Linear decision boundary is a hyperplane
Select hyperparameters with validation
Sample point = feature vector
decision fn: fn $f(x)$ that maps a sample point x to a scalar
$f(x) > 0$, if $x \in$ class c
$f(x) \le 0$, if $x \notin$ class c
decision boundary is $f(x) = 0$, $x \in \mathbb{R}^d$
$\{x : f(x) = 0\} :=$ isosurface of f for isovalue 0.
Euclidean norm: $|x| = \sqrt{x \cdot x} = \sqrt{x_1^2 + \dots + x_d^2}$
Normalize by: $\frac{x}{|x|}$
Hyperplane $\in \mathbb{R}^d$ has dim $d-1$.
$H = \{x : w \cdot x = -\alpha\}$, given linear decision fn: $f(x) = w \cdot x + \alpha$
w is normal vector of H.

Centroid: doesn't require linear separability
$\mu_c :=$ mean of all vectors $\in C$
$\mu_x :=$ mean of all $\notin C$.
$f(x) = (\mu_c - \mu_x) \cdot x - (\mu_c - \mu_x) \cdot \frac{\mu_c + \mu_x}{2}$
- decision boundary is hyperplane that bisects line segment w endpoints μ_c, μ_x

Perceptron
$y_i = \begin{cases} 1 & \text{if } X_i \in \text{class } C \\ -1 & \text{if } X_i \notin C \end{cases}$ $X_i \cdot w \ge 0$, if $y_i = 1$
Goal: $y_i X_i \cdot w \ge 0$, i.e. $X_i \cdot w \le 0$, if $y_i = -1$
$L(\hat{y}_i, y_i) = \begin{cases} 0, & \hat{y}_i y_i \ge 0 \\ -\hat{y}_i y_i & \text{ow} \end{cases}$ ⇒ want same sign
$R(w) = \sum_{i \in V} -y_i X_i \cdot w$, $V =$ all i where $y_i X_i \cdot w < 0$.
Goal: minimize risk $= \sum_{i=1}^{n} L(X_i \cdot w, y_i)$

gradient of R wrt w is direction of steepest ascent
SGD: pick 1 misclassified X_i, do GD on $L(X_i \cdot w, y_i)$
- simulate general hyperplane in d dim by using hyperplane thru origin in d+1 dimensions.
Perceptron algorithm will find a soln if possible
Margin: distance from decision boundary to nearest sample point
↳ $w \cdot x + \alpha = 1$ $w^T x + \alpha = 0$ (decision boundary)
→ slab of width $\frac{2}{|w|}$ containing no sample points
Opt: Find w, α that minimize $|w|^2$ subject to
$y_i(X_i \cdot w + \alpha) \ge 1$ i where $y_i(X_i \cdot w + \alpha) = 1$ lie on margin
:= quadratic program
This is a maximum margin classifier, aka hard margin SVM w is ⊥ to decision boundary, → dir of f
Signed distance from hyperplane to X_i is $\frac{w}{|w|} \cdot X_i + \frac{\alpha}{|w|}$

Hard margin SVMs fail if data not linearly separable, sensitive to outliers
Soft Margin SVMs:
- allow some points to violate the margin, w slack variables
$y_i(X_i \cdot w + \alpha) \ge 1 - \xi_i$, $\xi_i \ge 0$ } constraints
Point i has nonzero ξ_i iff it violates the margin
Opt: minimize $|w|^2 + C \sum_{i=1}^{n} \xi_i$ C affects bias variance tradeoff

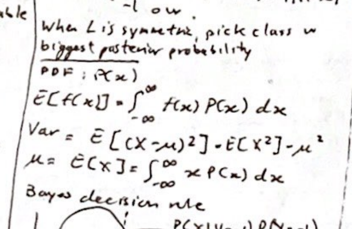
	small C	big C	
desire	maximize margin	keep most slack var zero or small, margin ↓	any point on slab/margin considered support vector
danger	underfitting	overfitting	
outliers	less sensitive	very sensitive	
boundary	more "flat"	more sinuous	

Nonlinear decision boundaries - nonlinear features that lift points into higher-d space.
- add features
- can lift points to ϕ space to make them linearly separable
- margin tends to get wider as degree increases
higher degree → overfitting
Edge detection: collect line orientations in local histograms, use histograms as features

Applications/Data
Model
Optimization ~~Algorithm~~ Problem
Optimization Algorithm

Types of Optimization Problems:
- find w that minimizes/maximizes a continuous objective fn $f(w)$
Finding global minimum is hard
For smooth f: unconstrained
- Gradient descent
  - blind
  - w linesearch
  - stochastic
- nonlinear conjugate gradient
- Newton's method
Nonsmooth f:
- Gradient descent
  - blind
  - direct line search
Linesearch finds a local minimum by solving an optimization problem in 1D

Constrained Optimization
- w that optimizes $f(w)$ subject to $g(w) = 0$, g is a smooth fn
→ Lagrange multipliers: transform constrained to unconstrained optimization.
Linear Program - $c \cdot w$, subject to $Aw \le b$
- linear objective fn + linear inequality constraints
Set of points that satisfy all constraints is a convex polytope called feasible region
- optimum $\in F$ is point furthest in direction c.
Active Constraints: constraints for which optimum achieves equality.
set of optimal pts always convex
Linearly separable iff feasible region isn't empty set
Hard: figuring out active constraints
- much more discrete than continuous unconstrained
Quadratic Program
- quadratic, convex objective fn + linear inequality constraints
$f(w) = w^T Q w + c^T w$, subject to $Aw \le b$
Q is PD ⇒ 1 local minimum
Convex Program
- Convex objective fn + convex inequality constraints

Posterior: $P(Y=y \mid X)$
Prior: $P(Y=y)$
Loss fn specifies badness for each incorrect prediction.
$R(r) = E[L(r(X), y)]$
Bayes Decision Rule: - assumes no loss for correct prediction
$r^*(x) = \begin{cases} 1 & \text{if } L(-1,1)P(Y=1 \mid X=x) > L(-1,1)P(Y=-1 \mid X=x) \\ -1 & \text{ow.} \end{cases}$
When L is symmetric, pick class w biggest posterior probability
PDF: $P(x)$
$E[f(x)] = \int_{-\infty}^{\infty} f(x) P(x)\, dx$
$Var = E[(X-\mu)^2] = E[X^2] - \mu^2$
$\mu = E[X] = \int_{-\infty}^{\infty} x P(x)\, dx$
Bayes decision rule


look up x, pick curve w highest probability

Classifiers
1) generative models e.g. LDA
assume points come from probability distr, different for each class
- fit distr params to class C points, giving $P(X \mid Y=c)$
- estimate $P(Y=c)$
- gives $P(Y \mid X)$
0-1 loss - pick class c that maximizes $P(Y=c \mid X=x)$
→ maximizes $P(X=x \mid Y=c) P(Y=c)$
2) discriminative models e.g. logistic regression model $P(Y \mid X)$ directly
3) Find decision boundary
- model $r(x)$ directly (no posterior)
1+2
$P(Y \mid X)$ tells you probability your guess is wrong
1
- can diagnose outliers: $P(X)$ very small
1
- hard to estimate distr accurately
Generative Model
- most popular when you have phenomena well approximated by normal distr + lots of sample points

Gaussian Discriminant Analysis models $P(Y=y \mid X)$ as logistic
- each class comes from normal distr
$X \sim N(\mu, \sigma^2)$: $P(x) = \frac{1}{\sqrt{2\pi}\,\sigma^d} \exp\left(-\frac{|x-\mu|^2}{2\sigma^2}\right)$
Bayes decision rule returns class C that maximizes $P(X=x \mid Y=C) P(Y=C)$
$Q_c(x) = \ln\left((\sqrt{2\pi})^d P(x) \pi_c\right) = \frac{-|x-\mu_c|^2}{2\sigma_c^2} - d \ln \sigma_c + \ln \pi_c$
QDA allows you to determine probability that your classification is correct
$P(Y=C \mid X=x) = s(Q_c(x) - Q_D(x))$, $s = \frac{1}{1+e^{-\gamma}}$
LDA:
- linear decision boundaries, less likely to overfit
Assume all Gaussians have same var σ
$Q_c(x) - Q_D(x) = \frac{(\mu_c - \mu_D) \cdot x}{\sigma^2} - \frac{|\mu_c|^2 - |\mu_D|^2}{2\sigma^2} + \ln \pi_c - \ln \pi_D$
$= w \cdot x + \alpha$. } decision boundary
Choose class that maximizes linear discriminant fn
$\frac{\mu_c \cdot x}{\sigma^2} - \frac{|\mu_c|^2}{2\sigma^2} + \ln \pi_c$

MLE to use GDA
- given $X_1, \dots, X_n$, find best fit Gaussian
$\mathcal{L}(\mu, \sigma; X_1, \dots, X_n) = P(X_1) \dots P(X_n)$
- maximize log likelihood.
$\ell(\mu, \sigma) = \ln P(X_1) + \dots + \ln P(X_n)$
$= \sum_{i=1}^{n} \left(\frac{-|X_i - \mu|^2}{2\sigma^2} - d \ln \sqrt{2\pi} - d \ln \sigma \right)$
$\nabla_\mu \ell = \sum_{i=1}^{n} \frac{X_i - \mu}{\sigma^2} = 0$
$\frac{\partial \ell}{\partial \sigma} = \sum_{i=1}^{n} \frac{|X_i - \mu|^2 - d\sigma^2}{\sigma^3} = 0$.
Use mean + variance of points in class C to estimate mean + variance of Gaussian for class C.
QDA: separate conditional mean + variance for each class C
$\hat{\pi}_c = \frac{n_c}{\sum_D n_D}$
LDA: one variance for all classes
↳ within class
→ use each point's distance from its class' mean
$\sigma^2 = \frac{1}{dn} \sum_c \sum_{i: y_i = c} |X_i - \hat{\mu}_c|^2$

$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} X_i$
$\hat{\sigma}^2 = \frac{1}{dn} \sum_{i=1}^{n} |X_i - \mu|^2$

Eigenvector $Av = \lambda v$
- v points in same direction after being multiplied by A
$A^k v = \lambda^k v$ $A^{-1} v = \frac{1}{\lambda} v$
Spectral Theorem: every real, symmetric n×n has real eigenvalues and n eigvec that are mutually orthogonal
- if 2 eigvec have same eigval, every linear combination of those eigvec are also an eigvec
Quadratic Form: shows how applying the matrix affects length of a vector.
$|Ax|^2 = x^T A^2 x$
~~Ellipsoid radii are the reciprocals of eigenvalues; matrix A^{-1} maps spheres to ellipsoids~~
bigger eigval ⇔ steeper hill ⇔ shorter ellipsoid radius.
A is diagonal ⇔ eigvec are coordinate axes ⇔ ellipsoids are axis aligned
Positive definite: $w^T B w > 0$ $\forall w \ne 0$ ⇔ $\lambda > 0$.
PSD: $w^T B w \ge 0$ $\forall w$ ⇔ $\lambda \ge 0$.
Indefinite: $\lambda > 0$, $\lambda < 0$.
Invertible: $\lambda \ne 0$.
+ eigval: curvature goes up
- eigval: curvature goes down
$A = V \Lambda V^T = \sum_{i=1}^{n} \lambda_i v_i v_i^T$
diagonal
↳ rotate to be axis aligned
$A^2 = V \Lambda V^T V \Lambda V^T = V \Lambda^2 V^T$
$M^{1/2} = A$
1) compute eigvec/val of M
2) take square roots of eigval
3) reassemble A w square roots as eigval, same eigvec
$\boxed{P(x)}_{\text{multivar}} = \frac{1}{(\sqrt{2\pi})^d \sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$, $d =$ # features
Σ: covar matrix
Σ^{-1}: precision matrix
$P(x) = n(q(x))$, $q(x) = (x-\mu)^T \Sigma^{-1} (x-\mu)$
$q(x)$ is squared distance from $\Sigma^{-1/2} x$ to $\Sigma^{-1/2} \mu$
$Cov(R, S) = E((R - E(R))(S - E(S))^T)$
$= E(RS^T) - \mu_R \mu_S^T$
R_i, R_j independent ⇒ $Cov(R_i, R_j) = 0$.
$Cov(R_i, R_j) = 0$ && multivar normal ⇒ R_i, R_j independent
All features pairwise independent ⇒ $Var(R)$ is diagonal
ow:
$Var(R) = \begin{bmatrix} Var(R_1) & Cov(R_1, R_2) & \dots & Cov(R_1, R_d) \\ Cov(R_2, R_1) & \ddots & & \vdots \\ \vdots & & \ddots & \\ Cov(R_d, R_1) & \dots & & Var(R_d) \end{bmatrix}$

$\Sigma = V \Gamma V^T$. eigval of Σ are variances along the eigvec, $\gamma_{11} = \sigma_1^2$
$\Sigma^{1/2} = V \Gamma^{1/2} V^T$: maps spheres to ellipsoids
- eigval are Gaussian widths / ellipsoid radii / standard deviations
Isocontours of multivariate normal distr are same as isocontours of quadratic form of Σ^{-1}.

Anisotropic Gaussian
QDA: $\hat{\Sigma}_c = \frac{1}{n_c} \sum_{i: y_i = c} (X_i - \hat{\mu}_c)(X_i - \hat{\mu}_c)^T$
LDA: $\hat{\Sigma} = \frac{1}{n} \sum_c \sum_{i: y_i = c} (X_i - \hat{\mu}_c)(X_i - \hat{\mu}_c)^T$
QDA: choosing C that maximizes $P(X=x \mid Y=c)\pi_c$ is equivalent to maximizing quadratic discriminant fn
$Q_c(x) = -\frac{1}{2}(x-\mu_c)^T \Sigma_c^{-1} (x-\mu_c) - \frac{1}{2} \ln |\Sigma_c| + \ln \pi_c$

For anisotropic: decision boundary could be hyperbola

* need to apply logistic fn to find decision boundary

LDA: Decision fn is linear, decision boundary is hyperplane.

Maximize linear discriminant fn:

$\mu_c^T \Sigma^{-1} x - \frac{1}{2}\mu_c^T \Sigma^{-1}\mu_c + \ln \pi_c$

For 2 classes:

LDA has $d+1$ params

QDA has $\frac{d(d+3)}{2}+1$ params

With features, LDA can give nonlinear boundaries, QDA nonquadratic

- LDA/QDA work well when data can only support simple decision boundaries such as linear/quadratic, bc Gaussians provide stable estimates

---

X: $n \times d$ design matrix of sample pts

Centering: subtracting μ^T from each row of X $\to \dot{X}$

$Var(R) = \frac{1}{n}\dot{X}^T\dot{X}$

decorrelating $\dot{X}$: applying $Z = \dot{X}V$, $Var(R) = V\Delta V^T$

↳ transforms sample points to eigvec coordinate system

$\to Var(Z) = \Delta$

sphering $\dot{X}$: applying transform $W = \dot{X}\,Var(R)^{-1/2}$

whitening X: centering + sphering, $X \to W$

W has covariance matrix I

Whitening: puts features on equal basis

Regression

- given point x, predict a numerical value

Regression Fns: linear: $h(x; w, \alpha) = w \cdot x + \alpha$

polynomial (equivalent to linear w poly features)

logistic $h(x; w, \alpha) = s(w \cdot x + \alpha)$

Loss Fns: z = prediction $h(x)$, y = true val

Squared Error: $L(z,y) = (z-y)^2$

Absolute Error: $L(z,y) = |z-y|$

Cross Entropy: $L(z,y) = -y \ln z - (1-y)\ln(1-z)$

Cost Fns: (minimize)

$J(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(X_i), Y_i)$ mean loss

$J(h) = \max_{N} L(h(X_i), Y_i)$ max loss

$J(h) = \sum_{i=1}^{n} w_i L(h(X_i), Y_i)$ weighted sum

$J(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(X_i), Y_i) + \lambda\|w\|^2$ L2 regularized

$J(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(X_i), Y_i) + \lambda\|w\|$ L_1 regularized

---

Least Squares Linear Regression

Find w that minimizes $|Xw-y|^2 = RSS(w)$

$= \sum_{i=1}^{n}(X_i \cdot w + \alpha - y_i)^2$

Soln: $X^TXw = X^Ty$

Let $X^+ = (X^TX)^{-1}X^T$

$\hat{y}_i = w \cdot X_i \Rightarrow \hat{y} = Xw = XX^+y = Hy$

Xw is subspace of $\mathbb{R}^n$ spanned by columns $X \in \mathbb{R}^{n \times d}$

at most $d+1$ dim

$y \in n$ dim space

Minimizing $|\hat{y}-y|^2$ finds $\hat{y}$ nearest y on subspace = orthogonal projection.

Advantages:

- easy to compute; just solve a linear system
- unique, stable soln.

Disadvantages:

- sensitive to outliers
- fails if X^TX is singular

Logistic Regression - discriminative

- fits probabilities in range (0,1)
- used for classification

no closed form soln

Find w that minimizes:

$J = -\sum_{i=1}^{n}\left(y_i \ln s(X_i \cdot w) + (1-y_i)\ln(1-s(X_i \cdot w))\right)$

$s'(\gamma) = s(\gamma)(1-s(\gamma))$

$\nabla_w J = -X^T(y - s(Xw))$

Gradient Descent Rule: $w \leftarrow w + \epsilon X^T(y - s(Xw))$

SGD: $w \leftarrow w + \epsilon(y_i - s(X_i \cdot w))X_i$

Least Squares Polynomial Regression

- replace each X_i with vector

$\Phi(X_i) = [X_{i1}^2 \;\; X_{i1}X_{i2} \;\; X_{i2}^2 \;\; X_{i1} \;\; X_{i2} \;\; 1]^T$

Weighted Least Squares Regression

- assign trusted sample points a higher weight ω

Greater $\omega_i \to$ work harder to minimize $|\hat{y}_i - y_i|^2$

Find w that minimizes

$(Xw-y)^T\Omega(Xw-y)$

$= \sum_{i=1}^{n}\omega_i(X_i \cdot w - y_i)^2$

Solve for w in normal eqn:

$X^T\Omega Xw = X^T\Omega y$

Newton's Method - iteratively reweighted least squares

- iterative optimization method for smooth fn $J(w)$
- faster than gradient descent

Idea: at point v.

Approximate $J(w)$ near v by quadratic fn. Jump to critical pt.

pick starting point w

repeat till convergence

$e \leftarrow (\nabla^2 J(w))^{-1}e = -\nabla J(w)$

$w \leftarrow w + e$

- The closer J is to quadratic, the faster Newton's converges.
- doesn't know difference btwn optima, must start close enough to soln.

Advantages

- tries to find right step length to reach min
- tries to choose better descent direction than just steepest descent

Disadvantages

- computing Hessian is expensive
- doesn't work for nonsmooth fns

Newton's for logistic regression.

$s_i = s(X_i \cdot w)$

$\nabla_w J = -X^T(y - s)$

$\nabla_w^2 J(w) = X^T\Omega X$, $\Omega = \begin{bmatrix} s_1(1-s_1) & & 0 \\ & \ddots & \\ 0 & & s_n(1-s_n)\end{bmatrix}$

$e \leftarrow (X^T\Omega X)e = X^T(y-s)$

Ω prioritizes points with $s_i \approx 0.5$, tunes out pts near 0/1

$\Rightarrow$ sample pts near decision boundary have biggest effect on iterations, make big contribution to logistic fit.

If n very large, save time by using a random subsample of pts per iteration, increase sample size as you go.

LDA vs Logistic Regression

LDA:

- stable for well-separated classes
- more accurate when classes nearly normal

Logistic Regression

- more emphasis on decision boundary (LDA gives equal weight)
- less sensitive to most outliers
- easy treatment of partial membership
  - LDA pts all or nothing
- more robust on non Gaussians

ROC curve evaluates classifier after it's trained

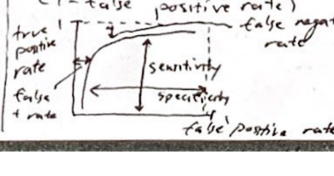
- shows rate of false positive vs ~~false negative~~ true positive

X-axis: false positive rate

Y-axis: true positive rate (sensitivity)

false-negative rate: vertical distance from curve to top (1-sensitivity)

true negative rate: horizontal distance from curve to right "specificity" (1 - false positive rate)



Upper Right Corner: always classify +

Lower Left: always classify -

diagonal: random classifiers

Classifiers' effectiveness = area under curve

Always right = 1.

Random = 1/2.

Model of Reality:

- sample pts from unknown prob distr
- y-values are sum of unknown, non-random fn + random noise

$\forall X_i,\ y_i = f(X_i) + \epsilon_i,\ \epsilon_i \sim D',\ \sim\mu = 0$

Risk for hypothesis h is expected loss

$R(h) = E[L]$

Empirical Distr: discrete uniform dist over sample pts.

Empirical Risk: expected loss under empirical distr

$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(X_i), y_i)$

~ Max likelihood explains where logistic loss fn comes from

---

2 sources of error in h

bias: error due to inability of h to fit f perfectly

variance: error due to fitting random noise in data

$R(h) = \underbrace{(E[h(z)] - f(z))^2}_{\text{bias}^2} + \underbrace{Var(h(z))}_{\text{variance}} + \underbrace{Var(\epsilon)}_{\text{irreducible error}}$

Underfitting: too much bias

Overfitting: too much variance

- variance $\to 0$ as $n \to \infty$
- Adding good feature reduces bias; adding bad feature rarely ~~??~~ increases
- adding a feature increases variance
- noise in test set affects only $Var(\epsilon)$
- noise in training affects bias + $Var(h)$

Ridge Regression

Find w that minimizes

$|Xw-y|^2 + \lambda|w'|_2^2$,

w' is w w α replaced by 0.

- regularization terms guarantees PD $\Rightarrow$ unique soln
- many minima = ill-posed
- regularization reduces variance

Ideally: features "normalized" to have same variance

↳ $(X^TX + \lambda I')w = X^Ty$

Subset Selection

- all features increase variance, but not all features reduce bias
- identify poorly predictive features, ignore them
- less overfitting, smaller test error

Alg: try all $2^d - 1$ nonempty subsets of features

→ train one classifier per subset

choose best classifier by cross-validation.

Heuristic 1: Forward stepwise selection

- repeatedly add best feature until validation errors start increasing $O(d^2)$

Heuristic 2: Backward stepwise selection

- start w all d features, remove feature whose removal gives best reduction in validation error $O(d^2)$

3: try to remove features w small weights

Lasso

- often naturally sets some weights to zero

Find w that minimizes $|Xw-y|^2 + \lambda\|w'\|_1$

$\|w'\|_1 = \sum_{i=1}^{d}|w_i|$

$\lambda\uparrow$, more + more weights go to 0.

$\nabla f(x) = \frac{\partial f}{\partial x}^T$

$(A^Tx)_i = \sum_{j=1} A_{ij}^T x_j$

$(Ax)_i = \sum_{j=1}^{d} A_{ij}X_j$

$f(x+\Delta) = f(x) + \frac{\partial f}{\partial x}\Delta + o(\|\Delta\|)$

Gradient $= \frac{\partial f}{\partial x}^T$

Posterior: $P(Y=y|x)$

$\theta_{MLE} = \arg\max_\theta P(X|\theta)$

$= \arg\max_\theta \prod_{i=1}^{n} P(x_i|\theta)$

$\theta_{MAP} = \arg\max_\theta P(\theta|X)$

$= \arg\max_\theta P(X|\theta)P(\theta)$

$= \arg\max_\theta \prod_{i=1}^{n} P(x_i|\theta)P(\theta)$

Covariance matrix must be PSD

Gaussian prior = L2 reg.

Laplace prior = L1 reg

$Cov = \frac{Corr}{\sigma_{x_1}\sigma_{x_2}}$

determinant = product of eigval

PDF of univariate Gaussian:

$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$

Bayes:

$P(Y|X) = \frac{P(X|Y)\cdot P(Y)}{P(X)}$

Maximum a posteriori

- maximizing posterior $P(w|X,y)$

length of ellipsoid axes for multivariate Gaussian are $\sqrt{\lambda_i}$ of Σ

- train on more data to improve training accuracy, less data to improve test
- Bayes risk is 0 when class distr don't overlap, prior for one class = 1
- add λI to covariance matrix causes isocontours to be more spherical

$s(0) = 1/2$ is decision boundary

No covariance terms: isocontour is axis aligned

- space between contours is $\sqrt{\lambda_i}$ of Σ
- higher SD $\Rightarrow$ larger gaps between significant isocontours.
- eigenvectors form orthonormal basis

* tell if something is convex by whether its Hessian is positive definite.

$A^TA = |A|^2$

- LDA maximizes between class variance over within-class variance

Shrinkage: minimize $|w|$

Support vectors:

- necessary to compute $f(x)$ in SVM
- have nonzero weight

Hessian: symmetric

$\frac{\partial^2 f}{\partial x_1^2} \quad \frac{\partial^2 f}{\partial x_1 \partial x_2}$

$\frac{\partial^2 f}{\partial x_2 \partial x_1} \quad \frac{\partial^2 f}{\partial x_2^2}$

Equivalent Objective Fn:

$$\min_{y} \sum_{i=1}^{K} \frac{1}{n_i} \sum_{y_j = i} \sum_{y_a = i} |X_j - X_a|^2$$

→ average squared distance from point to all other pts in cluster

K-Medoids clustering: only requires pairwise distances!
- generalizes k-means beyond Euclidean distance (means not good)
- specify distance fn $d(x, y)$ between points
- replace mean w medoid, sample pt that minimizes total distance to other pts in same cluster

medoid always a sample pt.

Hierarchical Clustering: only requires pairwise distances
- creates a tree, every subtree is a cluster
- bottom up; agglomerative clustering:
  - start with each point a cluster, repeatedly fuse pairs
- top-down; divisive clustering
  - start with all points in one cluster; repeatedly split it

Distance fns for clusters A, B
- complete linkage: max distance btwn any point in A, any pt in B
  $\max\{d(w, x) : w \in A, x \in B\}$
- single linkage: min distance
  $\min\{d(w, x) : w \in A, x \in B\}$
- average linkage: average over all pairs $a \in A$, $b \in B$
  $d(A, B) = \frac{1}{|A||B|} \sum_{w \in A} \sum_{x \in B} d(w, x)$
- centroid linkage: distance btwn means
  $d(\mu_A, \mu_B)$

Greedy Agglomerative: repeatedly fuse two clusters that minimize $d(A, B)$. $O(n^3)$

Dendrogram: Illustration of cluster hierarchy in which vertical axis encodes all linkage distances

Spectral Graph Clustering

Input: weighted undirected graph $G = (V, E)$, every pair of vertices has weight

edge weights = similarity measure

big weight means two vertices want to be in same cluster

Goal: cut G into 2 pieces G_i of similar size

e.g. minimize sparsity $\frac{Cut(G_1, G_2)}{Mass(G_1) Mass(G_2)}$.

$Cut(G_1, G_2)$ = total weight of cut edges

$Mass(G_1)$ = # of vertices in G_1

Denominator penalizes imbalanced cuts

$n = |V|$, $y_i = \begin{cases} 1, & \text{vertex } i \in G_1 \\ -1, & \text{vertex } i \in G_2 \end{cases}$

$w_{ij} \frac{(y_i - y_j)^2}{4} = \begin{cases} w_{ij} & (i,j) \text{ is cut} \\ 0 & (i,j) \text{ isn't cut} \end{cases}$

$$Cut(G_1, G_2) = \sum_{(i,j) \in E} w_{ij} \frac{(y_i - y_j)^2}{4} = \frac{y^T L y}{4},$$

$L_{ij} = \begin{cases} -w_{ij}, & i \neq j \quad \text{- off diagonal} \\ \sum_{k \neq i} w_{ik}, & i = j \quad \text{- sum of edge weights that go into vertex } i \end{cases}$

L is symmetric, n×n: matrix rep of G

Bisection: $1^T y = 0$

Find y that minimizes $y^T L y$ subject to $\forall i$, $y_i = 1$ or $y_i = -1$ and $1^T y = 0$ - balance

Let's relax binary constraint to allow fractional vertices

Now: y must lie on hypersphere of radius $\sqrt{n}$

- minimize $y^T L y$ on and $1^T y = 0$.

⇒ minimize $\frac{y^T L y}{y^T y}$ = Rayleigh quotient

The y that minimizes this is eigvec w smallest eigval.

λ_2 = second smallest eigval of L

eigvec v_2 = Fiedler vector

Spectral partitioning alg:
- compute Fiedler vector v_2 of L
  - sort components of v_2
  - try n−1 cuts between successive components. choose min sparsity cut

Vertex Masses:
- assign masses to vertices
- let M be diagonal matrix with vertex masses on diagonal
- new balance constraint: $1^T M y = 0$.
- revise sphere constraint to ellipsoid
  $y^T M y = Mass(G) = \sum M_{ii}$
- want Fiedler vector of generalized eigensystem $Lv = \lambda M v$

Greedy Divisive Clustering
- partition G into 2 subgraphs; recursively cluster them

Normalized Cut:
- vertex i's mass $M_{ii} = L_{ii}$ - sum of edge weights adjoining vertex i
- Image Segmentation
  $w_{ij} = \exp\left(-\frac{|w_i - w_j|^2}{\alpha} - \frac{|b_i - b_j|^2}{\beta}\right)$
  location w_i, brightness b_i.

⇒ closer pixels with similar color have greater weight.
- each pixel is vertex in graph

Clustering w Multiple Eigenvectors
- for k clusters, compute first k eigvec of $Lv = \lambda M v$

$$V = \begin{bmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_k \\ | & | & & | \end{bmatrix} = \begin{bmatrix} — V_1 — \\ \vdots \\ — V_n — \end{bmatrix}$$

- Row V_i is spectral vector for vertex i
- cluster vertices together if spectral vectors point in similar directions
- use k-means to cluster together vectors separated by small angles

Latent Factor Analysis

X_{ij} = occurrences of term j in doc i

SVD $X = UDV^T = \sum_{i=1} \delta_i u_i v_i^T$

diagonals greatest to smallest

- greatest δ_i, v_i lists terms in a genre/cluster of documents/words
- u_i lists documents in a genre with similar terms, i.e. mark similar documents.

δ_i is a genre.
- genre is hidden: nothing explicitly states it.
- LFA = clustering that allows clusters to overlap.

Applications:
- Fuzzy Search (related words)
- Denoising
- matrix compression
- collaborative filtering: fill in unknown values

Geometry of High Dimensional spaces
- shell between spheres of radii r, $r - \epsilon$
- random points from uniform dist in ball: nearly all in outer shell

from Gaussian: nearly all in some shell
- variations in distance less extreme in higher dimensional space

→ clustering less effective in higher dimensions

Random Projection: dimensionality reduction.
helps w filling in unknown values
- cheap alternative to PCA: don't have to compute eigvec.
- random subspace $S \subset \mathbb{R}^d$ of dim k, random ϵ, δ
  $k = \frac{2 \ln(1/\delta)}{\epsilon^2/2 - \epsilon^3/3}$,

For any pt q, let $\hat{q}$ be $\sqrt{\frac{d}{k}}$ times orthogonal projection of q onto S. ← preserves distance

With probability $\geq 1 - 2\delta$, distance between two points after projecting is within some range of original distance

For $q, w \in \mathbb{R}^d$, $(1-\epsilon)|qw|^2 \leq |\hat{q}\hat{w}|^2 \leq (1+\epsilon)|qw|^2$

Boosting

ADABOOST: adaptive boosting
- ensemble method
- train multiple learners on weighted sample points
  - use different weights for each learner
  - increases weights of misclassified sample points
  - gives bigger votes to more accurate learners
- weight for sample point grows according to how many of $G_1, \ldots, G_T$ misclassified it.

Metalearner $M(z) = \sum_{t=1}^{T} \beta_t G_t(z)$

M is continuous

Risk = average loss
$= \frac{1}{n} \sum_{i=1}^{n} L(M(X_i), y_i)$

Metalearner uses exp. loss fn

$L(\rho, \ell) = e^{-\rho\ell} = \begin{cases} e^{-\rho} & \ell = +1 \\ e^{\rho} & \ell = -1 \end{cases}$

exp loss fn pushes hard against badly misclassified points

Optimal learner G_T minimizes risk by minimizing sum of weights over all misclassified pts X_i.

$$w_i^{(T+1)} = w_i^{(T)} e^{-\beta_T y_i G_T(X_i)} = \begin{cases} w_i^{(T)} e^{-\beta_T}, & y_i = G_T(X_i) \\ w_i^{(T)} e^{\beta_T}, & y_i \neq G_T(X_i) \end{cases}$$

$\beta_T = \frac{1}{2} \ln\left(\frac{1 - err_T}{err_T}\right)$,

err_T = G_T's weighted error rate
$= \frac{\sum_{y_i \neq G_T(X_i)} w_i^{(T)}}{\sum_{i=1}^{n} w_i^{(T)}}$

→ more accurate learners get bigger votes in metalearner

Adaboost:
1. initialize weights $w_i \leftarrow \frac{1}{n}$
2. for $t \in 1$ to T
   a. Train G_t w weights w_t
   b. Compute err_t, β_t
   c. reweight points
3. return metalearner $\text{sign}\left(\sum_{t=1}^{T} \beta_t G_t(z)\right)$

Boost decision trees b.c...
- fast
- no hyperparameter search
- easy to make tree beat 55% training accuracy
- easy bias variance control.
- Adaboost + short trees is form of subset selection
- linear decision boundaries don't boost well.

posterior is approximated

Nearest Neighbors
- given query point q, find k sample pts nearest q
- distance metric of your choice
- regression: return average label of k points
- classification: return class w most votes from k pts OR return histogram of class probabilities

Exhaustive KNN
- maintain max heap w k shortest distances seen so far.

Query Time: $O(nd + n \log k)$
n points, d dims, k shortest distances

Preprocess training points to get sublinear query time
- 2-5 dimensions: Voronoi diagrams
- medium: k-d trees
- large: exhaustive w dim red.

Voronoi Diagram
- each point has cell which contains space closest to that point.

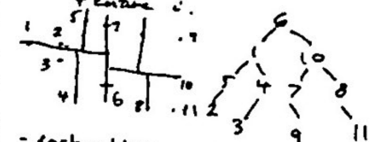
diagram = set of X's Voronoi cells.

Size (# of vertices) $\in O(n^{d/2})$

2D: $O(n \log n)$ to compute V.d. and trapezoidal map for pt location. $O(\log n)$ query time.

dD: use binary space partition tree for pt location.
- really only supports 1 nearest neighbor

k-d Trees
- decision tree...
- choose splitting feature w greatest width
  → maximize distance from leftmost value to rightmost
- splitting value: median point for feature i.



- each subtree represents particular box in space.

Query Algorithm maintains:
- nearest neighbor found so far - distance ↓ through alg
- binary heap of unexplored subtrees, keyed by distance from q. - dist. grows ↑

Alg:

Q ← heap containing root node w key zero

r ← ∞ . distance to nearest neighbor so far

while Q not empty and $(1+\epsilon) \cdot \text{minkey}(Q) < r$
  B ← removemin(Q)
  w ← B's sample point
  r ← min{r, dist(q, w)}
  B', B'' ← child boxes of B
  if $(1+\epsilon) \cdot \text{dist}(q, B') < r$, insert(Q, B', dist)
  " B''

return pt that determined r.

Approximate nearest neighbor speeds up alg. could be slow in high D. k-d tree

$\frac{1}{n} E(X) = E(\frac{X}{n})$

$Var(\frac{Y}{n}) = \frac{1}{n^2} Var(Y)$

$Cov(X, Y) = corr(X, Y) \cdot \sqrt{Var(X) \cdot Var(Y)}$

calculating distance in p polynomial space takes d^p, points in $\mathbb{R}^d$

features get raised to p polynomial ones, number features ^ p.

Valid Kernel:
- $k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$
- $K = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & k(x_i, x_j) & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{bmatrix}$ $\boxed{\text{PSD}}$

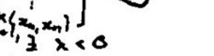
∴ if K isn't a valid kernel, ∃ $\lambda < 0$

$\frac{d}{dx} \|Ax - \lambda x\|_2^2 = -2x^T(Ax - \lambda x)$

- left multiplying by orthogonal matrix preserves length of vector

- cov matrix: $\hat{\Sigma} = \frac{1}{n} X^T X$

Convex Concave



- nonlinear
- 2 node types:
  1) internal nodes test feature values + branch accordingly
  2) leaf nodes specify class
- if all sample pts in a node are not of same class, we need to split
- choose best splitting feature j and splitting value β

$$S_\ell = \{i : X_{ij} < \beta\},\ S_r = \{i : X_{ij} \geq \beta\}$$

Try all features/all splits in a feature

$J(S) :=$ cost of S

Goal: minimize $J(S_\ell) + J(S_r)$ or

$$\frac{|S_\ell| J(S_\ell) + |S_r| J(S_r)}{|S_\ell| + |S_r|}$$

Cost: Measure entropy

let $P(Y = C) = P_c$

Surprise of being class C = $-\log_2 P_c$

Entropy:

$$H(S) = -\sum_c P_c \log_2 P_c,$$

P_c = proportion of points in S that are in class C.

↳ expected number of bits of info we need to transmit to identify class of a sample point chosen uniformly at random

$$H_{after} = \frac{|S_\ell| H(S_\ell) + |S_r| H(S_r)}{|S_\ell| + |S_r|}$$

Choose split that maximizes info gain

$H(S) - H_{after}$ = minimizing H_{after}

Key: cost function is concave

Training, quantitative features: $O(n'd)$ splits, d features, n' points in node

Total run time: $O(nd \cdot \text{depth})$

Decision Trees can use multivariate splits, i.e. consider more than one feature to split on

↳ better classifiers at cost of worse interpretability or speed

Regression: $J(S) = \frac{1}{|S|} \sum_{i \in S} (Y_i - \bar{Y})^2$,

$\bar{Y}$ = mean(Y_i) for points in S.

Can stop building tree early:

- next split doesn't reduce entropy enough
- most of node's points have same class
- node contains few sample pts
- cell's edges too tiny
- depth too great
- use cross validation

Pruning:

- greedily remove each split whose removal improves cross validation performance

Decision trees have high variance

- take average of output of
  - different learning algs
  - same learning alg on many training sets
  - bagging: same learning alg on many random subsamples of one training set
  - random forests: randomized decision trees on random subsamples
- Use learners with low bias - averaging reduces variance.

Bagging:

- Given n-point training sample, generate random subsample of size n' by sampling w replacement
- build learner, then repeat sampling.

Random Forests better approximate posteriors

- at each tree node, take random sample of m features, choose best split from m features

Disadvantage: lose interpretability / inference

Kernels don't have to explicitly compute features in high-d space

- Suppose $w = X^T a = \sum_{i=1}^n a_i X_i$ for some $a \in \mathbb{R}^n$
- optimize n dual weights a.

Kernel Ridge Regression:

- center X and y so means are zero.

$$(X^T X + \lambda I) w = X^T y$$

$$\Rightarrow w = X^T a,\quad a = \frac{1}{\lambda}(y - Xw)$$

$$\Rightarrow a = (XX^T + \lambda I)^{-1} y$$

decision boundary / probabilities are

$$\min_a |XX^T a - y|^2 + \lambda |X^T a|^2$$

Regression fn:

$$h(z) = w^T z = a^T X z = \sum_{i=1}^n a_i (X_i^T z)$$

$k(x, z) = x^T z$:= Kernel fn

$K = XX^T$ = $n \times n$ Kernel matrix,

$K_{ij} = k(X_i, X_j)$

Dual ridge reg alg:

$\forall i,j,\ K_{ij} \leftarrow k(X_i, X_j)$

Solve $(K + \lambda I) a = y$ for a.

for each test pt z

$$h(z) = \sum_{i=1}^n a_i k(X_i, z)$$

Polynomial kernel of degree p:

$$k(x, z) = (x^T z + 1)^p = \Phi(x)^T \Phi(z),\ \Phi(x)$$

contains every monomial in x of degree $0 \ldots p$

Win: we can compute $\Phi(x)^T \Phi(z)$ in $O(d)$ time instead of $O(d^p)$

- don't compute $\Phi(x), \Phi(z)$ directly
- Dualize $w = \Phi(X)^T a$, do everything in terms of a.

$$\Phi(X_i) \cdot w = [\Phi(X) w]_i = [\Phi(X) \Phi(X)^T a]_i = (Ka)_i$$

Gaussian Kernel:

$\exists\ \Phi(x)$ st

$$k(x, z) = \exp\left(-\frac{|x - z|^2}{2\sigma^2}\right),\quad O(d)$$

- think of Kernel k as measure of how similar points are

$h(z) = \sum_{j=1}^n a_j k(X_j, z)$ is linear combo of Gaussians centered at sample pts

→ each sample pt votes on classification of surrounding area

Popular!

- Oscillates less than polynomials - wide σ has less oscillation
- $k(x, z)$ interpreted as similarity measure
- Sample points vote for value of z, closer points get weightier vote.

Standard Kernel $x \cdot z$ assigns more weight to vectors that point in same direction. Direction vs Proximity

σ trades off bias vs variance

larger σ → wider Gaussians, smoother

↳ more bias, less variance.

Neural Nets

Logistic fn: $s(\gamma) = \frac{1}{1 + e^{-\gamma}}$

Training: SGD / BGD

$$J(h) = \frac{1}{n} \sum_{i=1}^n L(h(X_i), Y_i)$$

↳ average loss over predictions

- start w as random weights to differentiate and get better local minima.

Backprop:

$$\frac{\partial}{\partial \tau} L(z_1(\tau), z_2(\tau)) = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial \tau} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial \tau}$$

NN Variations

- regression: usually linear output units - omit sigmoid fn

Classification - for $k \geq 3$ classes, use softmax fn:

$$z_j(t) = \frac{e^{t_j}}{\sum_{i=1}^k e^{t_i}}$$

probability of input belonging to class j

Unit Saturation / Vanishing Gradient

- when unit output is close to 0 or 1 for most training points, $s' = s(1-s) \approx 0$.

Mitigation:

1) initial weight of edge into unit w fan-in η random w mean zero, SD $1/\sqrt{\eta}$

2) Set target values to 0.15 & 0.85 instead of 0 or 1.

$|X^T Z| / N \sim N$

small constant to s'

- hack so unit can't get stuck

4) Cross Entropy Loss instead of squared error

$$L(z, y) = -\sum_i \left(Y_i \ln z_i + (1 - Y_i) \ln(1 - z_i)\right)$$

$$\frac{\partial L}{\partial z_j} = \frac{z_j - y_j}{z_j(1 - z_j)},\quad z = \text{prediction},\ y = \text{truth}$$

- Used only for sigmoid + softmax outputs (classification)
- Regression: use linear outputs which don't saturate

5) replace sigmoids w ReLUs: $r(\gamma) = \max\{0, \gamma\}$

- vulnerable to exploding gradient problem because output is arbitrarily large; problem in deep / recurrent nw

Heuristics for Faster Training

SGD: faster than batch on large, redundant data sets. takes loss at one sample point

Normalizing Data

- Center each feature so mean is zero.
  - makes it easier for hidden units to get into good operating region of sigmoid or ReLU
- scale each feature so variance ≈ 1
  - makes each objective fn better conditioned so GD converges faster

Centering hidden units

- replace sigmoids with tanh $\gamma = \frac{e^{\gamma} - e^{-\gamma}}{e^{\gamma} + e^{-\gamma}} = 2s(2\gamma) - 1$
  - ↳ ranges from -1 to 1
- Different learning rate for each layer of weights
  - earlier layers have smaller gradients, need larger learning rate
- Emphasizing schemes (uncommon samples)
- Second order optimization
  - Hessian too large + expensive to compute
  - nonlinear conjugate gradient: works well for small nets + small data + regression
  - stochastic Levenberg Marquardt
    - approximates diagonal Hessian
- acceleration schemes: RMSProp, Adam, AMSGrad

Avoiding Bad Local Minima

- SGD
- momentum

$$w \leftarrow w + \Delta w$$

$$\Delta w \leftarrow -\epsilon \nabla J(w) + \beta \Delta w$$

Avoiding Overfitting

- NN ensemble. random initial weights + bagging
- L2 regularization (weight decay)
  - add $\lambda |w'|^2$ to cost fn
- Dropout
- Fewer hidden units
  - too few, can't learn well
  - too many, overfit

ConvNets - late layers: higher level concepts

1. Local Connectivity: Hidden unit in early layer connects only to small patch of units in prev layer
2. Shared Weights: groups of hidden units share same set of input weights, called mask/filter/kernel

Hidden units = features we learn

Goal of Unsupervised Learning:

- discover structure in data

PCA: reduce dimension before applying supervised learning method

Given points in $\mathbb{R}^d$, find k directions that capture most of variation

Orthogonal Projection of point x onto w = $(x \cdot w) w$ || $\frac{x \cdot w}{|w|^2} w$

- Want to project onto subspace spanned by orthonormal vectors.

$$\tilde{x} = \sum_{i=1}^k (x \cdot v_i) v_i,\quad v_1, \ldots, v_k \text{ are orthonormal}$$

X: $n \times d$ design matrix

→ $X^T X$ is PSD, $d \times d$

$0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_d$ eigval

$v_1, v_2, \ldots, v_d$ = corr eigvec

Principal directions are these eigvec

- Center X
- compute unit eigvec/val of $X^T X$
- choose k = dim of subspace we're projecting down to.
- pick k largest eigvec
- compute $x \cdot v_i$ of training/test data in PC space

$$\%\text{ of variability} = \frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^n \lambda_i}$$

- want to retain as much variability as possible

PCA Deriv. 2.

- find direction w that maximizes sample variance of projected data

$$\text{Max Var}(\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_n) = \frac{1}{n} \underbrace{\frac{w^T X^T X w}{w^T w}}$$

Rayleigh quotient of $X^T X$, w.

If w is eigvec v_i, Rayleigh quotient = λ_i of $X^T X$

Deriv 3: minimize proj. error

i.e. minimize error in direction orthogonal to principal component direction

- minimizing projection error = maximizing variance

Singular Value Decomposition if $n \geq d$

$$X = UDV^T = \sum_{i=1}^d \delta_i u_i v_i^T$$

$n \times d$, $n \times d$, $d \times d$, $d \times d$

$$U^T U = I,\quad V^T V = I$$

Orthonormal u_i's are left singular vectors of

orthonormal v_i's are right singular vectors of

- diagonal entries $\delta_1, \ldots, \delta_d$ are nonnegative singular values of X.

v_i is eigvec of $X^T X$ w eigval δ_i^2

u_i is eigvec of XX^T, $XX^T X$

- we need eigvec of $X^T X$ for PCA!

- can find k greatest singular values + corresponding vectors in $O(ndk)$ time

- row i of UD gives coordinates of sample point X_i in principal components space.

K-Means Clustering — increase K: bias ↑, boundary smoother

Goal: Partition n points into K disjoint clusters

- assign each input point X_i a cluster label
- cluster i's mean is $\mu_i = \frac{1}{n_i} \sum_{y_j = i}$

Find y that minimizes

$$\sum_{i=1}^k \sum_{y_j = i} |X_j - \mu_i|$$

(Sum of squared distances from points to cluster means)

NP-hard: $O(nk^n)$, n points, k clusters

1) y_j's are fixed; update μ_i's

2) μ_i's are fixed; update y_j's

- both steps decrease objective fn unless they change nothing

Starting K-means...

Forgy: choose K random sample pts to be initial μ_i's; go to 2.

Random partition: randomly assign each sample pt to a cluster; go to

K-means++: like Forgy, but center is chosen w preference for points far from previous centers.