Logistic Regression

Machine Learning – CSEP546

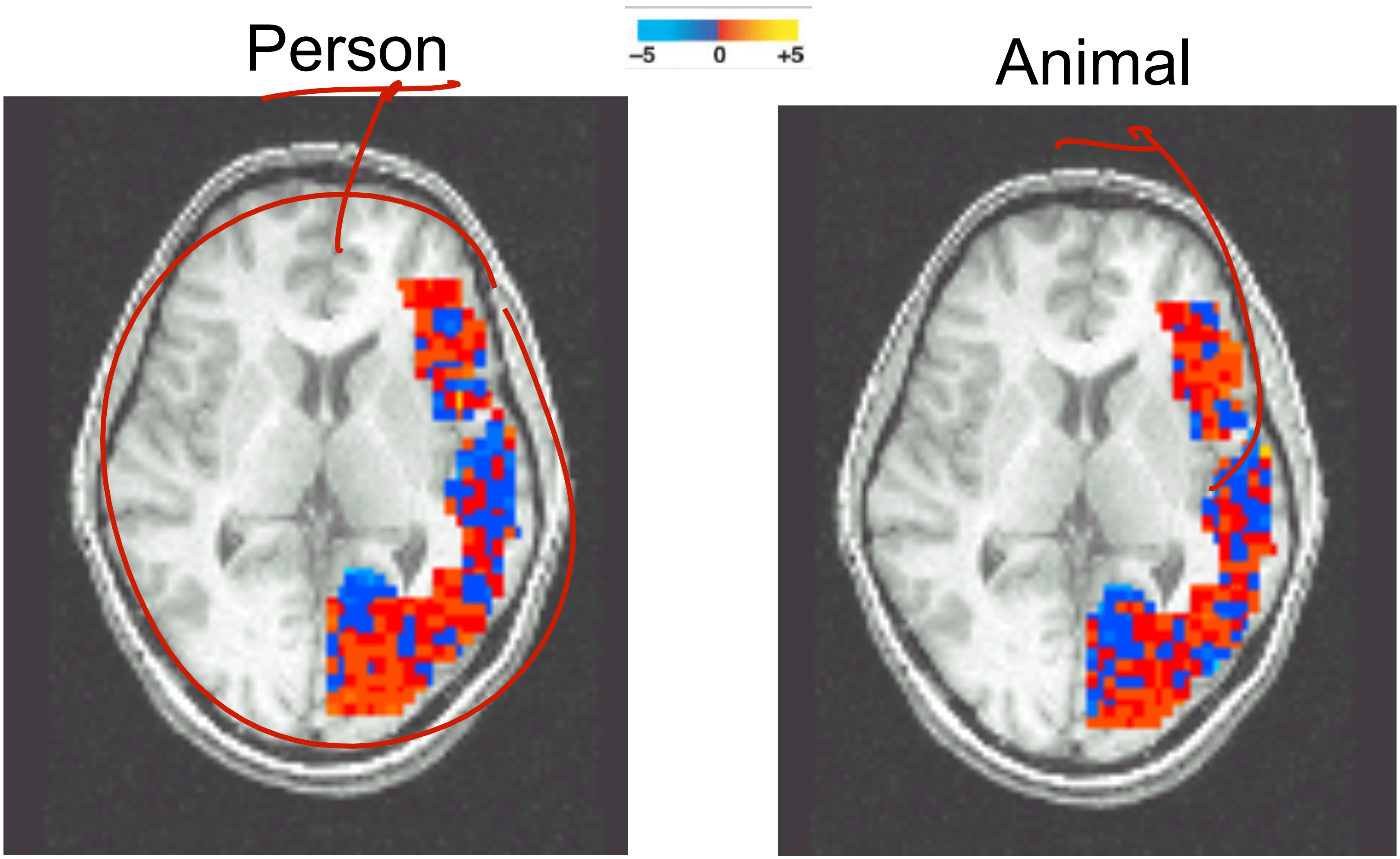
Carlos Guestrin
University of Washington

January 27, 2014

Reading Your Brain, Simple Example

[Mitchell et al.]

Pairwise classification accuracy: 85%



Classification

- **Learn**: h:**X** ↦ Y
  - **X** – features
  - Y – target classes

X = (GPA, grade, resume, ...)

Y = {hired, not hired}

- **Simplest case: Thresholding**

X: Load Computer

Y: alarm?

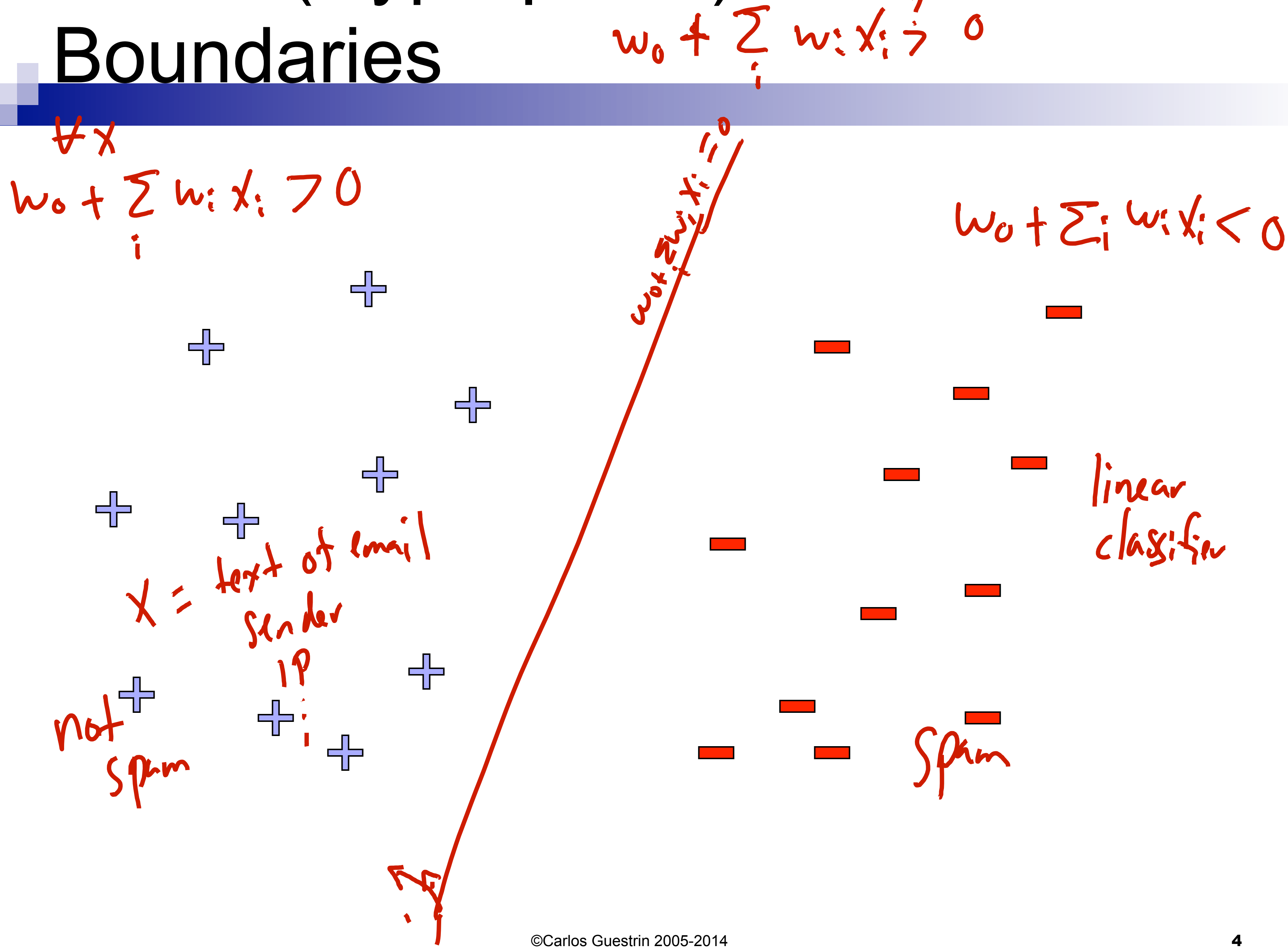
Load > 99% ⇒ alarm = true

x_i

else ⇒ alarm = false

$x_j > 27°C$

Linear (Hyperplane) Decision Boundaries



Classification

- **Learn**: h:**X** $\mapsto$ Y
  - **X** – features
  - Y – target classes

$\hat{y} = \text{sign}(w \cdot x) \leftarrow$ yes/no decision

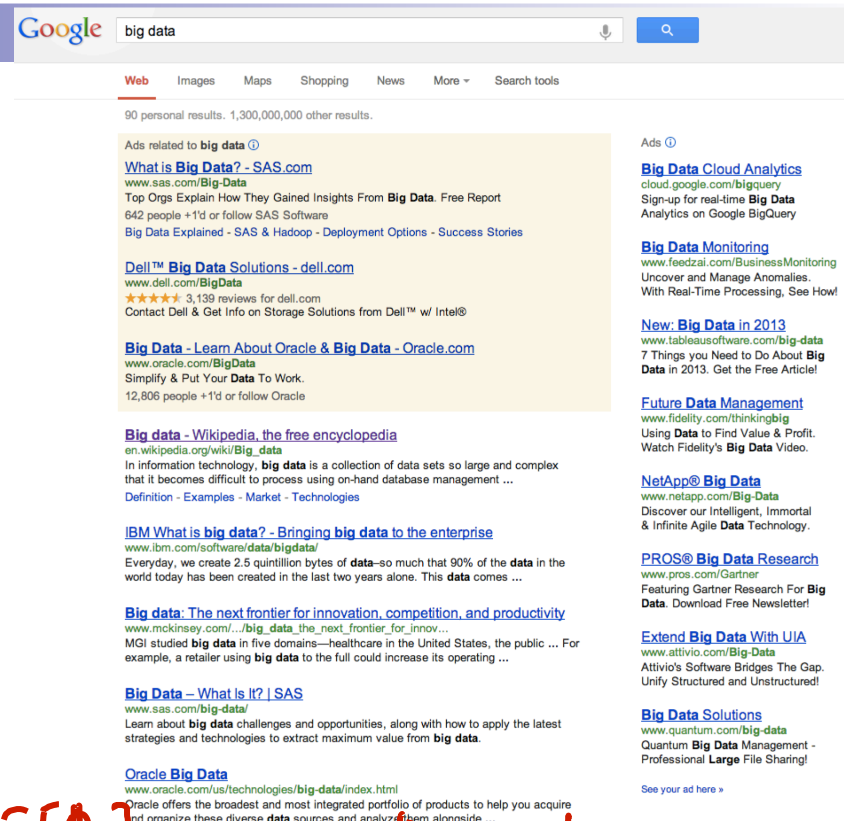
- Thus far: just a decision boundary

- What if you want probability of each class? P(Y|X)

$P(Y = \text{spam} \mid X = \text{text of email})$

$\hat{y} = \arg\max_y P(Y = y \mid X = \text{text of email})$

Ad Placement Strategies



- Companies bid on ad prices
  - $C_1 \to \$10$
  - $C_2 \to \$20$
  - $C_3 \to \$100$
- Which ad wins? (many simplifications here)
  - Naively: $C_3 \to \$100$
  - But: paid on click only
  - Instead:

e.g. $p(\text{click} \mid C_3, \text{'bigdata'}) = 0.01 \Rightarrow E[\$_3] = 0.01 * \$100 = \$1$

$p(\text{click} \mid C_1, \text{'bigdata'}) = 0.5 \Rightarrow E[\$_1] = 0.5 * \$10 = \$5$

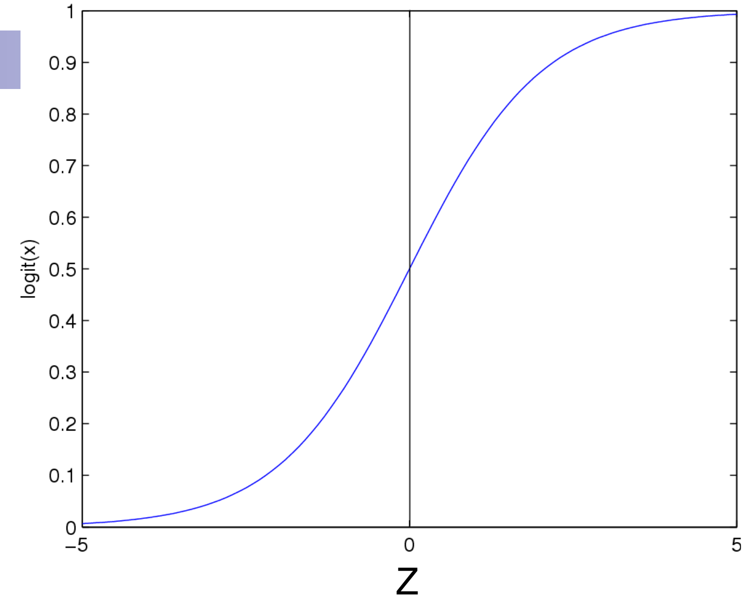
Link Functions

- Estimating P(Y|**X**): Why not use standard linear regression?

- Combing regression and probability?
  - Need a mapping from real values to [0,1]
  - A link function!

Logistic Regression

Logistic function (or Sigmoid): $\frac{1}{1+exp(-z)}$



- Learn P(Y|**X**) directly
  - Assume a particular functional form for link function
  - Sigmoid applied to a linear function of the input features:

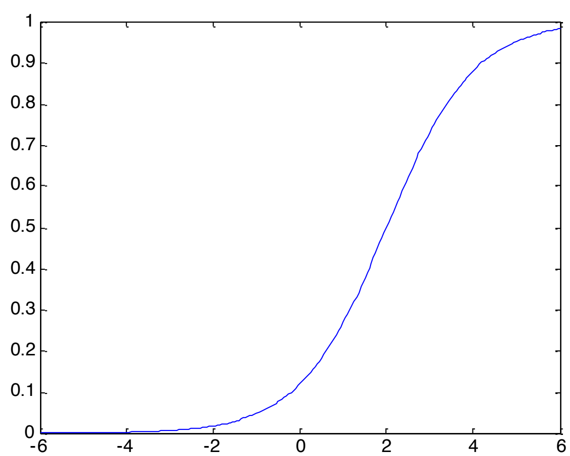
$$P(Y=0|X,W) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

Features can be discrete or continuous!

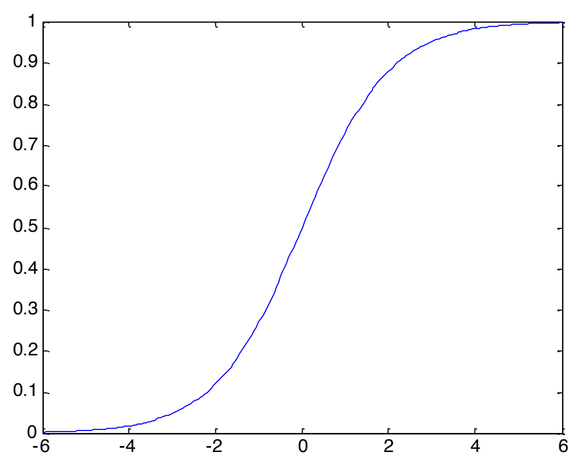
Understanding the sigmoid

$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{w_0 + \sum_i w_i x_i}}$$

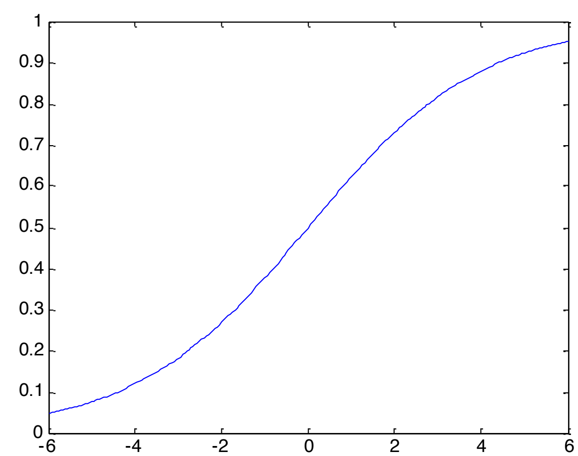
w_0=-2, w_1=-1



w_0=0, w_1=-1

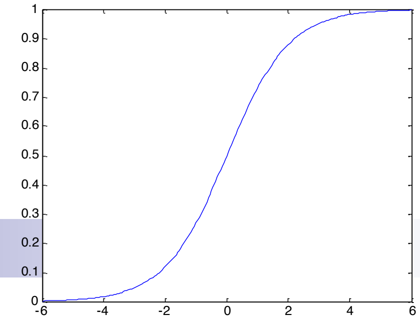


w_0=0, w_1=-0.5



Logistic Regression – a Linear classifier

$$\frac{1}{1 + exp(-z)}$$



$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{w_0 + \sum_i w_i x_i}}$$

Very convenient!

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

implies

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

implies

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

implies

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

linear classification rule!

Loss function: Conditional Likelihood

- Have a bunch of iid data of the form:

- Discriminative (logistic regression) loss function:
  Conditional Data Likelihood

$$\ln P(\mathcal{D}_Y \mid \mathcal{D}_{\mathbf{X}}, \mathbf{w}) = \sum_{j=1}^{N} \ln P(y^j \mid \mathbf{x}^j, \mathbf{w})$$

Expressing Conditional Log Likelihood

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

$$l(\mathbf{w}) \equiv \sum_j \ln P(y^j|\mathbf{x}^j,\mathbf{w})$$

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

$$\ell(\mathbf{w}) = \sum_j y^j \ln P(Y=1|\mathbf{x}^j,\mathbf{w}) + (1-y^j)\ln P(Y=0|\mathbf{x}^j,\mathbf{w})$$

$$= \sum_j y^j(w_0 + \sum_i^n w_i x_i^j) - \ln(1+exp(w_0+\sum_i^n w_i x_i^j))$$

Maximizing Conditional Log Likelihood

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

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

$$\begin{aligned} l(\mathbf{w}) &\equiv \ln \prod_j P(y^j|\mathbf{x}^j, \mathbf{w}) \\ &= \sum_j y^j (w_0 + \sum_i^n w_i x_i^j) - \ln(1 + exp(w_0 + \sum_i^n w_i x_i^j)) \end{aligned}$$

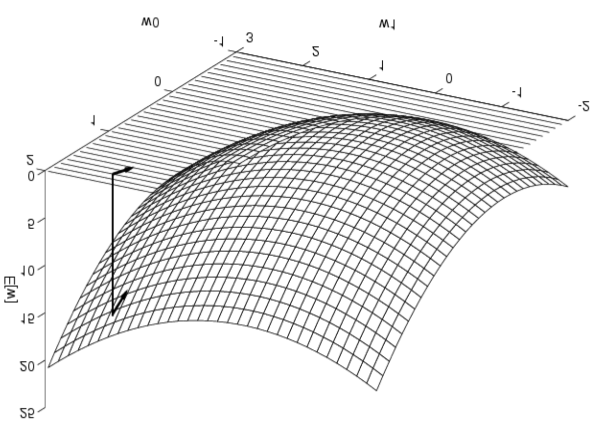
Good news: $l(\mathbf{w})$ is concave function of **w**, no local optima problems

Bad news: no closed-form solution to maximize $l(\mathbf{w})$

Good news: concave functions easy to optimize

Optimizing concave function – Gradient ascent

- Conditional likelihood for Logistic Regression is concave. Find optimum with gradient ascent



Gradient:
$$\nabla_{\mathbf{w}} l(\mathbf{w}) = [\frac{\partial l(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial l(\mathbf{w})}{\partial w_n}]'$$

Step size, η>0

Update rule:
$$\Delta \mathbf{w} = \eta \nabla_{\mathbf{w}} l(\mathbf{w})$$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \frac{\partial l(\mathbf{w})}{\partial w_i}$$

- Gradient ascent is simplest of optimization approaches
  - e.g., Conjugate gradient ascent can be much better

Coordinate Descent v. Gradient Descent

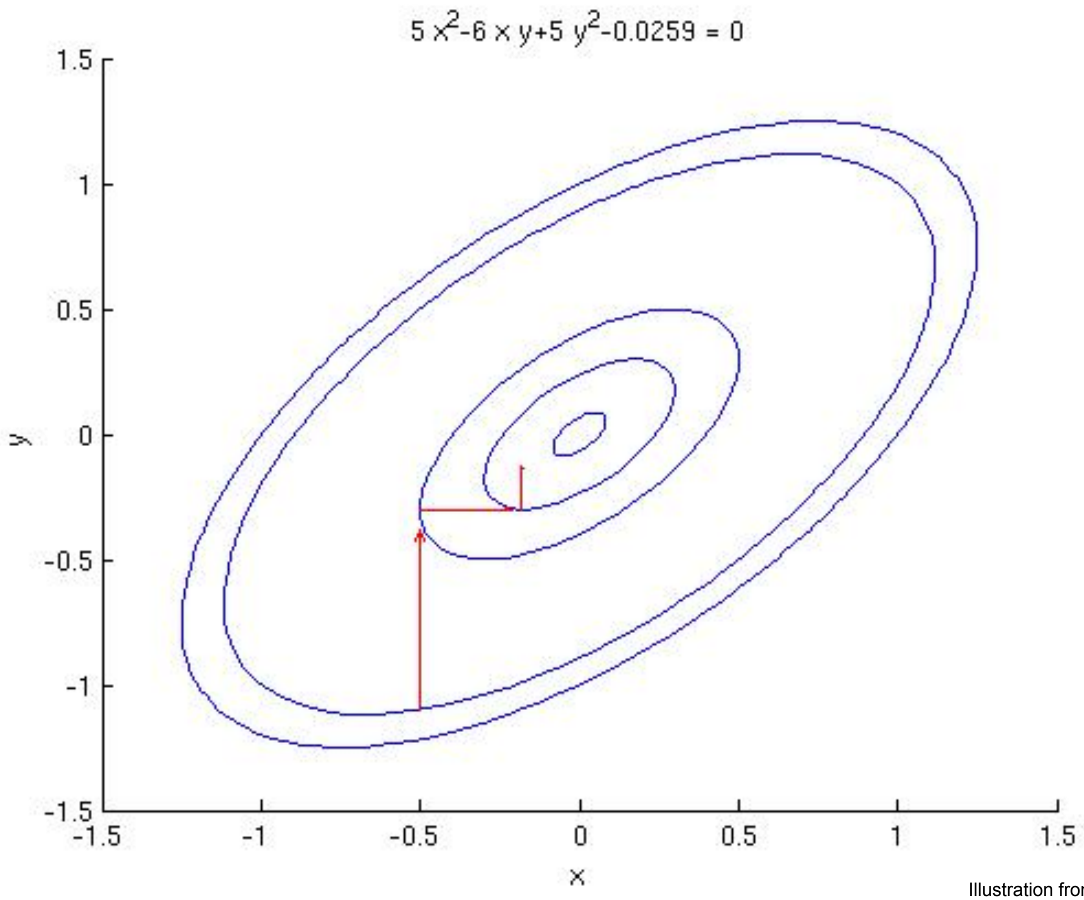


Illustration from Wikipedia

Maximize Conditional Log Likelihood: Gradient ascent

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

$$l(\mathbf{w}) = \sum_j y^j (w_0 + \sum_i^n w_i x_i^j) - \ln(1 + exp(w_0 + \sum_i^n w_i x_i^j))$$

$$\frac{\partial \ell(\mathbf{w})}{\partial w_i} = \sum_{j=1}^{N} x_i^j (y^j - P(Y=1|x^j, \mathbf{w}))$$

Gradient Descent for LR: Intuition

Gender	Age	Location	Income	Referrer	New or Returning	Clicked?
F	Young	US	High	Google	New	N
M	Middle	US	Low	Direct	New	N
F	Old	BR	Low	Google	Returning	Y
M	Young	BR	Low	Bing	Returning	N

Gender (F=1, M=0)	Age (Young=0, Middle=1, Old=2)	Location (US=1, Abroad=0)	Income (High=1, Low=0)	Referrer	New or Returning (New=1,, Returning =0)	Clicked? (Click=1, NoClick=0)

1. Encode data as numbers

2. Until convergence: for each feature
   a. Compute average gradient over data points
   b. Update parameter

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j [y^j - \widehat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w})]$$

Gradient Ascent for LR

Gradient ascent algorithm: iterate until change < ε

$$w_0^{(t+1)} \leftarrow w_0^{(t)} + \eta \sum_j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w}^{(t)})]$$

For i=1,…,k,

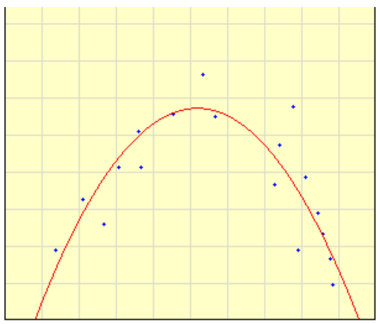
$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w}^{(t)})]$$

repeat

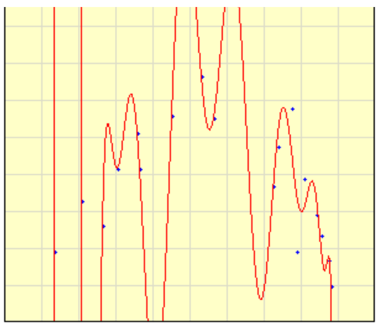
Regularization in linear regression

- Overfitting usually leads to very large parameter choices, e.g.:

$-2.2 + 3.1\,X – 0.30\,X^2$



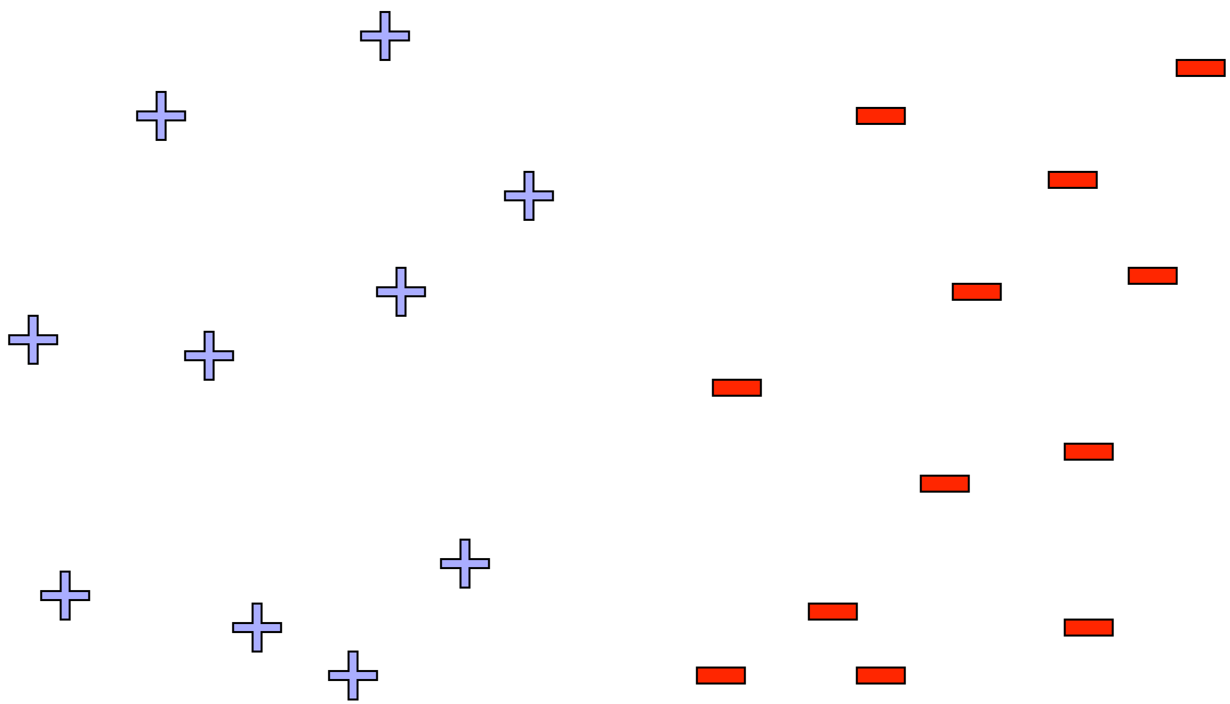
$-1.1 + 4{,}700{,}910.7\,X – 8{,}585{,}638.4\,X^2 + \ldots$



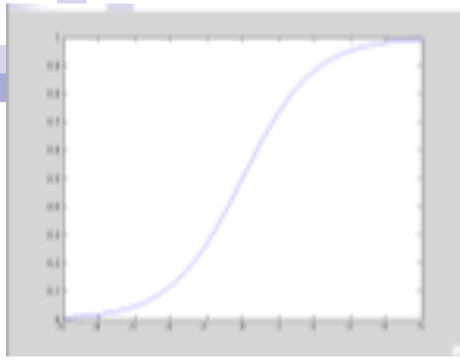
- Regularized least-squares (a.k.a. ridge regression), for $\lambda>0$:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

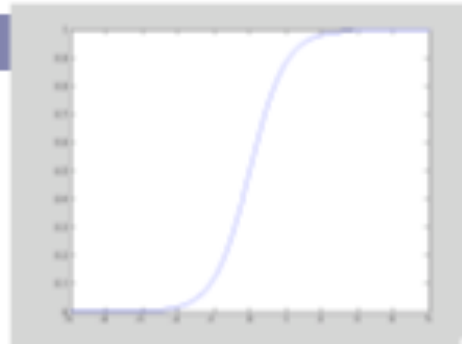
Linear Separability



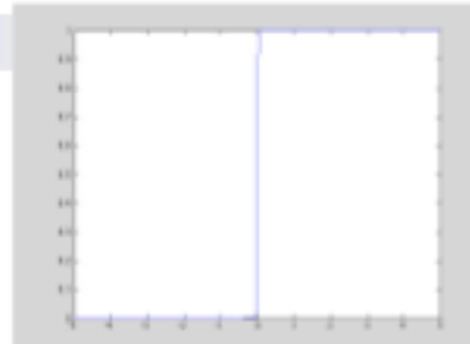
Large parameters → Overfitting



$$\frac{1}{1+e^{-x}}$$



$$\frac{1}{1+e^{-2x}}$$



$$\frac{1}{1+e^{-100x}}$$

- If data is linearly separable, weights go to infinity
  - In general, leads to overfitting:
- Penalizing high weights can prevent overfitting…

Regularized Conditional Log Likelihood

- Add regularization penalty, e.g., L_2:

$$\ell(\mathbf{w}) = \ln \prod_{j=1}^{N} P(y^j|\mathbf{x}^j, \mathbf{w}) - \frac{\lambda}{2}||\mathbf{w}||_2^2$$

- Practical note about w_0:

- Gradient of regularized likelihood:

Standard v. Regularized Updates

- Maximum conditional likelihood estimate

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \ \ln \prod_{j=1}^{N} P(y^j|\mathbf{x}^j, \mathbf{w})$$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w}^{(t)})]$$

- Regularized maximum conditional likelihood estimate

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \ \ln \prod_{j=1}^{N} P(y^j|\mathbf{x}^j, \mathbf{w}) - \frac{\lambda}{2}\sum_{i=1}^{k} w_i^2$$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w}^{(t)})] \right\}$$

Please Stop!! Stopping criterion

$$\ell(\mathbf{w}) = \ln \prod_j P(y^j | \mathbf{x}^j, \mathbf{w})) - \lambda ||\mathbf{w}||_2^2$$

- When do we stop doing gradient descent?

- Because *l*(**w**) is strongly concave:
  - i.e., because of some technical condition

$$\ell(\mathbf{w}^*) - \ell(\mathbf{w}) \leq \frac{1}{2\lambda} ||\nabla \ell(\mathbf{w})||_2^2$$

- Thus, stop when:

Digression: Logistic regression for more than 2 classes

- Logistic regression in more general case (C classes), where *Y in* {0,…,C-1}

Digression: Logistic regression more generally

- Logistic regression in more general case, where *Y in* {0,…,C-1}

for *c>0*

$$P(Y = c|\mathbf{x}, \mathbf{w}) = \frac{\exp(w_{c0} + \sum_{i=1}^{k} w_{ci} x_i)}{1 + \sum_{c'=1}^{C-1} \exp(w_{c'0} + \sum_{i=1}^{k} w_{c'i} x_i)}$$

for *c=0* (normalization, so no weights for this class)

$$P(Y = 0|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \sum_{c'=1}^{C-1} \exp(w_{c'0} + \sum_{i=1}^{k} w_{c'i} x_i)}$$

Learning procedure is basically the same as what we derived!

Stochastic Gradient Descent

Machine Learning – CSEP546
Carlos Guestrin
University of Washington

January 27, 2014

The Cost, The Cost!!! Think about the cost…

- What's the cost of a gradient update step for LR???

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w}^{(t)})] \right\}$$

Learning Problems as Expectations

- Minimizing loss in training data:
  - Given dataset:
    - Sampled iid from some distribution p(**x**) on features:
  - Loss function, e.g., squared error, logistic loss,…
  - We often minimize loss in training data:

$$\ell_{\mathcal{D}}(\mathbf{w}) = \frac{1}{N}\sum_{j=1}^{N} \ell(\mathbf{w}, \mathbf{x}^j)$$

- However, we should really minimize expected loss on all data:

$$\ell(\mathbf{w}) = E_{\mathbf{x}}\left[\ell(\mathbf{w}, \mathbf{x})\right] = \int p(\mathbf{x})\ell(\mathbf{w}, \mathbf{x})d\mathbf{x}$$

- So, we are approximating the integral by the average on the training data

SGD: Stochastic Gradient Ascent (or Descent)

- "True" gradient: $\nabla \ell(\mathbf{w}) = E_{\mathbf{x}} \left[\nabla \ell(\mathbf{w}, \mathbf{x}) \right]$

- Sample based approximation:

- What if we estimate gradient with just one sample???
  - Unbiased estimate of gradient
  - Very noisy!
  - Called stochastic gradient ascent (or descent)
    - Among many other names
  - VERY useful in practice!!!

Stochastic Gradient Ascent for Logistic Regression

- Logistic loss as a stochastic function:

$$E_{\mathbf{x}}\left[\ell(\mathbf{w},\mathbf{x})\right] = E_{\mathbf{x}}\left[\ln P(y|\mathbf{x},\mathbf{w}) - \lambda ||\mathbf{w}||_2^2\right]$$

- Batch gradient ascent updates:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \frac{1}{N}\sum_{j=1}^{N} x_i^{(j)}[y^{(j)} - P(Y=1|\mathbf{x}^{(j)},\mathbf{w}^{(t)})] \right\}$$

- Stochastic gradient ascent updates:
  - Online setting:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + x_i^{(t)}[y^{(t)} - P(Y=1|\mathbf{x}^{(t)},\mathbf{w}^{(t)})] \right\}$$

Stochastic Gradient Descent for LR: Intuition

Gender	Age	Location	Income	Referrer	New or Returning	Clicked?
F	Young	US	High	Google	New	N
M	Middle	US	Low	Direct	New	N
F	Old	BR	Low	Google	Returning	Y
M	Young	BR	Low	Bing	Returning	N

Gender (F=1, M=0)	Age (Young=0, Middle=1, Old=2)	Location (US=1, Abroad=0)	Income (High=1, Low=0)	Referrer	New or Returning (New=1,, Returning =0)	Clicked? (Click=1, NoClick=0)

1. Until convergence: get a data point
   a. Encode data as numbers
   b. For each feature
      i. Compute gradient for this data point
      ii. Update parameter

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + x_i^{(t)} [y^{(t)} - P(Y=1|\mathbf{x}^{(t)}, \mathbf{w}^{(t)})] \right\}$$

Stochastic Gradient Ascent: general case

- Given a stochastic function of parameters:
  - Want to find maximum

- Start from $\mathbf{w}^{(0)}$
- Repeat until convergence:
  - Get a sample data point $\mathbf{x}^t$
  - Update parameters:

- Works on the online learning setting!
- Complexity of each gradient step is constant in number of examples!
- In general, step size changes with iterations

What you should know…

- Classification: predict discrete classes rather than real values
- Logistic regression model: Linear model
  - Logistic function maps real values to [0,1]
- Optimize conditional likelihood
- Gradient computation
- Overfitting
- Regularization
- Regularized optimization
- Cost of gradient step is high, use stochastic gradient descent

What's the Perceptron Optimizing?

Machine Learning – CSEP546
Carlos Guestrin
University of Washington

January 27, 2014

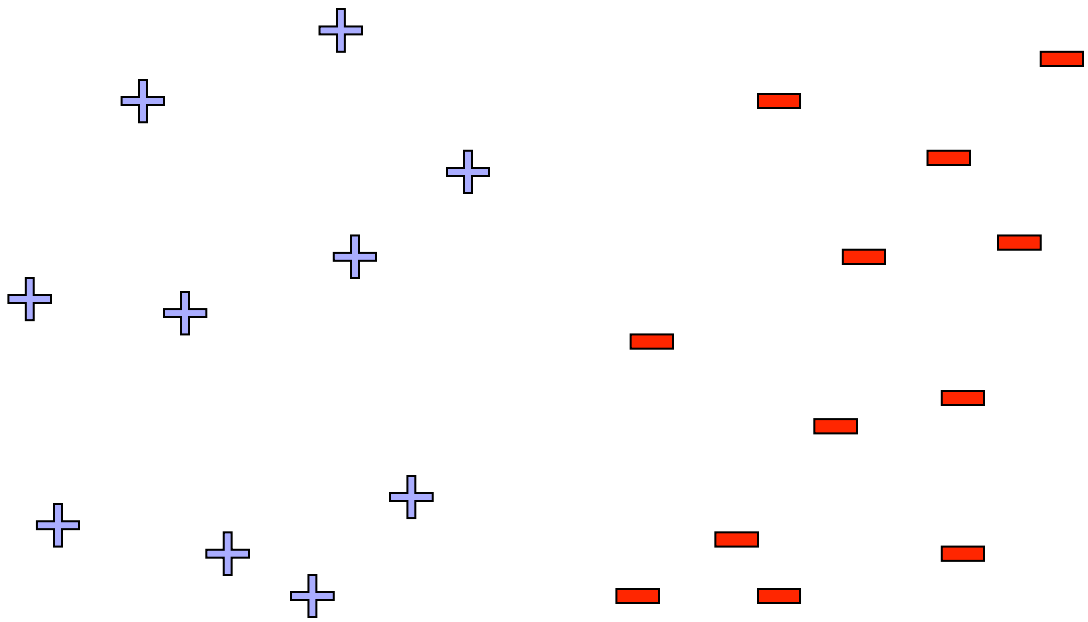
Remember our friend the Perceptron Algorithm

- At each time step:
  - Observe a data point:
  - Update parameters if make a mistake:

What is the Perceptron Doing???

- When we discussed logistic regression:
  - Started from maximizing conditional log-likelihood
- When we discussed the Perceptron:
  - Started from description of an algorithm
- What is the Perceptron optimizing????

Perceptron Prediction: Margin of Confidence



Hinge Loss

- Perceptron prediction:
- Makes a mistake when:
- Hinge loss (same as maximizing the margin used by SVMs)

Stochastic Gradient Descent for Hinge Loss

- SGD: observe data point $x^{(t)}$, update each parameter

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \eta_t \frac{\partial \ell(\mathbf{w}^{(t)}, x^{(t)})}{\partial w_i}$$

- How do we compute the gradient for hinge loss?

(Sub)gradient of Hinge

- Hinge loss:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \eta_t \frac{\partial \ell(\mathbf{w}^{(t)}, x^{(t)})}{\partial w_i}$$

- Subgradient of hinge loss:
  - If $y^{(t)} (w.\mathbf{x}^{(t)}) > 0$:
  - If $y^{(t)} (w.\mathbf{x}^{(t)}) < 0$:
  - If $y^{(t)} (w.\mathbf{x}^{(t)}) = 0$:
  - In one line:

Stochastic Gradient Descent for Hinge Loss

- SGD: observe data point $x^{(t)}$, update each parameter

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \eta_t \frac{\partial \ell(\mathbf{w}^{(t)}, x^{(t)})}{\partial w_i}$$

- How do we compute the gradient for hinge loss?

Perceptron Revisited

- SGD for hinge loss:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta_t \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}$$

- Perceptron update:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}$$

- Difference?

What you need to know

- Perceptron is optimizing hinge loss
- Subgradients and hinge loss
- (Sub)gradient decent for hinge objective

Support Vector Machines

Machine Learning – CSEP546
Carlos Guestrin
University of Washington

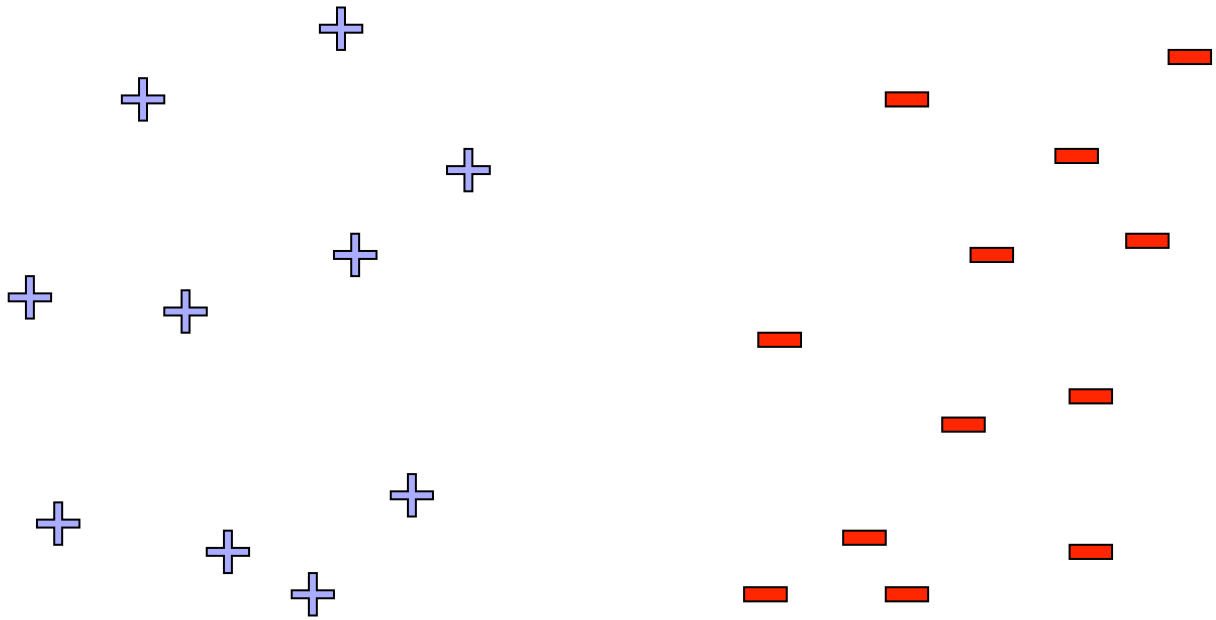
January 27, 2014

Support Vector Machines

- One of the most effective classifiers to date!
- Popularized kernels

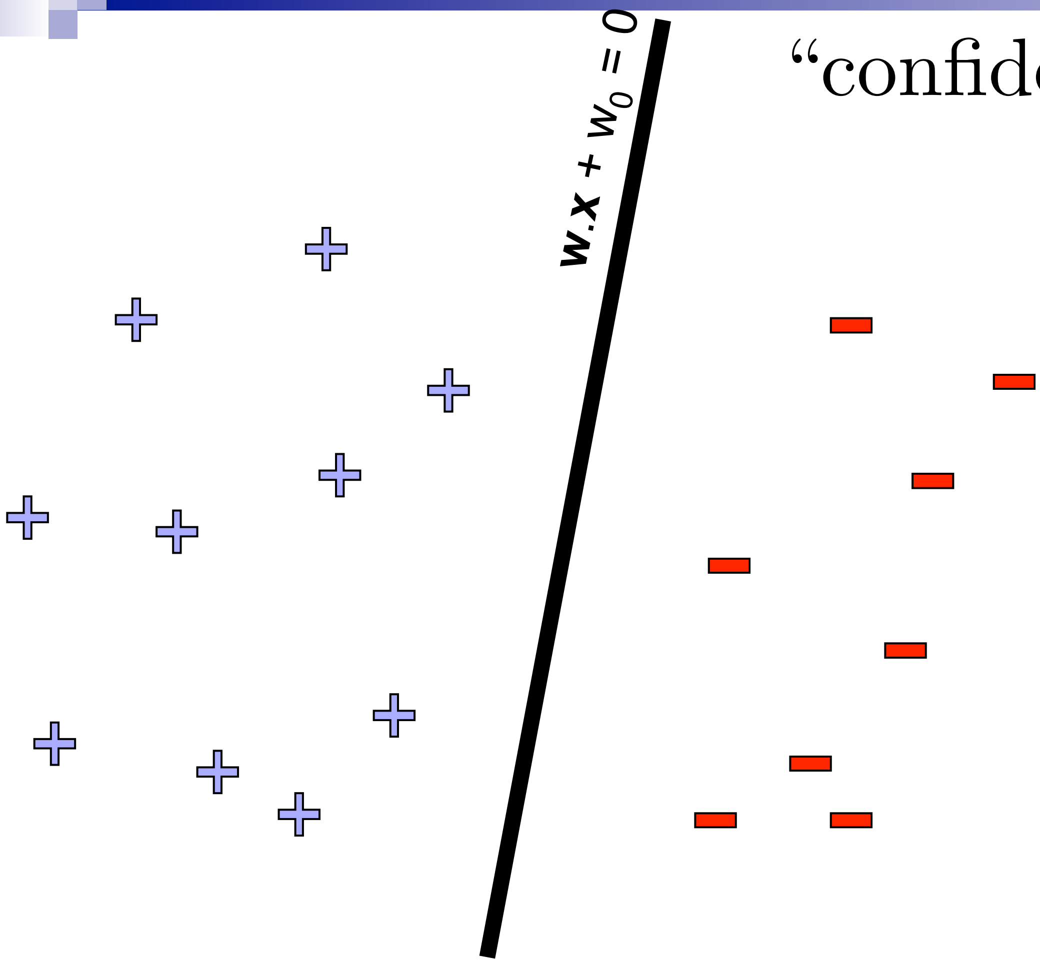
- There is a complicated derivation, but…
- Very simple based on what you've learned thus far!

Linear classifiers – Which line is better?

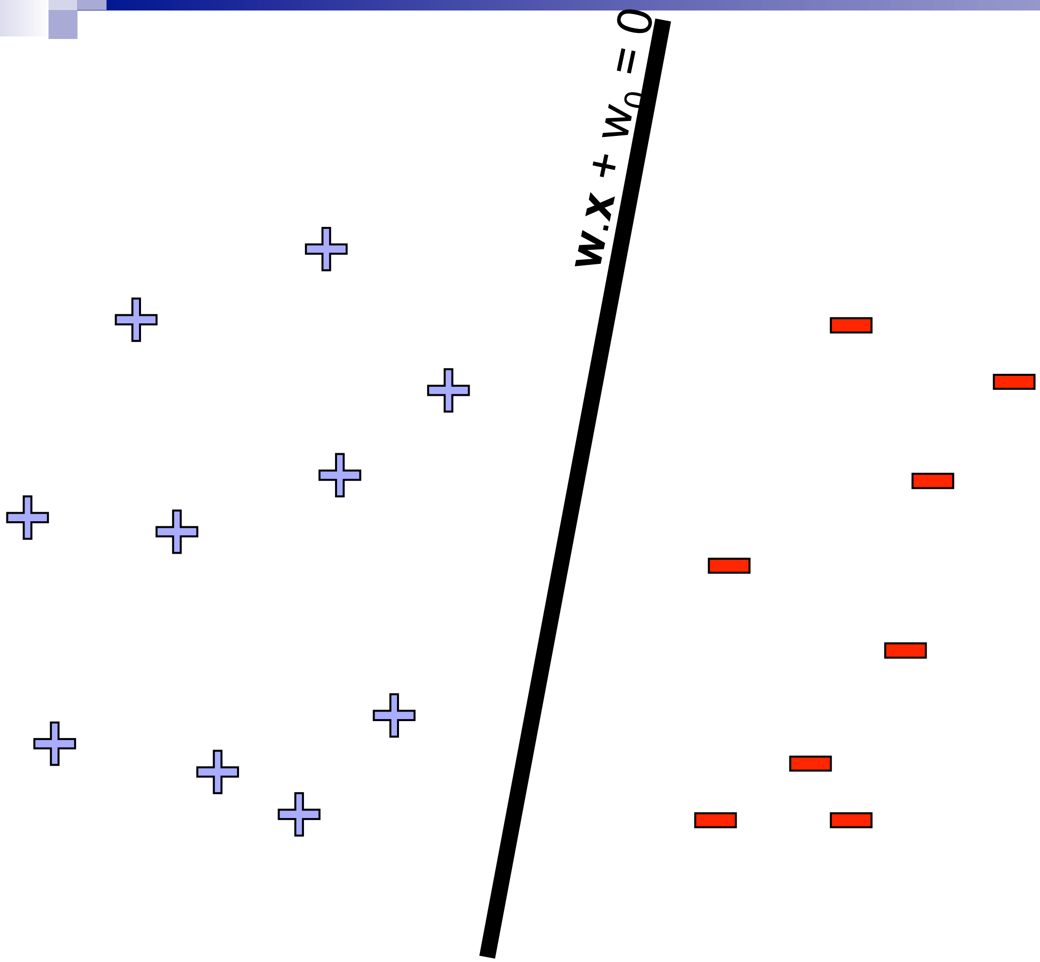


Pick the one with the largest margin!

$$\text{“confidence”} = y^j(\mathbf{w} \cdot \mathbf{x}^j + w_0)$$



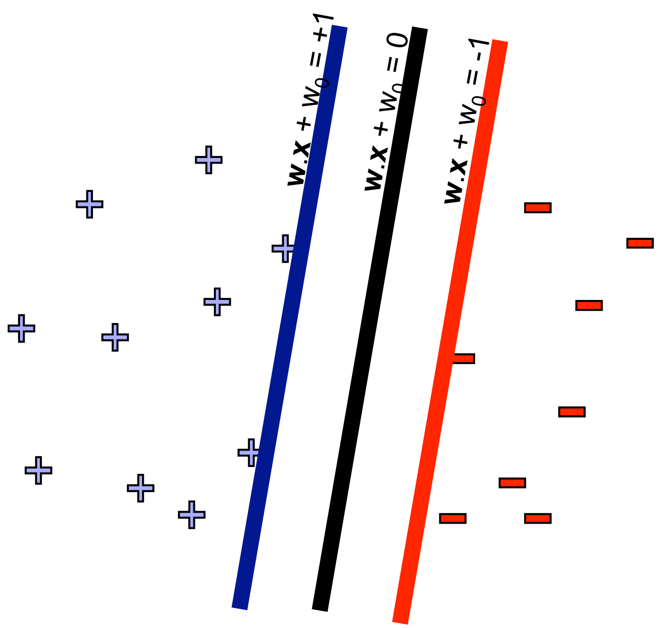
Maximize the margin



SVMs = Hinge Loss + L2 Regularization

- Maximizing Margin same as regularized hinge loss

- But, SVM "convention" is confidence has to be at least 1…



L2 Regularized Hinge Loss

- Final objective, adding regularization:

- But, again, in SVMs, convention slightly different (but equivalent)

$$\frac{||\mathbf{w}||_2^2}{2} + C \sum_{j=1}^{N} \left(1 - y^j(\mathbf{w} \cdot \mathbf{x}^j + w_0)\right)_+$$

SVMs for Non-Linearly Separable meet my friend the Perceptron…

- Perceptron was minimizing the hinge loss:

$$\sum_{j=1}^{N} \left(-y^j(\mathbf{w} \cdot \mathbf{x}^j + w_0)\right)_+$$

- SVMs minimizes the regularized hinge loss!!

$$||\mathbf{w}||_2^2 + C \sum_{j=1}^{N} \left(1 - y^j(\mathbf{w} \cdot \mathbf{x}^j + w_0)\right)_+$$

Stochastic Gradient Descent for SVMs

- Perceptron minimization:

$$\sum_{j=1}^{N} \left(-y^j(\mathbf{w}\cdot\mathbf{x}^j + w_0)\right)_+$$

- SGD for Perceptron:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)}\cdot\mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}$$

- SVMs minimization:

$$||\mathbf{w}||_2^2 + C\sum_{j=1}^{N} \left(1-y^j(\mathbf{w}\cdot\mathbf{x}^j + w_0)\right)_+$$

- SGD for SVMs:

What you need to know

- Maximizing margin
- Derivation of SVM formulation
- Non-linearly separable case
  - Hinge loss
  - A.K.A. adding slack variables
- SVMs = Perceptron + L2 regularization
- Can also use kernels with SVMs
- Can optimize SVMs with SGD
  - Many other approaches possible

Boosting

Machine Learning – CSEP546
Carlos Guestrin
University of Washington

January 27, 2014

Fighting the bias-variance tradeoff

- **Simple (a.k.a. weak) learners are good**
  - e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees)
  - Low variance, don't usually overfit too badly
- **Simple (a.k.a. weak) learners are bad**
  - High bias, can't solve hard learning problems
- Can we make weak learners always good???
  - **No!!!**
  - **But often yes…**

The Simplest Weak Learner: Thresholding, a.k.a. Decision Stumps

- **Learn**: h:**X** $\mapsto$ Y
  - **X** – features
  - Y – target classes

- **Simplest case: Thresholding**

Voting (Ensemble Methods)

- Instead of learning a single (weak) classifier, learn **many weak classifiers** that are **good at different parts of the input space**
- **Output class:** (Weighted) vote of each classifier
  - Classifiers that are most "sure" will vote with more conviction
  - Classifiers will be most "sure" about a particular part of the space
  - On average, do better than single classifier!

- **But how do you ???**
  - force classifiers to learn about different parts of the input space?
  - weigh the votes of different classifiers?

Boosting [Schapire, 1989]

- Idea: given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote

- On each iteration *t*:
  - weight each training example by how incorrectly it was classified
  - Learn a hypothesis – h_t
  - A strength for this hypothesis – α_t

- Final classifier:

- **Practically useful**
- **Theoretically interesting**

Learning from weighted data

- **Sometimes not all data points are equal**
  - Some data points are more equal than others
- **Consider a weighted dataset**
  - D(j) – weight of *j* th training example ($\mathbf{x}^j$,y^j)
  - Interpretations:
    - *j* th training example counts as D(j) examples
    - If I were to “resample” data, I would get more samples of “heavier” data points
- **Now, in all calculations, whenever used, *j* th training example counts as D(j) “examples”**

Boosting Cartoon

AdaBoost

- Initialize weights to uniform dist: $D_1(j) = 1/N$
- For t = 1…T
  - Train weak learner h_t on distribution D_t over the data
  - Choose weight α_t
  - Update weights:

$$D_{t+1}(j) = \frac{D_t(j)\exp(-\alpha_t y^j h_t(x^j))}{Z_t}$$

    - Where Z_t is normalizer:

$$Z_t = \sum_{j=1}^{N} D_t(j)\exp(-\alpha_t y^j h_t(x^j))$$

- Output final classifier:

Picking Weight of Weak Learner

- Weigh h_t higher if it did well on training data (weighted by D_t):

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

  - Where ε_t is the weighted training error:

$$\epsilon_t = \sum_{j=1}^{N} D_t(j) \mathbb{1}[h_t(x^j) \neq y^j]$$

AdaBoost Cartoon

$$D_{t+1}(j) = \frac{D_t(j)\exp(-\alpha_t y^j h_t(x^j))}{Z_t}$$

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

Why choose α_t for hypothesis h_t this way?

[Schapire, 1989]

$$Z_t = \sum_{j=1}^{N} D_t(j) \exp(-\alpha_t y^j h_t(x^j))$$

- Simple theoretical analysis:
  - Training error upper-bounded by product of normalizers

$$\frac{1}{N}\sum_{j=1}^{N} \mathbb{1}[H(x^j) \neq y^j] \leq \prod_{t=1}^{T} Z_t$$

  - Pick α_t to minimize upper-bound
    - Take derivative and set to zero!

Strong, weak classifiers

- If each classifier is (at least slightly) better than random
  - $\varepsilon_t < 0.5$

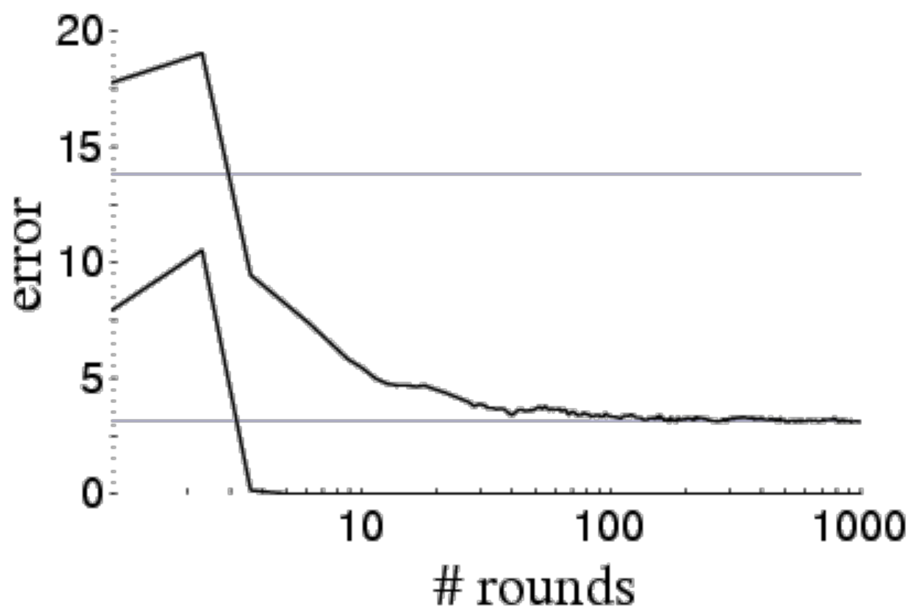
- AdaBoost will achieve zero *training error* (exponentially fast):

$$\frac{1}{N}\sum_{j=1}^{N}\mathbb{1}[H(x^j) \neq y^j] \leq \prod_{t=1}^{T} Z_t \leq \exp\left(-2\sum_{t=1}^{T}(1/2-\epsilon_t)^2\right)$$

- Is it hard to achieve better than random training error?

Boosting results – Digit recognition

[Schapire, 1989]

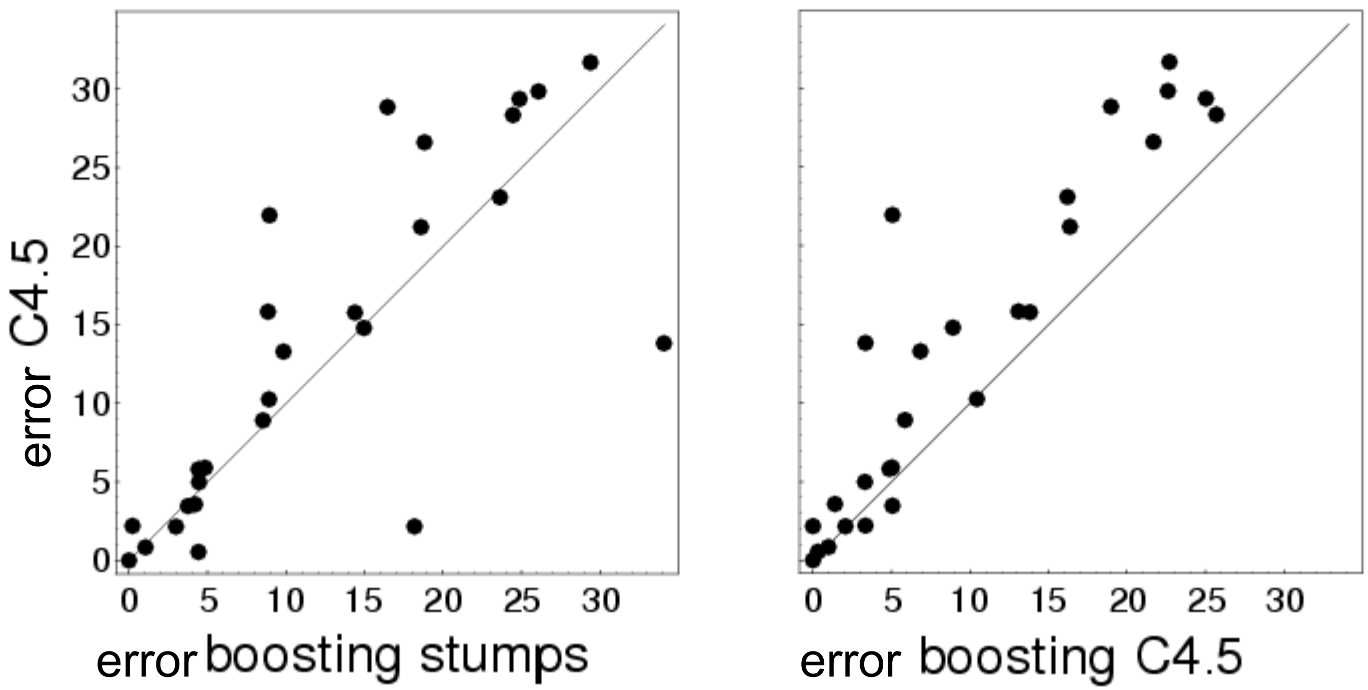


- Boosting often
  - Robust to overfitting
  - Test set error decreases even after training error is zero

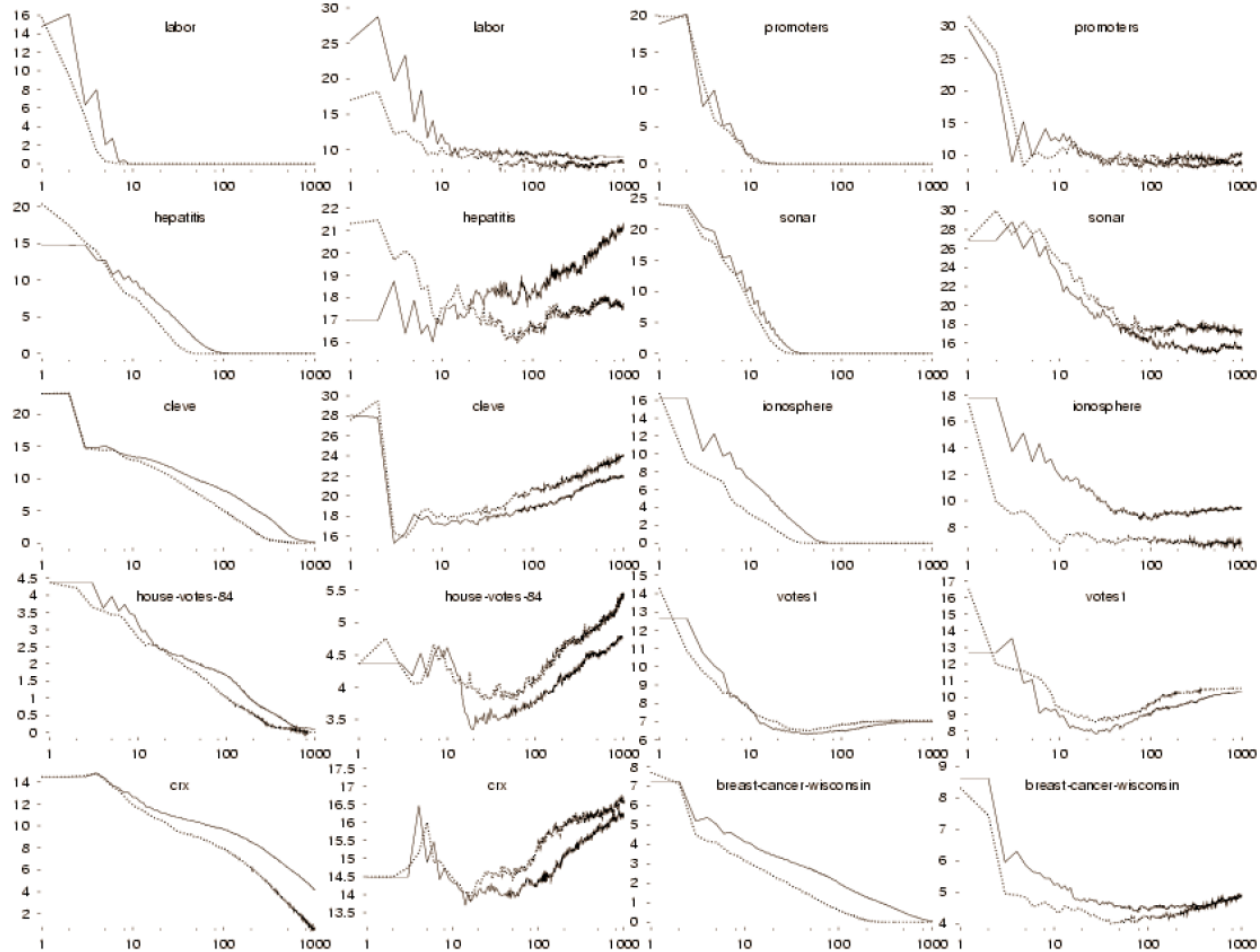
Boosting: Experimental Results

[Freund & Schapire, 1996]

Comparison of C4.5, Boosting C4.5, Boosting decision stumps (depth 1 trees), 27 benchmark datasets



AdaBoost and AdaBoost.MH on Train (left) and Test (right) data from Irvine repository. [Schapire and Singer, ML 1999]



What you need to know about Boosting

- Combine weak classifiers to obtain very strong classifier
  - Weak classifier – slightly better than random on training data
  - Resulting very strong classifier – can eventually provide zero training error
- AdaBoost algorithm
- Most popular application of Boosting:
  - Boosted decision stumps!
  - Very simple to implement, very effective classifier