Lecture 4: More classifiers and classes

C4B Machine Learning Hilary 2011 A. Zisserman

- Logistic regression
  - Loss functions revisited
- Adaboost
  - Loss functions revisited
- Optimization
- Multiple class classification

Logistic Regression

Overview

- Logistic regression is actually a classification method

- LR introduces an extra non-linearity over a linear classifier, $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$, by using a logistic (or sigmoid) function, $\sigma()$.

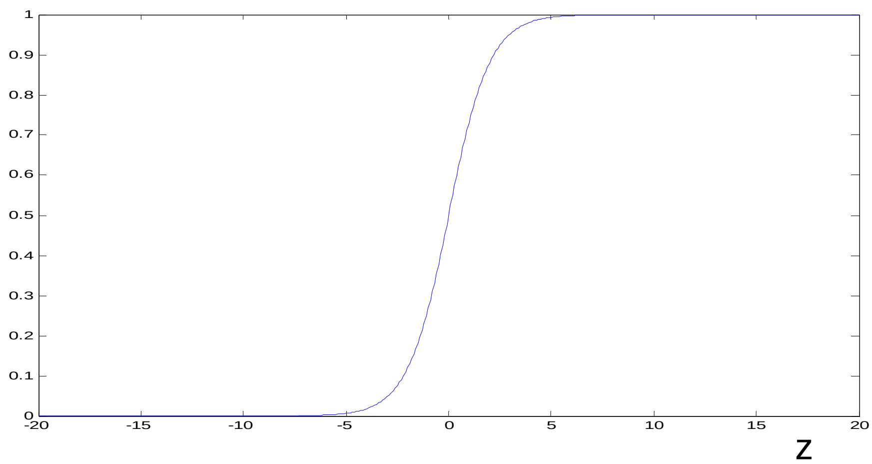
- The LR classifier is defined as

$$\sigma\left(f(\mathbf{x}_i)\right) \begin{cases} \geq 0.5 & y_i = +1 \\ < 0.5 & y_i = -1 \end{cases}$$

where $\sigma(f(\mathbf{x})) = \frac{1}{1+e^{-f(\mathbf{x})}}$

The logistic function or sigmoid function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



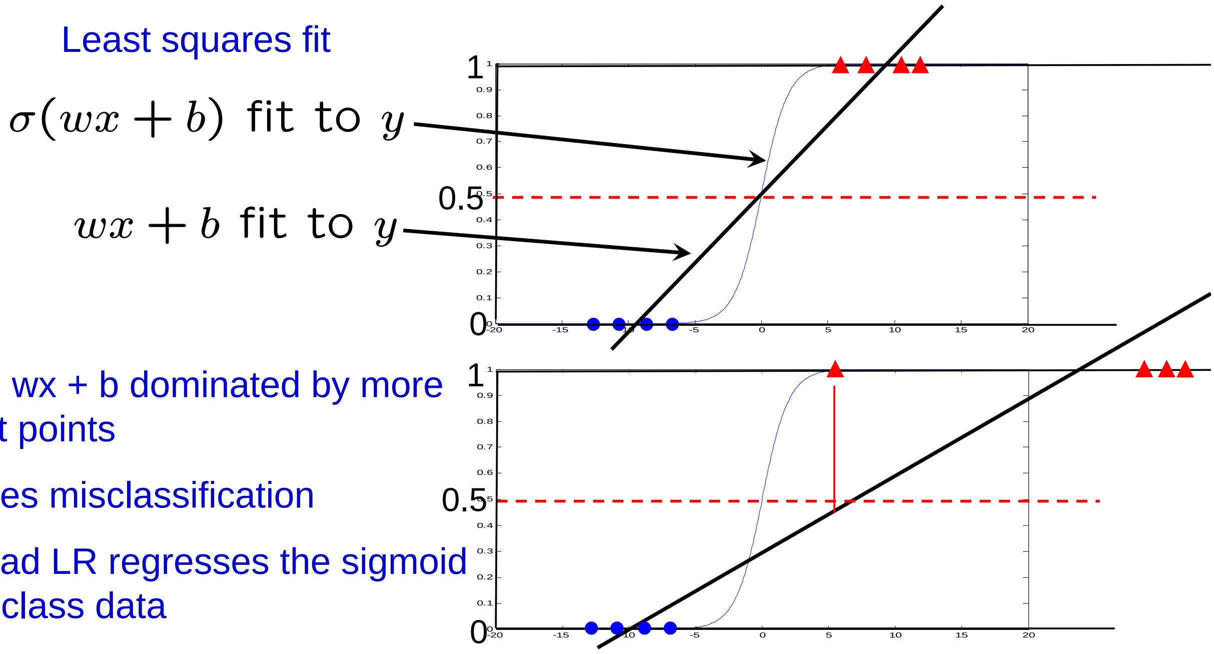
- As z goes from $-\infty$ to ∞, $\sigma(z)$ goes from 0 to 1, a "squashing function".

- It has a "sigmoid" shape (i.e. S-like shape)

- $\sigma(0) = 0.5$, and if $z = \mathbf{w}^\top \mathbf{x} + b$ then $||\frac{d\sigma(z)}{d\mathbf{x}}||_{z=0} = \frac{1}{4}||\mathbf{w}||$

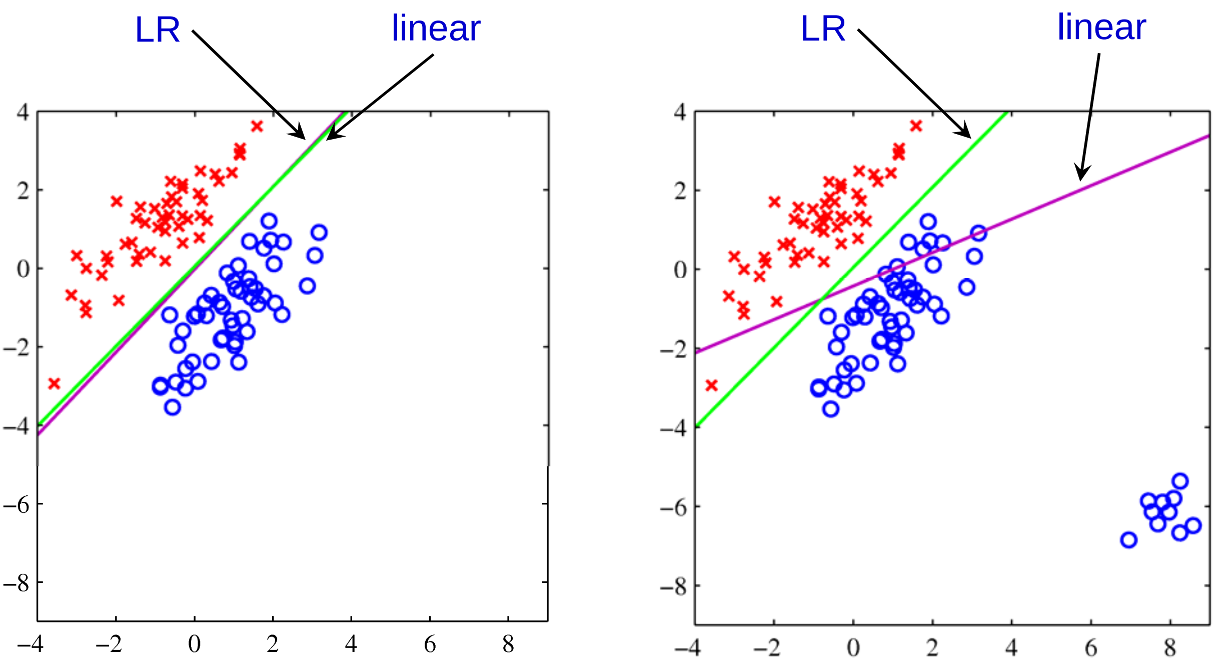
Intuition – why use a sigmoid?

Here, choose binary classification to be represented by $y_i \in \{0, 1\}$, rather than $y_i \in \{1, -1\}$



- fit of wx + b dominated by more distant points
- causes misclassification
- instead LR regresses the sigmoid to the class data

Similarly in 2D

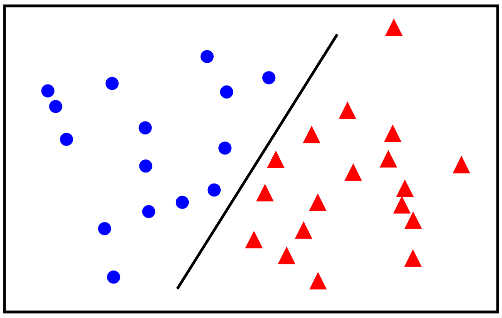


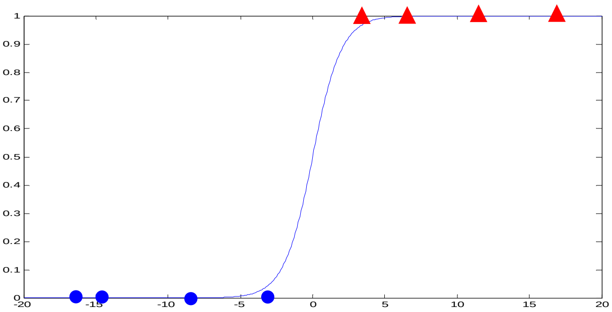
$\sigma(w_1x_1 + w_2x_2 + b)$ fit, vs $w_1x_1 + w_2x_2 + b$

Learning

In logistic regression fit a sigmoid function to the data { $\mathbf{x}_i$, y_i } by minimizing the classification errors

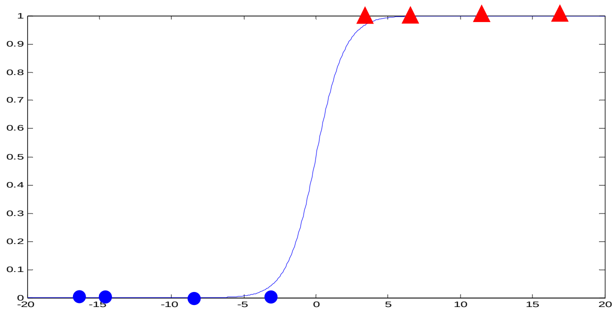
$$y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i)$$

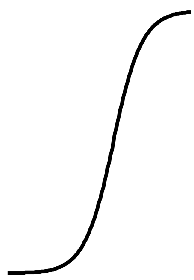


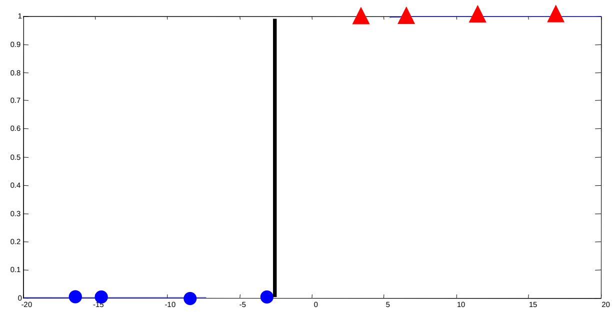


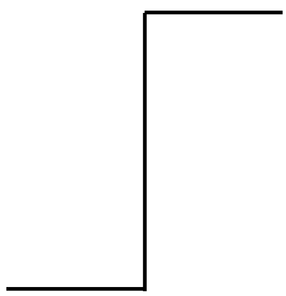
Margin property

A sigmoid favours a larger margin cf a step classifier









Probabilistic interpretation

- Think of $\sigma(f(\mathbf{x}))$ as the posterior probability that $y = 1$, i.e. $P(y = 1|\mathbf{x}) = \sigma(f(\mathbf{x}))$
- Hence, if $\sigma(f(\mathbf{x})) > 0.5$ then class $y = 1$ is selected
- Then, after a rearrangement

$$f(\mathbf{x}) = \log \frac{P(y = 1|\mathbf{x})}{1 - P(y = 1|\mathbf{x})} = \log \frac{P(y = 1|\mathbf{x})}{P(y = 0|\mathbf{x})}$$

which is the log odds ratio

Maximum Likelihood Estimation

Assume

$$\begin{aligned} p(y = 1|\mathbf{x}; \mathbf{w}) &= \sigma(\mathbf{w}^\top \mathbf{x}) \\ p(y = 0|\mathbf{x}; \mathbf{w}) &= 1 - \sigma(\mathbf{w}^\top \mathbf{x}) \end{aligned}$$

write this more compactly as

$$p(y|\mathbf{x}; \mathbf{w}) = \left(\sigma(\mathbf{w}^\top \mathbf{x})\right)^y \left(1 - \sigma(\mathbf{w}^\top \mathbf{x})\right)^{(1-y)}$$

Then the likelihood (assuming data independence) is

$$p(\mathbf{y}|\mathbf{x}; \mathbf{w}) \sim \prod_i^N \left(\sigma(\mathbf{w}^\top \mathbf{x}_i)\right)^{y_i} \left(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i)\right)^{(1-y_i)}$$

and the negative log likelihood is

$$L(\mathbf{w}) = -\sum_i^N y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i))$$

Logistic Regression Loss function

Use notation $y_i \in \{-1, 1\}$. Then

$$P(y = 1|\mathbf{x}) = \sigma(f(\mathbf{x})) = \frac{1}{1 + e^{-f(\mathbf{x})}}$$

$$P(y = -1|\mathbf{x}) = 1 - \sigma(f(\mathbf{x})) = \frac{1}{1 + e^{+f(\mathbf{x})}}$$

So in both cases

$$P(y_i|\mathbf{x}_i) = \frac{1}{1 + e^{-y_i f(\mathbf{x}_i)}}$$

Assuming independence, the likelihood is

$$\prod_i^N \frac{1}{1 + e^{-y_i f(\mathbf{x}_i)}}$$

and the negative log likelihood is

$$= \sum_i^N \log\left(1 + e^{-y_i f(\mathbf{x}_i)}\right)$$

which defines the loss function.

Logistic Regression Learning

Learning is formulated as the optimization problem

$$\min_{\mathbf{w} \in \mathbb{R}^d} \sum_i^N \underbrace{\log\left(1 + e^{-y_i f(\mathbf{x}_i)}\right)}_{\text{loss function}} + \underbrace{\lambda||\mathbf{w}||^2}_{\text{regularization}}$$

- For correctly classified points $-y_i f(\mathbf{x}_i)$ is negative, and $\log\left(1 + e^{-y_i f(\mathbf{x}_i)}\right)$ is near zero
- For incorrectly classified points $-y_i f(\mathbf{x}_i)$ is positive, and $\log\left(1 + e^{-y_i f(\mathbf{x}_i)}\right)$ can be large.
- Hence the optimization penalizes parameters which lead to such misclassifications

Comparison of SVM and LR cost functions

SVM

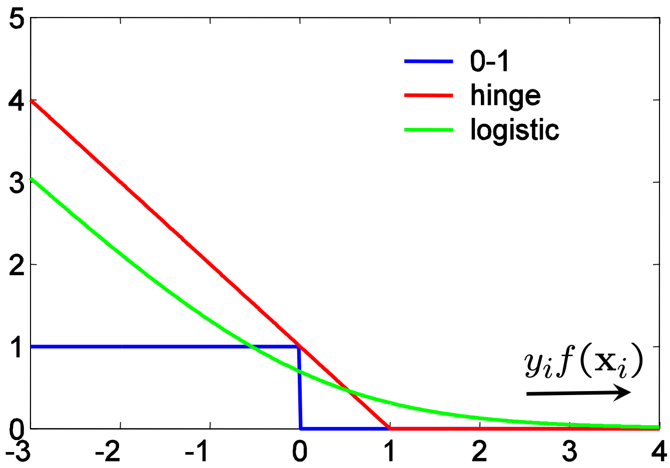
$$\min_{\mathbf{w}\in\mathbb{R}^d} C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) + ||\mathbf{w}||^2$$

Logistic regression:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \sum_i^N \log\left(1 + e^{-y_i f(\mathbf{x}_i)}\right) + \lambda||\mathbf{w}||^2$$

Note:

- both approximate 0-1 loss
- very similar asymptotic behaviour
- main difference is smoothness, and non-zero values outside margin
- SVM gives sparse solution for α_i



AdaBoost

Overview

- AdaBoost is an algorithm for constructing a strong classifier out of a linear combination

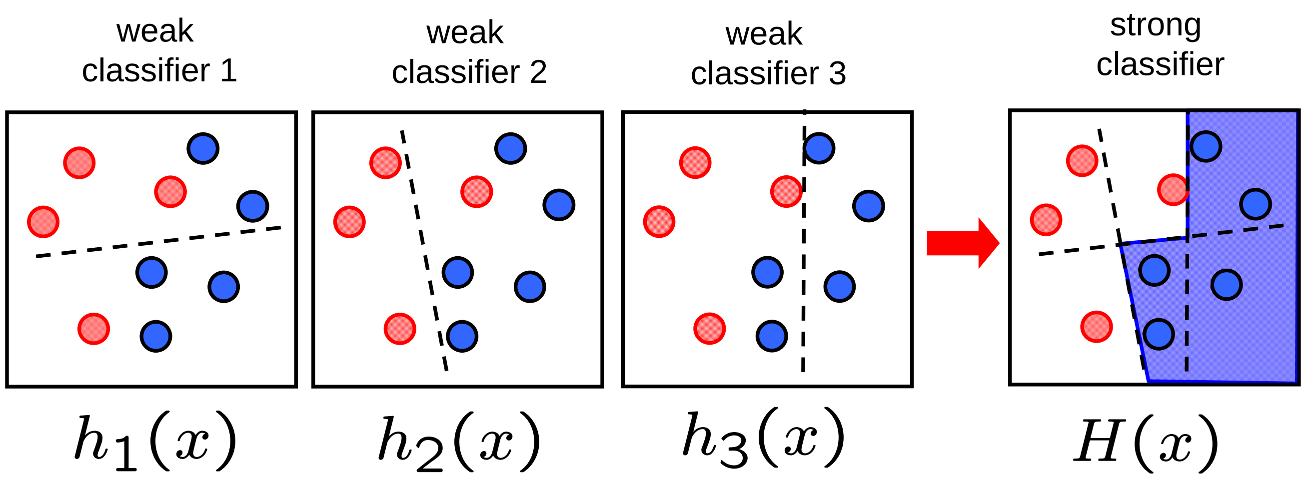
$$\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$$

of simple weak classifiers $h_t(\mathbf{x})$. It provides a method of choosing the weak classifiers and setting the weights α_t

Terminology

- weak classifier $h_t(\mathbf{x}) \in \{-1, 1\}$ for data vector $\mathbf{x}$
- strong classifier $H(\mathbf{x}) = \text{sign} \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$

Example: combination of linear classifiers $h_t(x) \in \{-1, 1\}$



$$H(x) = \text{sign}\left(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x)\right)$$

- Note, this linear combination is not a simple majority vote (it would be if $\alpha_t = 1, \forall t$)
- Need to compute α_t as well as selecting weak classifiers

AdaBoost algorithm – building a strong classifier

Start with equal weights on each x_i, and a set of weak classifiers $h_t(x)$

For t = 1 …,T

- Select weak classifier with minimum error

$$\epsilon_t = \sum_i \omega_i [h_t(x_i) \neq y_i] \quad \text{where } \omega_i \text{ are weights}$$

- Set

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$$

- Reweight examples (boosting) to give misclassified examples more weight

$$\omega_{t+1,i} = \omega_{t,i} e^{-\alpha_t y_i h_t(x_i)}$$

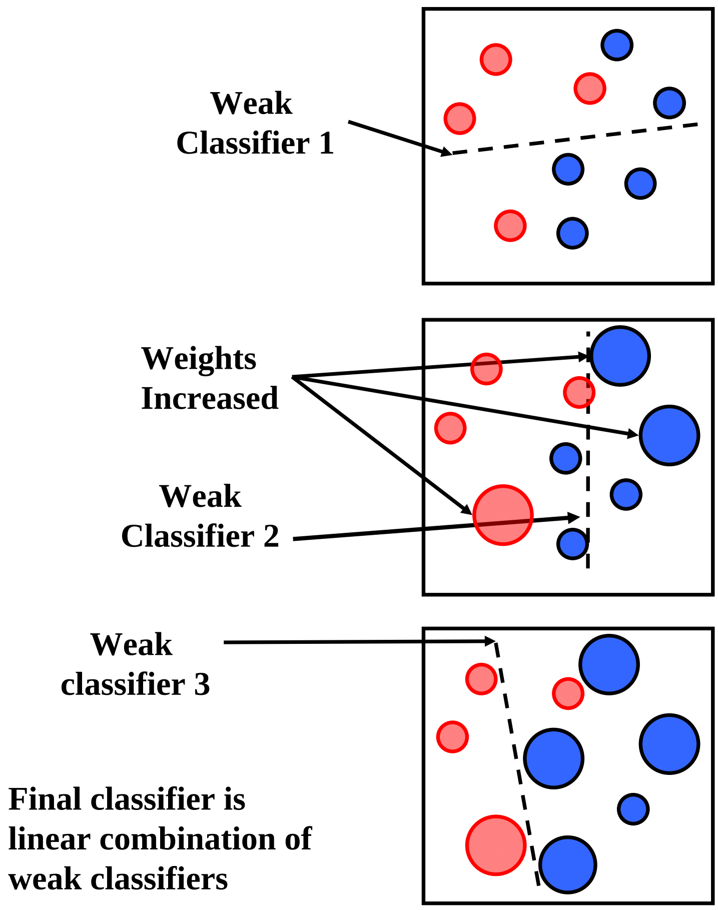
- Add weak classifier with weight α_t

$$H(x) = \text{sign} \sum_{t=1}^{T} \alpha_t h_t(x)$$

Example

start with equal weights on each data point (i)

$$\epsilon_j = \sum_i \omega_i [h_j(x_i) \neq y_i]$$



The AdaBoost algorithm (Freund & Shapire 1995)

- Given example data $(x_1, y_1), \ldots, (x_n, y_n)$, where $y_i = -1, 1$ for negative and positive examples respectively.
- Initialize weights $\omega_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = -1, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$
  1. Normalize the weights,
  $$\omega_{t,i} \leftarrow \frac{\omega_{t,i}}{\sum_{j=1}^{n} \omega_{t,j}}$$
  so that $\omega_{t,i}$ is a probability distribution.
  2. For each j, train a weak classifier h_j with error evaluated with respect to $\omega_{t,i}$,
  $$\epsilon_j = \sum_i \omega_{t,i}[h_j(x_i) \neq y_i]$$
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Set α_t as
  $$\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$$
  5. Update the weights
  $$\omega_{t+1,i} = \omega_{t,i} e^{-\alpha_t y_i h_t(x_i)}$$
- The final strong classifier is
$$H(x) = \text{sign} \sum_{t=1}^{T} \alpha_t h_t(x)$$

Why does it work?

The AdaBoost algorithm carries out a greedy optimization of a loss function

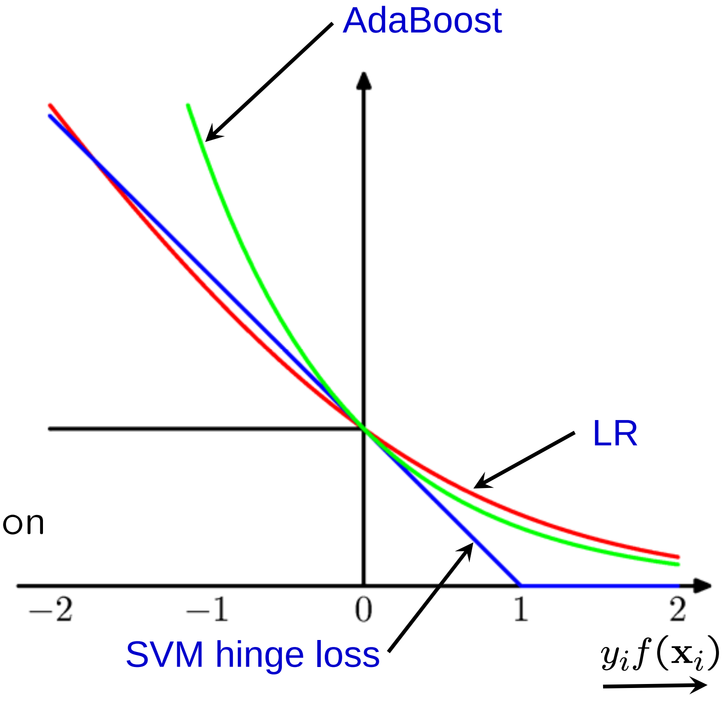
$$\min_{\alpha_i, h_i} \sum_i^N e^{-y_i H(\mathbf{x}_i)}$$

SVM loss function

$$\sum_i^N \max\left(0, 1 - y_i f(\mathbf{x}_i)\right)$$

Logistic regression loss function

$$\sum_i^N \log\left(1 + e^{-y_i f(\mathbf{x}_i)}\right)$$



Sketch derivation – non-examinable

The objective function used by AdaBoost is

$$J(H) = \sum_i e^{-y_i H(x_i)}$$

For a correctly classified point the penalty is $\exp(-|H|)$ and for an incorrectly classified point the penalty is $\exp(+|H|)$. The AdaBoost algorithm incrementally decreases the cost by adding simple functions to

$$H(x) = \sum_t \alpha_t h_t(x)$$

Suppose that we have a function B and we propose to add the function $\alpha h(x)$ where the scalar α is to be determined and $h(x)$ is some function that takes values in $+1$ or -1 only. The new function is $B(x) + \alpha h(x)$ and the new cost is

$$J(B + \alpha h) = \sum_i e^{-y_i B(x_i)} e^{-\alpha y_i h(x_i)}$$

Differentiating with respect to α and setting the result to zero gives

$$e^{-\alpha} \sum_{y_i = h(x_i)} e^{-y_i B(x_i)} - e^{+\alpha} \sum_{y_i \neq h(x_i)} e^{-y_i B(x_i)} = 0$$

Rearranging, the optimal value of α is therefore determined to be

$$\alpha = \frac{1}{2} \log \frac{\sum_{y_i = h(x_i)} e^{-y_i B(x_i)}}{\sum_{y_i \neq h(x_i)} e^{-y_i B(x_i)}}$$

The classification error is defined as

$$\epsilon = \sum_i \omega_i [h(x_i) \neq y_i]$$

where

$$\omega_i = \frac{e^{-y_i B(x_i)}}{\sum_j e^{-y_j B(x_j)}}$$

Then, it can be shown that,

$$\alpha = \frac{1}{2} \log \frac{1 - \epsilon}{\epsilon}$$

The update from B to H therefore involves evaluating the weighted performance (with the weights ω_i given above) ϵ of the "weak" classifier h.

If the current function B is $B(x) = 0$ then the weights will be uniform. This is a common starting point for the minimization. As a numerical convenience, note that at the next round of boosting the required weights are obtained by multiplying the old weights with $\exp(-\alpha y_i h(x_i))$ and then normalizing. This gives the update formula

$$\omega_{t+1,i} = \frac{1}{Z_t} \omega_{t,i} e^{-\alpha_t y_i h_t(x_i)}$$

where Z_t is a normalizing factor.

Choosing h The function h is not chosen arbitrarily but is chosen to give a good performance (low value of ϵ) on the training data weighted by ω.

Optimization

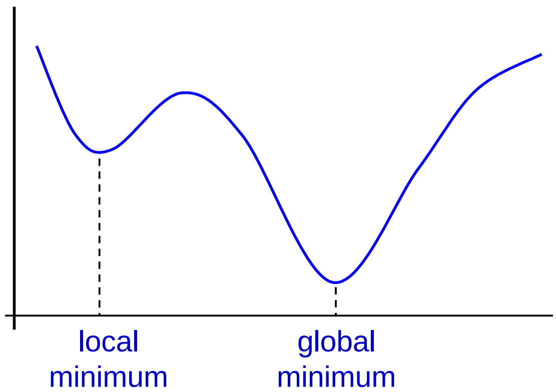
We have seen many cost functions, e.g.

SVM

$$\min_{\mathbf{w}\in\mathbb{R}^d} C\sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) + ||\mathbf{w}||^2$$

Logistic regression:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \sum_i^N \log\left(1 + e^{-y_i f(\mathbf{x}_i)}\right) + \lambda||\mathbf{w}||^2$$



- Do these have a unique solution?
- Does the solution depend on the starting point of an iterative optimization algorithm (such as gradient descent)?

If the cost function is convex, then a locally optimal point is globally optimal (provided the optimization is over a convex set, which it is in our case)

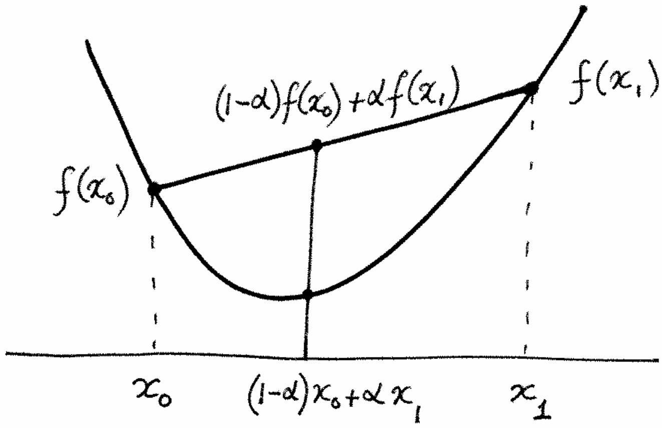
Convex functions

D − a domain in $\mathbb{R}^n$.

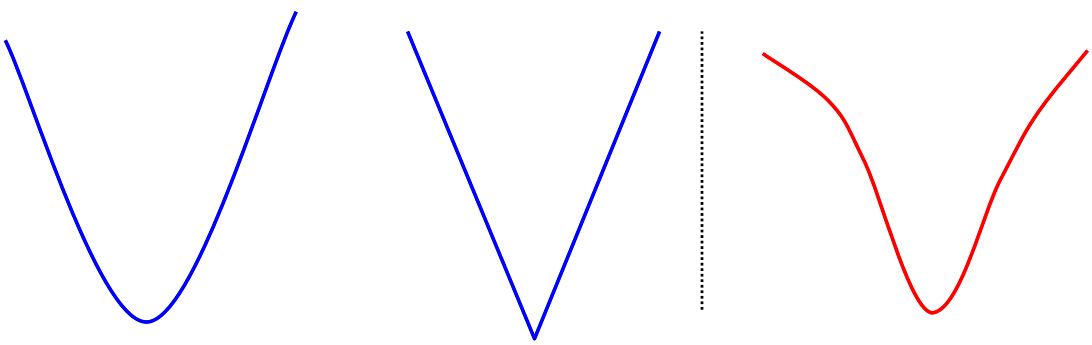
A convex function $f : D \rightarrow \mathbb{R}$ is one that satisfies, for any $\mathbf{x}_0$ and $\mathbf{x}_1$ in D:

$$f((1-\alpha)\mathbf{x}_0 + \alpha\mathbf{x}_1) \leq (1-\alpha)f(\mathbf{x}_0) + \alpha f(\mathbf{x}_1) \quad .$$

Line joining $(\mathbf{x}_0, f(\mathbf{x}_0))$ and $(\mathbf{x}_1, f(\mathbf{x}_1))$ lies above the function graph.



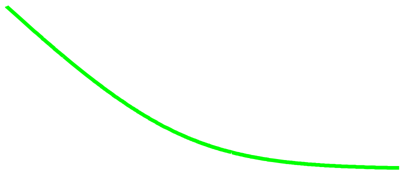
Convex function examples



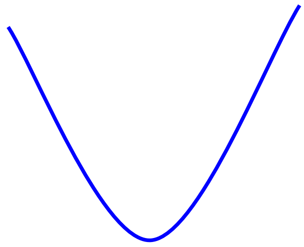
convex

Not convex

A non-negative sum of convex functions is convex



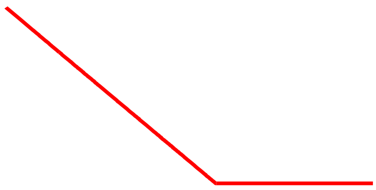
+



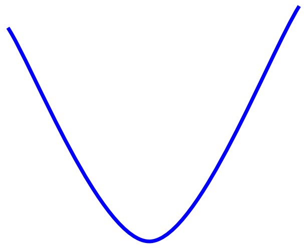
Logistic regression:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \sum_i^N \log\left(1+e^{-y_i f(\mathbf{x}_i)}\right) + \lambda||\mathbf{w}||^2$$

convex



+



SVM

$$\min_{\mathbf{w}\in\mathbb{R}^d} C\sum_i^N \max\left(0, 1-y_i f(\mathbf{x}_i)\right) + ||\mathbf{w}||^2$$

convex

Gradient (or Steepest) descent algorithms

To minimize a cost function $\mathcal{C}(\mathbf{w})$ use the iterative update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \mathcal{C}(\mathbf{w}_t)$$

where η is the learning rate.

In our case the loss function is a sum over the training data. For example for LR

$$\min_{\mathbf{w}\in\mathbb{R}^d} \mathcal{C}(\mathbf{w}) = \sum_i^N \log\left(1+e^{-y_i f(\mathbf{x}_i)}\right) + \lambda||\mathbf{w}||^2 = \sum_i^N \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}) + \lambda||\mathbf{w}||^2$$

This means that one iterative update consists of a pass through the training data with an update *for each point*

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \left(\sum_i^N \eta_t \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}_t) + 2\lambda\mathbf{w}_t\right)$$

The advantage is that for large amounts of data, this can be carried out point by point.

Gradient descent algorithm for LR

Minimizing $\mathcal{L}(\mathbf{w})$ using gradient descent gives [exercise] the update rule

$$\mathbf{w} \leftarrow \mathbf{w} - \eta(y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i))\mathbf{x}_i$$

where $y_i \in \{0, 1\}$

Note:

- this is similar, but not identical, to the perceptron update rule.

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \operatorname{sign}(\mathbf{w}^\top \mathbf{x}_i)\mathbf{x}_i$$

- there is a unique solution for $\mathbf{w}$
- in practice more efficient Newton methods are used to minimize L
- there can be problems with $\mathbf{w}$ becoming infinite for linearly separable data

Gradient descent algorithm for SVM

First, rewrite the optimization problem as an average

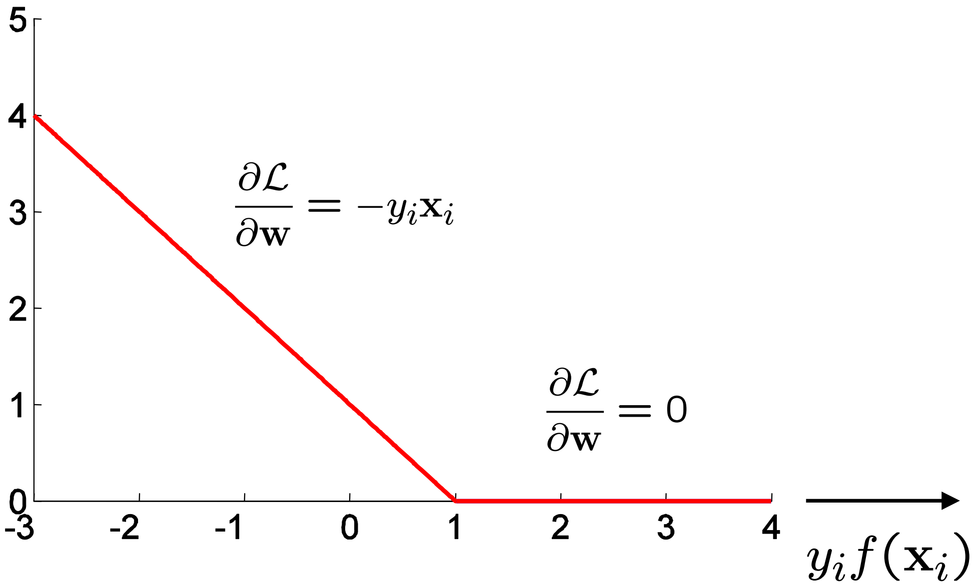
$$\begin{aligned}
\min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) &= \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{N}\sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) \\
&= \frac{1}{N}\sum_i^N \left(\frac{\lambda}{2}||\mathbf{w}||^2 + \max(0, 1 - y_i f(\mathbf{x}_i))\right)
\end{aligned}$$

(with $\lambda = 2/(NC)$ up to an overall scale of the problem) and $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$

Because the hinge loss is not differentiable, a sub-gradient is computed

Sub-gradient for hinge loss

$$\mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}) = \max(0, 1 - y_i f(\mathbf{x}_i)) \qquad f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b$$



Sub-gradient descent algorithm for SVM

$$\mathcal{C}(\mathbf{w}) = \frac{1}{N} \sum_i^N \left(\frac{\lambda}{2} ||\mathbf{w}||^2 + \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}) \right)$$

The iterative update is

$$\begin{aligned} \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta \nabla_{\mathbf{w}_t} \mathcal{C}(\mathbf{w}_t) \\ &\leftarrow \mathbf{w}_t - \eta \frac{1}{N} \sum_i^N (\lambda \mathbf{w}_t + \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}_t)) \end{aligned}$$

where η is the learning rate.

Then each iteration t involves cycling through the training data with the updates:

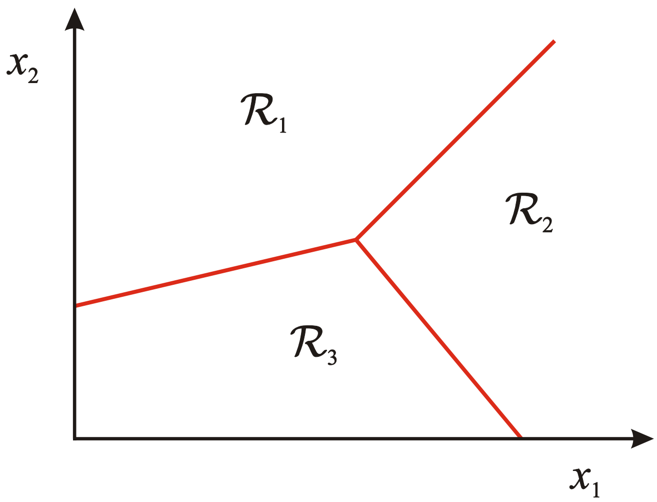
$$\begin{aligned} \mathbf{w}_{t+1} &\leftarrow (1 - \eta\lambda)\mathbf{w}_t + \eta y_i \mathbf{x}_i & \text{if } y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 1 \\ &\leftarrow (1 - \eta\lambda)\mathbf{w}_t & \text{otherwise} \end{aligned}$$

Multi-class Classification

Multi-Class Classification – what we would like

Assign input vector $\mathbf{x}$ to one of K classes C_k

Goal: a decision rule that divides input space into K *decision regions* separated by *decision boundaries*



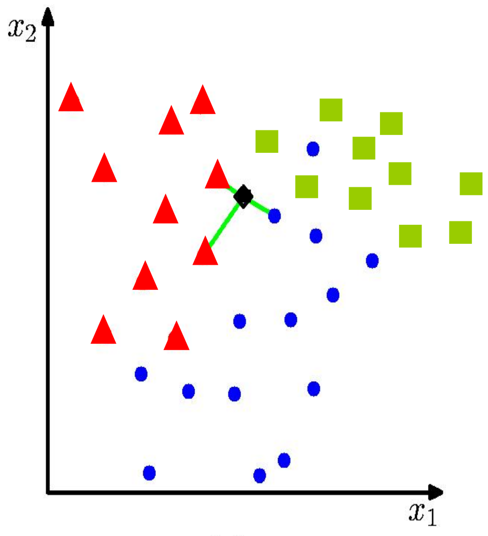
Reminder: K Nearest Neighbour (K-NN) Classifier

Algorithm

- For each test point, x, to be classified, find the K nearest samples in the training data
- Classify the point, x, according to the majority vote of their class labels

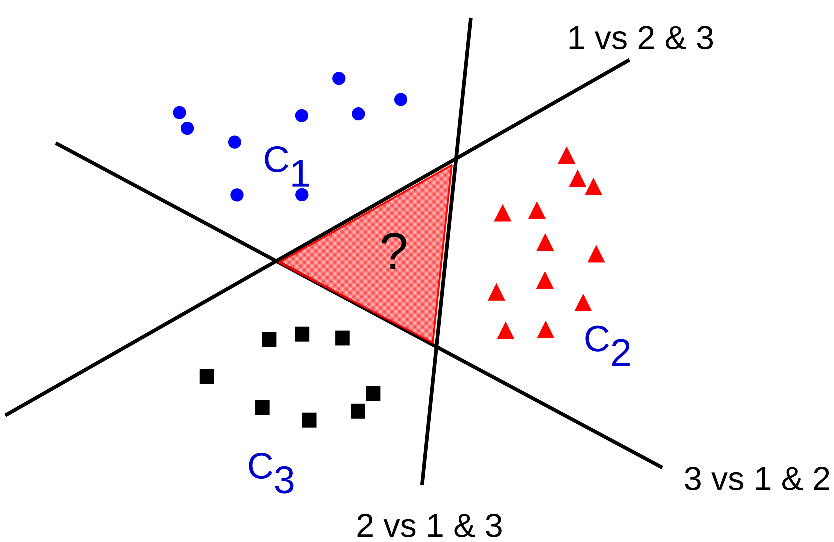
e.g. K = 3

• naturally applicable to multi-class case



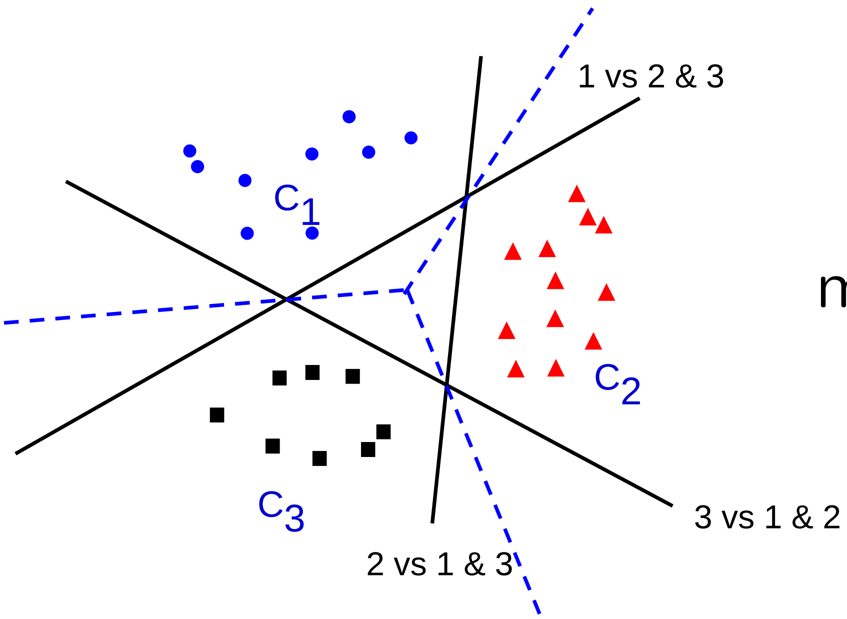
Build from binary classifiers …

• Learn: K two-class 1 vs the rest classifiers f_k (**x**)



Build from binary classifiers …

- Learn: K two-class 1 vs the rest classifiers f_k (**x**)
- Classification: choose class with most positive score



$$\max_k f_k(\mathbf{x})$$

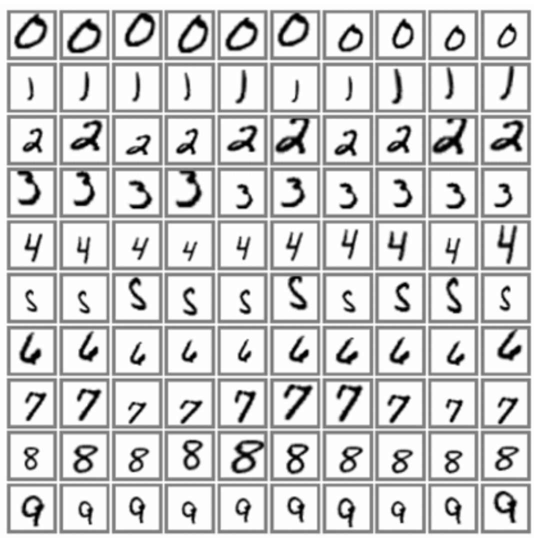
Application: hand written digit recognition

- Feature vectors: each image is 28 x 28 pixels. Rearrange as a 784-vector **x**

- Training: learn k=10 two-class 1 vs the rest SVM classifiers f_k (**x**)

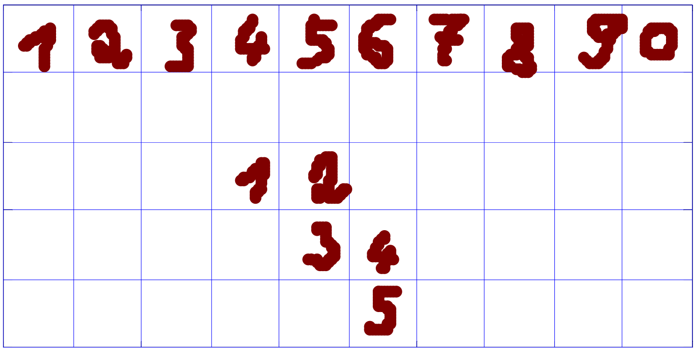
- Classification: choose class with most positive score

$$f(\mathbf{x}) = \max_k f_k(\mathbf{x})$$



Example

hand
drawn



classification

1	2	3	4	5	6	7	8	9	0
			1	2					
				3	4				
					5				

Background reading and more

- Other multiple-class classifiers (not covered here):
    - Neural networks
    - Random forests
- Bishop, chapters 4.1 – 4.3 and 14.3
- Hastie et al, chapters 10.1 – 10.6
- More on web page:
  http://www.robots.ox.ac.uk/~az/lectures/ml