EECS 16B Designing Information Devices and Systems II
Fall 2021

Note 17: PCA

Overview

In this note, we will examine a key application of the singular value decomposition (SVD) by using *low-rank approximations* for dimensionality reduction of data. When we collect data, there are potentially lots of things that are influencing the exact values that we see. Our goal is often to figure out the most important of these factors. This allows us to compress or distill our data down to its important essence, and remove the dimensions of data that are not imporant. The key new idea here is that of approximation. How can we do this using the linear-algebraic tools that we have built?

1 Analyzing Matrix Data

Imagine we are conducting some sort of statistical study — say, we are interested in how a group of n students enjoy a set of m movies. Each student gives a numerical rating of each movie, indicating how much they enjoyed it. We place these ratings into a matrix R, where R_{ij} is the rating the i^{th} student gives to the j^{th} movie. Our goal will be to draw conclusions from this dataset that enable us to make future predictions.

For instance, if we are told how much a new student enjoys a subset of our movies, we should be able to predict how much they will enjoy the other movies in our set. Alternatively, if we produce a new movie and are told how much a subset of our students enjoy it, we should be able to predict how much the other students in our study will enjoy it.

How on earth can we come up with these predictions? Aren't we all *unique and special*, so knowing how a student enjoys some movies provides no information on how they will enjoy the rest? In such a case, then yes, nothing could be done and our problem would be unsolvable.

However, in practice, it turns out that students are not as completely unique and special as all that! Typically, there is an underlying structure that lets us predict how much a student will enjoy each movie. For instance, imagine that every j^{th} film has a certain intrinsic "goodness" g_j, and every i^{th} student has a "sensitivity" s_i. Then the i^{th} student's rating of the j^{th} movie can be computed as the product of the student's sensitivity and the film's goodness, yielding

$$R_{ij} = s_i g_j. \tag{1}$$

In such an ideal scenario, our ratings matrix would simply be

$$R = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} \begin{bmatrix} g_1 & g_2 & \cdots & g_m \end{bmatrix} = \vec{s}\vec{g}^{\top}. \tag{2}$$

To take a slightly more complex scenario, imagine for simplicity that movies can be described by a mix of three properties - their levels of "romance," "comedy," and "action", represented by the vectors $\vec{r}$, $\vec{c}$, and $\vec{a}$. And imagine that each student computes how much they enjoy each movie by taking a linear combination of the "romance", "comedy", and "action" of each movie, where the coefficients of this linear combination vary with each student. Let these coefficients be stored in the vectors $\vec{s}_r$, $\vec{s}_c$, and $\vec{s}_a$ respectively. We'll call these coefficients the student's "sensitivities" to each of the three genres.

In such a case, we have that

$$R_{ij} = s_{r,i}r_j + s_{c,i}c_j + s_{a,i}a_j, \tag{3}$$

so the overall ratings matrix R can be expressed as

$$R = \vec{s}_r\,\vec{r}^\top + \vec{s}_c\,\vec{c}^\top + \vec{s}_a\,\vec{a}^\top. \tag{4}$$

The point is that, despite being an $n \times m$ matrix, an idealized R according to this model can be represented as the sum of only a constant number of outer products of n- and m-dimensional vectors.

What could we do if we knew these vectors? Well, imagine that we were given a new student and their ratings on a subset of the movies, stored in the vector $\vec{p}$. What do we want to know about them? Well, we've said that their sensitivities to romance, comedy, and action fully describe their movie preferences. So if we can recover their sensitivities to these genres, then we can make predictions about their ratings of a future movie, so long as we know what genres this future movie is in.

And how can we recover these sensitivities? We know that we can write each provided rating

$$p_j = w_r r_j + w_c c_j + w_a a_j, \tag{5}$$

where w_r, w_c, and w_a are the sensitivities of our new student. Stacking, we obtain the vector equation

$$\begin{bmatrix} r_1 & c_1 & a_1 \\ r_2 & c_2 & a_2 \\ \vdots & \vdots & \vdots \\ r_m & c_m & a_m \end{bmatrix} \begin{bmatrix} w_r \\ w_c \\ w_a \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_m \end{bmatrix}. \tag{6}$$

Using least squares, we can therefore recover the sensitivities w_r, w_c, and w_a, and use them to make future predictions. In a similar manner, if we are presented with a new movie and know how a subset of our students rate it, we can set up a least squares problem to solve for its romance, comedy, and action components, and then use these inferred components to make predictions for the rest of the students.

2 Analyzing Data Using the SVD

What did we rely on when making our predictions? Fundamentally, we relied on being able to write our large ratings matrix R as a low-rank sum of outer products. In reality, however, there will always be noise and variation in our data. We need some way to construct a "low-rank" approximation of a large matrix that preserves the fundamental structure of our data.

Let's see if the SVD helps. After all, given an $m \times n$ matrix A with rank r and singular values

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = 0, \tag{7}$$

we saw earlier that the SVD lets us write it as the sum of outer products

$$A = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^\top . \tag{8}$$

If our matrix is high-rank, i.e., $r \approx \min\{m, n\}$, then almost all of the singular values will be nonzero. However, we claim that when we have some linear, low-rank essential structure to our data, as is usually the case with real data such as images, most of our singular values will be very small. For instance, imagine that σ_1 and σ_2 are significantly larger than the others. Then we can write A as

$$(\sigma_1 \vec{u}_1 \vec{v}_1^\top + \sigma_2 \vec{u}_2 \vec{v}_2^\top) + \sum_{i=3}^{r} \sigma_i \vec{u}_i \vec{v}_i^\top , \tag{9}$$

where the contribution of the summation is very small in comparison to the first two terms. This motivates approximating our dataset as just

$$A_2 = \sigma_1 \vec{u}_1 \vec{v}_1^\top + \sigma_2 \vec{u}_2 \vec{v}_2^\top , \tag{10}$$

and using this model to make predictions in the manner described.

As it turns out, empirically, this approach works quite well, and is the heart of what is known as *principal component analysis*. Still, it would be nice to somehow quantifiably justify its efficacy. In particular, why is it that taking just the first few terms of the SVD provides us with the "best" low-rank approximation to our dataset?

3 Low-Rank Approximations

Let's make our intuition about the SVD more formal. Given a matrix A, we will set a goal of computing a "low-rank" approximation A_k that is of rank $\leq k$, but that is still "approximately equal" to A.

What does "approximately equal" mean? Well, to get A_k we wish to minimize the magnitude of the error $A - \widehat{A}$ across all $\widehat{A}$ with rank $\leq k$, and then take the minimizing $\widehat{A}$ as our A_k. Here the squared magnitude of a matrix is defined to be the sum of the squares of all its elements. Why? Because we have no a-priori reason to care about some entries more than other entries of the matrix. It turns out that this quantity is called the *Frobenius norm*, and we define it as

$$\|A\|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2 . \tag{11}$$

Why do we care about minimizing this quantity $\left\|A - \widehat{A}\right\|_F^2$? One way to think about it is that our low-rank approximation is essentially a model that we are trying to fit to our data. For instance, in the above "movies" example, we are essentially trying to fit a model of student movie preferences to observed data. So it makes sense that we should try to pick a model that minimizes the error between the predicted and observed data points.

Our conjecture is that the best such approximation of an $m \times n$ matrix A with SVD

$$A = U\Sigma V^\top = \begin{bmatrix} | & & | \\ \vec{u}_1 & \cdots & \vec{u}_m \\ | & & | \end{bmatrix} \begin{bmatrix} \Sigma_r & 0_{r\times(n-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix} \begin{bmatrix} - & \vec{v}_1^\top & - \\ - & \vec{v}_2^\top & - \\ & \vdots & \\ - & \vec{v}_n^\top & - \end{bmatrix} = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^\top. \tag{12}$$

is

$$A_k = \sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^\top, \tag{13}$$

where $\sigma_1 \geq \sigma_2 \geq \cdots$. In other words, we sum up only the largest k outer products, rather than all r of them. Note that if $k \geq r$, then since $\sigma_{r+1} = \cdots = \sigma_k = 0$, our claim is that the best approximation of A that is rank k or less is just $A_k = A$. This tracks with our intuition.

Our conjecture is in fact true. It is called the **Eckart-Young theorem**, which formally states that for the A_k defined above,

$$A_k = \underset{\substack{\widehat{A}\in\mathbb{R}^{m\times n} \\ \operatorname{rank}(\widehat{A})\leq k}}{\operatorname{argmin}} \left\| A - \widehat{A} \right\|_F^2. \tag{14}$$

A proof of the theorem is relegated to the next note, since the derivation is rather lengthy.

4 Applying Low-Rank Approximation to PCA

The above theorem tells us that if we want to find the best k-dimensional subspace to use to approximate a set of (real) data, we can take the data, arrange it into a matrix M by stacking the columns of data, and then take the SVD. Looking at the SVD in outer product form $M = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^\top$, where the σ_i are in decreasing order, we can just choose to truncate the sum to the top k terms. We get an approximation of all the data $\widehat{M} = \sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^\top$. More importantly, we also get a basis for our subspace of interest. Namely, the $\vec{u}_i$ gives an orthonormal basis for the k-dimensional subspace that best approximates our data. The first such $\vec{u}_1$ is called the first principal component (for columns), the second such $\vec{u}_2$ is called the second principal component (for columns), and so on.

If we had arranged the data into rows instead, then we could do the same thing and we would use the first $\vec{v}_i$ instead as the principal components. (Or if we want to keep them in row form, $\vec{v}_i^\top$.) The choice depends on nature of the data and what we are trying to do with it.

Once we have learned such a subspace from training data, we can use it for dimensionality reduction by projecting new raw data points onto the subspace. So instead of storing the entire data point, this allows us to store just the coefficients of the projections in the subspace basis we have found. For example, if the raw data points have $m = 100$ entries in them, and we decided to use $k = 3$ to choose a 3-dimensional subspace, then we have managed to distill the essential information into just 3 coefficients. These coefficients are also easier to calculate by the orthonormality of the $\vec{u}_i$. For a set of orthonormal vectors U, we know that its projection coefficients from least square simplifies to $(U^\top U)^{-1} U^\top \vec{x} = U^\top \vec{x}$ since $U^\top U = I$. So for the case of $k = 3$, we just keep

$$U^\top \vec{x} = \begin{bmatrix} \vec{u}_1^\top \\ \vec{u}_2^\top \\ \vec{u}_3^\top \end{bmatrix} \vec{x} = \begin{bmatrix} \vec{u}_1^\top \vec{x} \\ \vec{u}_2^\top \vec{x} \\ \vec{u}_3^\top \vec{x} \end{bmatrix} \tag{15}$$

instead of the whole m-dimensional raw data point $\vec{x}$. And to get the approximated data point, we just multiply the coefficients by the basis to get $UU^\top\vec{x}$. Such dimensionality reduction is often very useful in reducing the computational burden[1] for doing data analysis, classification, regression, etc.

In some problem contexts, the data that we are given has offsets from the origin. What do we mean? Consider two-dimensional data. Our approach to PCA will find the best line through the origin that goes through the data. But what if the data was actually along a line that didn't go through the origin? How would we find this? In that case, we would want to "center" the data first. These offsets are often estimated and removed before one does PCA analysis. Such offsets can be found by taking appropriate means of the training data, and then subtracting them from the data. However, it is useful to consider that "de-meaning" step as being something separate from the core task of subspace discovery. Instead, it should be considered an application-specific data cleaning or preprocessing step. It should only be done if you actually believe that such offsets might exist.

4.1 PCA Algorithm

We can formalize the above section with the following Principal Component Analysis (PCA) algorithm:

Inputs: a set of data points $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n$
Outputs: the k principal components which best approximate our data.

(a) Arrange the data into a matrix:

- We can either use data as columns: $X = \begin{bmatrix} | & | & \cdots & | \\ \vec{x}_1 & \vec{x}_2 & \cdots & \vec{x}_n \\ | & | & \cdots & | \end{bmatrix}$.
- Or we can use data as rows: $X = \begin{bmatrix} - & \vec{x}_1^\top & - \\ - & \vec{x}_2^\top & - \\ \vdots & \vdots & \vdots \\ - & \vec{x}_n^\top & - \end{bmatrix}$.

(b) Compute the SVD: $X = U\Sigma V^\top = \sum \sigma_i \vec{u}_i \vec{v}_i^\top$.

(c) Find the first k principal components.

- If our data is columns: choose $\vec{u}_1, \vec{u}_2, \ldots\vec{u}_k$.
- If our data is rows: choose $\vec{v}_1, \vec{v}_2, \ldots\vec{v}_k$.

(d) Project your data onto the principal components to get the underlying lower dimensional structure (which you can later use for clustering/classification).

- If our data is columns, the projection of data point $\vec{x}_i$ onto the k-dimensional subspace has coefficients $U_k^\top\vec{x}_i$ and projection $U_kU_k^\top\vec{x}_i = \sum_{p=1}^{k}(\vec{u}_p^\top\vec{x}_i)\vec{u}_p$. Doing it for all columns is $U_kU_k^\top X$.

[1]For reasons that you will learn in future courses like 189, the advantage is not simply computational. Distilling the data down by dimensionality reduction can also be helpful in eliminating irrelevant noise that could confuse subsequent stages of learning. Even more subtly, it can help make problems tractable that otherwise might seem impossible. For example, if the data that is being recorded is a microphone trace that is hundreds of samples long, then many approaches to learning how to classify words might require thousands of examples of each word. By reducing the dimensionality of the incoming microphone trace, it can become possible to learn from fewer examples using those approaches.

- If our data is rows, the projection of data point $\vec{x}_i$ onto the k-dimensional subspace has coefficients $V_k^\top \vec{x}_i$ and projection $V_k V_k^\top \vec{x}_i = \sum_{p=1}^{k} (\vec{v}_p^\top \vec{x}_i)\vec{v}_p$. Doing it for all rows is $XV_kV_k^\top$.

Notice that this projection approach works because of the orthonormality of the vectors in U and V. If the vectors in U and V were not orthonormal, the least squares projection would not simplify so nicely.

You may be asking, how do you choose the best number of principal components? Well, there are several ways. If you have an underlying idea of what the dimensionality of the underlying structure should be, you could try to use that. Alternatively, you could observe the singular values to see at which point they drop off significantly, as then they won't affect the approximation much. Then only use the principal components/singular vectors corresponding to relatively larger singular values of the matrix.

5 How to choose the Rank of our PCA Model

Let us explore how to make a decision for what rank we should choose for our PCA model. In the system identification setting, we faced a similar conundrum of how to evaluate our learned parameters. When it comes to learned models, the gold standard is to field test the performance of the model in the engineering application it was learned for, barring safety or ethical concerns. However, in the setting that it is not possible to field test, we may rely on a silver standard of splitting our available data into a training dataset and test dataset (alternatively, validation dataset). We learn the model from the first, and evaluate against the second by trying to predict the data in the testing set from what we have. We develop and describe precisely how to learn and validate the rank of a learned PCA model in the following sections.

5.1 Splitting the Data and Making Predictions

Let us have some data matrix X upon which we are running our PCA algorithm. To split our data in our data matrix we have to have some notion of predicting the left out parts of X from the part of X we train on. It is not obvious how to do this, but let us consider our goal: we would like to predict the data in our low-rank matrix if we have some subpart of it that we run the PCA algorithm on. We shall build some examples to get an idea of what subpart would help achieve this .

First, let us take a rank one matrix, $\underset{2\times 2}{X_1}$ defined in the following way.

$$X_1 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 6 & 8 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} y_1 & y_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} 3 & 4 \end{bmatrix} \tag{16}$$

We see in X_1 linearly dependent rows and columns. If we have the first row and the ratio of the entries of the two rows, we can figure out the second row. However, this latter ratio is not a natural quantity. We wish to work with only having entries of the matrix X_1, as this is the form our data comes in. It is however possible to say that if we know the top left corner entries, a_{11}, a_{12}, and a_{21}, we can compute this ratio and therefore predict the last entry a_{22}. Evaluating the outer product in terms of the x_i and y_i, $a_{11} = x_1y_1$, $a_{12} = x_1y_2$, and $a_{21} = x_2y_1$. Thus we can calculate $a_{22} = \frac{a_{12}a_{21}}{a_{11}}$. While this does not prevent issues like having $a_{11} = 0$, this first example gives us a candidate approach for splitting a matrix - we partition it into four blocks and use the entries we know to attempt reconstituting the other entries. For understanding, let us evaluate this approach when we are predicting in the simpler case of only one value in our data matrix and also consider how PCA comes into the mix.

We try a second example, a bigger rank one matrix, $\underset{m\times n}{X_2}$.

$$X_2 = \vec{q}\vec{r}^\top = \begin{bmatrix} \vec{q}_1 \\ q_2 \end{bmatrix} \begin{bmatrix} \vec{r}_1^\top & r_2 \end{bmatrix} = \begin{bmatrix} \vec{q}_1\vec{r}_1^\top & \vec{q}_1 r_2 \\ q_2\vec{v}_1^\top & q_2 r_2 \end{bmatrix} = \begin{bmatrix} \underset{m-1\times n-1}{X_{11}} & \underset{m-1\times 1}{X_{12}} \\ \underset{1\times n-1}{X_{21}} & \underset{1\times 1}{X_{22}} \end{bmatrix} \tag{17}$$

This time, we partition X_2 into four blocks by splitting the vectors we take an outer product of ($\vec{q} \in \mathbb{R}^m$ and $\vec{r} \in \mathbb{R}^n$) vertically and horizontally into a scalar (q_2, r_2) following a vector $(\vec{q}_1, \vec{r}_2^\top)$. We leave the bottom right corner element of X_2, X_{22}, as the quantity we will predict.

Let us say we know all of the entries of X_{11}, X_{12}, X_{21}, and wish to find or predict X_{22}. We could take the approach used in the previous example to attempt to find the entry X_{22}, but let us try to consider the context of what happens if we compute the SVD of $X_{11} = \vec{q}_1\vec{r}_1^\top$ and see if the structure of the SVD is helpful given its relationship to PCA.

Provided numbers, we can compute the SVD of X_{11}. However, what is of interest to us is the relationship between our $\vec{q}_1$ and $\vec{r}_1$ to our SVD and what it implies for our known matrices. We manipulate X_{11} in the following way.

$$\begin{aligned} X_{11} &= \vec{q}_1\vec{r}_1^\top & (18)\\ &= \left(\|\vec{q}_1\|\frac{\vec{q}_1}{\|\vec{q}_1\|}\right)\left(\|\vec{r}_1\|\frac{\vec{r}_1}{\|\vec{r}_1\|}\right)^\top & (19)\\ &= \left(\|\vec{q}_1\|\vec{u}\right)\left(\|\vec{r}_1\|\vec{v}\right)^\top & (20)\\ &= \|\vec{q}_1\|\|\vec{r}_1\|\vec{u}\vec{v}^\top & (21)\\ &= \sigma\vec{u}\vec{v}^\top & (22)\end{aligned}$$

We can see if some interesting consequences arise from defining $\sigma = \|\vec{q}_1\|\|\vec{r}_1\|$, $\vec{u} = \frac{\vec{q}_1}{\|\vec{q}_1\|}$ and $\vec{v} = \frac{\vec{r}_1}{\|\vec{r}_1\|}$. Going back to examine our other known components, we have $X_{12} = \vec{q}_1 r_2 = \|\vec{q}_1\| r_2\vec{u}$ and $X_{21} = q_2\vec{r}_1^\top = q_2\|\vec{r}_1\|\vec{v}^\top$. If we can compute the SVD of X_{11}, we know σ,$\vec{u}$, and $\vec{v}$, and so can try to get r_2 and q_2 by taking the appropriate products with X_{12} and X_{21}. We try the following:

$$\begin{aligned} \vec{u}^\top X_{12} &= \vec{u}^\top\left(\|\vec{q}_1\| r_2\vec{u}\right) = \|\vec{q}_1\| r_2 & (23)\\ X_{21}\vec{v} &= r_2\|\vec{r}_1\|\vec{v}^\top\vec{v} = q_2\|\vec{r}_1\| & (24)\end{aligned}$$

We are not quite there - notice that we have not gotten q_2, r_2, or even $q_2 r_2$ alone. However, by taking the product of the two values above, we get $\vec{u}^\top X_{12} \cdot X_{21}\vec{v} = \|\vec{q}_1\|\|\vec{r}_1\| q_2 r_2 = \sigma q_2 r_2$. Thus we can find what X_{22} exactly is!

$$X_{22} = \frac{\vec{u}^\top X_{12} \cdot X_{21}\vec{v}}{\sigma} = q_2 r_2 \tag{25}$$

Note that this computation has a division by σ which can be zero. However, our earlier computation relied on the implicit assumption that we had some nonzero vectors $\vec{q}_1$ and $\vec{r}_1$ with some nonzero norm. Such an assumption can be made true by a permutation of the rows and columns and rows of X_2 to ensure that X_{11} is not the $m-1\times n-1$ matrix of all zeroes as we are guaranteed at least one nonzero entry in X_2 if it is rank one.

Now that we have established a way to predict one element that has been sectioned off, there is a question of if we can make such predictions for a larger submatrix we are trying to predict, and if we can do so for a matrix of higher rank. It turns out that what we have done generalizes nicely. For a larger submatrix, we can repeat the above procedure for the other rows and other columns we have. Consider the following example to evaluate the generalizability of predicting a submatrix.

Let us take a rank two matrix $\underset{(m+l)\times(n+k)}{X_3}$ defined according to non-zero vectors $\vec{p}, \vec{r} \in \mathbb{R}^{m+l}$, $\vec{q}, \vec{s} \in \mathbb{R}^{n+k}$, with sub-vectors $\vec{p}_1, \vec{r}_1 \in \mathbb{R}^m$, $\vec{p}_2, \vec{r}_2 \in \mathbb{R}^l$, $\vec{q}_1, \vec{s}_1 \in \mathbb{R}^n$, and $\vec{q}_2, \vec{s}_2 \in \mathbb{R}^k$.

$$X_3 = \vec{p}\vec{q}^\top + \vec{r}\vec{s}^\top = \begin{bmatrix}\vec{p}_1\\ \vec{p}_2\end{bmatrix}\begin{bmatrix}\vec{q}_1^\top & \vec{q}_2^\top\end{bmatrix} + \begin{bmatrix}\vec{r}_1\\ \vec{r}_2\end{bmatrix}\begin{bmatrix}\vec{s}_1^\top & \vec{s}_2^\top\end{bmatrix} \tag{26}$$

$$= \begin{bmatrix}\vec{p}_1\vec{q}_1^\top + \vec{r}_1\vec{s}_1^\top & \vec{p}_1\vec{q}_2^\top + \vec{r}_1\vec{s}_2^\top\\ \vec{p}_2\vec{q}_1^\top + \vec{r}_2\vec{s}_1^\top & \vec{p}_2\vec{q}_2^\top + \vec{r}_2\vec{s}_2^\top\end{bmatrix} = \begin{bmatrix}\underset{m\times n}{X_{11}} & \underset{m\times k}{X_{12}}\\ \underset{l\times n}{X_{21}} & \underset{l\times k}{X_{22}}\end{bmatrix} \tag{27}$$

Let us assume that $\vec{p}_1^\top\vec{r}_1 = 0$ and $\vec{q}_1^\top\vec{s}_1 = 0$, where these vectors are scaled versions of the SVD vectors of X_{11}. We are guaranteed an SVD for X_{11}, and we can insure it is rank two by appropriate permutations of the rows and columns of X_3. We can define similar quantities as in the prior example and define relationships between the SVD of X_{11} and $\vec{p}_1, \vec{q}_1, \vec{r}_1, \vec{s}_1$.

$$X_3 = \begin{bmatrix}\vec{p}_1\vec{q}_1^\top + \vec{r}_1\vec{s}_1^\top & \vec{p}_1\vec{q}_2^\top + \vec{r}_1\vec{s}_2^\top\\ \vec{p}_2\vec{q}_1^\top + \vec{r}_2\vec{s}_1^\top & \vec{p}_2\vec{q}_2^\top + \vec{r}_2\vec{s}_2^\top\end{bmatrix} \tag{28}$$

$$= \begin{bmatrix}\|\vec{p}_1\|\|\vec{q}_1\|\frac{\vec{p}_1}{\|\vec{p}_1\|}\frac{\vec{q}_1^\top}{\|\vec{q}_1\|} + \|\vec{r}_1\|\|\vec{s}_1\|\frac{\vec{r}_1}{\|\vec{r}_1\|}\frac{\vec{s}_1^\top}{\|\vec{s}_1\|} & \|\vec{p}_1\|\frac{\vec{p}_1}{\|\vec{p}_1\|}\vec{q}_2^\top + \|\vec{r}_1\|\frac{\vec{r}_1}{\|\vec{r}_1\|}\vec{s}_2^\top\\ \vec{p}_2\|\vec{q}_1\|\frac{\vec{q}_1^\top}{\|\vec{q}_1\|} + \vec{r}_2\|\vec{s}_1\|\frac{\vec{s}_1^\top}{\|\vec{s}_1\|} & \vec{p}_2\vec{q}_2^\top + \vec{r}_2\vec{s}_2^\top\end{bmatrix} \tag{29}$$

$$= \begin{bmatrix}\sigma_1\vec{u}_1\vec{v}_1^\top + \sigma_2\vec{u}_2\vec{v}_2^\top & \|\vec{p}_1\|\vec{u}_1\vec{q}_2^\top + \|\vec{r}_1\|\vec{u}_2\vec{s}_2^\top\\ \vec{p}_2\|\vec{q}_1\|\vec{v}_1^\top + \vec{r}_2\|\vec{s}_1\|\vec{v}_2^\top & \vec{p}_2\vec{q}_2^\top + \vec{r}_2\vec{s}_2^\top\end{bmatrix} \tag{30}$$

$$X_{11} = \sigma_1\vec{u}_1\vec{v}_1^\top + \sigma_2\vec{u}_2\vec{v}_2^\top \tag{31}$$

$$X_{12} = \|\vec{p}_1\|\vec{u}_1\vec{q}_2^\top + \|\vec{r}_1\|\vec{u}_2\vec{s}_2^\top \implies \vec{u}_1^\top X_{12} = \|\vec{p}_1\|\vec{q}_2^\top, \vec{u}_2^\top X_{12} = \|\vec{r}_1\|\vec{s}_2^\top \tag{32}$$

$$X_{21} = \vec{p}_2\|\vec{q}_1\|\vec{v}_1^\top + \vec{r}_2\|\vec{s}_1\|\vec{v}_2^\top \implies X_{21}\vec{v}_1 = \vec{p}_2\|\vec{q}_1\|, X_{21}\vec{v}_2 = \vec{r}_2\|\vec{s}_1\| \tag{33}$$

$$X_{22} = \vec{p}_2\vec{q}_2^\top + \vec{r}_2\vec{s}_2^\top \implies X_{22} = \frac{X_{21}\vec{v}_1 \cdot \vec{u}_1^\top X_{12}}{\sigma_1} + \frac{X_{21}\vec{v}_2 \cdot \vec{u}_2^\top X_{12}}{\sigma_2} \tag{34}$$

Thus, we can predict X_{22} for an arbitrary rank matrix of arbitrary size if we have the SVD of X_{11}, and the matrices X_{12} and X_{21}. Notice the ordering of the products matter, now that we are taking outer products of vectors. We are now ready to fully state how to evaluate the training error and validation error for a PCA model.

5.2 Training Error for PCA Model

For a data matrix X, we partition it into 4 blocks, $X = \begin{bmatrix}X_{11} & X_{12}\\ X_{21} & X_{22}\end{bmatrix}$. we take X_{22} as our validation dataset, and all other entries as the training dataset. We compute a PCA model from the SVD of X_{11} with k principal components - we call the rank k approximation from which the principal components come from $\widehat{X}_{11}$. The training error is $\|X_{11} - \widehat{X}_{11}\|_F^2 = \sum_{i=k+1}^{r}\sigma_i^2$ where r is the rank of X_{11}.

5.3 Validation Error for PCA Model

For the same partitioning of data matrix X in the previous section, we now have X_{22} which we will use to validate if our PCA model coming from $\widehat{X}_{11}$ has a good choice of rank. If the rank k approximation of X_{11} is $\widehat{X}_{11} = \sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^\top$, we can compute an estimate for $\widehat{X}_{22}$ by using X_{12} and X_{21} and generalizing the pattern seen in eq. (34).

$$\widehat{X}_{22} = \sum_{i=1}^{k} \frac{X_{21}\vec{v}_i \cdot \vec{u}_i^\top X_{12}}{\sigma_i} \tag{35}$$

Our test or validation error can thus by defined by $\|X_{22} - \widehat{X}_{22}\|_F^2$. While this does not have an explicit expression, we wish to choose the number of principal components, k, such that we minimize this validation error.

6 PCA via Minimizing Reconstruction Error

We now show an alternative derivation of PCA which shines a new light on what PCA really does.

Remember that what we want to do is find the best low-dimensional subspace for our data, so that we can reduce the dimension of our data. If this subspace is very close to our data, the projection of the data onto the subspace has little error, but saves space by only having to store 1 coefficient for each dimension.

Another way of saying this is that we want to know the best directions to project our data points onto such that the error between the data point and the projection is minimized. This is called the **reconstruction error** and we will now formalize it with math. If we call the direction we are projecting onto $\vec{w}$, and normalize it so $\|\vec{w}\| = 1$, then the squared reprojection error of data point $\vec{x}_i$ is $\left\|\vec{x}_i - (\vec{x}_i^\top \vec{w})\vec{w}\right\|^2$. We want to minimize the total reprojection error for all our n data points so we want to solve

$$\underset{\vec{w}\,:\,\|\vec{w}\|=1}{\operatorname{argmin}} \sum_{i=1}^{n} \left\|\vec{x}_i - (\vec{x}_i^\top \vec{w})\vec{w}\right\|^2 \tag{36}$$

Solving this optimization problem will give you the best projection direction, which we will show is exactly the 1st principal component!

We first simplify this norm:

$$\begin{aligned}
\left\|\vec{x}_i - (\vec{x}_i^\top \vec{w})\vec{w}\right\|^2 &= \left(\vec{x}_i - (\vec{x}_i^\top \vec{w})\vec{w}\right)^\top \left(\vec{x}_i - (\vec{x}_i^\top \vec{w})\vec{w}\right) && (37)\\
&= \vec{x}_i^\top \vec{x}_i + (\vec{x}_i^\top \vec{w})^2 \vec{w}^\top \vec{w} - 2(\vec{x}_i^\top \vec{w})\vec{x}_i^\top \vec{w} && (38)\\
&= \|\vec{x}_i\|^2 + (\vec{x}_i^\top \vec{w})^2 \cdot 1 - 2(\vec{x}_i^\top \vec{w})^2 && (39)\\
&= \|\vec{x}_i\|^2 - (\vec{x}_i^\top \vec{w})^2 && (40)
\end{aligned}$$

Since the first term is constant with respect to our optimization variable $\vec{w}$, and since minimizing a negative is equivalent to maximizing a positive, we can rewrite our problem as

$$\underset{\vec{w}\,:\,\|\vec{w}\|=1}{\operatorname{argmin}} \sum_{i=1}^{n} -(\vec{x}_i^\top \vec{w})^2 = \underset{\vec{w}\,:\,\|\vec{w}\|=1}{\operatorname{argmax}} \sum_{i=1}^{n} (\vec{x}_i^\top \vec{w})^2 \tag{41}$$

We now will manipulate the summation term by rearranging the terms in the inner product:

$$\begin{aligned}
\underset{\vec{w}:\,\|\vec{w}\|=1}{\operatorname{argmax}} \sum_{i=1}^{n} (\vec{x}_i^\top \vec{w})^2 &= \underset{\vec{w}:\,\|\vec{w}\|=1}{\operatorname{argmax}} \sum_{i=1}^{n} \vec{w}^\top \vec{x}_i \vec{x}_i^\top \vec{w} && (42)\\
&= \underset{\vec{w}:\,\|\vec{w}\|=1}{\operatorname{argmax}} \vec{w}^\top \left(\sum_{i=1}^{n} \vec{x}_i \vec{x}_i^\top \right) \vec{w} && (43)\\
&= \underset{\vec{w}:\,\|\vec{w}\|=1}{\operatorname{argmax}} \vec{w}^\top X X^\top \vec{w} && (44)
\end{aligned}$$

Here we assume that our data matrix X has the data points $\vec{x}_i$ as columns, and so we can use the outer product form of matrix multiplication. Plugging in the SVD expression $U\Sigma V^\top$ in for X,

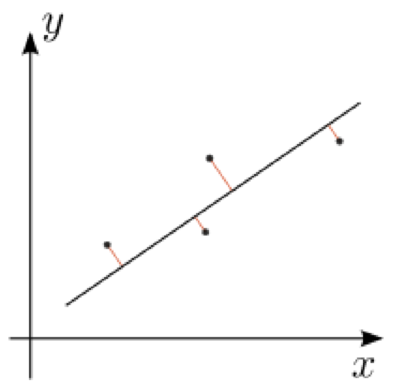
$$\begin{aligned}
\underset{\vec{w}:\,\|\vec{w}\|=1}{\operatorname{argmax}} \vec{w}^\top X X^\top \vec{w} &= \underset{\vec{w}:\,\|\vec{w}\|=1}{\operatorname{argmax}} \vec{w}^\top U \Sigma V^\top V \Sigma^\top U^\top \vec{w} && (45)\\
&= \underset{\vec{w}:\,\|\vec{w}\|=1}{\operatorname{argmax}} \vec{w}^\top U \Sigma \Sigma^\top U^\top \vec{w} && (46)
\end{aligned}$$

Let us assume that each data point $\vec{x}_i \in \mathbb{R}^m$ so X is $m \times n$, U is $m \times m$, and Σ is $m \times n$. Then multiplying Σ by $\Sigma^\top$, we should expect to get a $m \times m$ square matrix with squared singular values along the diagonal. We also perform a change of basis with $U^\top \vec{w} = \vec{\tilde{w}}$. Since orthonormal matrices don't affect norms, our norm constraint is still that $\left\|\vec{\tilde{w}}\right\| = 1$. This allows us to rewrite our maximization as

$$\underset{\vec{\tilde{w}}:\,\left\|\vec{\tilde{w}}\right\|=1}{\operatorname{argmax}} \; \vec{\tilde{w}}^\top \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_m^2 \end{bmatrix} \vec{\tilde{w}} \qquad (47)$$

Remember that σ_1 is the largest of all the singular values. Thus, since we have to pick a vector $\vec{\tilde{w}}$ of norm 1, we would maximize our objective by selecting $\vec{\tilde{w}} = \vec{e}_1$. Now, to find the actual value of $\vec{w}$, we can substitute this into $\vec{w} = U\vec{\tilde{w}}$ to find that $\vec{w} = U\vec{e}_1 = \vec{u}_1$. Thus, the direction that minimizes the reconstruction error will be the first principal component of the data matrix, $\vec{u}_1$. Similarly, the orthogonal direction with the second least reconstruction error will be the second principal component $\vec{u}_2$ and so forth.

This interpretation tells us the principal components (U or V vectors depending on column or row data) are exactly the directions that reduce the orthogonal projection error. This can be seen graphically in the following figure:



Note how this error metric is different from least squares as that minimizes the vertical error to the line of best fit, while PCA minimizes the orthogonal projection error to the line [2].

Contributors:

- Rahul Arya.
- Anant Sahai.
- Ayan Biswas.
- Druv Pai.
- Ashwin Vangipuram.
- Kamyar Salahi.
- Moses Won.

[2]This idea is also called total least squares, which is discussed more in-depth in CS 189.