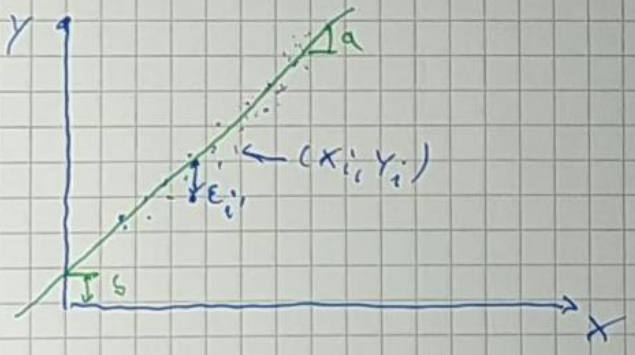
Ordinary Least Squares

- simplest case: only one feature: $X \in \mathbb{R}$, $Y \in \mathbb{R}$



model: $Y = X \cdot \beta + \varepsilon + b$

(β: slope, ε: noise, b: intercept)

assume: $\varepsilon \sim \mathcal{N}(0, \sigma^2)$

(mean, variance is independent of X_i and is same for all instances)

$$y = f(X, \eta) \quad \Leftrightarrow \quad p(Y \mid X)$$

$$y = X \cdot \beta + b + \varepsilon$$

deterministic eq of features and random number ε

$$r = Y - (X\beta + b) \sim \varepsilon \sim \mathcal{N}(0, \sigma^2)$$

posterior prob of ~~resid~~ residual r is Gaussian

- determine optimal parameters β, b by maximum likelihood principle

$\hat{=}$ choose $\hat{\beta}, \hat{b}$ such that likelihood of TS $p(TS \mid \hat{\beta}, \hat{b})$ is maximized,

simplify via the i.i.d assumption $\Rightarrow$ probability of TS factorizes over i

$$p(TS \mid \beta, b) = \prod_{i=1}^{N} p((x_i, y_i) \mid \beta, b) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \frac{(y_i - (x_i\beta + b))^2}{\sigma^2}\right)$$

instead of maximizing this, we minimize the negative log-likelihood (NLL)

$$-\log p(TS \mid \beta, b) = \sum_{i=1}^{N} \left[\frac{(y_i - (x_i\beta + b))^2}{2\sigma^2} + \log \sqrt{2\pi\sigma^2} \right]$$

drop, because argmin is independent of σ^2

$$\hat{\beta}, \hat{b} = \underset{\beta, b}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - (x_i\beta + b))^2$$

ordinary least squares problem

To solve: set derivative of the loss to zero: $\text{Loss}(\{(x_i, y_i)\}_{i=1}^N) = \sum_{i=1}^N (y_i - (x_i\beta + b))^2$

$$\frac{\partial \text{Loss}}{\partial b} = -2 \sum_{i=1}^N (y_i - (x_i\beta + b)) \stackrel{!}{=} 0$$

$$\sum_{i=1}^N y_i = \sum_{i=1}^N x_i\beta + \underbrace{\sum_{i=1}^N b}_{N \cdot b} \qquad | : N$$

$$\frac{1}{N}\sum_{i=1}^N y_i = \bar{y} = \beta \cdot \bar{x} + b \qquad \bar{x} = \frac{1}{N}\sum_{i=1}^N x_i$$

$\Rightarrow$ regression line always goes through the center of mass $(\bar{x}, \bar{y})$

$\Leftrightarrow$ when we center the data before doing regression, the regression line always goes through the origin $\Rightarrow$ $\boxed{b = 0}$

$\Rightarrow$ in the sequel, we assume centered data $\boxed{x \leftarrow x - \bar{x}, \; y \leftarrow y - \bar{y}}$

centered loss: $\text{Loss} = \sum_{i=1}^N (y_i - x_i\beta)^2$

$$\frac{\partial \text{Loss}}{\partial \beta} = -2 \sum_i x_i (y_i - x_i\beta) \stackrel{!}{=} 0 \qquad \left(\sum_{i=1}^N x_i^2\right) \cdot \beta = \sum_{i=1}^N x_i y_i \qquad \boxed{\hat{\beta} = \frac{\sum_i x_i y_i}{\sum_i x_i^2}}$$

Generalize to multiple features: $x_i \in \mathbb{R}^D$ $x_i\beta$ scalar product, $\beta \in$ column vector $\mathbb{R}^D$

$\text{Loss} = \sum_i (y_i - x_i\beta)^2 = (y - x\beta)^T (y - x\beta)$ in matrix notation

$$\frac{\partial \text{Loss}}{\partial \beta} = -2 x^T (y - x\beta) \stackrel{!}{=} 0$$

x: $N \times D$ y: $N \times 1$ β: $D \times 1$

$$\boxed{(x^T x)\beta = x^T y}$$ normal equations of OLS problem

- assume "true generative process" for data: $y_i = x_i \beta^* + b^* + \varepsilon_i \quad \varepsilon_i \sim N(0, \sigma^2)$
  β^* and b^* unknown, must be learned — variance independent of i
- if data is centered: $\frac{1}{N}\sum_i x_i = 0 \quad \frac{1}{N}\sum_i y_i = 0 \Rightarrow \hat{b} = 0$ regardless of β
  $\Rightarrow$ assume throughout that data is centered
- maximum likelihood loss $\rightarrow$ negative log $\rightarrow$ squared loss

$$\hat{\beta} = \arg\min_\beta \sum_i (y_i - x_i \beta)^2 = \arg\min_\beta (y - X\beta)^T (y - X\beta)$$

- setting the derivative $\frac{\partial}{\partial \beta}$ to zero: normal equations

$$(X^T X) \cdot \beta = X^T y$$

- formal solution

$$\hat{\beta} = \underbrace{(X^T X)^{-1} X^T}_{X^+} y$$

X^+: pseudo-inverse, Moore-Penrose inverse

$X^+ = \underbrace{(X^T X)^{-1}}_{S} X^T$ is a generalization of the inverse matrix X^{-1} (which only exists if X is square ($N = D$)) to the case $N > D$

for the pseudo-inverse to exist, the scatter matrix $S = X^T X$ must be invertible

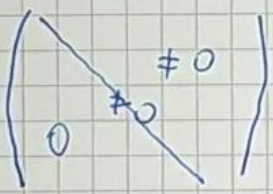
S is square $D \times D$ and is invertible if features are <u>not redundant</u>.

features are redundant if one feature can be expressed as a linear combination of the others: $x_j = \sum_{j' \neq j} w_{j'} x_{j'}$ (x_j: one column of X; $x_{j'}$ is redundant)

if no feature is redundant, X has <u>full rank</u> $\Rightarrow S^{-1}$ exists

Algorithms to solve the normal equations:

① Cholesky decomposition: factorize $S = R^T \cdot R$ — R: upper triangular matrix, R^T: lower —"—



always exists if X (and therefore S) have full rank, simple algorithm

$\Rightarrow$ normal equations $R^T \cdot \underbrace{R \cdot \beta}_{=z} = X^T y$

two places: - solve $R^T \cdot z = X^T y$ via forward substitution for z

- solve $R \cdot \beta = z$ via backward substitution for β

$D = 3$

$$r_{11} \cdot z_1 = (X^T y)_1$$
$$r_{21} \cdot z_1 + r_{22} \cdot z_2 = (X^T y)_2$$
$$r_{31} \cdot z_1 + r_{32} \cdot z_2 + r_{33} \cdot z_3 = (X^T y)_3$$

$$z_1 = \frac{(X^T y)_1}{r_{11}}$$
$$z_2 = \frac{(X^T y)_2 - r_{21} \cdot z_1}{r_{22}}$$
$$z_3 = \frac{(X^T y)_3 - r_{31} \cdot z_1 - r_{32} \cdot z_2}{r_{33}}$$

advantages: - easy

- S can be computed without ever creating full matrix X, is a good thing when N is very big

```
S = 0
for i = 1 ... N
    for j = 1 ... D
        for j' = 1 ... D
            S_jj' = S_jj' + X_ij · X_ij'
```

only need a single row of X at a time

disadvantage: numerically less stable than other alg.

condition number of a matrix: $\kappa = \|X\| \, \|X^+\| \geq \frac{\lambda_{max}}{\lambda_{min}}$

$\|\cdot\|$ some norm of X

- all eigenvalues of S non-negative
- if a feature is redundant, $\lambda_{min} = 0$

  if it is close to redundant, ~~[illegible]~~

  $\lambda_{min} \ll \lambda_{max}$

$\Rightarrow \kappa = \frac{\lambda_{max}}{\lambda_{min}}$ is large

$\Rightarrow$ ~~compute~~ computing S^{-1} results in large numerical errors

$S = X^T X \quad \Rightarrow \quad \kappa(S) = \kappa(X)^2$

Frobenius norm $\|X\|_F^2 = \sum_{ij} X_{ij}^2$

spectral norm $\|X\|_2 = \lambda_{max}$ largest eigenvalue of X

[spectral norm $\|X^+\| = \frac{1}{\lambda_{min}}$ smallest eigenvalue of X singular

why is this a problem: rule of thumb for the numerical error of solving a linear system: $A \cdot \beta = b$, ~~where~~

~~error in β = $\kappa(A) \cdot \mu$~~

number of valid digits in $\beta = -\log_{10}(\kappa \cdot \mu)$ μ machine precision of the floating point data type

$\mu_{float32} = 10^{-7}$ $\mu_{float64} = 10^{-16}$

$\Rightarrow$ if X has bad condition, e.g. $\kappa = 10^8$

S has very bad condition $\kappa(S) = \kappa(X)^2 = 10^{16}$

$\Rightarrow$ even if we use float64 ("double") $\Rightarrow \kappa(S) \cdot \mu = 1$

$\Rightarrow$ at best one valid digit $\Rightarrow$ alternative alg. work with X directly $\Rightarrow$ 8 valid digits

Alg. (2) singular value decomposition of X

can always express $X = U \cdot \Lambda \cdot V^T$

$N \times D$ ↑ U orthonormal $D \times D$; Λ diagonal $D \times D$; V^T orthonormal $N \times D$

elements of $\Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \ddots \\ & & & \lambda_D \end{pmatrix}$ are called "singular values" of X

= generalization of eigenvalues to rectangular matr.

solve normal eq. using SVD: $X^T = V \cdot \Lambda \cdot U^T$

$$S = X^T X = V \cdot \Lambda \cdot \underbrace{U^T \cdot U}_{= \mathbb{I}} \cdot \Lambda \cdot V^T = V \cdot \Lambda^2 \cdot V^T$$

$$S^{-1} = (X^T X)^{-1} = \underbrace{V^{-T}}_{(V^T)^{-1} = (V^{-1})^{-1} = V} \cdot \Lambda^{-2} \cdot \underbrace{V^{-1}}_{V^T} = V \cdot \Lambda^{-2} V^T$$

$$X^+ = (X^T X)^{-1} \cdot X^T = V \cdot \Lambda^{-2} \underbrace{V^T \cdot V}_{\mathbb{I}} \cdot \Lambda \cdot U^T = V \Lambda^{-1} U^T$$

$$\Rightarrow \boxed{\hat{\beta} = X^+ y = V \Lambda^{-1} U^T \cdot y}$$

compute SVD of X, invert the singular values $\Lambda^{-1} = \begin{pmatrix} 1/\lambda_1 & & 0 \\ & \ddots & \\ 0 & & 1/\lambda_D \end{pmatrix}$, calculate $\hat{\beta}$

advantages: - work with X directly ⇒ numerically very stable

- also works for redundant features: some (or several) $\lambda_j = 0$

⇒ do not touch them when computing $\hat{\beta}$

- disadvantages of SVD: - needs X explicitly, problematic for big N
  - SVD decomposition alg. very complicated to implement
  ⇒ always use library routine, not re-program it

(3) LSQR algorithm [Paige & Saunders 1982]: when X is very big

[SVD works on standard computers up to 1000×1000 matrices]

In practice, large feature matrices often have special structure,

e.g. • sparse; most elements of X are 0 (but X still full rank)

• Toeplitz: every row is a shifted version of previous row

⇒ handle X of size $10^6 \times 1000$ and above

least-squares alg. takes advantage of the special structure

decomposition $X = U \; B \; V^T$

U: orthonormal

B: upper bi-diagonal matrix $B = \begin{pmatrix} b_{11} & b_{12} & & 0 \\ & \ddots & \ddots & \\ & & & b_{D-1,D} \\ 0 & & & b_{DD} \end{pmatrix}$

trick 1: compute the decomposition such that only two rows / columns of U and V are needed simultaneously at any time

trick 2: alg. never accesses X directly, but only via matrix-vector product subroutines; if it needs $X \cdot a$ ⇒ call x_times_vec(a)

$X^T \cdot b$ ⇒ call xt_times_vec(b)

user must provide these subroutines to the LSQR program, these subroutines are tailored to special structure of X