01/25/2016

Classifiers

- You are given a set of n samples, each with d features.
- Some samples belong to a certain class $\mathcal{O}$; some do not.
- Example: sample are bank loans, features are income and age (d=2). Some are in class defaulted; some are not. Goal: Predict whether future borrowers will default based on their income and age.
- Represent each sample as a point in a d-dimensional space, called a feature vector (aka predictors, independent variables).
- Decision boundary: the boundary chosen by our classifier to separate $\mathcal{O}$ from not $\mathcal{O}$.
- Some (not all) classifiers work by computing a predictor function: A function $f(x)$ that maps sample point x to a scalar such that,

$$f(x) > 0 \; if \; x \;\in class \; \mathcal{O}$$
$$f(x) \leq 0 \; if \; x \;\notin \; class \; \mathcal{O}$$

  (aka decision function, or discriminant function).
- For these classifiers, the decision boundary is,

$$\{x \in \mathbb{R}^d : f(x) = 0\}$$

  That is the set of all points where the prediction function is zero. Usually this set is a $(d-1)$-dimensional surface in $\mathbb{R}^d$.
- $\{x : f(x) = 0\}$ is also called an isosurface (aka isocontours) of f for the isovalue 0.
- Linear classifier: The decision boundary is a hyperplane. Usually uses a linear predictor function.
- Overfitting: when sinuous (having many curves and turns) decision boundary fits sample data so well that it doesn't classify future (test set) items well.

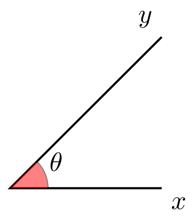
Math Review

- Vectors:

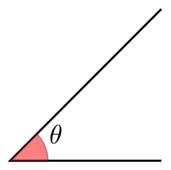
$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix}^T$$

  Think of x as a point in $\mathbb{R}^d$.
- Conventions (often, but not always):
  - Uppercase roman = matrix.
  - Lowercase roman = vector.
  - Greek = scalar.
  - Other scalars: n = number of samples, d = number of features or dimension of sample, i, j, and k = indices.
  - Functions (often scalars): $f()$, $s()$, etc.
- Inner products (aka dot products)
  - $x \cdot y = x_1y_1 + x_2y_2 + \cdots + x_dy_d$
  - Also written x^Ty.
  - Clearly, $f(x) = w \cdot x + \alpha$ is a linear function in x.

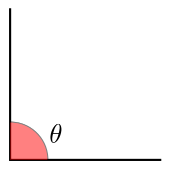
- Eucledian norms: $||x|| = \sqrt{x \cdot x} = \sqrt{x_1^2 + x_2^2 + \cdots + x_d^2}$
  - $||x||$ is the length (aka Eucledian length) of a vector x.
  - Given a vector x, $\frac{x}{||x||}$ is a unit vector (length 1).
  - "Normalize" a vector x: replace x with $\frac{x}{||x||}$.
- Use dot product to compute angles:



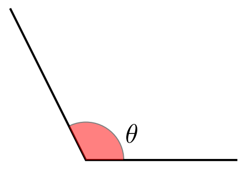
$$cos\ \theta = \frac{x \cdot y}{||x|| \cdot ||y||} = \frac{x}{||x||} \cdot \frac{y}{||y||}$$



$$acute,\ cos\ \theta > 0$$



$$right,\ cos\ \theta = 0$$



$$obtuse,\ cos\ \theta < 0$$

- Given a linear predictor function $f(x) = w \cdot x + \alpha$, decision boundary is

$$H = \{x : w \cdot x = -\alpha\}$$

  - The set H is called a hyperplane (A line in 2D, a plane in 3D).
- Theorem: Let $\vec{xy}$ be a vector that lies in H. Then $w \cdot (y - x) = 0$.
  Proof: x and y lie on the hyperplane H. $\therefore w \cdot (y - x) = -\alpha - (-\alpha) = 0$.
- w is called the normal vector of H. w is normal (perpendicular) or orthogonal to H.
- If w is a unit vector, $w \cdot x + \alpha$ is called the signed distance from x to H i.e. its distance, but positive on one side of H; negative on the other. Moreover the distance from H to the origin is α. Hence $\alpha = 0$ if and only if H contains the origin.
- The coefficients in w, plus α are called weights or regression coefficients. Goal of many Machine Learning algorithms is to find what the weights should be.
- The input data is linearly separable if $\exists$ a hyperplane that separates all samples $\in \mathcal{O}$ from all samples $\notin \mathcal{O}$.

Perceptron algorithm

- (Frank Rosenblatt, 1957) Slow, but correct for linearly separable samples. Uses a numerical optimization algorithm: gradient descent.
- Consider n sample vectors $x_1, x_2, \ldots x_n$.

- For each sample, let

$$y_i = \begin{cases} 1 & \text{if } x_i \in \mathcal{O} \\ -1 & \text{if } x_i \notin \mathcal{O} \end{cases}$$

- Goal: find weights w such that

$$\begin{aligned} x_i \cdot w \geq 0 \quad & \text{if } y_i = 1 \\ x_i \cdot w \leq 0 \quad & \text{if } y_i = -1 \end{aligned}$$

- Equivalently: $y_i x_i \cdot w \geq 0$. Inequality is called a <u>constraint</u>.
- Idea: We define a <u>risk function</u> R that is positive if some constraint is violated. Then we use optimization to choose w that minimizes R.
- Define the <u>loss function</u>

$$L(y, y_i) = \begin{cases} 0 & \text{if } y_i y \geq 0 \\ -y_i y & \text{otherwise} \end{cases}$$

- Define the <u>risk function</u> (aka <u>objective function</u> or <u>cost function</u>)

$$R(w) = \sum_{i=1}^{n} L(x_i \cdot w, y_i) = \sum_{i \in V} -y_i \cdot x_i \cdot w$$

  where $x_i \cdot w$ is our prediction, y_i is the correct classification and v is the set of indices i for which $y_i x_i \cdot w < 0$.
- If w classifies $X_1, \ldots, X_n$ correctly, then $R(w) = 0$. Otherwise, $R(w) > 0$; we want to find a better w.
- **<u>Goal</u>: Solve this <u>optimization problem</u>; Find w that minimizes $R(w)$.**

01/27/2016

Perceptron algorithm continued

- Duality between x-space and w-space:

x-space (primal)	w-space (dual)
hyperplan e: $\{y : w \cdot y = 0\}$	point: w

  - If a point $x \in H$ (hyperplane), then its dual hyperplane x^* contains the dual point H^* (it also preserves orientation).
- If we want to enforce inequality $x \cdot w \geq 0$, that means:
  - x should be in the correct side of $\{y : y \cdot w = 0\}$ in x-space.
  - w should be on the correct side of $\{v : x \cdot v = 0\}$ in w-space.
  - Add image
  - Note: if data is linearly separable there will always be a section where you can put w.

Optimization algorithm 1 : Gradient descent on R.

- Given a starting point w, find gradient of R with respect to w; this is the direction of the steepest ascent. Take a step in the opposite direction.

$$\nabla R(w) = \begin{bmatrix} \frac{\partial R}{\partial w_1} \\ \frac{\partial R}{\partial w_2} \\ \vdots \\ \frac{\partial R}{\partial w_d} \end{bmatrix} \quad \text{and} \quad \nabla(z \cdot w) = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_d \end{bmatrix} = z$$

$$\nabla R(w) = \sum_{i \in V} \nabla - y_i x_i \cdot w = \sum_{i \in V} -y_i x_i$$

- At any point w, we walk downhill in direction of steepest descent; $-\nabla R$.

Algorithm 1 Gradient descent

```
$w \leftarrow$ arbitrary non-zero starting point (good choice is any $y_i x_i$)
while R(w) > 0 do
    $V \leftarrow$ the set of indices $i$ for which $y_i x_i \cdot w < 0$
    $w \leftarrow w + \epsilon \sum_{i \in V} y_i x_i$
end while
```

- Here ϵ is the step size (aka learning rate chosen empirically).
- Problem: Slow! Each step takes $\mathcal{O}(nd)$ time.

Optimization algorithm 2 : Stochastic gradient descent

- Idea: Each step, pick one misclassified x_i; do gradient descent on loss function $L(x_i \cdot w, y_i)$.
- Called perceptron algorithm. Each step takes $\mathcal{O}(d)$ time.

Algorithm 2 Perceptron algorithm

```
while some $y_i X_i < 0$ do
    $w \leftarrow w + \epsilon y_i x_i$
end while
```

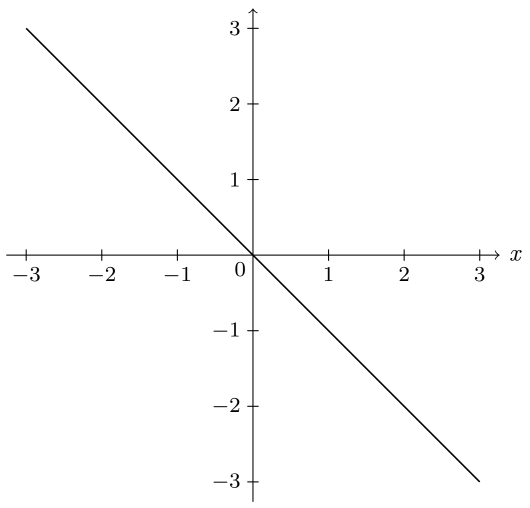
- What if separating hyperplane doesn't pass through the origin?
  - Add a fictitious dimension.
  - Hyperplane: $w \cdot x + \alpha = 0$,

$$\begin{bmatrix} w_1 & w_2 & \dots & w_d & \alpha \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ 1 \end{bmatrix}$$

  - Now we have samples in $\mathbb{R}^{d+1}$, all lying on plane $X_{d+1} = 1$.
- Perceptron Convergence Theorem: If data is linearly separable, perceptron algorithm finds a correct linear classifier in at most $\mathcal{O}(\frac{R^2}{\gamma^2})$ iterations; where $R = \max |X_i|$ is the "radius of data" and γ is the "margin" (length of longest feature vector).

Maximum Margin Classifiers

- The margin of a linear classifier is the distance from the decision boundry to the nearest sample.
- Lets make the margin as big as possible.



- We enforce constraints $y_i(w \cdot x_i + \alpha) \geq 1 \;\; \forall i \in [1, n]$.
- If $||w|| = 1$, constraints imply the margin is at least 1.
- But we allow to have any length, so the margin is at least $\frac{1}{||w||}$.
- There is a slab of width $\frac{2}{||w||}$ that contains no samples.
- To maximize the margin, minimize $||w||$.
- Optimization Problem:

Find w and α that maximize $||w||^2$ subject to $y_i(w \cdot x_i + \alpha) \geq 1 \;\; \forall i \in [1, n]$.

- This is called a quadratic program in $d + 1$ dimensions and n constraints.
- It has one unique solution!
- The solution gives us a maximum margin classifier, aka a hard margin support vector machine (SVM).

02/08/2016

Decision Theory

- Multiple samples with different classes could lie on the same point.
- We want a probabilistic classifier.
- Suppose 10% of the population has cancer; 90% doesn't. We have a probability distribution for calorie intake, $P(X|Y)$:

Calories(X)	< 1200	$1200 - 1600$	> 1600
Cancer (Y=1)	20%	50%	30%
no cancer (Y=-1)	1%	10%	89%

- Recall: $P(X) = P(X|Y=1)P(Y=1) + P(X|Y=-1)P(Y=-1)$
- $P(1200 \leq X \leq 1600) = 0.5 \cdot 0.1 + 0.1 \cdot 0.9 = 0.14$
- You meet guy eating $x = 1400$ cals/day. Guess whether he has cancer?
- **Bayes' Theorem**:

$$\begin{aligned} P(A=a|B) &= \frac{P(B|A=a)P(A=a)}{P(B)} \\ P(Y=1|X=1400) &= \frac{P(X=1400|Y=1)P(Y=1)}{P(X=1400)} = \frac{0.05}{0.14} \\ P(Y=-1|X=1400) &= \frac{P(X=1400|Y=-1)P(Y=-1)}{P(X=1400)} \frac{0.09}{0.14} \\ P(Y=1|X=1400) &= \frac{5}{14} \approx 36\% \text{ probability guy with 1400 cal/day has cancer} \end{aligned}$$

- A <u>loss function</u> $L(z, y)$ specifies badness if true class is y; classifier prediction is z.

$$L(z,y) = \begin{cases} 1 & \text{if } z = 1, y = -1 \\ 5 & \text{if } z = -1, y = 1 \\ 0 & \text{otherwise} \end{cases}$$

- Definitions:
  - loss function above is <u>asymmetrical</u>
  - The <u>0-1 loss function</u> is 1 for incorrect predictions, 0 for correct.
- Let $r : \mathbb{R}^d \to \pm 1$ be a <u>decision rule</u>, aka <u>classifier</u>: a function that maps a feature vector x to 1 ("in class") or -1 ("not in class").
- The <u>risk</u> for r is the expected loss over all values of x, y:

$$\begin{aligned} R(r) &= E[L(r(X), Y)] \\ &= \sum_x (L(r(x), 1)P(Y=1|X=x) + L(r(x), -1)P(Y=-1|X=x))P(x) \\ &= \sum_x (L(r(x), 1)\frac{P(X=x|Y=1)(P(Y=1)}{P(x)} + L(r(x), 1)\frac{P(X=x|Y=-1)(P(Y=-1)}{P(x)})P(x) \\ &= P(Y=1)\sum_x L(r(x), 1)P(X=x|Y=1) + P(Y=-1)\sum_x L(r(x), -1)P(X=x|Y=-1) \end{aligned}$$

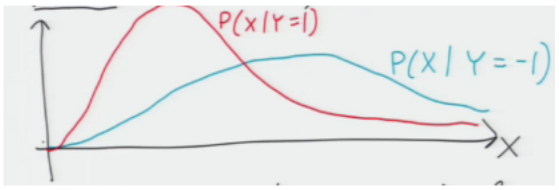
- The Bayes optimal decision rule aka Bayes classifier is the r that minimizes $R(r)$; call it r^*. Assuming $L(z,y)=0$ for $z=y$:

$$r^*(x) = \begin{cases} 1 & \text{if } L(-1,1)P(Y=1|X=x) > L(1,-1)P(Y=-1|X=x) \\ -1 & \text{otherwise} \end{cases}$$

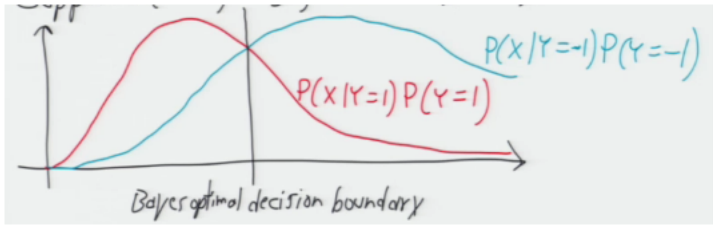
- In cancer example, $r^*=1$ for all intakes ≤ 1600; $r^*=-1$ for intakes ≥ 1600, then the Bayes risk, aka optimal risk is:

$$R(r^*) = 0.1(5\cdot 0.3) + 0.9(1\cdot 0.01 + 1\cdot 0.1) = 0.249$$

- Suppose X has a continuous probability density function (PDF):
- Review:
  - probability that random variable $X \in [x_1, x_2] = \int_{x_1}^{x_1} P(x)dx$
  - area under whole curve $\int_{-\infty}^{\infty} P(x)dx = 1$
  - expected value of $f(x): E[f(x)] = \int_{-\infty}^{\infty} f(x)P(x)dx$
  - mean $\mu = E[x] = \int_{-\infty}^{\infty} xP(x)dx$
  - variance $\sigma^2 = E[(X-\mu)^2] = E[X^2] - \mu^2$



  - Suppose $P(Y=1)=\frac{1}{3}$, $P(Y=-1)=\frac{2}{3}$.



- Define risk as before, replace summations with integrals.

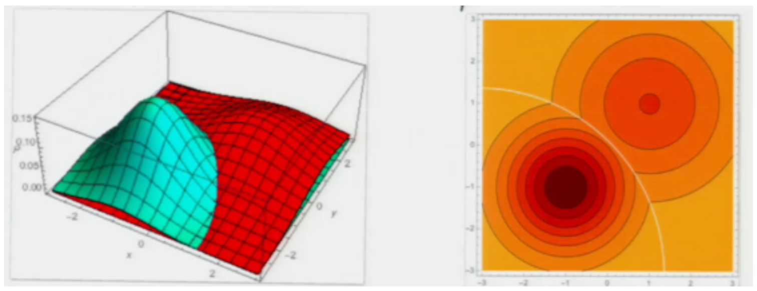
$$\begin{aligned} R(r) &= E[L(r(X),Y)] \\ &= \int_x (L(r(x),1)P(Y=1|X=x) + L(r(x),-1)P(Y=-1|X=x))P(x)dx \\ &= P(Y=1)\int_x L(r(x),1)P(X=x|Y=1)dx + P(Y=-1)\int_x L(r(x),-1)P(X=x|Y=-1)dx \end{aligned}$$

- If L is 0-1 loss, $R(r) = P(r(x))$ is wrong

- For Bayes decision rule, Bayes Risk is the area under the minimum of the functions above. Assuming $L(z, y) = 0$ for $x = y$:

$$R(r^*) = \int min_{y \pm 1} L(-y, y) P(X = x|Y = y) P(Y = y) dx$$

- Bayes optimal decision boundary: $\{x : P(Y = 1|X = x) = 0.5\}$.



3 Ways to Build Classifiers

1. Generative models (e.g. LDA)
   - Assume samples come from probability distributions, different for each class.
   - Guess form of distributions.
   - For each class C, fit distribution parameters for class C samples, giving $P(X|Y = C)$.
   - For each C, estimate $P(Y = C)$.
   - Bayes' Theorem gives $P(Y|X)$.
   - If 0-1 loss, pick class C that maximizes $P(Y = C|X = x)$. Equivalently, maximizes $P(X = x|Y = C)P(Y = C)$.
2. Discriminative models (e.g. logistic regression)
   - Model $P(Y|X)$ directly
3. Find decision boundary (e.g. SVM).
   - Model $r(x)$ directly (no posterior).

- Advantages of (1, 2): $P(Y|X)$ tells you probability your guess is wrong.
- Advantage of (1): you can diagnose outliers: $P(x)$ is very small.
- Disadvantages of (1): often hard to estimate distribution accurately; real distributions rarely match standard ones.

02/10/2016

Gaussian Discriminant Analysis

- Fundamental assumption: each class comes from a normal distribution (Gaussian).

$$X \sim \mathcal{N}(\mu, \sigma^2) : P(X) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{|x-\mu|^2}{2\sigma^2}}$$

- For each class c, suppose we estimate mean μ_c, variance σ_c^2, and prior $\pi_c = P(Y = c)$.
- Given x, Bayes' rule $r^*(x)$ return class C that maximizes $P(X = x|Y = c)\pi_c$.
- $\ln z$ is monotonically increasing for $z > 0$, so its equivalent to maximize,

$$\begin{aligned} Q_c(x) &= \ln(\sqrt{2\pi} P(X = c|Y = c)\pi_c) \\ &= -\frac{|x - \mu_c|^2}{2\sigma^2} - \ln \sigma_c + \ln \pi_c \end{aligned}$$

- $Q_c(x)$ is quadratic in x.

Quadratic Discriminant Analysis (QDA)

- Suppose only 2 classes c, d, Then,

$$r^*(x) = \begin{cases} c & \text{if } Q_c(x) - Q_d(x) > 0 \\ d & \text{otherwise} \end{cases}$$

- The $Q_c(x) - Q_d(x)$ prediction function is quadratic in x.
- Bayes decision boundary is $Q_c(x) - Q_d(x) = 0$.
- In 1D, Bayesian decision boundary may have 1 or 2 points.
- In d-D, Bayesian decision boundary is a quadric.
- To recover posterior probabilities in 2-class case, use Bayes:

$$P(Y = c|X) = \frac{P(X|Y = c)\pi_c}{P(X|Y = c)\pi_c + P(X|Y = d)\pi_d}$$

- Recall $e^{Q_c(x)} = \sqrt{2\pi} P(x)\pi_c$.

$$\begin{aligned} P(Y = c|X = x) &= \frac{e^{Q_c(x)}}{e^{Q_c(x)} + e^{Q_d(x)}} \\ &= \frac{1}{1 + e^{Q_c(x) - Q_d(x)}} \\ &= s(Q_c(x) - Q_d(x)) \end{aligned}$$

- Where s($\cdot$) is the logistic function aka sigmoid function,

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

  - Monotonically increasing.

- $s(0) = \frac{1}{2}$.
- $s(\infty) \to 1$.
- $s(-\infty) \to -1$.
- always $\in [0, 1] \to$ probabilities.

Linear Discriminant Analysis (LDA)

- Fundamental assumption: all the Gaussians have the same variance σ only difference between classes is the mean μ_i.
- Then,

$$Q_c(x) - Q_d(x) = \frac{(\mu_c - \mu_d) \cdot x}{\sigma^2} - \frac{\mu_c^2 - \mu_d^2}{2\sigma^2} + \ln \pi_c + \ln \pi_d$$

- Now its a linear classifiers! Choose c that maximizes,

$$\frac{\mu_c \cdot x}{\sigma^2} - \frac{\mu_c^2}{2\sigma^2} + \ln \pi_c$$

- In 2-class case, decision boundary is $w \cdot x + \alpha = 0$.
- If $\pi_c = \pi_d = \frac{1}{2} \implies (\mu_c - \mu_d) \cdot x - \frac{(\mu_c - \mu_d)}{2} = 0$
- This is the centroid method!
- In 2-class case, Bayes posterior is $P(Y = c|X = x) = s(w \cdot x + \alpha)$

Maximum Likelihood Estimation of Parameters (Ronald Fisher, circa 1912)

- Lets flip biased coins. Heads with probability p; tails with probability $1 - p$.
- 10 flips, 8 heads, 2 tails. What is the most likely value of p?
- Binomial Distribution: $X \sim B(n, p)$

$$P[X = x] = \binom{n}{x} p^x (1 - p)^{n-x}$$

- Our example: n=10,

$$P[X = 10] = 45p^8(1 - p)^2 = \mathcal{L}(x)$$

- Probability of 8 heads in 10 flips: written as a function $\mathcal{L}(p)$ of distribution parameter(s); this is the <u>likelihood function</u>
- <u>Maximum likelihood estimation</u> (MLE): A method of estimating parameters of a statistical model by picking the parameters that maximize the likelihood function.

$$\boxed{\text{Find } p \text{ that maximizes } \mathcal{L}(p)}$$

- Solve this example by setting derivative equal to 0:

$$\frac{d\mathcal{L}}{dp} = 360p^7(1 - p)^2 - 90p^8(1 - p) = 0$$
$$\implies 4(1 - p) - p = 0 \implies p = 0.8$$

- The log likelihood $\mathcal{L}(\cdot)$ is the ln of the likelihood $\mathcal{L}(\cdot)$.

Likelihood of a Gaussian

- Given samples $x_1, x_2, \ldots, x_n$ find best-fit Gaussian.
- Likelihood of generating these samples is,

$$\mathcal{L}(\mu, \sigma; x_1, \ldots, x_n) = P(x_1)P(x_2)\ldots P(x_n)$$

- Log likelihood is,

$$l(\mu, \sigma) = \ln \sum_{i=1}^{n} P(x_i)$$

- Want to set $\nabla_\mu l = 0$, and $\frac{\partial l}{\partial \sigma} = 0$.
- Natural log of Gaussian distribution,

$$\ln P(x) = -\frac{|x-\mu|^2}{2\sigma^2} - \ln \sqrt{2\pi} - \ln \sigma$$

- taking the gradient,

$$\nabla_\mu l = \sum_i \frac{x_i - \mu}{\sigma^2} \implies \hat{\mu} = \frac{1}{n}\sum_i x_i$$

$$\frac{\partial l}{\partial \sigma} = \sum_i \frac{|x_i - \mu|^2 - \sigma^2}{\sigma^3} = 0 \implies \hat{\sigma^2} = \frac{1}{n}\sum_i |x_i - \mu|^2$$

- We don't know μ exactly, so substitute $\hat{\mu}$ in the last equation above.
- For QDA: estimate mean and variance of each class as above, and estimate the priors (for each class c):

$$\hat{\pi_c} = \frac{n_c}{\sum_d n_d} \leftarrow \text{ denominator is the sum of samples in all classes}$$

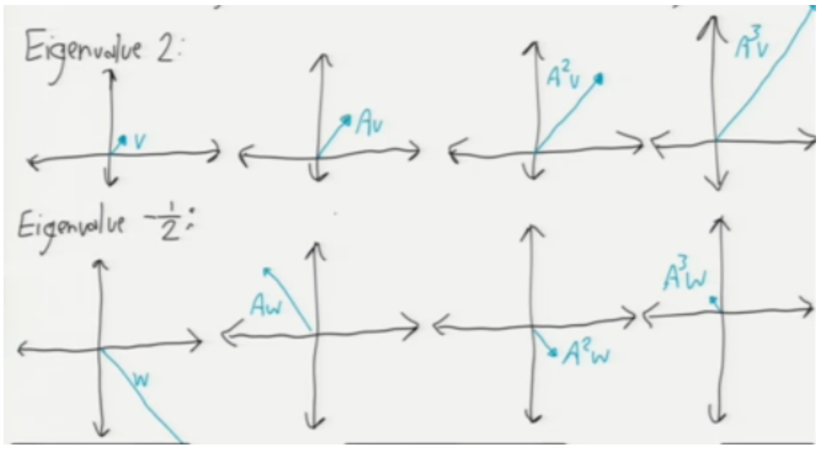
- For LDA: same mean and priors; one variance for all classes:

$$\hat{\sigma^2} = \frac{1}{n}\sum_c \sum_{i:y_i=c} |x_i - \mu_c|^2$$

02/17/2016

Eigenvectors

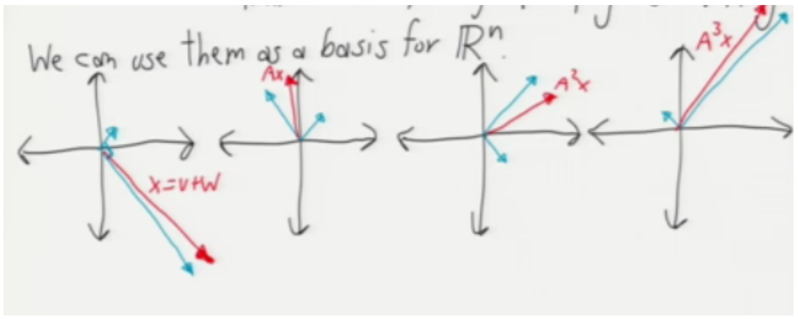
- Given a matrix A, if $Av = \lambda v$ for some vector $v \neq 0$, scalar λ, then v is an eigenvector of A and λ is the associated eigenvalue of A.



- Theorem: if v is an eigenvector of A with eigenvalue λ, then v is an eigenvector of A^k with eigenvalue λ^k.
- Proof: $A^2v = A(\lambda v) = \lambda^2 v$ etc.
- Theorem: moreover, if A is invertible, then v is an eigenvector of A^{-1} with eigenvalue $\frac{1}{\lambda}$.
- Proof: $A^{-1}v = \frac{1}{\lambda}A^{-1}Av = \frac{1}{\lambda}v$.
- Spectral Theorem: Every symmetric nxn matrix has n eigenvectors that are mutually orthogonal,

$$v_i^T v_j = 0, \forall i \neq j$$

- We can use them as a basis for $\mathbb{R}^n$.

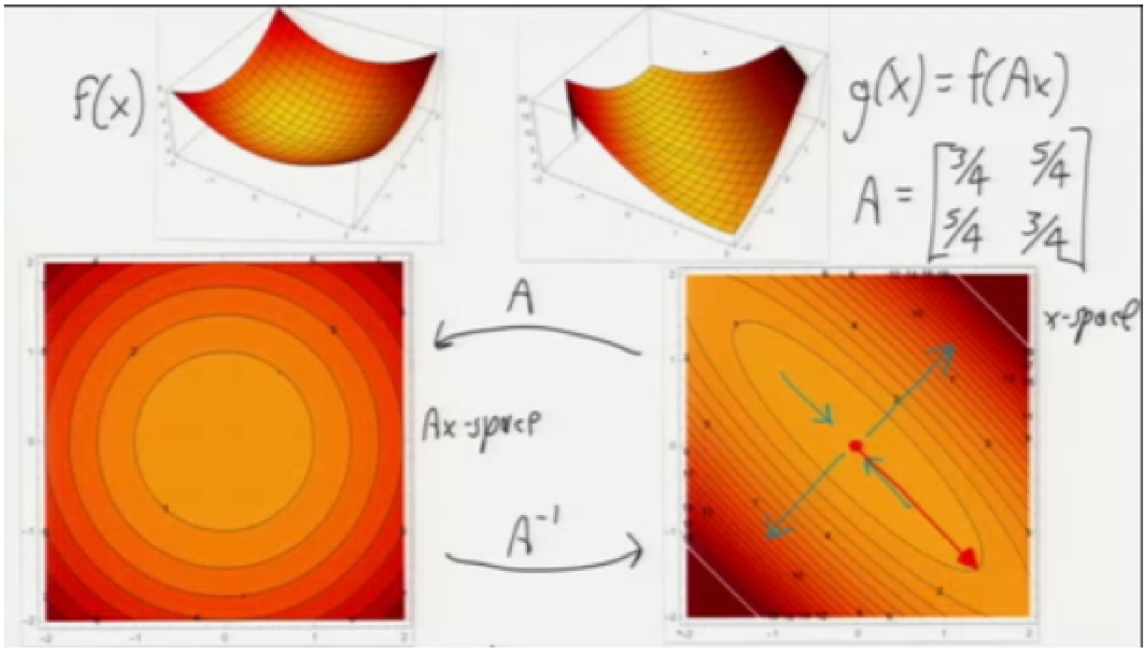


- Write x as a linear combination of eigenvectors:

$$x = \alpha v + \beta w$$
$$A^k x = \alpha \lambda_v^k v + \beta \lambda_w^k w$$

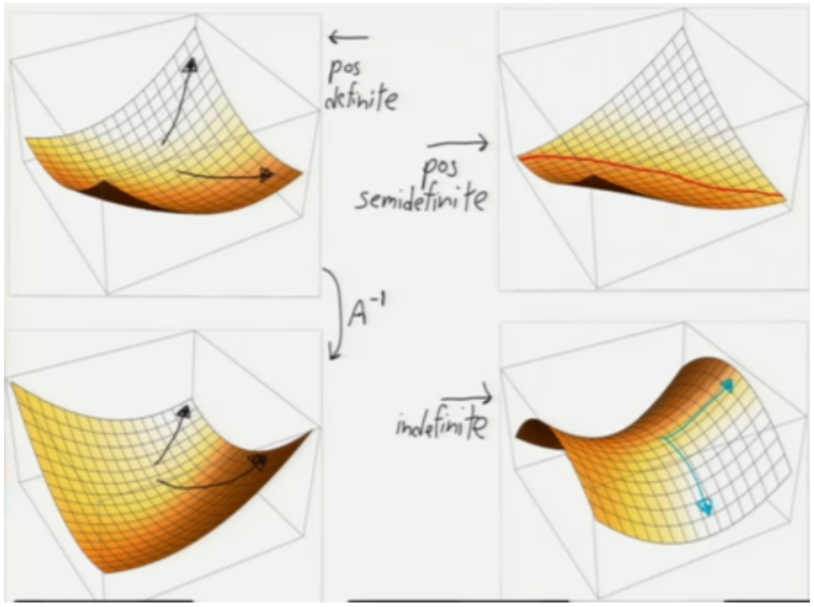
- Ellipsoids

$$f(x) = x^T x \Leftarrow \text{quadratic; isotropic; isosurfaces are spheres.}$$
$$g(x) = f(Ax) \Leftarrow \text{Asymmetric.}$$
$$= x^T A^2 x \Leftarrow \text{quadratic form of the matrix } A^2 \text{ anisotropic; isosurfaces are ellipsoids}$$



- $g(x) = 1$ is an ellipsoid with axes $v_1, v_2, \ldots, v_n$ and radii $\frac{1}{\lambda_1}, \frac{1}{\lambda_2}, \ldots, \frac{1}{\lambda_n}$ (eigevalues of A) because if v_i has length $\frac{1}{\lambda_i}$ (red arrow), $g(v_i) = f(\lambda_i v_i) = 1 \Rightarrow v_i$ lies on the ellipsoid.

- Bigger eigenvalue $\Leftrightarrow$ steeper slope $\Leftrightarrow$ shorter ellipsoid radius.
- Alternative interpretation:
  - Ellipsoids are spheres in a distance metric A^2.
  - Call $M = A^2$ a metric tensor because the distance between samples x and z in stretched space is $d(x, z) = |Ax - Az| = \sqrt{(x-z)^T M (x-z)}$.
  - Ellipsoids are "spheres" in this metric: $\{x : d(x, \text{center}) = \text{isovalue}\}$.
- A square matrix B is,
  - positive definite if $w^T B w > 0$ for all $w \neq 0 \Leftrightarrow$ all positive eigenvalues.
  - positive semidefinite if $w^T B w \geq 0$ for all $w \neq 0 \Leftrightarrow$ all non-eigenvalues.
  - indefinite if at least one positive eigenvalue and one negative eigenvalue.
  - invertible if no zero eigenvalue.



- Metric tensor must be symmetric positive definite (SPD).
- Special case: M and A are diagonal matrices $\Leftrightarrow$ eigenvectors are coordinate axes $\Leftrightarrow$ ellipsoids are axis-aligned.

Building a Quadratic with specified eigenvectors and eigenvalues

- Choose n mutually orthogonal unit n-vectors $v_1, \ldots, v_n$ Let, $V = [v_1, v_2, \ldots, v_n]$
- Observe: $V^TV = I \Rightarrow V^T = V^{-1} \Rightarrow VV^T = I$.
- V is orthogonal matrix: acts like rotation (or reflection).
- Choose some inverse radii λ_i:
- Let,

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_n \end{bmatrix}$$

- Theorem: $A = V\Lambda V^T = \sum_{i=1}^n \lambda_i v_i v_i^T$ has chosen eigenvectors and eigenvalues.
- Proof: $AV = V\Lambda \Leftarrow$ definition of eigenvectors! (in matrix form).
- This is a matrix factorization called the eigen-decomposition
- Λ is the diagonalized version of A.
- V^T rotates the ellipsoid to be axis-aligned.
- This is also a recipe for building quadratics with axes v_i, radii $\frac{1}{\lambda_i}$.
- Given SPD metric tensor M, we can find symmetric square root $A = M^{\frac{1}{2}}$:
  - compute eigenvectors and eigenvalues of M
  - take square roots of $M's$ eigenvalues
  - reassemble matrix A.

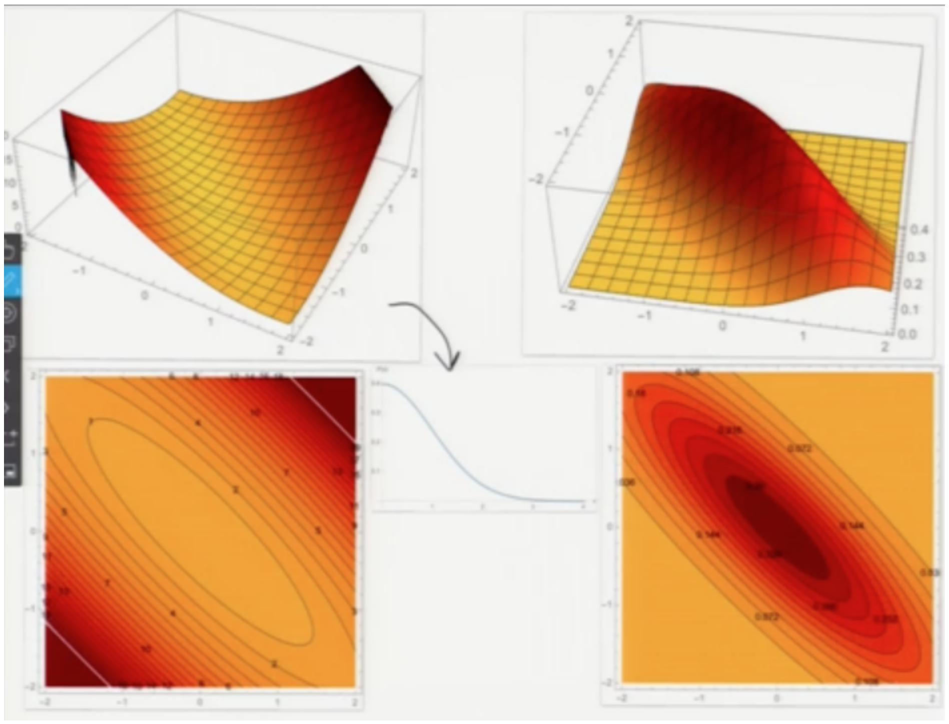
02/22/2016

Anisotropic Multivariate Gaussians

- $X \sim \mathcal{N}(\mu, \Sigma) \Leftarrow X$ is random d-vector with mean μ.

$$P(x) = \frac{1}{\sqrt{(2\pi)^d}\sqrt{|\Sigma|}} \exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

- Σ is the dxd SPD covariance matrix
- Σ^{-1} is the dxd SPD precision matrix; serves as a metric tensor.
- Write $P(x) = n(q(x))$, where $q(x) = (x-\mu)^T\Sigma^{-1}(x-\mu)$. Note, $n : \mathbb{R} \to \mathbb{R}$, exponential, $q : \mathbb{R}^d \to \mathbb{R}$, quadratic.
- Principle: given $f : \mathbb{R} \to \mathbb{R}$, isosurfaces of $f(q(x))$ are same as $q(x)$ (different isovalues), except that some might be "combined."



- Covariance:

$$\begin{aligned} \text{Cov}(X,Y) &= E[(X - E[X])(Y - E[Y])^T] \\ &= E[XY^T] - \mu_x\mu_y^T \\ \text{Var}(X) &= \text{Cov}(X,X) \end{aligned}$$

  - For a Gaussian, one can show $\text{Var}(X) = \Sigma$. Hence,

$$\Sigma = \begin{bmatrix} \text{Var}(X_1) & \text{Cov}(X_1,X_2) & \dots & \text{Cov}(X_1,X_d) \\ \text{Cov}(X_2,X_1) & \text{Var}(X_2) & \dots & \text{Cov}(X_2,X_d) \\ \vdots & & \ddots & \vdots \\ \text{Cov}(X_d,X_1) & \text{Cov}(X_d,X_2) & \dots & \text{Var}(X_d) \end{bmatrix}$$

  - X_i, X_j independent $\Rightarrow \text{Cov}(X_i,X_j) = 0$.

- $\text{Cov}(X_i, X_j) = 0$ and they come from a joint normal distribution $\Rightarrow X_i, X_j$ independent.
- All features pairwise independent $\Rightarrow \Sigma$ is diagonal.

$$\Sigma \text{ is diagonal} \Leftrightarrow \text{axis-aligned Gaussian; squared radii on the diagonal.}$$
$$\Leftrightarrow P(x) = P(X_1)P(X_2)\dots P(X_d)$$

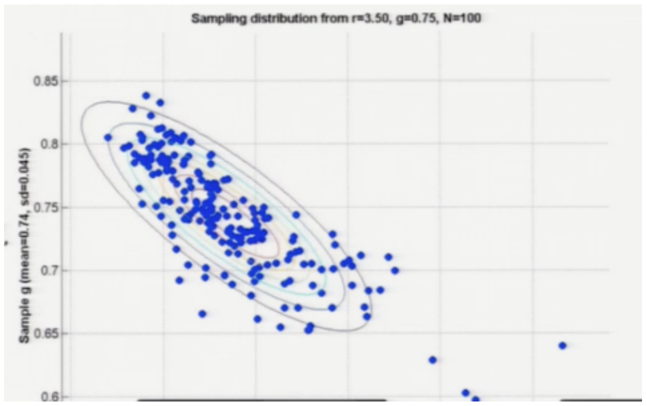
$$\begin{tikzcd}
\overset{\text{Covariance Matrix}}{\Sigma} \arrow[rr, leftrightarrow, "\text{inverse}"] \arrow[d, bend right, "\surd"'] & & \overset{\text{Precision Matrix (Metric)}}{M = \Sigma^{-1}} \arrow[d, bend right, "\surd"'] \\
\underset{\text{(Maps spheres to ellipsoids)}}{\Sigma^{1/2}} \arrow[rr, leftrightarrow, "\text{inverse}"] \arrow[u, bend right, "\text{square}"'] & & \underset{\text{(Maps ellipsoids to spheres)}}{A = \Sigma^{-1/2}} \arrow[u, bend right, "\text{square}"']
\end{tikzcd}$$

- Eigenvalues of $\Sigma^{1/2}$ are ellipsoid radii (standard deviations along the eigenvectors).
- Eigenvalues of Σ are variances along along eigenvectors.
- Diagonalizing $\Sigma = V\Lambda V^T$, $\Sigma^{\frac{1}{2}} = V\Lambda^{\frac{1}{2}}V^T$.

Maximum Likelihood estimation for anisotropic Gaussians

- Given samples $x_1, \dots, x_n$ and classes $y_1, \dots y_n$ find the best-fit Gaussians.
- For QDA:

$$\hat{\Sigma_c} = \frac{1}{n_c}\sum_{i:y_i=c}(x_i - \mu_c)(x_i - \mu_c)^T \Leftarrow \text{conditional covariance for samples in class c}$$



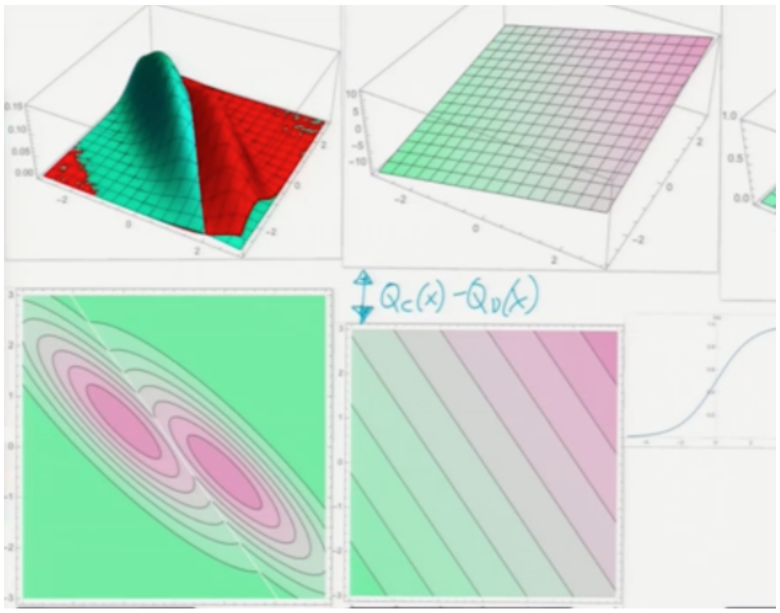
  - Priors π_c and means $\hat{\mu_c}$ same as before.
  - $\hat{\Sigma_c}$ is the positive semidefinite. If some zero eigenvalue, must eliminate the zero-variance dimension.
- For LDA:

$$\hat{\Sigma} = \frac{1}{n}\sum_c\sum_{i:y_i=c}(x_i - \mu_c)(x_i - \mu_c)^T \Leftarrow \text{pooled within class covariance matrix}$$

- QDA:
  - π_c, μ_c, Σ_c may be different for each class c.
  - Goal is to choose c that maximizes $P(X = x|Y = c)\pi_c$, which is equivalent to maximizing the quadratic discriminant function,

$$\begin{aligned} Q_c(x) &= \ln\left(\sqrt{(2\pi)^d}P(x)\pi_c\right) \\ &= -\frac{1}{2}q_c(x) - \frac{1}{2}\ln|\Sigma_c| + \ln\pi_c \end{aligned}$$

- 2 classes: Prediction function $Q_c(x) - Q_d(x)$ is quadratic, but may be indefinite.
- Since the prediction function is quadratic $\Rightarrow$ Bayes decision boundary is quadric.
- Posterior is $P(Y = c|X = x) = s(Q_c(x) - Q_d(x))$ where $s(\cdot)$ is the logistic function.



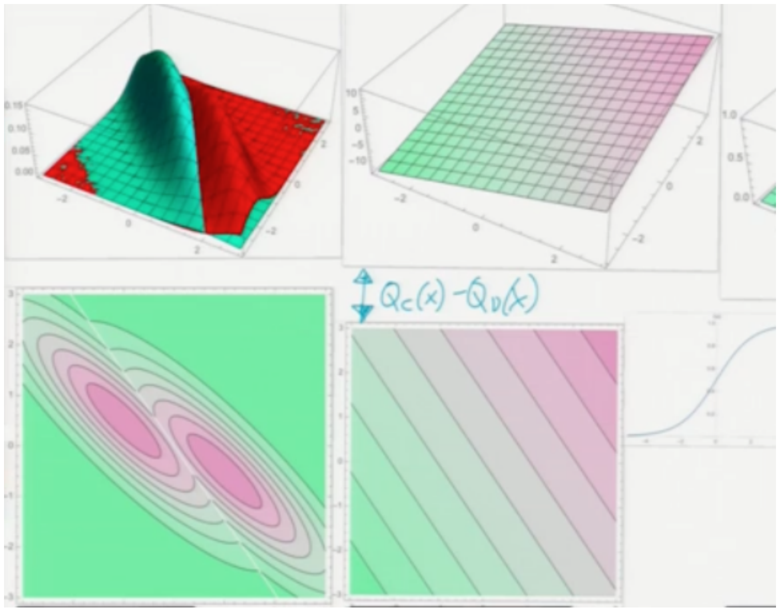
- LDA:
  - Once Σ for all classes.

$$Q_c(x) - Q_d(x) = (\mu_c - \mu_d)^T \Sigma^{-1} x - \frac{\mu_c^T \Sigma^{-1} \mu_c - \mu_d^T \Sigma^{-1} \mu_d}{2} + \ln \pi_c - \ln \pi_d$$

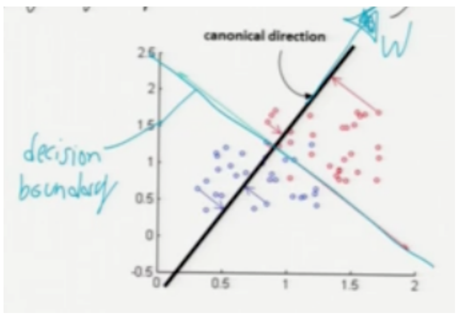
  - Choose class c that maximizes the linear discriminant function,

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

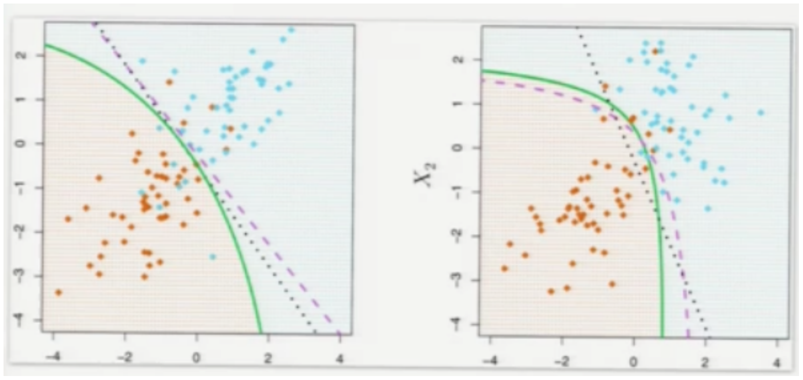
  - 2 classes:
    * Decision boundary is $w^T x + \alpha = 0$
    * Posterior is $P(Y = c|X = x) = s(w^T x + \alpha)$.



- <u>Notes</u>
  - Changing prior π_c (or loss) is easy: if its LDA adjust α.
  - LDA is often interpreted as projecting samples onto the normal vector.



  - For 2 classes,
    - LDA has $d+1$ parameters (w, α).
    - QDA has $\frac{d(d+3)}{2} + 1$ parameters
    - $\Rightarrow$ QDA more likely to overfit.



02/24/2016

Regression aka fitting curves to data

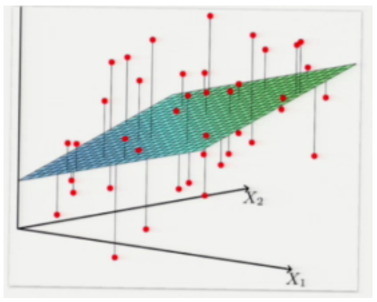
- Classification: given sample x, predict class (often binary).
- Regression given sample, x, predict a numerical value.
  - Choose form of regression function $h(x; p)$ with parameters p.
    - like predictor function in classification.
  - Choose a cost function (objective function) to optimize.
    - Usually based on a loss function; e.g. risk function = expected loss.
- Some regression functions:
  1. Linear: $h(x; w, \alpha) = w^T x + \alpha$.
  2. Polynomial.
  3. Logistic: $h(x; w, \alpha) = s(w^T x + \alpha)$. Recall: logistic function $s(\gamma) = \frac{1}{1+e^{-\gamma}}$
- Some loss functions: let z be prediction $h(x)$; y be true value.
  A. $L(z, y) = (z - y)^2$ squared error.
  B. $L(z, y) = |z - y|$ absolute error.
  C. $L(z, y) = -yln(z) - (1 - y)ln(1 - z)$ logistic loss.
- Some cost functions to minimize:
  a. $J(h) = \frac{1}{n} sum_{i=1}^{n} L(h(X_i), y_i)$ mean loss.
  b. $J(h) = \max_{i=1}^{n} L(h(X_i, y_i))$ maximum loss.
  c. $J(h) = \sum \omega_i L(h(X_i, y_i))$ weighted sum.
  d. $J(h) = \frac{1}{n} \sum L(h(X_i, y_i)) + \lambda |w|^2$ ℓ_2 penalized/regularized.
  e. $J(h) = \frac{1}{n} \sum L(h(X_i, y_i)) + \lambda |w|$ ℓ_1 penalized/regularized.
- Some combinations:

Name	Regression	Loss	Cost	Algorithm
Least-squares linear regression	1	A	a	quadratic, minimize w/calculus
Weighted least-squares linear	1	A	c	quadratic, minimize w/calculus
Ridge regression	1	A	d	quadratic, minimize w/calculus
Lasso	1	A	d	minimize w/gradient descent
Logistic regression	3	C	a	minimize w/gradient descent
Least absolute deviations	1	B	a	minimize w/linear program
Chebyshev criterion	1	B	b	minimize w/linear program

Least-Squares Linear Regression 1 + A + a.

- Optimization problem:

$$\boxed{\text{Find } w, \alpha \text{ that minimizes } \sum_{i=1}^{n} (x_i \cdot w + \alpha - y_i)^2}$$



- Convention:

- X is nxd <u>design matrix</u> of samples.
- y is d-vector of dependent scalars.
- $X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & \\ \vdots & \vdots & \dots & \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$
- Usually $n > d$.
- Sample row vector is X_i^T.
- Column vector is X_{*j}.
- $y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$
- Recall fictitious dimension trick: replace $x \cdot w + \alpha$ with,

$$[x_1\ x_2\ \dots\ x_d\ 1] \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ \alpha \end{bmatrix}$$

  Now X is an nx$(d+1)$ matrix; w is a $(d+1)$-vector.

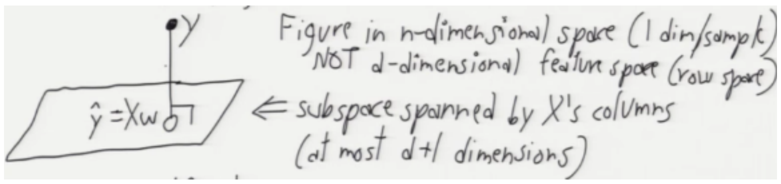
- Rewritten optimization problem:

$$\boxed{\text{Find } w \text{ that minimizes } |Xw - y|^2}$$

- Optimize by calculus, minimize residual sum of squares:

$$RSS(w) = w^T X^T X w - 2y^T X w + y^T y = 0$$
$$\implies X^T X w = X^T y \Leftarrow \text{The } \underline{\text{normal equations}}$$

- If $X^T X$ is singular, problem is under-constrained.
- We use a linear solver to find $w = (X^T X)^{-1} X^T y$.
- $X^+ = (X^T X)^{-1} X^T$ is the <u>pseudoinverse</u> of X and is a $(d+1)$xn matrix.
- Observe: $X^+ X = (X^T X)^{-1} X^T X = I \Leftarrow (d+1)x(d+1)$.
- Observe: the predicted values of y are $\hat{y} = h(x;) \Rightarrow = Xw = XX^+ y = Hy$.
- where $H = XX^+$ is the <u>hat matrix</u> because it puts the hat on y.
- Interpretation as a projection:
  - $\hat{y} = Xw \in \mathbb{R}^n$ is a linear combination of columns of X.
  - For fixed X, varying w, Xw is a subspace of $\mathbb{R}^n$ spanned by columns.


y
Figure in n-dimensional space (1 dim/sample) NOT d-dimensional) feature space (row space)
ŷ = Xw
⇐ subspace spanned by X's columns (at most d+1 dimensions)

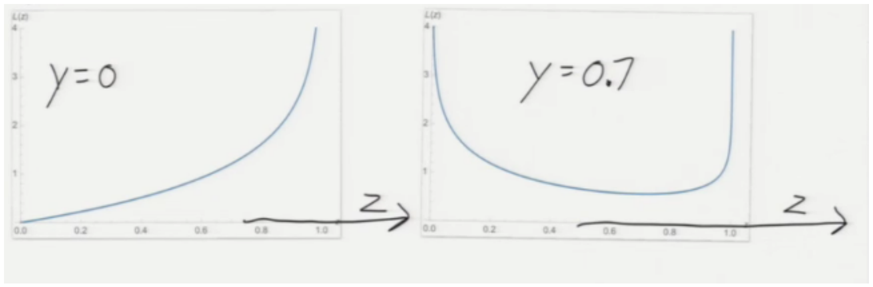
  - Minimizing $|\hat{y} - y|$ find point $\hat{y}$ nearest y on subspace $\Rightarrow$ project y onto subspace.
  - Error is smallest when line is perpendicular to subspace: $X^T(Xw - y) = 0 \Rightarrow$ the normal equations!
  - Hat matrix H (also called projection matrix) does the projecting.

- Advantages:
  - Easy to compute; just solve a linear system.
  - Unique, stable solution.
- Disadvantages:
  - Very sensitive to outliers, because error is squared!
  - Fails if $X^T X$ is singular.

Logistic Regression (David Cox, 1953) 3 + C + a

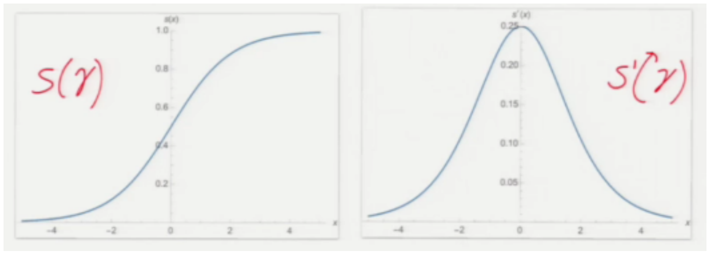
- Fits "probabilities" in range (0, 1).
- Usually used for classification. The input y_i's can be probabilities, but in most applications they are all 0 or 1.
- QDA, LDA: generative models.
- Logistic regression: discriminative model.
- Optimization problem:

$$\boxed{\text{Find } w, \alpha \text{ that minimizes } J = \sum_{i=1}^{n} (y_i \ln s(X_i \cdot w + \alpha) + (1 - y_i) \ln(1 - s(X_i \cdot w + \alpha))}$$



- $-J(w, \alpha)$ is convex. Solve by gradient ascent.

$$s'(\gamma) = \frac{d}{d\gamma} \frac{1}{1 + e^{-\gamma}} = \frac{e^{-\gamma}}{(1 + e^{-\gamma})^2}$$
$$= s(\gamma)(1 - s(\gamma))$$

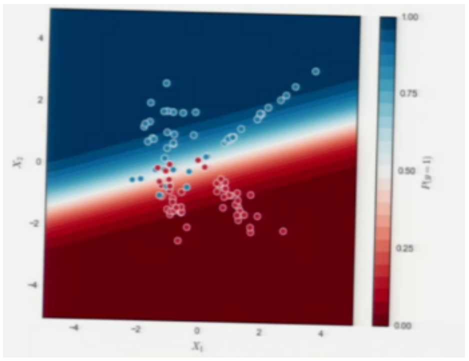


- Let $s_i = s(X_i \cdot w + \alpha)$,

$$\nabla_w J = \sum \left(\frac{y_i}{s_i} \nabla s_i - \frac{1 - y_i}{1 - s_i} \nabla s_i \right)$$
$$= \sum \left(\frac{y_i}{s_i} - \frac{1 - y_i}{1 - s_i} \right) s_i (1 - s_i) X_i$$
$$= \sum (y_i - s_i) X_i$$

- Gradient ascent rule: $w \leftarrow w + \epsilon \sum_{i=1}^{n} (y_i - s(X_i \cdot w + \alpha)) X_i$

- Stochastic gradient ascent: $w \leftarrow w + \epsilon(y_i - s(X_i \cdot w + \alpha))X_i$ works best if we shuffle samples in random order; process one by one.
- For very large n, sometimes converges before we visit all samples!
- Starting from $w = 0, \alpha = 0$ works well in practice.



02/29/2016

Weighted Least-Squares Regression

- Assign each sample a weight ω_i; collect them in an nxn diagonal matrix Ω.
- Greater $\omega_i \Rightarrow$ work harder to minimize $|\hat{y}_i - y_i|$. Recall: $\hat{y} = Xw$.
- Optimization problem:

$$\boxed{\text{Find } w \text{ that minimizes } (Xw - y)^T \Omega (Xw - y)}$$

- Note,

$$(Xw - y)^T \Omega (Xw - y) = \sum_{i=1}^{n} \omega_i ((Xw)_i - y_i)^2$$

- Sove for w in normal equations:

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

- Note: $\Omega^{\frac{1}{2}} \hat{y}$ is orthogonal projections of $\Omega^{\frac{1}{2}} y$ onto subspace spanned by columns of $\Omega^{\frac{1}{2}} X$.

Newton's Method

- Iterative optimization method for smooth functions $J(w)$ often much faster than gradient descent.
- Idea: You're at point v in a space, where you're trying to optimize w. Approximate $J(w)$ near v by quadratic function. Jump to its unique critical point. Repeat until bored.
- Taylor series about v:

$$\nabla J(w) = \nabla J(v) + (\nabla^2 J(v))(w - v) + \mathcal{O}(|w - v|^2)$$

- $\nabla^2 J(v)(w - v)$ is the Hessian matrix of J at v.
- Find critical point w by setting $\nabla J(w) = 0$:

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

Algorithm 3 Newton's Method

```
pick starting point $w$
while not "converged" do
    $e \leftarrow$ solution to linear system $(\nabla^2 J(w))e = -\nabla J(w)$
    $w \leftarrow w + e$
end while
```

- Warning: Doesn't know difference between min, max or saddle points. Starting point must be "close enough" to desired solution.
- Facts:
  - If objective function J is actually a quadratic function, you'll jump to the minimum in one iteration. The closer J is to quadratic the faster it is, and vice versa.
  - Superior to blind gradient descent: 1) Doesn't just take a blind step length downhill it actually tries to figure out the bottom of the curve. 2) It doesn't just pick one direction of steepest descent it tries to optimize all at once.
  - Biggest disadvantage: Computing the Hessian is very expensive.

Logistic Regression

- Recall: $s'(\gamma) = s(\gamma)(1 - s(\gamma))$, $s_i = s(X_i \cdot w)$.
- $\nabla_w J(w) = \sum_{i=1}^{n}(y_i - s_i)X_i = X^T(y - s)$, where s is n-vector with components s_i.

$$\nabla^2 J(w) = -\sum_{i=1}^{n} s_i(1 - s_i)X_i X_i^T = X^T \Omega X$$

  where Ω is a diagonal matrix with components $s_i(1 - s_i)$.
- Ω is positive definite for all w,so $X^T\Omega X$ is positive semidefinite, so $-J(w)$ is convex.

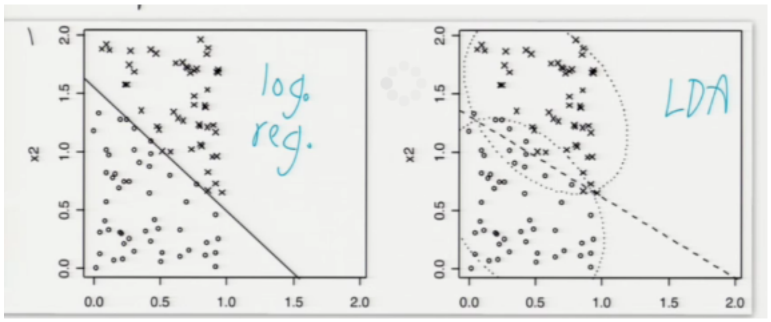
Algorithm 4 Newton's Method

```
while not "converged" do
    Solve for $e$ in normal equations: $(X^T\Omega X)e = X^T(y - s)$
    $w \leftarrow w + e$
end while
```

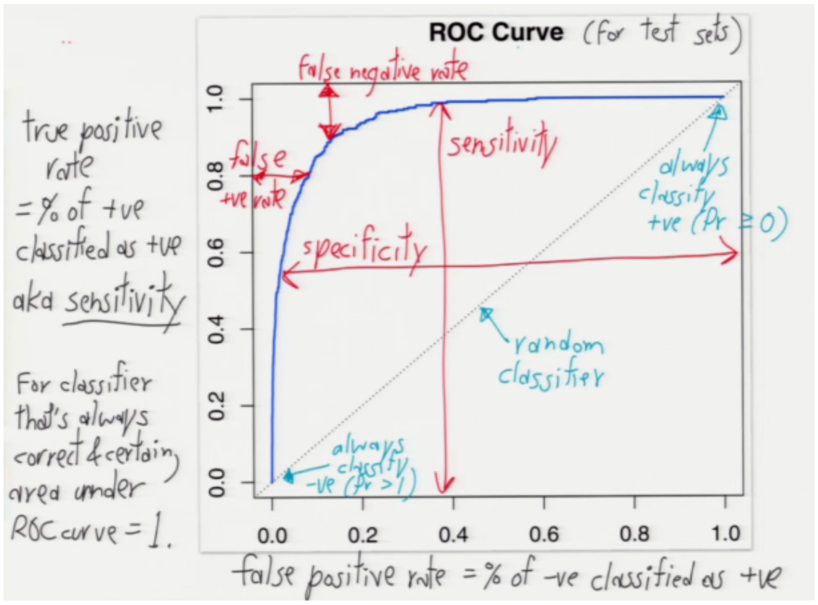
  Remember Ω, s are functions of w.
- An example of "iteratively re-weighted least squares."
- Ω prioritizes samples with s_i near 0.5; tunes out samples that are near 0/1.
- Idea: If n very large, save time by using a random subsample of the samples per iteration. Increase sample size as you go.

LDA vs. Logistic regression

- Advantages of LDA:
  - For well-separated classes, LDA stable; logistic regression surprisingly unstable.
  - For more than 2 classes easy & elegant, logistic regression needs modifying ("softmax regression").
  - Slightly more accurate when classes nearly normal, especially if n is small.
- Advantages of Logistic regression:
  - More emphasis on decision boundary.



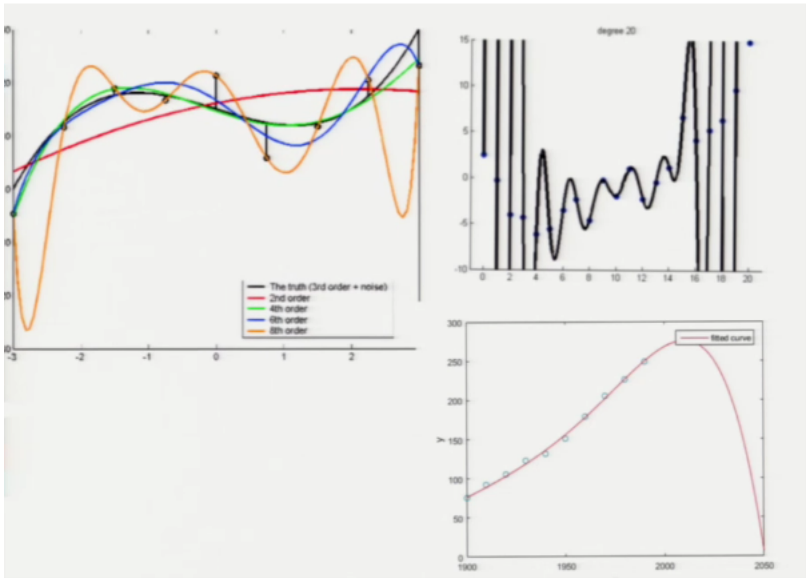
  - Hence less sensitive to outliers.
  - Easy and elegant treatment of "partial" class membership; LDA is all-or-nothing.
  - More robust on some non-gaussian distributions (e.g. large skew).
- ROC Curve



- Receiver Operating Characteristics
- A way of telling you information about the test set, not part of the learning algorithm.
- Run classifier on test set, and it shows the rate of false positives vs. true positives for a range of settings.
- x-axis false positive rate = % of negative samples accidentally classified as positive.
- y-axis true positive rate = % of positive that actually got classified as positive (aka sensitivity).

Least-Squares Polynomial Regression

- Replace each X_i with feature vector $\Phi(X_i)$ with all terms of degree $0 \ldots p$.
- e.g. $\phi(X_i) = \begin{bmatrix} X_{i1}^2 & X_{i1}X_{i2} & X_{i2}^2 & X_{i1} & X_{i2} & 1 \end{bmatrix}^T$
- Can also use non-polynomial features (e.g. edge detectors).
- Otherwise just like linear or logistic regression.
- Logistic regression plus quadratic features = same logistic posteriors as QDA. Very easy to overfit!



03/02/2016

Statistical Justification for Regression

- Typical model of reality:
  - Samples come from unknown probability distribution $X_i \sim D$.
  - y-values are sum of unknowns, non-random surface plus random noise: for all X_i,

$$y_i = f(X_i) + \epsilon_i$$

- Goal of regression: find h that estimates f.
- Ideal approach: choose $h(x) = E_y[Y|X = x]$

Least-squares Regression from Max Likelihood

- Suppose $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$; then $y_i|X_i \sim \mathcal{N}(f(X_i), \sigma^2)$.
- Recall that log likelihood for normal distribution is,

$$lnP(y_i) = -\frac{|y_i - \mu|^2}{2\sigma^2} - constant \qquad \Leftarrow \mu = f(X_i)$$
$$ln(P(y_1)P(y_2)\dots P(y_n)) = lnP(y_1) + lnP(y_2) + \dots + lnP(y_n)$$

- Takeaway: If you apply the principle of max likelihood to linear regression with an input model that assumes gaussian noise $\Rightarrow$ find f by least-squares.

Empirical Risk

- The <u>risk</u> for hypothesis h is the expected loss $R(h) = E[L]$ over all X, Y.
- Discriminative model: we don't know X's distribution D. How can we minimize the risk?
- <u>Empirical distribution</u>: A discrete probability that is the sample set, with each sample equally likely.
- <u>Empirical risk</u>: expected loss over empirical distribution $\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(X_i), y_i))$.
- Takeaway: this is why we minimize the sum of loss functions.

Logistic regression from Max Likelihood

- If we accept the logistic regression function, what cost function should we use?
- Given arbitrary sample x, write probability it is in (not in) the class: (fictitious dimension: x ends w/1; w ends w/α).

$$P(y = 1|x; w) = h(x; w) \qquad \Leftarrow h(x; w) = s(w^T x)$$
$$P(y = 0|x; w) = 1 - h(x; w)$$

- Combine these 2 facts into 1 expression:

$$P(y|x; w) = h(x)^y (1 - h(x))^{1-y}$$

- Likelihood is,

$$L(w; x_1, \dots, x_n) = \prod_{i=1}^{n} P(y_i|X_i; w)$$
$$l(w) = lnL(w) = \sum_{i=1}^{n} lnP(y_i|X_i; w)$$
$$= \sum (y_i lnh(X_i) + (1 - y_i)ln(1 - h(X_i)))$$

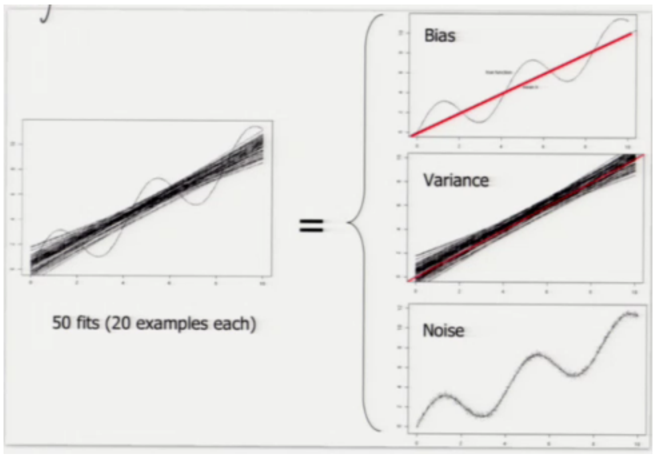
- which is negated logistic cost function J(w).

The Bias-variance Decomposition

- There are 2 sources of error in a hypothesis h:
  - bias: error due to inability of hypothesis h to fit f perfectly. e.g. fitting quadratic f with a linear ℓ.
  - variance: error due to fitting random noise in data. e.g. we fit linear f with a linear h, yes $h \neq f$.
- Model: generate samples $X_1 \dots X_n$ from some distribution D. Values $y_i = f(X_i) + \epsilon_i$. Fit hypothesis h to X, y.
- Now h is a random variable; i.e. its weights are random.
- Consider an arbitrary point $z \in \mathbb{R}^d$ (not necessarily a sample!) and $\gamma = f(z) + \epsilon$.
- Note: $\mathrm{E}[\gamma] = f(z); \mathrm{Var}(\gamma) = \mathrm{Var}(\epsilon)$;
- Risk function when loss is squared error:
- Here we are taking the expectation over all possible training sets X, y and values of γ.

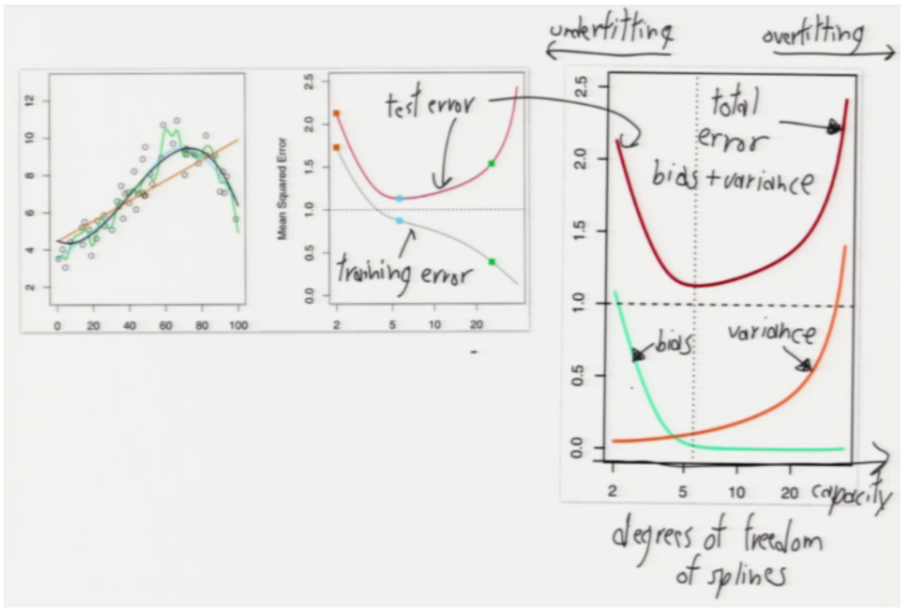
$$\begin{aligned}
R(h) &= E[L(h(z), \gamma)] \\
&= E[(h(z) - \gamma)^2] \\
&= E[h(z)^2] + E[\gamma^2] - 2E[\gamma h(z)] \\
&= \mathrm{Var}(h(z)) + E[h(z)]^2 + \mathrm{Var}(\gamma) + E[\gamma]^2 - 2E[\gamma]E[h(z)] \\
&= (E[h(z)] - E[\gamma])^2 + \mathrm{Var}(h(z)) + \mathrm{Var}(\gamma) \\
&= E[h(z) - f(z)]^2 + \mathrm{Var}(h(z)) + \mathrm{Var}(\epsilon)
\end{aligned}$$

  - We take expectation over possible training sets, X, y and values of γ.
  - $E[h(z) - f(z)]^2$: is square of the bias of method.
  - $\mathrm{Var}(h(z))$: variance of method.
  - $\mathrm{Var}(\epsilon)$: irreducible error (comes from test point not from training).
- This is point-wise version. Mean version: Let $z \sim D$ be random variable; take expectation of the squares bias, variance over z



  - Under-fitting: too much bias.
  - Over-fitting caused by too much variance.
- Training error reflects bias but not variance; test error reflects both.
- For many distributions, variance $\to 0$ as $n \to \infty$.
- If h can fit f exactly, for many distributions bias $\to 0$ as $n \to \infty$.

- If h cannot fit f well, bias is large at "most" points.
- Adding a good feature reduces bias; adding a bad feature rarely increases it.
- Adding a feature usually increases variance.
- Cannot reduce irreducible error.
- Noise in test set affects only var(ϵ); noise in training set affects only bias and Var(h).
- For real-world data, f is rarely knowable (and noise model might be wrong).
- But we can test learning algorithms by choosing f and making synthetic data.



- Example: Least-Squares Linear Regression:
  - No fictitious dimension.
  - Model: $f(z) = v^T z$.
  - Let e be a noise n-vector $e \sim \mathcal{N}(0, \sigma^2)$.
  - Training values: $y = Xv + e$. Input to regression algorithm are y, X.
  - Linear regression computes weights:

$$w = X^+ y = X^+(Xv + e) = v + X^+ e$$

  - Bias is,

$$E[h(z) - f(z)] = E[w^T z - v^T z] = E[z^T X^+ e] = z^T X^+ E[e] = 0$$

  - Warning: This does not mean $h(z) - f(z)$ is everywhere 0. Sometimes positive, sometimes negative, mean over training sets is 0.
  - Variance is,

$$\begin{aligned}
\text{Var}(h(z)) &= \text{Var}(w^T z) = \text{Var}(z^T w) \\
&= \text{Var}(z^T(v + X^+ e)) \\
&= \text{Var}(z^T v + z^T X^+ e)) \\
&= \text{Var}(z^T X^+ e) \\
&= \sigma^2 |z^T X^+|^2 \\
&= \sigma^2 |z^T (X^T X)^{-1} X^T)|^2 \\
&= \sigma^2 z^T (X^T X)^{-1} X^T X (X^T X)^{-1} z \\
&= \sigma^2 z^T (X^+ X)^{-1} z
\end{aligned}$$

  - If choose coordinate system so $E[X] = 0$ it simplifies to $\approx \sigma^2 \frac{d}{n}$.
  - Takeaways: Bias can be zero when hypothesis function can fit the real one. Variance portion of RSS (overfitting) decreases as $\frac{1}{n}$, increases as d.

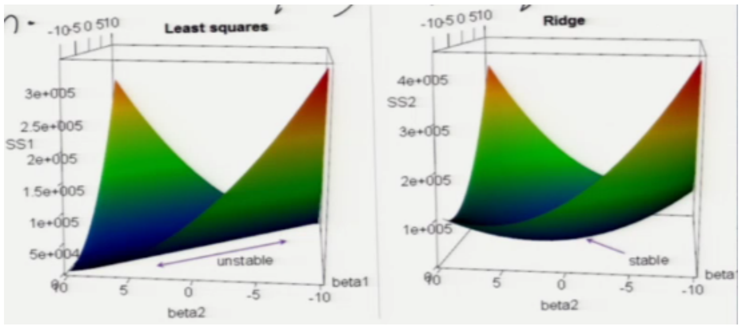
03/07/2016

Ridge Regression (aka Tikhonov regularization)

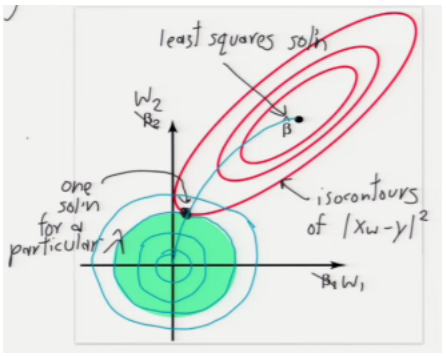
- (1) + (A) + ℓ_2 penalized mean loss (d).
- Optimization problem $J(w)$:

$$\boxed{\text{Find } w \text{ that minimizes } |Xw - y|^2 + \lambda |w'|^2}$$

  Where w' is w with components α replaced by 0. X has fictitious dimension but we DON'T penalize α.
- Adds a penalty term to encourage small $|w'|$ – called shrinkage.
- Why? Guaranteed positive definite normal equations; always unique solution. e.g. when $d > n$ always semi-definite.



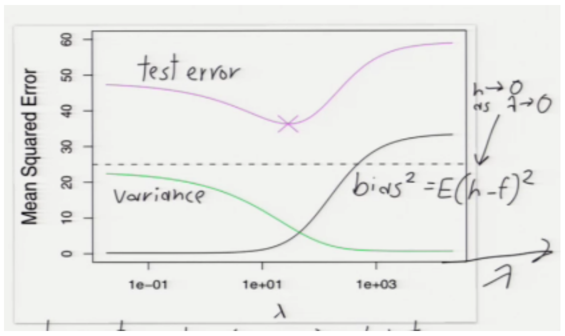
- Reduces over-fitting by reducing variance by penalizing large weights.


least squares sol'n
w_2
β_2
$\hat\beta$
one sol'n for a particular λ
isocontours of $|Xw-y|^2$
β_1 w_1

- Setting $\nabla J = 0$ gives equations,

$$(X^T X + \lambda I')w = X^T y$$

- where I' is identity matrix with bottom right set to zero.
- Algorithm: solve for w. Return $h(z) = w^T x$.
- Increasing $\lambda \Rightarrow$ more regularization; smaller $|w'|$.
- Given our data model $y = Xv + e$, where e is noise.
- Variance of ridge regression is $\mathrm{Var}(x^T (X^T X + \lambda I')^{-1} X^T e)$.
- As $\lambda \to \infty$, variance $\to 0$, but bias increases.



- λ is a hyper-parameter; tune by (cross)-validation.
- Ideally features should be "normalized" to have same variance.
- Alternative: Use asymmetric penalty by replacing I' with other diagonal matrix.

Bayesian justification for ridge regression

- Assign a prior probability on w': a Gaussian centered at 0.
- Posterior probability $\approx$ likelihood of $w \cdot$ prior $P(w') \leftarrow$ Gaussian PDF.
- Maximize the log posterior, ℓn likelihood + $\ell n P(w')$ = -const$|Xw - y|^2$ - const$|w'|^2$ - constant.
- This method (using likelihood, but maximizing posterior) is called maximum a posteriori (MAP).

Kernels

- Recall: with d input features, degree-p polynomials blow up to $\mathcal{O}(d^p)$ features.
- Today we use magic to use those features without computing them!
- Observation: In many learning algorithms:
  - The weights can be written as a linear combination of input samples.
  - We can use inner products of $\phi(x)'$s only $\Rightarrow$ don't need to compute $\phi(x)$!
  - Suppose $w = X^T a = \sum_{i=1}^n a_i X_i$ for some $a \in \mathbb{R}^n$.
  - Substitute this identity into algorithms and optimize n dual weights (aka dual parameters a, instead of $d+1$ primal weights w.

Kernel Ridge Regression

- Center X and y so their means are zero; $X_i \leftarrow X_i - \mu_x$. By centering the matrix we minimize the penalization of α
- This lets us replace I' with I in normal equations:

$$(X^T X + \lambda I)w = X^T y$$
$$\Rightarrow w = \frac{1}{\lambda}(X^T y - X^T X w) = X^T a \qquad \text{where } a = \frac{1}{\lambda}(y - Xw)$$

- This shows that w is a linear combination of samples. To compute a:

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

- a is the dual solution; solves the dual form of ridge regression:

$$\boxed{\text{Find } a \text{ that minimizes } |XX^T - y|^2 + \lambda |X^T a|^2}$$

- Regression function is:

$$h(z) = w^T z = a^T X z = \sum_{i=1}^n a_i (X_i^T z) \Leftarrow \text{weighted sum of inner products}$$

- Let $k(x,z) = x^T z$ be kernel function.
- Let $K = XX^T$ be nxn kernel matrix. Note $K_{ij} = k(X_i, X_j)$.
- K is singular if $n > d$. In that case no solution if $\lambda = 0$.
- Summary of kernel ridge regression:
  - Solve $(K + \lambda I)a = y$ for a $\Leftarrow \mathcal{O}(n^3)$ time.
  - $K_{ij} = k(X_i, X_j)\ \forall i, j \Leftarrow \mathcal{O}(n^2 d)$ time.
  - for each test point z: $h(z) = \sum_{i=1}^{n} a_i k(X_i, z) \Leftarrow \mathcal{O}(nd)$ time.
- Do not use X directly: only $k(\cdot,\cdot)$.
- Dual: solve nxn linear system.
- Primal: solve dxd linear system.

The Kernel Trick (aka kernelization)

- The polynomial kernel of degree p is $k(x,z) = (x^T z + 1)^p$.
- Theorem: $(x^T z + 1)^p = \phi(x)^T \phi(z)$ where $\phi(x)$ contains every monomial in x of degree $0 \dots p$.
- Example for $d = 2, p = 2$.

$$\begin{aligned}
(x^T z + 1)^2 &= x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1 z_1 x_2 z_2 + 2x_1 z_1 + 2x_2 z_2 + 1 \\
&= \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2x_1 x_2} & \sqrt{2x_1} & \sqrt{2x_2} & 1 \end{bmatrix} \cdot \begin{bmatrix} z_1^2 \\ z_2^2 \\ \sqrt{2z_1 z_2} \\ \sqrt{2z_1} \\ \sqrt{2z_2} \\ 1 \end{bmatrix} \\
&= \phi(x)^T \phi(z)
\end{aligned}$$

- Key win: compute $\phi(x)^T \phi(z)$ in $\mathcal{O}(d)$ time instead of $\mathcal{O}(d^p)$ even though $\phi(x)$ has length $\mathcal{O}(d^p)$.
- Kernel ridge regression replaces X_i with $\phi(X_i)$:
  - Let $k(x,z) = \phi(x)^T \phi(z)$.

03/09/2016

Kernels continued

- Kernel Perceptrons
- Note: Everywhere below, we can replace X_i with $\phi(X_i)$

Algorithm 5 Perceptron algorithm

```
while some $y_i X_i \cdot w < 0$ do
    $w \leftarrow w + \epsilon y_i x_i$
end while
while still have test points $z$ do
    $f(z) \leftarrow w^T x$
end while
```

- Kernelize with $w = X^T a$; $X_i \cdot w = (XX^T a)_i = (Ka)_i$.

Algorithm 6 Dual Perceptron algorithm

```
$a = [y_1, 0, \ldots, 0]^T$
$K_{ij} = K(X_i, X_j)\ \forall i, j$
while some $y_i (Ka)_i \cdot w < 0$ do
    $a \leftarrow a + \epsilon y_i$
end while
while still have test points $z$ do
    $f(z) \leftarrow \sum_{j=1}^{n} a_j K(X_j, z)$
end while
```

- Computing $K_{ij} \Leftarrow \mathcal{O}(n^2 d)$ time (kernel trick).
- Optimization for first while loop: maintain Ka, $\mathcal{O}(n)$ time.
- Second loop runs in $\mathcal{O}(nd)$ time or we can compute $w = X^T a$ once in $\mathcal{O}(nd')$ time and evaluate test points in $\mathcal{O}(d')$ time per point, where d' is the length of $\phi(\cdot)$.

Kernel Logistic Regression

- Stochastic gradient ascent step:

$$a_i \leftarrow a_i + \epsilon(y_i - s((Ka)_i))$$

- Gradient ascent step:

$$a \leftarrow a + \epsilon(y_i - s((Ka)) \Leftarrow \text{ applying s component-wise to vector } Ka$$

- for each test point z: $h(z) \leftarrow s(\sum_{j=1}^{n} a_j k(X_j, z))$.

The Gaussian Kernel

- Gaussian kernel, aka radial basis function kernel
- There exists a $\phi(x)$ such that:

$$k(x, z) = e^{(-\frac{|x-z|^2}{2\sigma^2})}$$

- Key observation: $h(z) = s(\sum_{j=1}^{n} a_j k(X_j, z))$ is a linear combination of Gaussian centered at samples.
- Very popular in practice:
  - Gives very smooth h
  - Behaves somewhat like k-nearest neighbor, but smoother.
  - Oscillates less than polynomial (depending on σ).
  - $k(x, z)$ can be interpreted as a "similarity measure" y value more influenced by points around it.
  - Gaussian is maximum when $x = z$, goes to 0 as distance increases.
  - Samples "vote" for value at z, but closer samples get bigger vote.
- σ trades off bias vs. variance:
  - larger $\sigma \to$ wider Gaussians, smoother $h \to$ more bias, less variance.
  - Choose by (cross)-validation.