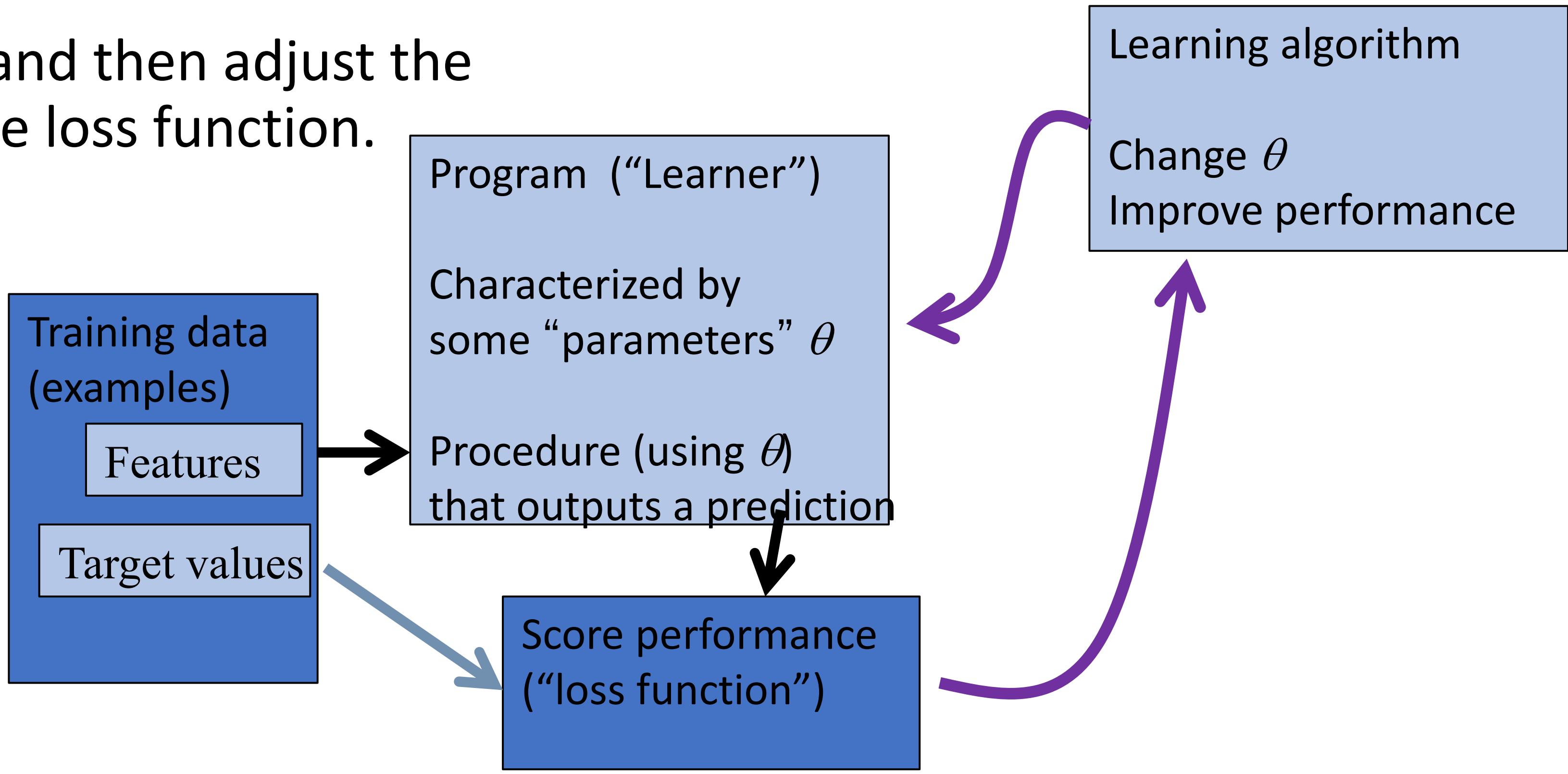
Linear Classifiers

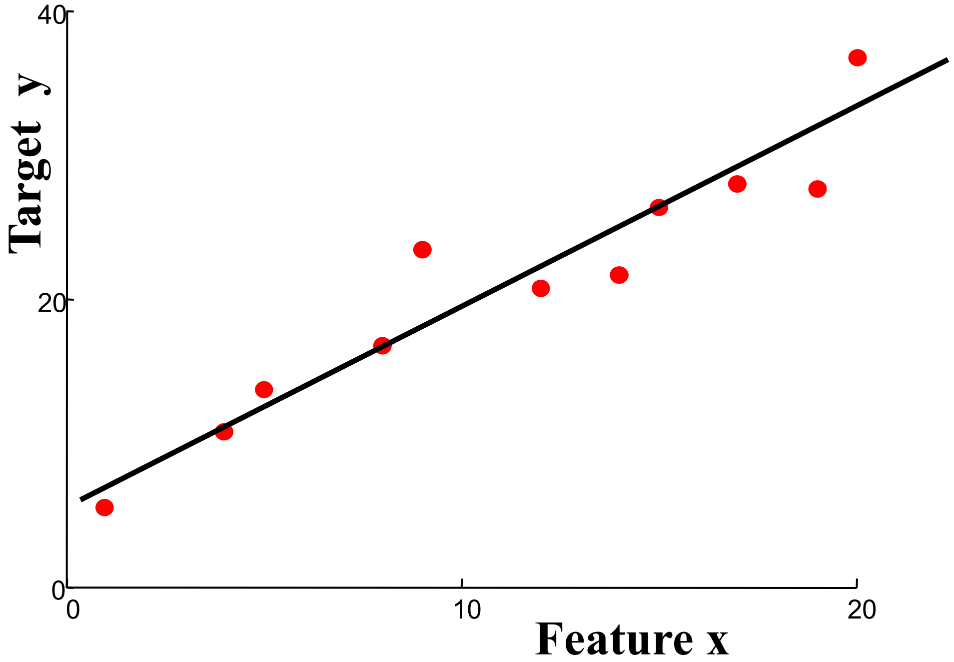
Adopted from slides by Alexander Ihler

Supervised Learning

- **Given** examples of a function $(X, Y = F(X))$
- **Find** function $\hat{Y} = h(X)$ to estimate $F(X)$
  - Discrete Y: Classification
- Formulate a loss function and then adjust the parameters to minimize the loss function.



Linear regression



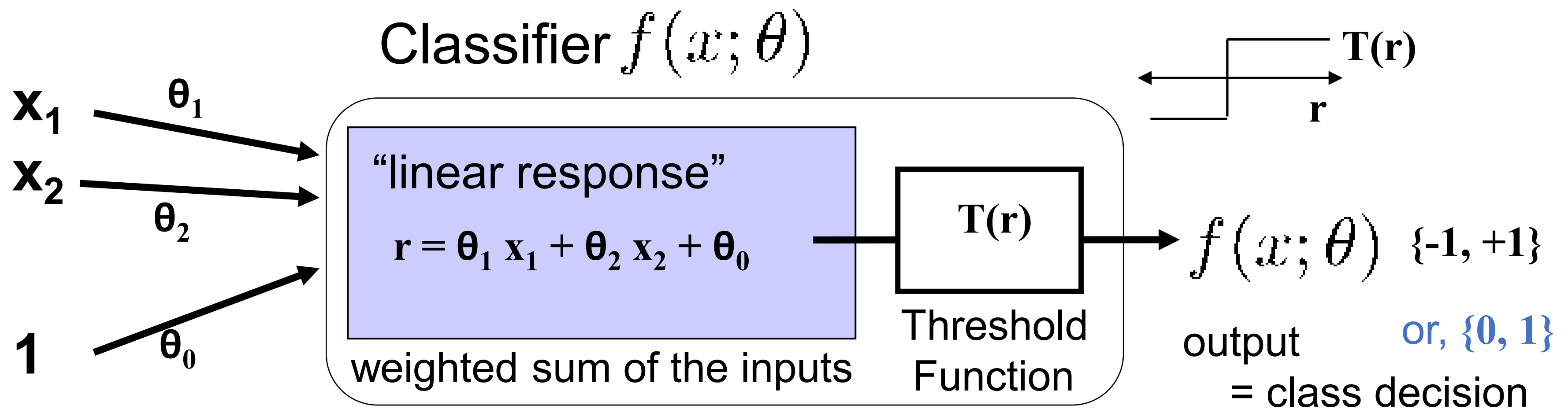
"Predictor":
Evaluate line:

$$r = \theta_0 + \theta_1 x_1$$

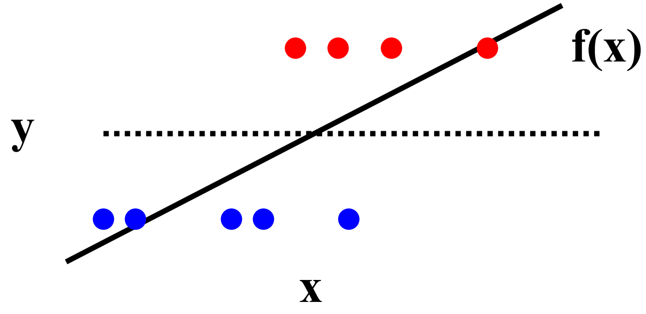
return r

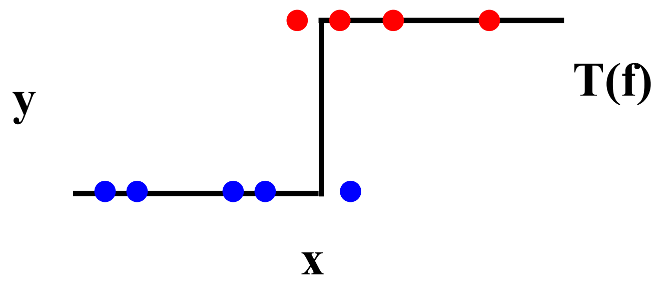
- Contrast with classification
  - Classify: predict discrete-valued target y
  - Initially: "classic" binary { -1, +1} classes; generalize later

Linear Classifier (2 features)



Visualizing for one feature "x":

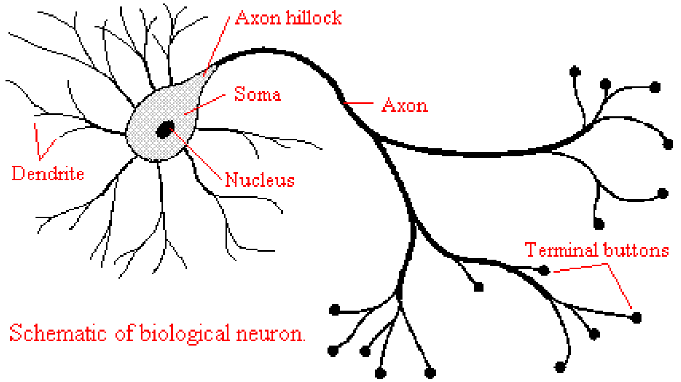




Notations

- Inputs:
  - $x_1, x_2, \ldots\ldots\ldots\ldots, x_{n-1}, x_n$ are the values of the n features
  - $x_0 = 1$ (a constant input)
  - $x = (x_0, x_1, x_2, \ldots\ldots\ldots\ldots, x_n)$: feature vector
- Weights (parameters):
  - $\theta_0, \theta_1, \theta_2, \ldots\ldots\ldots\ldots, \theta_n$,
  - we have n+1 weights: one for each feature + one for the constant
  - $\theta = (\theta_0, \theta_1, \theta_2, \ldots\ldots\ldots\ldots, \theta_n)$: parameter vector
- Linear response
  - $\theta_0 x_0 + \theta_1 x_1 + \ldots \theta_n x_n = \theta x$
- Threshold function
  - $T(r)$
- Linear classifier
  - $f(x; \theta) = T(\theta x)$

Perceptrons

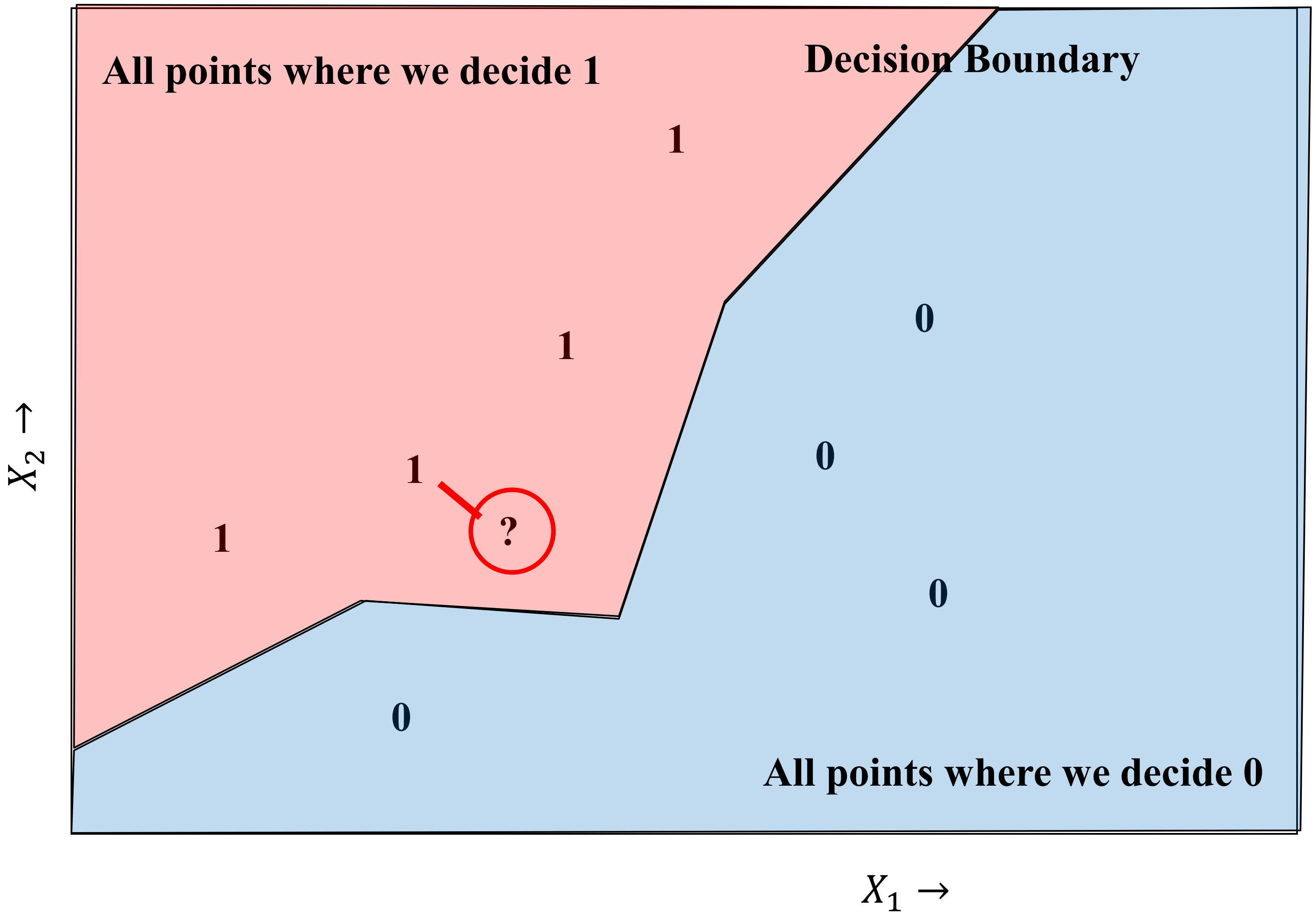


- Perceptron = a linear classifier
  - The parameters θ are sometimes called weights ("w")
    - real-valued constants (can be positive or negative)
  - Input features $x_1 \ldots x_n$;

- A perceptron calculates 2 quantities:
  - 1. A weighted sum of the input features
  - 2. This sum is then thresholded by the T(.) function

- Perceptron: a simple artificial model of human neurons
  - weights = "synapses"
  - threshold = "neuron firing"

Nearest neighbor classifier

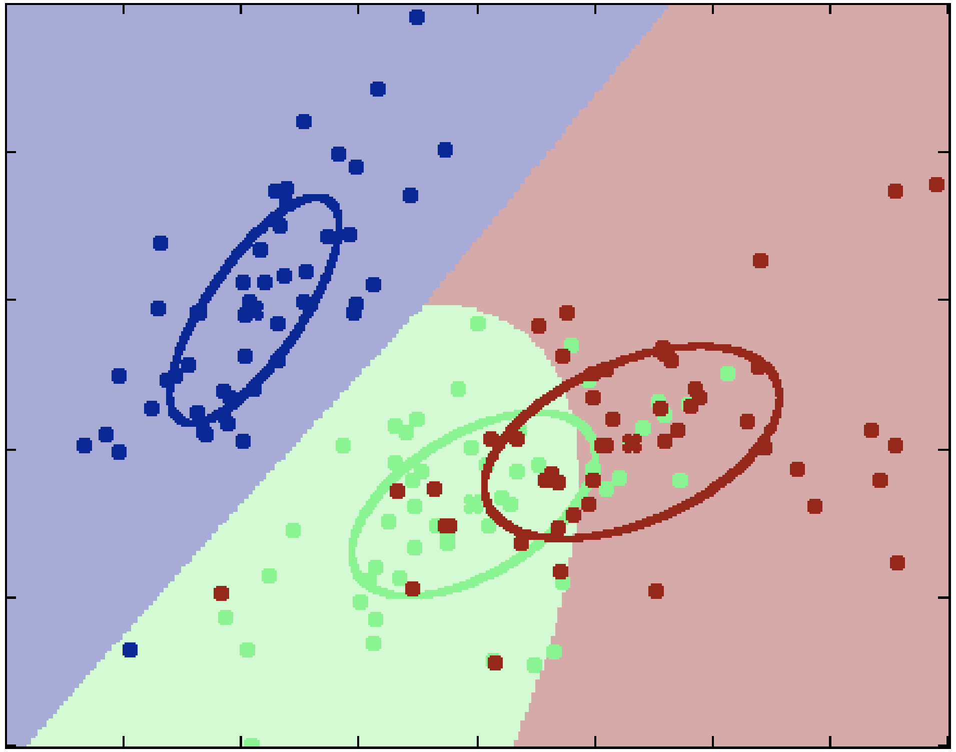


Example: Gaussian Bayes for Iris Data

- Fit Gaussian distribution to each class {0,1,2}

$$p(y) = \text{Discrete}(\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3})$$

$$p(x_1, x_2 | y = 0) = \mathcal{N}(x\,;\,\mu_0, \Sigma_0)$$
$$p(x_1, x_2 | y = 1) = \mathcal{N}(x\,;\,\mu_1, \Sigma_1)$$
$$p(x_1, x_2 | y = 2) = \mathcal{N}(x\,;\,\mu_2, \Sigma_2)$$



Perceptron Decision Boundary

- The perceptron is defined by the decision algorithm:

$$f(x;\theta) = \begin{cases} +1 & \text{if } \theta \cdot x^T > 0 \\ -1 & \text{otherwise} \end{cases}$$

$\boxed{\text{or } f(x;\theta) = T(\theta x)}$

- The perceptron represents a hyperplane decision surface in n-dimensional space
  - A point in 1D, a line in 2D, a plane in 3D, etc.

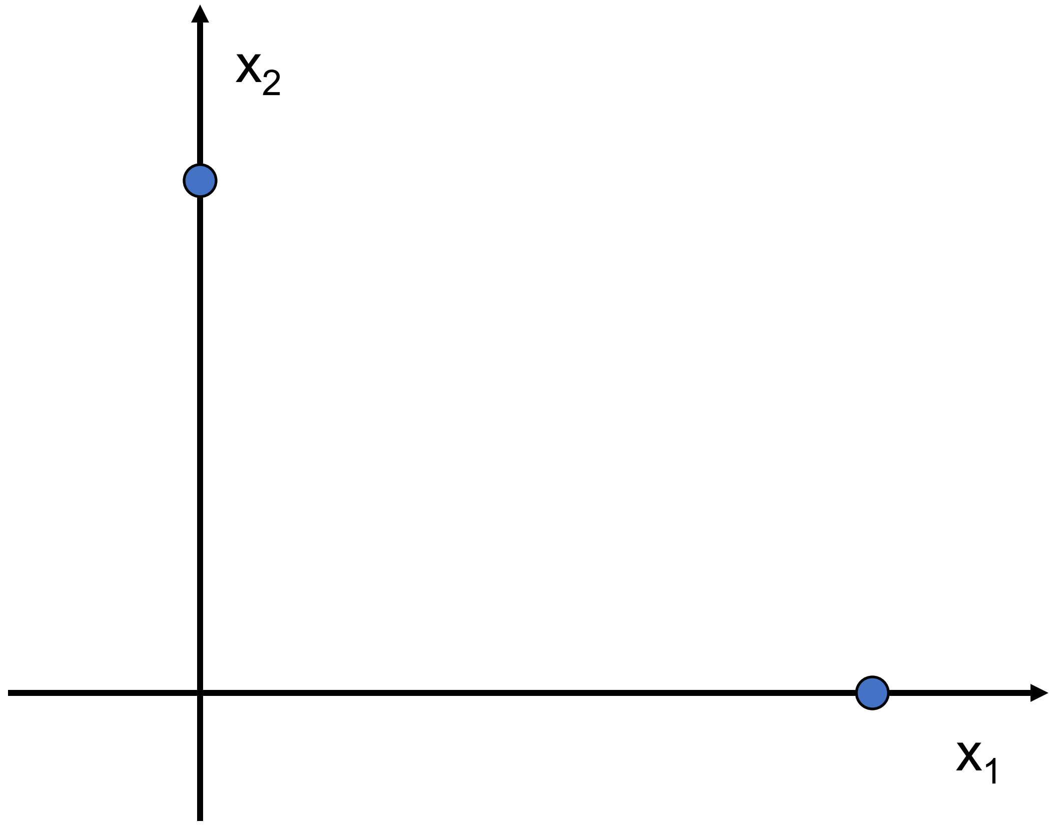
- The equation of the hyperplane is given by

$$\theta \cdot \underline{x}^T = 0$$

This defines the set of points that are on the boundary.

Example, Linear Decision Boundary

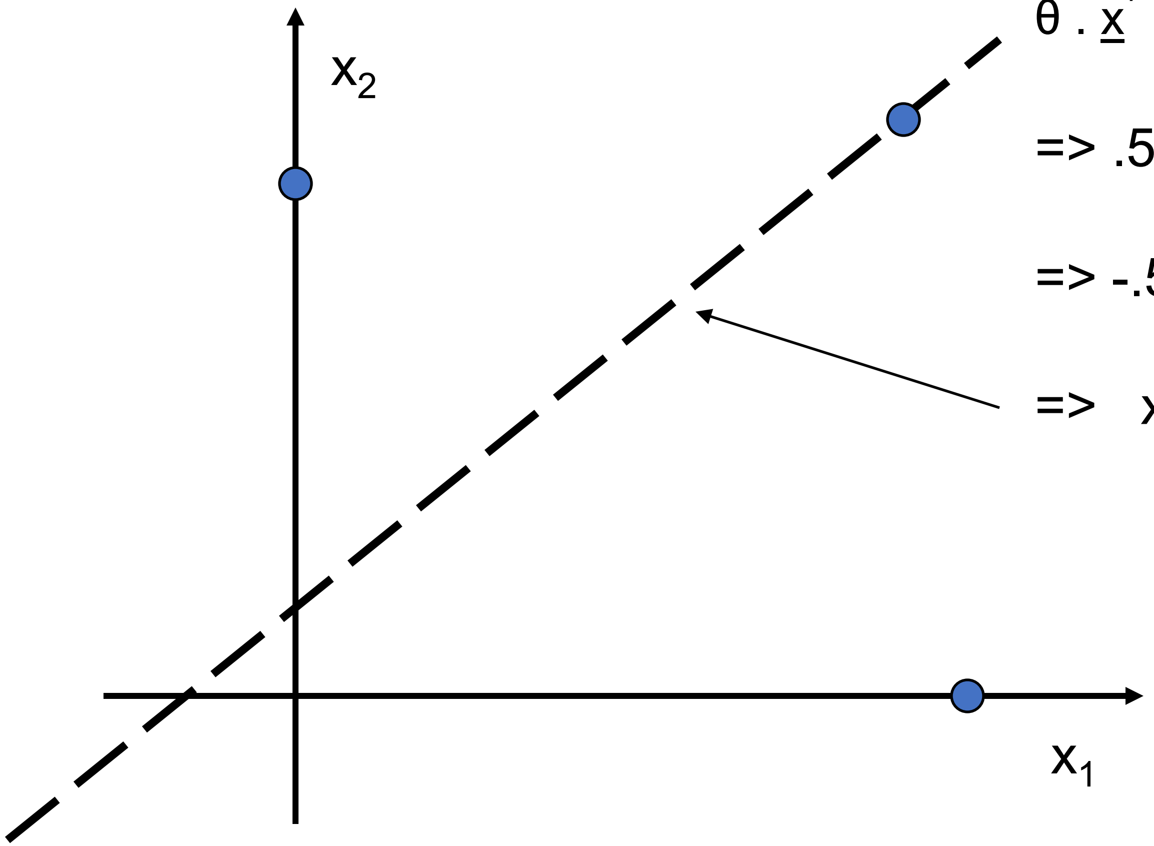
$$\theta = (\theta_0, \theta_1, \theta_2) = (1, .5, -.5)$$



From P. Smyth

Example, Linear Decision Boundary

$$\theta = (\theta_0, \ \theta_1, \ \theta_2) = (1, \ .5, \ -.5)$$



$\theta \,.\, \underline{x}' = 0$

$\Rightarrow .5 * x_1 - .5 * x_2 + 1 * 1 = 0$

$\Rightarrow -.5\, x_2 = -.5\, x_1 - 1$

$\Rightarrow \ x_2 = x_1 + 2$

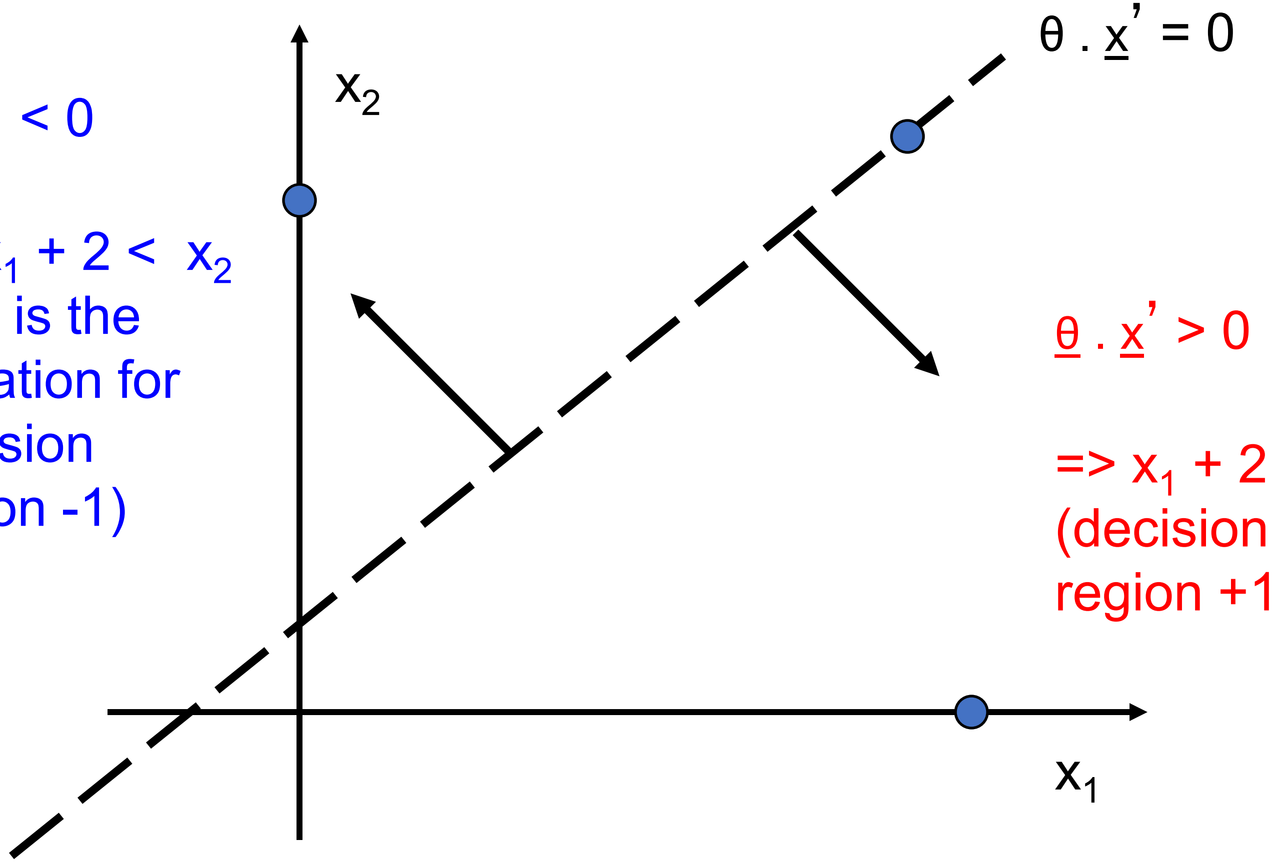
From P. Smyth

Example, Linear Decision Boundary

$$\theta = (\theta_0,\ \theta_1,\ \theta_2) = (1,\ .5,\ -.5)$$

$\underline{\theta} \,.\, \underline{x}' < 0$

$\Rightarrow x_1 + 2 < x_2$
(this is the equation for decision region -1)



$\underline{\theta} \,.\, \underline{x}' > 0$

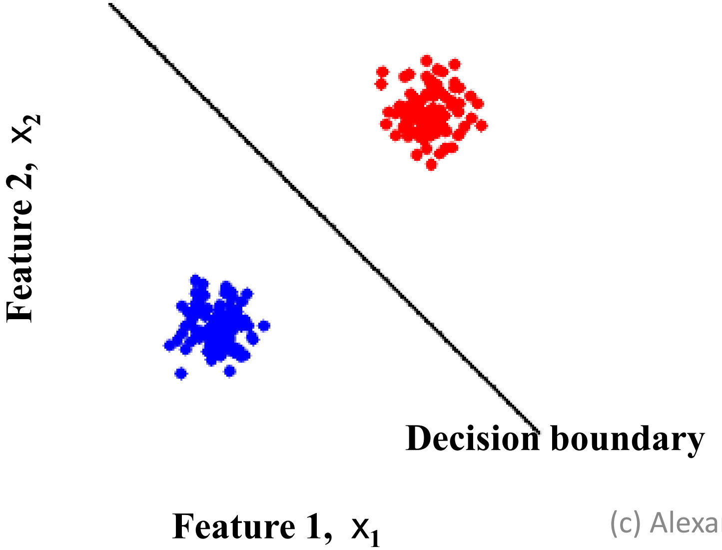
$\Rightarrow x_1 + 2 > x_2$
(decision region +1)

From P. Smyth

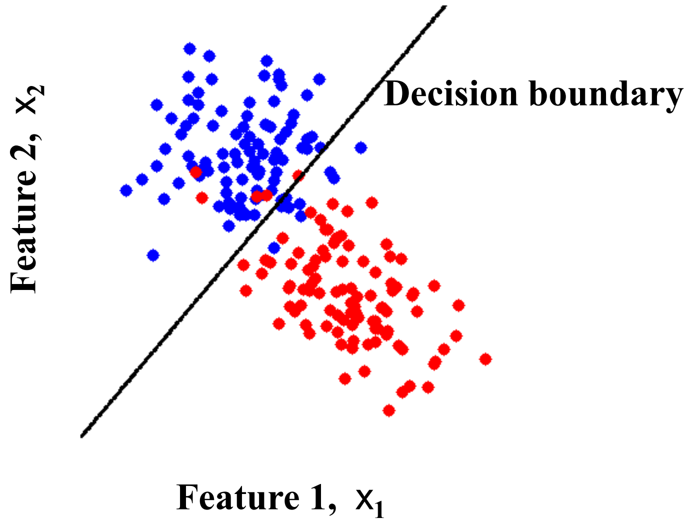
Separability

- A data set is separable by a learner if
  - There is some instance of that learner that correctly predicts all the data points
- Linearly separable data
  - Can separate the two classes using a straight line in feature space
  - in 2 dimensions the decision boundary is a straight line

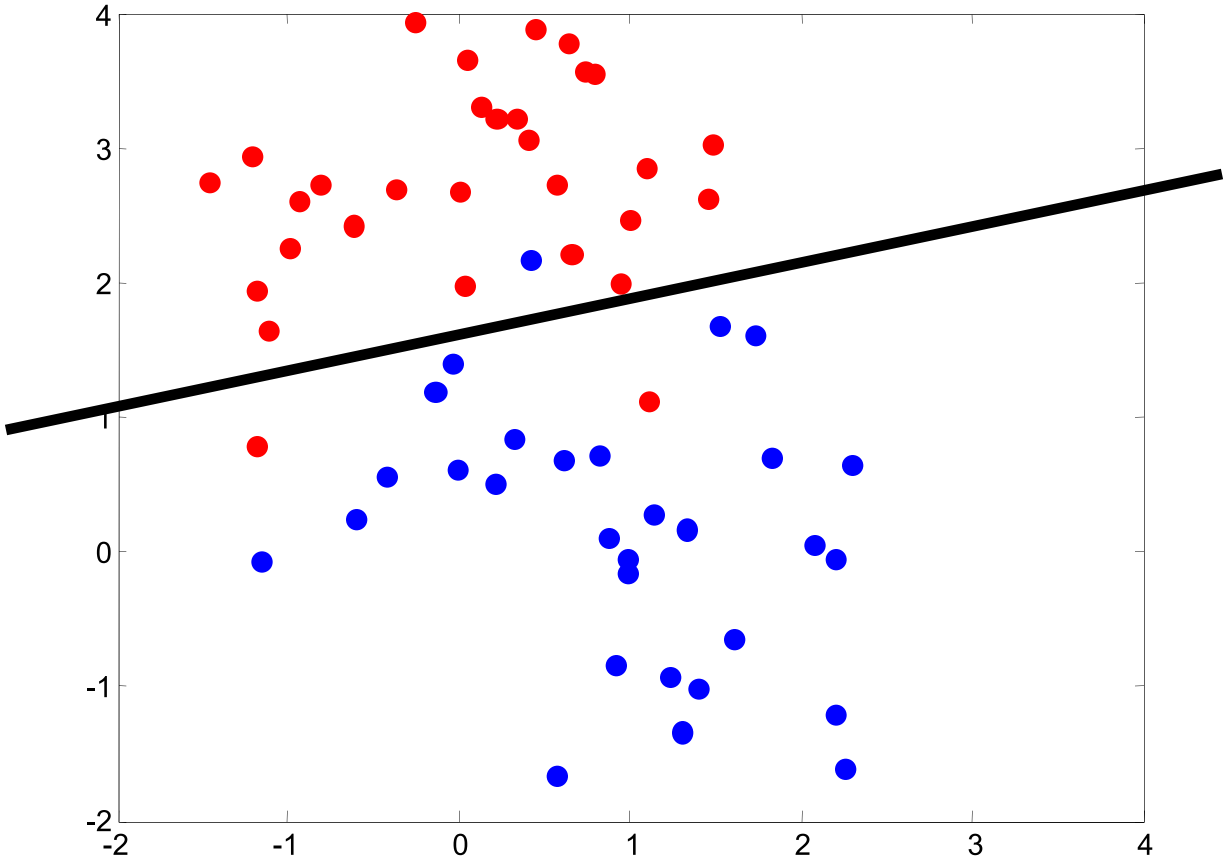
Linearly separable data



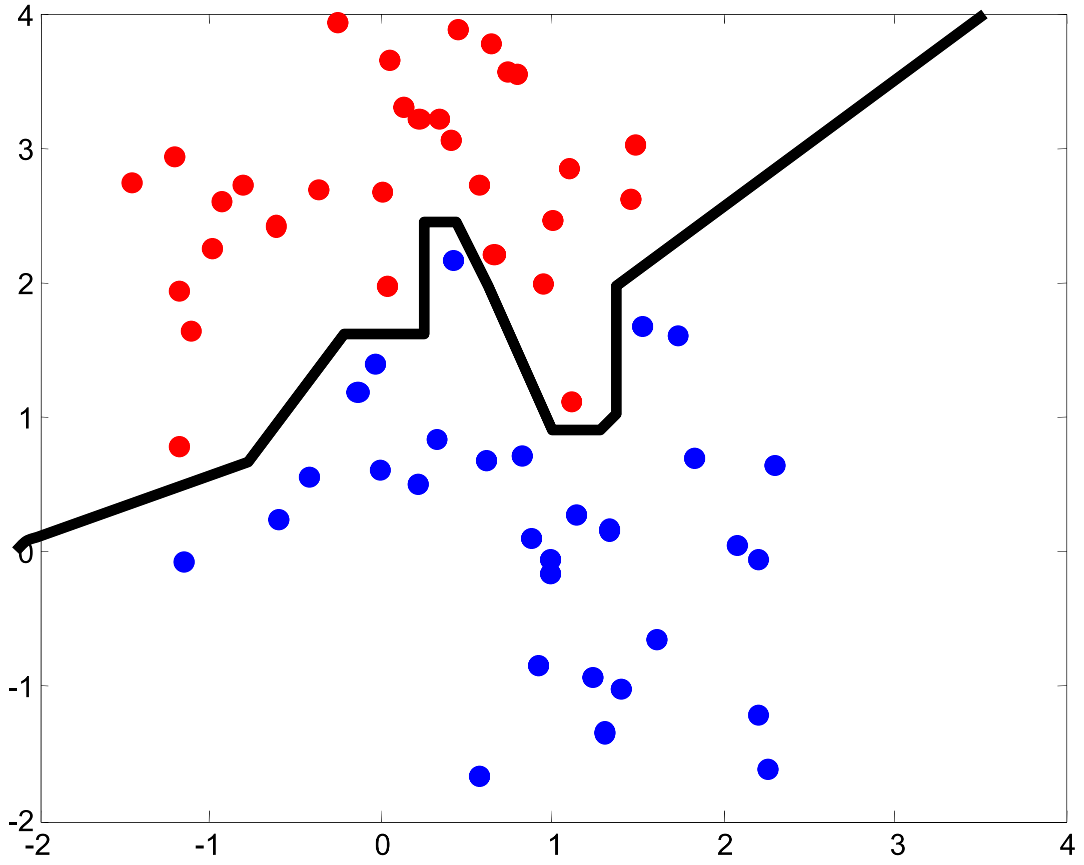
Linearly non-separable data



Another example



Non-linear decision boundary

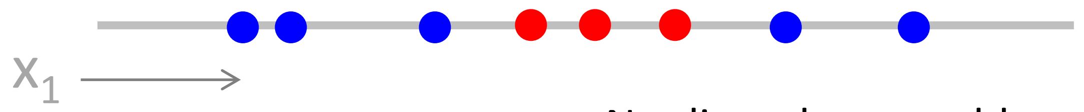


(c) Alexander Ihler

Adding features

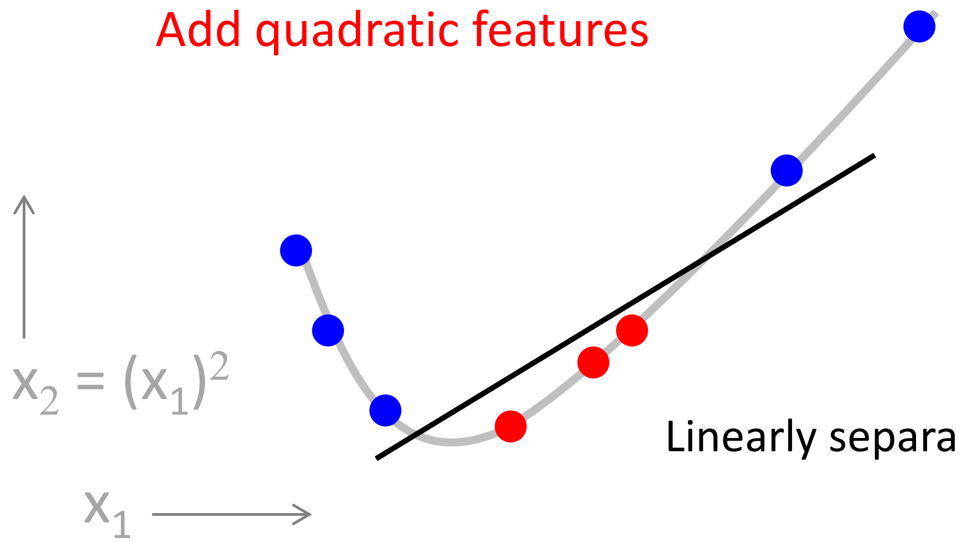
- Linear classifier can't learn some functions

1D example:



Not linearly separable

Add quadratic features

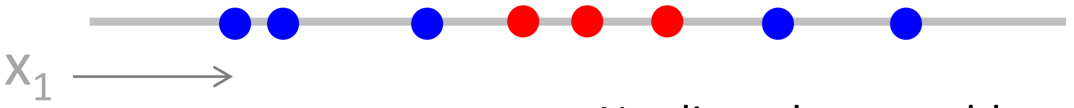


Linearly separable in new features…

Adding features

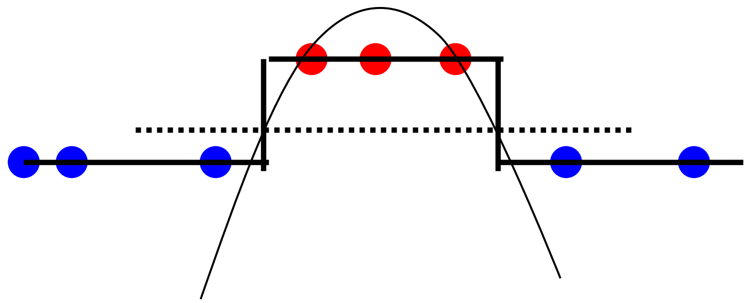
- Linear classifier can't learn some functions

1D example:



Not linearly separable

Quadratic features, visualized in original feature space:

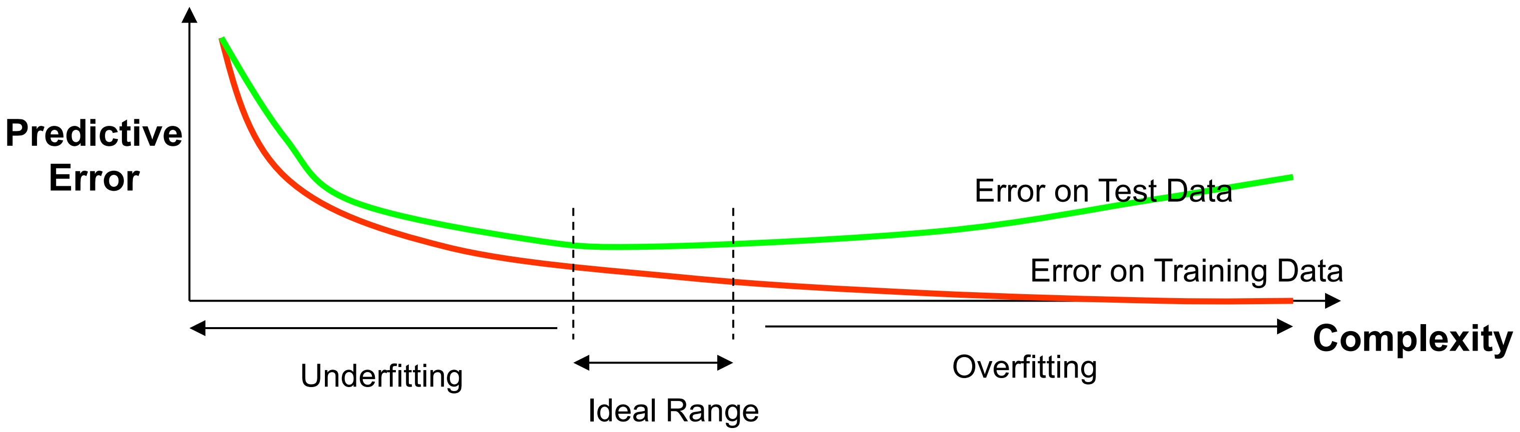


$$y = T(\, a\, x^2 + b\, x + c\,)$$

More complex decision boundary: $ax^2+bx+c = 0$

Effect of dimensionality

- Data are increasingly separable in high dimension – is this a good thing?
- "Good"
  - Separation is easier in higher dimensions (for fixed # of data m)
  - Increase the number of features, and even a linear classifier will eventually be able to separate all the training examples!
- "Bad"
  - Remember training vs. test error? Remember overfitting?
  - Increasingly complex decision boundaries can eventually get all the training data right, but it doesn't necessarily bode well for test data…



Linear Classifiers: Learning

Learning the Classifier Parameters

- Learning from Training Data:
  - training data = labeled feature vectors
  - Find parameter values that predict well (low error)
    - error is estimated on the training data
    - "true" error will be on future test data
- Define a loss function $J(\theta)$:
  - Classifier error rate (for a given set of weights $\underline{\theta}$ and labeled data)
- Minimize this loss function (or, maximize accuracy)
  - An optimization or search problem over the vector $(\theta_1, \theta_2, \theta_0)$

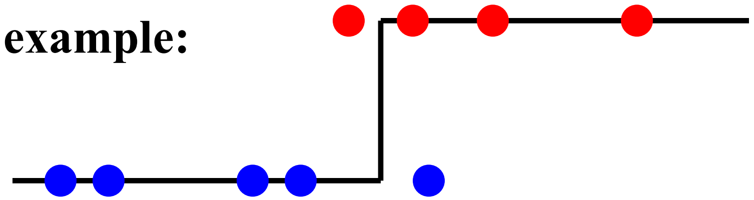
Training a linear classifier

- How should we measure error?
  - Natural measure = “fraction we get wrong”   (error rate)

$$\text{err}(\theta) = \frac{1}{m} \sum_i \mathbb{1}\left[y^{(i)} \neq f(x^{(i)};\theta)\right] \quad \text{where} \quad \mathbb{1}\left[y \neq \hat{y}\right] = \begin{cases} 1 & y \neq \hat{y} \\ 0 & \text{o.w.} \end{cases}$$

- But, hard to train via gradient descent
  - Not continuous
  - As decision boundary moves, errors change abruptly

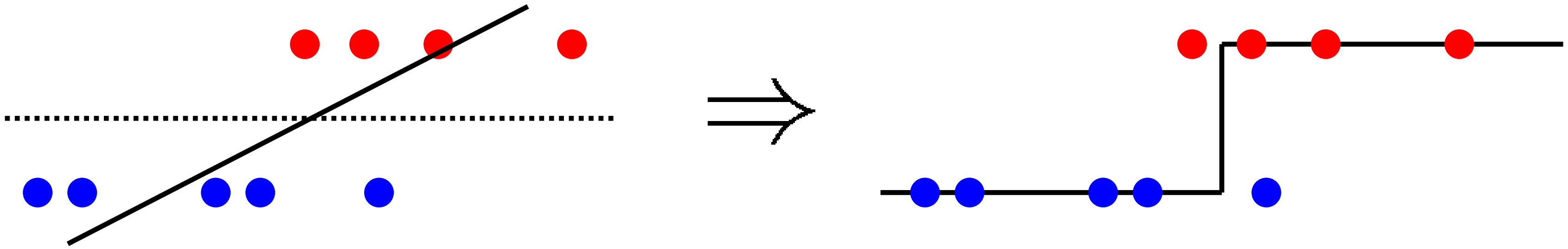
1D example:



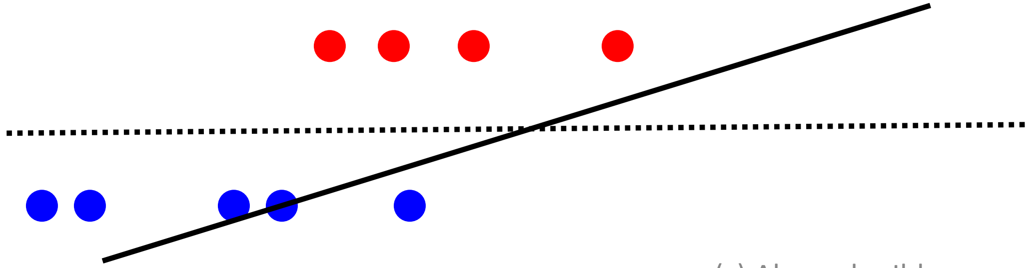
T(f) = -1 if f < 0
T(f) = +1 if f > 0

Linear regression?

- Simple option: set θ using linear regression



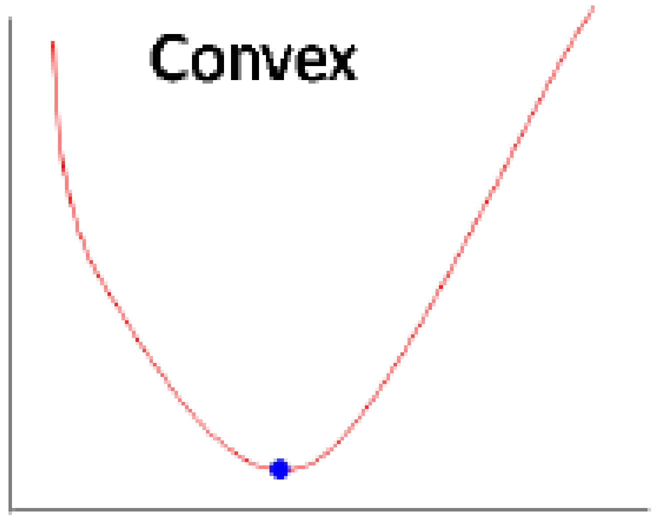
- In practice, this often doesn't work so well…
  - Consider adding a distant but "easy" point
  - MSE distorts the solution

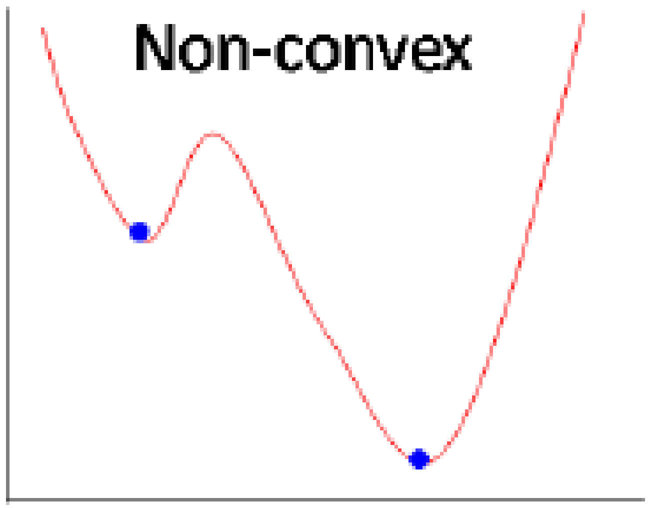


Surrogate loss functions

- Another solution: use a “smooth” loss
  - e.g., use a smooth surrogate function to approximate the loss function

Convex?





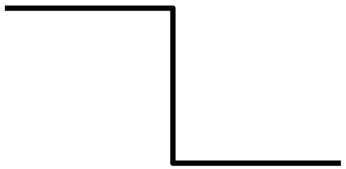
Surrogate loss functions

Class y = {-1, 1}

$$J(\theta) = \frac{1}{m}\sum_i \mathbf{1}\left[y^{(i)} \neq \operatorname{sign}\left(\theta x^{(i)}\right)\right]$$

$$= \frac{1}{m}\sum_i \mathbf{1}\left[y^{(i)} \cdot \theta x^{(i)} < 0\right]$$

$$= \frac{1}{m}\sum_i \boldsymbol{L}\left(y^{(i)} \cdot \theta x^{(i)}\right)$$



0 / 1 Loss

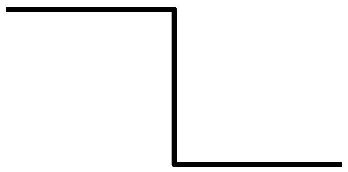
Surrogate loss functions

Class y = {0, 1}

$$J(\theta) = \frac{1}{m}\sum_i \mathbf{1}\left[y^{(i)} \neq \mathbf{1}[\theta x^{(i)} > 0]\right]$$

$$= \frac{1}{m}\sum_i \left(y^{(i)}\mathbf{1}[\theta x^{(i)} < 0] + (1 - y^{(i)})\mathbf{1}[\theta x^{(i)} > 0]\right)$$

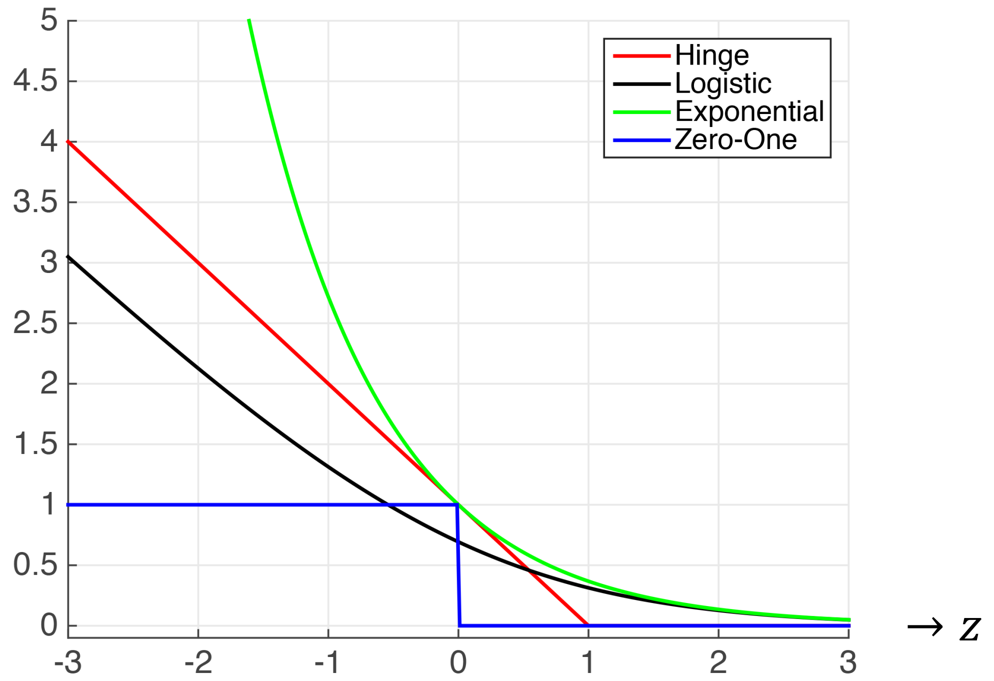
$$= \frac{1}{m}\sum_i \left(y^{(i)}\boldsymbol{L}(\theta x^{(i)}) + (1 - y^{(i)})\boldsymbol{L}(-\theta x^{(i)})\right)$$



0 / 1 Loss

Surrogate loss functions

- 0-1: $L(z) = \mathbf{1}[z < 0]$
- Logistic: $L(z) = -\log \sigma(z) = -\log \frac{1}{1+e^{-z}}$
- Exponential: $L(z) = e^{-\beta z}$
- Hinge: $L(z) = \max\{0, 1 - z\}$
- ...



Generic classification formulation

Class y = {0, 1}

$$J(\theta) = \frac{1}{m}\sum_i \left(y^{(i)}\phi\left(\theta x^{(i)}\right) + \left(1 - y^{(i)}\right)\phi\left(-\theta x^{(i)}\right)\right)$$

Class y = {-1, 1}

$$J(\theta) = \frac{1}{m}\sum_i \phi\left(y^{(i)}\theta x^{(i)}\right)$$

Generic classification formulation w/ Regularization

Class y = {0, 1}

$$J(\theta) = \frac{1}{m}\sum_{i}\left(y^{(i)}\phi\left(\theta x^{(i)}\right) + \left(1 - y^{(i)}\right)\phi\left(-\theta x^{(i)}\right)\right) + \frac{\lambda}{2m}\left|\left|\theta\right|\right|^2$$

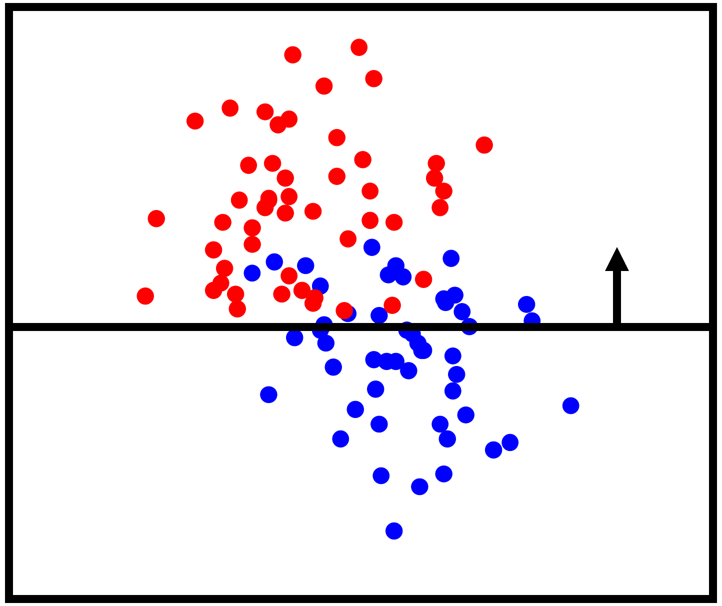
Class y = {-1, 1}

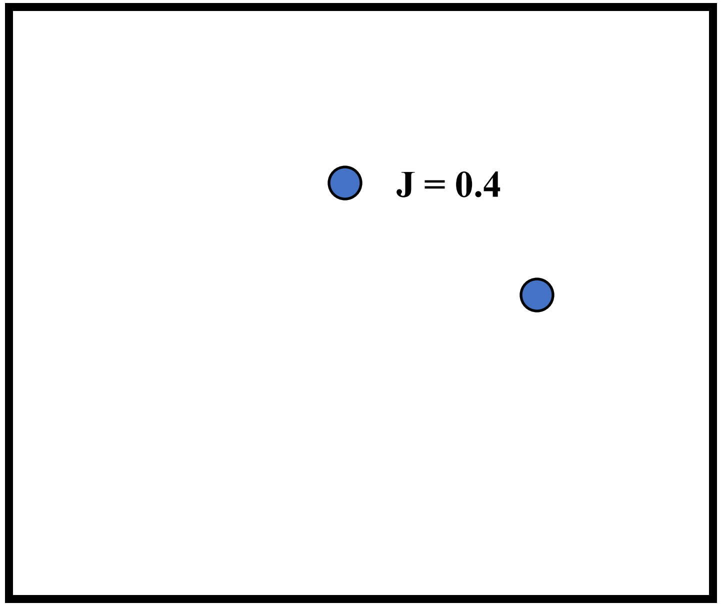
$$J(\theta) = \frac{1}{m}\sum_{i}\phi\left(y^{(i)}\theta x^{(i)}\right) + \frac{\lambda}{2m}\left|\left|\theta\right|\right|^2$$

Training the Classifier

- Once we have a smooth measure of quality, we can find the "best" settings for the parameters

- Example: 2D feature space ⇔ parameter space

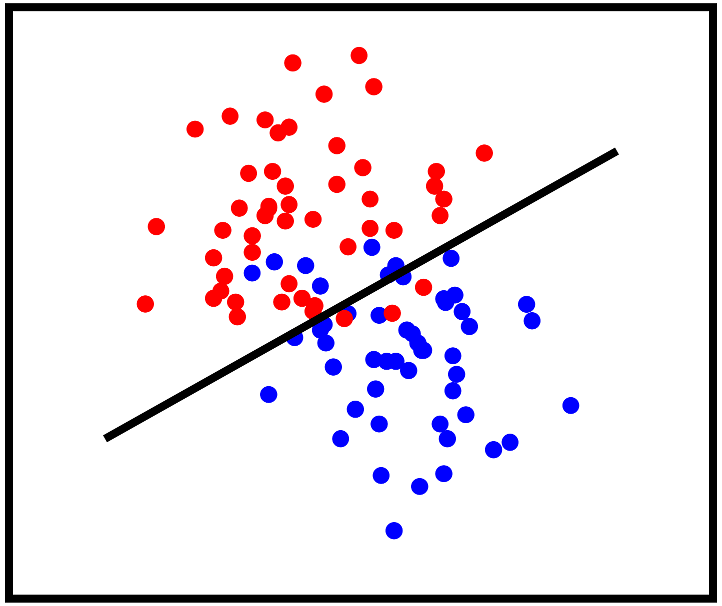


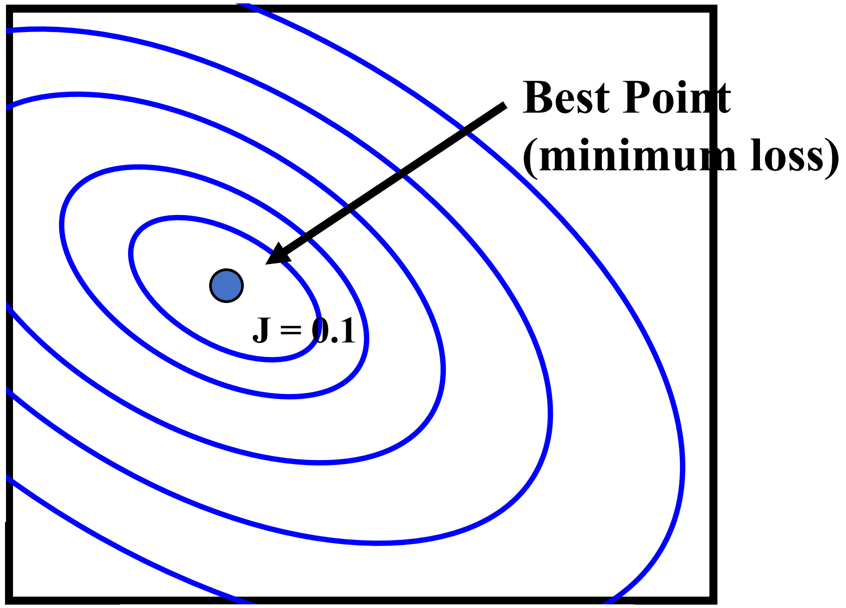


Training the Classifier

- Once we have a smooth measure of quality, we can find the "best" settings for the parameters

- Example: 2D feature space ⇔ parameter space

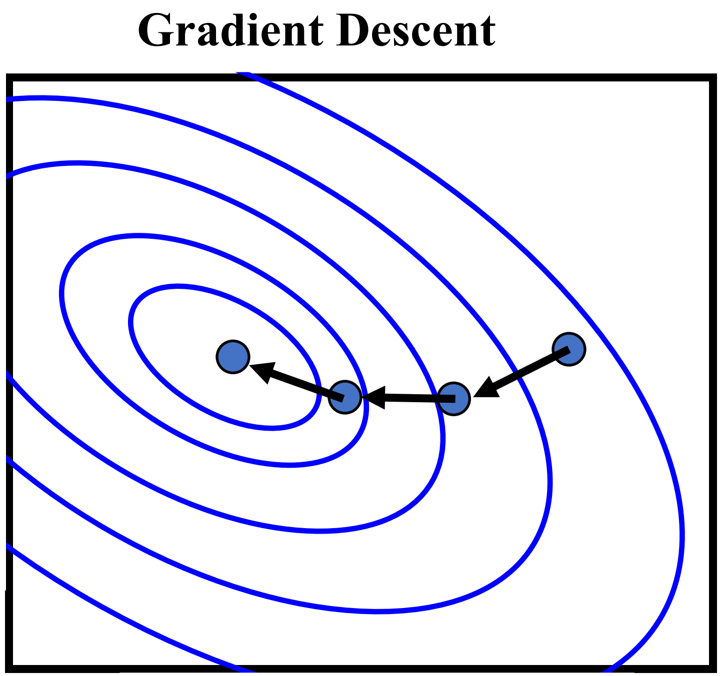




Minimizing the loss function

- As in linear regression, this is now just optimization

- Methods:
  - Gradient descent
    - Improve loss by small changes in parameters (“small” = learning rate)



Gradient of general loss functions

Class y = {0, 1}

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_i \left(y^{(i)} \frac{\partial \phi\left(\theta x^{(i)}\right)}{\partial \theta_j} + \left(1 - y^{(i)}\right) \frac{\partial \phi\left(-\theta x^{(i)}\right)}{\partial \theta_j} \right)$$

Class y = {-1, 1}

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_i \frac{\partial \phi\left(y^{(i)} \theta x^{(i)}\right)}{\partial \theta_j}$$

Logistic Regression

Maximum likelihood learning

joint probability

conditional probability

$$p(D)\,p(W \mid D) = p(D,W) = p(W)\,p(D \mid W)$$

Prior probability of weight vector W

Probability of observed data given W

$$\boxed{p(W \mid D)} = \frac{\boxed{p(W)}\;\boxed{p(D \mid W)}}{\boxed{p(D)}}$$

Posterior probability of weight vector W given training data D

$$\boxed{\int_W p(W)\,p(D \mid W)}$$

Maximize sums of log probs

- We want to maximize the product of the probabilities of the outputs on the training cases
  - Assume the output errors on different training cases, i, are independent.

$$p(D|W) = \prod_i p(d^{(i)}\,|W)$$

- Because the log function is monotonic, it does not change where the maxima are. So we can maximize sums of log probabilities

$$\log p\,(D|W) = \sum_i \log p\,(d^{(i)}\,|W)$$

- This is called maximum likelihood learning.

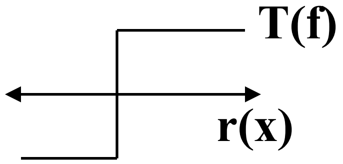
Minimum negative log-likelihood: $$-\log p\,(D|W) = -\sum_i \log p\,(d^{(i)}\,|W)$$

For classification…

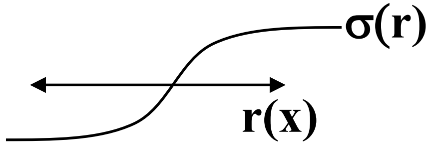
- Minimum negative log-likelihood:

$$-\log p\,(Y|X,\theta) = -\sum_i \log p\,(y^{(i)}\,|x^{(i)},\theta)$$

Logistic regression

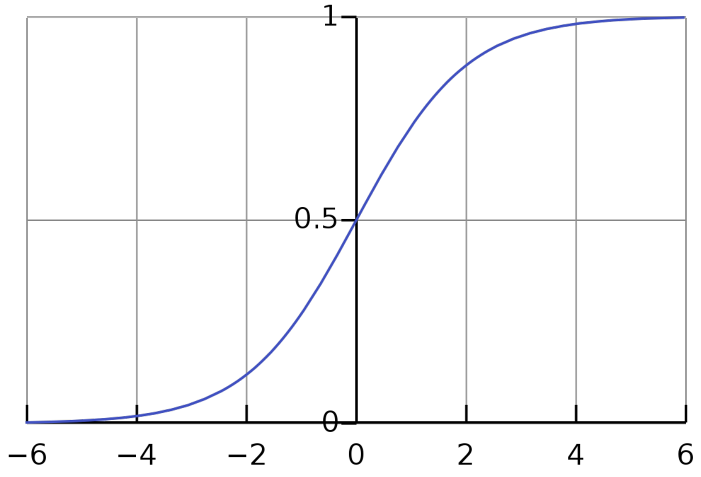


- Use a "smooth" function to approximate the threshold function $T(r) \Rightarrow \sigma(r)$



- Logistic "sigmoid", looks like an "S"

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



Logistic regression

- Intepret $\sigma(\theta x)$ as a probability that $y = 1$, i.e., $P(Y = 1|x;\theta) = \sigma(\theta x)$
- Use a negative log-likelihood loss function
  - If $y = 1$, loss is $-\log P[y = 1] \;=\; -\log \sigma(\theta x)$
  - If $y = 0$, loss is $-\log P[y = 0] \;=\; -\log(1 - \sigma(\theta x))$

- Can write this succinctly:

$$J(\underline{\theta}) = -\frac{1}{m}\Big(\sum_i \underbrace{y^{(i)} \log \sigma(\theta \cdot x^{(i)})}_{\text{Nonzero only if y=1}} + \underbrace{(1-y^{(i)}) \log(1-\sigma(\theta \cdot x^{(i)}))}_{\text{Nonzero only if y=0}}\Big)$$

Logistic regression

$$J(\underline{\theta}) = -\frac{1}{m}\Big(\sum_i y^{(i)} \log\sigma(\theta\cdot x^{(i)}) + (1-y^{(i)})\log(1-\sigma(\theta\cdot x^{(i)}))\Big)$$

$$J(\theta) = \frac{1}{m}\sum_i \left(y^{(i)}\phi\left(\theta x^{(i)}\right) + \left(1-y^{(i)}\right)\phi\left(-\theta x^{(i)}\right)\right)$$

$$\phi(z) = -\log\left(\sigma(z)\right) = -\log\frac{1}{1+e^{-z}}$$

$$\phi(-z) = -\log\left(\sigma(-z)\right) = -\log\frac{1}{1+e^{z}} = -\log\frac{e^{-z}}{1+e^{-z}} = -\log\left(1-\frac{1}{1+e^{-z}}\right) = -\log\left(1-\sigma(z)\right)$$

Gradient Equations

$$(\ln z)' = \frac{1}{z}$$

$$(\sigma(z))' = \sigma(z)\big(1 - \sigma(z)\big)$$

- Logistic neg-log likelihood loss:

$$J(\underline{\theta}) = -\frac{1}{m}\Big(\sum_i y^{(i)} \log \sigma(\theta \cdot x^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(\theta \cdot x^{(i)}))\Big)$$

- What's the derivative with respect to one of the parameters?

$$\frac{\partial J(\theta)}{\partial \theta_j}$$

$$= -\frac{1}{m}\sum_i \Big(y^{(i)} \Big(1 - \sigma\big(\theta x^{(i)}\big)\Big) x_j^{(i)} - \big(1 - y^{(i)}\big)\sigma\big(\theta x^{(i)}\big) x_j^{(i)}\Big)$$

$$= \frac{1}{m}\sum_i \big(\sigma\big(\theta x^{(i)}\big) - y^{(i)}\big) x_j^{(i)}$$

(Batch) Gradient descent

Initialize θ
Do {
  $\theta \leftarrow \theta - \alpha \nabla J(\theta)$
} **while** (stop condition)

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \log \sigma\left(\theta x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(\theta x^{(i)}\right)\right)\right)$$

$$\frac{\partial J(\theta)}{\theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(\sigma\left(\theta x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

Stochastic gradient descent

- Instead of evaluating gradient over all examples evaluate it for each **individual** training example

Initialize θ
Do {
  for each i
  $\theta \leftarrow \theta - \alpha \nabla J^{(i)}(\theta)$
} while (stop condition)

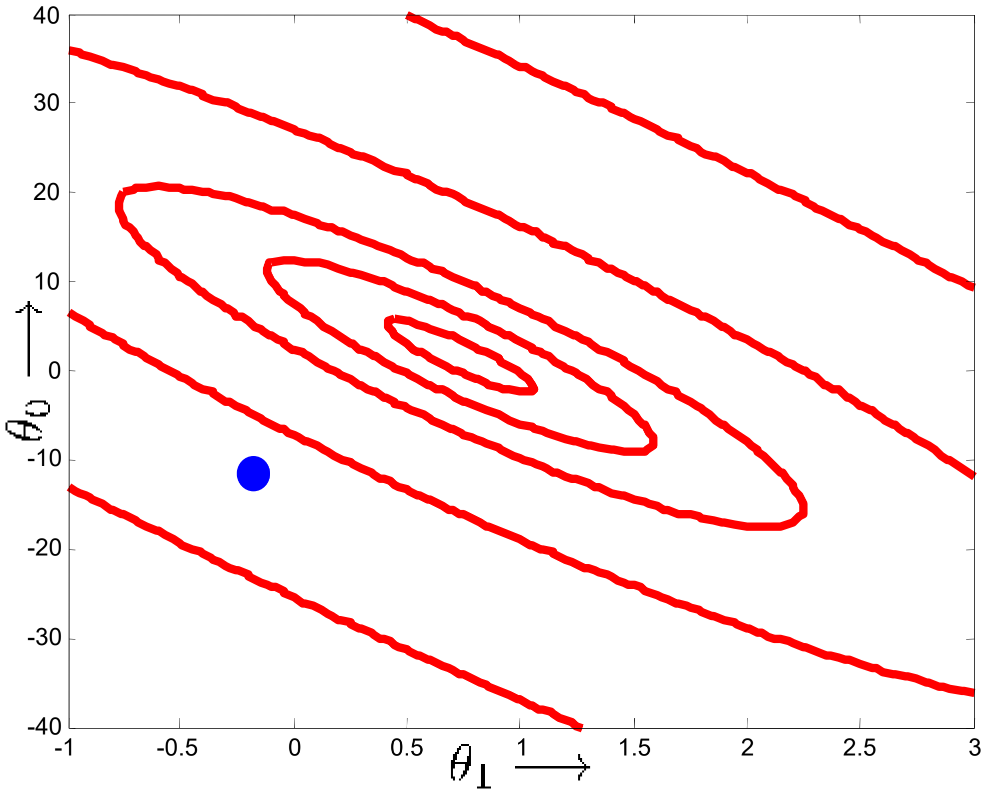
$$J^{(i)}(\theta) = y^{(i)} \log \sigma\left(\theta x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(\theta x^{(i)}\right)\right)$$

$$\frac{\partial J^{(i)}(\theta)}{\theta_j} = \left(\sigma\left(\theta x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

Stochastic gradient descent

- Update based on each datum at a time
  - Find residual and the gradient of its part of the error & update

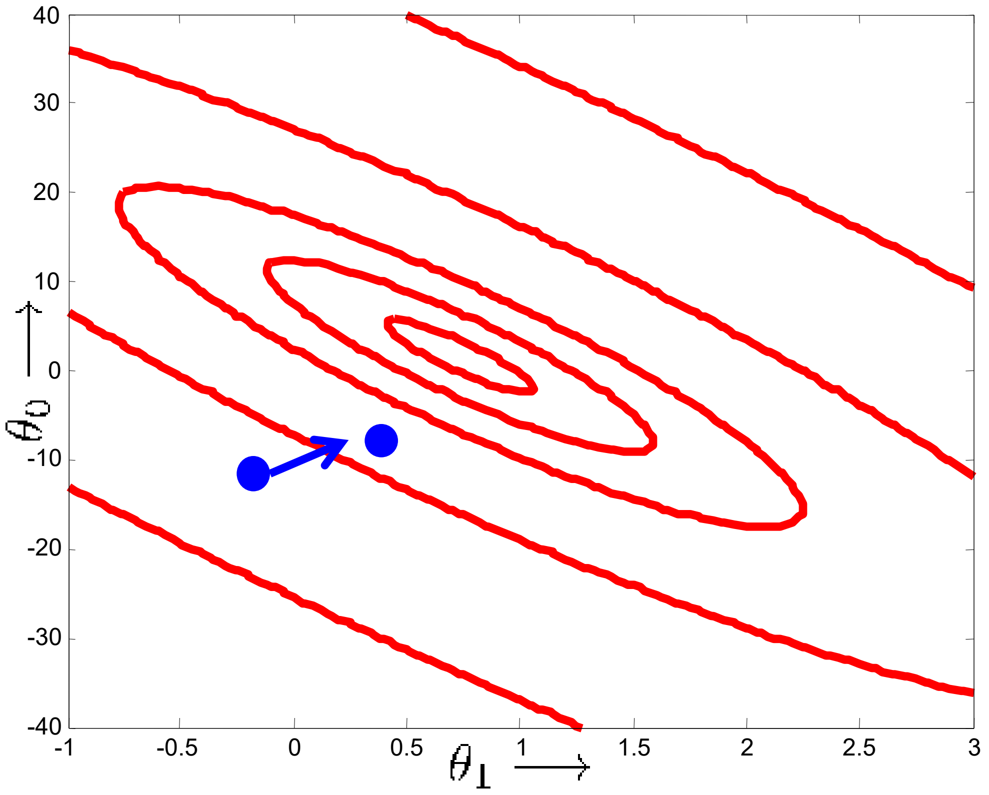
Initialize θ
Do {
  for i = 1 : m
  $\theta \leftarrow \theta - \alpha \nabla J^{(i)}(\theta)$
} while (stop condition)



Stochastic gradient descent

- Update based on each datum at a time
  - Find residual and the gradient of its part of the error & update

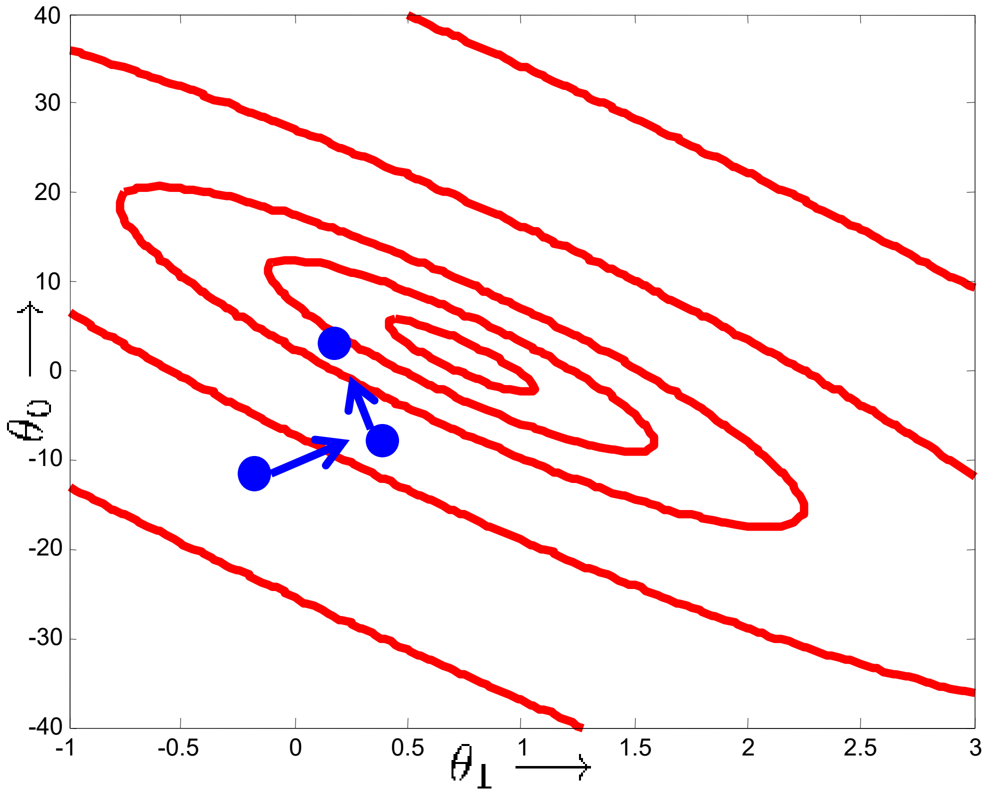
Initialize θ
Do {
  for i = 1 : m
  $\theta \leftarrow \theta - \alpha \nabla J^{(i)}(\theta)$
} while (stop condition)



Stochastic gradient descent

- Update based on each datum at a time
  - Find residual and the gradient of its part of the error & update

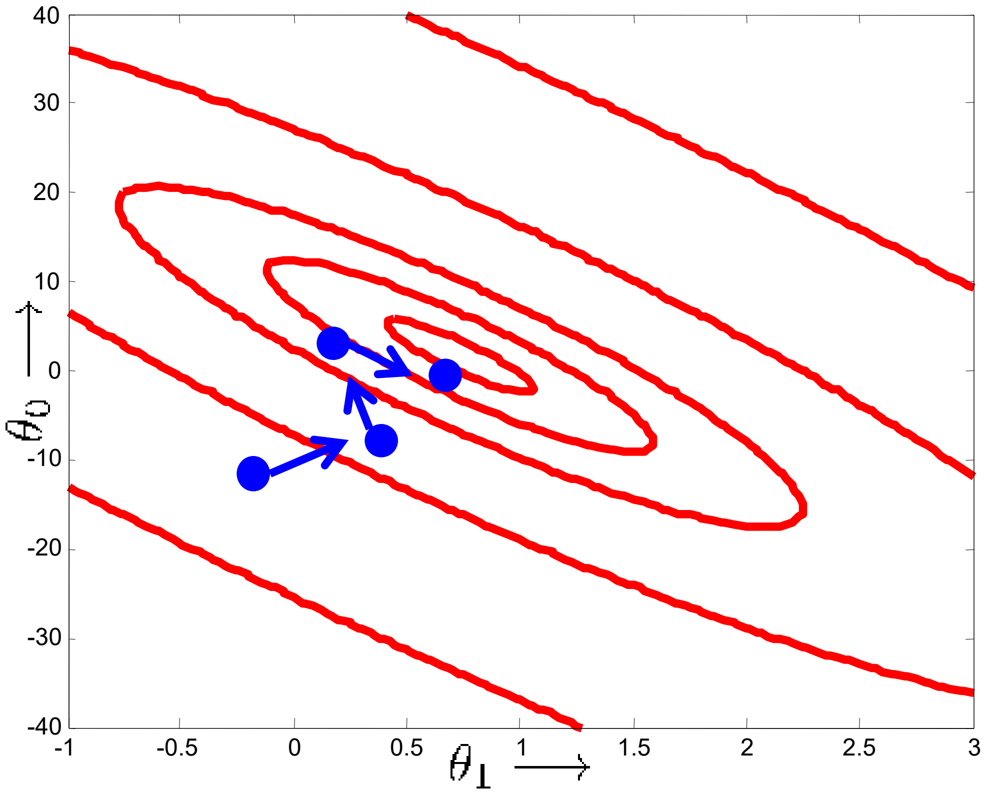
Initialize θ
Do {
  for i = 1 : m
  $\theta \leftarrow \theta - \alpha \nabla J^{(i)}(\theta)$
} while (stop condition)



Stochastic gradient descent

- Update based on each datum at a time
  - Find residual and the gradient of its part of the error & update

Initialize θ
Do {
  for i = 1 : m
  $\theta \leftarrow \theta - \alpha \nabla J^{(i)}(\theta)$
} while (stop condition)



Stochastic gradient descent

Initialize θ
Do {
  for i = 1 : m
  $\theta \leftarrow \theta - \alpha \nabla J^{(i)}(\theta)$
} while (stop condition)

- Benefits
  - Lots of data = many more updates per pass
  - Computationally faster
- Drawbacks
  - No longer strictly "descent"
  - Stopping conditions may be harder to evaluate
    (Can use "running estimates" of J(.), etc.)
- Related: mini-batch updates, etc.

Summary

- Linear classifier ⇔ perceptron

- Measuring quality of a decision boundary
  - Error rate (0/1 loss)
  - Surrogate functions

- Learning the weights of a linear classifier from data
  - Reduces to an optimization problem
  - Perceptron algorithm
  - Using surrogate functions, we can do gradient descent
  - Gradient equations & update rules (BGD and SGD)
  - Multiclass logistic regression (softmax function)