Last time

• Basis functions for linear regression
• Normal equations
• QR
• SVD - briefly
This time

- Geometry of least squares (again)
- SVD – more slowly
- LMS
- Ridge regression
Geometry of least squares

Columns of $X$ define a $d$-dimensional linear subspace in $n$-dimensions. $\hat{y}$ is projection of $y$ into that subspace. Here $n=3$, $d=2$.

\[
X = \begin{pmatrix} 1 & 2 \\ 1 & -2 \\ 1 & 2 \end{pmatrix}, \quad y = \begin{pmatrix} 8.8957 \\ 0.6130 \\ 1.7761 \end{pmatrix}, \quad \hat{y} = X\hat{w} = \begin{pmatrix} 5.3359 \\ 0.6130 \\ 5.3359 \end{pmatrix}
\]

\[
X = \begin{pmatrix} 0.5774 & 0.5774 \\ 0.5774 & -0.5774 \\ 0.5774 & 0.5774 \end{pmatrix}, \quad y = \begin{pmatrix} 0.9784 \\ 0.0674 \\ 0.1954 \end{pmatrix}, \quad \hat{y} = \begin{pmatrix} 0.7048 \\ 0.0810 \\ 0.7048 \end{pmatrix} \quad \text{Unit norm}
\]
Orthogonal projection

• Projection of \( y \) onto \( X \)

\[
\operatorname{Proj}(y; X) = \arg\min_{\hat{y} \in \text{span}\{x_1, ..., x_n\}} \| y - \hat{y} \|_2.
\]

• Let \( r = y - \hat{y} \). Residual must be orthogonal to \( X \). Hence

\[
x_j^T (y - \hat{y}) = 0 \Rightarrow X^T (y - Xw) = 0 \Rightarrow w = (X^T X)^{-1} X^T y
\]

• Prediction on training set

\[
\hat{y} = X\hat{w} = X(X^T X)^{-1} X^T y \overset{\text{def}}{=} H y
\]

\text{Hat matrix}

• Residual is orthogonal

\[
X^T (y - H y) = X^T (y - X \hat{w}) = X^T y - X^T X (X^T X)^{-1} X^T y = 0
\]
This time

- Geometry of least squares (again)
- SVD – more slowly
- LMS
- Ridge regression
Eigenvector decomposition (EVD)

• For any square matrix $A$, we say $\lambda$ is an eval and $u$ is its evec if

$$Au = \lambda u, \quad u \neq 0.$$ 

• Stacking up all evecs/vals gives

$$AU = U\Lambda = \begin{pmatrix} u_1 & u_2 & \cdots & u_n \end{pmatrix} \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}$$

• If evecs linearly independent

$$A = U\Lambda U^{-1}.$$ 

diagonalization
EVD of symmetric matrices

- If $A$ is symmetric, all its evals are real, and all its evecs are orthonormal, $u_i^T u_j = \delta_{ij}$
- Hence $U^T U = UU^T = I$, $|U| = 1$.
- and

$$A = U \Lambda U^T = \sum_{i=1}^{n} \lambda_i u_i u_i^T$$

$$A = \begin{pmatrix} u_1 & u_2 & \cdots & u_n \end{pmatrix} \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix} \begin{pmatrix} - & u_1^T & - \\ - & u_2^T & - \\ \vdots & \vdots & \vdots \\ - & u_n^T & - \end{pmatrix}$$

$$= \lambda_1 \begin{pmatrix} | \end{pmatrix} (- u_1^T -) + \cdots + \lambda_n \begin{pmatrix} | \end{pmatrix} (- u_n^T -)$$
For any real matrix

\[ A = U \Sigma V^T = \sigma_1 \begin{pmatrix} u_1 \end{pmatrix} (-v_1^T -) + \cdots + \sigma_r \begin{pmatrix} u_r \end{pmatrix} (-v_r^T -) \]

\[ U^T U = I \]
\[ V^T V = VV^T = I \]
Truncated SVD

• Rank k approximation to a matrix

\[ A_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_k^T = \mathbf{U}_{:,1:k} \Sigma_{1:k,1:k} \mathbf{V}_{:,1:k}^T \]
load clown; % built-in image
[U,S,V] = svd(X,0);
k = 20;
Xhat = (U(:,1:k)*S(1:k,1:k)*V(:,1:k)');
image(Xhat);
• If $A$ is symmetric positive definite, then $\text{svals}(A) = \text{evals}(A)$, $\text{leftSvec}(A) = \text{rightSvec}(A) = \text{evecs}(A)$ modulo sign changes.

```matlab
>> A = randpd(3)
A =
0.9302 0.4036 0.7065
0.4036 0.8049 0.4521
0.7065 0.4521 0.5941

>> [U, S, V] = svd(A)
U =
-0.6597 0.5148 -0.5476
-0.5030 -0.8437 -0.1872
-0.5584 0.1520 0.8155

S =
1.8361 0 0
0 0.4772 0
0 0 0.0159

V =
-0.6597 0.5148 -0.5476
-0.5030 -0.8437 -0.1872
-0.5584 0.1520 0.8155
```
SVD and EVD

- For arbitrary real matrix $A$
- $\text{leftSvecs}(A) = \text{evecs}(A A')$
- $\text{rightSvecs}(A) = \text{evecs}(A' A)$
- $\text{Svals}(A)^2 = \text{evals}(A' A) = \text{evals}(A A')$
SVD for least squares

- We have

\[ X = UDV^T \]

\[ \hat{w} = (X^TX)^{-1}X^Ty \]

\[ X^TXw = X^Ty \text{ (premultiply by } X^TX) \]

\[ VDU^TUDV^Tw = VDU^Ty \text{ (SVD expansion)} \]

\[ VD^2V^Tw = VDU^Ty \text{ (since } U^TU = I \text{ and } DD = D^2) \]

\[ D^2V^Tw = DU^Ty \text{ (premultiply by } V^T) \]

\[ V^Tw = D^{-1}U^Ty \text{ (premultiply by } D^{-2}) \]

\[ w = VD^{-1}U^Ty \text{ (premultiply by } V) \]

```matlab
[U, D, V] = svd(X, 0);
Dinv = diag(1./(diag(D)));
w = V*Dinv*U'*y;
```

What if \( D_j = 0 \) (so rank of \( X \) is less than \( d \))?
Pseudo inverse

- If \( D_j = 0 \), use

\[
    w = V D^\dagger U^T y \quad \text{def} \quad X^\dagger y, \quad D^\dagger = \text{diag}(\sigma^{-1}_1, \ldots, \sigma^{-1}_r, 0, \ldots, 0)
\]

```matlab
function B = pinv(A)
[U,S,V] = svd(A,0);
s = diag(S);
r = sum(s > tol);  % rank
w = diag(ones(r,1) ./ s(1:r));
B = V(:,1:r) * w * U(:,1:r)';
```

- Of all solutions \( w \) that minimize \( ||Xw - y|| \), the \texttt{pinv} solution also minimizes \( ||w|| \)

\[
    w = X\backslash y;
w2 = \texttt{pinv}(X) * y;
\]

\[
\text{[norm(w) norm(w2)]}
\]

\[
>> 10.8449 \quad 10.8440
\]
This time

- Geometry of least squares (again)
- SVD – more slowly
- LMS
- Ridge regression
Gradient descent

- QR and SVD take $O(d^3)$ time
- We can find the MLE by following the gradient

$$w_{k+1} = w_k - \eta_k g(w_k)$$

$$g(w) \propto X^T (Xw - y) = \sum_{i=1}^n x_i (w^T x_i - y_i)$$

- $O(d)$ per step, but may need many steps
Stochastic gradient descent

- Approximate the gradient by looking at a single data case
  \[ g(w_k) \approx x_i (w^T x_i - y_i) \]

- Can be used to learn online

---

**Algorithm 1: LMS algorithm**

1. Initialize \( w \)
2. \( t \leftarrow 0 \)
3. repeat
   4. \( t \leftarrow t + 1 \)
   5. \( i \leftarrow t \mod n \)
   6. \( w \leftarrow w + \eta(y_i - w^T x_i) x_i \)
   7. \( \eta \leftarrow \eta \times s \)
4. until converged
This time

- Geometry of least squares (again)
- SVD – more slowly
- LMS
- Ridge regression
Ridge regression

- Minimize penalized negative log likelihood

\[-\ell(w) + \lambda \|w\|^2_2\]

- Weight decay, shrinkage, L2 regularization, ridge regression
Regularization D=14

\[ \lambda = 0 \]

\[ \lambda = 10^{-5} \]

\[ \lambda = 10^{-3} \]
Why it works

- Coefficients if $\lambda=0$ (MLE)

$-0.18, 10.57, -110.28, -245.63, 1664.41, 2647.81, -96527669.94, 19319.66, -41625.65, -16626.90, 31483.81, 54$ 

- Coefficients if $\lambda=10^{-3}$

$-1.54, 5.52, 3.66, 17.04, -2.63, -23.06, -0.37, -8.49, 7.92, 5.40, 8.29, 7.75, 1.78, 2.03, -8.42,$

- Small weights mean the curve is almost linear
  (same is true for sigmoid function)
Ridge regression

• The objective function is

\[ w = \arg \min_w \sum_{i=1}^{n} (y_i - x_i^T w - w_0)^2 + \lambda \sum_{j=1}^{d} w_j^2 \]

• We don’t shrink \( w_0 \). We should standardize first.

• Constrained formulation

\[ w = \arg \min_w \sum_{i=1}^{n} (y_i - x_i^T w - w_0)^2 \text{ s.t. } \sum_{j=1}^{d} w_j^2 \leq t \]

• Find the penalized MLE

\[
J(w) = (y - Xw)^T (y - Xw) + \lambda w^T w \quad \text{See book}
\]

\[
w = (X^T X + \lambda I)^{-1} X^T y
\]
• Recall

\[ w = (X^T X + \lambda I)^{-1} X^T y \]

• Expanded data:

\[ \tilde{X} = \begin{pmatrix} X \\ \sqrt{\lambda} I_d \end{pmatrix}, \quad \tilde{y} = \begin{pmatrix} y \\ 0_{d \times 1} \end{pmatrix} \]

\[ J(w) = (\tilde{y} - \tilde{X}w)^T(\tilde{y} - \tilde{X}w) = (y - Xw)^T(y - Xw) + \lambda w^T w \]

\[ \hat{w}_{ridge} = \tilde{X} \backslash \tilde{y}. \]
• Recall

\[ w = (X^T X + \lambda I)^{-1} X^T y \]

• Homework: let \( X = U \Sigma V^T \).

\[ w = V(D^2 + \lambda I)^{-1} DU^T y \]

• Cheap to compute for many lambdas (regularization path), useful for CV
• We have

\[ \hat{y} = X\hat{w}_{ridge} = UDV^T V(D^2 + \lambda I)^{-1} DU^T y \]

\[ = U\tilde{D}U^T y = \sum_{j=1}^{d} u_j \tilde{D}_{jj} u_j^T y \]

\[ \tilde{D}_{jj} \overset{\text{def}}{=} [D(D^2 + \lambda I)^{-1} D]_{jj} = \frac{d_j^2}{d_j^2 + \lambda} \]

\[ \hat{y} = X\hat{w}_{ridge} = \sum_{j=1}^{d} u_j \frac{d_j^2}{d_j^2 + \lambda} u_j^T y \]

\[ \hat{y} = X\hat{w}_{ls} = (UDV^T)(VD^{-1}U^T y) = UU^T y = \sum_{j=1}^{d} u_j u_j^T y \]

\[ d_j^2 / (d_j^2 + \lambda) \leq 1 \quad \text{Filter factors} \]
Ridge and PCA

- $D_j^2$ are the eigenvalues of empirical cov mat $X^T X$.
- Small $d_j$ are directions $j$ with small variance: these get shrunk the most, since most ill-determined

$$\hat{y} = X\hat{w}_{ridge} = \sum_{j=1}^{d} u_j \frac{d_j^2}{d_j^2 + \lambda} u_j^T y$$
Principal components regression

- Can set $Z = \text{PCA}(X,K)$ then $w = \text{regress}(X,y)$ using a pcaTransformer object
- PCR sets (transformed) dimensions $K+1,...,d$ to zero, whereas ridge uses all weighted dimensions. Ridge predictions usually more accurate.
- Feature selection (see later) sets (original) dimensions $K+1,...,d$ to zero. Ridge is usually more accurate, but may be less interpretable.
Degrees of freedom

All have $D=14$ but clearly differ in their effective complexity

\[ \hat{y} = S(X)y \]

\[ df(S) \overset{\text{def}}{=} \text{trace}(S) \]

\[ df(\lambda) = \sum_{j=1}^{d} \frac{d_j^2}{d_j^2 + \lambda} \]
Tikhonov regularization

$$\min_f \frac{1}{2} \int_0^1 (f(x) - y(x))^2 \, dx + \frac{\lambda}{2} \int_0^1 [f'(x)]^2 \, dx$$
Discretization

\[
\min_{f} \frac{1}{2} \int_{0}^{1} (f(x) - y(x))^2 dx + \frac{\lambda}{2} \int_{0}^{1} [f'(x)]^2 dx
\]

\[
\min_{f} \frac{1}{2} \sum_{i=1}^{n-1} (f_i - y_i)^2 + \frac{\lambda}{2} \sum_{i=1}^{n-1} (f_{i+1} - f_i)^2
\]

\[
\min_{f} \frac{1}{2} \sum_{i=1}^{n} (f_i - y_i)^2 + \frac{\lambda}{4} \sum_{i=1}^{n} \left[ (f_i - f_{i-1})^2 + (f_i - f_{i+1})^2 \right]
\]

Boundary conditions: \(f_0 = f_1, f_{n+1} = f_n\)
Matrix form

\[
\begin{align*}
\min f & \quad \frac{1}{2} \sum_{i=1}^{n} (f_i - y_i)^2 + \frac{\lambda}{4} \sum_{i=1}^{n} \left[ (f_i - f_{i-1})^2 + (f_i - f_{i+1})^2 \right] \\
J(w) &= ||y - w||^2 + \lambda ||Dw||^2 \\
D &= \begin{pmatrix}
-1 & 1 \\
-1 & 1 \\
\vdots & \vdots \\
-1 & 1
\end{pmatrix}
\\
||Dw||^2 &= w^T(D^T D)w = \sum_{i=1}^{n-1} (w_{i+1} - w_i)^2
\\
D^T D &= \begin{pmatrix}
1 & -1 & 0 & 0 & \cdots & 0 & 0 & 0 \\
-1 & 2 & -1 & 0 & \cdots & 0 & 0 & 0 \\
0 & -1 & 2 & -1 & \cdots & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \cdots & -1 & 2 & -1 \\
0 & 0 & 0 & 0 & \cdots & 0 & -1 & 1
\end{pmatrix}
\end{align*}
\]
$$\min_w \left\| \left( \frac{I_n}{\sqrt{\lambda}D} \right) w - \begin{pmatrix} y \\ 0 \end{pmatrix} \right\|^2$$

**Listing 1:**

```matlab
D = spdiags(ones(N-1,1)*[-1 1], [0 1], N-1, N);
A = [speye(N); sqrt(lambda) * D];
b = [y; zeros(N-1,1)];
w = A \ b;
```