CSCI 4022 Fall 2021
UV Wrapup, Outliers Intro

Announcements and Reminders

1. HW8 this Wednesday
2. Project check-in after break. All you need is pre-processing and preliminary results! I'll give feedback on the proposals during break, but please make Piazza posts if you want any advice on things you can do with your data sets.
3. **Last day** of theory lecture is Wednesday, with associated notebooks on Friday. We have an exam review, some practice/application lectures, and some project presentations for the 5 lecture days after break. Home stretch!!

The Missing Data Decomposition

UV Decomposition: *Find* entries of U and V so that $M' = UV$ is close to M.

$$\begin{bmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{bmatrix} \heartsuit = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \\ u_{51} & u_{52} \end{bmatrix} \times \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \end{bmatrix}$$

Plan: We'll develop an iterative solution to optimize the entries of U and V

- Need some starting values
- Could pick all 1s, random numbers, all equal to $(1/d) \cdot mean$ of M's non-blank entries, etc.

Whatever we do, there are *lots* of degrees of freedom in our selection of the u_{ij} and v_{ij}

- Many local minima in the RMSE surface
- Want to use a handful of different starting values for U and V and compare

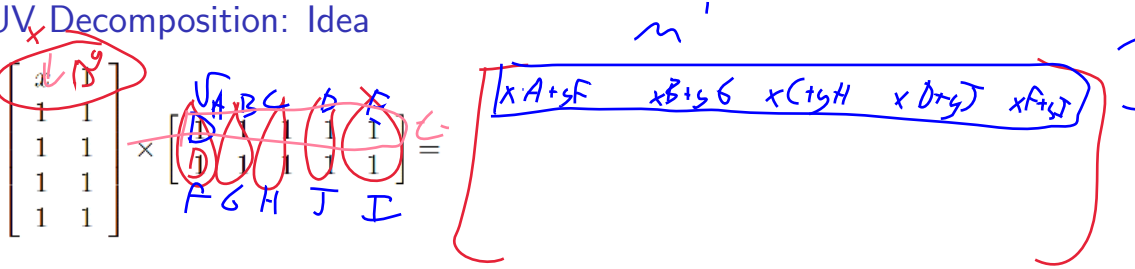
UV Decomposition: Idea

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Suppose we start with some non-random initialization of U, V all-1s.

- Then our first estimate of M would be M' consisting of all 2s, which... isn't very good.
- We want to **optimize** over the values in U and V

UV Decomposition: Idea



Suppose we start with some non-random initialization of U, V all-1s.

- Then our first estimate of M would be M' consisting of all 2s, which... isn't very good.
- We want to **optimize** over the values in U and V
- Let's *improve* on U by making the first entry - u_{11} - better. Call this value x

UV Decomposition: Idea

$$\begin{bmatrix} x & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x+1 & x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Suppose we start with some non-random initialization of U, V all-1s.

- Then our first estimate of M would be M' consisting of all 2s, which... isn't very good.
- We want to **optimize** over the values in U and V
- Let's *improve* on U by making the first entry - u_{11} - better. Call this value x
- What's the best x? The one that provides the best M'!
- **Choose** x that minimizes RMSE. But **note:** x *only affects* the top row of M'.

UV Decomposition: Optimization

$$\begin{bmatrix} x & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x+1 & x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Choose x that minimizes RMSE. But **note:** x *only affects* the top row of M'
...Reminder that $M_{1,:}$ =[5 2 4 4 3]

... and the best choice is a classic calculus problem: find the deriv, set equal to zero!

UV Decomposition: Optimization

$$\begin{bmatrix} x & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x+1 & x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Choose x that minimizes RMSE. But **note:** x *only affects* the top row of M'

...Reminder that $M_{1,:} =$[5 2 4 4 3]

$$\begin{aligned} SS &= (5-(x+1))^2 + (2-(x+1))^2 + (4-(x+1))^2 + (4-(x+1))^2 + (3-(x+1))^2 \\ &= (4-x)^2 + (1-x)^2 + (3-x)^2 + (3-x)^2 + (2-x)^2 \end{aligned}$$

... and the best choice is a classic calculus problem: find the deriv, set equal to zero!

UV Decomposition: Optimization

$$\begin{bmatrix} x & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x+1 & x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Choose x that minimizes RMSE. But **note:** x *only affects* the top row of M'

...Reminder that $M_{1,:}$ =[5 2 4 4 3]

$$\begin{aligned} SS &= (5-(x+1))^2 + (2-(x+1))^2 + (4-(x+1))^2 + (4-(x+1))^2 + (3-(x+1))^2 \\ &= (4-x)^2 + (1-x)^2 + (3-x)^2 + (3-x)^2 + (2-x)^2 \end{aligned}$$

... and the best choice is a classic calculus problem: find the deriv, set equal to zero!

$$\frac{d}{dx}SS = -2[(4-x) + (1-x) + (3-x) + (3-x) + (2-x)]$$

$$0 \stackrel{set}{=} -2[(13-5x)] \implies x = 13/5 = 2.6.$$

UV Decomposition: Optimization

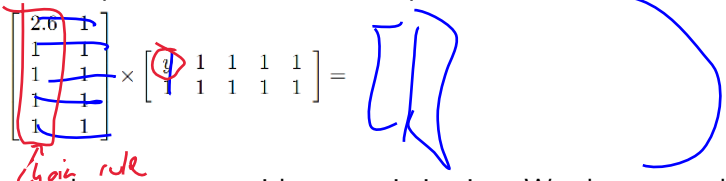
So we update U, and then also update M'.

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 3.6 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

.. and we move on with our optimization. We choose another entry of either u or v. Suppose we move onto the first entry in V, v_{11}. Call it y.

UV Decomposition: Optimization

So we update U, and then also update M'.



.. and we move on with our optimization. We choose another entry of either u or v. Suppose we move onto the first entry in V, v_{11}. Call it y.

- What's the best y? The one that provides the best M'!
- **Choose** y that minimizes RMSE. But **note:** y *only affects* the first column of M'.

UV Decomposition: Optimization

So we update U, and then also update M'.

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} y & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2.6y+1 & 3.6 & 3.6 & 3.6 & 3.6 \\ y+1 & 2 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 & 2 \end{bmatrix}$$

.. and we move on with our optimization. We choose another entry of either u or v. Suppose we move onto the first entry in V, v_{11}. Call it y.

- What's the best y? The one that provides the best M'!
- **Choose** y that minimizes RMSE. But **note:** y *only affects* the first column of M'.
- ...Reminder that $M_{:,1}$ =[5 3 2 2 4]^T, so we solve:

$$\begin{aligned} SS &= (5-(2.6y+1))^2 + (3-(y+1))^2 + (2-(y+1))^2 + (2-(y+1))^2 + (4-(y+1 \\ &= (4-2.6y)^2 + (2-y)^2 + (1-y)^2 + (1-y)^2 + (3-y)^2 \end{aligned}$$

$$\frac{d}{dy}SS = -2[(2.6)(4-2.6y) + (2-y) + (1-y) + (1-y) + (3-y)] \implies y = 17.4/10.76$$

UV Decomposition: Optimization

...and we repeat

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1.617 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 5.204 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \end{bmatrix}$$

For terms like u_{31} we have to deal with blanks in M. What do we do?

UV Decomposition: Optimization

...and we repeat

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ z & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1.617 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 5.204 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2.617 & 2 & 2 & 2 & 2 \\ 1.617z+1 & z+1 & z+1 & z+1 & z+1 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \end{bmatrix}$$

For terms like u_{31} we have to deal with blanks in M. What do we do?

- **Choose** z that minimizes RMSE. Now we're in *row 3* of M'.

UV Decomposition: Optimization

...and we repeat

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ z & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1.617 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 5.204 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2.617 & 2 & 2 & 2 & 2 \\ 1.617z+1 & z+1 & z+1 & z+1 & z+1 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \end{bmatrix}$$

For terms like u_{31} we have to deal with blanks in M. What do we do?

- **Choose** z that minimizes RMSE. Now we're in *row 3* of M'.
- ...Reminder that $M_{3,:} = [2 \_ 3\ 1\ 4]^T$, so we simply omit that term.

$$SS = (2-(1.617z+1))^2 + (3-(z+1))^2 + (1-(z+1))^2 + (4-(z+1))^2$$

$$\frac{d}{dz}SS = -2[1.617(2-(1.617z+1)) + (3-(z+1)) + (1-(z+1)) + (4-(z+1))]$$

$$\implies z = 6.617/5.615 = 1.178$$

UV Decomposition: Optimization

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1.178 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1.617 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 5.204 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.905 & 2.178 & 2.178 & 2.178 & 2.178 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \end{bmatrix}$$

We continue until we have updated all the elements of both U and V. Some thoughts:

1. How do we pick which order to update the elements of U and V?

   We could pick some arbitrary order and stick with it each round of updating

   ...But that might introduce some kind of bias (if the columns/rows are ordered)

   If you can, *permuting* the order each time can avoid this (but if the matrices are large, then generating a permutation of their elements can be computationally taxing)

UV Decomposition: General Case

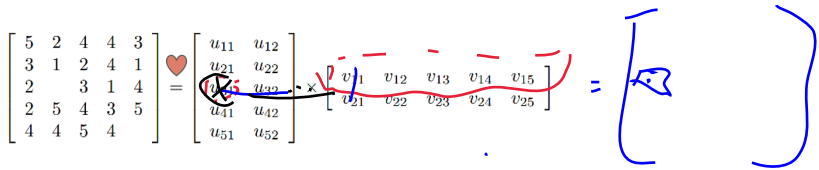
Let $M := m \times n$

$M := m \times n$

$U := m \times d$

$V := d \times n$

$P := UV = M'$



In general, what is the update for an arbitrary element of U or V? Consider u_{rs}.

1. u_{rs} only affects row r of P.
2. The elements of that row of P are sum of the dot products of row r of u with *every* column of v. In other words:

$$p_{rj} = \sum_{k=1}^{d} u_{rk} v_{kj} = \sum_{k \neq s} u_{rk} v_{kj} + x v_{sj}$$

UV Decomposition: General Case

Consider updating u_{rs}. Then $p_{rj} = \sum_{k=1}^{d} u_{rk}v_{kj} = \sum_{k \neq s} u_{rk}v_{kj} + xv_{sj}$ and the resulting sum of squares error per term is from $M - P$:

$$(m_{rj} - p_{rj})^2 = \left(m_{rj} - \sum_{k \neq s} u_{rk}v_{kj} + xv_{sj} \right)^2$$

for a total error in row r of

$$\sum_j \left(m_{rj} - \sum_{k \neq s} u_{rk}v_{kj} + xv_{sj} \right)^2$$

We take a derivative with respect to x, set it equal to zero:

$$0 = \sum_j -2v_{sj} \left(m_{rj} - \sum_{k \neq s} u_{rk}v_{kj} + xv_{sj} \right)$$

UV Decomposition: General Case

Solving for $x = u_{rs}$ looks kind of like an average, but a bit gross:

$$x = \frac{\sum_j v_{sj} \left(m_{rj} - \sum_{k \neq s} u_{rk} v_{kj} \right)}{\sum_j v_{sj}^2}$$

and a nearly identical calculation that replaces rows for columns shows that to update v_{rs}:

$$y = \frac{\sum_i u_{ir} \left(m_{is} - \sum_{k \neq r} u_{ik} v_{ks} \right)}{\sum_i u_{ir}^2}$$

Sanity check: are these derivative-equals-zero values actually local *minima*? How would we prove this?

UV Decomposition: Implementation Notes

Recall from Recommender Systems that frequently we want to normalize the utility matrix M to account for differences in the ways users rate items, or differences in item ratings.

Some options: (in context of rec. systems but maps to other applications too)

1. Subtract from each non-blank element m_{ij} the average rating of user i
2. Subtract from each non-blank element in column j the average rating of item j
3. Do both of these, in either order
4. From element m_{ij} subtract $\frac{1}{2}\cdot$ (the average of user i + the average of item j)
5. Or similar.

Crucially, whatever we do, if we make predictions, must add back in whatever was subtracted out after we've computed U and V.

UV Decomposition: Implementation Notes

One danger we may run into in optimization/estimation problems (UV, and others) is *overfitting* our solution to the training data, and not performing well on out-of-sample or new data.

Some options to combat this:

1. Only move an element a fraction of the way towards the RMSE-minimizing value (of course, using a convergence/tolerance check may make us just move all the way there, eventually!)
2. Take several different UV decompositions (different initial conditions), and average all of their predictions for a particular entry in M

   If we do this, we can weight by each solutions eventual RMSE
3. Can use a loose convergence criterion to stop revisiting elements in U and V

   Idea: Maintain a list of "good" estimates, only work on others.

UV Decomposition: Big Picture

Given an $n \times m$ utility or data matrix M, with some unknown/blank entries...

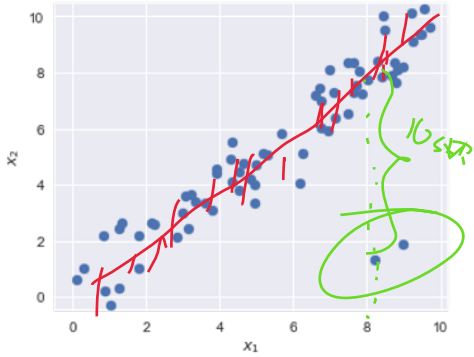
1. Preprocess M: e.g. normalize
2. Initialize: decompose with *many* different initial U and V because local minima lurk
3. Iterate: do multiple training epochs, looping over all elements in U and V each time
4. Converge: don't expect RMSE to reach 0, but once its epoch-to-epoch change is $<$ tolerance we can break. We can also stop once no component improves RMSE by more than some threshold (e.g. a max noem)
5. Would another choice of d work better? Try it, and recall **elbow** plots of d as the x-axis and $RMSE$ as the y-axis. More d is always (expected to be) less $RMSE$, but with diminishing returns.

Outliers: Z-scores and Random Forests

Definition: An outlier is an observation which deviates so much from other observations as to arouse suspicion it was generated by a different mechanism. (from Hawkin, 1980)

Example: Consider the linear model $Y = \alpha + \beta x + \varepsilon$, where $\varepsilon \sim N(0, \sigma^2)$.

Does it look like the data set here was generated by this process?



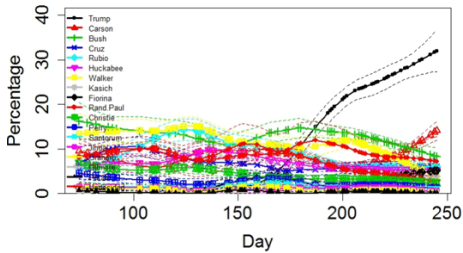
Example: Consider a utility matrix, representing 100 users' ratings of 100,000 movies. S'pose a user has rated every single movie as a 5 (out of 5 stars).

Sources of Outliers

Outliers come from a variety of different sources:

1. Measurement or sampling errors
2. Data entry errors
3. Processing errors (incl. code bugs)
4. Novel process/changepoint
5. Possibly intentionally introduced to stress-test code pipelines

2016 Republican President Candidate Popularity



Danger of Outliers

Detecting outliers (and removing them) is important!

We want to characterize/summarize our specific data set.

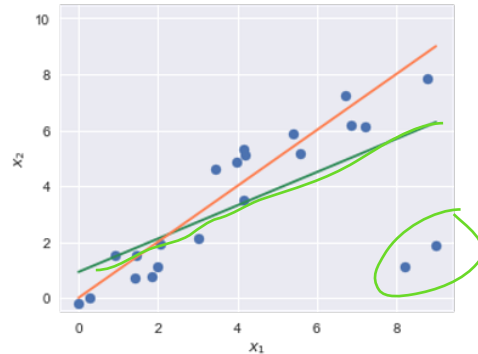
Assume a particular process generated our data

Outliers come from a different process

Building a predictive model that includes outliers means it wasn't trained on data that came from the process we want to make predictions about

...So our predictions won't be of the thing we want!

2016 Republican President Candidate Popularity



Considerations of Outliers

When considering methods to detect outliers, think about:

- How many features are there?

  Univariate vs multivariate

  High-dimensional?

- Can you assume the data follows some kind of distribution?

  Does it *look* like it follows a distribution?

  Result: Parametric vs non-parametric methods

Overview of Outlier Methods: Many Options

1. Z-score/extreme value analysis (parametric) Also: Mahalanobis distance extension
2. Probabilistic/statistical modeling (parametric) (right-tailed distributions)
3. Linear regression models, PCA (parametric)
4. Proximity/density-based models (non-parametric)
5. Information theory models
6. Higher-dimensional outlier detection methods

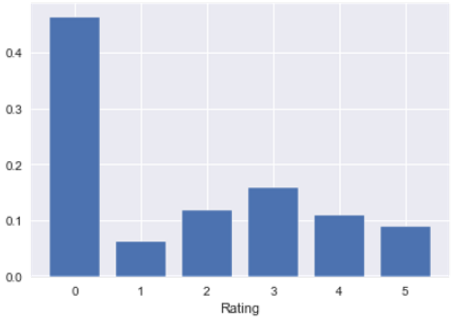
Today: Z-score + Mahalanobis distance (parametric, simple, tough with higher dim'l data)
Next time: Isolation forests (iForests) (non-parametric, more sophisticated, works well with high dim'l data, parallelizes readily)

Z-scores

$$z = \frac{\text{Data} - \text{mean}(\text{data})}{\text{sd}(\text{data})}$$

Very few values outside of $-2.5 < z < 2.5$

- Relies on assumption of an underlying normal distribution of your data
- If the data aren't (approx.) normal, a transformation to normal might be an important preliminary step.



Some simple transformations:

1. Removing extreme values.
2. Logarithm, Exponentiate.

Example: Consider a utility matrix of users' movie ratings, where 0 denotes a user has not yet seen/rated a particular movie.

Normality

Goal: Make non-normal data look more normal. tools: some simple transformations:

1. Removing extreme values.
2. Logarithm, Exponentiate.
3. **Box-Cox**: For positive data Y, transform the data via

$$Y' = \begin{cases} \frac{Y^\lambda - 1}{\lambda} & \lambda \neq 0 \\ \ln Y & \lambda = 0 \end{cases}$$

where we *choose* the value of $\lambda \in [-5, 5]$ to maximize how close to normal the resulting Y' looks.

"Functional Distance"

Note: Ostensibly we need a measure for "most normal." While these are probability distributions and/or data sets, we can instead do a distance-between-functions on their *cdfs*: norms here tend to be of the form $d(Y, Z) = \int_\Omega f_Y(x) - f_Z(x)\, dx$; note that squaring or other powers inside the integral are also common, as in Pythagorean/Euclidean distance norms.

For comparing cdfs, we know the anti-derivatives of f and g (the two cdfs F and G!), so we often use an L^1 type difference. The statistic

$$d(F, G) = \max_x |F(x) - G(x)|$$

is actually the basis for the most common "are these distributions equal" hypothesis test, called the *Kolmogorov–Smirnov* test.

There are other goodness-of-fit tests, including our good ol' standby maximum likelihood, where we choose $\lambda \in [-5, 5]$ to make $P(Y'|Y' \sim N(\bar{Y}', s^2_{Y'}))$ as large as possible! This one is the usual with Box-Cox.

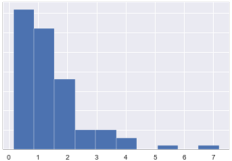
Example

Example: Sketch the CDFs of $U[-2, 2]$ and $N(0, 1)$. How different are they?

Box Cox in Action

Goal: Make non-normal data look more normal.

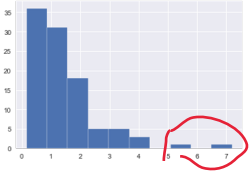
Example: Consider the data at top-left. What do different values of λ do?



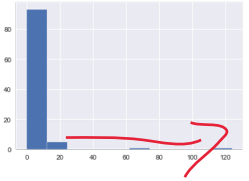
Box Cox in Action

Goal: Make non-normal data look more normal.

Example: Consider the data at top-left. What do different values of λ do?



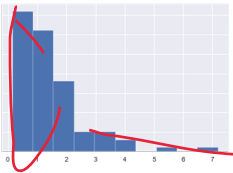
First try: $\lambda = 3$



Box Cox in Action

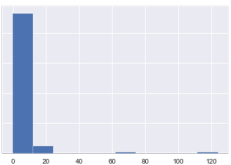
Goal: Make non-normal data look more normal.

Example: Consider the data at top-left. What do different values of λ do?



Second try: $\lambda = -3$

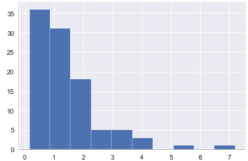
First try: $\lambda = 3$



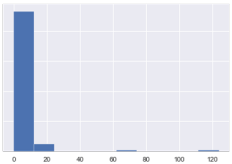
Box Cox in Action

Goal: Make non-normal data look more normal.

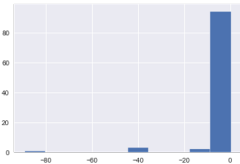
Example: Consider the data at top-left. What do different values of λ do?



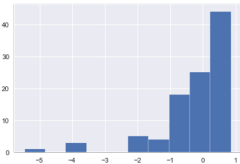
First try: $\lambda = 3$



Second try: $\lambda = -3$



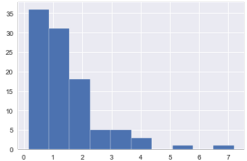
Third try: $\lambda = -1$



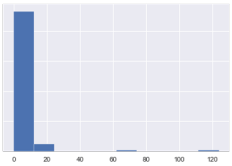
Box Cox in Action

Goal: Make non-normal data look more normal.

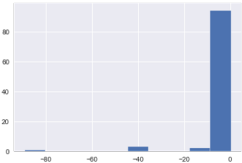
Example: Consider the data at top-left. What do different values of λ do?



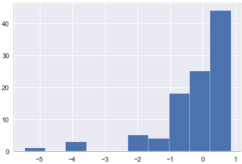
First try: $\lambda = 3$



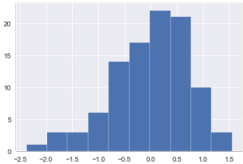
Second try: $\lambda = -3$



Third try: $\lambda = -1$



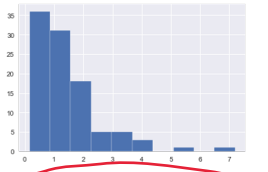
Fourth try: $\lambda = -0.25$



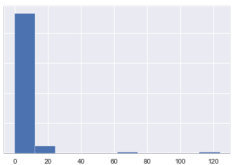
Box Cox in Action

Goal: Make non-normal data look more normal.

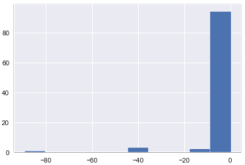
Example: Consider the data at top-left. What do different values of λ do?



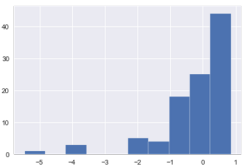
First try: $\lambda = 3$



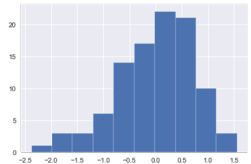
Second try: $\lambda = -3$



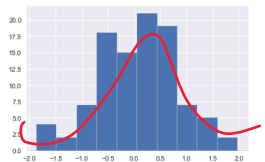
Third try: $\lambda = -1$



Fourth try: $\lambda = -0.25$



Last try: $\lambda = 0; Y' = \ln Y$

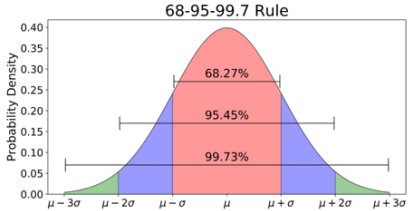


Z-Scores

Each data point can then be converted into a **Z-score** via the usual standardization transform:

1. X = original data (or transformed-to-normal data)
2. $\bar{X}$= mean of data X
3. s = sample standard deviation of X
4. Z = # standard deviations that a data point is away from the mean

$$Z = \frac{X - \bar{X}}{s}$$



Classification: Label data points as outliers if their (absolute) Z-score is greater than some threshold. Typical choices: Z = 2.5, 3 or 3.5, motivated by the "68-95-99.7 rule"

Mahalanobis distance

Con of Z-Scores:
Z-score becomes difficult in higher dimensions: we can compute how far a given datum $\vec{x}$ is from the average datum $\bar{X}$, but now each of these are d-dimensional.

We definitely *don't* want to just take a usual distance $x - \bar{x}$ and hit it with a norm (like L_2 distance). Why not? The data in x might be *correlated:* some dimensions might be basically redundant, so being an outlier in one would probabilistically make you also an outlier in other dimensions.

So what do we do?

Mahalanobis distance is a measure of how far points are from *the general structure of the data*.

Recall: (from GMMs) that the higher-dimensional normal distribution functions like an *ellipse* of points (or more like a rugby-ball shaped point cloud)

Mahalanobis distance

The resulting distance is a multivariate form of Z-scores. **Definition**: The *Mahalanobis* distances of a datum (row vector) $\vec{x}$ is given by:

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \bar{x})^T C^{-1} (\vec{x} - \bar{x})}$$

where:

1. $\bar{x}$ is the d-dimensional mean of the data. Component i is the mean of all the data along dimension i.
2. C is the sample covariance matrix, given by $C = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x}) (x_j - \bar{x})^T$ Recall that the covariance between columns i and j is a measure of whether they shrink and grow in unison: does knowledge of one help (linearly) predict the other.

Mahalanobis distance

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \bar{x})^T C^{-1} (\vec{x} - \bar{x})}$$

Note: This is *very closely related* to PCA and SVD! The pieces of the covariance matrix $(x_i - \bar{x})(x_j - \bar{x})^T$ should look a lot like the $M^T M$ matrix we compute eigen-pairs from. In fact, when we standardize the data first, the matrix we're finding eigenpairs of *is the covariance*. This is why those methods are often explained in the **units** of variance: amount of variance in the data explained.

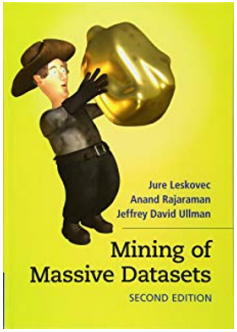
For the multivariate normal, the pdf of f depends solely on the Mahalanobis distance of the datum. Visually, points of equivalent Mahalanobis distance along a data set are on shared elliptical contours, where the ellipse is centered at $\bar{x}$ and oriented along its covariance.

Mahalanobis distance visualized

Acknowledgments

Next time: outliers wrapup

Some material is adapted/adopted from Mining of Massive Data Sets, by Jure Leskovec, Anand Rajaraman, Jeff Ullman (Stanford University) `http://www.mmds.org`



Special thanks to Tony Wong for sharing his original adaptation and adoption of slide material.