7.5 Principal Component Analysis

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 problem 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 $\texttt{diag}(S) = \{\sigma_1, \sigma_2, \ldots, \sigma_r\}$ 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, PCA also needs to store the shift from the origin, a vector $\tilde{c} = (\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_d) \in \mathbb{R}^d$.

7.6 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 $k \ll d$, 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 through distances. There are two common variants:

- We are provided a set of n objects X, and a bivariate function $\texttt{d} : X \times X \to \mathbb{R}$ 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."
- 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 product

(i.e., 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

$$\langle a_i, a_j \rangle = \frac{1}{2}\left(\|a_i\|^2 + \|a_j\|^2 - \|a_i - a_j\|^2\right). \tag{7.1}$$

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 (7.1). That is we can set $\|a_i - a_j\|^2 = D_{i,j}^2$. However, we 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 (7.1), we can then derive the similarity matrix AA^T. Specifically, each entry is set

$$[AA^T]_{i,j} = \frac{1}{2}(D_{i,1}^2 + D_{j,1}^2 - D_{i,j}^2).$$

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 $k = 10$, this is not totally assured to work; yet MDS is used a lot nonetheless.

7.7 Linear Discriminant Analysis

Another tool that can be used to learn a Euclidian distance for data is *Linear Discriminant Analysis* (or LDA). This term has a few variants, we focus on the multi-class setting. This means we begin with a data set $X \subset \mathbb{R}^d$, and a known a partition of X into k classes (or clusters) $S_1, S_2, \ldots, S_k \subset X$, so $\bigcup S_i = X$ and $S_i \cap S_j = \emptyset$ for $i \neq j$.

Let $\mu_i = \frac{1}{|S_i|}\sum_{x \in S_i} x$ be the mean of class i, and let $\Sigma_i = \frac{1}{|S_i|}\sum_{x \in S_i}(x - \mu_i)(x - \mu_i)^T$ by its covariance.

Similarly, we can represent the overall mean as $\mu = \frac{1}{|X|}\sum_{x \in X} x$. Then we can then represent the *between class covariance* as

$$\Sigma_B = \frac{1}{|X|}\sum_{i=1}^{k} |S_i|(\mu_i - \mu)(\mu_i - \mu)^T.$$

In contrast the overall *within class covariance* is

$$\Sigma_W = \frac{1}{|X|}\sum_{i=1}^{k} |S_i|\Sigma_i = \frac{1}{|X|}\sum_{i=1}^{k}\sum_{x \in S_i}(x - \mu_i)(x - \mu_i)^T.$$

The goal of LDA is a representation of X in a k'-dimensional space that maximizes the between class covariance while minimizing the within class covariance. This is often formalized as finding the set of vectors u which maximize

$$\frac{u^T \Sigma_B u}{u^T \Sigma_W u}.$$

For any $k' \leq k-1$, we can directly find the orthogonal basis $U = \{u_1, u_2, \ldots, u_{k'}\}$ that maximizes the above goal with an eigen-decomposition. In particular, U is the top k' eigenvectors of $\Sigma_W^{-1}\Sigma_B$. Then to obtain the best representation of X we set the new data set as

$$\tilde{X} \leftarrow \pi_U(X)$$

so $\tilde{x} = \pi_U(x) = (\langle x, u_1\rangle, \langle x, u_2\rangle, \ldots, \langle x, u_{k'}\rangle) \in \mathbb{R}^{k'}$.

This retains the dimensions which show difference between the classes, and similarity among the classes. The removed dimensions will tend to show variance within classes without adding much difference between the classes. Conceptually, if the data set can be well-clustered under the k-means clustering formulation, then the U (say when rank $k' = k-1$) describes a subspace with should pass through the k centers $\{\mu_1, \mu_2, \ldots, \mu_k\}$; capturing the essential information needed to separate the centers.

7.8 Distance Metric Learning

When using PCA, one should enforce that the input matrix A has the same units in each column (and each row). What should one do if this is not the case?

Lets re-examine the root of the problem: the Euclidean distance. It takes two vectors $p, q \in \mathbb{R}^d$ (perhaps rows of A) and measures:

$$\mathbf{d}_{\text{Euc}}(p,q) = \|p-q\| = \sqrt{\langle p-q, p-q\rangle} = \sqrt{\sum_{i=1}^{d}(p_i - q_i)^2}.$$

If each row of a data set A represents a data point, and each column and attribute, then the operation $(p_i - q_i)^2$ is fine since p_i and q_i have the same units (they quantify the same attribute). However the $\sum_{i=1}^{d}$ over these terms adds together quantities that may have different units.

The naive solution is to just brush away those units. These are normalization approaches where all values p_i and q_i (in column i) are divided by a constant s_i with the same units as the elements in that column, and maybe adding a constant. In one approach s_i is chosen so all values in each column lie in $[0,1]$. In the other common approach, the s_i values are chosen to normalize: so that the standard deviation in each column is 1. Note that both of these approaches are affected oddly by outliers – a single outlier can significantly change the effect of various data points. Moreover, if new dimensions are added which have virtually the same value for each data point; these values are inflated to be as meaningful as another signal direction. As a result, these normalization approaches can be brittle and affect the meaning of the data in unexpected ways. However, they are also quite common, for better or worse.

If we are to choose a good metric in a principled way, we must know something about which points should be close and which should be far. In the *distance metric learning* problem we assume that we have two sets of pairs; the close pairs $C \subset X \times X$ and the far pairs $F \subset X \times X$. This process starts with a dataset $X \subset \mathbb{R}^d$, and close and far pairs C and F and tries to find a metric so the close pairs have distance as small as possible, while the far pairs have distance as large as possible.

In particular, we restrict to a Mahalanobis distance defined with respect to a positive semidefinite matrix $M \in \mathbb{R}^{d\times d}$ on points $p, q \in \mathbb{R}^d$ as

$$\mathbf{d}_M(p,q) = \sqrt{(p-q)^T M (p-q)}.$$

So given X and sets of pairs C and F, the goal is to find M to make the close point have small $\mathbf{d}_M$ distance,

and far points have large $\mathbf{d}_M$ distance. Specifically, we will consider finding the optimal distance $\mathbf{d}_{M^*}$ as

$$M^* = \max_M \min_{\{x_i,x_j\}\in F} \mathbf{d}_M(x_i,x_j)^2$$
$$\text{such that} \sum_{\{x_i,x_j\}\in C} \mathbf{d}_M(x_i,x_j)^2 \leq \kappa.$$

That is we want to maximizes the closest pair in the far set F, while restricting that all pairs in the close set C have their sum of squared distances are at most κ, some constant. We will not explicitly set κ, but rather restrict M in some way so on average it does not cause much stretch. There are other reasonable similar formulations, but this one will allow for simple optimization.

Notational Setup. Let $H = \sum_{\{x_i,x_j\}\in C}(x_i - x_j)(x_i - x_j)^T$; note that this is a sum of *outer* products, so H is in $\mathbb{R}^{d\times d}$. For this to work we will need to assume that H is full rank; otherwise we don't have enough close pairs to measure. Or we can set $H = H + \delta I$ for a small scalar δ.

Further, we can restrict M to have trace $\mathsf{Tr}(M) = d$, and hence satisfying some constraint on the close points. Recall that the trace of a matrix M is the sum of M's eigenvalues. Let $\mathbb{P}$ be the set of all positive semidefinite matrices with trace d; hence the identity matrix I is in $\mathbb{P}$. Also, let

$$\triangle = \{\alpha \in \mathbb{R}^{|F|} \mid \sum \alpha_i = 1 \ \& \text{ all } \alpha_i \geq 0\}.$$

Let $\tau_{i,j} \in F$ (or simply $\tau \in F$ when the indexes are not necessary) to represent a far pair $\{x_i, x_j\}$. And let $X_{\tau_{i,j}} = X_{i,j} = (x_i - x_j)(x_i - x_j)^T \in \mathbb{R}^{d\times d}$, an outer product. Let $\tilde{X}_\tau = H^{-1/2} X_\tau H^{-1/2}$. It turns out our optimization goal is now equivalent (up to scaling factors, depending on κ) to finding

$$\arg\max_{M\in\mathbb{P}} \min_{\alpha\in\triangle} \sum_{\tau\in F} \alpha_\tau \langle \tilde{X}_\tau, M\rangle.$$

Here $\langle X, M\rangle = \sum_{s,t} X_{s,t} M_{s,t}$, a dot product over matrices, but since X will be related to an outer product between two data points, this makes sense to think of as $\mathbf{d}_M(X)$.

Algorithm 7.8.1 Optimization for DML

Initialize $M_0 = I$.
for $t = 1, 2, \ldots, T$ **do**
  Set $G = g_\sigma(M_{t-1})$
  Let $v_t = v_{\sigma,M_{t-1}}$; the maximal eigenvalue of G.
  Update $M_t = \frac{t-1}{t} M_{t-1} + \frac{1}{t} v_t v_t^T$.
return $M = M_T$.

Optimization procedure. Given the formulation above, we will basically try to find an M which stretches the far points as much as possible while keeping $M \in \mathbb{P}$. We do so using a general procedure referred to as Frank-Wolfe optimization, which increases our solution using one data point (in this case a far pair) at a time.

Set $\sigma = d \cdot 10^{-5}$ as a small smoothing parameter. Define a gradient as

$$g_\sigma(M) = \frac{\sum_{\tau\in F} \exp(-\langle \tilde{X}_\tau, M\rangle/\sigma)\tilde{X}_\tau}{\sum_{\tau\in F} \exp(-\langle \tilde{X}_\tau, M\rangle/\sigma)}.$$

Observe this is a weighted average over the $\tilde{X}_\tau$ matrices. Let $v_{\sigma,M}$ be the maximal eigenvector of $g_\sigma(M)$; the direction of maximal gradient.

Then the algorithm is simple. Initialize $M_0 \in \mathbb{P}$ arbitrarily; for instance as $M_0 = I$. Then repeatedly find for $t = 1, 2, \ldots$ as (1) find $v_t = v_{\mu, M_{t-1}}$, and (2) set $M_t = \frac{t-1}{t} M_{t-1} + \frac{1}{t} v_t v_t^T$. This is summarized in Algorithm 7.8.1.

7.9 Matrix Completion

A common scenario is that a data set P is provided, but is missing some (or many!) of its attributes. Lets focus on the case where $P \in \mathbb{R}^{n \times d}$ that is an $n \times d$ matrix, that we can perhaps think of as n data points in $\mathbb{R}^d$, that is with d attributes each. For instance, Netflix may want to recommend a subset of n movies to d customers. Each movie has only been seen, or even rated, by a small subset of all d customers, so for each (movie, customer) pair (i, j), it could be a rating (saying a score from $[0, 10]$), denoted $P_{i,j}$. But most pairs (i, j) are empty, there is no rating yet. A recommendation would be based on the predicted score for unseen movies.

The typical notation defines a set $\Omega = \{(i, j) \mid P_{i,j} \neq \emptyset\}$ of the rated (movie, customer) pairs. Then let $\Pi_\Omega(P)$ describe the subset of pairs with scores and $\Pi_\Omega^\perp(P)$ the compliment, the subset of pairs without scores. The goal is to somehow fill in the values $\Pi_\Omega^\perp(P)$.

The simplest variants fill in the value as the average of all existing values in a row (the average rating in a movie). Or it could be the average of existing scores of a column (average rating of a customer). Or an average of these averages. But these approaches are not particularly helpful for personalizing the rating for a customer (e.g., a customer who likes horror movies but not rom-coms is likely to score things differently than one with the opposite preferences).

A common assumption in this area is that there are some simple "latent factors" which determine a customers preferences (e.g., they like horror, and thrillers, but not rom-coms). A natural way to capture this is to assume there is some "low-rank" structure in P. That is we would like to find a low-rank model for P that fits the observed scores $\Pi_\Omega(P)$. The most common formulation looks like ridge regression:

$$P^* = \arg \min_{X \in \mathbb{R}^{n \times d}} \frac{1}{2} \|\Pi_\Omega(P - X)\|_F^2 + \lambda \|X\|_*.$$

Here $\|X\|_*$ is the *nuclear norm* of a matrix, it corresponds to the sum of its singular values (recall squared Frobenius norm is different since it is the sum of *squared* singular values), and it serves as a regularization term which biases the solution towards being low-rank.

A simple, and common way to approach this problems is iterative, and outlined in Algorithm 7.9.1. Start with some guess for X (e.g., average of average of rows and of columns of P_Ω). Take the svd of X to obtain $USV^T \leftarrow \mathsf{svd}(X)$. Shrink all singular values by λ or 0 (similar to Frequent Directions in Chapter 11.3, but not on the squared values). This operation ϕ_λ is defined for diagonal matrix S as

$$\phi_\lambda(S) = \mathsf{diag}((S_{11} - \lambda)_+, (S_{22} - \lambda)_+, \ldots, (S_{dd} - \lambda)_+),$$

where $(x - \lambda)_+ = \max\{0, x - \lambda\}$. Then we update $\hat{X} = U \phi_\lambda(S) V^T$; this provides a lower rank estimate since $\phi_\lambda(S)$ will set some of the singular values to 0. Finally, we refill the known values Ω as

$$X \leftarrow \Pi_\Omega(P) + \Pi_\Omega^\perp(\hat{X}),$$

and repeat until things do not change much on an update step (it has "converged").

This typically does not need too many iterations, but if n and d are large, then computing the SVD can be expensive. Various matrix sketching approaches (again see Chapter 11.3) can be used in its place to estimate a low-rank approximation more efficiently – especially those that pay attention to matrix sparsity. This is appropriate since the rank will be reduced in the ϕ_λ step regardless, and is the point of the modeling.