15 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:

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

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 have the same units.

The braindead 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 it is chosen so all values in each column lie in $[0, 1]$. In the other common approach, they are normalized so that the mean value in each column is 0, and 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.

We will approach 3 related settings and discuss more principled ways to compare high dimensional data with inconsistent of missing units.

15.1 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 $\mathsf{d} : X \times X \to \mathbb{R}$ 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 (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

$$\langle a_i, a_j \rangle = \frac{1}{2}\left(\|a_i\|^2 + \|a_j\|^2 - \|a_i - a_j\|^2\right). \tag{15.1}$$

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 (15.1). That is we can set $\|a_i - a_j\|^2 = D^2_{i,j}$. 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 $\|a_j\|^2 = \|a_1 - a_j\|^2 = D^2_{1,j}$. Using this assumption and equation (15.1), 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.

15.2 Linear Discriminant Analysis

Another tool that can be used to learn Euclidian distance for data is a *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 $X \subset \mathbb{R}^d$, these is 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 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.

15.3 Distance Metric Learning

The first approach MDS required that all distances were known ahead of time, and then a low-dimensional Euclidean embedding can be generated. The second approach LDA requires that the data X is somehow clustered or labeled into k classes before the analysis starts. In many settings these assumptions may be unrealistic.

However, if we are to choose a good metric, 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 are as small as possible, while the far pairs are 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 (following Ying+Li in JMLR12).

The standard approaches in the literature set up an optimization procedure and then run a "solver" to find the best M. We will instead describe an approach which is a bit less opaque.

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 because since X will be related to an outer product between two data points, this makes sense to think of as $\mathbf{d}_M(X)$.

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 15.3.1.

Algorithm 15.3.1 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 $M = M_T$.
```