Midterm review

Instructor: Alan Ritter

Midterm

- Tuesday
- Can use whole class time
- No book, notes.
- Can bring a calculator, but should be OK without one.

Problem 1

Consider the following data set on lung diseases. Your goal is to build a Naïve Bayes classifier that predicts whether a person has Bronchitis or Tuberculosis, given his/her symptoms.

Disease	X-ray Shadow	Dyspnea	Lung Inflamation
Bronchitis	Yes	Yes	Yes
Bronchitis	Yes	Yes	Yes
Bronchitis	No	No	Yes
Tuberculosis	Yes	Yes	No
Tuberculosis	Yes	No	No
Tuberculosis	No	No	Yes

1. *(1 point)* List the distributions that would be learned if you use a maximum likelihood estimate (MLE) to estimate the parameters of a Naïve Bayes model from this data (e.g. $P(\text{Dyspnea}|\text{Bronchitis}) = ?$). Include all of the parameters. Show your work.

2. *(1 point)* Based on your learned model, diagnose a patient with the following symptoms (show your work):

X-ray Shadow	Dyspnea	Lung Inflamation
Yes	No	Yes

Problem 2

Assume you are given a dataset of n real numbers $D = \{X^1, X^2, \ldots, X^n\}$, $X^i \in \mathcal{R}$. Derive the maximum likelihood mean, μ and variance, σ parameters of a 1-dimensional Gaussian distribution.

1. *(1 point)* Write down the log-likelihood of D as a function of μ and σ, $\mathcal{L}(\mu, \sigma)$.

2. *(1 point)* Compute the partial derivative of $\mathcal{L}(\mu, \sigma)$ with respect to μ, equate to zero and solve for μ.

3. *(1 point)* Compute the partial derivative of $\mathcal{L}(\mu, \sigma)$ with respect to σ, equate to zero and solve for σ.

Problem 3

Consider a training sample of inputs $X^1, X^2, \ldots, X^n$ and outputs $Y^1, Y^2, \ldots, Y^n$, where inputs are real-valued vectors $X^i \in \mathcal{R}^V$ and outputs are binary $Y^i \in [0, 1]$.

Recall (from the slides presented in class) that the conditional log likelihood can be written as follows:

$$\begin{aligned}
\mathcal{L}(W) &= \sum_l Y^l \log P(Y = 1|X^l, W) + (1 - Y^l) \log P(Y = 0|X^l, W) \\
&= \sum_l Y^l \log \frac{P(Y = 1|X^l, W)}{P(Y = 0|X^l, W)} + \log P(Y = 0|X^l, W) \\
&= \sum_l Y^l \left(w_0 + \sum_{i=1}^{V} w_i X_i^l \right) - \log \left(1 + \exp(w_0 + \sum_{i=1}^{V} w_i X_i^l) \right)
\end{aligned}$$

1. *(2 points)* Show that the partial derivative of $\mathcal{L}(W)$ with respect to w_i is as follows:

$$\frac{\partial \mathcal{L}(W)}{\partial w_i} = \sum_l X_i^l (Y^l - P(Y = 1|X^l, W))$$

2. *(2 points)* Now, assume a zero-mean Gaussian prior over the weights:

$$P(w_i) = \mathcal{N}(0, \sigma)$$

Write down the expression for the posterior distribution over w_i, and derive the gradient (e.g. that can be used for estimating MAP parameters in gradient decent).

Beta-Binomial Model

Maximum Likelihood Estimation Recipe

1. Use the log-likelihood
2. Differentiate with respect to the parameters
3. *Equate to zero and solve

*Often requires numerical approximation (no closed form solution)

An Example

- Let's start with the simplest possible case
  - Single observed variable
  - Flipping a bent coin

- We Observe:
  - Sequence of heads or tails
  - HTTTTTHTHT
- Goal:
  - Estimate the probability that the next flip comes up heads

Assumptions

- Fixed parameter θ_H
  - Probability that a flip comes up heads
- Each flip is independent
  - Doesn't affect the outcome of other flips
- (IID) Independent and Identically Distributed

Example

- Let's assume we observe the sequence:
  - HTTTTTTHTHT
- What is the **best** value of θ_H ?
  - Probability of heads
- Intuition: should be 0.3 (3 out of 10)
- Question: how do we justify this?

Maximum Likelihood Principle

- The value of θ_H which maximizes the probability of the observed data is best!
- Based on our assumptions, the probability of "HTTTTTTHTHT" is:

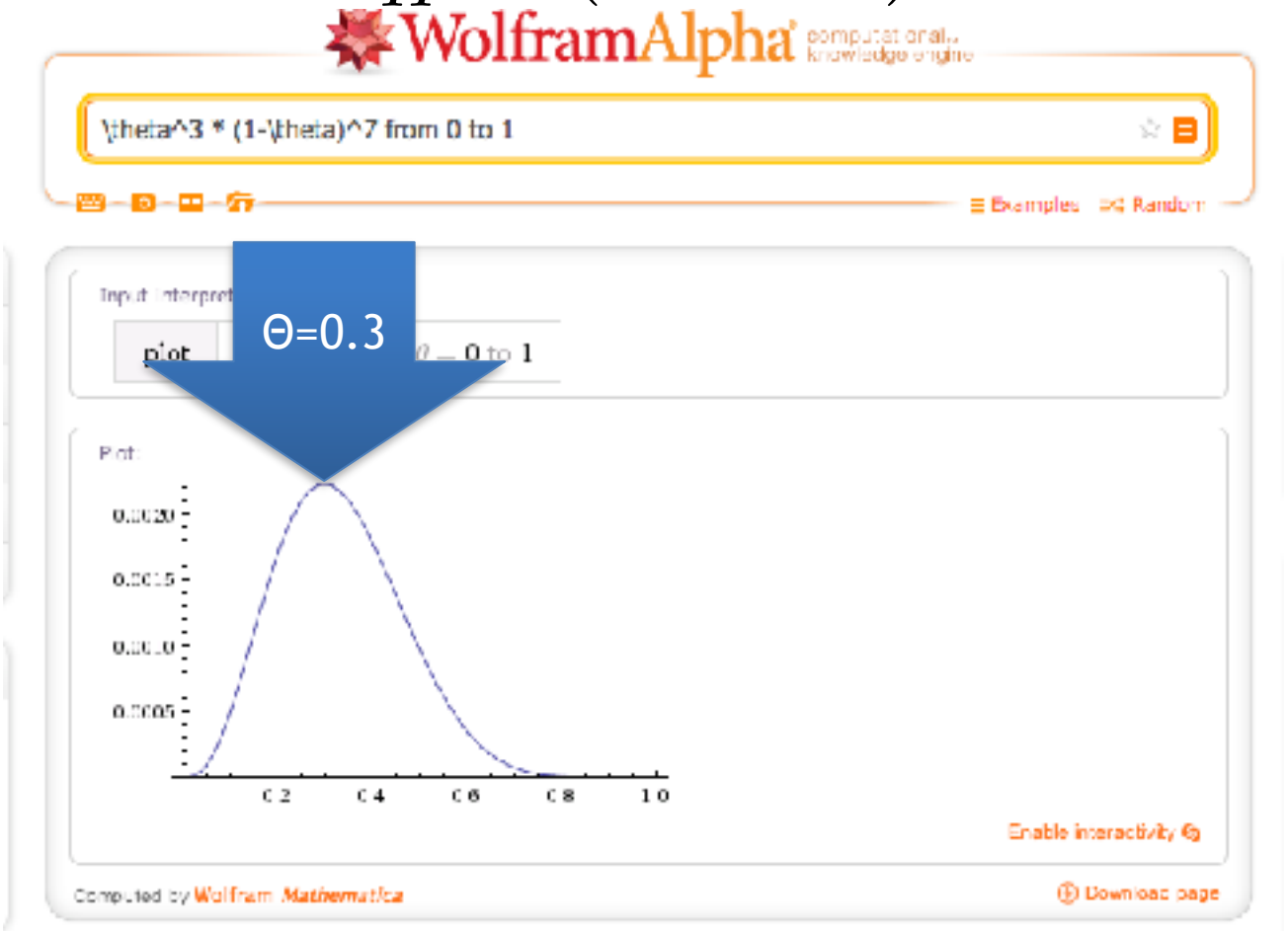
$$
\begin{aligned}
&P(x_1 = H, x_2 = T, \ldots, x_m = T; \theta_H) \\
&= P(x_1 = H; \theta_H) P(x_2 = T; \theta_H), \ldots P(x_m = T; \theta_H) \\
&= \theta_H \times (1 - \theta_H), \times \ldots \times \theta_H \\
&= \theta_H^3 \times (1 - \theta_H)^7
\end{aligned}
$$

This is the Likelihood Function

Maximum Likelihood Principle

- Probability of "HTTTTTTHTHT" as a function of θ_H

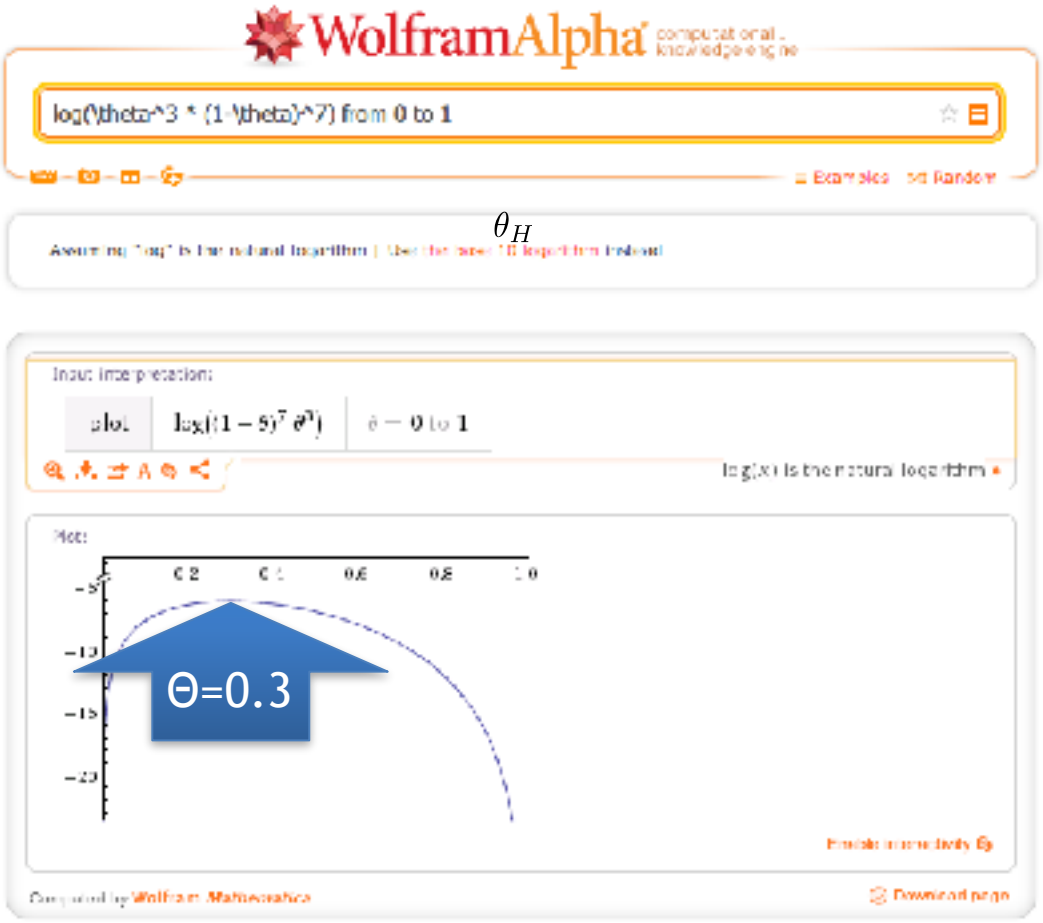
$$\theta_H^3 \times (1-\theta_H)^7$$



Maximum Likelihood Principle

- Probability of "HTTTTTTHTHT" as a function of θ_H

$$\log(\theta_H^3 \times (1-\theta_H)^7)$$



Maximum Likelihood value of θ_H

$$\frac{\partial}{\partial \theta_H} \log(\theta_H^{\#H}(1-\theta_H)^{\#T}) = 0$$

$$\frac{\partial}{\partial \theta_H} \log(\theta_H^{\#H}) + \log((1-\theta_H)^{\#T}) = 0$$

Log Identities

$$\frac{\partial}{\partial \theta_H} \#H \log(\theta_H) + \#T \log(1-\theta_H) = 0$$

Maximum Likelihood value of θ_H

$$\frac{\partial}{\partial \theta_H} \#H \log(\theta_H) + \#T \log(1 - \theta_H) = 0$$

$$\frac{\#H}{\theta_H} - \frac{\#T}{1 - \theta_H} = 0$$

$$\vdots$$

$$\hat{\theta} = \frac{\#H}{\#H + \#T}$$

Bayesian Parameter Estimation

- Let's just treat θ_H like any other variable
- Put a prior on it!
  - Encode our prior knowledge about possible values of θ_H using a probability distribution
- Now consider two probability distributions:

$$P(x_i|\theta_H) = \begin{cases} \theta_H, & \text{if } x_i = H \\ 1 - \theta_H, & \text{otherwise} \end{cases}$$

$$P(\theta_H) = ?$$

Posterior Over θ_H

$$P(\theta|x_1 = H, x_2 = T, \ldots, x_m = T)$$

$$= \frac{P(x_1 = H, x_2 = T, \ldots, x_m = T|\theta)P(\theta)}{P(x_1 = H, x_2 = T, \ldots, x_m = T)}$$

$$= \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$$

My rule is so cool! 😜

How can we encode prior knowledge?

- Example: The coin doesn't look very bent
  - Assign higher probability to values of θ_H near 0.5
- Solution: The **Beta Distribution**

$$P(\theta_H|\alpha,\beta) = \frac{1}{B(\alpha,\beta)}\theta_H^{\alpha-1}(1-\theta_H)^{\beta-1}$$

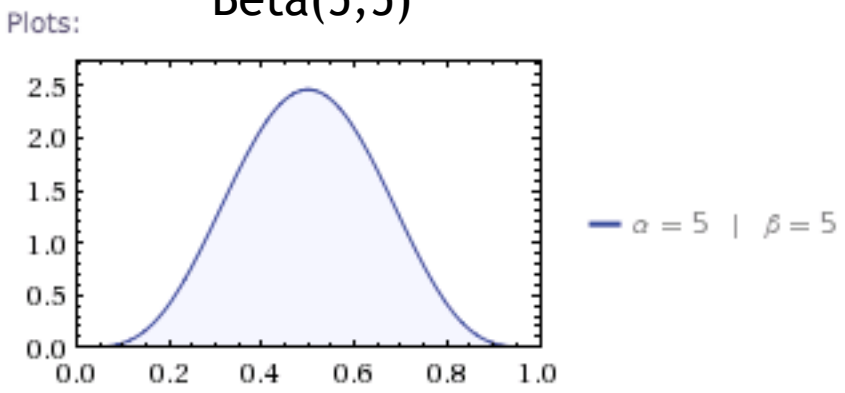
Hyper-Parameters

$$B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

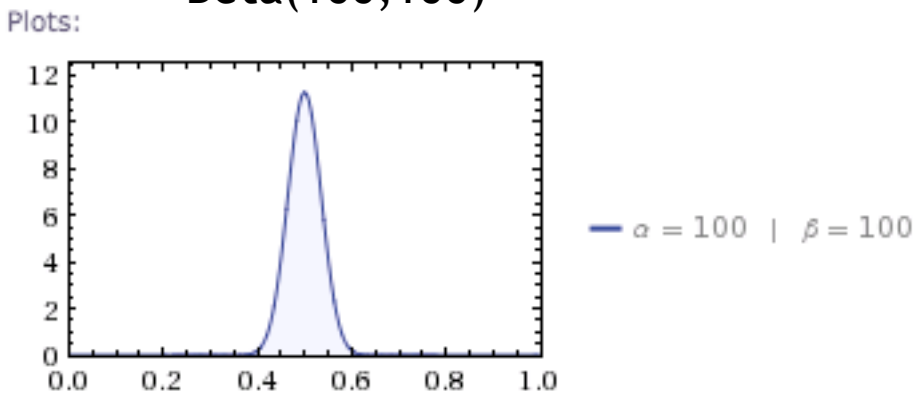
Gamma is a continuous generalization of the Factorial Function

Beta Distribution

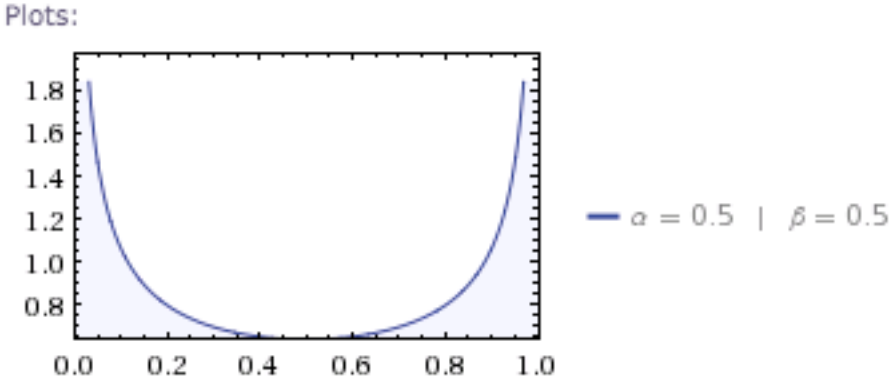
Beta(5,5)



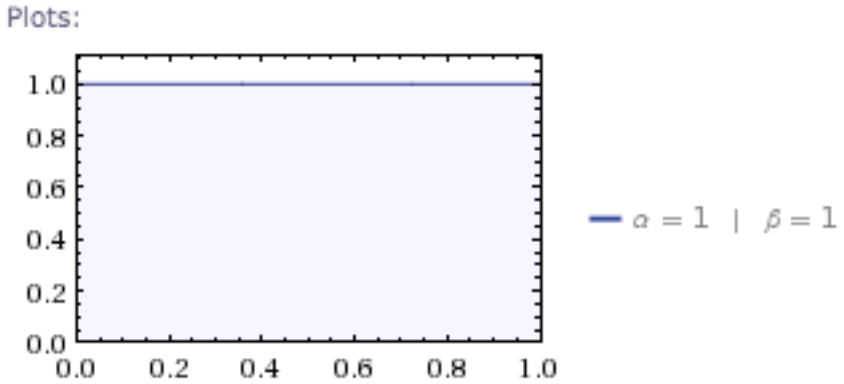
Beta(100,100)



Beta(0.5,0.5)



Beta(1,1)



MAP Estimate

$$\theta^{MAP} = \arg\max_{\theta} P(\theta|D)$$

$$= \frac{\#H + \alpha - 1}{\#T + \#H + \alpha + \beta - 2}$$

-Add-N smoothing
-Pseudo-counts

Logistic Regression

Logistic Regression

Idea:

- Naïve Bayes allows computing P(Y|X) by learning P(Y) and P(X|Y)

- Why not learn P(Y|X) directly?

- Consider learning f: X → Y, where
  - X is a vector of real-valued features, < X_1 … X_n >
  - Y is boolean
  - assume all X_i are conditionally independent given Y
  - model $P(X_i \mid Y = y_k)$ as Gaussian $N(\mu_{ik},\sigma_i)$
  - model P(Y) as Bernoulli (π)

- What does that imply about the form of P(Y|X)?

$$P(Y = 1|X =< X_1, ...X_n >) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

Derive form for $P(Y|X)$ for Gaussian $P(X_i|Y=y_k)$ assuming $\sigma_{ik} = \sigma_i$

$$P(Y=1|X) = \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1) + P(Y=0)P(X|Y=0)}$$

$$= \frac{1}{1 + \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}}$$

$$= \frac{1}{1 + \exp(\ln \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})}$$

$$= \frac{1}{1 + \exp(\ (\ln \frac{1-\pi}{\pi}) + \sum_i \ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)})}$$

$$P(x \mid y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} \; e^{\frac{-(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

$$\sum_i \left(\frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2} \right)$$

$$P(Y=1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

Very convenient!

$$P(Y = 1|X =< X_1, ...X_n >) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

implies

$$P(Y = 0|X =< X_1, ...X_n >) =$$

implies

$$\frac{P(Y = 0|X)}{P(Y = 1|X)} =$$

implies

$$\ln \frac{P(Y = 0|X)}{P(Y = 1|X)} =$$

Very convenient!

$$P(Y=1|X=<X_1,...X_n>)=\frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

implies

$$P(Y=0|X=<X_1,...X_n>)=\frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

implies

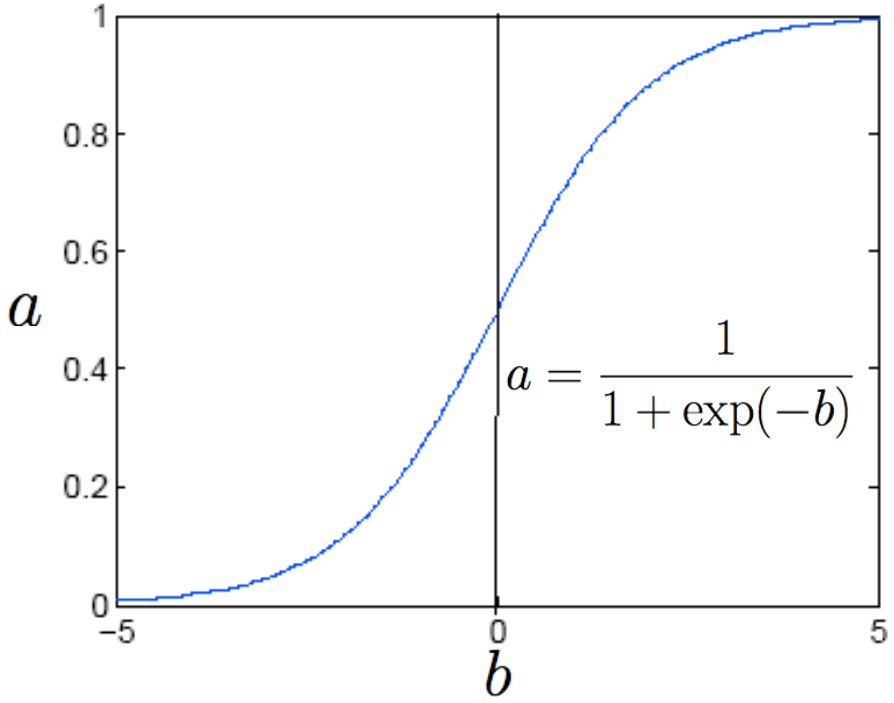
$$\frac{P(Y=0|X)}{P(Y=1|X)}=exp(w_0+\sum_i w_i X_i)$$

implies

$$\ln\frac{P(Y=0|X)}{P(Y=1|X)}=w_0+\sum_i w_i X_i$$

linear classification rule!

Logistic function



$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

Logistic regression more generally

- Logistic regression when Y not boolean (but still discrete-valued).
- Now $y \in \{y_1 \ldots y_R\}$: learn R sets of weights

$$P(y = y_k|X) = \frac{\exp(w_{k0} + \sum_{i=1}^{n} w_{ki}X_i)}{\sum_{j=1}^{R} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji}X_i)}$$

Training Logistic Regression: MCLE

- we have L training examples: $\{\langle X^1, Y^1\rangle, \ldots \langle X^L, Y^L\rangle\}$

- maximum likelihood estimate for parameters W

$$W_{MLE} = \arg\max_W P(< X^1, Y^1 > \ldots < X^L, Y^L > |W)$$
$$= \arg\max_W \prod_l P(< X^l, Y^l > |W)$$

- maximum conditional likelihood estimate

Training Logistic Regression: MCLE

- Choose parameters W=<w_0, ... w_n> to maximize conditional likelihood of training data

where

$$P(Y = 0|X, W) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

$$P(Y = 1|X, W) = \frac{exp(w_0 + \sum_i w_i X_i)}{1 + exp(w_0 + \sum_i w_i X_i)}$$

- Training data D = $\{\langle X^1, Y^1\rangle, \ldots \langle X^L, Y^L\rangle\}$
- Data likelihood = $\prod_l P(X^l, Y^l|W)$
- Data conditional likelihood = $\prod_l P(Y^l|X^l, W)$

$$W_{MCLE} = \arg\max_W \prod_l P(Y^l|W, X^l)$$

Expressing Conditional Log Likelihood

$$l(W) \equiv \ln \prod_l P(Y^l|X^l, W) = \sum_l \ln P(Y^l|X^l, W)$$

$$P(Y = 0|X, W) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

$$P(Y = 1|X, W) = \frac{exp(w_0 + \sum_i w_i X_i)}{1 + exp(w_0 + \sum_i w_i X_i)}$$

$$\begin{aligned} l(W) &= \sum_l Y^l \ln P(Y^l = 1|X^l, W) + (1 - Y^l) \ln P(Y^l = 0|X^l, W) \\ &= \sum_l Y^l \ln \frac{P(Y^l = 1|X^l, W)}{P(Y^l = 0|X^l, W)} + \ln P(Y^l = 0|X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l)) \end{aligned}$$

Maximizing Conditional Log Likelihood

$$P(Y = 0|X, W) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

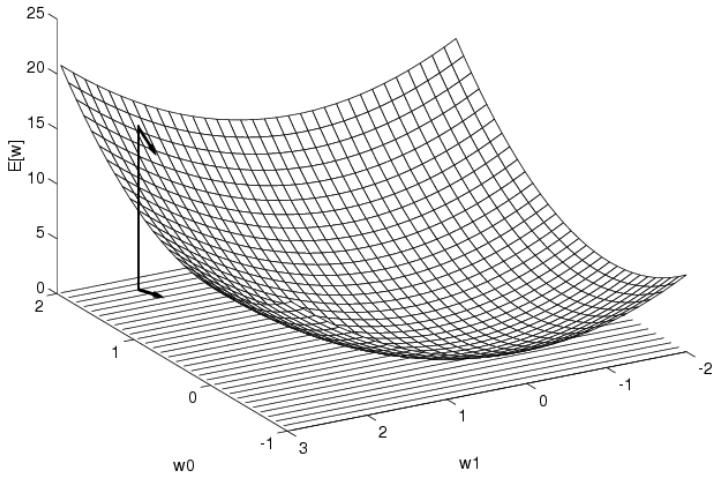
$$P(Y = 1|X, W) = \frac{exp(w_0 + \sum_i w_i X_i)}{1 + exp(w_0 + \sum_i w_i X_i)}$$

$$\begin{aligned} l(W) &\equiv \ln \prod_l P(Y^l|X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l)) \end{aligned}$$

Good news: $l(W)$ is concave function of W

Bad news: no closed-form solution to maximize $l(W)$

Gradient Descent



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Maximize Conditional Log Likelihood: Gradient Ascent

$$
\begin{aligned}
l(W) &\equiv \ln \prod_l P(Y^l|X^l, W) \\
&= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l))
\end{aligned}
$$

$$
\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))
$$

Maximize Conditional Log Likelihood: Gradient Ascent

$$l(W) \equiv \ln \prod_l P(Y^l|X^l, W)$$

$$= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l))$$

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

Gradient ascent algorithm: iterate until change < ε

For all i, repeat

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

That's all for M(C)LE. How about MAP?

- One common approach is to define priors on W
  - Normal distribution, zero mean, identity covariance
- Helps avoid very large weights and overfitting
- MAP estimate

$$W \leftarrow \arg\max_{W} \ln P(W) \prod_{l} P(Y^l | X^l, W)$$

- let's assume Gaussian prior: W ~ N(0, σ)

MLE vs MAP

- Maximum conditional likelihood estimate

$$W \leftarrow \arg\max_W \ \ln \prod_l P(Y^l|X^l, W)$$

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

- Maximum a posteriori estimate with prior W~N(0,σI)

$$W \leftarrow \arg\max_W \ \ln[P(W) \ \prod_l P(Y^l|X^l, W)]$$

$$w_i \leftarrow w_i - \eta\lambda w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

MAP estimates and Regularization

- Maximum a posteriori estimate with prior W~N(0,σI)

$$W \leftarrow \arg\max_W \ \ln[P(W) \prod_l P(Y^l|X^l,W)]$$

$$w_i \leftarrow w_i - \eta\lambda w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l,W))$$

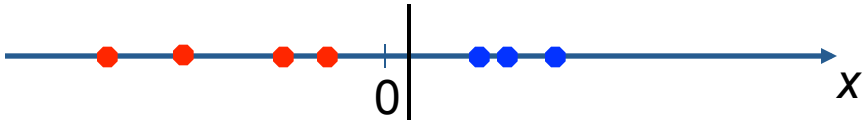
called a "regularization" term

- helps reduce overfitting
- keep weights nearer to zero (if P(W) is zero mean Gaussian prior), or whatever the prior suggests
- used very frequently in Logistic Regression

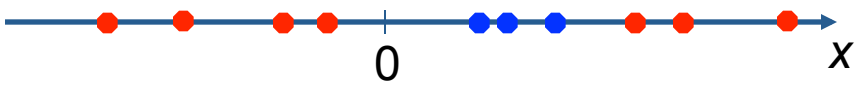
Kernel Methods

Non-linear features: 1D input

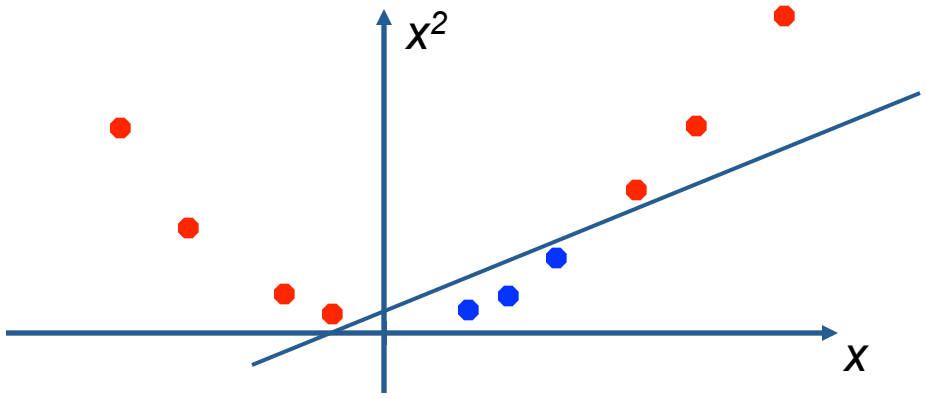
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?



- How about… mapping data to a higher-dimensional space:

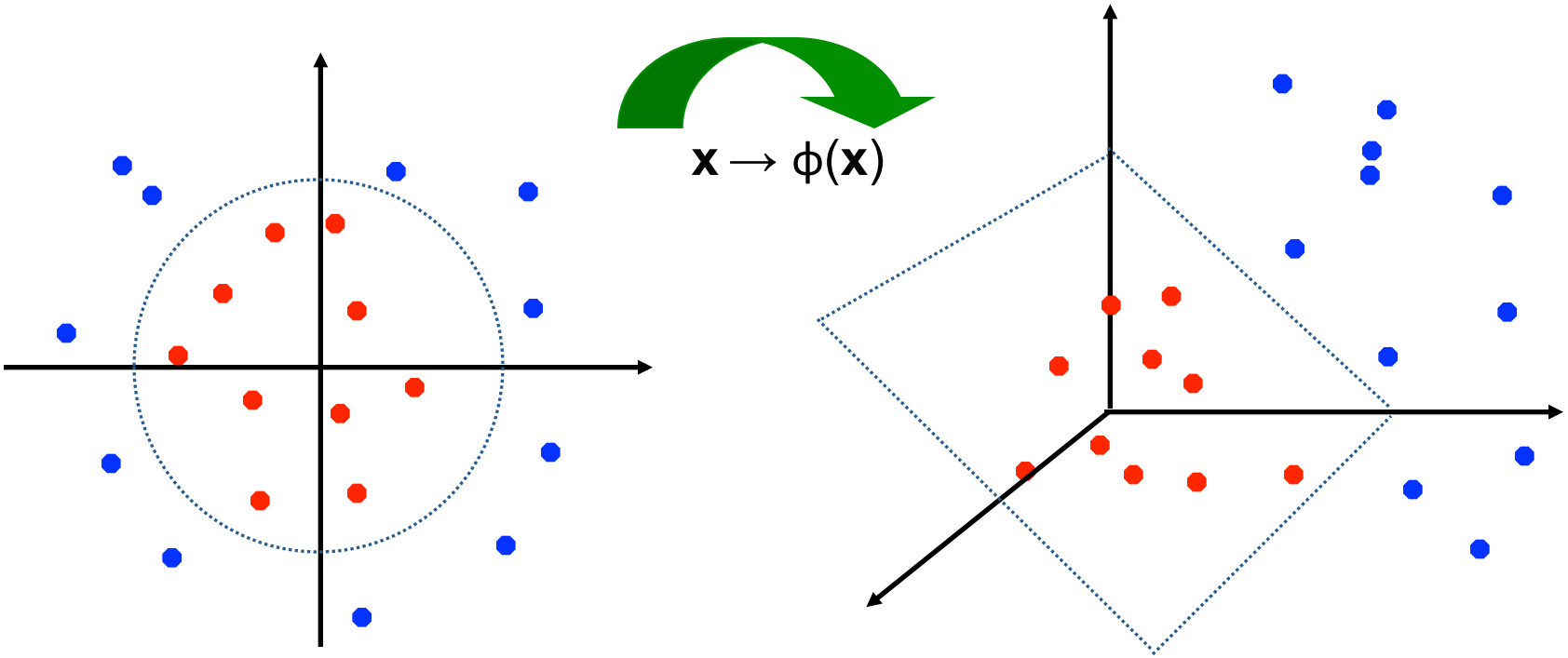


Feature spaces

- General idea: map to higher dimensional space
  - if **x** is in R^n, then $\phi(\mathbf{x})$ is in R^m for m>n
  - Can now learn feature weights **w** in R^m and predict:

$$y = sign(\mathbf{w} \cdot \phi(\mathbf{x}))$$

  - Linear function in the higher dimensional space will be non-linear in the original space



Efficient dot-product of polynomials

Polynomials of degree exactly *d*

d=1

$$\phi(u).\phi(v) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} . \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1v_1 + u_2v_2 = u.v$$

d=2

$$\phi(u).\phi(v) = \begin{pmatrix} u_1^2 \\ u_1u_2 \\ u_2u_1 \\ u_2^2 \end{pmatrix} . \begin{pmatrix} v_1^2 \\ v_1v_2 \\ v_2v_1 \\ v_2^2 \end{pmatrix} = u_1^2v_1^2 + 2u_1v_1u_2v_2 + u_2^2v_2^2$$
$$= (u_1v_1 + u_2v_2)^2$$
$$= (u.v)^2$$

For any *d (we will skip proof):*

$$K(u, v) = \phi(u).\phi(v) = (u.v)^d$$

- Cool! Taking a dot product and an exponential gives same results as mapping into high dimensional space and then taking dot product

The "Kernel Trick"

- A *kernel function* defines a dot product in some feature space.

$$K(\mathbf{u},\mathbf{v}) = \boldsymbol{\phi}(\mathbf{u}) \bullet \boldsymbol{\phi}(\mathbf{v})$$

- Example:

  2-dimensional vectors $\mathbf{u}=[u_1 \;\; u_2]$ and $\mathbf{v}=[v_1 \;\; v_2]$; let $K(\mathbf{u},\mathbf{v})=(1 + \mathbf{u}\bullet\mathbf{v})^2$,

  Need to show that $K(\mathbf{x}_i,\mathbf{x}_j)= \boldsymbol{\phi}(\mathbf{x}_i) \bullet \boldsymbol{\phi}(\mathbf{x}_j)$:

  $K(\mathbf{u},\mathbf{v})=(1 + \mathbf{u}\bullet\mathbf{v})^2 = 1+ u_1^2v_1^2 + 2\, u_1v_1\, u_2v_2 + u_2^2v_2^2 + 2u_1v_1 + 2u_2v_2 =$

  $= [1,\ u_1^2,\ \sqrt{2}\, u_1u_2,\ u_2^2,\ \sqrt{2}u_1,\ \sqrt{2}u_2] \bullet [1,\ v_1^2,\ \sqrt{2}v_1v_2,\ v_2^2,\ \sqrt{2}v_1,\ \sqrt{2}v_2] =$

  $= \boldsymbol{\phi}(\mathbf{u}) \bullet \boldsymbol{\phi}(\mathbf{v})$, where $\boldsymbol{\phi}(\mathbf{x}) = [1,\ x_1^2,\ \sqrt{2}\, x_1x_2,\ x_2^2,\ \sqrt{2}x_1,\ \sqrt{2}x_2]$

- Thus, a kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each $\boldsymbol{\phi}(\mathbf{x})$ explicitly).
- But, it isn't obvious yet how we will incorporate it into actual learning algorithms…

"Kernel trick" for The Perceptron!

- Never compute features explicitly!!!
  - Compute dot products in closed form K(u,v) = Φ(u) • Φ(v)
- Standard Perceptron:
  - set w_i=0 for each feature i
  - set a^i=0 for each example i
  - For t=1..T, i=1..n:
    - $y = sign(w \cdot \phi(x^i))$
    - if y ≠ y^i
      - $w = w + y^i \phi(x^i)$
      - a^i += y^i
  - At all times during learning:

$$w = \sum_k a^k \phi(x^k)$$

- Kernelized Perceptron:
  - set a^i=0 for each example i
  - For t=1..T, i=1..n:
    - $y = sign((\sum_k a^k \phi(x^k)) \cdot \phi(x^i))$

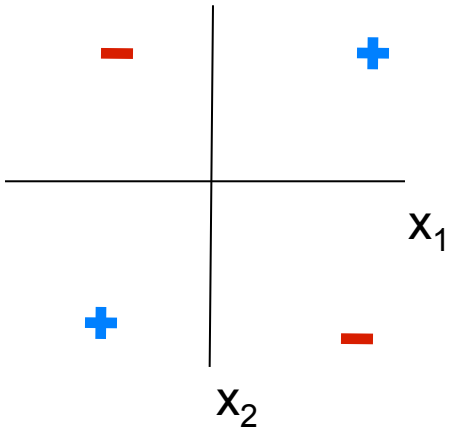
$$= sign(\sum_k a^k K(x^k, x^i))$$

    - if y ≠ y^i
      - a^i += y^i

Exactly the same computations, but can use K(u,v) to avoid enumerating the features!!!

- set a^i=0 for each example i
- For t=1..T, i=1..n:
  - $y = sign(\sum_k a^k K(x^k, x^i))$
  - if y ≠ y^i
    - a^i += y^i

x_1	x_2	y
1	1	1
-1	1	-1
-1	-1	1
1	-1	-1



$K(u,v) = (u \bullet v)^2$

e.g.,

$K(x^1,x^2)$

$= K([1,1],[-1,1])$

$= (1x-1+1x1)^2$

$= 0$

K	x^1	x^2	x^3	x^4
x^1	4	0	4	0
x^2	0	4	0	4
x^3	4	0	4	0
x^4	0	4	0	4

Initial:

- a = [a^1, a^2, a^3, a^4] = [0,0,0,0]

t=1,i=1

- $\Sigma_k a^k K(x^k,x^1)$ = 0x4+0x0+0x4+0x0 = 0, sign(0)=-1
- a^1 += y^1 → a^1+=1, new a= [1,0,0,0]

t=1,i=2

- $\Sigma_k a^k K(x^k,x^2)$ = 1x0+0x4+0x0+0x4 = 0, sign(0)=-1

t=1,i=3

- $\Sigma_k a^k K(x^k,x^3)$ = 1x4+0x0+0x4+0x0 = 4, sign(4)=1

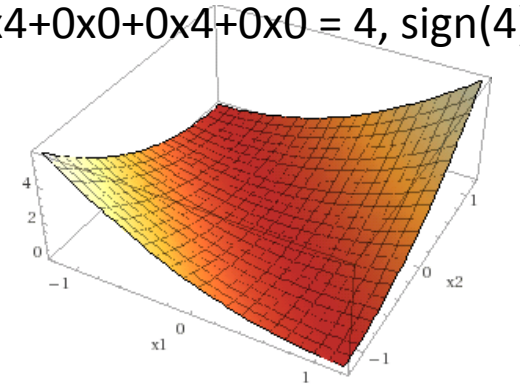
t=1,i=4

- $\Sigma_k a^k K(x^k,x^4)$ = 1x0+0x4+0x0+0x4 = 0, sign(0)=-1

t=2,i=1

- $\Sigma_k a^k K(x^k,x^1)$ = 1x4+0x0+0x4+0x0 = 4, sign(4)=1

…

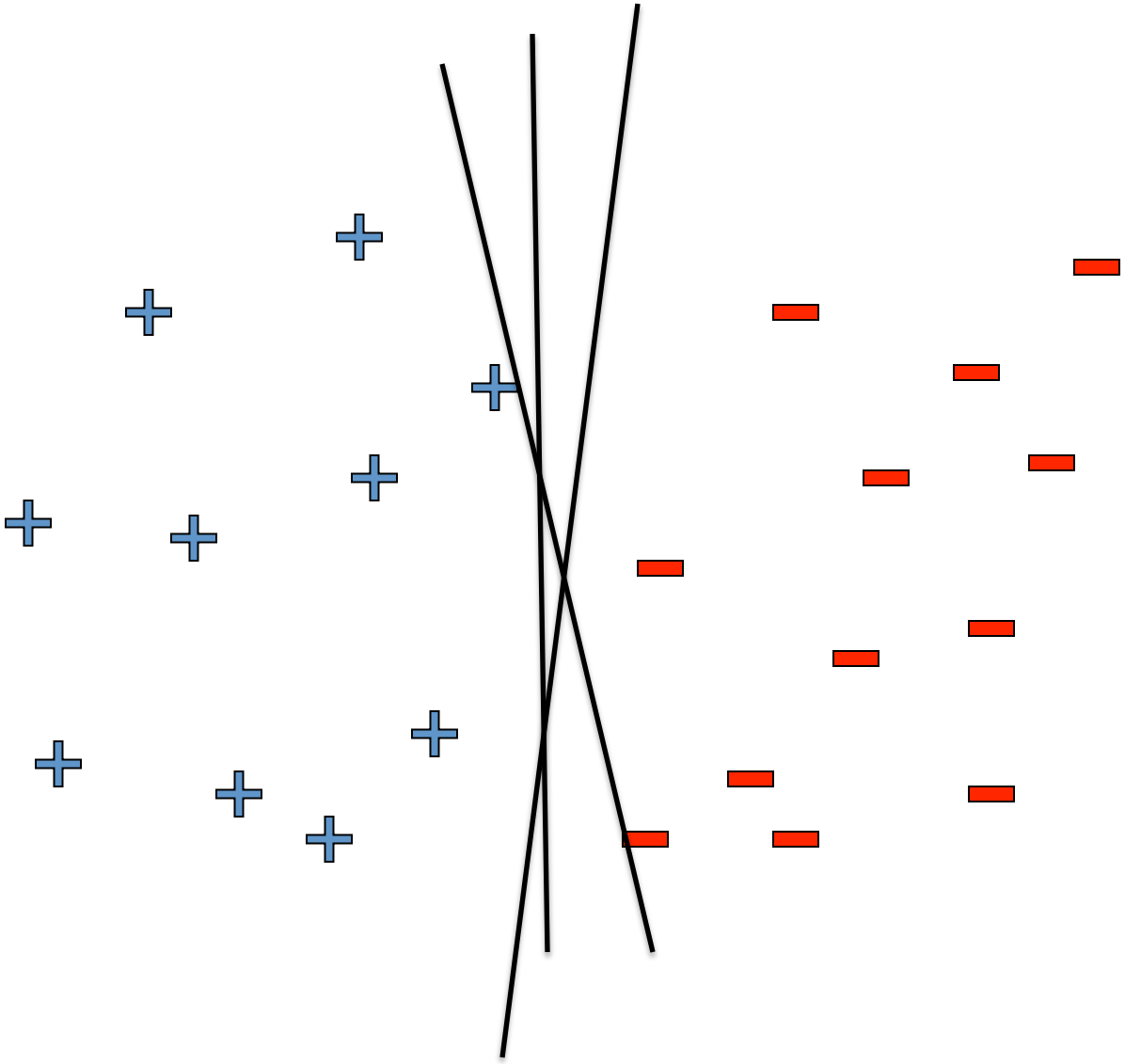


Converged!!!

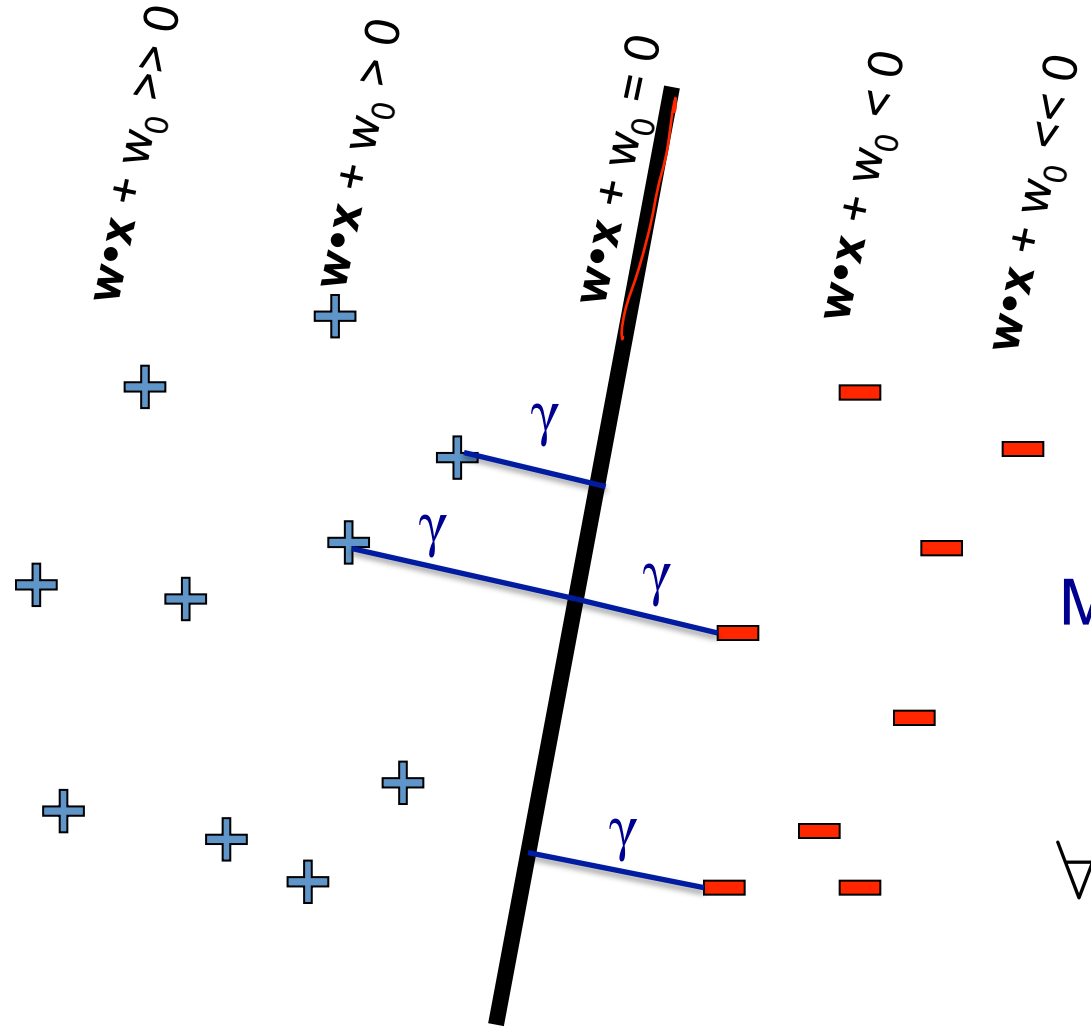
- $y=\Sigma_k a^k K(x^k,x)$
  $= 1\times K(x^1,x)+0\times K(x^2,x)+0\times K(x^3,x)+0\times K(x^4,x)$
  $= K(x^1,x)$
  $= K([1,1],x)$ (because x^1=[1,1])
  $= (x_1+x_2)^2$ (because $K(u,v) = (u \bullet v)^2$)

Support Vector Machines

Linear classifiers – Which line is better?



Pick the one with the largest margin!



$\mathbf{w \bullet x} = \sum_i w_i x_i$

Margin for point j:

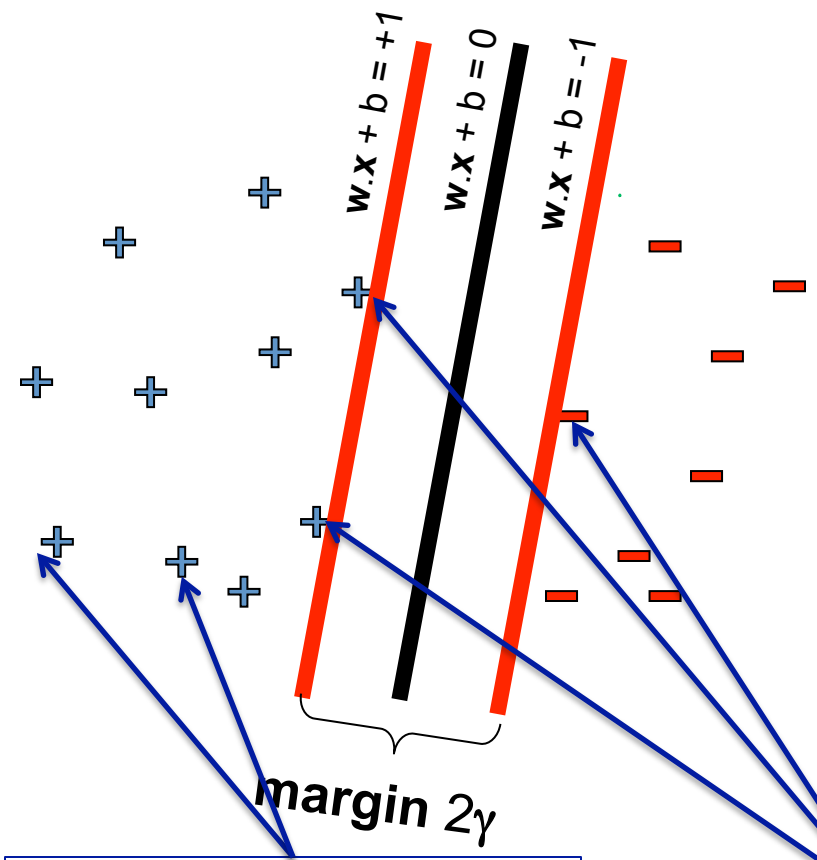
$$\gamma^j = y^j (w \cdot x^j + w_0)$$

Max Margin:

$$\max_{\gamma, w, w_0} \gamma$$
$$\forall j . y^j (w \cdot x^j + w_0) > \gamma$$

Hard-margin SVM

Support vector machines (SVMs)



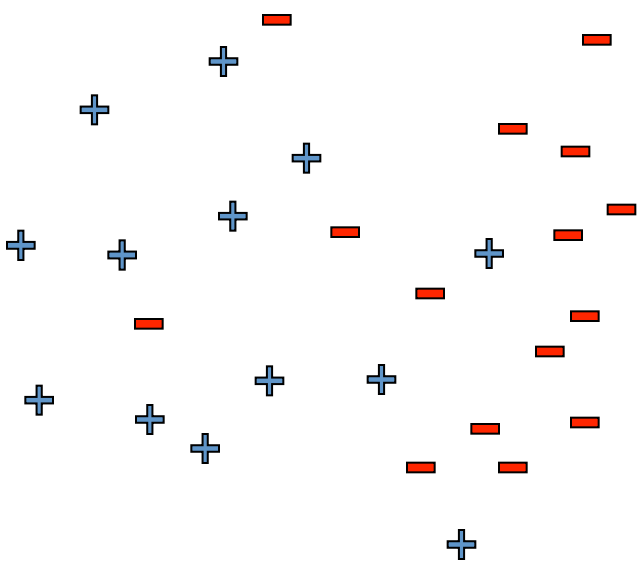
$$\min_{w,w_0} \frac{1}{2}\|w\|_2^2$$
$$\forall j.y^j(w \cdot x^j + w_0) \geq 1$$

- Solve efficiently by quadratic programming (QP)
  - Well-studied solution algorithms
  - Not simple gradient ascent, but close
- Decision boundary defined by support vectors

Non-support Vectors:
- everything else
- moving them will not change w

Support Vectors:
- data points on the canonical lines

What if the data is still not linearly separable?

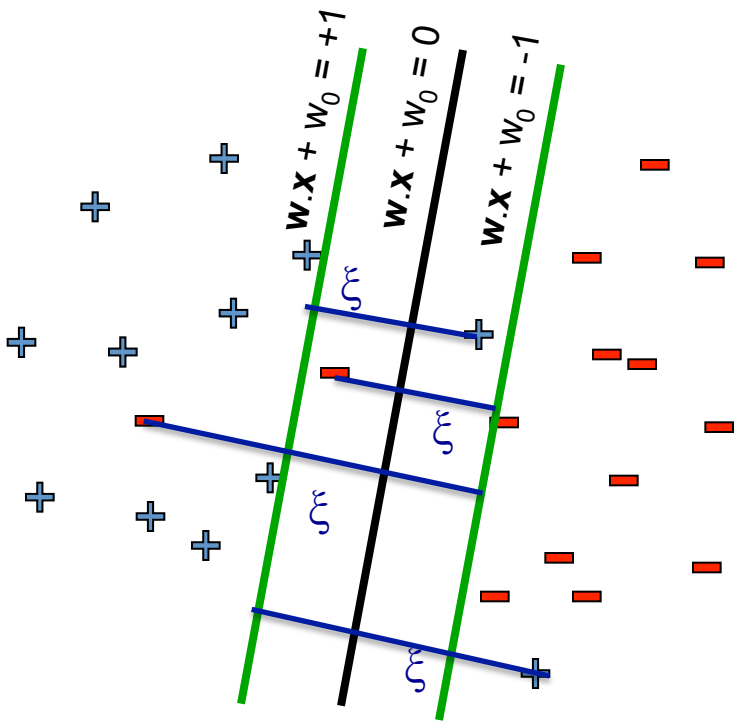


$$\min_{w,w_0} \frac{1}{2}\|w\|_2^2 \text{ + C \#(mistakes)}$$

$$\forall j. y^j (w \cdot x^j + w_0) \geq 1$$

- First Idea: Jointly minimize $\|w\|_2^2$ and number of training mistakes
  - How to tradeoff two criteria?
  - Pick C on development / cross validation
- Tradeoff #(mistakes) and $\|w\|_2^2$
  - 0/1 loss
  - Not QP anymore
  - Also doesn't distinguish near misses and really bad mistakes
  - NP hard to find optimal solution!!!

Slack variables – Hinge loss



$$\min_{w,w_0} \frac{1}{2}\|w\|_2^2 \; + C\,\Sigma_j\,\xi^j$$

$$\forall j.y^j(w \cdot x^j + w_0) \geq 1 - \xi^j \quad , \xi^j \geq 0$$

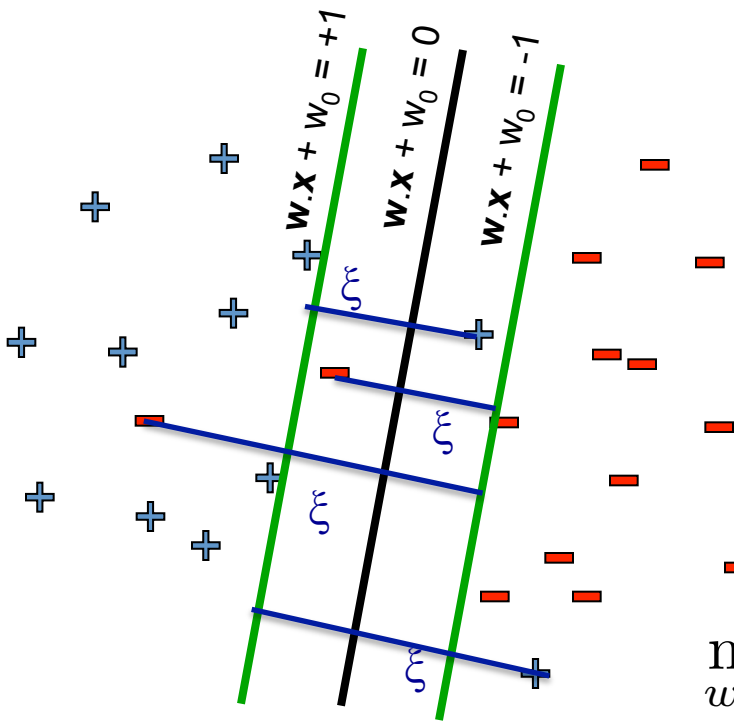
Slack Penalty $C > 0$:

- $C=\infty$ → have to separate the data!
- $C=0$ → ignore data entirely!
- Select on dev. set, etc.

For each data point:

- If margin ≥ 1, don't care
- If margin < 1, pay linear penalty

Slack variables – Hinge loss



$$\min_{w,w_0} \frac{1}{2}\|w\|_2^2 \; + C\,\Sigma_j\,\xi^j$$

$$\forall j.y^j(w\cdot x^j + w_0) \geq 1 - \xi^j \quad , \xi^j \geq 0$$

$[x]_+ = \max(x,0)$

$$\min_{w,w_0} \frac{1}{2}\|w\|_2^2 + C\sum_{j=1}^{N}[1 - y^j(w\cdot x^j + w_0)]_+$$

Regularization

Hinge Loss

Solving SVMs:

- Differentiate and set equal to zero!
- No closed form solution, but quadratic program (top) is concave
- Hinge loss is not differentiable, gradient ascent a little trickier…

SVMs vs Regularized Logistic Regression

$$f(x) = w_0 + \sum_i w_i x_i$$

SVM Objective:

$$\arg\min_{\mathbf{w}, w_0} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{j=1}^{N}[1 - y^j f(x^j)]_+$$

$[x]_+ = \max(x,0)$

Logistic regression objective:

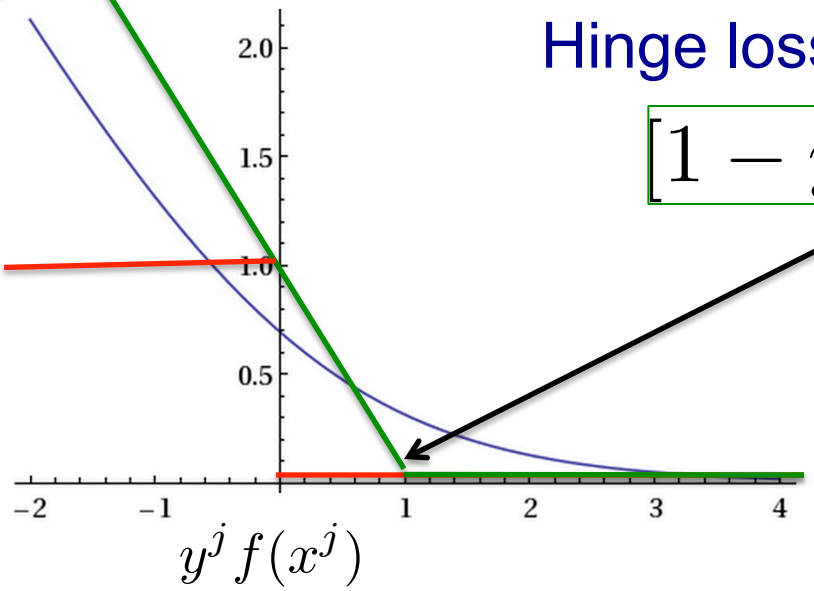
$$\arg\min_{\mathbf{w}, w_o} \lambda\|\mathbf{w}\|_2^2 + \sum_{j=1}^{N}\ln(1 + \exp(-y^j f(x^j)))$$

Tradeoff: same l_2 regularization term, but different error term

Graphing Loss vs Margin

Logistic regression:

$$\ln(1+\exp(-y^j f(x^j)))$$



Hinge loss:

$$[1-y^j f(x^j)]_+$$

0-1 Loss:

$$\delta(f(x^j) \neq y^j)$$

We want to smoothly approximate 0/1 loss!

Summary

- HW#3
- Beta/Binomial
- MAP vs. MLE
- Logistic Regression
- SGD
- Kernel Methods
- SVMs