

## Lecture 2: More on regression. Gradient descent. Classification

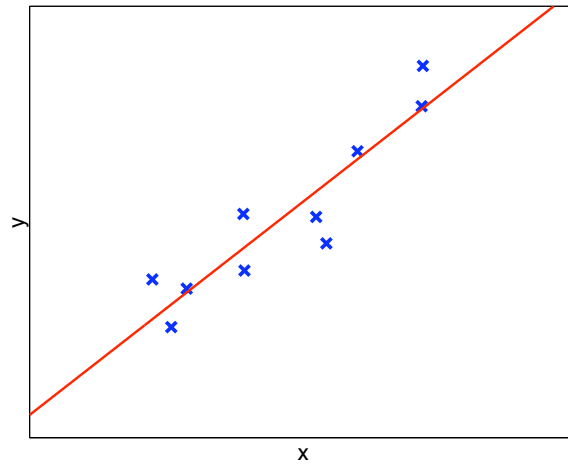
- Polynomial regression
- Overfitting
- Gradient descent
- Introduction to classification
- The perceptron learning algorithm
- A probabilistic analysis of linear regression

### Recall from last time

- The problem of supervised learning: given data  $D \subset \mathcal{X} \times \mathcal{Y}$  find a hypothesis  $h : \mathcal{X} \rightarrow \mathcal{Y}$  which approximates well the given data
- Supervised learning algorithms make specific choices about the hypothesis class, error function used to evaluate the approximation and algorithm for error minimization
- Linear regression:
  - Consider  $h$  to be a linear function
  - Consider minimizing the mean squared error between  $h$  and the true values on data set  $D$
  - Compute the gradient of the MSE and set it to 0
- We obtain a closed-form solution for the parameters

## Example

$$\mathbf{w} = (X^T X)^{-1} X^T Y$$



## Polynomial fits

- Suppose we want to fit a higher-degree polynomial to the data.  
(E.g.,  $y = w_2 x^2 + w_1 x^1 + w_0$ .)
- Suppose for now that there is a single input variable per training sample.
- How do we do it?

## Answer: linear regression

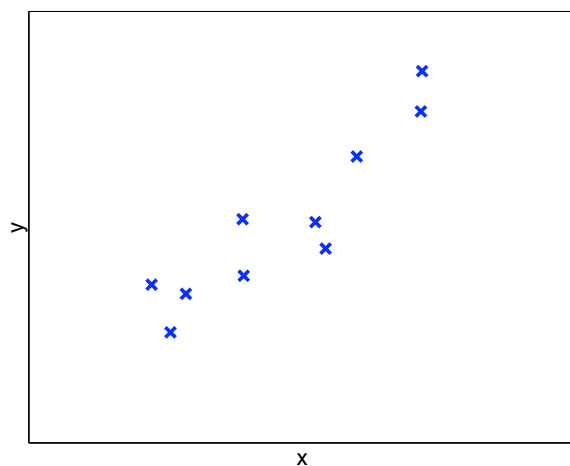
(Sometimes called polynomial regression.)

- Given data:  $(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$ .
- Suppose we want a degree- $d$  polynomial fit.
- Let  $Y$  be as before and let

$$X = \begin{bmatrix} x_1^d & \dots & x_1^2 & x_1 & 1 \\ x_2^d & \dots & x_2^2 & x_2 & 1 \\ \vdots & & \vdots & \vdots & \vdots \\ x_m^d & \dots & x_m^2 & x_m & 1 \end{bmatrix}$$

- Solve the linear regression  $X\mathbf{w} \approx Y$ .

## Example of quadratic regression



$x$	$y$
0.86	2.49
0.09	0.83
-0.85	-0.25
0.87	3.10
-0.44	0.87
-0.43	0.02
-1.10	-0.12
0.40	1.81
-0.96	-0.83
0.17	0.43

## Data matrices

$$X = \begin{bmatrix} 0.75 & 0.86 & 1 \\ 0.01 & 0.09 & 1 \\ 0.73 & -0.85 & 1 \\ 0.76 & 0.87 & 1 \\ 0.19 & -0.44 & 1 \\ 0.18 & -0.43 & 1 \\ 1.22 & -1.10 & 1 \\ 0.16 & 0.40 & 1 \\ 0.93 & -0.96 & 1 \\ 0.03 & 0.17 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{bmatrix}$$

## $X^T X$

$$X^T X = \begin{bmatrix} 0.75 & 0.01 & 0.73 & 0.76 & 0.19 & 0.18 & 1.22 & 0.16 & 0.93 & 0.03 \\ 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0.75 & 0.86 & 1 \\ 0.01 & 0.09 & 1 \\ 0.73 & -0.85 & 1 \\ 0.76 & 0.87 & 1 \\ 0.19 & -0.44 & 1 \\ 0.18 & -0.43 & 1 \\ 1.22 & -1.10 & 1 \\ 0.16 & 0.40 & 1 \\ 0.93 & -0.96 & 1 \\ 0.03 & 0.17 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 4.11 & -1.64 & 4.95 \\ -1.64 & 4.95 & -1.39 \\ 4.95 & -1.39 & 10 \end{bmatrix}$$

## $X^T Y$

$$X^T Y =$$

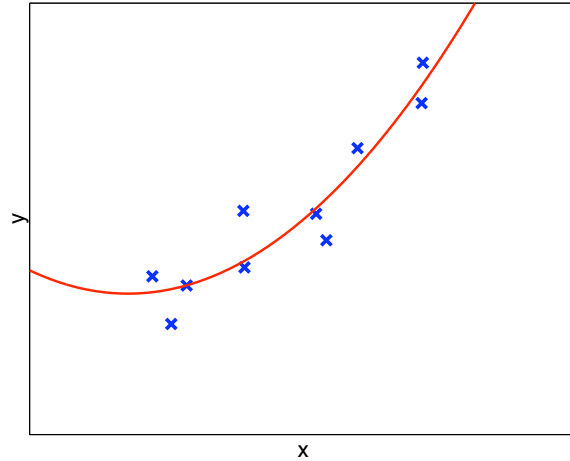
$$\begin{bmatrix} 0.75 & 0.01 & 0.73 & 0.76 & 0.19 & 0.18 & 1.22 & 0.16 & 0.93 & 0.03 \\ 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{bmatrix}$$
$$= \begin{bmatrix} 3.60 \\ 6.49 \\ 8.34 \end{bmatrix}$$

## Solving for $w$

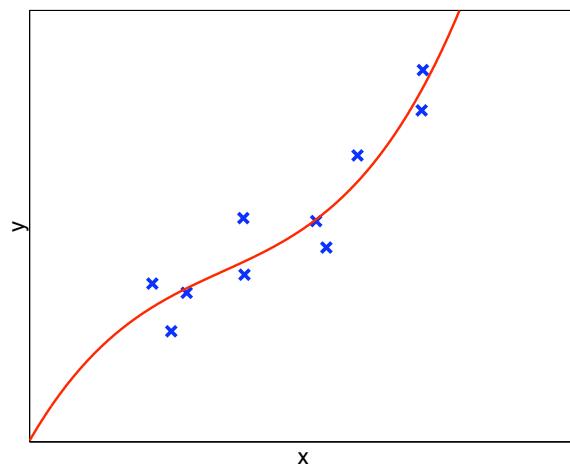
$$\mathbf{w} = (X^T X)^{-1} X^T Y = \begin{bmatrix} 4.11 & -1.64 & 4.95 \\ -1.64 & 4.95 & -1.39 \\ 4.95 & -1.39 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 3.60 \\ 6.49 \\ 8.34 \end{bmatrix} = \begin{bmatrix} 0.68 \\ 1.74 \\ 0.73 \end{bmatrix}$$

So the best order-2 polynomial is  $y = 0.68x^2 + 1.74x + 0.73$ .

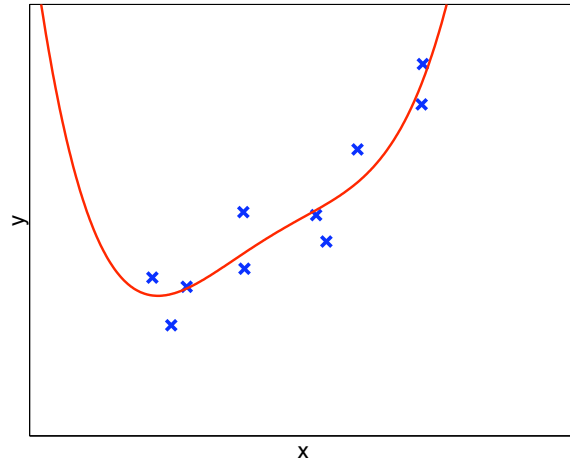
Data and curve  $y = 0.68x^2 + 1.74x + 0.73$



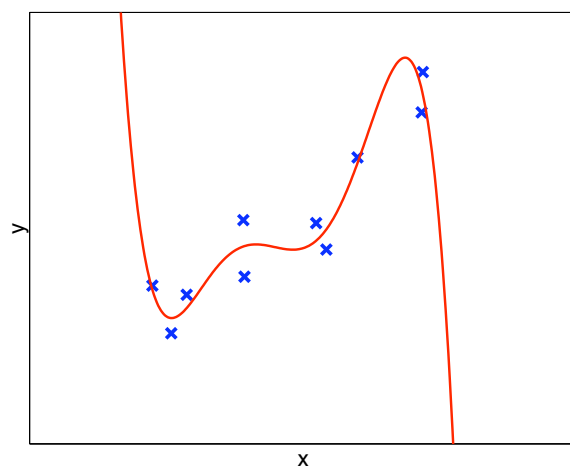
Order-3 fit



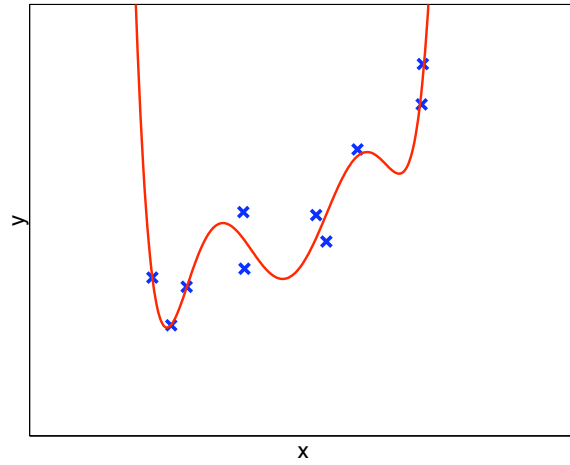
### Order-4 fit



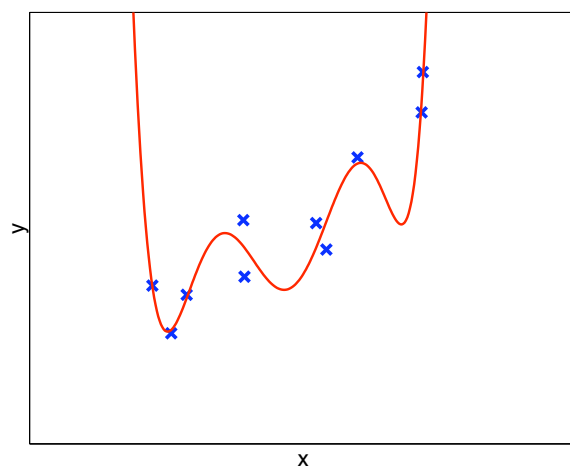
### Order-5 fit



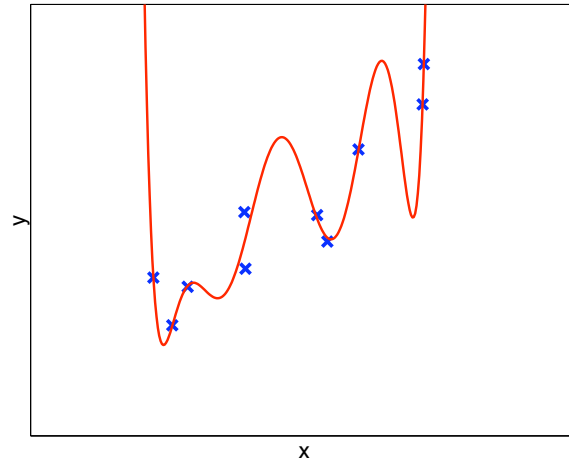
### Order-6 fit



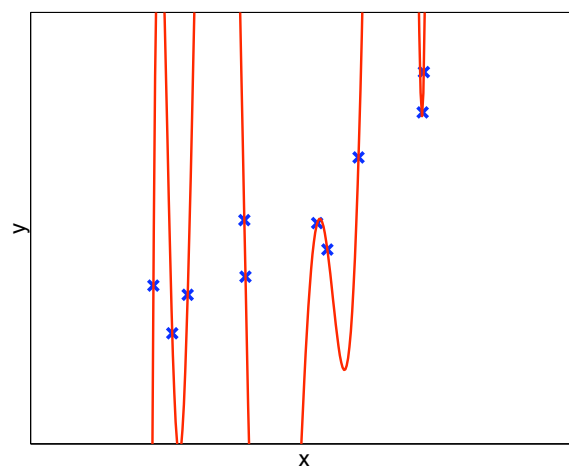
### Order-7 fit



### Order-8 fit



### Order-9 fit



## Overfitting

- A general, **HUGELY IMPORTANT** problem for all machine learning algorithms
- We can find a hypothesis that predicts perfectly the training data but **does not generalize** well to new data
- E.g., a lookup table!
- We are seeing an instance here: if we have a lot of parameters, the hypothesis "memorizes" the data points, but is wild everywhere else.

## Overfitting more formally

- Every hypothesis has a "true" error  $J^*(h)$  (measured on all possible data items we could ever encounter)
- Because we do not have all the data, we measure the error on the training set  $J_D(h)$
- Suppose we compare hypotheses  $h_1$  and  $h_2$  on the training set, and  $J_D(h_1) < J_D(h_2)$
- If  $h_2$  is "truly" better, i.e.  $J^*(h_2) < J^*(h_1)$ , our algorithm is overfitting.
- We need theoretical and empirical methods to guard against it!

## Leave-one-out cross-validation

- How can we choose the best  $d$  for an order- $d$  polynomial fit to the data?
- Repeat the following procedure:
  - Leave out one instance from the training set, to estimate the true prediction error for the best order- $d$  fit for  $d \in \{1, 2, \dots, 9\}$ .
  - Use all the other data items for finding  $w$
  - Measure the error on the instance left out
  - This is an unbiased estimate of the true prediction error
- Choose the  $d$  with lowest average prediction error

## Cross-validation

- A general procedure for estimating the true error of a predictor
- The data is split into three subsets:
  - A training set used only to find the parameters  $w$
  - A validation set used to find the right hypothesis class (e.g. the degree of the polynomial)
  - A test set used to report the prediction error of the algorithm
- These set must be disjoint!
- The process is repeated several times, and the results are averaged to provide error estimates.

## Estimating true error for $d = 1$

$D = \{(0.86, 2.49), (0.09, 0.83), (-0.85, -0.25), (0.87, 3.10), (-0.44, 0.87), (-0.43, 0.02), (-1.10, -0.12), (0.40, 1.81), (-0.96, -0.83), (0.17, 0.43)\}$ .

Iter	$D_{train}$	$D_{valid}$	Error <sub>train</sub>	Error <sub>valid</sub>
1	$D - \{(0.86, 2.49)\}$	(0.86, 2.49)	0.4928	0.0044
2	$D - \{(0.08, 0.83)\}$	(0.09, 0.83)	0.1995	0.1869
3	$D - \{(-0.85, -0.25)\}$	(-0.85, -0.25)	0.3461	0.0053
4	$D - \{(0.87, 3.10)\}$	(0.87, 3.10)	0.3887	0.8681
5	$D - \{(-0.44, 0.87)\}$	(-0.44, 0.87)	0.2128	0.3439
6	$D - \{(-0.43, 0.02)\}$	(-0.43, 0.02)	0.1996	0.1567
7	$D - \{(-1.10, -0.12)\}$	(-1.10, -0.12)	0.5707	0.7205
8	$D - \{(0.40, 1.81)\}$	(0.40, 1.81)	0.2661	0.0203
9	$D - \{(-0.96, -0.83)\}$	(-0.96, -0.83)	0.3604	0.2033
10	$D - \{(0.17, 0.43)\}$	(0.17, 0.43)	0.2138	1.0490
mean:			0.2188	0.3558

## Cross-validation results

$d$	Error <sub>train</sub>	Error <sub>valid</sub>
1	0.2188	0.3558
2	0.1504	0.3095
3	0.1384	0.4764
4	0.1259	1.1770
5	0.0742	1.2828
6	0.0598	1.3896
7	0.0458	38.819
8	0.0000	6097.5
9	0.0000	6097.5

- Optimal choice:  $d = 2$ . Overfitting for  $d > 2$
- Why are  $d = 8$  and  $d = 9$  the same?

## Linear and polynomial regression summary

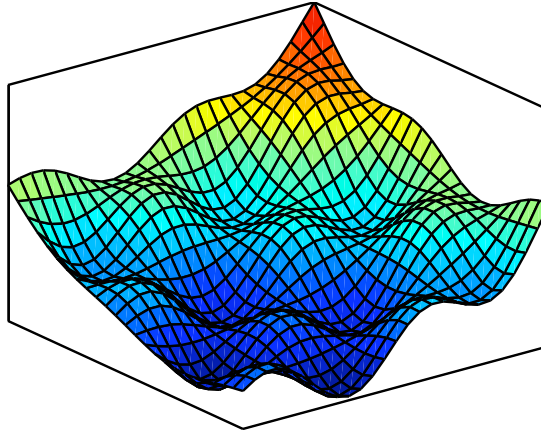
- We can fit linear and polynomial functions in polynomial time by solving  $\mathbf{w} = (X^T X)^{-1} X^T Y$ .
- We can use cross-validation to choose the best order of polynomial to fit our data.
- Issue: How many coefficients does an order- $d$  polynomial have if there are two input variables?  $m$  input variables?
  - Often, one will use powers of individual input variables but no cross terms, or only select cross-terms (based on domain knowledge).
- The inverse  $(X^T X)^{-1}$  may not exist if we have too few samples and/or try to fit too many parameters. Dimensionality reduction or “regularization” can solve this problem.

## Some problems

- If the data set is large, we may not be able to compute  $(X^T X)^{-1}$ , or this process may be very slow
- We could sub-sample the data (select fewer attributes and instances), but this will lose information
- More generally, if the hypothesis class  $\mathcal{H}$  is complicated, we may not be able to solve for the optimal parameter vector  $\mathbf{w}$ .
- A more general procedure for optimizing an error function: gradient descent

## Gradient descent for optimization

- The gradient of  $f$  at a point  $\langle u_1, u_2, \dots, u_n \rangle$  can be thought of as a vector indicating which way is “uphill”.



- If this is an error function, we want to move “downhill” on it, i.e., in the direction opposite to the gradient

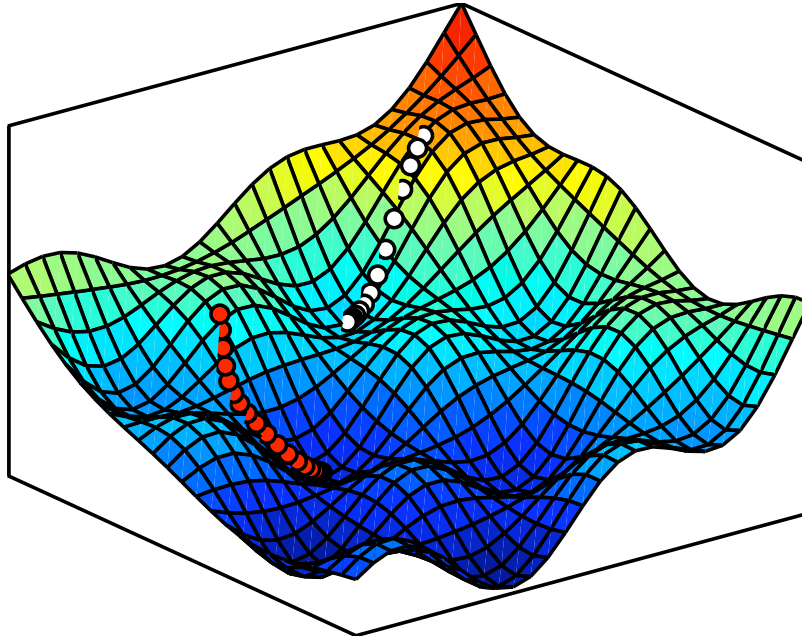
## Gradient descent

- The basic algorithm assumes that  $\nabla f$  is easily computed
- We want to produce a sequence of vectors  $\mathbf{u}^1, \mathbf{u}^2, \mathbf{u}^3, \dots$  with the goal that:
  - $f(\mathbf{u}^1) > f(\mathbf{u}^2) > f(\mathbf{u}^3) > \dots$
  - $\lim_{i \rightarrow \infty} \mathbf{u}^i = \mathbf{u}$  and  $\mathbf{u}$  is locally optimal.
- The algorithm: Given  $\mathbf{u}^0$ , do for  $i = 0, 1, 2, \dots$

$$\mathbf{u}^{i+1} = \mathbf{u}^i - \alpha_i \nabla f(\mathbf{u}^i),$$

where  $\alpha_i > 0$  is the step size or learning rate for iteration  $i$ .

## Example gradient descent traces



## Convergence

- Convergence depends in part on the  $\alpha_i$ .
- If they are too large (such as constant) oscillation or “bubbling” may occur.  
(This suggests the  $\alpha_i$  should tend to zero as  $i \rightarrow \infty$ .)
- If they are too small, the  $\mathbf{u}^i$  may not move far enough to reach a local minimum.

## Robbins-Monroe conditions

- The  $\alpha_i$  are a Robbins-Monroe sequence if:
  - $\sum_{i=0}^{\infty} \alpha_i = +\infty$
  - $\sum_{i=0}^{\infty} \alpha_i^2 < \infty$
- E.g.,  $\alpha_i = \frac{1}{i+1}$  (averaging)
- E.g.,  $\alpha_i = \frac{1}{2}$  for  $i = 1 \dots T$ ,  $\alpha_i = \frac{1}{2^2}$  for  $i = T + 1, \dots (T + 1) + 2T$  etc
- These conditions, along with appropriate conditions on  $f$  are sufficient to ensure convergence of the  $\mathbf{u}^i$ .
- Many variants are possible: e.g., we may use at each step a random vector with mean  $\nabla f(\mathbf{u}^i)$ ; this is stochastic gradient descent.

## “Batch” versus “On-line” optimization

- Often in machine learning the error function,  $J$ , is a sum of errors attributed to each instance: ( $J = J_1 + J_2 + \dots + J_m$ .)
- In batch gradient descent, the true gradient is computed at each step:

$$\nabla J = \nabla J_1 + \nabla J_2 + \dots + \nabla J_m.$$

- In on-line gradient descent, at each iteration one instance,  $i \in \{1, \dots, m\}$ , is chosen at random and only  $\nabla J_i$  is used in the update.
- Why prefer one or the other?

## “Batch” versus “On-line” optimization

- Batch is simple, repeatable.
- On-line:
  - Requires less computation per step.
  - Randomization may help escape poor local minima.
  - Allows working with a stream of data, rather than a static set (hence “on-line”).

## Termination

There are many heuristics for deciding when to stop gradient descent.

1. Run until  $\|\nabla f\|$  is smaller than some threshold.
2. Run it for as long as you can stand.
3. Run it for a short time from 100 different starting points, see which one is doing best, goto 2.
4. ...

## Batch gradient descent for linear regression

- Start with an initial guess for  $\mathbf{w}$
- Repeatedly change  $\mathbf{w}$  to make  $J(\mathbf{w})$  smaller:

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} J(\mathbf{w}), \quad \forall j = 0 \dots n$$

- For linear hypotheses, we get:

$$w_j \leftarrow w_j + \alpha \sum_{i=1}^m (y_i - h_{\mathbf{w}}(\mathbf{x}_i)) x_{i,j}$$

- This method is also known as LMS update rule or Widrow-Hoff learning rule

## Incremental (stochastic) gradient descent

1. Sample (choose) a training example  $\langle \mathbf{x}, y \rangle$ , in a randomized way
- 2.

$$w_j \leftarrow w_j + \alpha (y - h_{\mathbf{w}}(\mathbf{x})) x_j$$

3. Repeat at will

Advantages:

- Better for large data sets
- Often faster than batch gradient descent
- Less prone to local minima

## [Quasi-]linear models for classification

- Recall: in a binary classification problem the outputs,  $y_i$ , can take two discrete values. (As convenient, we will assume they are  $-1$  and  $+1$ , or  $0$  and  $1$ .)
- Can we develop linear models for classification as we did for regression?
- What happens if we just apply linear regression as is?

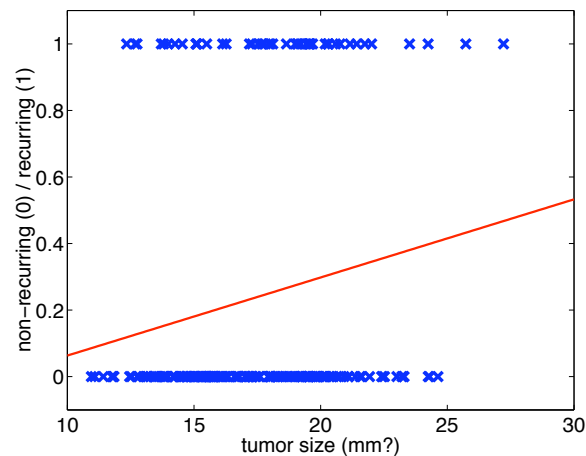
---

September 6, 2005

37

COMP-652 Lecture 2

### Example: Wisconsin data



What is the meaning of the output in this case???

---

September 6, 2005

38

COMP-652 Lecture 2

## Output of a classifier

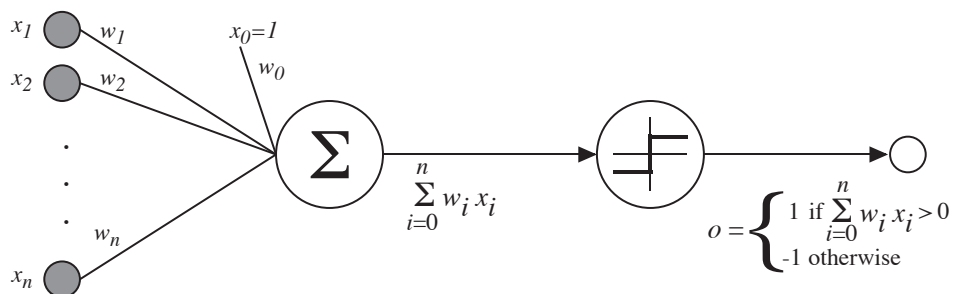
- Useful predictions could be:
  - The predicted class
  - The probability that the example belongs to a given class
- The second approach arguably is more useful
- Just applying linear regression as is gives us *neither*

September 6, 2005

39

COMP-652 Lecture 2

## Perceptron



- We can take a linear combination and threshold it:

$$h_{\mathbf{w}}(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

This is called a perceptron.

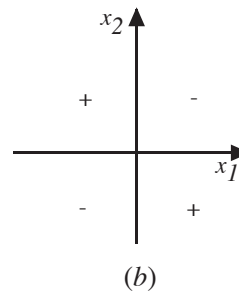
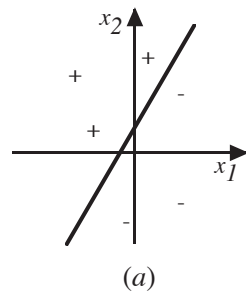
- The output is taken as the predicted class.

September 6, 2005

40

COMP-652 Lecture 2

## Decision surface of a perceptron



- The decision surface is a line (examples on each side are classified differently)
- Represents some useful functions.  
Example: what weights represent  $AND(x_1, x_2)$ ?
- But some functions not linearly separable! E.g. XOR.  
To represent them, we would need networks of perceptron-like elements.

## Learning for perceptrons

- We seek  $\mathbf{w}$  which maximize the number of correctly classified instances:

$$J(\mathbf{w}) = \text{number of instances misclassified by } h_{\mathbf{w}}$$

This is also called *the 0-1 loss function*.

- Correctly classifying instance  $i$  means  $y_i(\mathbf{w}^T \mathbf{x}_i) > 0$ , where  $y_i \in \{-1, 1\}$ .
- Hence, we can use as an error function the following

$$J(\mathbf{w}) = - \sum_{i \in \text{Misclassified}} y_i(\mathbf{w}^T \mathbf{x}_i)$$

- This will be 0 if all examples are correctly classified, positive otherwise

## How do we find the best $w$ ?

- Stochastic gradient descent on the previous error function:
  - Initialize  $w$  somehow.
  - While some misclassified samples remain:
    1. Choose a misclassified sample,  $i$ .
    2.  $w \leftarrow w + \alpha y_i x_i$ , where  $\alpha$  is a step-size parameter.
- If the data is linearly separable, then under the appropriate conditions on  $\alpha$  this converges to a  $w$  with zero error.
- If the data is not linearly separable, the weights oscillate

## What is a good error function for classification?

- The 0-1 loss makes sense if we want to predict the label of an instance, but is not differentiable
- This will make our life hard if we want to apply gradient descent on the error function, as well as other optimization procedures.
- In general, how do we find error functions?  
E.g., for linear regression, we used the sum-squared error, but why is this a good choice?

## Linear regression revisited: A probabilistic assumption

- Assume  $y_i$  is a noisy target value, generated from a linear hypothesis
- More specifically, assume that there exists  $\mathbf{w}$  such that:

$$y_i = h_{\mathbf{w}}(\mathbf{x}_i) + e_i$$

where  $e_i$  is random variable (noise) drawn independently for each  $\mathbf{x}_i$  according to some Gaussian (normal) distribution with mean zero and variance  $\sigma$ .

If you do not recall Gaussians, check out Duda, Hart & Stork, Sect. 2.5, the tutorial, or any basic probability textbook.

- How should we choose our parameter vector  $\mathbf{w}$ ?

## Bayes theorem in learning

Let  $h$  be a hypothesis and  $D$  be the set of training data. Using Bayes theorem, we have:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)},$$

where:

- $P(h)$  = prior probability of hypothesis  $h$
- $P(D)$  = prior probability of training data  $D$
- $P(h|D)$  = probability of  $h$  given  $D$
- $P(D|h)$  = probability of  $D$  given  $h$

## Choosing hypotheses

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

What is the most probable hypothesis given the training data?

**Maximum a posteriori (MAP) hypothesis**  $h_{MAP}$ :

$$\begin{aligned} h_{MAP} &= \arg \max_{h \in H} P(h|D) \\ &= \arg \max_{h \in H} \frac{P(D|h)P(h)}{P(D)} \text{ (using Bayes theorem)} \\ &= \arg \max_{h \in H} P(D|h)P(h) \end{aligned}$$

## Maximum likelihood estimation

$$h_{MAP} = \arg \max_{h \in H} P(D|h)P(h)$$

- If we assume  $P(h_i) = P(h_j)$  (all hypotheses are equally likely a priori) then can further simplify, and choose the **maximum likelihood (ML) hypothesis**:

$$h_{ML} = \arg \max_{h \in H} P(D|h) = \arg \max_{h \in H} L(h)$$

- Standard assumption: the training examples are independent of each other (i.i.d. samples)
- This allows us to simplify  $P(D|h)$ :

$$P(D|h) = \prod_{i=1}^m P(\langle \mathbf{x}_i, y_i \rangle | h) = \prod_{i=1}^m P(y_i | \mathbf{x}_i; h)$$

## The log trick

- We want to maximize:

$$L(h) = \prod_{i=1}^m P(y_i | \mathbf{x}_i; h)$$

This is a product, and products are hard to maximize!

- Instead, we will maximize  $\log L(h)$ ! (the log-likelihood function)

$$\log L(h) = \sum_{i=1}^m \log P(y_i | \mathbf{x}_i; h)$$

## Maximum likelihood for linear regression

- Adopt the assumption that:

$$y_i = h_{\mathbf{w}}(\mathbf{x}_i) + e_i,$$

where  $e_i$  are normally distributed with mean 0 and variance  $\sigma$

- Hence,

$$L(\mathbf{w}) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2} \left( \frac{y_i - h_{\mathbf{w}}(\mathbf{x}_i)}{\sigma} \right)^2}$$

because the noise variables  $e_i$  are from a Gaussian distribution

## Applying the log trick

$$\begin{aligned}\log L(\mathbf{w}) &= \sum_{i=1}^m \log \left( \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2} \frac{(y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2}{\sigma^2}} \right) \\ &= \sum_{i=1}^m \log \left( \frac{1}{\sqrt{2\pi\sigma^2}} \right) - \sum_{i=1}^m \frac{1}{2} \frac{(y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2}{\sigma^2}\end{aligned}$$

Maximizing the right hand side is the same as minimizing:

$$\sum_{i=1}^m \frac{1}{2} \frac{(y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2}{\sigma^2}$$

This is our old friend, the sum-squared-error function!

## Maximum likelihood hypothesis for linear estimators

- Under the assumption that the training examples are i.i.d. and that we have Gaussian target noise, the maximum likelihood parameters  $\mathbf{w}$  are those minimizing the sum squared error:

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \sum_{i=1}^m (y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2$$

- This makes explicit the hypothesis behind minimizing the sum-squared error
- Next lecture we will use a similar approach for classification.