Basis expansions

- Linear regression, logistic regression, LDA
  - classification by linear hyperplanes
  - easy to fit and interpret
- $f(Y|X)$ is non-linear and non-additive
  - augment X with transformations of X
  - $h_m(X): R^p \to R$
  - The model $f(X) = \sum_{m=1}^{M} \beta_m h_m(X)$
  - linear basis expansion in X
- widely used h_m
  - linear model $h_m(X) = X_m, m=1..p$
  - polynomial terms $h_m(X) = X_m^2$ tweak one region ↑ flap another
    - # of variables grows exponentially in p
  - Function of a vector $h_m(X) = \log(X_j)$, $\sqrt{X_j}$, $\|X\|$
  - Indicators $h_m(X) = I(L_m \le X_k \le U_m)$
    Piecewise constant model in each region
  - Piece-wise polynomials and splines
    - Dictionary D of basis functions
- ★ Method for controlling model complexity
  restriction, selection, regularization
  - restriction: limit the class of functions
    $f(X) = \sum_{j=1}^{p} f_j(X_j) = \sum_{j=1}^{p}\sum_{m=1}^{M_j} \beta_{jm} h_{jm}(X_j)$
    M_j is the limitation
  - selection: adaptively scan the dict and include h_m that significantly contribute to the fit of model
    AIC, BIC, significance test
  - regularization: use entire dict but restrict the coefficients
    ridge and lasso.

$RSS(f,\lambda) = \sum_{i=1}^{N}(y_i - f(x_i))^2 + \lambda\int (f''(t))^2 dt$

- λ: smooth parameter, choose λ using cross-validation
  - $\lambda = 0$ any function
  - $\lambda = \infty$ linear function
- maximized with a natural cubic spline
- equivalent to generalized ridge regression

$EPE(\hat f_\lambda) = E(Y - \hat f_\lambda)^2 = Var(Y) + E[Bias^2(\hat f_\lambda) + Var(\hat f_\lambda)]$
$= \sigma^2 + MSE(\hat f_\lambda)$

- Piecewise fits
  1. piecewise constant
     - $h_1(X) = I(X < \xi_1)$, $h_2(X) = I(\xi_1 \le X < \xi_2)$
       $h_3 = I(\xi_2 \le X)$
     - $\hat f(X) = \sum_{m=1}^{3} \hat\beta_m h_m(X)$
     - $\hat\beta_m = \bar Y_m$ the mean of mth region
  2. piecewise linear
     3 extra basis functions:
     $h_{m+3} = h_m(X)X$, $m = 1,..,3$
  3. restricted piecewise linear
     - $f(\xi_1^-) = f(\xi_1^+)$ $f(\xi_2^-) = f(\xi_2^+)$
     - 4 free parameters out of 6
       $h_1(X) = 1$, $h_2(X) = X$, $h_3(X)$
       $= (X - \xi_1)_+$ $h_4(X) = (X - \xi_2)_+$

- Splines
  - smoother functions
    cubic splines: continuous first and second derivatives at the knots.
    $h_1(X) = 1$, $h_2(X) = X$, $h_3(X) = X^2$, $h_4(X) = X^3$
    $h_5(X) = (X - \xi_1)_+^3$ $h_6(X) = (X - \xi_2)_+^3$
  - 6 basis functions
    6-dimensional linear space of functions
    $6 = (3 \text{ regions}) \times (4 \text{ parameters per region})$
    $- (2 \text{ knots}) \times (3 \text{ constrains per knot})$
  - more generally
    Order-M spline with knots $\xi_j, j=1,...,K$
    - piecewise polynomial up to order M
    - has continuous derivatives up to order $M-2$
      piecewise constant: order-1
      piecewise continuous linear: order-2
      cubic: order-4
    - basis set: $h_j(X) = X^{j-1}, j=1,...,M$
      $h_{M+l}(X) = (X - \xi_l)_+^{M-1}, l = 1,...,K$
    - #: $M+K = (K+1)\times M - K\times(M-1)$
      $M+K$ dimensional

Cubic spline is the lowest-order spline for which the knot-discontinuity is not visible to human eye

- Natural cubic splines
  - motivation
    extra stability
    - polynomials and splines have erratic behavior near boundaries
    - variance explode
    - extrapolation is problematic
  - extra constraints natural cubic
    - variance-bias tradeoff
      bias ↑ variance ↓
    - Linear functions beyond boundary knots
  - K knots ⟶ K basis functions
  - start from basis → impose constraints
    derive reduced basis:
    $N_1(X) = 1$ $N_2(X) = X$
    $N_{k+2}(X) = d_k(X) - d_{K-1}(X)$
    where $d_k(X) = \frac{(X-\xi_k)_+^3 - (X-\xi_K)_+^3}{\xi_K - \xi_k}$
  - second and third derivatives are 0 for $X \ge \xi_K$

Example: South African Heart Disease
Logistic regression:
$\log\frac{P(chd=1|X)}{P(chd=0|X)} = \log\frac{P(chd=1|X)}{1-P(chd=1|X)}$
$= \theta_0 + h_1(X_1)^T\theta_1 + \dots + h_p(X_p)^T\theta_p$
- θ_j vector for term (predictor) j, coefficients of natural spline basis functions h_j
- use 4 natural spline basis functions for each term
- 3 internal knots + 2 boundary knots, uniform quantiles
- binary predictor has single coefficient
- more compactly: combine p vectors of basis functions + constant term in a big vector $h(X)$, then the model is simply $h(X)^T\theta$
- backward stepwise selection + AIC to drop terms
- plot prediction ± 2SE
  $\hat f_j(X_j) = h_j(X_j)^T \hat\theta_j$

Fitting multivariate models
example: linear additive model
$$Y = \alpha + \sum_{j=1}^{p} f_j(X_j) + \varepsilon$$
1. $\hat\alpha = \frac{1}{N}\sum_1^N y_i$ $\hat f_j \equiv 0$
2. While True:
   $\hat f_j = S_j[\{y_i - \hat\alpha - \sum_{k\ne j}\hat f_k(x_{ik})\}_1^N]$
   $\hat f_j = \hat f_j - \frac{1}{N}\sum_{i=1}^{N}\hat f_j(x_{ij})$
   until $\hat f_j$ change less than a prespecified threshold.

- S_j is a cubic spline.
- Iteratively smooth the residual fit for one predictor at a time till converge.

Kernel methods

- + One-dimensional kernel smoothers
  - different model at each point x_0
    - only use observations close to target x_0
    - weight neighbors x_i with kernel $K_\lambda(x_0, x_i)$
    - λ parameter ← smoothing parameter
  - kernel is a function that quantifies the similarity of two observations.
- + K nearest neighbors
  - $\hat f(x_0) = Ave(y_i | x_i \in N_k(x_0))$ (discontinuous in x) bumpy
- + Nadaraya-Watson kernel-weighted average
  - $\hat f(x_0) = \frac{\sum_{i=1}^{N} K_\lambda(x_0,x_i)y_i}{\sum_{i=1}^{N} K_\lambda(x_0,x_i)}$
    $K_\lambda(x_0, x) = D\left(\frac{|x - x_0|}{\lambda}\right)$ and
    $D(t) = \begin{cases}\frac{3}{4}(1-t^2) & \text{if } |t| \le 1 \\ 0 & \text{otherwise}\end{cases}$ ← Epanechnikov quadratic kernel
  - Points near the boundary have weight ~0 so it is smooth
- + Adaptive width
  - $h_\lambda(x_0)$ - width function that determines the neighborhood of x_0
  - $K_\lambda = D\left(\frac{|x - x_0|}{h_\lambda(x_0)}\right)$
  - concerns: the choose of λ ⟶

- larger λ → lower variance higher bias
- $h_\lambda(x)$ constant → bias is constant
  variance inversely proportional to density of points
- Nearest neighbor → variance is constant
  bias inversely proportional to density of points
- + other concerns:
  1. ties in x_i → averaging y_i using additional weight w_i
  2. boundary issue: neighborhoods tend to contain less points on the boundary.
  3. Epanechnikov and tri-cube function are compact kernel
     tri-cube $D(t) = \begin{cases}(1-|t|^3)^3 & \text{if } |t| \le 1; \\ 0 & \text{otherwise}\end{cases}$
     Gaussian kernel is non-compact

Local linear regression (LLR)
- motivation
  asymmetry of the kernel on the boundary of the domain → bias
- solution: fit straight lines rather than constant
  Also helps if values of x are unequally spaced, will make a $\boxed{\text{first-order}}$ correction.
- LLR solves weighted least squares problem
  for each x_0:
  $\min_{\alpha(x_0)\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0,x_i)[y_i - \alpha(x_0) - \beta(x_0)x_i]^2$
  only use it to evaluate the fit at single point x_0. $\hat f(x_0) = \hat\alpha(x_0) + \hat\beta(x_0)x_0$
- define $b(x)^T = (1, x)$
  B: $N\times 2$ regression matrix with ith row $b(x_i)^T$
  $W(x_0)$: $N\times N$ diagonal matrix with ith diagonal element $K_\lambda(x_0, x_i)$

  Then
  $\hat f(x_0) = b(x_0)^T(B^TW(x_0)B)^{-1}B^TW(x_0)y$
  $= \sum_{i=1}^{N} l_i(x_0) y_i$ $l_i(x_0)$ local function

  $l_i(x_0)$ combine the weighting kernel K_λ and least squares
  ↑ equivalent kernel

Local polynomial regression
- minimize
  $\min_{\alpha(x_0)\beta_j(x_0), j=1...d} \sum_{i=1}^{N} K_\lambda(x_0,x_i)\left[y_i - \alpha(x_0) - \sum_{j=1}^{d}\beta_j(x_0)x_i^j\right]^2$
  solution: $\hat f(x_0) = \hat\alpha(x_0) + \sum_{j=1}^{d}\hat\beta_j(x_0)x_0^j$
  avoid "trimming the hills and filling the valleys"
- Bias-variance tradeoff as d ↑ variance ↑ bias ↓ $\left\{ Var(\hat f(x_0)) = \sigma^2\|l(x_0)\|\right.$
  polynomial degree
- Local linear fits help bias dramatically at the boundaries at a modest cost in variance. Local quadratic fits do little at the boundaries for bias, but increase the variance a lot.
- Local quadratic fits tend to be most helpful in reducing bias due to curvature in the interior of the domain.

Choice of λ ← choose using cross-validation
- Epanechnikov/tri-cube: radius of the support region
- Gaussian: λ is the standard deviation
- K-NN: λ is the number k
  ↑ span k/N
- bias-variance tradeoff
  window size narrow, # small observations, large variance, small bias
  window size wide: large bias, small variance
- model complexity
  estimated degree of freedom
  $df = trace(S_\lambda)$ $(S_\lambda)_{ij} = l_i$

Local regression in R^p
- p-dimensional kernel
  polynomial fit of degree d
- $b(X)$: vector of polynomial terms in X of maximum degree
  - $d=1, p=2$: $(1, X_1, X_2)$
    $d=2, p=2$: $(1, X_1, X_2, X_1^2, X_2^2, X_1X_2)$
  At each $x_0 \in R^p$ solve
  $\min_{\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0,x_i)(y_i - b(x_i)^T\beta(x_0))^2$
  where $K_\lambda(x_0, x) = D\left(\frac{\|x - x_0\|}{\lambda}\right)$
  $\|\cdot\|$ Euclidean norm
  $\|x\| = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$

Structured kernel
- as dimension ↑ # neighbor points ↓
  The fraction of points close to the boundary increases to one as the dimension grows
- Local regression becomes less useful when dimension > 2 or 3
- Need additional constraints on local regression, to counter curse of dimension
- option: modify the kernel
  use a positive semidefinite matrix A to weigh different coordinates
- $K_{\lambda,A}(x_0, x) = D\left(\frac{(x - x_0)^T A (x - x_0)}{\lambda}\right)$
  Restrict A to downgrade or omit directions

Local likelihood
- Any parametric model can be made local if the fitting method accommodates observation weights.
  log likelihood: $\cancel{l(\beta)}$
  $l(\beta) = \sum_{i=1}^{N} l(y_i, x_i^T\beta)$
  local one:
  $l(\beta(x_0)) = \sum_{i=1}^{N} K_\lambda(x_0,x_i)\, l(y_i, x_i^T\beta(x_0))$
  Eg. logistic regression and log-linear models.

Kernel density estimation
- natural (bumpy) estimate $\hat f_X(x_0) = \frac{\# x_i \in N(x_0)}{N\lambda}$
- smooth Parzen estimate is preffered
  $\hat f_X(x_0) = \frac{1}{N\lambda}\sum_{i=1}^{N} K_\lambda(x_0, x_i)$
  Eg. Gaussian kernel
  $K_\lambda(x_0, x_i) = \phi(|x - x_0|/\lambda)$
  ϕ pdf of $N(0,1)$
  $\hat f_X(x) = \frac{1}{N}\sum_{i=1}^{N}\phi_\lambda(x - x_i)$
- equivalent to local weighted average

Kernel density classification
- Nonparametric classification
  $\hat P(G=j|X=x_0) = \frac{\hat\pi_j \hat f_j(x_0)}{\sum_{k=1}^{J}\hat\pi_k \hat f_k(x_0)}$
- Naïve Bayes
  $\hat P(G=k|X=x_0) = \frac{\hat\pi_k \hat f_k(x_0)}{\sum_{j=1}^{J}\hat f_j \hat f_j(x_0)} = \frac{\hat\pi_k\prod_{l=1}^{p}\hat f_{kl}(x_0)}{\sum_{j=1}^{J}\hat\pi_j\prod_{l=1}^{p}\hat f_{jl}(x_0)}$
- often works well when
  - biased class densities, but less variance
  - bias may be small near decision boundary
- connection to GAMs
  $\log\frac{P(G=k|X)}{P(G=J|X)} = \log\frac{\pi_k\prod_{l=1}^{p}f_{kl}(X)}{\pi_J\prod_{l=1}^{p}f_{Jl}(X)}$
  $= \log\frac{\pi_k}{\pi_J} + \sum_{l=1}^{p}\log\frac{f_{kl}(X)}{f_{Jl}(X)}$
  $= \beta_{0k} + \sum_{l=1}^{p} g_l(X)$
  1. similar to GAM
  2. GAMs do not specify probability distribution on X, naive bayes does
  3. LDA vs logistic regression
     generative ↗ ⇕ ↘ discriminant
     naive bayes vs GAM

- Support Vector Machines
  - support vector classifier
  - Kernel trick
    - Define high-dimensional feature vector
    - Prevents underfitting
  - Sparsity and large margin principle
    - Ensure we don't use all dimensions
    - Prevent overfitting
  - separable case
    - Data: $\{(x_i, y_i)\}_{i=1,...,N}$, $x_i \in R^p$, $y_i \in \{-1, 1\}$
    - Define a hyperplane:
      $\{x: f(x) = x^T\beta + \beta_0 = 0\}$, $\|\beta\| = 1$
    - classification rule:
      $G(x) = sign(x^T\beta + \beta_0)$
    - can find β such that $y_if(x_i) > 0$ for all i ↓

- can find β to maximize the margin between training points with class 1 and -1
- optimization problem
  $\max_{\beta,\beta_0,\|\beta\|=1} M$ s.t. $y_i(x_i^T\beta + \beta_0) \ge M$
  $i = 1,...,N$
  ⇕
  $\min_{\beta,\beta_0}$ s.t. $y_i(x_i^T\beta + \beta_0) \ge 1$

non-separable case
- Goal: maximize $\|M\|$
  some points can be on the wrong side of margin
- solution: slack variables ξ
  $\max_{\beta,\beta_0,\|\beta\|=1} M$ s.t. $\xi_i \ge 0$, $\sum_{i=1}^{N}\xi_i \le$ const
  $y_i(x_i^T\beta + \beta_0) \ge M - \xi_i$ or
  $y_i(x_i^T\beta + \beta_0) \ge M(1 - \xi_i)$ standard one
  1. ξ is the proportional amount by which the prediction $f(x_i) = x_i^T\beta + \beta_0$ is on the wrong side of its margin.
  2. Misclassifications occur when $\xi > 1$
     Bound $\sum\xi \le$ constant ⟹ bound the total number of training misclassifications.
  3. equivalent to
     $\min_{\beta,\beta_0}\|\beta\|$ s.t. $y_i(x_i^T\beta + \beta_0) \ge 1 - \xi_i$
     $\xi_i \ge 0$ $\sum\xi_i \le$ constant

Computation
- is quadratic with linear inequality constraints.
  solution: quadratic programming
- is equivalent:
  $\min_{\beta,\beta_0}\frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i$ s.t.
  $\xi_i \ge 0$, $y_i(x_i^T\beta + \beta_0) \ge 1 - \xi_i$
- C: cost parameter
  ∞ for separable case
- solution: $\hat\beta = \sum_{i=1}^{N}\hat\alpha_i y_i x_i$
  1. $\hat\alpha_i \ne 0$ only for support vectors where $y_i(x_i^T\beta + \beta_0) \ge M(1-\xi_i)$
  2. $\xi_i = 0 \to 0 < \alpha_i < C$ $\xi_i > 0 \to \alpha_i$

Decision
$\hat G(x) = sign[\hat f(x)] = sign[x^T\hat\beta + \hat\beta_0]$

Computation details
- objective function
  $\min_{\beta,\beta_0}\frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i$ s.t. $\xi_i \ge 0$,
  $y_i(x_i^T\beta + \beta_0) \ge 1 - \xi_i$

$\boxed{\text{Large } C}$ ⟶ small margin, discourage $\xi_i > 0$, fewer support points, more overfitting, more wiggly boundary ★

Lagrange Primal function
- $L_P = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i$
- To minimize w.r.t β, β_0 and ξ_i, set derivatives to 0:
  $\beta = \sum_{i=1}^{N}\alpha_i y_i x_i$ ① $0 = \sum_{i=1}^{N}\alpha_i y_i$ ② $\alpha_i = C - \mu_i$ ③
  $\alpha_i, \mu_i, \xi_i \ge 0 \ \forall i$
- substitute to primal, the dual objective
  $L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_{i'}y_iy_{i'}x_i^Tx_{i'}$ ④
  Karush-Kuhn-Tucker conditions
  $\alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] = 0$ ⑤
  $\mu_i\xi_i = 0$ ⑥
  $y_i(x_i^T\beta + \beta_0) - (1 - \xi_i) \ge 0$ ⑦
- combine ① - ⑦
  $\hat\beta = \sum_{i=1}^{N}\hat\alpha_i y_i x_i$

- Kernel trick $f(x) = \beta_0 + \sum_{i \in S}\alpha_i K(x, x_i)$
- + Create non-linear classifiers
  - transform feature space to higher-dimensions
  - separate the transformed features by maximum margin hyperplane
  - # of dimension ↑ overfitting ↑
- + Dual objective function
  $L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_{i'}y_iy_{i'}x_i^Tx_{i'}$
  ↓
  $L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_{i'}y_iy_{i'}K(x_i, x_{i'})$
- + polynomial kernel $K(x_i, x_{i'}) = (x_i^Tx_{i'} + 1)^d$
  - d is a tunable parameter
  - requires one addition and one exponention more than the origin dot product
- + RBF/gaussian
  $K(x_i, x_{i'}) = e^{-\frac{\|x_i - x_{i'}\|^2}{r}}$ r is a parameter
- + sigmoid
  $K(x_i, x_{i'}) = \tanh(\kappa_1 x_i^T x_{i'} - \kappa_2)$

- Example: polynomial kernel
  feature $x = (x_1, x_2)$
  kernel degree = 2 $K(x, x') = (\langle x, x'\rangle + 1)^2$
  $K(x, x') = (\langle x, x'\rangle + 1)^2$ $1 + 2x_1x_1' + 2x_2x_2' + x_1^2x_1'^2 + x_2^2x_2'^2 + 2x_1x_1'x_2x_2'$
  $= (1 + x_1x_1' + x_2x_2')^2 = $
  equivalent to basis functions
  $h_1(x) = 1$ $h_2(x) = \sqrt2 x_1$ $h_3(x) = \sqrt2 x_2$ $h_4(x) = x_1^2$ $h_5(x) = x_2^2$
  $h_6(x) = \sqrt2 x_1x_2$ → $K(x, x') = (\langle x, x'\rangle + 1)^2 = \langle h(x), h(x')\rangle$
  no need to explicitly specify h, less flexible, easy computation

SVM regularization
Solution to convex constraint optimization in SVM is same as solution to
$\max_{\beta,\beta_0}\sum_{i=1}^{N}[1 - y_i(h^T(x)\beta + \beta_0)]_+ + \lambda\|\beta\|^2$ $\lambda = \frac{1}{C}$ ★
- same as ridge regression, different loss function.

		minimizing function
$L(y, f(x))$		
$\log[1 + e^{-yf(x)}]$	binomial deviance	$f(x) = \log\frac{P(Y=+1\|x)}{P(Y=-1\|x)}$
$[1 - yf(x)]_+$	Hinge loss	$f(x) = sign[P(Y=+1\|x) - \frac{1}{2}]$
$[y - f(x)]^2 = [1 - yf(x)]^2$	squared error	$f(x) = 2P(Y=+1\|x) - 1$

Tree based method
- Decision tree
- impurity → choose split to minimize node "impurity"
  $i(t) = \phi(p_1, p_2, ..., p_J)$, maximized at $(\frac{1}{J}, \frac{1}{J}, ..., \frac{1}{J})$
  minimized at $(0, 0, ..., 1)$, is a symmetric function
- Quality of split s at node t
  reduction of impurity $\Delta i(s,t) = i(t) - \pi(l)i(l) - \pi(r)i(r)$
- Measures of node impurity
  - Resubstitution error $i(t) = 1 - \max_j p(j|t)$
    % of misclassified cases
    ? Problem: ignore where misclassification occurs
    400|400 → 300|100, 100|300; 400|400 → 200|400, 200|0
    left $\Delta i = \frac{1}{2} - \frac{1}{2}\times\frac{1}{4} - \frac{1}{2}\times\frac{1}{4} = \frac{1}{4}$
    right $\Delta i = \frac{1}{2} - \frac{3}{4}\times\frac{1}{3} - 0 = \frac{1}{4}$
  - Gini index (bernoulli variance)
    Two class: $i(t) = p(0|t)\cdot p(1|t) = p(0|t)(1 - p(0|t))$
    Multiple class: $i(t) = \sum_{j=1}^{J} p(j|t)\cdot(1 - p(j|t))$
    Impurity reduction ⟶ variance reduction
  - Entropy
    Two class: $i(t) = -p(0|t)\cdot\log p(0|t) - p(1|t)\cdot\log p(1|t)$
    Multi class: $i(t) = -\sum_{j=1}^{J} p(j|t)\cdot\log p(j|t)$
    Average amount of info gathered by drawing a point from the node ⟹ Max impurity reduction = min loss of info

Search for split

400|400 → 300|100, 100|300 ; 400|400 → 200|400, 200|0

reduction of impurity

Gini index

left $\Delta i = \frac{1}{2}\times\frac{1}{2} - \frac{1}{2}\times\frac{3}{4}\times\frac{1}{4} - \frac{1}{2}\times\frac{3}{4}\times\frac{1}{4} = 0.0625$

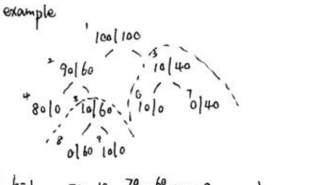
right $\Delta i = \frac{1}{2}\times\frac{1}{2} - \frac{3}{4}\times\frac{2}{3}\times\frac{1}{3} - \frac{1}{4}\times1\times0 = 0.0833$

So the right one is better

The algorithm for constructing tree is simply keeping split until the impurities of leaves are 0

- Local greedy search; globally suboptimal
  eg. true labels $P \text{ xor } Q$ can't be classified by splitting on P or Q
- Partial solution: grow maximal tree, then prune.

example



$k=1$ $\quad g_1(t_3) = \dfrac{\frac{70}{200}\times\frac{10}{70} - \frac{70}{200}\times\frac{60}{70}\times 0 - 0}{2-1} = \frac{1}{20}$

$g_1(t_2) = \dfrac{\frac{50}{200}\times\frac{10}{50} - 0}{2-1} = \frac{1}{20}$ $\quad$ prune t_3, t_5

$g_1(t_1) = \dfrac{\frac{150}{200}\times\frac{60}{150} - 0 - 0 - 0 - 0}{3-1} = \frac{3}{20}$

Cost-complexity pruning

- Tree T, $R(T)$ resubstitution error, $|T|$ tree size
  Total cost of the tree $C_\alpha(T) = R(T) + \alpha|T|$
  α is the parameter penalizing tree complexity
- For every α, there exists a smallest subtree $T(\alpha)$ of T_{max}:
  1. No subtree of T_{max} has lower cost than $T(\alpha)$
     $C_\alpha(T(\alpha)) = \min_{T \le T_{max}} C_\alpha(T)$
  2. If there is a tie, we pick the smallest tree:
     If $C_\alpha(T) = C_\alpha(T(\alpha))$ the $T(\alpha) \le T$
- It is impossible to have two non-nested subtrees with same min cost
- Vary α, we can construct a decreasing sequence of subtrees $T_1 > T_2 > T_3 > \ldots > \{t_1\}$ ← root node
- T_1: min subtree of T_{max} with same Resubstitution error $\quad R(t) = \hat{r}(t)\cdot p(t)$
  merge two node if $R(t) = R(l) + R(r)$
- cost of T_t
  1. As an intermediate node: $C_\alpha(T_t) = R(T_t) + \alpha|T_t|$
  2. as leaf: $C_\alpha(T) = R(T) + \alpha\cdot 1$
     $C_\alpha(t) = C_\alpha(T_t) \rightarrow R(T_t) + \alpha|T_t| = R(t) + \alpha$
     $\alpha = \dfrac{R(T_t) - R(t)}{|T_t| - 1}$
  - For any t, increase α beyond this level, pruned tree is better
  - Obtain the next tree by prune the current
  - T_k is the smallest minimizing subtree for $\alpha \in [\alpha_k, \alpha_{k+1})$

$k=2$: $\quad g_2(t_2) = \dfrac{\frac{150}{200}\times\frac{60}{150} - 0 - \frac{70}{200}\times\frac{10}{70}}{3-1} = \frac{5}{20}$

$g_2(t_1) = \dfrac{1\times\frac{100}{200} - 0 - \frac{70}{200}\times\frac{10}{70} - \frac{50}{200}\times\frac{10}{50}}{3-1} = \frac{4}{20}$ $\quad$ prune t_1

$\alpha: \{\alpha_1 = 0,\ \alpha_2 = \frac{1}{20},\ \alpha_3 = \frac{4}{20}\}$

Select optimal tree ← single tree: unstable, not smooth,

1. construct full tree on full dataset 2. compute $\{\alpha_1, \alpha_2, \ldots\}$
3. obtain $\{T_1, T_2 \ldots t\}$ for α 4. define sequence $\{\beta_j = \sqrt{\alpha_{j-1}\cdot\alpha_j}\}$
5. cross-validation a. build a sequence of trees over β_j b. compute prediction error for left-out obs
6. For every β_j, sum the # misclassifications over folds
7. select β_j with min # ↗ 8. Report the β_j subtree of full data

Bagging (bootstrap aggregation)

- motivation
  Average prediction on B independent datasets to ↓ prediction variance with unchanged bias
  $\hat{f}_{avg} = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x)$, $\quad Var\{\hat{f}_{avg}\} = \dfrac{Var\{\hat{f}^b(x)\}}{B}$
  Sampling with replacement, grows maximal trees
- Features
  + out of bag error: virtually equivalent to leave-one-out cross-validation error
  + Variable importance: total amount of decrease in impurity or RSS by splitting on the predictor
  - Loss of interpretation: a ~~bb~~ bagged tree ≠ a tree
  Under square loss $\quad$ bagging will always decrease MSE
  $E\{Y - \hat{f}^*(x)\}^2 = E\{Y - \hat{f}_{bag}(x) + \hat{f}_{bag}(x) - \hat{f}^*(x)\}^2$
  $= E\{Y - \hat{f}_{bag}(x)\}^2 + E\{\hat{f}_{bag}(x) - \hat{f}^*(x)\}^2 \ge E\{Y - \hat{f}_{bag}(x)\}^2$
  ↑ not always hold for 0-1 cost

Random forest

- Motivation
  - bagging, prediction from trees can be correlated because of single dominant predictor
  - Average correlated predictions does not ↓ variance
- random forest decorrelates the trees,
  - introduces diversity in trees,
  - Randomly selects $m \ll p$ predictors per split
  - on average $(p-m)/p$ of the splits skip strong predictor
  - random forest with $(m=p)$ = bagging
  - As with bagging, large B won't result in overfitting

- Boosting
- Motivation
  Trees, bagging, RF can overlook a few hard-to-classify obs
- Boosting
  grow tree sequentially, slowly; next tree depends on previous one each step, weight of unexplained obs ↑
- Parameters
  - number of trees B: Large B tends to overfit
  - shrinkage parameter λ: small λ can require large B
  - number of splits d: $d=1 \rightarrow$ additive model, each tree involves one variable
    d splits can involve at most d variables (interaction step)
- Boosting algorithm
  - every step update $\hat{f}$ by adding in a shrunken version of the new tree: $\hat{f}(x) = \hat{f}(x) + \lambda\hat{f}^b(x)$
    update residuals: $r_i = r_i - \lambda\hat{f}^b(x_i)$
  - boosted model: $\hat{f}(x) = \sum_{b=1}^{B}\lambda\hat{f}^b(x)$

General approach: AdaBoost

$Y \in \{-1, 1\}$, classifier $G(x)$, $\overline{err} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \ne G(x_i))$

- combine "weak" classifiers to produce powerful "committee"
- Algorithm:
  Initialization: $w_i = \frac{1}{N}$
  For $m = 1$ to M
  1. Fit a classifier $G_m(x)$ to the training data using weights w_i
  2. Compute $err_m = \dfrac{\sum_i^N w_i I(y_i \ne G_m(x_i))}{\sum_i^N w_i}$ $\quad \alpha_m = 2\beta_m$
  3. Compute $\alpha_m = \log((1-err_m)/err_m)$
  4. $w_i = w_i \exp[\alpha_m \cdot I(y_i \ne G_m(x_i))]$ ← w_i↑ if misclassified

  Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} G_m(x)\alpha_m\right]$
- Basis function expansion Basis functions are G_m

More generally: Forward stagewise additive modeling

- Basis function expansion: $f(x) = \sum_{m=1}^{M}\beta_m b(x, \gamma_m)$
  - β_m: expansion coefficients
  - $b(x, \gamma_m)$ simple basis functions eg. $G_m \in \{-1, 1\}$

  eg. γ 1. single hidden layer neural network, γ parameterizes a linear combination of the input variables
  2. splines, γ parameterizes the variables and values for the knots.
  3. trees, γ parameterizes the splits and predictions at the terminal nodes.
- Loss function
  $\min_{\beta_m, \gamma_m}\sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M}\beta_m b(x_i, \gamma_m)\right)$
  simplified to be
  $\min_{\beta, \gamma}\sum_{i=1}^{N} L(y_i, \beta b(x_i, \gamma))$
- Algorithm
  for $m=1$ to M:
  $(\beta_m, \gamma_m) = \arg\min_{\beta, \gamma}\sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i, \gamma))$
  $f_m(x) = f_{m-1}(x) + \beta_m b(x, \gamma_m)$

  For squared loss $\quad L(y, f(x)) = (y - f(x))^2$
  $L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) = (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2$
  $= (r_{im} - \beta b(x_i, \gamma))^2$
  r_{im}, residual of mth model for ith obs
  $\beta b(x_i, \gamma)$: fits to the current residuals.

Connection of AdaBoost and forward stagewise modeling

- AdaBoost: equivalent to forward stagewise modeling with exponential loss $L(y, f(x)) = \exp(-y f(x))$
  solving $(\beta_m, G_m) = \min_{\beta, G}\sum_{i=1}^{N}\exp(-y_i(f_{m-1}(x_i) + \beta G(x_i)))$
  $= \min_{\beta, G}\sum_{i=1}^{N} w_i^{(m)}\cdot\exp[-\beta y_i G(x_i)]$
  $w_i^{(m)} = \exp[-y_i f_{m-1}(x_i)]$

  The weights depends on $f_{m-1}(x_i)$, not β or G
- Solution: $G_m = \arg\min_G\sum_{i=1}^{N} w_i^{(m)} I(y_i \ne G(x_i))$
  $\beta_m = \frac{1}{2}\log\frac{1 - err_m}{err_m}$
  $w_i^{(m+1)} = w_i^{(m)}\cdot e^{\alpha_m I(y_i \ne G_m(x_i))}\cdot e^{-\beta_m}$

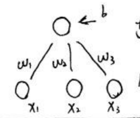
Stochastic gradient descent

$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$

learning rate, gradient, Loss function

Local minimum

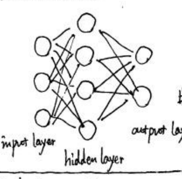
The neuron



$y = F\left(\sum_i w_i x_i + b\right)$

$F(x) = \max(0, x)$ Rectified Linear Unit (ReLU)

Neural networks



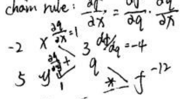
6 neurons
20 weights
26 parameters
neurons + # weights = # parameters

Back Propagation

$f(x, y, z) = (x+y)z \quad q$

$\frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1 \quad \frac{\partial f}{\partial q} = z \quad \frac{\partial f}{\partial z} = q$

chain rule: $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\cdot\frac{\partial q}{\partial x}$



example: $\frac{\partial f_1}{\partial x} = -4 \quad \frac{\partial f_1}{\partial y} = 3 \quad \frac{\partial f_2}{\partial z} = 1 \quad \frac{\partial f_2}{\partial w} = 0$

$\frac{\partial f_3}{\partial f_1} = 1 \quad \frac{\partial f_3}{\partial f_2} = 1 \quad \frac{\partial f}{\partial f_3} = 2$

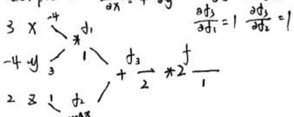


Image classification Network

- K classes → K units in final layer
- normalize so that they sum to 1
- cost: $-\ln(y_c)$ penalty for misclassification

1st layer convolution

Image value × kernel value = convolved feature

$n\times m \quad 3\times3 \quad (n-2)\times(m-2)$

eg. $7\times7 \times 3\times3 \rightarrow 5\times5$
$32\times32 \times 3\times3 \rightarrow 30\times30$

example: Input volume: $32\times32\times4$ $\quad$ # of parameters
5 kernel: $3\times3\times4$ $\quad 3\times3\times4\times5+5$
output volume: $30\times30\times5$

Hyper-parameters: width, height, depth, stride, zero-padding

max-pooling: $\begin{bmatrix}1&1&2&4\\5&6&7&8\\3&2&1&0\\1&2&3&4\end{bmatrix} \rightarrow \begin{bmatrix}6&8\\3&4\end{bmatrix}$ results in half $\quad n\times n \rightarrow \frac{n}{2}\times\frac{n}{2}$

no parameters

convoluting layer → kernel layer → convoluting → kernel

number of parameters doesn't depend on the image size only depend on the kernel

Recall: logistic regression

2 class: sigmoid activation function

$\log(\text{odds}) = \beta_0 + \beta' X \quad \pi(x) = \dfrac{e^{\beta_0 + \beta' X}}{1 + e^{\beta_0 + \beta' X}} = \dfrac{1}{1 + e^{-\beta_0 - \beta' X}}$

K class: baseline category logistic regression

$\log\frac{\pi_k(x)}{\pi_1(x)} = \alpha_k + \beta_k' X, \quad k = 2, \ldots, K$

since $\pi_k(x) = \pi_1(x) e^{\alpha_k + \beta_k' X}$ and $\sum_{k=1}^{K}\pi_k(x) = 1$

$\pi_1(x) = \dfrac{1}{1 + \sum_{k=2}^{K}\exp(\alpha_k + \beta_k' X)}, \quad k=1$

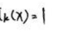
$\pi_j(x) = \dfrac{e^{\alpha_k + \beta_k' X}}{1 + \sum_{k=2}^{K} e^{\alpha_k + \beta_k' X}}, \quad k = 2, 3, \ldots, K$

Alternative logistic regression with K classes

- Log-linear model, multinomial distribution
  $\log\pi_k(x) = \alpha_k + \beta_k' X - \log Z$
  $\pi_k(x) = \frac{1}{Z}e^{\alpha_k + \beta_k' X} \quad \sum_{k=1}^{K}\pi_k(x) = 1$
  $Z = \sum_{k=1}^{K} e^{\alpha_k + \beta_k' X}$
  $\pi_k(x) = \dfrac{e^{\alpha_k + \beta_k' X}}{\sum_{k=1}^{K} e^{\alpha_k + \beta_k' X}}$ ← softmax function
  - exp exaggerates the differences between X
  - close to 1 when $\alpha_k + \beta_k' X$ ≈ max
  - only K-1 free parameters

Activation function

sigmoid: $\sigma(v) = \dfrac{1}{1 + e^{-v}}$



eg. $\sigma(\frac{1}{2}v)\ \sigma(10v)\ \sigma(sv)$ larger s means hard activation at $v=0$

$\sigma(s(v - v_0))$ shifts threshold to v_0

Deep learning: NN with >1 hidden layer

Convolutional NN: train NN on local ~~fields~~ fields.

Single hidden layer neural network

- K-class classification
  $Y_i \in (0, 1)$, p input features X
- Hidden features:
  created from linear combinations of inputs
  $Z_m = \sigma(\alpha_{0m} + \alpha_m' X), \quad m = 1, \ldots, M$
  $T_k = \beta_{0k} + \beta_k' Z, \quad k = 1, \ldots, K$
  $f_k(x) = g_k(T), \quad k = 1, \ldots, K$
  Z_m: basis expansions of original inputs
- Activation function
  eg. sigmoid $\sigma(v) = \dfrac{1}{1 + e^{-v}}$
  linear σ → linear model
- Output function
  softmax $g_k(T) = \dfrac{e^{T_k}}{\sum_l e^{T_l}}$

example:

$T_k = \beta_{0k} + \beta_{1k}\cdot Z_1 + \beta_{2k}\cdot Z_2 + \beta_{3k}\cdot Z_3$

$= \beta_{0k} + \beta_{1k}\cdot\dfrac{1}{1 + \exp(-(\alpha_{01} + \alpha_{11}X_1 + \alpha_{21}X_2))}$

$+ \beta_{2k}\cdot\dfrac{1}{1 + \exp\{-(\alpha_{02} + \alpha_{12}X_1 + \alpha_{22}X_2)\}}$

$+ \beta_{3k}\cdot\dfrac{1}{1 + \exp\{-(\alpha_{03} + \alpha_{13}X_1 + \alpha_{23}X_2)\}}$

$\alpha_{0m}\ \beta_{0k}$ → bias nodes

Optimizing the likelihood: gradient descent

loss function $J(\theta_0, \theta_1)$

- start with some θ_0, θ_1
- simultaneously change θ_0, θ_1 in direction of steepest descent
  $\theta_j^{r+1} = \theta_j^r - \eta_r\frac{\partial}{\partial\theta_j}J(\theta_0^r, \theta_1^r)$
- stop when reaching local minimum
- results sensitive to starting values

Logistic regression: parameter estimation

minimize $J(\beta) = -\log L = -\left[\sum_{i=1}^{n} Y_i\log\pi_i + \sum_{i=1}^{n}(1 - Y_i)\log(1 - \pi_i)\right]$

partial derivatives: $\dfrac{\partial J(\beta)}{\partial\beta_j} = \sum_{i=1}^{n}(\pi_i - y_i)x_j$

$\beta_j^{r+1} = \beta_j^r - \eta_r\sum_{i=1}^{n}(\pi_i^r - y_i)x_j$

Improvements: $\phi(\eta) = f(\theta_k + \eta d_k)$

- Linear search algorithm: automatically picks η_k
- Newton's method: takes into account the curvature of the space.

NN learning (parameter estimation)

Loss or Risk

Regression: RSS $R(\theta) = \sum_{k=1}^{K}\sum_{i=1}^{N}(y_i - f_k(x_i))^2$

classification: cross-entropy $R(\theta) = -\sum_{k=1}^{K}\sum_{i=1}^{N} y_i\log f_k(x_i)$

maximum likelihood parameter estimation

Back ~~Propt~~ Propagation

- Objective function $\quad Z = \sigma(\alpha_{0m} + \alpha_m' x_i)$
  $R(\theta) = \sum_{i=1}^{N} R_i = \sum_{i=1}^{N}\sum_{k=1}^{K}(y_{ij} - f_k(x_i))^2, \quad f_k(x_i) = g_k(\beta_{0k} + \beta_k' Z)$
- Derivatives $\quad \frac{\partial g_k}{\partial T}\cdot\frac{\partial T}{\partial\beta_{mk}}$
  $\dfrac{\partial R_i}{\partial\beta_{km}} = -2(y_{ik} - f_k(x_i))g_k'(\beta_k' z_i)\cdot z_{mi}$ $\quad \frac{\partial T}{\partial z}\ \frac{\partial z}{\partial v}\ \frac{\partial v}{\partial\alpha}$
  $\dfrac{\partial R_i}{\partial\alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k' z_i)\ \beta_{km}\ \sigma'(\alpha_m' x_i)\ x_{il}$
- Gradient descent at iteration $r+1$
  $\beta_{km}^{r+1} = \beta_{km}^r - \eta_r\sum_{i=1}^{N}\dfrac{\partial R_i}{\partial\beta_{km}^r} \quad \alpha_{ml}^{r+1} = \alpha_{ml}^r - \eta_r\sum_{i=1}^{N}\dfrac{\partial R_i}{\partial\alpha_{ml}^r}$
- Can write the derivatives as
  $\dfrac{\partial R_i}{\partial\beta_{km}} = \delta_{ki}z_{mi} \quad \dfrac{\partial R_i}{\partial\alpha_{ml}} = s_{mi}x_{il}$
  δ_{ki} and s_{mi} are "errors" of the current model at the output and hidden levels.
- update $r \rightarrow r+1$ in two passes: (back-propagation algorithm)
  Forward pass: compute the $\hat{f}_k(x_i)$ predicted values
  Backward pass: compute δ_{ki}, back to s_{mi}, and gradient
- Weight decay: prone to overfitting
  Minimize $R(\theta) + \lambda\left(\sum_{km}\beta_{km}^2 + \sum_{ml}\alpha_{ml}^2\right)$
  - $\lambda \ge 0$ tuning parameters λ large → weights more linearity
  - estimated by cross-validation
  - predictors are scaled so equally affected by regularization
  - more hidden nodes can be compensated for more regularization

Example NN