14 Singular Value Decomposition

For any high-dimensional data analysis, one's first thought should often be: *can I use an SVD?* The singular value decomposition is an invaluable analysis tool for dealing with large high-dimensional data. In many cases, data in high dimensions, most of the dimensions do not contribute to the structure of the data. But filtering these takes some care since it may not be clear which ones are important, if the importance may come from a combination of dimensions. The singular value decomposition can be viewed as a way of finding these important dimensions, and thus the key relationships in the data.

On the other hand, the SVD is often viewed as a numerical linear algebra operation that is done on a matrix. It decomposes a matrix down into three component matrices. These matrices have structure, being orthogonal or diagonal.

The goal of this note is to bridge these views and in particular to provide geometric intuition for the SVD.

The SVD is related to several other tools which will also consider:

- **PCA (Principal Component Analysis)**: a geometric interpretation, after *centering* the data
- **Eigen-decomposition**: shares the same components after data has been made "square."
- **MDS (Multidimensional Scaling)**: Given just pairwise distances, convert to eigen-decomposition

Data. We will focus on a dataset $A \subset \mathbb{R}^d$ where A is a set of n "points." At the same time, we will think of A as a $n \times d$ data matrix. Each row corresponds to a point, and each column corresponds to a dimension. We will use Euclidean distance, so it is essential **all dimensions have the same units!** *(Some interpretations reverse these so a column is a point, a row is a dimension – but they are really the same.)*

14.0.1 Projections

Different than in linear regression this family of techniques will measure error as a projection from $a_i \in \mathbb{R}^d$ to the closest point $\pi_F(a_i)$ on F. To define this we will use linear algebra.

First recall, that given a unit vector $u \in \mathbb{R}^d$ and any data point $p \in \mathbb{R}^d$, then the dot product

$$\langle u, p \rangle$$

is the norm of p projected onto the line through u. If we multiply this scalar by u then

$$\pi_u(p) = \langle u, p \rangle u,$$

and it results in the point on the line through u that is closest to data point p. This is a *projection of p onto u.*

To understand this for a subspace F, we will need to define a basis. *For now we will assume that F contains the origin $(0, 0, 0, \ldots, 0)$ (as did the line through u).* Then if F is k-dimensional, then this means there is a k-dimensional basis $U_F = \{u_1, u_2, \ldots, u_k\}$ so that

- For each $u_i \in U_F$ we have $\|u_i\| = 1$, that is u_i is a unit vector.
- For each pair $u_i, u_j \in U_F$ we have $\langle u_i, u_j \rangle = 0$; the pairs are orthogonal.
- For any point $x \in F$ we can write $x = \sum_{i=1}^{k} \alpha_i u_i$; in particular $\alpha_i = \langle x, u_i \rangle$.

Given such a basis, then the projection on to F of a point $p \in \mathbb{R}^d$ is simply

$$\pi_F(p) = \sum_{i=1}^{k} \langle u_i, p \rangle u_i.$$

Thus if p happens to be exactly in F, then this recovers p exactly.

The other powerful part of the basis U_F is the it defines a *new coordinate system*. Instead of using the d original coordinates, we can use new coordinates $(\alpha_1(p), \alpha_2(p), \ldots, \alpha_k(p))$ where $\alpha_i(p) = \langle u_i, p \rangle$. To be clear $\pi_F(p)$ is still in $\mathbb{R}^d$, but there is a k-dimensional representation if we restrict to F.

When F is d-dimensional, this operation can still be interesting. The basis we choose $U_F = \{u_1, u_2, \ldots, u_d\}$ could be the same as the original coordinate axis, that is we could have $u_i = e_i = (0, 0, \ldots, 0, 1, 0, \ldots, 0)$ where only the ith coordinate is 1. But if it is another basis, then this acts as a rotation (with possibility of also a mirror flip). The first coordinate is rotated to be along u_1; the second along u_2; and so on. In $\pi_F(p)$, the point p does not change, just its representation.

14.0.2 SSE Goal

As usual our goal will be to minimize the sum of squared errors. In this case we define this as

$$\mathsf{SSE}(A, F) = \sum_{a_i \in A} \|a_i - \pi_F(a_i)\|^2,$$

and our desired k-dimensional subspace F is

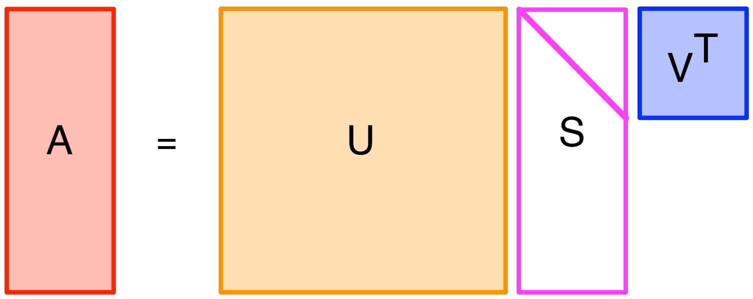
$$F^* = \arg\min_F \mathsf{SSE}(A, F)$$

As compared to linear regression, this is much less a "proxy goal" where the true goal was prediction. Now we have no labels (the y_i values), so we simply try to fit a model through all of the data.

14.1 The SVD Operator

First we document what the following operation in matlab does:

$$[U, S, V] = \texttt{svd}(A)$$



The backend of this (in almost any language) calls some very carefully optimized Fortran code as part of the LAPACK library. First of all, no information is lost since we can simply recover the original data as $A = USV^T$, up to numerical precision, and the Fortran library is optimized to provide very high numerical precision.

The structure that lurks beneath. The matrix S only has non-zero elements along its diagonal. So $S_{i,j} = 0$ if $i \neq j$. The remaining values $\sigma_1 = S_{1,1}$, $\sigma_2 = S_{2,2}$, $\ldots$, $\sigma_r = S_{r,r}$ are known as the *singular values*. They have the property that

$$\sigma_1 \geq \sigma_2 \geq \ldots \sigma_r \geq 0$$

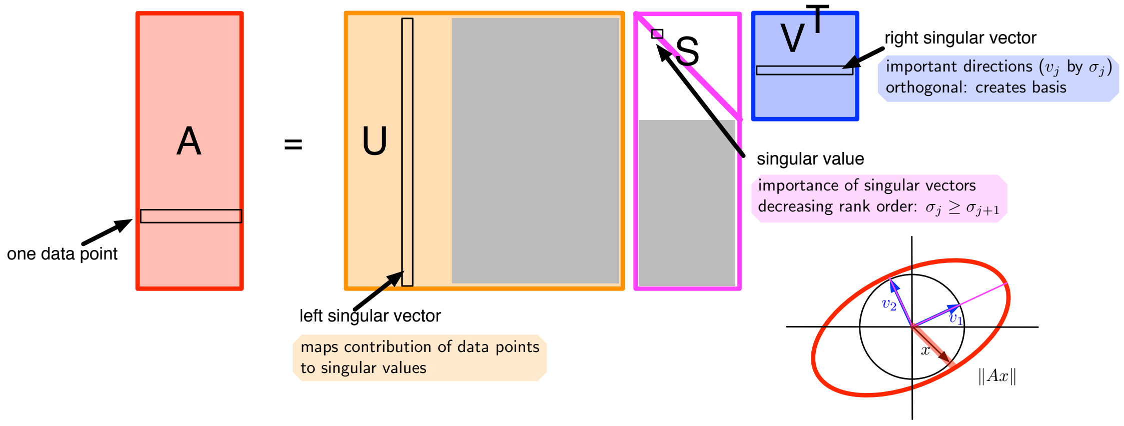
where $r \leq \min\{n, d\}$ is the rank of the matrix A. So the number of non-zero singular values reports the rank (this is a numerical way of computing the rank or a matrix).

The matrices U and V are orthogonal. Thus, their columns are all unit vectors and orthogonal to each other (within each matrix). The columns of U, written $u_1, u_2, \ldots, u_d$, are called the *left singular vectors*; and the columns of V, written $v_1, v_2, \ldots, v_n$, are called the *right singular vectors*.

This means for any vector $x \in \mathbb{R}^d$, the columns of V (the right singular vectors) provide a basis. That is, we can write

$$x = \sum_{i=1}^{d} \alpha_i v_i \quad \text{for} \quad \alpha_i = \langle x, v_i \rangle.$$

Similarly for any vector $y \in \mathbb{R}^n$, the columns of U (the left singular vectors) provide a basis. This also implies that $\|x\| = \|V^T x\|$ and $\|y\| = \|yU\|$.


A
=
U
S
V^T
one data point
left singular vector
maps contribution of data points
to singular values
singular value
importance of singular vectors
decreasing rank order: $\sigma_j \geq \sigma_{j+1}$
right singular vector
important directions (v_j by σ_j)
orthogonal: creates basis
v_2
v_1
x
$\|Ax\|$

Tracing the path of a vector. To illustrate what this decomposition demonstrates, a useful exercise is to trace what happens to a vector $x \in \mathbb{R}^d$ as it is left-multiplied by A, that is $Ax = USV^T x$.

First $V^T x$ produces a new vector $\xi \in \mathbb{R}^d$. It essentially changes no information, just changes the basis to that described by the right singular values. For instance the new i coordinate $\xi_i = \langle v_i, x \rangle$.

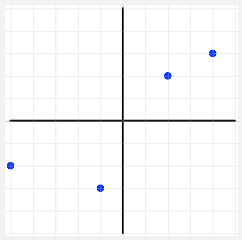
Next $\eta \in \mathbb{R}^n$ is the result of $SV^T x = S\xi$. It scales ξ by the singular values of S. Note that if $d < n$ (the case we will focus on), then the last $n - d$ coordinates of η are 0. In fact, for $j > r$ (where $r = \textsf{rank}(A)$) then $\eta_j = 0$. For $j \leq r$, then the vector η is stretched longer in the first coordinates since these have larger values.

The final result is a vector $y \in \mathbb{R}^n$, the result of $Ax = USV^T x = U\eta$. This again just changes the basis of η so that it aligns with the left singular vectors. In the setting $n > d$, the last $n - d$ left singular vectors are meaningless since the corresponding entries in η are 0.

Working backwards ... this final U matrix can be thought of mapping the effect of η onto each of the data points of A. The η vector, in turn, can be thought of as scaling by the content of the data matrix A (the U and V^T matrices contain no scaling information). And the ξ vector arises via the special rotation matrix V^T that puts the starting point x into the right basis to do the scaling (from the original d-dimensional coordinates to one that suits the data better).

Example: Tracing through the SVD
Consider a matrix

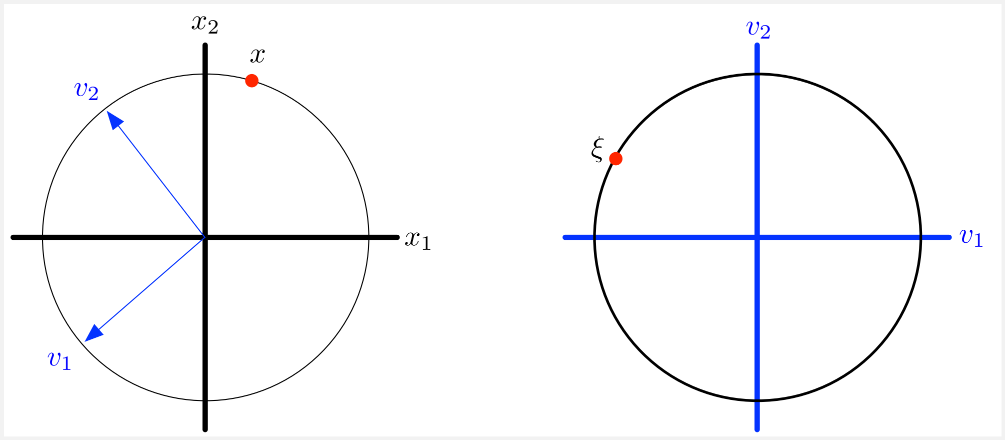
$$A = \begin{pmatrix} 4 & 3 \\ 2 & 2 \\ -1 & -3 \\ -5 & -2 \end{pmatrix},$$



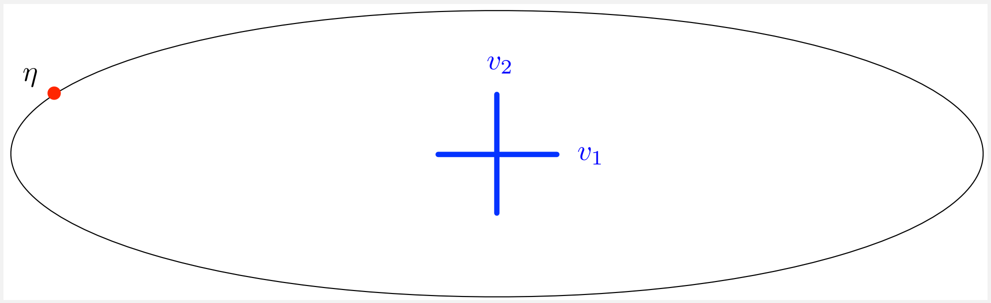
and its SVD $[U, S, V] = \mathsf{svd}(A)$:

$$U = \begin{pmatrix} -0.6122 & 0.0523 & 0.0642 & 0.7864 \\ -0.3415 & 0.2026 & 0.8489 & -0.3487 \\ 0.3130 & -0.8070 & 0.4264 & 0.2625 \\ 0.6408 & 0.5522 & 0.3057 & 0.4371 \end{pmatrix}, \quad S = \begin{pmatrix} 8.1655 & 0 \\ 0 & 2.3074 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad V = \begin{pmatrix} -0.8142 & -.5805 \\ -0.5805 & 0.8142 \end{pmatrix}.$$

Now consider a vector $x = (0.243, 0.97)$ (scaled very slightly so it is a unit vector, $\|x\| = 1$). Multiplying by V^T rotates (and flips) x to $\xi = V^T x$; still $\|\xi\| = 1$



Next multiplying by S scales ξ to $\eta = S\xi$. Notice there are an imaginary third and fourth coordinates now; they are both coming out of the page! Don't worry, they won't poke you since their magnitude is 0.



Finally, $y = U\eta = Ax$ is again another rotation of η in this four dimensional space.

14.1.1 Best Rank-k Approximation

So how does this help solve the initial problem of finding F^*, which minimized the SSE? The singular values hold the key.

It turns out that there is a *unique* singular value decomposition, up to ties in the singular values. This means, there is exactly one (up to singular value ties) set of right singular values which rotate into a basis so that $\|Ax\| = \|SV^T x\|$ for all $x \in \mathbb{R}^d$ (recall that U is orthogonal, so it does not change the norm, $\|U\eta\| = \|\eta\|$).

Next we realize that the singular values come in sorted order $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r$. In fact, they are defined so that we choose v_1 so it maximizes $\|Av_1\|$, then we find the next singular vector v_2 which is orthogonal to v_1 and maximizes $\|Av_2\|$, and so on. Then $\sigma_i = \|Av_i\|$.

If we define F with the basis $U_F = \{v_1, v_2, \ldots, v_k\}$, then

$$\|x - \pi_F(x)\|^2 = \left\| \sum_{i=1}^{d} v_i \langle x, v_i \rangle - \sum_{i=1}^{k} v_i \langle x, v_i \rangle \right\|^2 = \sum_{i=k+1}^{d} \langle x, v_i \rangle^2.$$

so the projection error is that part of x in the last $(d-k)$ right singular vectors.

But we are not trying to directly predict new data here (like in regression). Rather, we are trying to approximate the data we have. We want to minimize $\sum_i \|a_i - \pi_F(a_i)\|^2$. But for any unit vector u, we recall now that

$$\|Au\|^2 = \sum_{i=1}^{n} \langle a_i, u \rangle.$$

Thus the projection error can be measured with a set of orthonormal vectors $w_1, w_2, \ldots, w_{d-k}$ which are each orthogonal to F, as $\sum_{j=1}^{n-k} \|Aw_j\|^2$. When defining F as the first k right singular values, then these orthogonal vectors are the remaining $(n-k)$ right singular vectors, so the projection error is

$$\sum_{i=1}^{n} \|a_i - \pi_F(a_i)\|^2 = \sum_{j=k+1}^{d} \|Av_j\|^2 = \sum_{j=k+1}^{d} \sigma_j^2.$$

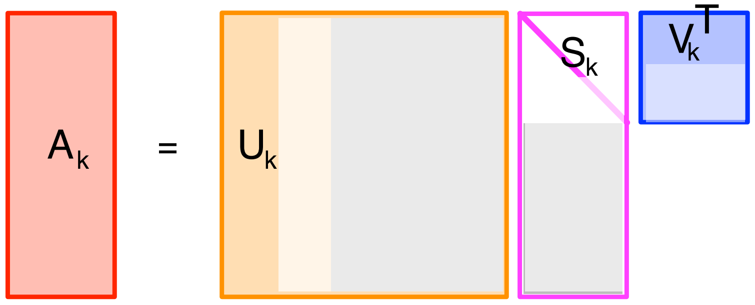
And thus by how the right singular vectors are defined, this expression is minimized when F is defined as the span of the first k singular values.

Best rank-k approximation. A similar goal is to find the *best rank-k approximation of A*. That is a matrix $A_k \in \mathbb{R}^{n \times d}$ so that $\mathsf{rank}(A_k) = k$ and it minimizes both

$$\|A - A_k\|_2 \quad \text{and} \quad \|A - A_k\|_F.$$

Note that $\|A - A_k\|_2 = \sigma_{k+1}$ and $\|A - A_k\|_F^2 = \sum_{j=k+1}^{d} \sigma_j^2$. And recall that for a matrix M we define the *Frobenius norm* $\|M\|_F^2 = \sum_{i,j} M_{i,j}^2$ and the *spectral norm* $\|M\|_2 = \max_x \|Mx\|/\|x\|$.

Remarkably, this A_k matrix also comes from the SVD. If we set S_k as the matrix S in the decomposition so that all but the first k singular values are 0, then it has rank k. Hence $A_k = US_kV^T$ also has rank k and is our solution. But we can notice that when we set most of S_k to 0, then the last $(d-k)$ columns of V are meaningless since they are only multiplied by 0s in US_kV^T, so we can also set those to all 0s, or remove them entirely (along with the last $(d-k)$ columns of S_k). Similar we can make 0 or remove the last $(n-k)$ columns of U. These matrices are referred to as V_k and U_k respectively, and also $A_k = U_k S_k V_k^T$.



In another view, we can also write a matrix A with rank r as

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T,$$

where each $u_i v_i^T$ is a $n \times d$ matrix with rank 1.

We next will relate the SVD to other common matrix analysis forms: PCA, Eigendecomposition, and MDS. One may find literature that uses slightly different forms of these terms (they are often intermingled), but I believe this is the cleanest, most consistent, mapping.

14.2 Principle Component Analysis (PCA)

Recall that the original goal of this topic was to find the k-dimensional subspace F to minimize

$$\|A - \pi_F(A)\|_F^2 = \sum_{a_i \in A} \|a_i - \pi_F(a_i)\|^2.$$

We have not actually solved this yet. The top k right singular values V_k of A only provided this bound assuming that F contains the origin: $(0, 0, \ldots, 0)$. However, this might not be the case!

Principal Component Analysis (PCA) is an extension of the SVD when we do not restrict that the subspace V_k must go through the origin. It turns out, like with simple linear regression, that the optimal F must go through the mean of all of the data. So we can still use the SVD, after a simple preprocessing step called centering to shift the data matrix so its mean is exactly at the origin.

Specifically, *centering* is adjusting the original input data matrix $A \in \mathbb{R}^{n \times d}$ so that each column (each dimension) has an average value of 0. This is easier than it seems. Define $\bar{a}_j = \frac{1}{n}\sum_{i=1}^{n} A_{i,j}$ (the average of each column j). Then set each $\tilde{A}_{i,j} = A_{i,j} - \bar{a}_j$ to represent the entry in the ith row and jth column of centered matrix $\tilde{A}$.

There is a *centering matrix* $C_n = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ where I_n is the $n \times n$ identity matrix, $\mathbf{1}$ is the all-ones column vector (of length n) and thus $\mathbf{1}\mathbf{1}^T$ is the all-ones $n \times n$ matrix. Then we can also just write $\tilde{A} = C_n A$.

Now to perform PCA on a data set A, we compute $[U, S, V] = \texttt{svd}(C_n A) = \texttt{svd}(\tilde{A})$.

Then the resulting singular values $\textsf{diag}(S) = \{\sigma_1, \sigma_2, \ldots, \sigma_r\}$ are known as the *principal values*, and the top k right singular vectors $V_k = [v_1\ v_2\ \ldots\ v_k]$ are known as the top-k *principal directions*.

This often gives a better fitting to the data than just SVD. The SVD finds the best rank-k approximation of A, which is the best k-dimensional subspace (up to Frobenius and spectral norms) **which passes through the origin**. If all of the data is far from the origin, this can essentially "waste" a dimension to pass through the origin. However, we also need to store the shift from the origin, a vector $\tilde{c} = (\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_d) \in \mathbb{R}^d$.

14.3 Eigenvalues and Eigenvectors

A related matrix decomposition to SVD is the eigendecomposition. This is only defined for a square matrix $B \in \mathbb{R}^{n \times n}$.

An *eigenvector* of B is a vector v such that there is some scalar λ that

$$Bv = \lambda v.$$

That is, multiplying B by v results in a scaled version of v. The associated value λ is called the *eigenvalue*. As a convention, we typically normalize v so $\|v\| = 1$.

In general, a square matrix $B \in \mathbb{R}^{n \times n}$ may have up to n eigenvectors (a matrix $\texttt{V} \in \mathbb{R}^{n \times n}$) and values (a vector $\texttt{l} \in \mathbb{R}^n$). Some of the eigenvalues may be complex numbers (even when all of its entries are real!).

Again it is easy to compute with matlab as `[V,L] = eigs(B)`.

For this reason, we will focus on positive semidefinite matrices. A *positive definite matrix* $B \in \mathbb{R}^{n\times n}$ is a symmetric matrix with all positive eigenvalues. Another characterization is for every vector $x \in \mathbb{R}^n$ then $x^T Bx$ is positive. A *positive semidefinite matrix* $B \in \mathbb{R}^{n\times n}$ may have some eigenvalues at 0 and are otherwise positive; equivalently for any vector $x \in \mathbb{R}^n$, then $x^T Bx$ may be zero or positive.

How do we get positive semi-definite matrices? Lets start with a data matrix $A \in \mathbb{R}^{n\times d}$. Then we can construct two positive semidefinite matrices

$$B_R = A^T A \qquad \text{and} \qquad B_L = AA^T.$$

Matrix B_R is $d \times d$ and B_L is $n \times n$. If the rank of A is d, then B_R is positive definite. If the rank of A is n, then B_L is positive definite.

Eigenvectors and eigenvalues relation to SVD. Next consider the SVD of A so that $[U, S, V] = \mathsf{svd}(A)$. Then we can write

$$B_R V = A^T AV = (V SU^T)(USV^T)V = V S^2.$$

Note that the last step follows because for orthogonal matrices U and V, then $U^T U = I$ and $V^T V = I$, where I is the identity matrix, which has no effect. The matrix S is a diagonal square[1] matrix $S = \mathsf{diag}(\sigma_1, \sigma_2, \ldots, \sigma_d)$. Then $S^2 = SS$ (the product of S with S) is again diagonal with entries $S^2 = \mathsf{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2)$.

Now consider a single column v_i of V (which is the ith right singular vector of A). Then extracting this column's role in the linear system $B_R V = V S^2$ we obtain

$$B_R v_i = v_i \sigma_i^2.$$

This means that ith right singular vector of A is an eigenvector (in fact the ith eigenvector) of $B_R = A^T A$. Moreover, the ith eigenvalue λ_i of B_R is the ith singular value of A squared: $\lambda_i = \sigma_i^2$.

Similarly we can derive

$$B_L U = AA^T U = (USV^T)(V SU^T)U = US^2,$$

and hence the left singular vectors of A are the eigenvectors of $B_L = AA^T$ and the eigenvalues of B_L are the squared singular values of A.

14.4 Multidimensional Scaling

Dimensionality reduction is an abstract problem with input of a high-dimensional data set $P \subset \mathbb{R}^d$ and a goal of finding a corresponding lower dimensional data set $Q \subset \mathbb{R}^k$, where $k << d$, and properties of P are preserved in Q. Both low-rank approximations through direct SVD and through PCA are examples of this: $Q = \pi_{V_k}(P)$. However, these techniques require an explicit representation of P to start with. In some cases, we are only presented P more abstractly. There two common situations:

- We are provided a set of n objects X, and a bivariate function $\mathtt{d} : X \times X \to \mathbb{R}$ that returns a distance between them. For instance, we can put two cities into an airline website, and it may return a dollar amount for the cheapest flight between those two cities. This dollar amount is our "distance."

[1]Technically, $S \in \mathbb{R}^{n\times d}$. To make this simple argument work, lets first assume w.l.o.g. (without loss of generality) that $d \le n$. Then the bottom $n - d$ rows of S are all zeros, which mean the right $n - d$ rows of U do not matter. So we can ignore both these $n - d$ rows and columns. Then S is square. This makes U no longer orthogonal, so $U^T U$ is then a projection, not identity; but it turns out this is a project to the span of A, so the argument still works.

- We are simply provided a matrix $D \in \mathbb{R}^{n \times n}$, where each entry $D_{i,j}$ is the distance between the ith and jth point. In the first scenario, we can calculate such a matrix D.

Multi-Dimensional Scaling (MDS) has the goal of taking such a distance matrix D for n points and giving low-dimensional (typically) Euclidean coordinates to these points so that the embedded points have similar spatial relations to that described in D. If we had some original data set A which resulted in D, we could just apply PCA to find the embedding. It is important to note, in the setting of MDS we are typically just given D, and *not* the original data A. However, as we will show next, we can derive a matrix that will act like AA^T using only D.

A *similarity matrix* M is an $n \times n$ matrix where entry $M_{i,j}$ is the similarity between the ith and the jth data point. The similarity often associated with Euclidean distance $\|a_i - a_j\|$ is the standard inner (or dot product) $\langle a_i, a_j \rangle$. We can write

$$\|a_i - a_j\|^2 = \|a_i\|^2 + \|a_j\|^2 - 2\langle a_i, a_j \rangle,$$

and hence

$$\langle a_i, a_j \rangle = \frac{1}{2}\left(\|a_i\|^2 + \|a_j\|^2 - \|a_i - a_j\|^2\right). \tag{14.1}$$

Next we observe that for the $n \times n$ matrix AA^T the entry $[AA^T]_{i,j} = \langle a_i, a_j \rangle$. So it seems hopeful we can derive AA^T from D using equation (14.1). That is we can set $\|a_i - a_j\|^2 = D_{i,j}^2$. However, we need also need values for $\|a_i\|^2$ and $\|a_j\|^2$.

Since the embedding has an arbitrary shift to it (if we add a shift vector s to *all* embedding points, then no distances change), then we can arbitrarily choose a_1 to be at the origin. Then $\|a_1\|^2 = 0$ and $\|a_j\|^2 = \|a_1 - a_j\|^2 = D_{1,j}^2$. Using this assumption and equation (14.1), we can then derive the similarity matrix AA^T. Then we can run the eigen-decomposition on AA^T and use the coordinates of each point along the first k eigenvectors to get an embedding. This is known as *classical MDS*.

It is often used for k as 2 or 3 so the data can be easily visualized.

There are several other forms that try to preserve the distance more directly, where as this approach is essentially just minimizing the squared residuals of the projection from some unknown original (high-dimensional embedding). One can see that we recover the distances with no error if we use all n eigenvectors – if they exist. However, as mentioned, there may be less than n eigenvectors, or they may be associated with complex eigenvalues. So if our goal is an embedding into $k = 3$ or $k = 10$, there is no guarantee that this will work, or even what guarantees this will have. But MDS is used a lot nonetheless.