MSA220/MVE440 Statistical Learning for Big Data

Lecture 11

Rebecka Jörnsten

Mathematical Sciences
University of Gothenburg and Chalmers University of Technology

Data representations

With big data we often need to find efficient data representations of a smaller dimension for both visualization and computation.

- Past lectures: SVD, PCA, SOM
- Today: MDS, NMF

SVD

SVD (singular value decomposition) is a workhorse that underpins much of the modeling we do!

- Data matrix X of dimension $n \times p$
- Before you do anything, you want to center and scale the columns of X!!!
- Otherwise the scale of individual variables dominate the representation and visualization is weird without centering
- We want to approximate the observations x_i in X by a lower-rank model
- Find the lower-rank model V_q to minimize the L2 error

$$\sum_{i=1}^{N} ||x_i - V_q \lambda_i||^2$$

where V_q is a $p \times q$ has *orthogonal* columns and λ_i is the variable specific coefficient

SVD

- If we knew V_q we can easily solve for λ in the problem below

$$\sum_{i=1}^{N} ||x_i - V_q \lambda_i||^2$$

- $\lambda_i = V_q^T x_i$
- Now given λ we want to find V_q:

$$\sum_{i=1}^{N} ||x_i - V_q V_q^T x_i||^2$$

- $V_q V_q^T = H_q$ is a projection matrix that maps x_i onto the space spanned by columns in V_q (this btw looks a lot like regression, yes?)
- The solution to the problem is the svd of $X = UDV^T$ where V_q is the first q columns of V

SVD

- $X = UDV^T$ where U is a $n \times p$ matrix, D is a diagonal $p \times p$ matrix and V is a $p \times p$ matrix where $U^T U = I$, $V^T V = I$
- We can also write $VX = UD$
- UD are called the *principal components*
- VX is the rotation V applied to the data X to project it onto the principal component space.
- The entries of each column in V are called *loadings* and tell you how much each original variable contribute to the make-up of the new dimension in PC space
- The leading components in V correspond to the largest values of D

SVD

- Another way of looking at SVD is building a structure from orthogonal components. To see this write

$$X = UDV^T = \sum_{j=1}^{P} d_j u_j v_j^T$$

  where u_j is a $n \times 1$ vector and v_j^T is a $1 \times p$ vector.
- Each produce $u_j v_j^T$ construct a $n \times p$ matrix representation of X
- Scaled by d_j they represent approximation of X in orthogonal directions.
- The first component is the best rank 1 approximation of X

SVD

- Best rank q approximation

$$X_q = \sum_{j=1}^{q} d_j u_j v_j^T$$

with approximation error

$$||X - X_q||^2 = || \sum_{j=q+1}^{p} d_j u_j v_j^T ||^2 = \sum_{j=q+1}^{p} d_j^2$$

- We approximate the matrix by a sum over layers!

Pros and Cons with (sparse) SVD

- SVD does explain matrices as best possible (in terms of L2 error)
- But difficult to interpret!
- Sparsity *can* help - but that depends on the actual "true" sparsity of the data
- The sum of layers is also difficult to interpret since the loadings can be both positive and negative

Pros and Cons with (sparse) SVD

- The 1st component is usually very close to an average
- E.g. average 16*16 digit image - looks like a bunch of blurry superimposed digits
- To approximate a particular digit you add and subtract blurry versions of each digit - correcting for "mistakes" in each layer
- The problem is that we insisted on an orthogonal transformation

Non-negative matrix factorization

- What if the idea of layer summation is our key feature?
- Each layer should add some information to the representation (not correct previous layers)
- NMF:

$$X = WH, \quad W \geq 0 \quad H \geq 0$$

- X is our $n \times p$ matrix where each row is an observation and each column a feature
- W is a $n \times r$ matrix, or *basis* which gives you the coordinates for each of the n observations in the lower-dimensional space....
- ... indexed by H: a $r \times p$ matrix of coefficients, or a *codebook*.

Non-negative matrix factorization

- NMF:

$$X = WH, \quad W \geq 0 \quad H \geq 0$$

- Example from the handwritten digits: Let's say we choose to approximate the data with rank K
- H will contain K images of the same size as each original digit, illuminating important pixels that summarize the data
- W is a matrix where each row j tells you how to combine the images in H to recreate the jth digit in the data set.

Data representations

- Both SVD and NMF try to find a linear dimension reduction of the data to summarize the data well
- The difference lies in the assumed structure of the dimension reduction
- SVD creates orthogonal components (perhaps sparse)
- NMF creates component-wise non-negative coefficients and basis elements.
- NMF - applied to non-negative data (but you can translate or run NMF separately on positive and negative data,....etc)

Non-negative matrix factorization

- Let's say we have found a rank K approximation
- We can approximate the jth observation by

$$\hat{X}_j = \sum_{l=1}^{K} W_{jl} H_{l.}$$

- Since we have a non-negative contraint on W and H this consists of adding layers together, no corrections or subtractions.
- The result of the non-negative constraint is that the coefficients in H tend to be *sparse*!!!

Non-negative matrix factorization

- How do we obtain the NMF representation?
- Several algorithms exist
- First, just think about how the problem is written (with L2-loss - there are other options here)

$$\min_{W,H} ||X - WH||^2 \quad W \geq 0, \quad H \geq 0$$

- This kind of looks like regression...

Non-negative matrix factorization

- $$\min_{W,H} ||X - WH||^2 \quad W \geq 0, \quad H \geq 0$$
- The above problem is NP hard
- We use starting values and iterative procedures
- Converges to stationary points

Non-negative matrix factorization

- $$\min_{W,H} ||X - WH||^2 \qquad W \geq 0, \quad H \geq 0$$
- The above problem is ill-posed
- Many solutions equally good.
- Careful about reading too much into explicit form of W and H
- Impose more structure on these: sparsity or other regularization
- Also tricky to choose the rank K - application dependent

Non-negative matrix factorization

- $$\min_{W,H} ||X - WH||^2 = \sum_{j=1}^{p} H_j'(W'W)H_j - 2H_j(W'X_j) + ||X_j||^2$$
- Turned into p separate non-negative LS problems
- These can be solved in several ways
- The problem is symmetric in W and H
- Two-block coordinate descent - update H given W and W given H

Non-negative matrix factorization

- Let $F(W, H) = ||X - WH||^2$
- First order optimality conditions

$$W \geq 0 : \nabla_W F = WHH^t - XH^t \geq 0, W * \nabla_W F = 0$$

$$H \geq 0 : \nabla_H F = W^t WH - W^t X \geq 0, H * \nabla_H F = 0$$

where $*$ is component-wise multiplication

- Multiplicative method (Lee and Seung, 1999)
- From the first order conditions we can obtain updates as

$$W * \frac{XH^t}{WHH^t}$$

$$H * \frac{W^tX}{W^tWH}$$

- It's just a gradient based update since

$$W * \frac{XH^t}{WHH^t} = W - \frac{W}{WHH^t} * \nabla_W F$$

$$H * \frac{W^tX}{W^tWH} = H - \frac{H}{W^tWH} * \nabla_H F$$

Non-negative matrix factorization

- Alternating least squares
- Just iterate LS keeping W fixed and then H fixed
- negative elements are projected onto the nonnegative solution (set to 0)
- Fast and simple
- Tricky if we want to use more complicated regularizations, then most people use multiplicative updates
- ANNLS - slower but works well in practice

Non-negative matrix factorization

- How to start off the computation?
- SVD solution
- Clustering solution for W
- Random starts

Self-organizing maps

- We can visualize large data set by looking at the leading principal components
- SOM - self-organizing maps is a very different way of looking at data
- We construct an artificial lower dimensional space where to explore the data

Self-organizing maps

- We construct a rectangular grid of *prototypes* m_j
- The prototypes live in the higher p-dim space but are parameterized by grid-points in a (usually) 2-dim space
- We can initialize with m_j in the two-dimensional space from the leading PC - i.e. draw up a rectangular grid in the PC1-PC2 plot and let m_j be the grid-coordinate points.
- We're now going to update the prototypes to better summarize the data, which corresponds to bending the PC plane to be able to map it to a rectangular grid.
- For each observation x_i we find the closest (euclidean distance) prototype m_j
- For all neighbors (on the grid) m_k of m_j we move them toward x_i (in p-space):

$$m_k = m_k + \alpha(x_i - m_k)$$

- α is the learning rate

Multi-dimensional scaling

- What's good and bad about SOMS?
- PRO: simple to use and interpret and customize (distance metrics to use)
- CON: need to revisit data points in update so problem with big n. May not be sufficient to visualize data in 2 dimensions when p is large
- MDS: only use the pairwise distances so cheaper updates
- MDS: not restricted to 2-dim space

Multi-dimensional scaling

- We compute all the pairwise distances between objects i and j: d_{ij}
- We can be clever about using appropriate distances here depending on the variable types (`daisy` package in R)
- We want to find observations z_i in a low-dimensional space such that

$$\sum_{i \neq i'} (d_{ii'} - ||z_i - z_{i'}||)^2$$

  is small.
- We can scale the mapping distance by $d_{ii'}$ which preserved small distances better
- We can also use rank-based mapping (called non-metric scaling) - depending if subsets of data are very spread out.

Multi-dimensional scaling

- We compute all the pairwise distances between objects i and j: d_{ij}
- We want to find observations z_i in a low-dimensional space such that

$$\sum_{i \neq i'} (d_{ii'} - ||z_i - z_{i'}||)^2$$

  is small.
- How? Spectral decomposition of centered $d_{:ii'}$ and use leading eigenvectors (we'll see more about this when we do spectral clustering - for now think about the fact that the leading vectors summarize the dominant directions in a matrix - i.e. the structure of the pairwise distances).
- Alternatively - solve the above problem explicitly via iterative gradient descent.

- tSNE is an extension of MDS. (Paper can be found here).
- Here we use a kernel based distance between observations i and j and interpret this as a probability

$$p_{j|i} = Gaussian - pdf(d_{ij}, \sigma_i^2) / \sum_{k \neq i} Gaussian - pdf(d_{ik}, \sigma_i^2)$$

  where σ_i is the bandwidth of the kernel around reference point i. You create a symmetric distance by taking the average of the two conditional distributions
- Even more simple if you use the same bandwidth everywhere

$$p_{ij} = Gaussian - pdf(d_{ij}, \sigma^2) / \sum_{k,l \neq k} Gaussian - pdf(d_{kl}, \sigma^2)$$

- We now try to construct a d-dimensional space y that mimics these densities where we define the pdf in this space as

tSNE

- How do we measure distance in the y-space? Natural thing would be to use gaussian densities there too (called SNE)
- In the SNE, the authors observed that the y-space got "crowded" in that slightly similar observations were forced to be very similar in the low-dimensional space
- To remedy this, tSNE uses a more long-tailed distribution to describe the densities in y-space (Cauchy distribution)

$$q_{ij} = \frac{(1 + d(y_i, y_j))^{-1}}{\sum_{k \neq i}(1 + d(y_i, y_k))^{-1}}$$

  where d is the squared euclidean distance
- We match p and q by minimizing the Kullback-Leibler distance $\sum_{i \neq j} p_{ij} \log(\frac{p_{ij}}{q_{ij}})$
- How? Gradient descent.
- So it's related to MDS, but with a different treatment of distances and a different cost function.

Isomap

- Isomap is similar to MDS - we work with a matrix of distances between observations
- Use distances based on a shortest path in a graph connecting observations
- The graph is produced by connecting only objects that are within a certain euclidean distance of eachother, or is within a set of k nearest neighbors.
- This can capture quite local behaviour - nonlinear transformation of data
- We will revisit this idea when we look at Spectral Clustering next week.