Lecture 18: Least squares continued

Admin: Homework 7 due, homework 8 out tonight.

Simple application of SVD:

**Theorem:** For any matrix $A$,
- $\text{rank}(A^TA) = \text{rank}(A)$
- $\text{R}(A^TA) = \text{R}(A^T)$
- $\text{N}(A^TA) = \text{N}(A)$.

**Proof:** We have seen this already, but it is much easier to prove using the SVD.

Let the SVD of $A$ be

$$ A = \sum_{i} \lambda_i u_i \mathbf{v}_i^T \quad \Rightarrow \text{rank}(A) = \# \{ i \mid \lambda_i > 0 \} $$

$$ A^TA = (\sum_{i} \lambda_i v_i \mathbf{u}_i^T)(\sum_{j} \lambda_j u_j \mathbf{v}_j) $$

$$ = \sum_{i} \lambda_i^2 v_i \mathbf{v}_i^T $$

since $u_i \cdot u_j = \{ 1 \text{ if } i = j \}$

This is a SVD for $A^TA \Rightarrow \text{rank}(A^TA) = \# \{ i \mid \lambda_i^2 > 0 \}$

$$ \Rightarrow \text{rank}(A^TA) = \text{rank}(A). \Box $$

LEAST SQUARES" FITTING & APPLICATIONS TO DATA ANALYSIS

**Problem:** System of equations

$$ AX = B $$

but $\text{R} \neq \text{R}(A)$!

$\Rightarrow$ Find $x$ to minimize $\|Ax - b\|$

**Algebraically:**

$$ A = \sum_{i} \lambda_i u_i \mathbf{v}_i^T $$

$$ A^T = \sum_{i} \frac{1}{\lambda_i} \mathbf{v}_i^T u_i $$

**Reading:**

Meyer 4.6

Strang 3.3
\[ A' = \sum_{i: \lambda_i > 0} \lambda_i \vec{v}_i \vec{a}_i \] (pseudo-inverse)

\[ \Rightarrow x = A^+ b \]

minimizes \( \|Ax - b\| \)

**Example: Linear regression**

\[ \text{pinv}([1 \ 2; \ 1 \ 3; \ 1 \ 4]) \ast [5; \ 7; \ 11] \]

\[ \text{ans} = \]

\[ \begin{array}{c}
-1.3333 \\
3.0000
\end{array} \]

Setting: \( m \) data points

\[ (x_1^{(i)}, x_2^{(i)}, y^{(i)}), \ldots, (x_k^{(i)}, y^{(i)}) \]
\[(x_{1}^{(s)}, x_{2}^{(s)}, \ldots, x_{k}^{(s)}, y^{(s)})
\]
\[
(x_{1}^{(m)}, x_{2}^{(m)}, \ldots, x_{k}^{(m)}, y^{(m)})
\]

\[\text{components known exactly, e.g., date/time}\]
\[\text{component that we want to predict}\]

Goal: Find the best linear predictor for \(y\),
\[a_0, a_1, a_2, \ldots, a_k \in \mathbb{R}\]
to minimize total squared error:
\[
\sum_{s=1}^{m} |a_0 + a_1 x_1^{(s)} + a_2 x_2^{(s)} + \ldots + a_k x_k^{(s)} - y^{(s)}|^2
\]

Answer:
\[
\begin{pmatrix}
1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_k^{(1)} \\
1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_k^{(2)} \\
1 & x_1^{(3)} & x_2^{(3)} & \cdots & x_k^{(3)} \\
1 & \vdots & \vdots & \ddots & \vdots \\
1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_k^{(m)}
\end{pmatrix} \begin{pmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_k
\end{pmatrix} = \begin{pmatrix}
y^{(1)} \\
y^{(2)} \\
y^{(3)} \\
\vdots \\
y^{(m)}
\end{pmatrix}
\]

The least-squares solution is \(A^+y\).

Four ways to find \(x\) to minimize \(\|Ax - b\|\):

1. Compute pseudo-inverse \(A^+\), set \(x = A^+b\).
   **Pros:** Easy to remember, one line `pinv(A)*b` in Matlab
   **Cons:** Slow for large \(A\), numerically unstable

2. Find \(x\) manually, with calculus:
\[
E_{\text{squared error}} = \left\| \begin{pmatrix}
a_1 \\
a_2 \\
\vdots \\
a_m
\end{pmatrix} \begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_m
\end{pmatrix} - \begin{pmatrix}
b_1 \\
b_2 \\
\vdots \\
b_m
\end{pmatrix} \right\|^2
\]
\[
= \sum_{s=1}^{m} (x_1 + a_2 x_2 - b_0)^2
\]
\[ \frac{\partial}{\partial x_1} = 2 \sum_j (x_1 + a_j x_2 - b_j) = 0 \quad \Rightarrow \quad m x_1 + (\sum_j a_j) x_2 = (\sum_j b_j) \]
\[ \frac{\partial E}{\partial x_2} = 2 \sum_j a_j (x_1 + a_j x_2 - b_j) = 0 \quad \Rightarrow \quad (\sum_j a_j) x_1 + (\sum_j b_j) x_2 = (\sum_j a_j) b_j \]

Let \[ \bar{a} = \frac{1}{m} \sum_j a_j \] (average value of \(a_j\))
\[ \bar{b} = \frac{1}{m} \sum_j b_j \] (average of \(b_j\’s\))

First equation
\[ \Rightarrow \text{intercept } x_1 = \bar{b} - \bar{a} x_2 \]
(this is 0 if the data is centered so \(\bar{a} = \bar{b} = 0\))

Second equation
\[ \Rightarrow x_2 = \frac{\sum_j a_j b_j - m \bar{a} \bar{b}}{\sum_j a_j^2 - m \bar{a}^2} \]
\[ = \frac{\text{Cov}(A, B)}{\text{Var}(A)} \quad \text{if } A \text{ and } B \text{ are random variables} \]
with \(\text{Pr}[\sum_j (a_j, b_j)] = \frac{1}{m}\).

Pros: None (okay, it gives an easy closed-form solution)

3. Solve \( A^T A x = A^T b \)

**Theorem:** Let \( A \) be an \( m \times n \) real matrix with rank \( n \).

Then for any \( b \),
- The equation \( A^T A x = A^T b \) is feasible
  (i.e., it has a solution \( x \))
- The unique solution is
  \[ (A^T A)^{-1} A^T b = A^T b \]

**Proof:** There is a solution because
- \( R(A^T A) = R(A^T) \) (shown above)
- \( A^T b \in R(A^T) \)
\[ \Rightarrow A^T b \in R(C(A^T A)) \]

The solution is unique because \( N(A) = \{0\} \) (because \( \text{rank}(A) = \dim R(A^T) = n \), by assumption, and \( \text{by rank-nullity } \dim R(A^T) + \dim N(A) = n \))
(because \( \text{rank}(A) = \dim \mathcal{R}(A^T) = n \), by assumption,
and by rank-nullity \( \dim \mathcal{R}(A^T) + \dim \mathcal{N}(A) = n \))

So why is \((A^TA)^{-1}A^T b = A^+ b\), as claimed?
There are several ways of seeing it.
Algebraically: Using the SVD of \( A \),
\[
A^+ = \sum_{i: \lambda_i > 0} \frac{1}{\lambda_i} v_i u_i^T,
\]
\[
(A^TA)^{-1}A^T = \left[ \sum_{i} \frac{1}{\lambda_i^2} v_i v_i^T \right] (\sum_{d} \lambda_d v_d u_d^T)
\]
\[
= (\sum_{i} \frac{1}{\lambda_i} v_i v_i^T) (\sum_{d} \lambda_d v_d u_d^T)
\]
\[\text{since rank}(A) = n, \lambda_1, \ldots, \lambda_n \text{ are all } > 0 \]
\[
= \sum_{i} \frac{1}{\lambda_i} v_i u_i^T
\]
\[= A^+ \checkmark
\]

More geometrically:
\[x = A^+ b \text{ solves } Ax = P_{\mathcal{R}(A)} b\]
\[\Rightarrow A^T A x = A^T P_{\mathcal{R}(A)} b = A^T b \text{ since } A^T = A^T P_{\mathcal{R}(A)} \text{ (in general } A = P_{\mathcal{R}(A)} A P_{\mathcal{R}(A)^T}) \]

**Corollary:** For any \( m \times n \) real matrix \( A \), of rank \( n \),
\[A^+ = (A^TA)^{-1}A^T\]

**Proof:** Since we just showed \( A^+ b = (A^TA)^{-1}A^T b \)
for any \( b \).

**Pros:** Fast

**Cons:** condition number of \( A^T A \)
\[
= (\text{condition number of } A)^2
\]

**Method:**
1. Solve \( \begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \)
2. \text{Pros: Maintains sparsity of } A \)
Problem 4.6.9, p.237 of Meyer

Latest research has focused on finding fast randomized approximation algorithms, based on dimension reduction à la Johnson-Lindenstrauss Lemma, e.g.,


Evaluating the quality of a fit:

Of course there is a lot of statistics to consider when running linear regressions.

The most basic statistic is the “R² value”, also known as the “coefficient of determination.”

Definition: Given data points \((x_1, y_1), \ldots, (x_m, y_m)\), and the least-squared-error fit line \(y = \alpha + \beta x\),

\[
R^2 = 1 - \frac{\sum_j (y_j - \alpha - \beta x_j)^2}{\sum_j (y_j - \bar{y})^2}
\]

where \(\bar{y} = \frac{1}{m} \sum_j y_j\)

\[
= 1 - \frac{\text{total squared error from best-fit line}}{\text{total squared error from horizontal line } \bar{y}}
\]

Observe: \(0 \leq R^2 \leq 1\)

and higher is better

• Collinear data points will have \(R^2 = 1\)
  (since best-fit line will fit exactly)
• But what counts as a “good fit”
  depends very much on the field (e.g., macroeconomics)

Interpretation of \(R^2\): Explained variance

If the data were distributed randomly about the mean \(\bar{y}\),
the observed variance is \(\text{Var}(Y) = \frac{1}{m} \sum_{j=1}^{m} (y_j - \bar{y})^2\).

For data distributed randomly off the line \(y = \alpha + \beta x\),

i.e., \(y_j = \alpha + \beta x_j + e_j\), the remaining variance is

\[
\frac{1}{m} \sum_j e_j^2 = \frac{1}{m} \sum_j (y_j - \alpha - \beta x_j)^2.
\]
\( R^2 \) measures how much of the observed variance is explained by the observed \( x \) data points.

**Probabilistic interpretation: Why least squares?**

Choosing \( \alpha \) to minimize \( \| A\alpha - \tilde{y} \| \) is geometrically natural, but why is this natural in data analysis?

One reason (of several): In a natural probabilistic model, least-squares regression finds the maximum likelihood estimate of \( \alpha \).

**Model:** Assume that the \( y \)-values are generated via

\[
y(i) = \alpha \cdot x(i) + \epsilon(i)
\]

where \( \epsilon(i) \sim N(0, \sigma^2) \) — independent, identically distributed, Gaussian random noise.

\[
\Rightarrow p(x(i) \mid x(i)) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2\sigma^2}(y(i) - \alpha \cdot x(i))^2}
\]

↑ probability density function of \( y(i) 

for model parameters \( \alpha \)

\[
\Rightarrow p(x(i), ... y(m) \mid x(i), ... x(m)) \frac{1}{(2\pi\sigma^2)^{m/2}} e^{\frac{-1}{2\sigma^2} \sum_{i=1}^{m} (y(i) - \alpha \cdot x(i))^2}
\]

\[
\Rightarrow \text{Setting } \alpha \text{ to minimize } \sum_{i} (y(i) - \alpha \cdot x(i))^2
\]

maximizes the likelihood of the observed data.

(Notice: This holds for any value of the variance \( \sigma^2 \),

and \( \sigma \) need not be known to choose \( \alpha \).

**Linear regression with weights:**

If errors between observations are correlated, or if some observations are more reliable than others, then minimizing the weighted sum of squared errors is better than unweighted.

**Example:** Say

\[
y_1 = \alpha x_1 + \epsilon_1 \quad , \quad \epsilon_1 \sim N(0, 1) \quad \text{← more reliable}
\]

\[
y_2 = \alpha x_2 + \epsilon_2 \quad , \quad \epsilon_2 \sim N(0, 2) \quad \text{← less reliable}
\]

\[
\Rightarrow \sum_{i} w_i (y_i - \alpha x_i)^2
\]

EE 441 Page 7
\[ y_2 = a x_2 + \epsilon_2 \quad , \quad \epsilon_2 \sim N(0, 2) \quad \text{less reliable} \]

\[ P_N[y_1, y_2 | x_1, x_2] = \frac{1}{(2\pi)^{1/2} \sigma_1 \sigma_2} e^{-\frac{1}{2} (y_1 - ax_1)^2 \sigma_1^{-2} (y_2 - ax_2)^2 \sigma_2^{-2}} \]

\[ \approx \exp\left[-\frac{1}{2} \left\{ (y_1 - ax_1)^2 + \frac{1}{2} (y_2 - ax_2)^2 \right\} \right] \]

\[ \Rightarrow \text{The max-likelihood } \lambda \text{ minimizes the weighted sum} \]

\[ S = \frac{1}{\sigma_1^2} (y_1 - ax_1)^2 + \frac{1}{\sigma_2^2} (y_2 - ax_2)^2 \]

\[ \frac{1}{2} \frac{\partial S}{\partial a} = \frac{x_1 y_1}{\sigma_1^2} (y_1 - ax_1) + \frac{x_2 y_2}{\sigma_2^2} (y_2 - ax_2) = 0 \]

\[ \Rightarrow \lambda = \frac{x_1 y_1}{\sigma_1^2} + \frac{x_2 y_2}{\sigma_2^2} \]

More generally: To choose \( \lambda \) to minimize

\[ \sum_{j=1}^{n} \omega_j (y_j - x_j \cdot \lambda)^2, \]

observe

\[ \| W (y - Ax) \|^2 \quad \text{where} \quad W = \begin{pmatrix} \omega_1 & \omega_2 & \cdots & 0 \\ 0 & 0 & \cdots & \omega_m \end{pmatrix} \]

\[ A = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} \]

This is standard least-squares for the equation

\[ WA = Wy \]

\[ \Rightarrow \text{best } \lambda \text{ satisfies} \]

\[ A^T W^T W A \lambda = A^T W^T W y , \quad \text{etc.} \]

Practical application: Locally weighted linear regression

Example:

```math
x = Table[{i, data[[i, 1]]}, {i, 1, m}];
linear_regression = LinearModelFit[x, x, x];
```

![Data plot and linear regression graph](image)
```math
\textbf{Linear regression}
```

```
data = ListPlot[data]
X = Table[{1, data[[j, 1]]}, {j, 1, m}];
Y = data[[All, 2]]; 
Ymean = Total@Y/n;
linear = PseudoInverse[X].Y; 
plinlinear = Plot[{1, x}.linear, {x, -6, 6};
Show[data, plotlinear];
```

```
"R^2 value:"
R2Linear = 1 - \frac{\sum_{j=1}^{n} (y[j] - Y[j].linear)^2}{\sum_{j=1}^{n} (y[j] - Ymean)^2} // N;
```

```
R^2 value:
0.915751
```

```
\textbf{Cubic regression}
```

```
\textbf{Higher Order Regression} data, order
```

```
Module[{m, y, ymean, X, a, plot, min, max, R2}, 
m = Length[data];
y = data[[All, 2]]; 
X = Table[Prepend[Table[data[[j, 1]], {k, 1, order}], 1], {j, 1, m}]; // N;
a = PseudoInverse[X].Y;
Print["a = ", a // Chop];
{min, max} = {Min[data[[All, 1]]] - 2, Max[data[[All, 1]]] + 2};
plot = Plot[Table[a, {k, 0, order}].x, {x, min, max}];
Ymean = Total@Y/n;
R2 = 1 - \frac{\sum_{j=1}^{n} (y[j] - Y[j].a)^2}{\sum_{j=1}^{n} (y[j] - Ymean)^2} // N;
Print["R2 = ", R2];
```

```
\textbf{11th order} \((1, x, x^2, \ldots, x^9)\)
```

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
Locally weighted linear regression

To predict \( y \) at point \( x \), run weighted linear regression giving higher weights to data points close to \( x \)!

\[ w_i = e^{-\frac{1}{2\sigma^2} (x_i - x)^2} \]

\[
m = \text{Length}[\text{data}];
\]
\[
y = \text{data}[\text{All}, 2];
\]
\[
X = \text{Table}[[1, \text{data}[j, 1]], \{j, 1, m\}] /\!\!/ \Sigma[n, 32] &;
\]
\[
\tau = 1;
\]
\[
\text{weightedlinearplot} = \text{Plot}[\text{\textbar}
\]
\[
W = \text{DiagonalMatrix}[\text{Table}[\text{Exp}[-\frac{1}{2\tau^2} (\text{data}[j, 1] - x)^2], \{j, 1, m\}]];
\]
\[
\sigma = \text{PseudoInverse}[W.X].W.y;
\]
\[
\{1, \sigma, \}
\]
\[
(x, -10, 10);
\]
\[
\text{Show}[\text{dataplot, weightedlinearplot, PlotRange} \rightarrow \{-8, 8\}, \{-6, 6\}\}]
\]

For \( x = -2.5 \), the closest data points \((-3,-1)\) and \((-2,0)\) are weighted highest.

\[
\textbf{Note:} \, \text{After running weighted linear regression at } x, \text{ we don't keep the line—just the prediction at } x.
\]
\[
\bullet \text{ This is slower than running linear regression just once.}
\]
\[
\bullet \text{ Full "training set" must be kept to make predictions. (known as a "non-parametric" learning algorithm)}
\]
THE SINGULAR-VALUE DECOMPOSITION
AND DISTANCE TO LOWER-RANK MATRICES

Example:

$$A = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 \end{pmatrix}$$

rank (A) = 4

Intuitively:
What is the rank-one matrix closest to A?

$$B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\|B - A\| = \left\| \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 \end{pmatrix} \right\| = \frac{1}{2}$$

What is the rank-two matrix closest to A?

$$C = \begin{pmatrix} 1 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\|C - A\| = \frac{1}{2}$$

What is the rank-three matrix closest to A?

$$D = \begin{pmatrix} 1 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\|D - A\| = \frac{1}{4}$$

Theorem: Let

$$A = \sum_{i=1}^{r} \lambda_i \mathbf{u}_i \mathbf{v}_i^T$$

be an SVD of a matrix A, with singular values

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0,$$

where r = rank (A).

Then for any k < r, the rank-k matrix closest to A is

$$B = \sum_{i=1}^{k} \lambda_i \mathbf{u}_i \mathbf{v}_i^T$$

Observe:

B is just a truncation of A’s SVD, keeping the k largest singular values. B’s distance to A is

$$\|A - B\| = \left\| \sum_{i=k+1}^{r} \lambda_i \mathbf{u}_i \mathbf{v}_i^T \right\| = \lambda_{k+1}.$$
has full rank, but is $\frac{1}{1000}$ close to a rank-one (singular) matrix.

This has many, many (many) applications.

Proof: The rank-$k$ matrix $B$ reaches $\|B-A\| = \lambda_{k+1}$; so we just need to show that no other rank-$k$ matrix gets closer to $A$.

Let $C$ be any rank-$k$ matrix (with the same dimensions as $A$). To show $\|C-A\| \geq \lambda_{k+1}$, we need to find a vector $x$ with $\|x\| = 1$ and

$$\|(C-A)x\| \geq \lambda_{k+1}.$$  \hspace{1cm} ---- our goal!

\[
\begin{bmatrix}
\lambda_1 & 0 & \ldots & 0 \\
0 & \lambda_2 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & \lambda_k
\end{bmatrix}
\begin{bmatrix}
\lambda_1 & 0 & \ldots & 0 \\
0 & \lambda_2 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & \lambda_k
\end{bmatrix}
\]

then set $\xi = \xi_3$ to get $\|C-A\| = \|A\| = \lambda_3$.

Intuition: All vectors $x$ in $\text{Span} \{v_1, v_2, \ldots, v_{k+1}\}$ are stretched by at least $\lambda_{k+1}$, and there's no way $C$ can cancel out all these $k+1$ dimensions!

If these are all $m \times n$ matrices, then $\dim N(C) = n - k$.

Since $\dim \text{Span} \{v_1, \ldots, v_{k+1}\} = k+1$, they must intersect.

(Alternative proof: $C \cdot \begin{pmatrix} \lambda_1 & \lambda_2 & \ldots & \lambda_{k+1} \end{pmatrix}$ has rank $k$, so must have non-trivial nullspace.

For this $x$, $\|(C-A)x\| = \|A\| \geq \lambda_{k+1} \|x\|$ \hspace{1cm} $\Box$

Remark:

Dimension reduction is a major theme in computational linear algebra and most approaches for data analysis.

* Linear regression, as we have presented it, effectively reduces the dimension of the data by 1, e.g., fitting a 1D line to a 2D cloud of data points.

* The Johnson-Lindenstrauss Lemma projects $n$ data points to $\log n$ (random) dimensions, approximately preserving angles & lengths (after rescaling).
• Matrix rank reduction via the SVD is another example. Moreover, it is used in Principal Component Analysis for reducing the dimensionality of data sets. We'll see this later.