenbases, Diagonalization, Eigen Decomposition

Recall we discussed that we can translate (find coordinates of) a vector given in standard bases of $\mathbb{R}^n$ in another bases (a set of linearly independent vectors). Similarly we can perform transformation in the other bases and reverse translate them into the standard bases. Supposes we are given a bases (in the form a matrix) B. Columns of B are basis vectors (linearly independent). This necessarily means that B is invertible.

In the following red the coordinates in standard bases are denoted by color red and those in the bases B are denoted in blue color. The following is a schematic diagram of transforming a vector $\mathbf{x}$ by a $n \times n$ linear transformation T.

$$T_B = B^{-1} T B \qquad\qquad T = B T_B B^{-1}$$

Suppose that the new bases B is composed of n eigen vectors of T. i.e. $B = \{\mathbf{b_1}, \dots, \mathbf{b_n}\}$ be bases - columns B are eigenvectors of T. For $1 \leq i \leq n$, $T\mathbf{b_i} = \lambda_i \mathbf{b_i}$

How does $T\mathbf{x}$ looks like in eigenbasis?

$$
\begin{aligned}
T\mathbf{x} = T(\alpha_1\mathbf{e_1} + \ldots + \alpha_n\mathbf{e_n}) &= T(\beta_1\mathbf{b_1} + \ldots + \beta_n\mathbf{b_n}) \\
= \beta_1 T\mathbf{b_1} + \ldots + \beta_n T\mathbf{b_n} &= \beta_1\lambda_1\mathbf{b_1} + \ldots + \beta_n\lambda_n\mathbf{b_n}
\end{aligned}
$$

$$
= \begin{bmatrix} | & | & & | \\ \mathbf{b_1} & \mathbf{b_2} & \ldots & \mathbf{b_n} \\ | & | & & | \end{bmatrix} \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} = BD\mathbf{x_B} = BDB^{-1}\mathbf{x}
$$

$$
\text{where } D = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}
$$

We get that

$$
T\mathbf{x} = BDB^{-1}\mathbf{x}
$$

We discussed that the reason for this is that it is much easier to compose the transformation many times, in this form it is very easy to take T to a higher power (compose it many times)

- $T = BDB^{-1}$
- $T^2 = BDB^{-1}BDB^{-1} = BDIDB^{-1} = BDDB^{-1} = BD^2B^{-1}$
- $T^3 = BD^2B^{-1}BDB^{-1} = BD^2DB^{-1} = BD^3B^{-1}$
- $T^4 = BD^3B^{-1}BDB^{-1} = BD^3DB^{-1} = BD^4B^{-1}$
- $T^k = \ldots = BD^kB^{-1}$

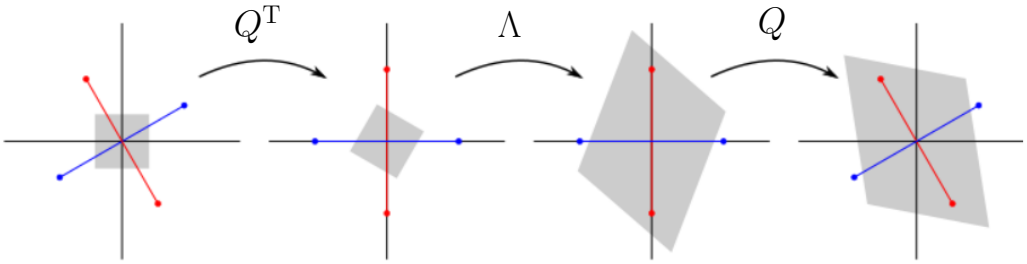
It is easy to see that $D^k = \begin{bmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_n^k \end{bmatrix}$

3.4.2 Eigen Decomposition

(also called spectral decomposition) is the factorization of a matrix in terms of its eigenvalues and eigenvectors. Only diagonalizable matrices can be factorized in this way. If a square matrix A has n linearly independent eigenvectors $\mathbf{q}_i$ $(i = 1, \ldots, n)$. Then A can be factorized as

$$
\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}
$$

where Q is the square $n \times n$ matrix whose ith column is the eigenvector $\mathbf{q}_i$ of A, and Λ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, $\Lambda(i, i) = \lambda_i$. We apply eigendecompostion to the covariance matrix C.



3.4.3 Real Symmetric Matrices

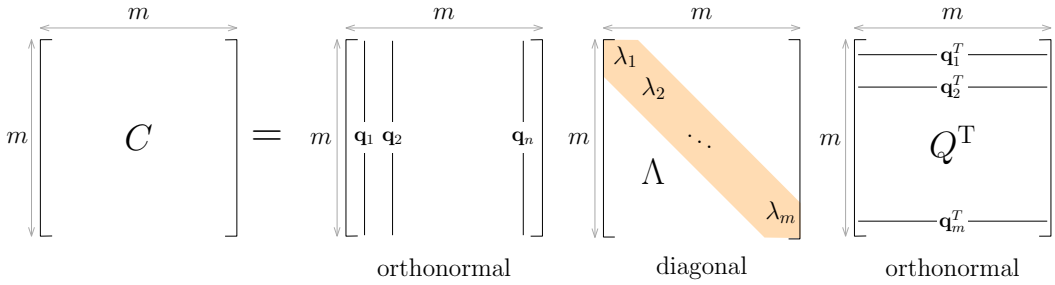
Recall that the covariance matrix has all values reals and it is symmetric. We use the following well-known and fundamental result from linear algebra.

For every $n \times n$ real symmetric matrix, the eigenvalues are real and the eigenvectors can be chosen real and orthonormal.

Thus a real symmetric matrix A can be decomposed as

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$$

where Q is an **orthogonal matrix** whose columns are the eigenvectors of A, and Λ is a diagonal matrix whose entries are the eigenvalues of A.



3.4.4 Orthogonal Matrices

An orthogonal matrix, or orthonormal matrix, is a real square matrix whose columns and rows are orthonormal vectors, i.e. length of each column is 1 and all pairs of columns are orthogonal, their dot-product is 0.

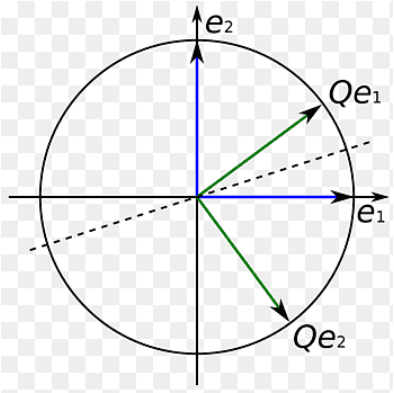
Properties of orthogonal matrices

- $Q^{\mathrm{T}}Q = QQ^{\mathrm{T}} = I$, where Q^{T} is the transpose of Q and I is the identity matrix.
- An equivalent characterization: inverse of an orthogonal matrix Q is its transpose

$$Q^{-1} = Q^{T}$$

- An orthogonal matrix Q is necessarily invertible

- As a linear transformation, an orthogonal matrix preserves length of vectors, i.e. $\|Q\mathbf{v}\| = \|\mathbf{v}\|$ for any vector v (they are called unitary transformations). In other words as linear transformation an orthogonal matrix Q only achieves a rotation, reflection or permutation of coordinates.



3.4.5 Eigendecomposition of covariance matrix

Recall that we want to find a vector $\mathbf{v}$ such that

$$\arg\max_{\mathbf{v},\|\mathbf{v}\|=1} \sum_{x_i \in X} \langle x_i, \mathbf{v} \rangle^2 := \arg\max_{\mathbf{v},\|\mathbf{v}\|=1} \mathbf{v}^T X^T X \mathbf{v} := \arg\max_{\mathbf{v},\|\mathbf{v}\|=1} \mathbf{v}^T C \mathbf{v} = \arg\max_{\mathbf{v},\|\mathbf{v}\|=1} \sum_{i=1}^{m} \lambda_i v_i^2 \tag{9}$$

Let $C = QDQ^T$, where Q is the orthogonal matrix of eigenvectors of C and D is a diagonal matrix with eigenvalues corresponding to columns in Q in increasing order.

We know that $\mathbf{e}_1$ is the direction of matrix stretch under D. To get the direction of max stretch under $C = QDQ^T$, we need to find out the vector that gets mapped to $\mathbf{e}_1$ under Q^T, because that vector will get stretched the most under DQ^T and Q being orthogonal will not stretch or shrink it any further. In other words we the vector of max stretch under C is $(Q^T)^{-1}\mathbf{e}_1$. Which by orthogonality of Q^T is $Q\mathbf{e}_1$. Recall that definition of linear transform $Q\mathbf{e}_1$ is just the first column of Q or the eigenvector of C corresponding to the leading eigenvalue.

Thus the first principal component for projection of a dataset X is the leading eigenvector of $C = XX^T$.

To get the top k principal components we use the first k leading eigenvectors of C. This is very easy to see, again first in the case when C is a diagonal matrix.

This completes the linear algebraic formulation of PCA. We project the data onto the top k eigenvectors of the covariance matrix. We formulate this in the algorithm below.

There are a few important issues that we need to address

3.5 PCA: The Algorithm

Here we give the step by step algorithm for principal component analysis and dimensionality reduction via PCA.

INPUT: Data set: $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, each $\mathbf{x}_i$ is m dimensional vector.

Make dataset into a $n \times m$ matrix, each row is a data point.

Step-1: Zero Center the data

$\overline{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$ ▷ m-d mean vector, coordinate-wise mean

Replace each data point $\mathbf{x}_i$ by $\mathbf{x}_i = \mathbf{x}_i - \overline{\mathbf{x}}$

Step-2: Compute the covariance matrix With zero-centering this is also the correlation matrix

$C = X^T X$ ▷ Normalize it by n or $n - 1$

Step-3: Compute eigenvectors of C and let Q be the matrix with eigenvectors of C as its columns

Transform the data matrix i.e. $Y = XQ$

To reconstruct original data we can use $X = YQ^{\mathrm{T}}$

For reducing dimenisons from m to k, remove the last $m - k$ columns of Q to get Q_k and compute $Y' = XQ_k$. Y' is a $n \times k$ matrix

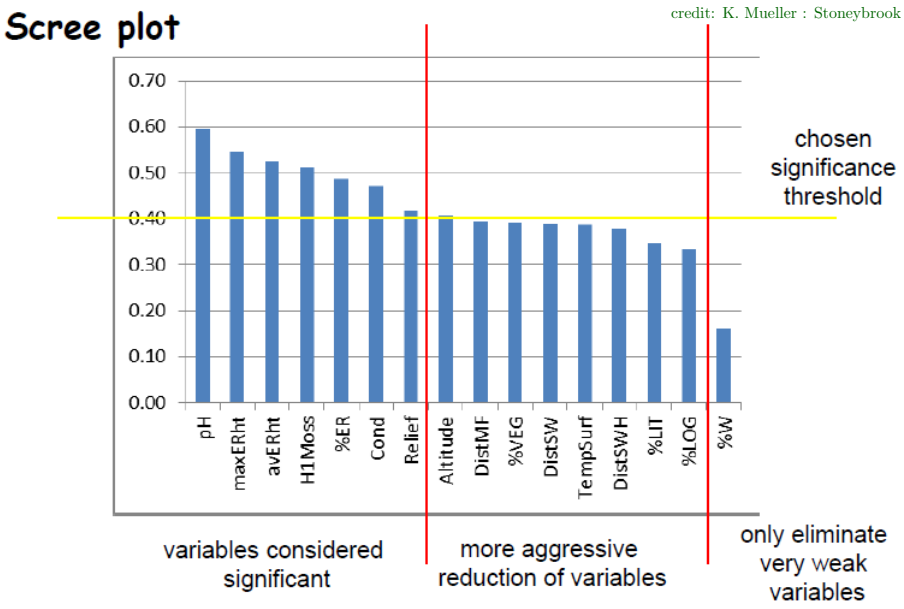
To (approximately) reconstruct use $X' = Y'Q_k^{\mathrm{T}}$

3.5.1 Complexity of the Algorithm

Complexity of this PCA algorithm is as follows:

- Zero centering the data takes $O(nm)$ (input scan)
- Computing the covariance matrix takes $O(m^2 n)$ – there are $O(m^2)$ entries in C and each value takes $O(n)$ multiplication and addition
- Eigenvalue decomposition require $O(m^3)$ time for a $m \times m$ matrix.
- Transformation and dimensionality reduction require $n \times m \times k$ time
- Total runtime is $O(nm^2 + m^3)$.

For large datasets this algorithm becomes computationally infeasible.
We use SVD method to compute PCA, in order to avoid computing the covariance matrix.
Eigenvectors can also be computed using power iteration methods discussed below.



3.5.2 Number of principal components needed: Scree Plot

This is essentially the same question as what is the right number of clusters for a dataset. Recall we discussed that we quantify the goodness of clustering by some measure and find a point k such if we cluster the dataset into more than k cluster the goodness measure does not improve much.

We use a similar idea here. It is very important to determine the goal of PCA. If we want to get the best graphical visualization of data then clearly the number of principal components should be 2 or 3. For dimensionality reduction, it would depend on the required accuracy of the analytic task at hand.

Since we know that the variance explained by a principal component is the corresponding eigenvalues. We select k such that that the remaining $m - k$ eigen values are very small. This is essentially applying the elbow method to the so-called Scree plot - a bar graph (or line plot) of eigenvalues that shows the fraction of total variation in data explained by the corresponding eigenvalues. We select the (elbow or knee) point, where there is a "substantial" drop in the eigenvalue.

A quantified measured is to threshold the amount variance explained by the k components relative to the total variance in the data. i.e. choose k such that

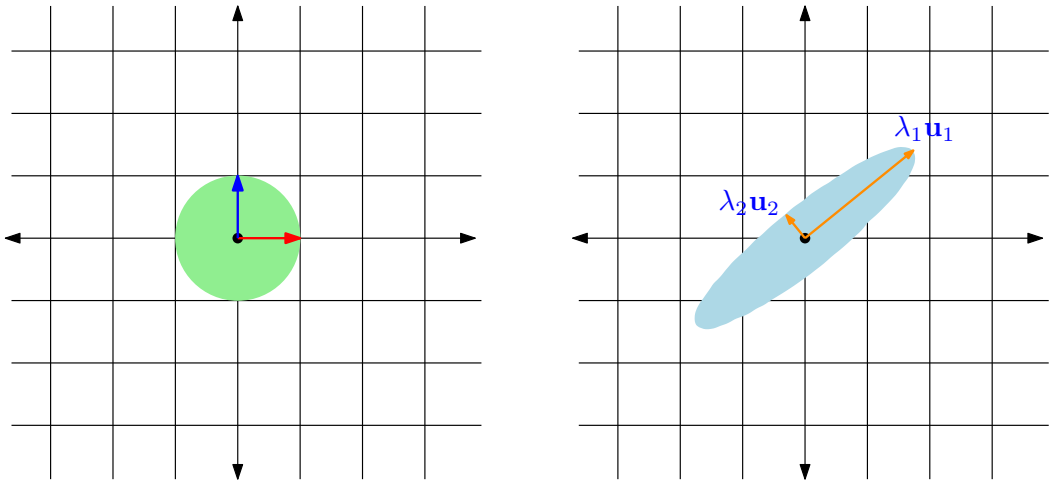
$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{m} \lambda_i} \geq (1 - \epsilon) \quad \text{for a user parameter} 1 < \epsilon < 1$$

3.5.3 The Power Iteration method to compute eigen vectors

Note two important aspects of eigenvalues. First the top eigenvector (the eigenvector corresponding to the largest eigenvalue) of a matrix A corresponds to the direction in which the

linear transformation A stretches the vector the most. Secondly, it is well-known that if A has eigenvalues $\lambda_1, \lambda_2, \ldots,$, then A^k has eigenvalues $\lambda_1^k, \lambda_2^k, \ldots,$,

We use these ideas to develop a numerical method to compute the top eigenvector of a matrix. We know that a matrix in addition to stretching, rotates and reflects vectors. In 2d, we pictured linear transformation by the change to unit square that became a parallelogram after the transformation. One of the corners of the parallelogram had the longest stretch. It will be good to picture what happens to a unit circle that will become an ellipse as point on the circle are unit vectors. The farthest corner of the parallelogram or the longest axis of the ellipse corresponds to the top eigenvector.



Now imagine the same ellipse for the matrix A^k for a large integer k, it is not very hard to see that ellipse will be very long and thin (in $2d$), as its longest axis will have length λ_1^k and second longest axis will have length λ_2^k (their difference will be amplified). Note that λ_1 does not have to be bigger than 1 for this phenomenon.

If we repeatedly transform a random vector $\mathbf{v}$ with the matrix A, i.e. compute $A(A(A \ldots A(A(\mathbf{v}))))$, the vector $\mathbf{v}$ will get stretched so much in the direction of the top eigenvector of A that the final image $(A^k(\mathbf{v}))$ will lie almost entirely in the direction of the top eigenvector. Another way to see this almost all points on the unit circle get mapped to points that are close to the longest axis of the ellipse (image of the circle by A^k).

Algorithm Power Iteration to compute top eigenvector of a matrix $A = X^T X$

$\mathbf{v}_0 \leftarrow$ RANDOM-UNIT-VECTOR() ▷ Generate random direction

$i \leftarrow 0$

while stopping criteria is not met **do**

$\mathbf{v}_{i+1} \leftarrow A\mathbf{v}_i$

$\mathbf{v}_{i+1} \leftarrow \dfrac{\mathbf{v}_{i+1}}{\|\mathbf{v}_i\|}$ ▷ Normalize to get unit vector

$i \leftarrow i + 1$

The stopping criterion usually is based on a threshold for convergence , i.e. continue until $\|\mathbf{v}_{i+1}\| - \|\mathbf{v}_i\|$ is below a certain threshold.

Running time of this algorithm depends on the so-called spectral gap of the matrix, $\frac{\lambda_2}{\lambda_1}$. One can prove the following lemma regarding the quality of this algorithm (the dot-product between the output of the algorithm and the top eigenvector of A, notice that if this dot-product is 1, it means the output is "inline" with the top eigenvector)

Lemma 2. *Let* $\mathbf{u}$ *be the top eigenvector of the matrix* A*. For a random vector* $\mathbf{v}_0$ *and a positive integer* k*, we have that*

$$Pr\left[|\langle A^k\mathbf{v}_0, \mathbf{u}\rangle| \geq 1 - 2\sqrt{n}\left(\frac{\lambda_2}{\lambda_1}\right)^k\right] \geq \frac{1}{2}$$

The probability can be amplified by repeating the algorithm many times independently and selecting the output with largest $A\mathbf{v}_k$.

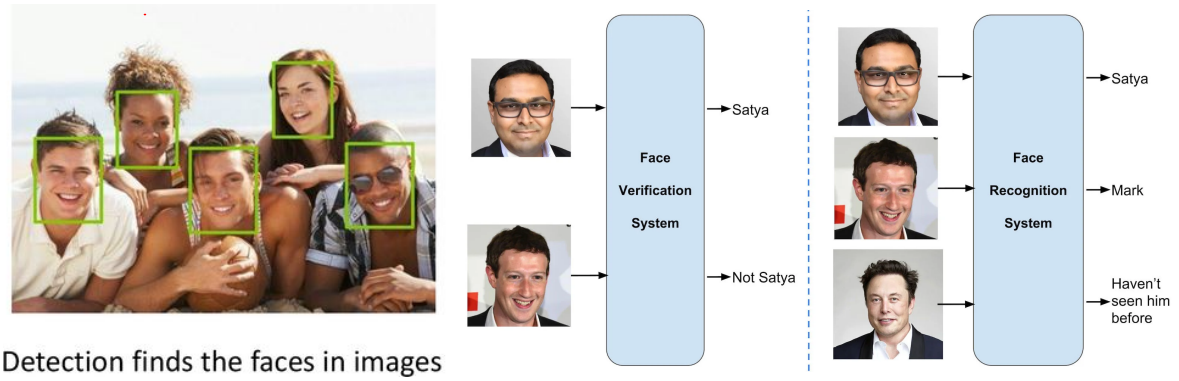
Remark 1. *Notice that in the above algorithm* $\mathbf{v}_k = A^k\mathbf{v}$*. In order to avoid* k *matrix vector multiplications, one can compute* $\mathbf{v}_k$ *by computing* A^k *using repeated squaring* $A, A^2, A^4, A^4, A^8, \ldots$ *and do one matrix-vector multiplication in the end. Computing* A^k *using repeated squaring takes* $O(\log k)$ *matrix-matrix multiplications. For large* k *this is substantially faster.*

Some comments of the power method to compute eigenvectors.

- If $|\lambda_2/\lambda_1|$ is close to 1, then convergence is very slow
- Matrix vector multiplication is very fast if the matrix is sparse (many 0 values), in that case using algorithm for sparse matrices, this is quite fast algorithm
- As stated it only computes the top eigenvector (the top principal component in our case), to get more principal components we do the following
  - After computing $\mathbf{v}_1$, the first eigenvector, we project the matrix onto $\mathbf{v}_1$ and subtract it out
  - i.e. Compute A' which is the "residual matrix". . Row i of A' is $\mathbf{a}_i - \langle\mathbf{a}_i, \mathbf{v} - 1\rangle\mathbf{v}_i$
  - Recursively compute the top $k-1$ eigenvectors of A'
  - To compute the bottom eigenvector (e.g. of the Laplacian matrix, that we will study for spectral clustering), the algorithm is called inverse power iteration method
  - This follows from the fact that eigenvalues of A^k are $\lambda_1^k, \lambda_2^k, \ldots,$. For $k = -1$, we get that eigen values of the inverse A^{-1} of A are $1/\lambda_1, 1/\lambda_2, \ldots,$.
  - Notice that since $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$, we get that $1/\lambda_1 \leq 1/\lambda_2 \leq \ldots \leq 1/\lambda_n$.
  - Thus applying the power iteration method to A^{-1} we can compute the smallest eigenvalue.
  - With a little bit of linear algebra computing the inverse can be avoided
  - To compute all eigenvectors the algorithm is called the QR algorithm

4 PCA: Case Studies and Examples

A classic application of PCA is to represent images in a lower dimensionality space. PCA has been used (with reasonable success) for image compression and face recognition tasks.



Detection finds the faces in images

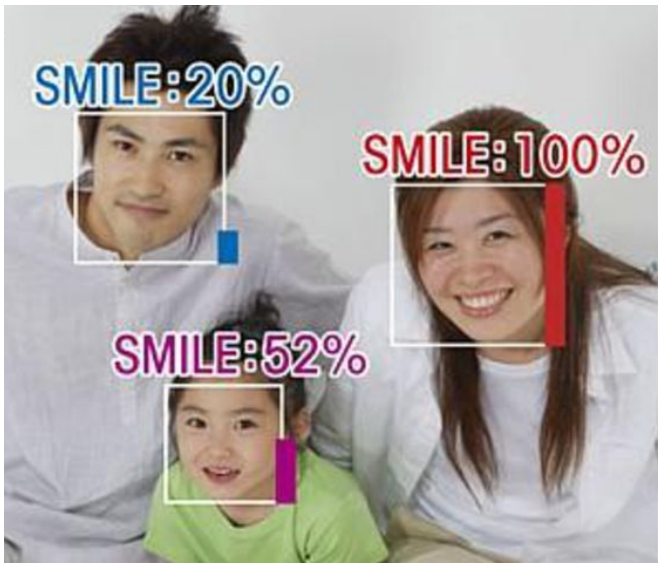
J. Niebles & R. Krishna @ Stanford

source: learnopencv.com

Applications of face detection, face verification and face recognition include
Surveillance

GILLES SABRIETHE NEW YORK TIMES/REDUX

Emotion and Expression Detection

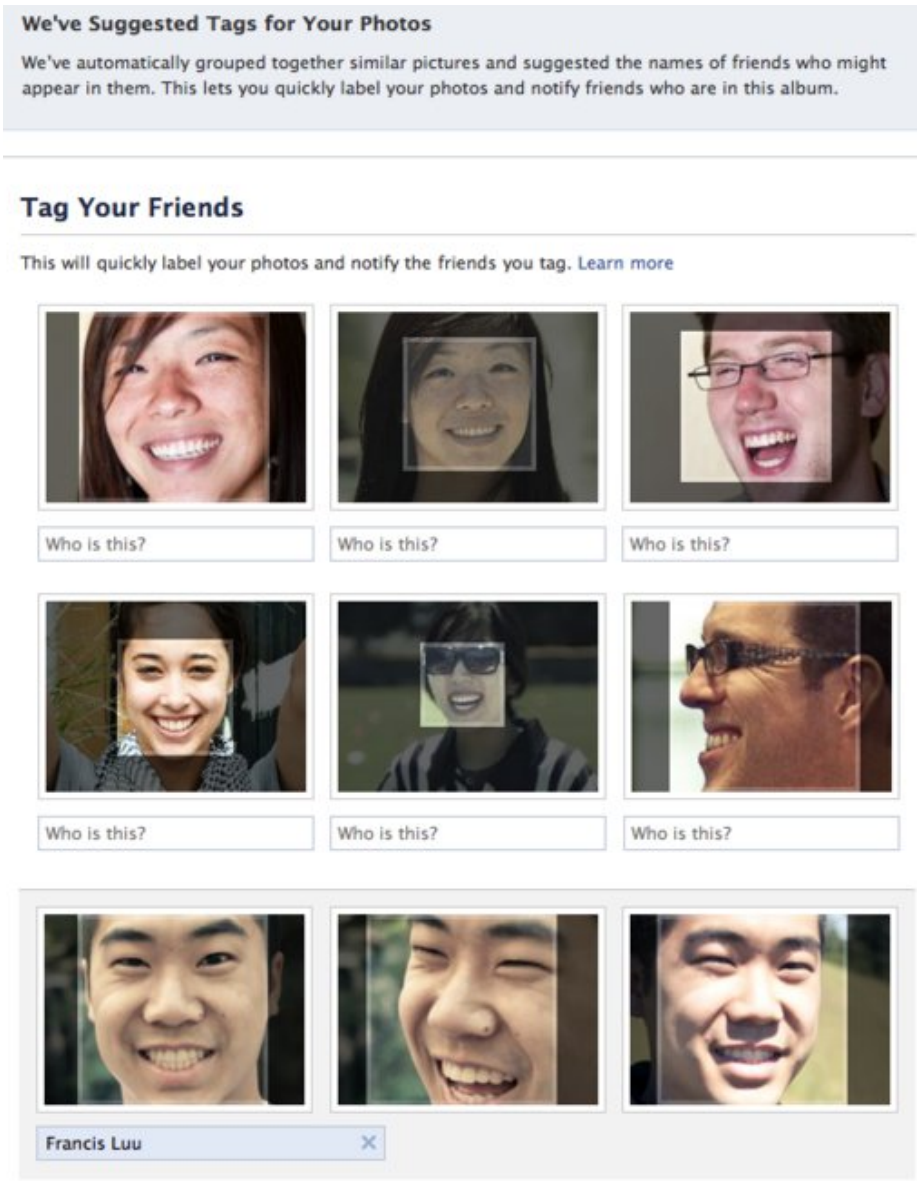


J. Niebles & R. Krishna @ Stanford

Photo Album Organization

J. Niebles & R. Krishna @ Stanford

Facebook Auto Tag Suggestions



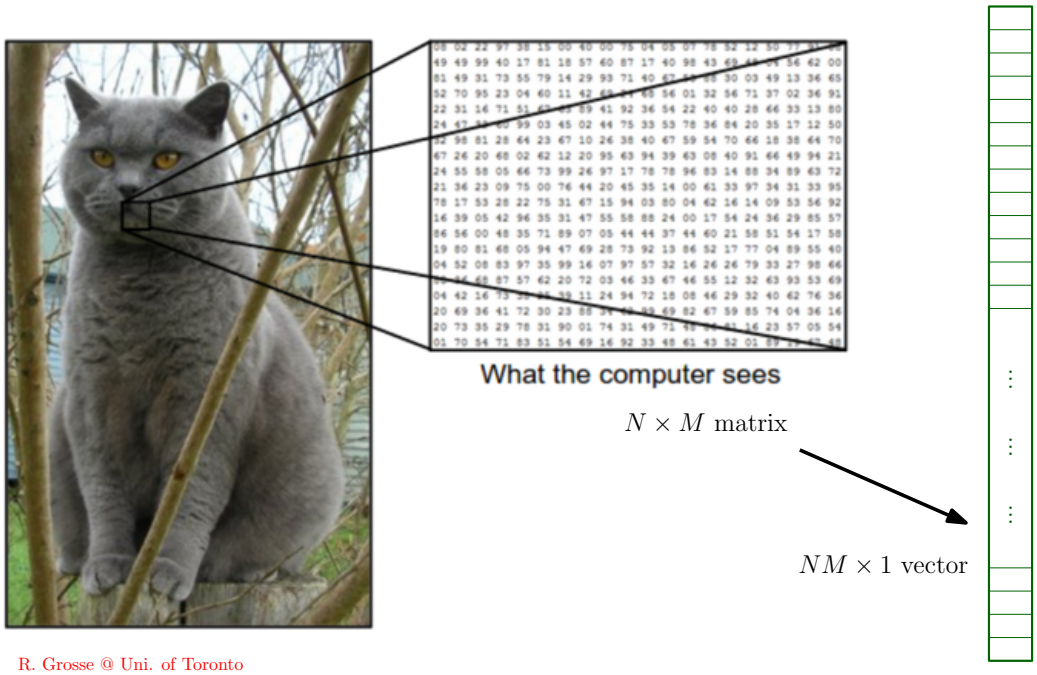
4.1 Images as Vectors

Input Images

Dataset For each face there should be a few training examples

All faces should be centered

Represent images by vectors



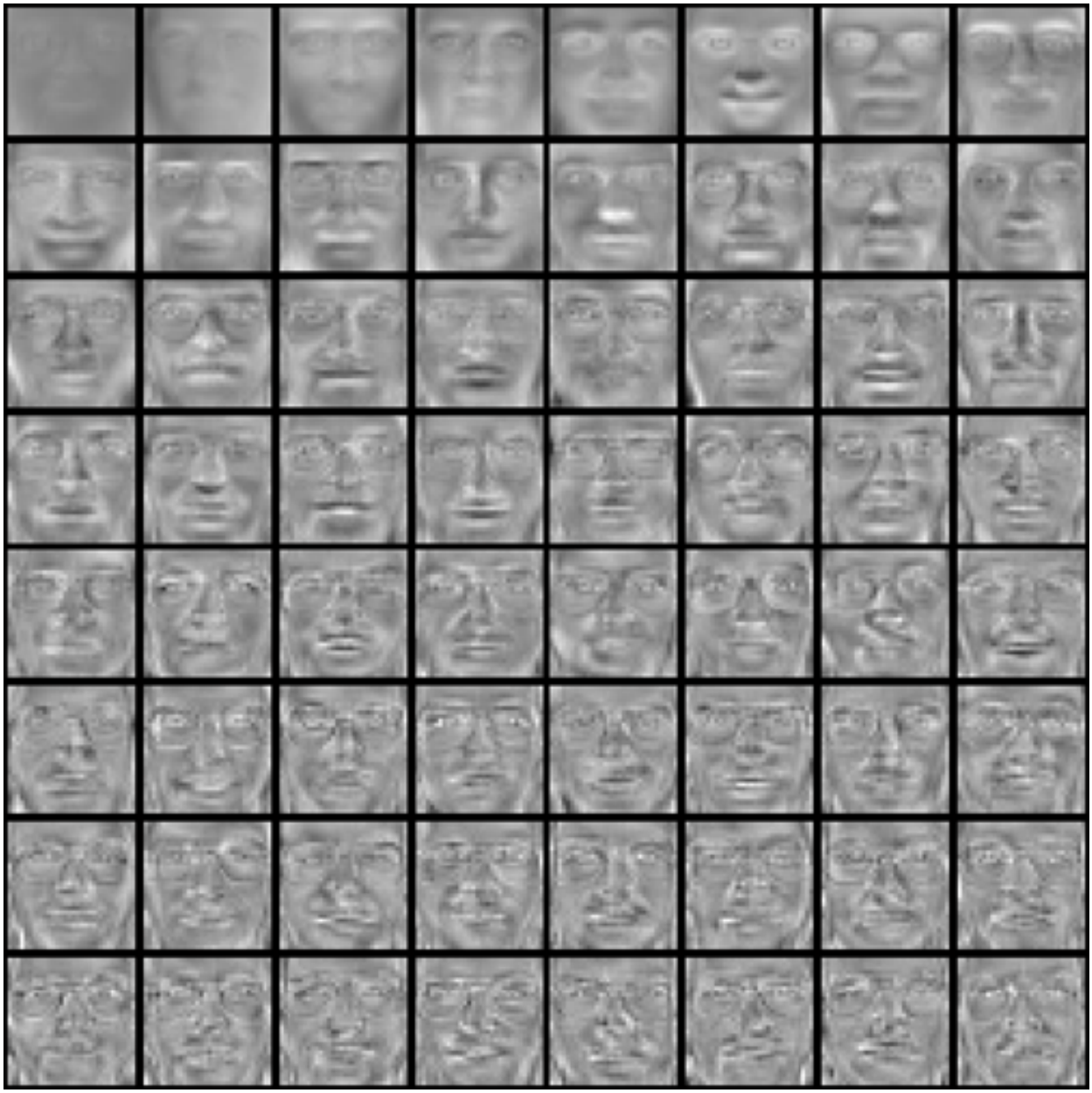
R. Grosse @ Uni. of Toronto

Mean Face

Mean face $\bar{\mathbf{x}}$

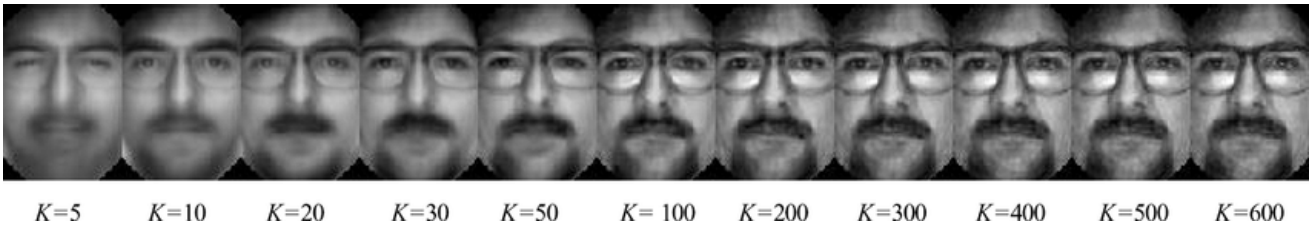
Eigen Faces

Top eigenvectors: $u_1, \ldots, u_k$ (visualized as images - eigenfaces)



Represent each image as a linear combination of the top k eigenfaces, i.e. project faces onto these components

Effect of number of principal components on reconstruction

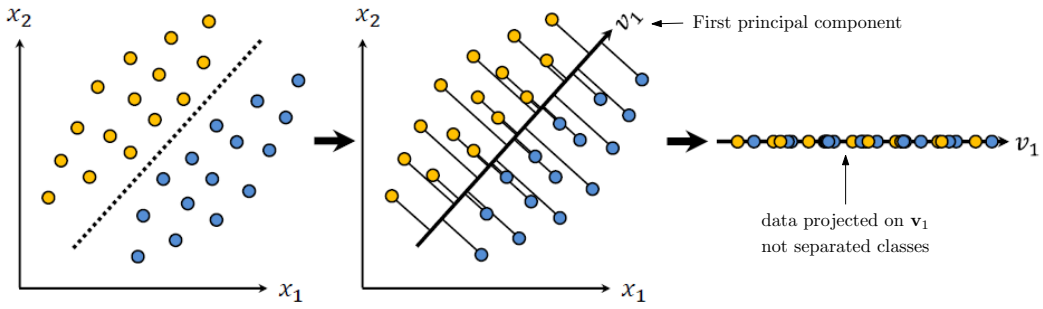


Recognition: Project the training sample onto the same k principal components after subtracting the mean face. Use k-nearest neighbors (by the new representations) and make a prediction based on that.

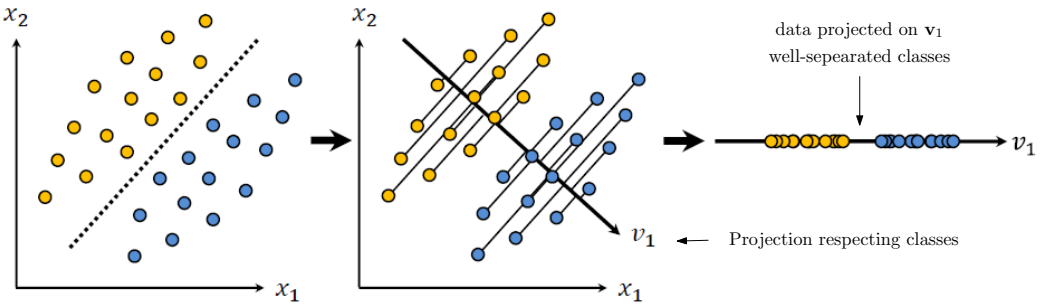
Face Detection: For a region R of the image, project R onto the principal components. If the ℓ_2 distance of the new representation of R with R is not significant, then R is a face.

5 Limitations of PCA and Other feature extraction and dimensionality reduction methods

PCA does not take into account any class labels (a completely unsupervised approach) thus it does not necessarily help separate data based on classes and may not lead to better classification (based on reduced dimensional data).

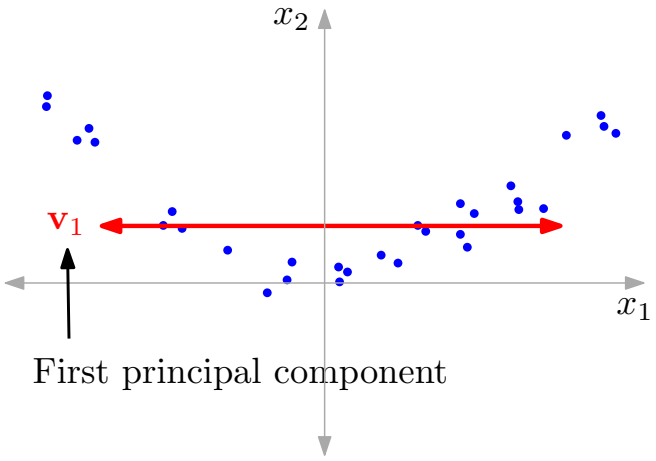


Linear Discriminant Analysis (LDA): Seeks a projection that best discriminates the data



Principal components are linear combinations of the original features. If original features are related in a non-linear fashion PCA may not capture it

Non-linear structure may not be captured



In addition of PCA other related linear dimensionality reduction methods include

- **Factor Analysis**
- **Independent Component Analysis:** Seeks a projection that preserves as much information in the data as possible

Non-linear methods include

- **Laplacian Eigenmaps**
- **ISOMAP**
- **Local Linear Embedding** Embedding to low dimensional manifolds