16B Prof Anant Sahai

Extra-credit opportunities announced.

Today: Finish Movie Model
"Checking your work" – validating with other data
Dimensionality Reduction for classification ← happening in lab!
Justifying this approach
Nonlinear Models & Local Approximation
Control around equilibria

Illustrative conceptual example: Movie Ratings,

People

$$R = \begin{array}{c|ccc|} & p_1 & \cdots & p_n \\ \hline m_1 & & & \\ m_2 & & r_{ij}\text{: rating of person } j \text{ for movie } i & \\ \vdots & & & \\ m_m & & & \\ \hline \end{array}$$

movies

$r_{ij} \in \mathbb{R}$
real number score.

16-philosophy: Make a model,
Learn the model from data,
Use the model to make predictions.

Can Modify our model to reflect what is possible: Each movie has a vector $\vec{g}$ of qualities associated

Each person has a vector $\vec{s}$ of sensitivities.

SVD Gave Us: (Outer-Product Form)

$$R = \sum_{k=1}^{r} \sigma_k \vec{u}_k \vec{v}_k^T$$

$$r(\vec{g}, \vec{s}) = \sum_{k=1}^{4} \sigma_k g_k s_k$$

numbers

Pull this out instead of trying to allocate it.

Truncate to 4 terms: $\hat{R} = \sum_{k=1}^{4} \sigma_k \vec{u}_k \vec{v}_k^T$

Now our learned model has $\vec{g}_i = \begin{bmatrix} (\vec{u}_1)_i \\ (\vec{u}_2)_i \\ (\vec{u}_3)_i \\ (\vec{u}_4)_i \end{bmatrix}$

Recall $\vec{u}$'s are m long

$\vec{v}$'s are n long.

$\vec{s}_j = \begin{bmatrix} (\vec{v}_1)_j \\ (\vec{v}_2)_j \\ (\vec{v}_3)_j \\ (\vec{v}_4)_j \end{bmatrix}$

$(\vec{v}_4)_j\ ]$

How to use this for prediction?

e.g. Given a new person, how can we extract their sensitivities $\vec{s}$?

Assume we're given $\vec{r}$: how does this person rate the m movies?

We need to go from $\vec{r}$ to $\hat{\vec{s}}$

ratings on known movies

estimated sensitivities for this person.

Can do this by least squares?

Aside let's look at a person we already know.

j^{th} person: $$\vec{r}_j = \sum_{k=1}^{4} \sigma_k \vec{u}_k (\vec{v}_k^T)_j$$

This is a scalar: j^{th} position in vector $\vec{v}_k$

How can I get $\vec{s}_j$ from $\vec{r}_j$?

$$(\vec{s}_j)_k = \frac{1}{\sigma_k} \vec{u}_k^T \vec{r}$$

: I can get the k^{th} sensitivity by projecting $\vec{r}$ onto the k^{th} principal component ($\vec{u}_k$) and scaling by $\frac{1}{\sigma_k}$ to fit our model

Similarly: We can get $\vec{g}_i$ from $\vec{q}_i^T = [r_{i1}\ r_{i2}\ \cdots\ r_{in}]$

By $$(\vec{g}_i)_k = \frac{1}{\sigma_k} \vec{v}_k^T \vec{q}_i$$

$\Rightarrow$ So for a new person with ratings $\vec{r}$,

Can get $$\hat{\vec{s}} = \begin{bmatrix} \frac{1}{\sigma_1} \vec{u}_1^T \vec{r} \\ \frac{1}{\sigma_2} \vec{u}_2^T \vec{r} \\ \vdots \\ \frac{1}{\sigma_4} \vec{u}_4^T \vec{r} \end{bmatrix}$$

Note:
Dividing by σ_i is a warning flag.
$\Rightarrow$ Beware of small σ_i.

And similarly for a new movie. $$\hat{g} = \begin{bmatrix} \frac{1}{\sigma_1} \vec{v}_1^T \vec{q} \\ \vdots \\ \frac{1}{\sigma_4} \vec{v}_4^T \vec{q} \end{bmatrix}$$

Can mess things up.

So. Given a new person (who comes with ratings $\vec{r}$ on the m known movies) and a new movie (which comes with ratings $\vec{q}$ by the n known people)

we can predict the rating $r = \sum_{k=1}^{4} \sigma_k \hat{g}_k \hat{s}_k$

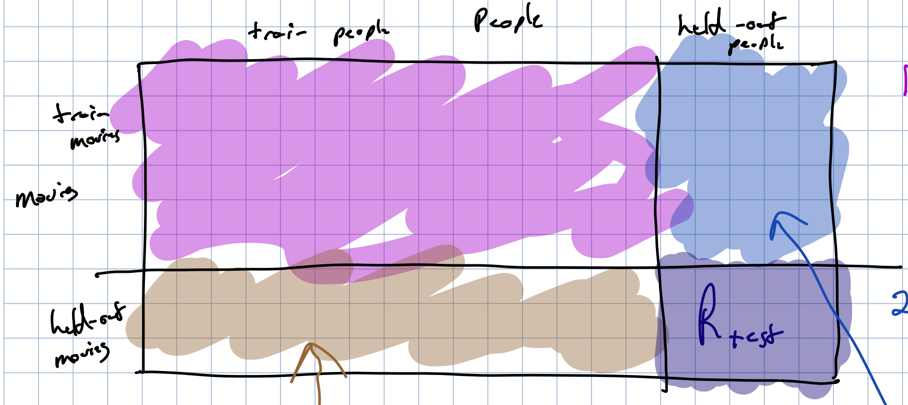
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\uparrow\ \ \uparrow$

Fit as above.

<u>Question</u>: How do we "check our work"

e.g. How do we justify that 4 is the right number of components to use? ← 4 qualities are important.

From Model-order-selection problem on the HW, we know we need to use some extra data.



1. Fit model on training Data: ie take SVD of that purple block, etc...

2. Test it out on held-out data.

a. Use data to estimate sensitivities for held-out people

b. Use data to estimate goodnesses for held-out movies.

c. Use R_{test} to check quality of predictions. Compare Predictions to actual ratings. (Use Mean-Squared Error)

This procedure, sweeping through candidate #s of components can justify model order choice.

Recall: This is the "silver standard" (like a "unit test" for learning from data)

The Gold Standard is always an "integration test" or "functional test."
See if it works in practice.

Step back to classification.

Wanted a way to reduce the dimensionality of data.

$\Longrightarrow$ Want to keep most important parts.

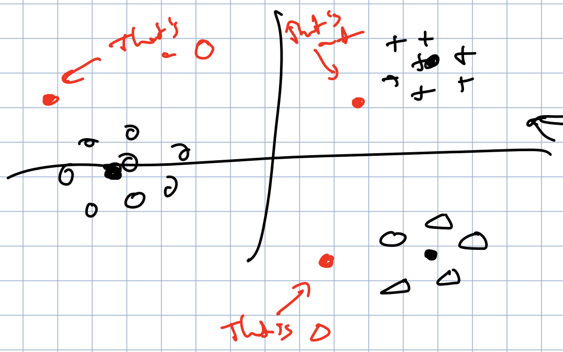
Can use the PCA approach: Project onto $\vec{u}_i$ if data is columns or $\vec{v}_i$ if data is rows.

Heuristic ← Based on hope that accurately capturing the pattern in the data helps with classification.

By reducing the dimensionality, we get an easier problem.

In particular, in 2d or 3d can visualize.

e.g. Three Categories $+$, Δ, o



Given.

How would we categorize a new point?

First & Most Basic Approach: Which group is the new point closest to?

An approach: Compute the centers of each group, and compare distances to centers.

LAB will do this approach.

How can we justify our approach to learning the pattern by SVD.

We are viewing $\hat{R} = \sum_{i=1}^{\ell} \sigma_i \vec{u}_i \vec{v}_i^T$ as an approximation to R.

In what sense is this a good approximation?!

→ $\hat{R}$ centroids has its columns all in an ℓ-dim subspace

So $\hat{R}$ is "simple" ← low-dimensional.

Is $\hat{R}$ "close" to R?

We will our favorite squared-error.

Want $\|R - \hat{R}\|_F^2$ to be as small as possible.

↑ Frobenius Norm

Treats R and $\hat{R}$ like big vectors.

Underlying model: $r_{ij} = \left[\sum_{k=1}^{\ell} \sigma_k (\vec{g}_k)_i (\vec{s}_k)_j\right] + w_{ij}$

↑ Hopefully small.

Key Question: Why does the SVD help?!

$$\|R - \hat{R}\|_F = \|\overbrace{U \Sigma V^T}^{\text{Full SVD}} - \hat{R}\|_F$$

Premultiply by U^T $= \|\Sigma V^T - U^T \hat{R}\|_F$

transposing $= \|V \Sigma^T - \hat{R}^T U\|_F$

Premultiply by V^T $= \|\Sigma^T - V^T \hat{R}^T U\|_F$

transposing $= \|\Sigma - \underbrace{U^T \hat{R} V}_{\text{still a rank-}\ell\text{ matrix. or lower}}\|_F$

ℓ

since $\hat{R} = \sum_{k=1} \sigma_k (\vec{g}_k) \vec{s}_k^T$

$$U^T \hat{R} V = \sum_{k=1}^{\ell} \sigma_k \underbrace{(U^T \vec{g}_k)}_{\text{just a vector.}} (\vec{s}_k^T V)$$

There are ℓ of them.

$\Rightarrow U^T \hat{R} V$ has rank at most ℓ.

So $U^T \hat{R} V$ should be the best rank-ℓ approximation to

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & & 0 & & & \\ 0 & \sigma_2 & & \vdots & & 0 & \\ 0 & 0 & \ddots & 0 & & & \\ & 0 & & \sigma_\ell & & & \\ & & & 0 & \sigma_{\ell+1} & & \\ & 0 & & \vdots & & \ddots & \\ & & & 0 & & & \sigma_r \end{bmatrix}$$

Want to keep these

Since $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_\ell \geq \sigma_{\ell+1} \cdots$

Intuitively clear! should just keep the ℓ biggest #s.

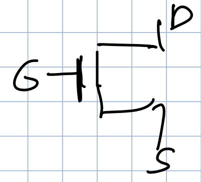
Want $U^T \vec{g}_k$ to be the k-th column of the identity. So $\vec{g}_k = \vec{u}_k$. Similarly we want $\vec{s}_k^T V$ to be the k^{th} row of the identity. So $\vec{s}_k = \vec{v}_k$.

The full theorem is called the Eckart-Young-Mirsky Thm.

$\Rightarrow$ The best rank-ℓ approximation to a matrix is given by keeping the first ℓ terms in the SVD.

Step Way Back: All of our 16AB models have been linear except Switches / comparators $\leftarrow$ nonlinear.

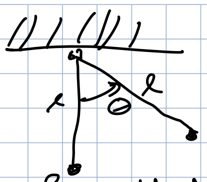
Real World is largely nonlinear.



$\leftarrow$ Actually smoothly nonlinear.

Almost all mechanical systems are nonlinear.

Consider a pendulum:



e.g. Get sines & cosines in equations governing motion

Need a tool & Philosophy for attacking nonlinear problems.

Tool: Local Approximation: Just Pretend Things are linear.

Taylor Series: $f(x) = f(x_0) + f'(x_0)(x-x_0)$

$+ \frac{f''(x_0)\ (x-x_0)^2}{2}$

$+ \dots$

When x is close to x_0, the higher-order terms contribute very little.

$\implies$ lump all dropped terms into the disturbance

Consider: $f(x,u) = 32 + x^3 u^2$

expand around (x_0, u_0)

$$f(x_0+\delta x,\ u_0+\delta u) = 32 + (x_0+\delta x)^3 (u_0+\delta u)^2$$

$$= 32 + (x_0^3 + 3x_0^2\delta x + 3x_0(\delta x)^2 + \delta x)^3$$
$$\cdot (u_0^2 + 2u_0\delta u + (\delta u)^2)$$

If δx and δu are both tiny, then $(\delta x)(\delta u)$

$(\delta x)^2$, etc.

All higher powers are extremely small.

$$f(x_0+\delta x,\ u_0+\delta u) \approx 32 + x_0^3u_0^2 + \boxed{3x_0^2u_0^2}\,\delta x$$
$$+ \boxed{2x_0^3u_0}\,\delta u$$
$$+ \text{super tiny} \leftarrow \text{lump this into disturbance}$$

This gives an approximate "linear" model in the neighborhood of (x_0, u_0).

↑Actually, "affine" since there is the constant term in front