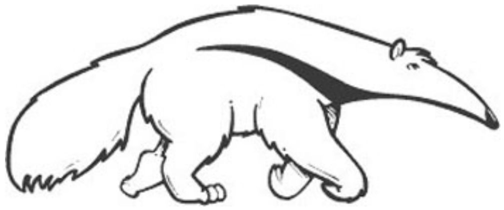
+

Machine Learning and Data Mining

Linear classification

Prof. Alexander Ihler

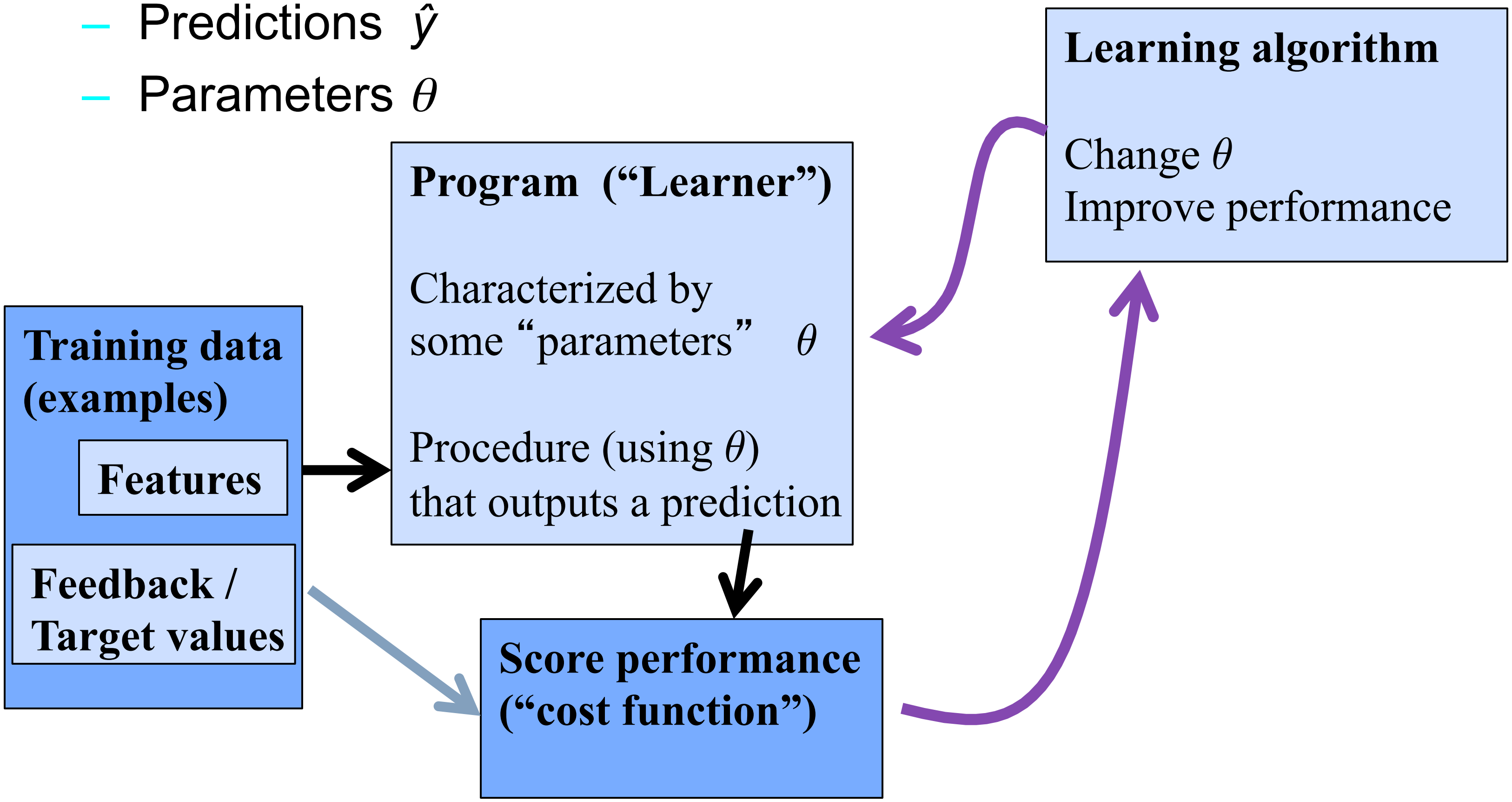


BREN:ICS
INFORMATION AND COMPUTER SCIENCES

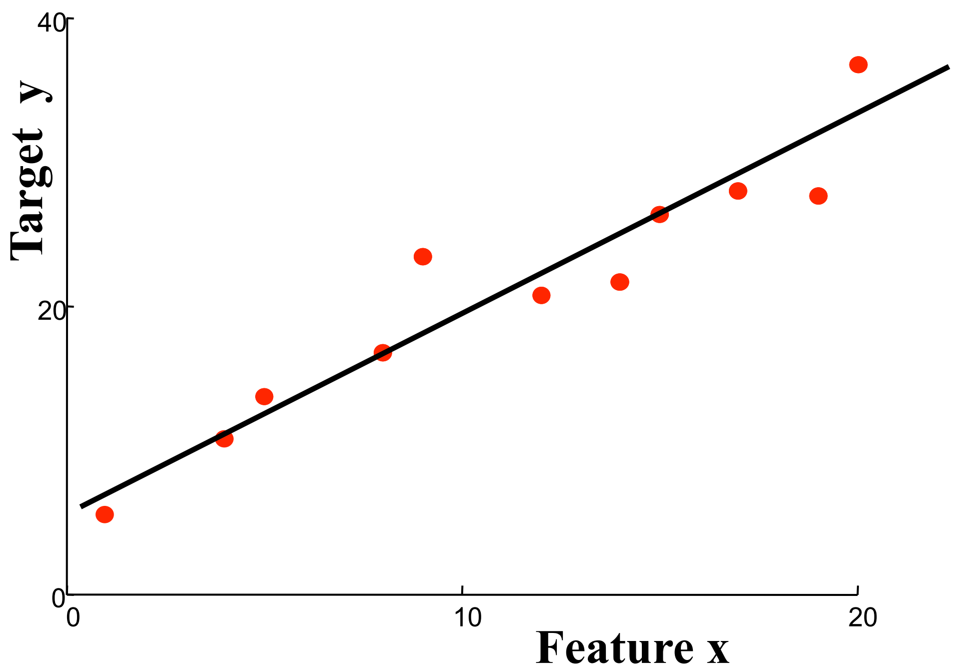
UNIVERSITY of CALIFORNIA IRVINE

Supervised learning

- Notation
  - Features x
  - Targets y
  - Predictions $\hat{y}$
  - Parameters θ



Linear regression



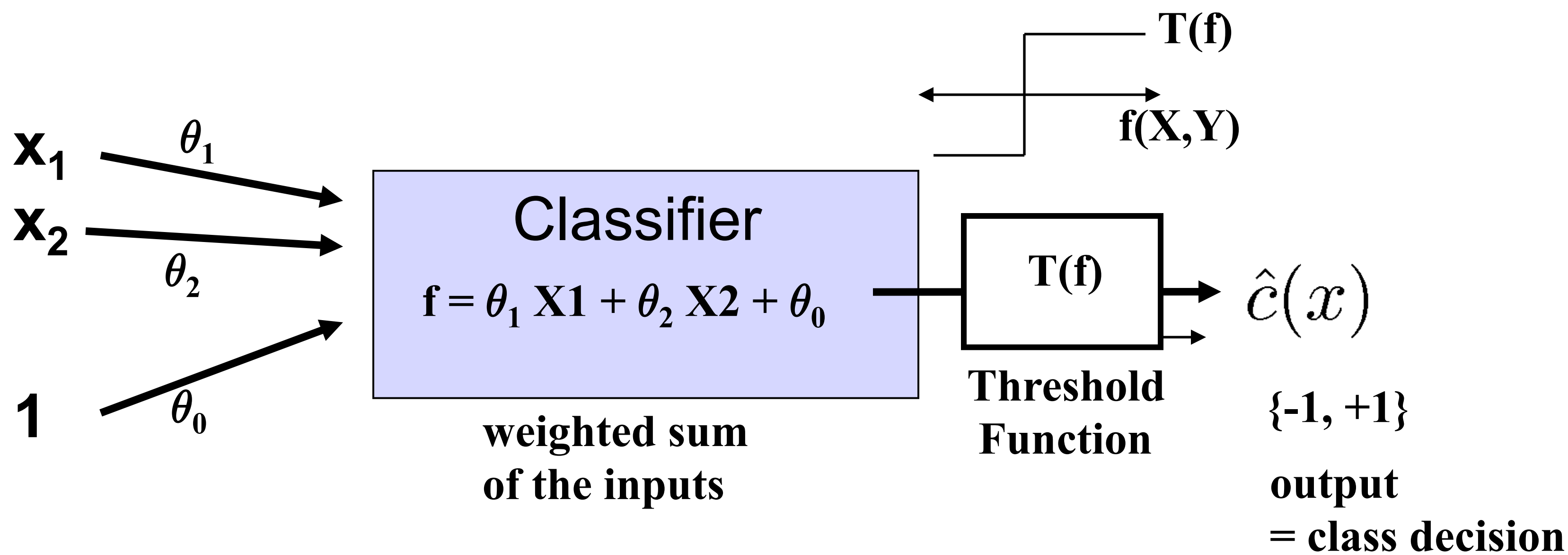
"Predictor":
Evaluate line:

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

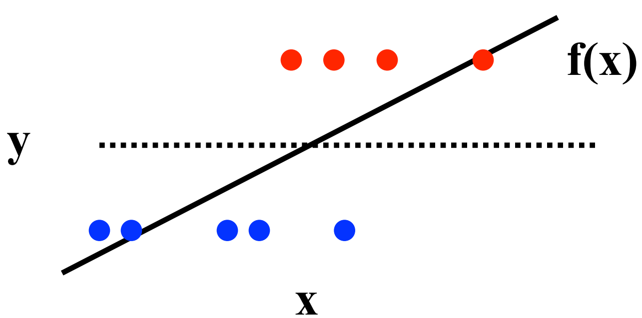
return r

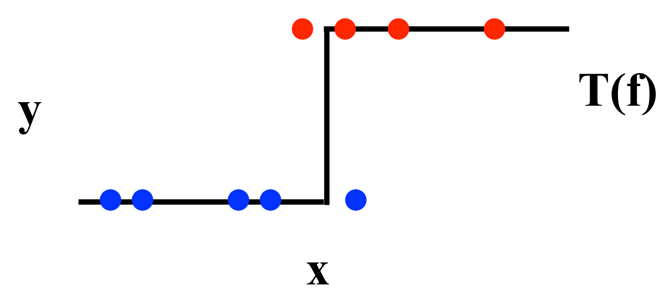
- Contrast with classification
  - Classify: predict discrete-valued target y

Perceptron Classifier (2 features)



Visualizing for one feature "x":





Perceptrons

- Perceptron = a linear classifier
  - The parameters θ are sometimes called weights ("w")
    - real-valued constants (can be positive or negative)
  - Define an additional constant input "1"

- 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"

Notation

- Inputs:
  - $x_0, x_1, x_2, \ldots\ldots\ldots\ldots, x_d,$
  - $x_1, x_2, \ldots\ldots\ldots\ldots, x_{d-1}, x_d$ are the values of the d features
  - $x_0 = 1$ (a constant input)
  - $\underline{\mathbf{x}} = (x_0, x_1, x_2, \ldots\ldots\ldots\ldots, x_d)$: feature vector (row vector)
- Weights (parameters):
  - $\theta_0, \theta_1, \theta_2, \ldots\ldots\ldots\ldots, \theta_d,$
  - we have d+1 weights
  - one for each feature + one for the constant
  - $\underline{\theta} = (\theta_0, \theta_1, \theta_2, \ldots\ldots\ldots\ldots, \theta_d)$: parameter vector (row vector)
- Linear response
  - $\theta_0 x_0 + \theta_1 x_1 + \ldots \theta_d x_d = \underline{\theta} \cdot \underline{x}'$ then threshold

(Matlab)

```
>> f = th*x'; f = sum(th.*x); yhat = sign(f);
```

Perceptron Decision Boundary

- The perceptron is defined by the decision algorithm:

$$o(x_1, x_2, \ldots, x_d, x_{d+1}) \begin{cases} = \quad 1 & (\text{if } \underline{\theta} \,.\, \underline{\mathbf{x}}' > 0) \\ = \quad -1 & (\text{otherwise}) \end{cases}$$

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

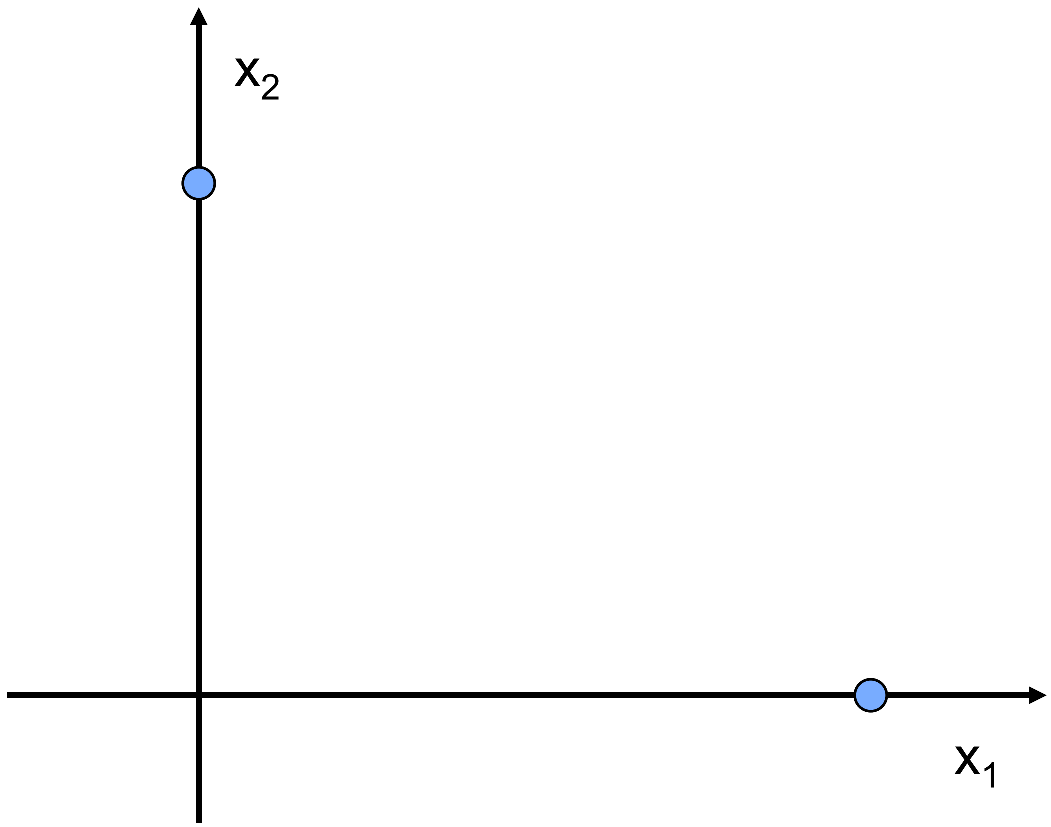
- The equation of the hyperplane is given by

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

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

Example, Linear Decision Boundary

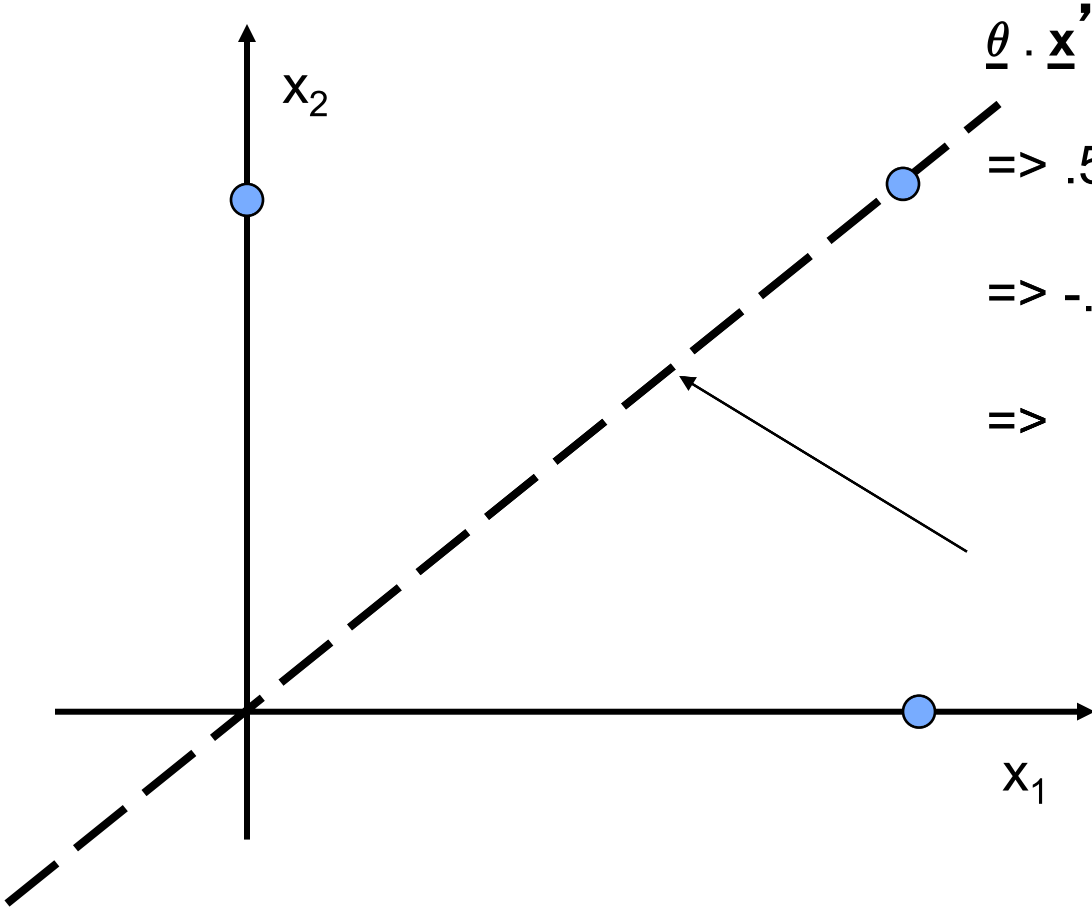
$$\underline{\theta} = (\theta_1, \theta_2, \theta_0) = (.5, -.5, 0)$$



From P. Smyth

Example, Linear Decision Boundary

$$\underline{\theta} = (\theta_1, \ \theta_2, \ \theta_0)$$
$$= (.5, \ -.5, \ 0)$$



$\underline{\theta} \cdot \underline{\mathbf{x}}' = 0$

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

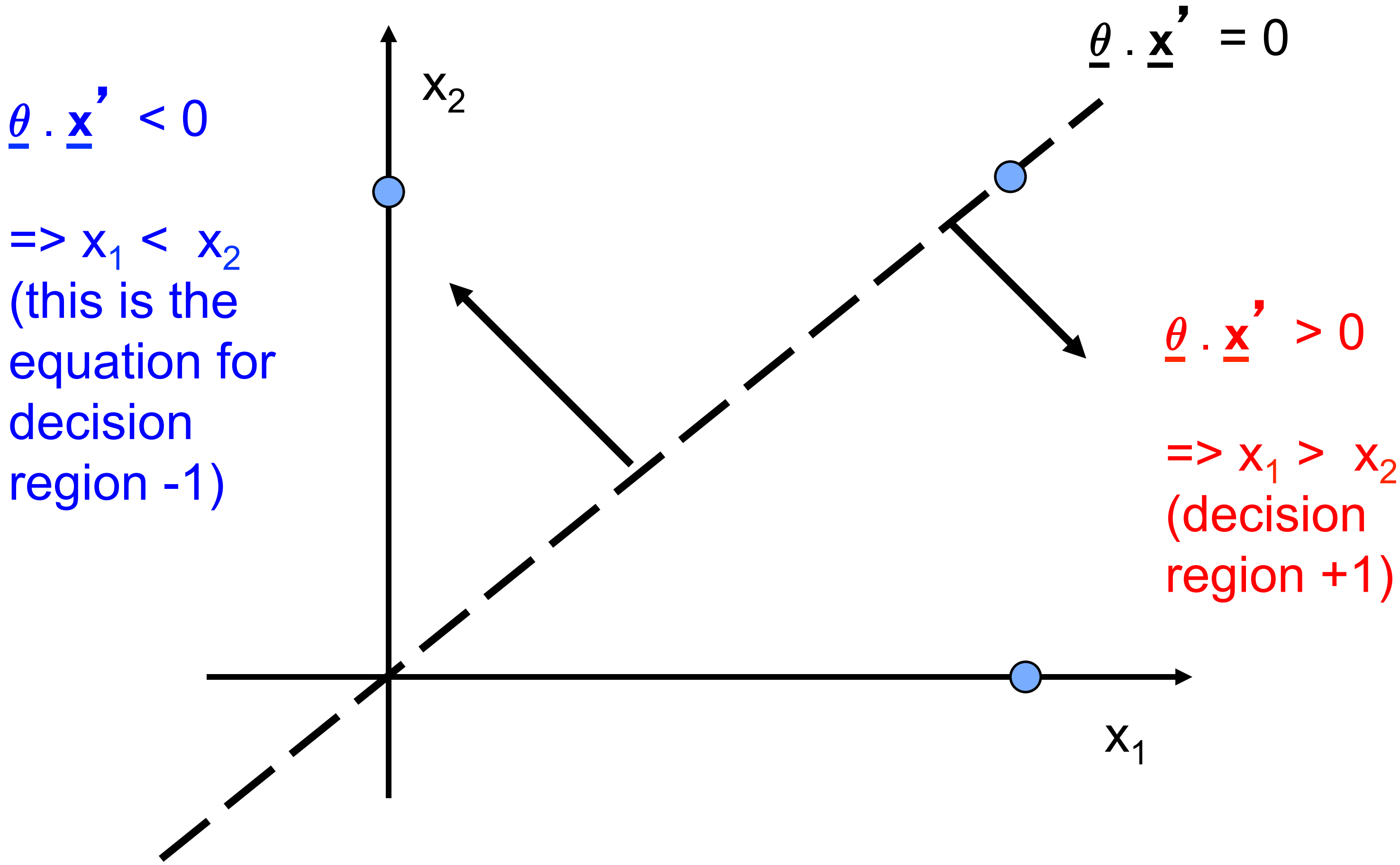
$\Rightarrow -.5\ x_2 = -.5\ x_1 + 0$

$\Rightarrow \quad x_2 = x_1$

From P. Smyth

Example, Linear Decision Boundary

$$\underline{\theta} = (\theta_1, \ \theta_2, \ \theta_0)$$
$$= (.5, \ -.5, \ 0)$$

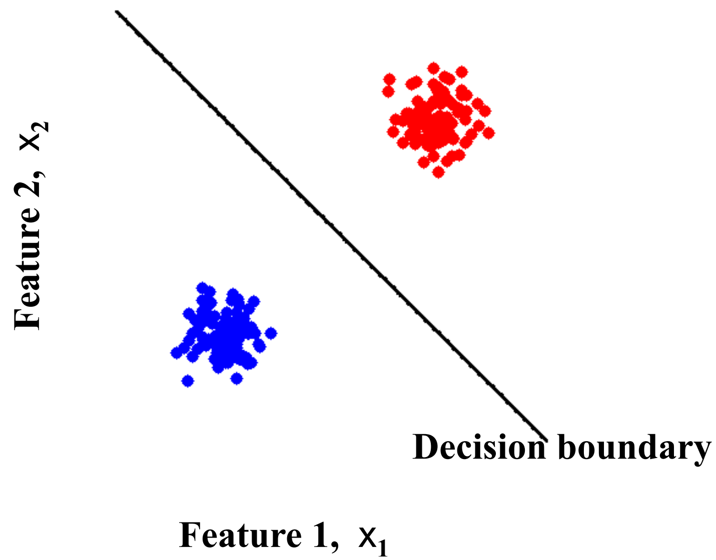


From P. Smyth

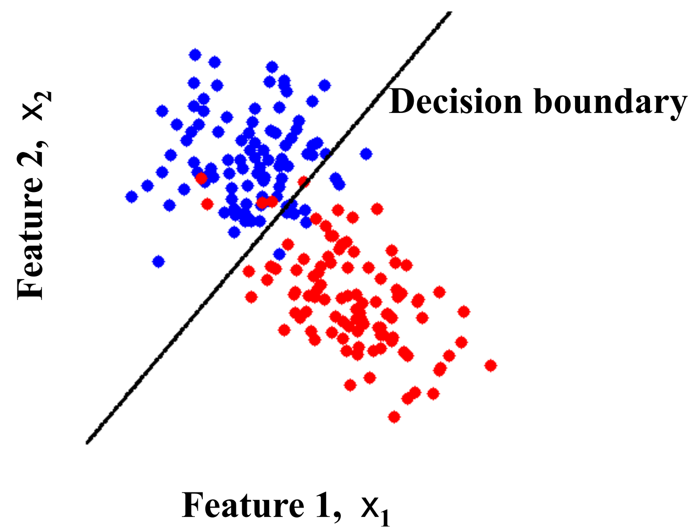
Separability

- 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

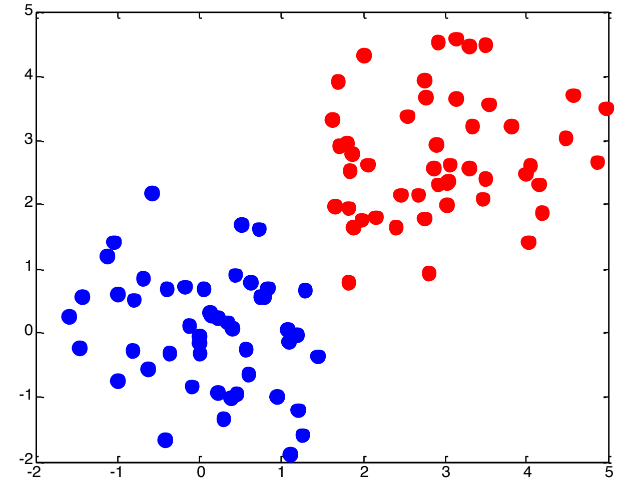
Linearly separable data



Linearly non-separable data

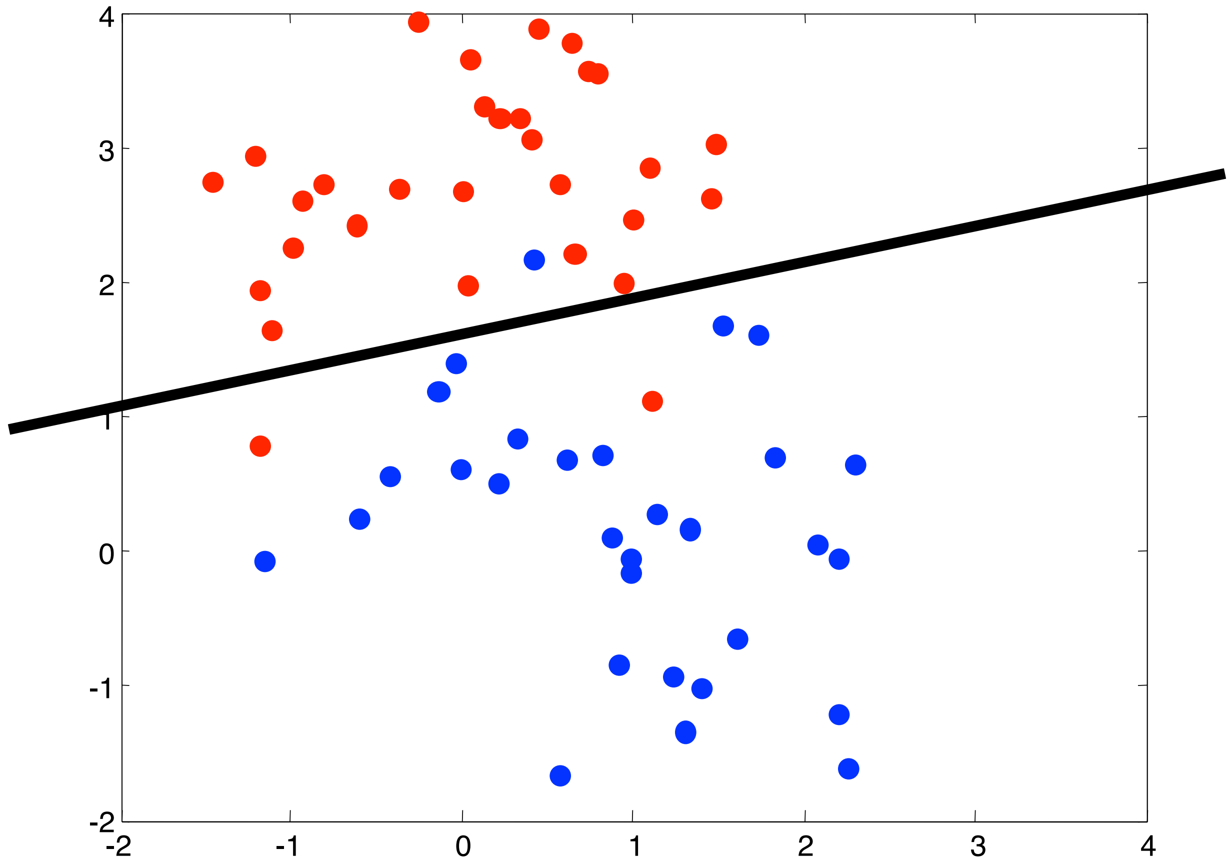


Class overlap

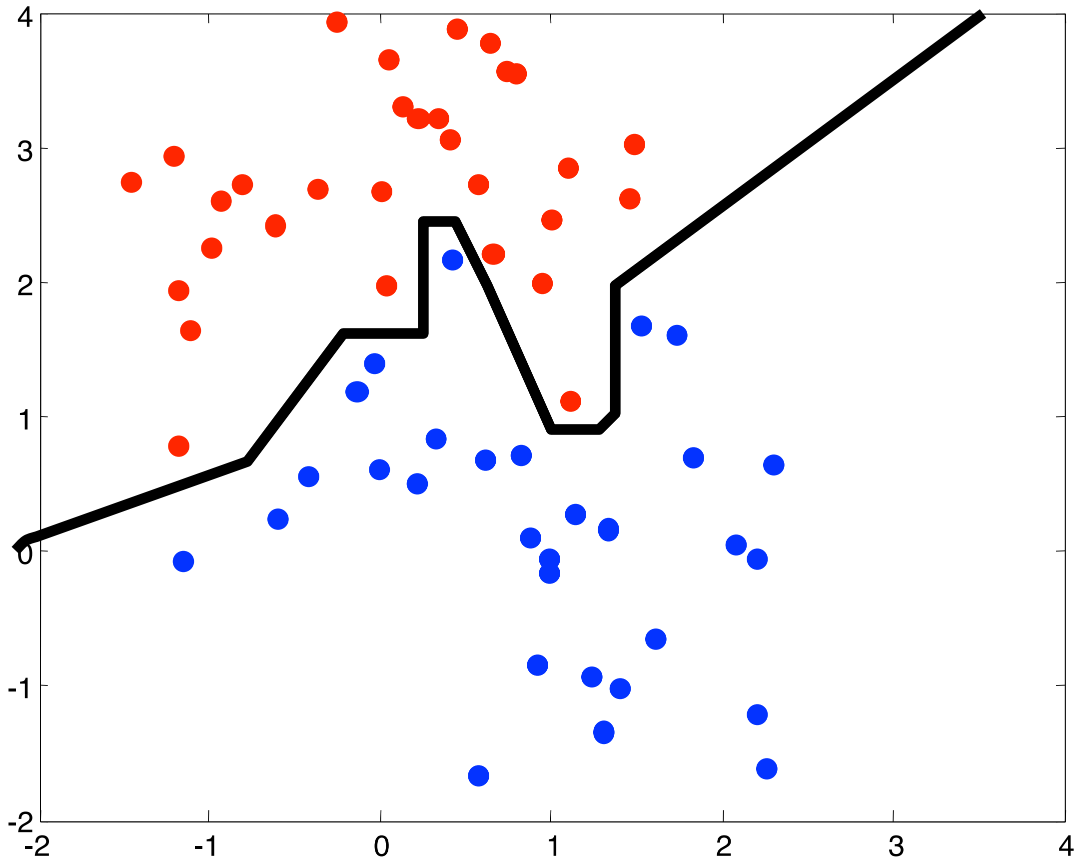


- Classes may not be well-separated
- Same observation values possible under both classes
  - High vs low risk; features {age, income}
  - Benign/malignant cells look similar
  - …
- Common in practice
- May not be able to perfectly distinguish between classes
  - Maybe with more features?
  - Maybe with more complex classifier?
- Otherwise, may have to accept some errors

Another example

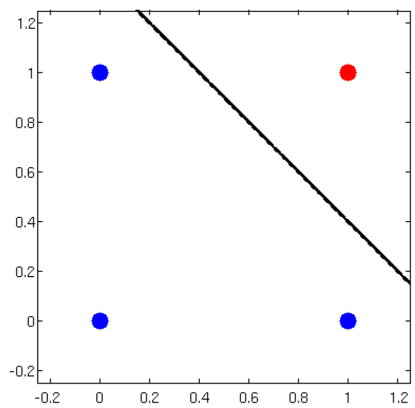


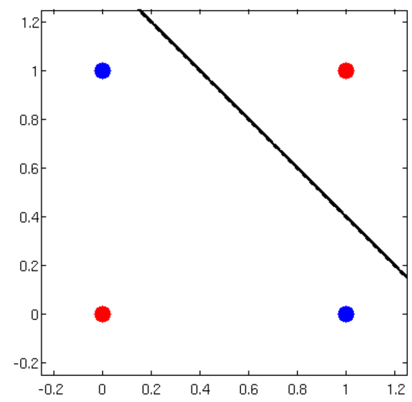
Non-linear decision boundary



Representational Power of Perceptrons

- What mappings can a perceptron represent perfectly?
  - A perceptron is a linear classifier
  - thus it can represent any mapping that is linearly separable
  - some Boolean functions like AND (on left)
  - but not Boolean functions like XOR (on right)





Adding features

- Linear classifier can't learn some functions

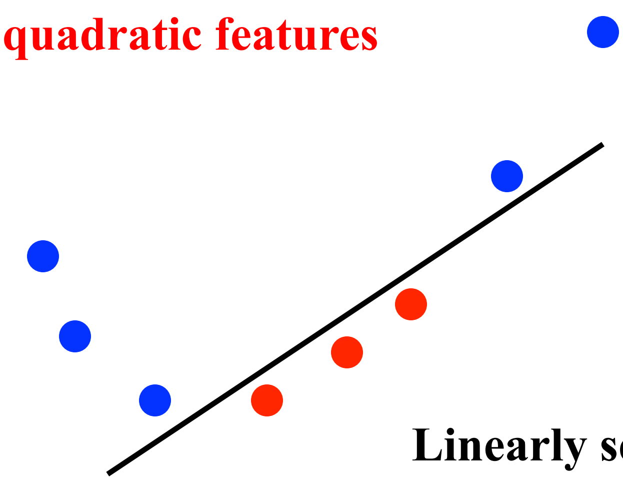
1D example:

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

Not linearly separable

Add quadratic features

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



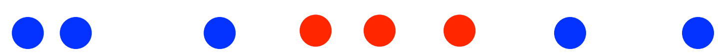
Linearly separable in new features…

Adding features

- Linear classifier can't learn some functions

1D example:

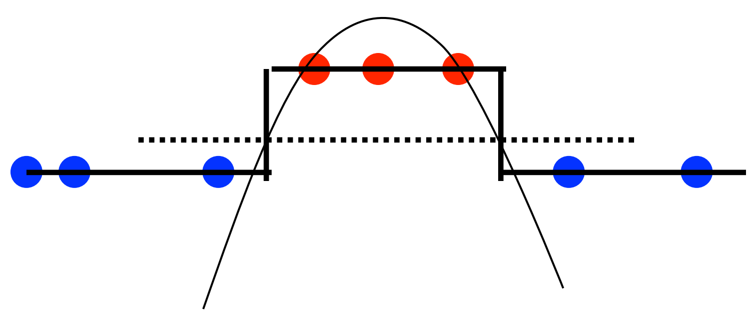
$$y = T(\, b\, x + c\,)$$



Not linearly separable

Quadratic features, visualized in original feature space:

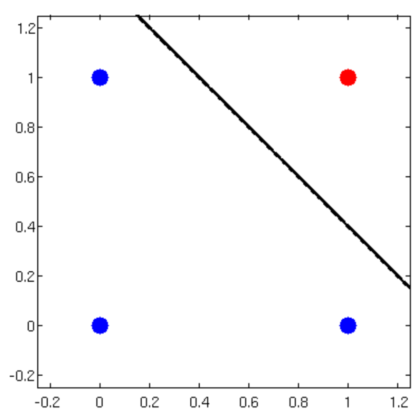
$$y = T(\, a\, x^2 + b\, x + c\,)$$

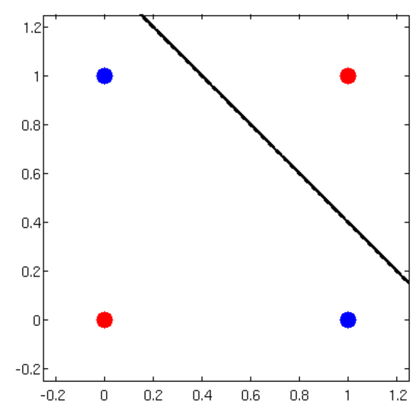


More complex decision boundary: $ax^2+bx+c = 0$

Representational Power of Perceptrons

- What mappings can a perceptron represent perfectly?
  - A perceptron is a linear classifier
  - thus it can represent any mapping that is linearly separable
  - some Boolean functions like AND (on left)
  - but not Boolean functions like XOR (on right)

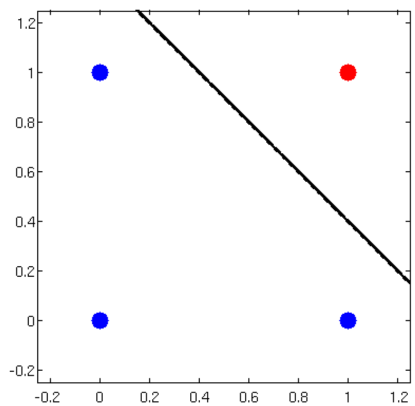


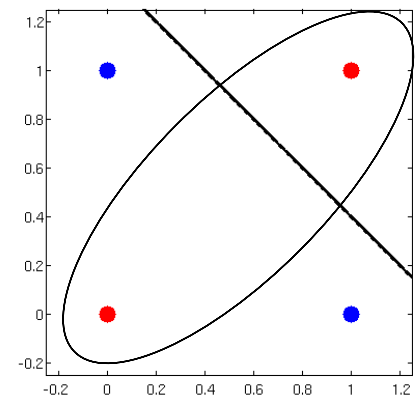


What kinds of functions would we need to learn the data on the right?

Representational Power of Perceptrons

- What mappings can a perceptron represent perfectly?
  - A perceptron is a linear classifier
  - thus it can represent any mapping that is linearly separable
  - some Boolean functions like AND (on left)
  - but not Boolean functions like XOR (on right)





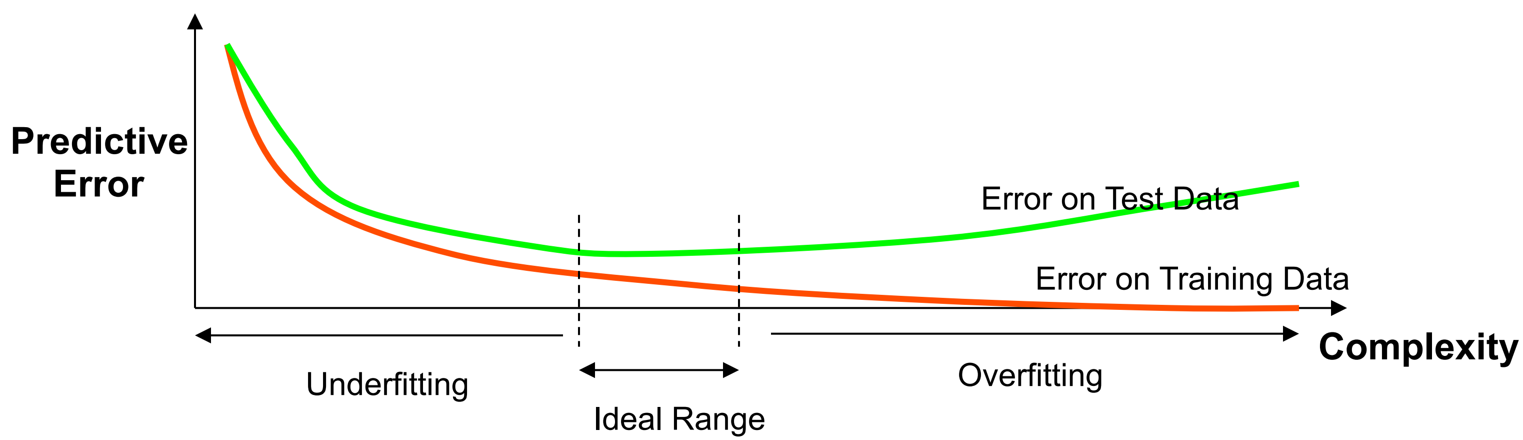
What kinds of functions would we need to learn the data on the right?
Ellipsiodal decision boundary: $a\,(x_1 - b)^2 + c\,(x_2 - d)^2 = 0$

Effect of dimensionality

- Data are increasingly separable in high dimension – is this a good thing?

- “Good”
  - Separation is easier in higher dimensions (for fixed N)
  - 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…

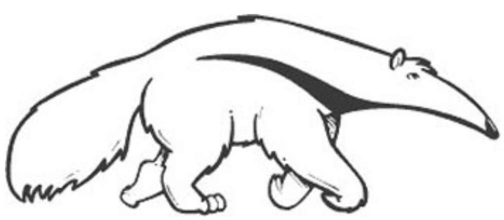


+

Machine Learning and Data Mining

Linear classification: Learning

Prof. Alexander Ihler



BREN:ICS
INFORMATION AND COMPUTER SCIENCES
UNIVERSITY of CALIFORNIA IRVINE

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 an objective function $J(\underline{\theta})$:
  - Classifier accuracy (for a given set of weights $\underline{\theta}$ and labeled data)

- Maximize this objective function (or, minimize error)
  - An optimization or search problem over the vector $(\theta_1, \theta_2, \theta_0)$

Learning the Weights from Data

An Example of a Training Data Set

Example	x_1	x_2	….	x_d	true class label, y
$\underline{x}(1)$	3.4	-1.2	…..	7.1	1
$\underline{x}(2)$	4.1	-3.1	…..	4.6	-1
$\underline{x}(3)$	5.7	-1.0	…..	6.2	-1
$\underline{x}(4)$	2.2	4.1	…..	5.0	1
$\underline{x}(n)$	1.2	4.3	…..	6.1	1

Training a linear classifier

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

$$\mathrm{err}(\underline{\theta}) \;=\; 1/N \sum \;\delta\Big(\,\hat{y}(i) \neq y(i)\,\Big)$$

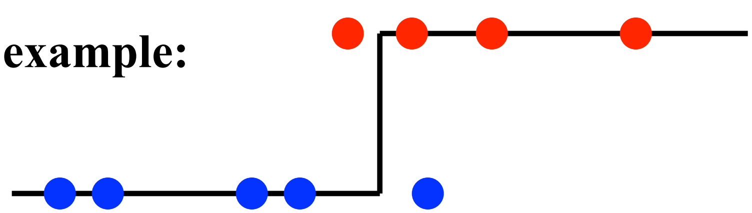
where $\delta\Big(\,\hat{y}(i) \neq y(i)\,\Big) = 0$ if $\hat{y}(i) = y(i)$, and 1 otherwise

(Matlab)

```
>> yh = sign(th*X'); err = mean(y ~= yh);
```

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

1D example:

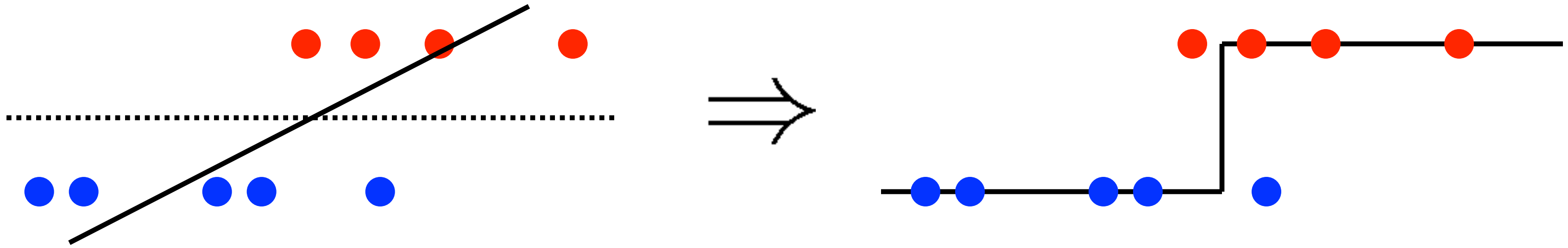


T(f) = -1 if f < 0

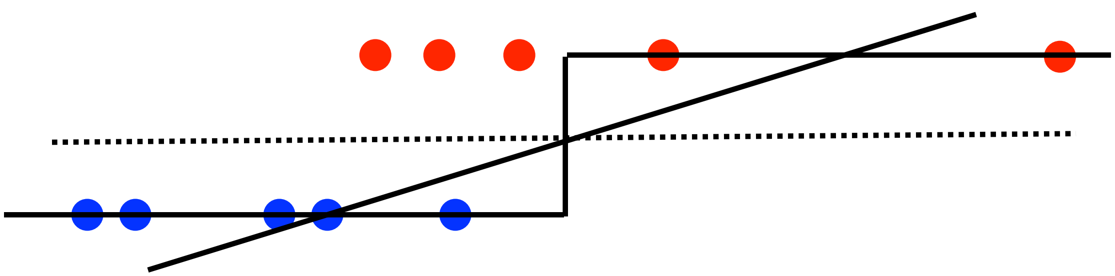
T(f) = +1 if f > 0

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



Perceptron algorithm

- Perceptron algorithm: an SGD-like algorithm

  While (~done)

  For each data point j:

  $\hat{y}(j) = T(\underline{\theta} * \underline{x}(j))$: predict output for data point j

  $\underline{\theta} \leftarrow \underline{\theta} + \alpha\, (y(j) - \hat{y}(j))\, \underline{x}(j)$: "gradient-like" step

- Compare to linear regression + MSE cost
  - Identical update to SGD for MSE except error uses thresholded $\hat{y}(j)$ instead of linear response $\underline{\theta}$ x' so:
  - (1) For correct predictions, $y(j) - \hat{y}(j) = 0$
  - (2) For incorrect predictions, $y(j) - \hat{y}(j) = \pm 2$

"adaptive" linear regression: correct predictions stop contributing

Perceptron algorithm

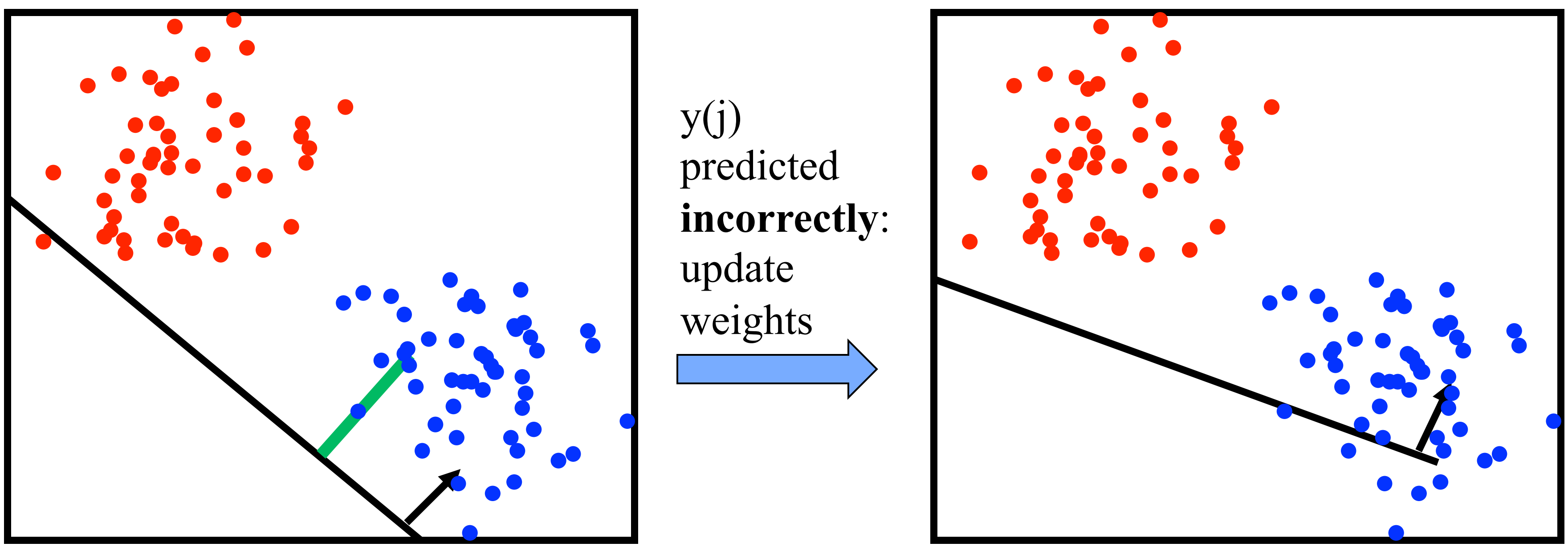
- Perceptron algorithm: an SGD-like algorithm

  While (~done)

  For each data point j:

  $\hat{y}(j) = T(\, \underline{\theta} * \underline{x}(j)\,)$: predict output for data point j

  $\underline{\theta} \leftarrow \underline{\theta} + \alpha\, (\, y(j) - \hat{y}(j)\,)\, \underline{x}(j)$: "gradient-like" step



Perceptron algorithm

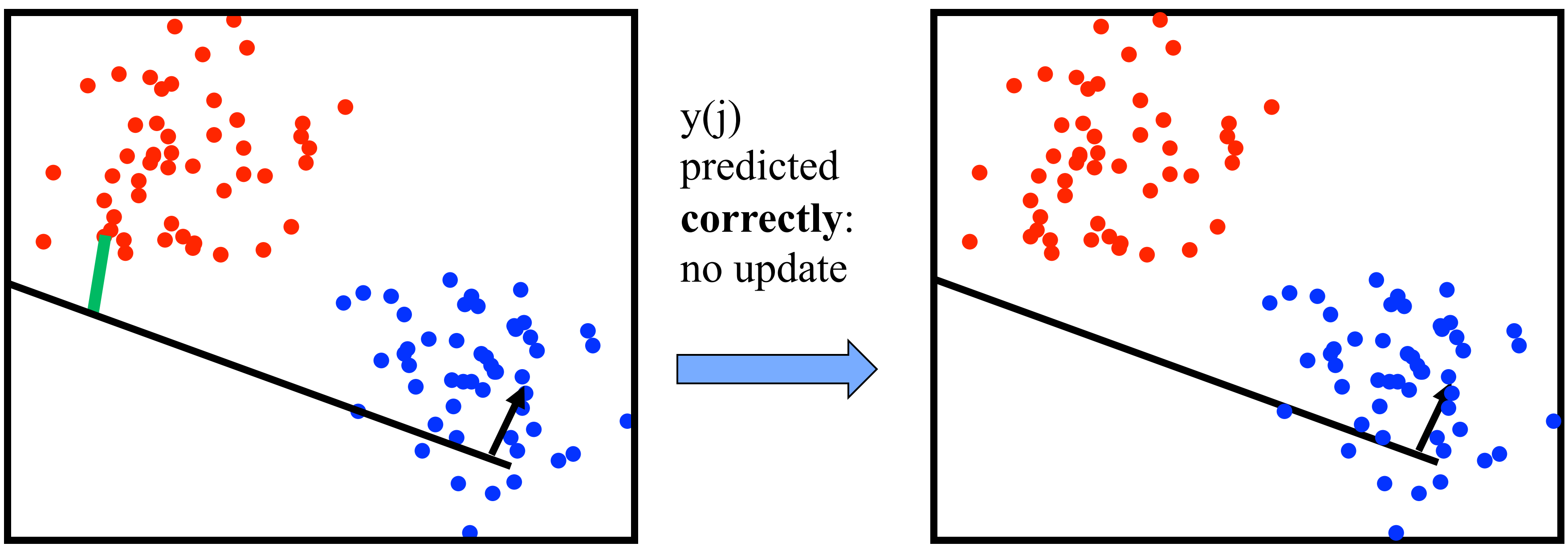
- Perceptron algorithm: an SGD-like algorithm

  While (~done)

  For each data point j:

  $\hat{y}(j) = T(\, \underline{\theta} * \underline{x}(j) \,)$: predict output for data point j

  $\underline{\theta} \leftarrow \underline{\theta} + \alpha\, (\, y(j) - \hat{y}(j)\,)\, \underline{x}(j)$: "gradient-like" step



Perceptron algorithm

- Perceptron algorithm: an SGD-like algorithm

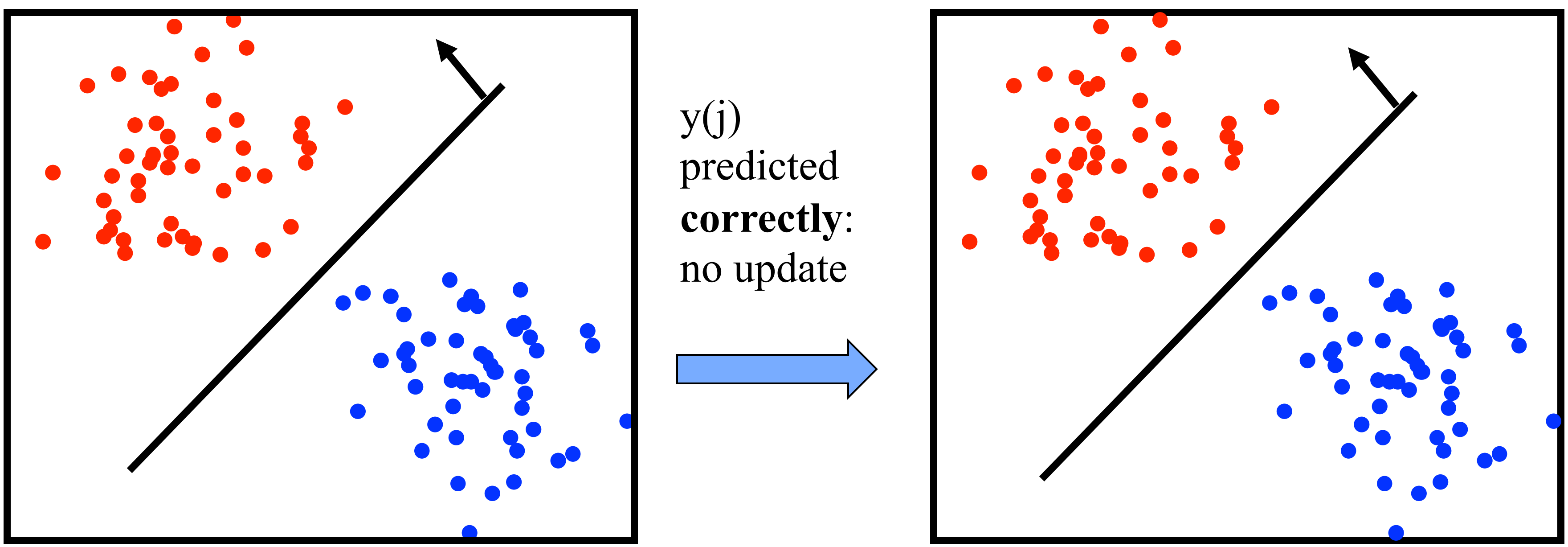
  While (~done)

  For each data point j:

  $\hat{y}(j) = T(\underline{\theta} * \underline{x}(j))$: predict output for data point j

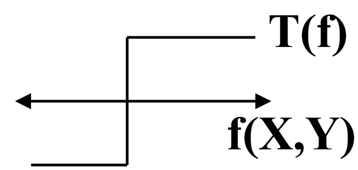
  $\underline{\theta} \leftarrow \underline{\theta} + \alpha\,(y(j) - \hat{y}(j))\,\underline{x}(j)$: "gradient-like" step

  (Converges if data are linearly separable)

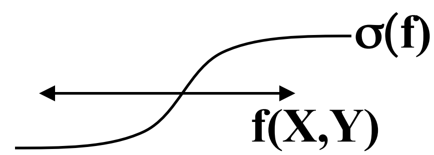


Surrogate loss functions

- Another solution: use a "smooth" loss
  - e.g., approximate the threshold function



  - Usually some smooth function of distance
    - Example: "sigmoid", looks like an "S"



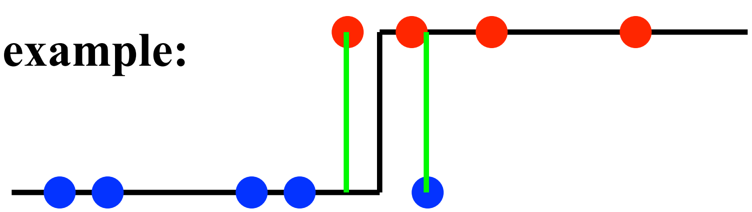
  - Now, measure e.g. MSE

$$J(\underline{\theta}) = \frac{1}{m} \sum_j \Big(\ \sigma(\ f(x^{(i)}) \ ) - y^{(i)} \ \Big)^2$$

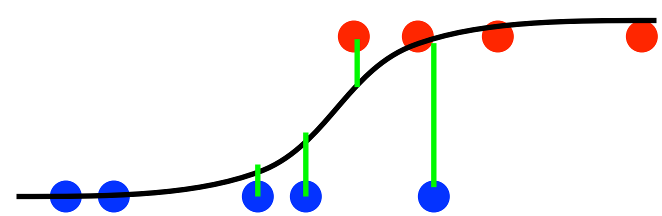
Class y = {0, 1} …

  - Far from the decision boundary: |f(.)| large, small error
  - Nearby the boundary: |f(.)| near 1/2, larger error

1D example:



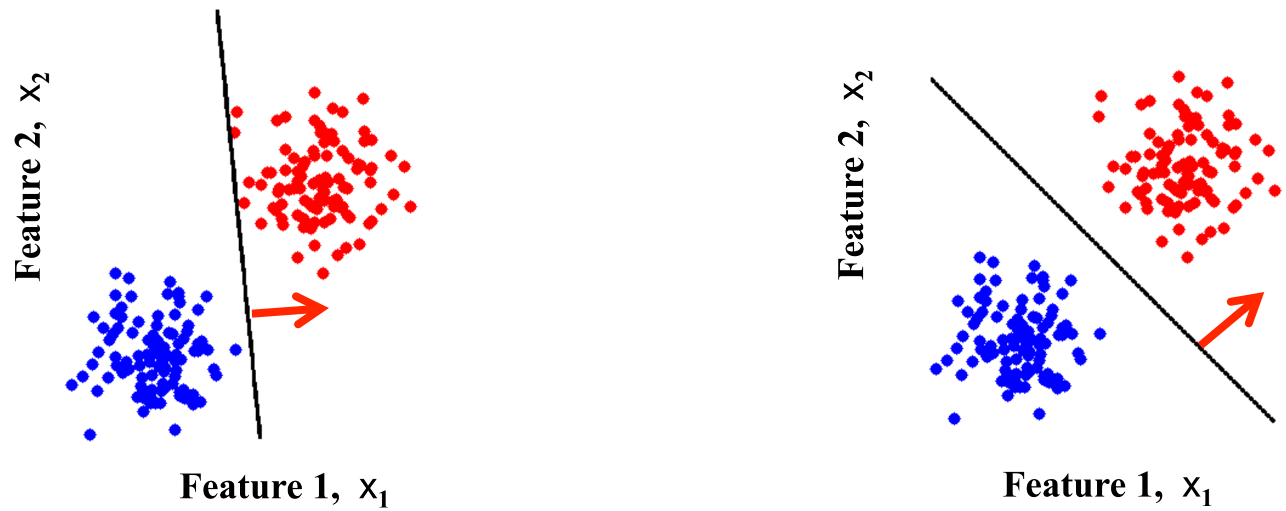
Classification error = MSE = 2/9



MSE = $(0^2 + 1^2 + .2^2 + .25^2 + .05^2 + \ldots)/9$

Beyond misclassification rate

- Which decision boundary is "better"?
  - Both have zero training error (perfect training accuracy)
  - But, one of them seems intuitively better…

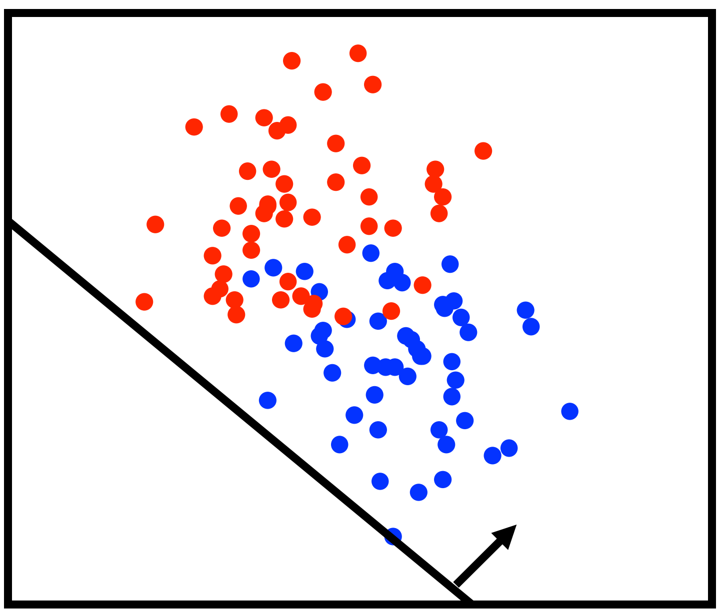


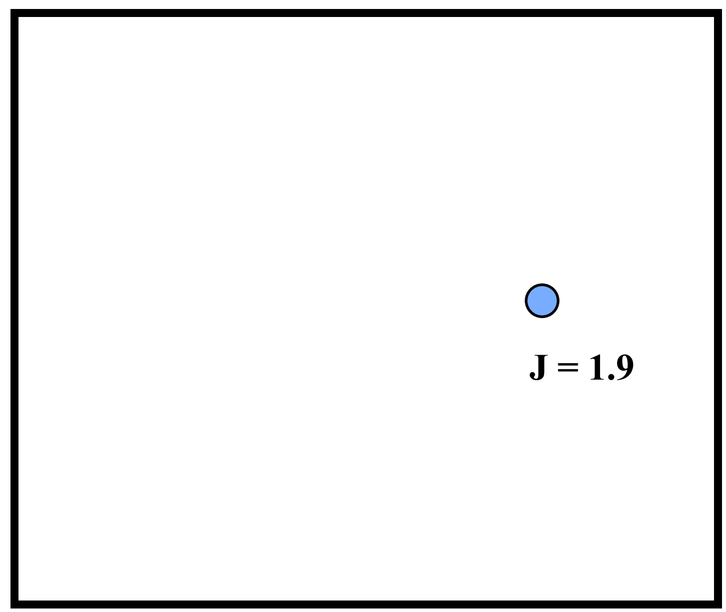
- Side benefit of "smoothed" error function
  - Encourages data to be far from the decision boundary
  - See more examples of this principle later...

Training the Classifier

- Once we have a smooth measure of quality, we can find the “best” settings for the parameters of f(X1,X2) = a*X1 + b*X2 + c

- Example: 2D feature space ⇔ parameter space

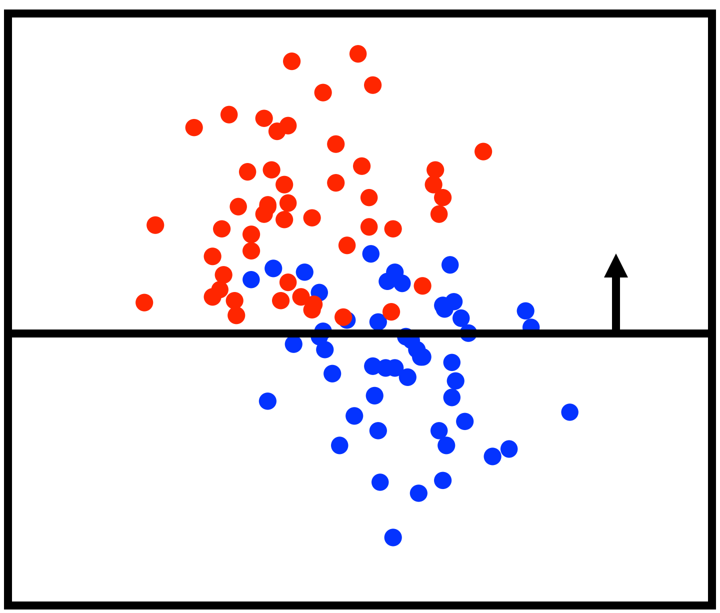


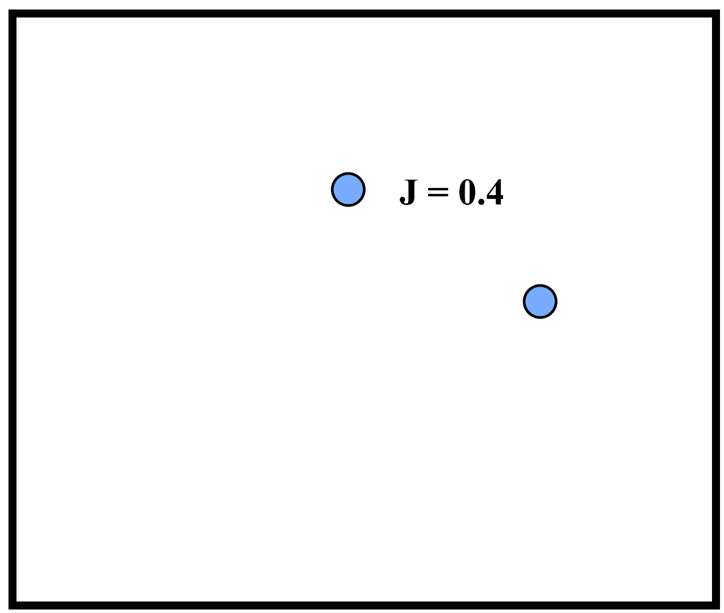


Training the Classifier

- Once we have a smooth measure of quality, we can find the “best” settings for the parameters of
  f(X1,X2) = a*X1 + b*X2 + c

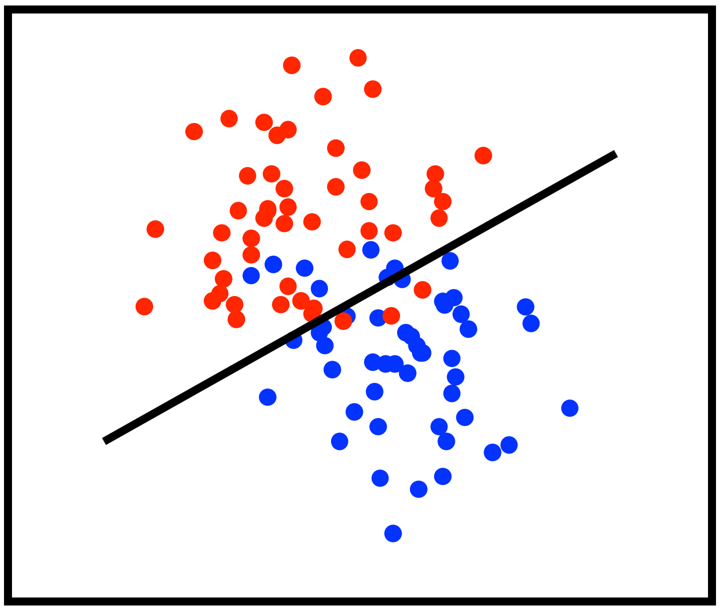
- Example: 2D feature space ⇔ parameter space

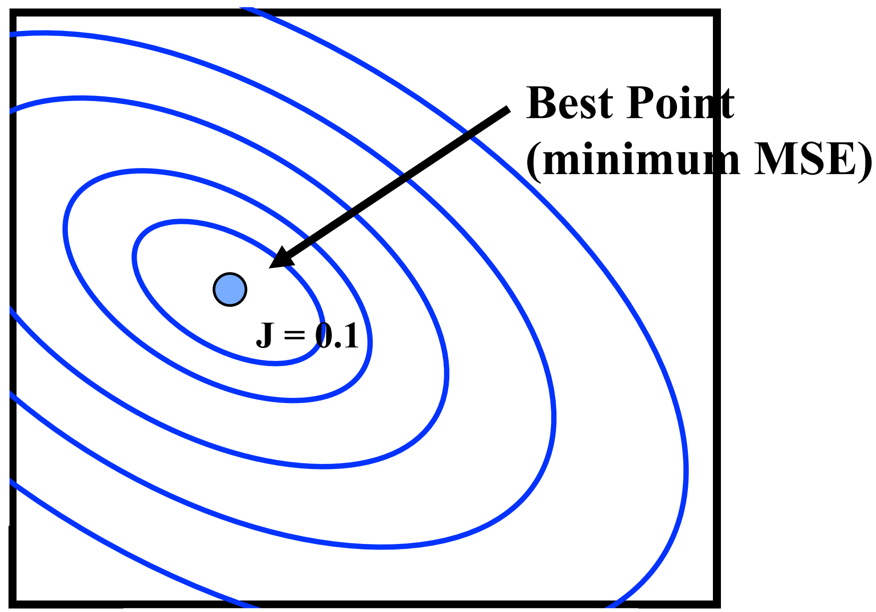




Training the Classifier

- Once we have a smooth measure of quality, we can find the "best" settings for the parameters of
  f(X1,X2) = a*X1 + b*X2 + c
- Finding the minimum loss J(.) in parameter space…





- [a b c] = ?

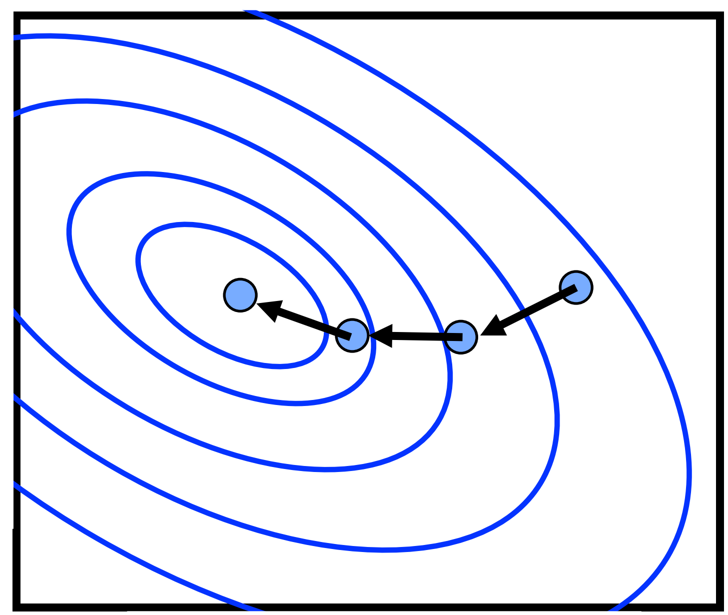
[arctan(a/b), c] = [-pi/4, 1]

Finding the Best MSE

- As in linear regression, this is now just optimization

- Methods:
  - Gradient descent
    - Improve loss by small changes in parameters (“small” = learning rate)
  - Or, substitute your favorite optimization algorithm…
    - Coordinate descent
    - Stochastic search
    - Genetic algorithms

Gradient Descent



Gradient Equations

- MSE (note, depends on function $\sigma(.)$)

$$J(\underline{\theta} = [a, b, c]) = \frac{1}{N} \sum_i (\sigma(a x_1^{(i)} + b x_2^{(i)} + c) - y^{(i)})^2$$

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

$$\frac{\partial J}{\partial a} = \frac{1}{N} \sum_i 2 \boxed{(\sigma(\theta \cdot x^{(i)}) - y^{(i)})} \boxed{\partial\sigma(\theta \cdot x^{(i)}) \; x_1^{(i)}}$$

Error between class and prediction

Sensitivity of prediction to changes in parameter "a"

- Similar for parameters b, c [replace x_1 with x_2 or 1 (constant)]

Saturating Functions

- Many possible "saturating" functions
- "Logistic" sigmoid (scaled for range [0,1]) is

$$\sigma(z) = 1 / (1 + \exp(-z))$$

- Derivative is

(to predict: threshold at ½)

$$\partial\sigma(z) = \sigma(z)\,(1-\sigma(z))$$

- Matlab Implementation:

```
function s = sig(z)
% value of [0,1] sigmoid
   s = 1 ./ (1+exp(-z));

function ds = dsig(x)
% derivative of (scaled) sigmoid
   ds = sig(z) .* (1-sig(z));
```

For range [-1 , +1]:

$$\rho(z) = 2\,\sigma(z) - 1$$

$$\partial\rho(z) = 2\,\sigma(z)\,(1-\sigma(z))$$

To predict: threshold at zero

Logistic regression

- Intepret $\sigma(\underline{\theta}\ x')$ as a probability that y = 1
- Use a negative log-likelihood loss function
  – If y = 1, cost is - log Pr[y=1] = - log $\sigma(\underline{\theta}\ x')$
  – If y = 0, cost is - log Pr[y=0] = - log (1 - $\sigma(\underline{\theta}\ x')$)

- Can write this succinctly:

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

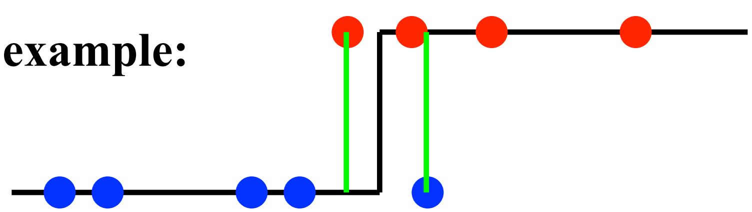
Logistic regression

- Intepret $\sigma(\underline{\theta}\ x')$ as a probability that y = 1
- Use a negative log-likelihood loss function
  - If y = 1, cost is - log Pr[y=1] = - log $\sigma(\underline{\theta}\ x')$
  - If y = 0, cost is - log Pr[y=0] = - log (1 - $\sigma(\underline{\theta}\ x')$)
- Can write this succinctly:

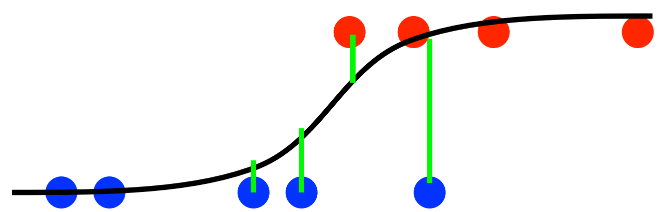
$$J(\underline{\theta}) = -\frac{1}{m} \sum_i y^{(i)} \log \sigma(\theta \cdot x^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(\theta \cdot x^{(i)}))$$

- Convex! Otherwise similar: optimize J(θ) via …

1D example:



Classification error = MSE = 2/9



NLL = - (log(.99) + log(.97) + …)/9

Gradient Equations

- Logistic neg-log likelihood loss:

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

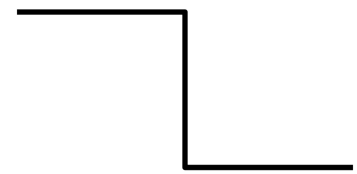
- What's the derivative with respect to one of the parameters?

$$\frac{\partial J}{\partial a} = -\frac{1}{m}\sum_i y^{(i)} \frac{1}{\sigma(\theta \cdot x^{(i)})} \; \partial\sigma(\theta \cdot x^{(i)}) \; x_1^{(i)} + (1 - y(i)) \ldots$$

$$= -\frac{1}{m}\sum_i y^{(i)}(1 - \sigma(\theta \cdot x^{(i)})) \; x_1^{(i)} - (1 - y^{(i)}) \ldots$$

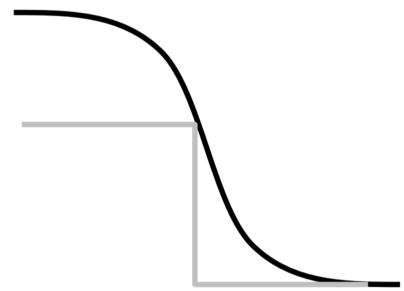
Surrogate loss functions

- Replace 0/1 loss $\Delta_i(\theta) = \delta\big(T(\theta x^{(i)}) \neq y^{(i)}\big)$
  with something easier:

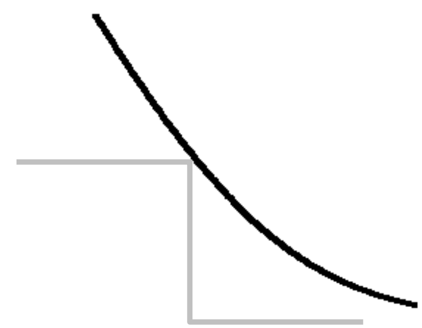


0/1 Loss

- Logistic MSE
  $$J_i(\theta) = 4\big(\sigma(\theta x^{(i)}) - y^{(i)}\big)^2$$



- Logistic Neg Log Likelihood
  $$J_i(\underline{\theta}) = -y^{(i)} \log \sigma(\theta \cdot x^{(i)}) + \ldots$$



Summary

- Linear classifier ⇔ perceptron
- Visualizing the decision boundary
- Measuring quality of a decision boundary
  - Logistic sigmoid + MSE criterion
  - Logistic Regression
- Learning the weights of a linear classifer from data
  - Reduces to an optimization problem
  - Perceptron algorithm
  - For MSE or Logistic NLL, we can do gradient descent
  - Gradient equations & update rules
- Extending features and separability