EEE5015: Machine Learning & Artificial Intelligence

Zhiyun Lin

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY
电子与电气工程系
DEPARTMENT OF ELECTRONIC AND ELECTRICAL ENGINEERING

Lecture 06: Linear Regression and Linear Classification

- Linear Regression
  - Single-dimensional input
  - Multi-dimensional inputs
  - Fitting a polynomial
  - Generalization
  - Regularized least squares

Recap: Regression

- Regression: to predict continuous outputs *t*
  - Consider proper features (inputs): *x* (or **x** if vectors)
  - Training examples, many $x(\mathrm{i})$ for which *t(i)* is known (labeled)
  - A model, a function that represents the relationship between *x* and t

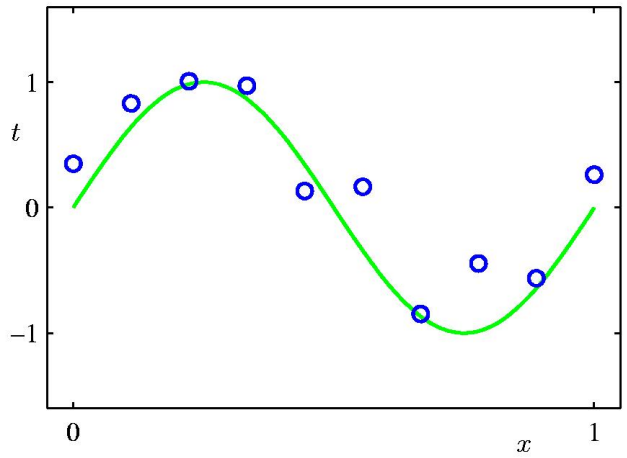
$$y = f(x, w)$$

  - A loss or a cost or an objective function, which tells us how well our model approximates the training examples
  - Optimization, a way of finding the parameters w of our model that minimizes the loss function

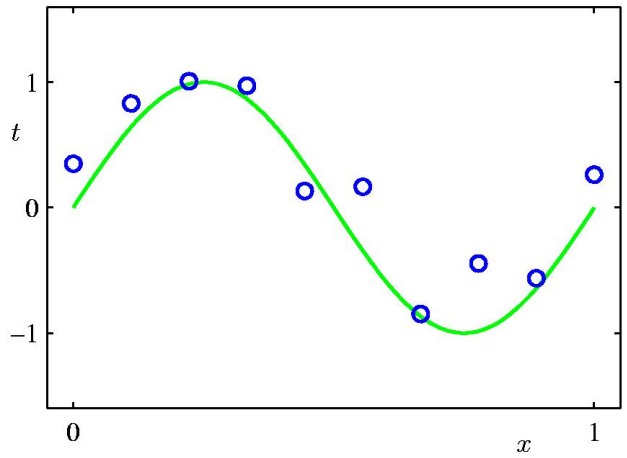
Today: Linear Regression

- Linear regression
  - continuous outputs
  - simple model (linear)
- Introduce key concepts:
  - loss functions
  - generalization
  - optimization
  - model complexity
  - regularization

Squared Error (SSE/MSE) Loss



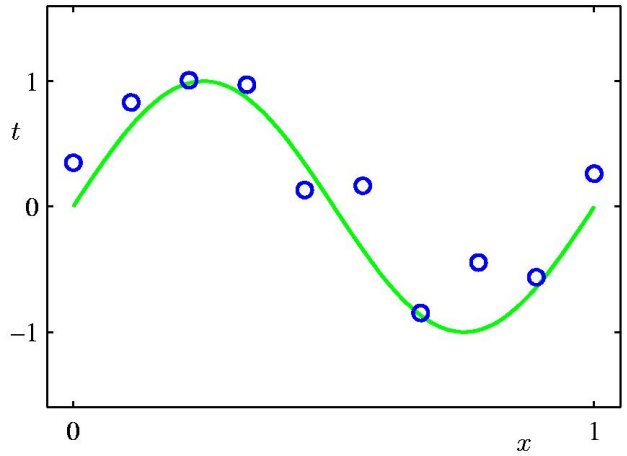
Squared Error (SSE/MSE) Loss



- Define a model

$$y(x) = \text{function}(x, \mathbf{w})$$

Squared Error (SSE/MSE) Loss



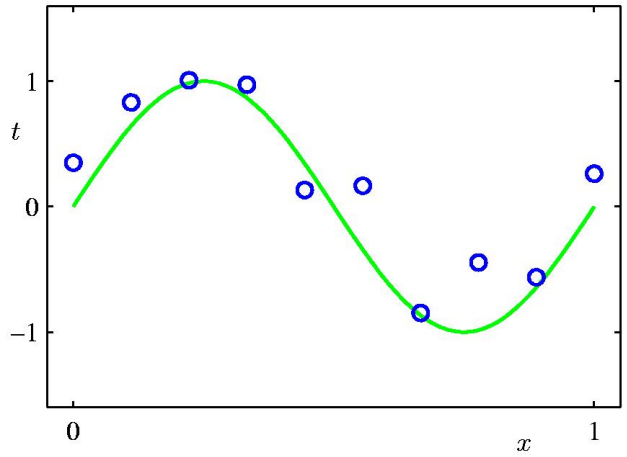
- Define a model

$$\text{Linear:} \qquad y(x) = w_0 + w_1 x$$

- Standard loss/cost/objective function measures the squared error between y and the true value t

$$\ell(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - y(x^{(n)})]^2$$

Squared Error (SSE/MSE) Loss



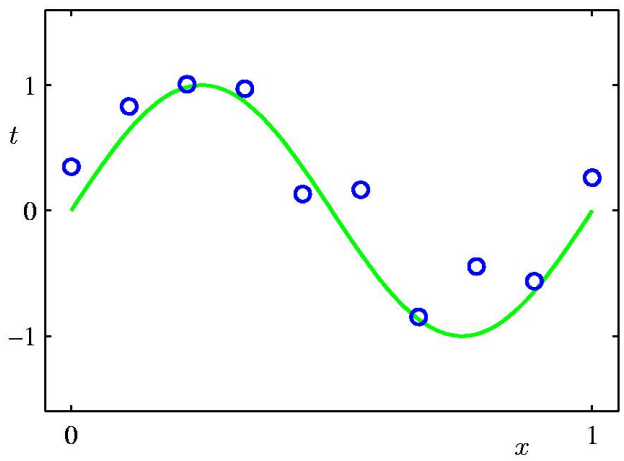
- Define a model

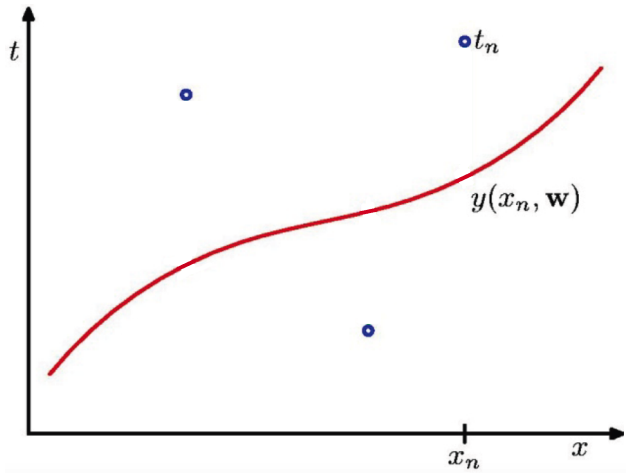
$$\text{Linear:} \qquad y(x) = w_0 + w_1 x$$

- Standard loss/cost/objective function measures the squared error between y and the true value t

$$\text{Linear model:} \qquad \ell(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2$$

Squared Error (SSE/MSE) Loss





- Define a model

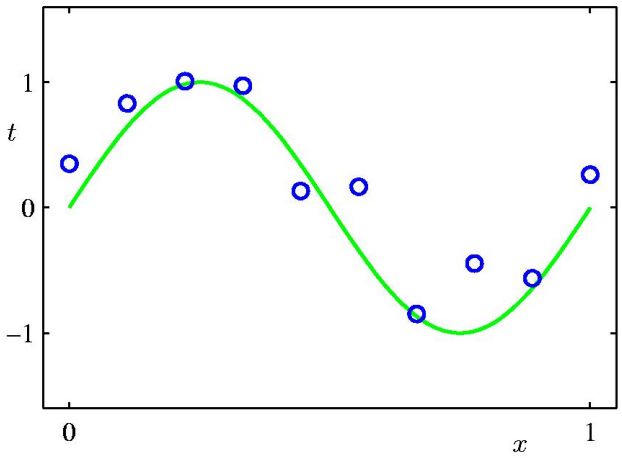
$$\text{Linear:} \qquad y(x) = w_0 + w_1 x$$

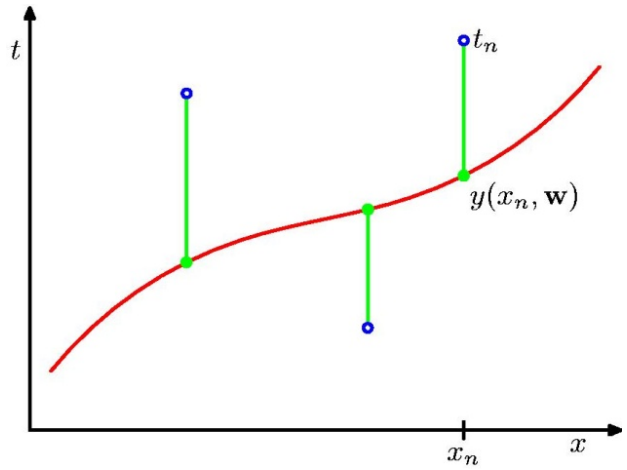
- Standard loss/cost/objective function measures the squared error between y and the true value t

$$\text{Linear model:} \qquad \ell(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2$$

- For a particular hypothesis ($y(x)$ defined by a choice of $\mathbf{w}$, drawn in red), what does the loss represent geometrically?

Squared Error (SSE/MSE) Loss





- Define a model

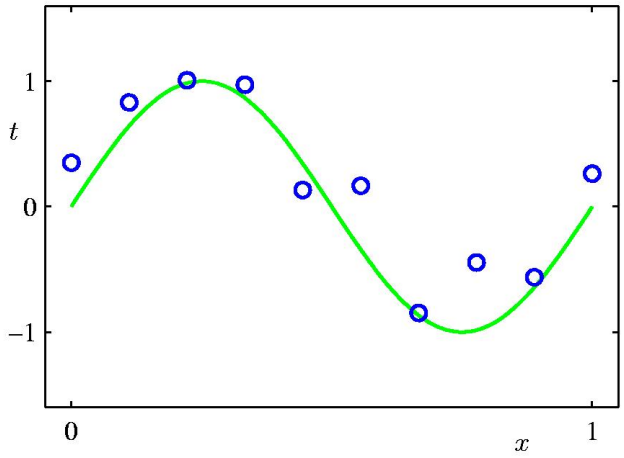
$$\text{Linear:} \qquad y(x) = w_0 + w_1 x$$

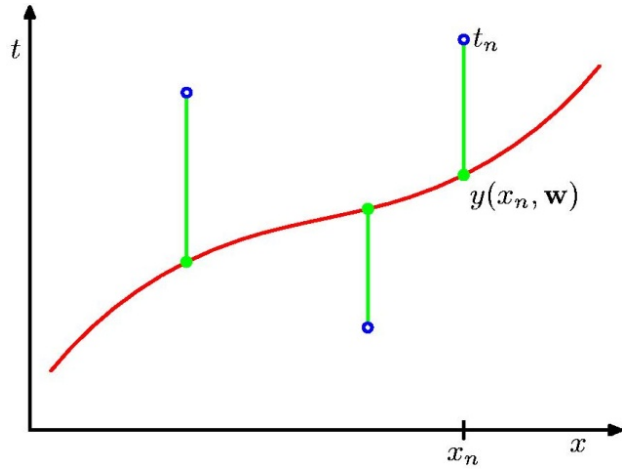
- Standard loss/cost/objective function measures the squared error between y and the true value t

$$\text{Linear model:} \qquad \ell(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2$$

- The loss for the red hypothesis is the **sum of the squared vertical errors** (squared lengths of green vertical lines)

Squared Error (SSE/MSE) Loss





- Define a model

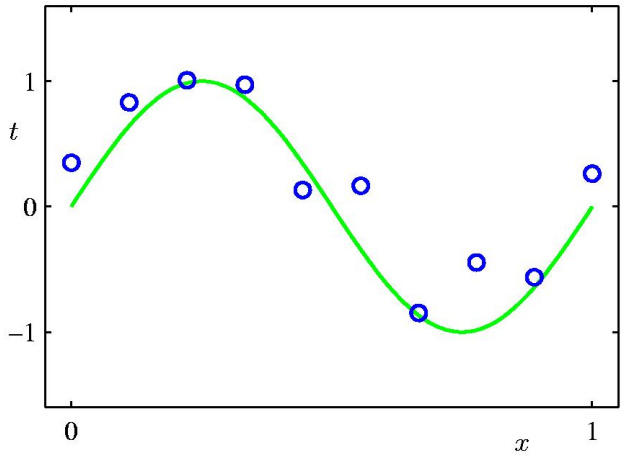
$$\text{Linear:} \qquad y(x) = w_0 + w_1 x$$

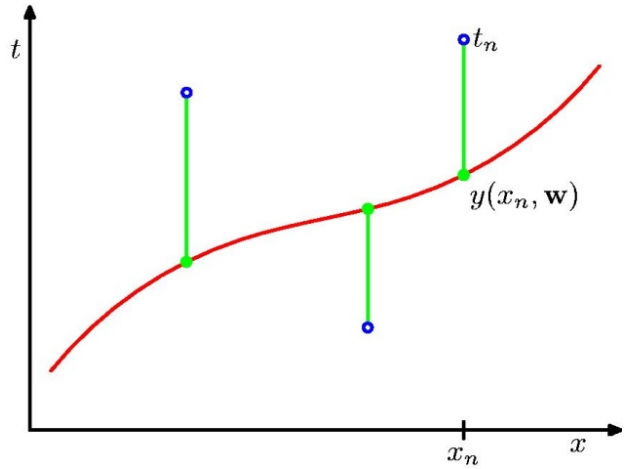
- Standard loss/cost/objective function measures the squared error between y and the true value t

$$\text{Linear model:} \qquad \ell(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2$$

- How do we obtain weights $\mathbf{w} = (w_0, w_1)$?

Squared Error (SSE/MSE) Loss





- Define a model

$$\text{Linear:} \qquad y(x) = w_0 + w_1 x$$

- Standard loss/cost/objective function measures the squared error between y and the true value t

$$\text{Linear model:} \qquad \ell(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2$$

- How do we obtain weights $\mathbf{w} = (w_0, w_1)$? Find $\mathbf{w}$ that minimizes loss $\ell(\mathbf{w})$

Optimizing the Objective

- One straightforward method: gradient descent

Optimizing the Objective

- One straightforward method: gradient descent
  - initialize $\mathbf{w}$ (e.g., randomly)

Optimizing the Objective

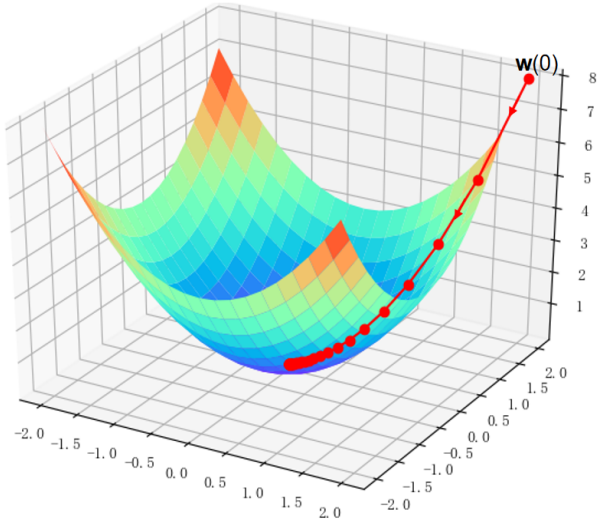
- One straightforward method: gradient descent
  - initialize $\mathbf{w}$ (e.g., randomly)
  - repeatedly update $\mathbf{w}$ based on the gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \lambda \frac{\partial \ell}{\partial \mathbf{w}}$$

Optimizing the Objective

- One straightforward method: gradient descent
  - initialize $\mathbf{w}$ (e.g., randomly)
  - repeatedly update $\mathbf{w}$ based on the gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \lambda \frac{\partial \ell}{\partial \mathbf{w}}$$

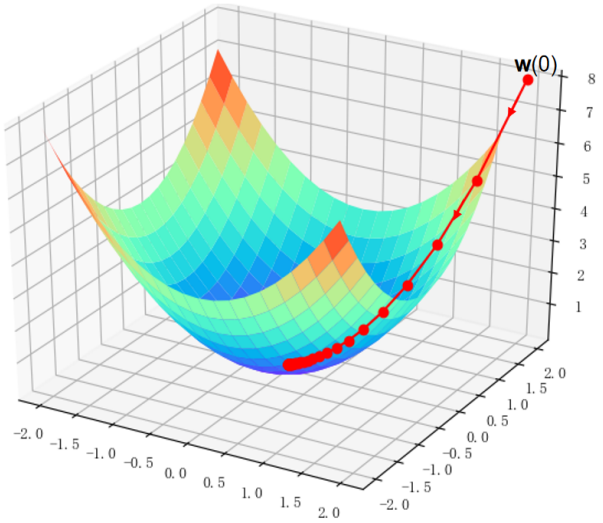


Optimizing the Objective

- One straightforward method: gradient descent
  - initialize $\mathbf{w}$ (e.g., randomly)
  - repeatedly update $\mathbf{w}$ based on the gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \lambda \frac{\partial \ell}{\partial \mathbf{w}}$$

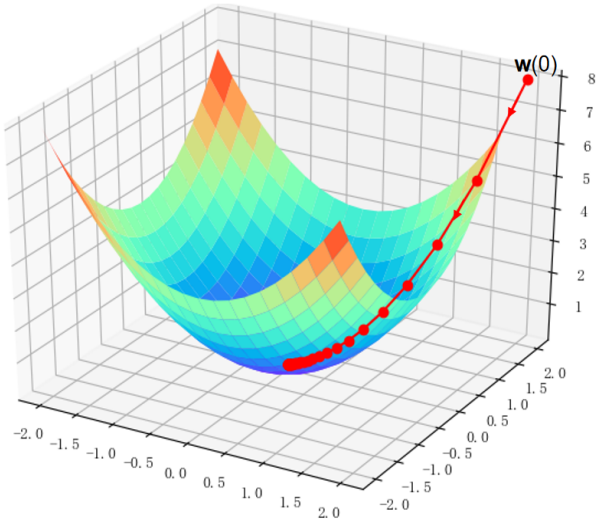
- λ is the learning rate



Optimizing the Objective

- One straightforward method: gradient descent
  - initialize $\mathbf{w}$ (e.g., randomly)
  - repeatedly update $\mathbf{w}$ based on the gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \lambda \frac{\partial \ell}{\partial \mathbf{w}}$$



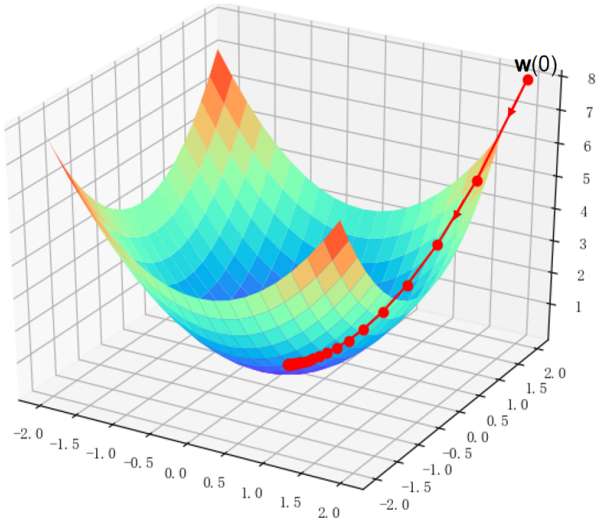
- λ is the learning rate
- For a single training case, this gives the LMS update rule (Least Mean Squares):

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda(t^{(n)} - y(x^{(n)}))\begin{bmatrix} 1 \\ x^{(n)} \end{bmatrix}$$

Optimizing the Objective

- One straightforward method: gradient descent
  - initialize $\mathbf{w}$ (e.g., randomly)
  - repeatedly update $\mathbf{w}$ based on the gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \lambda \frac{\partial \ell}{\partial \mathbf{w}}$$



- λ is the learning rate
- For a single training case, this gives the LMS update rule (Least Mean Squares):

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda \underbrace{(t^{(n)} - y(x^{(n)}))}_{\text{error}} \begin{bmatrix} 1 \\ x^{(n)} \end{bmatrix}$$

- Note: As error approaches zero, so does the update ($\mathbf{w}$ stops changing)

Optimizing Across Training Set

- Two ways to generalize this for all examples in training set:

Optimizing Across Training Set

- Two ways to generalize this for all examples in training set:
  1. Batch updates: sum or average updates across every example n, then change the parameter values

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda \sum_{n=1}^{N} (t^{(n)} - y(x^{(n)})) \begin{bmatrix} 1 \\ x^{(n)} \end{bmatrix}$$

Optimizing Across Training Set

- Two ways to generalize this for all examples in training set:
  1. Batch updates: sum or average updates across every example n, then change the parameter values

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda \sum_{n=1}^{N} (t^{(n)} - y(x^{(n)})) \begin{bmatrix} 1 \\ x^{(n)} \end{bmatrix}$$

  2. Stochastic/online updates: update the parameters for each training case in turn, according to its own gradients

Algorithm 1 Stochastic gradient descent

1: Randomly shuffle examples in the training set
2: **for** $i = 1$ to N **do**
3: Update:

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda (t^{(i)} - y(x^{(i)})) \begin{bmatrix} 1 \\ x^{(n)} \end{bmatrix} \text{ (update for a linear model)}$$

4: **end for**

Optimizing Across Training Set

- Two ways to generalize this for all examples in training set:
  1. Batch updates: sum or average updates across every example n, then change the parameter values

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda \sum_{n=1}^{N} (t^{(n)} - y(x^{(n)})) \begin{bmatrix} 1 \\ x^{(n)} \end{bmatrix}$$

  2. Stochastic/online updates: update the parameters for each training case in turn, according to its own gradients

  - Underlying assumption: sample is independent and identically distributed (i.i.d.)

Analytical Solution?

- For some objectives we can also find the optimal solution analytically
- This is the case for linear least-squares regression
- How?

Analytical Solution?

- For some objectives we can also find the optimal solution analytically
- This is the case for linear least-squares regression
- How?
- Compute the derivatives of the objective wrt $\mathbf{w}$ and equate with 0

Analytical Solution?

- For some objectives we can also find the optimal solution analytically
- This is the case for linear least-squares regression
- How?
- Compute the derivatives of the objective wrt $\mathbf{w}$ and equate with 0
- Define:

$$\mathbf{t} = [t^{(1)}, t^{(2)}, \ldots, t^{(N)}]^T$$

$$\mathbf{X} = \begin{bmatrix} 1, x^{(1)} \\ 1, x^{(2)} \\ \ldots \\ 1, x^{(N)} \end{bmatrix}$$

Analytical Solution?

- For some objectives we can also find the optimal solution analytically
- This is the case for linear least-squares regression
- How?
- Compute the derivatives of the objective wrt $\mathbf{w}$ and equate with 0
- Define:

$$\mathbf{t} = [t^{(1)}, t^{(2)}, \ldots, t^{(N)}]^T$$

$$\mathbf{X} = \begin{bmatrix} 1, x^{(1)} \\ 1, x^{(2)} \\ \ldots \\ 1, x^{(N)} \end{bmatrix}$$

- Then:

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

(work it out!)

Multi-dimensional Inputs

- One method of extending the model is to consider other input dimensions

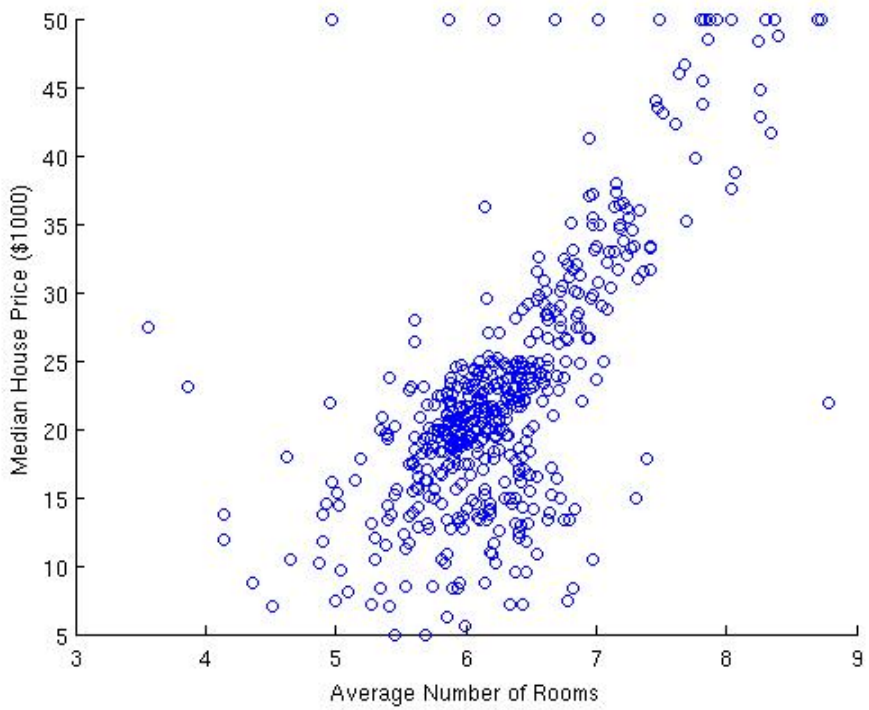
$$y(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2$$

Multi-dimensional Inputs

- One method of extending the model is to consider other input dimensions

$$y(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2$$

- In the Boston housing example, we can look at the number of rooms



Linear Regression with Multi-dimensional Inputs

- Imagine now we want to predict the median house price from these multi-dimensional observations

Linear Regression with Multi-dimensional Inputs

- Imagine now we want to predict the median house price from these multi-dimensional observations
- Each house is a data point n, with observations indexed by j:

$$\mathbf{x}^{(n)} = \left(x_1^{(n)}, \cdots, x_j^{(n)}, \cdots, x_d^{(n)}\right)$$

Linear Regression with Multi-dimensional Inputs

- Imagine now we want to predict the median house price from these multi-dimensional observations
- Each house is a data point n, with observations indexed by j:

$$\mathbf{x}^{(n)} = \left(x_1^{(n)}, \cdots, x_j^{(n)}, \cdots, x_d^{(n)}\right)$$

- We can incorporate the bias w_0 into $\mathbf{w}$, by using $x_0 = 1$, then

$$y(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j = \mathbf{w}^T \mathbf{x}$$

Linear Regression with Multi-dimensional Inputs

- Imagine now we want to predict the median house price from these multi-dimensional observations
- Each house is a data point n, with observations indexed by j:

$$\mathbf{x}^{(n)} = \left(x_1^{(n)}, \cdots, x_j^{(n)}, \cdots, x_d^{(n)}\right)$$

- We can incorporate the bias w_0 into $\mathbf{w}$, by using $x_0 = 1$, then

$$y(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j = \mathbf{w}^T \mathbf{x}$$

- We can then solve for $\mathbf{w} = (w_0, w_1, \cdots, w_d)$. How?

Linear Regression with Multi-dimensional Inputs

- Imagine now we want to predict the median house price from these multi-dimensional observations
- Each house is a data point n, with observations indexed by j:

$$\mathbf{x}^{(n)} = \left(x_1^{(n)}, \cdots, x_j^{(n)}, \cdots, x_d^{(n)}\right)$$

- We can incorporate the bias w_0 into $\mathbf{w}$, by using $x_0 = 1$, then

$$y(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j = \mathbf{w}^T \mathbf{x}$$

- We can then solve for $\mathbf{w} = (w_0, w_1, \cdots, w_d)$. How?
- We can use gradient descent to solve for each coefficient, or compute $\mathbf{w}$ analytically (how does the solution change?)

More Powerful Models?

- What if our linear model is not good? How can we create a more complicated model?

Fitting a Polynomial

- What if our linear model is not good? How can we create a more complicated model?
- We can create a more complicated model by defining input variables that are combinations of components of $\mathbf{x}$

Fitting a Polynomial

- What if our linear model is not good? How can we create a more complicated model?
- We can create a more complicated model by defining input variables that are combinations of components of $\mathbf{x}$
- Example: an M-th order polynomial function of one dimensional feature x:

$$y(x, \mathbf{w}) = w_0 + \sum_{j=1}^{M} w_j x^j$$

where x^j is the j-th power of x

Fitting a Polynomial

- What if our linear model is not good? How can we create a more complicated model?
- We can create a more complicated model by defining input variables that are combinations of components of $\mathbf{x}$
- Example: an M-th order polynomial function of one dimensional feature x:

$$y(x, \mathbf{w}) = w_0 + \sum_{j=1}^{M} w_j x^j$$

  where x^j is the j-th power of x
- We can use the same approach to optimize for the weights $\mathbf{w}$

Fitting a Polynomial

- What if our linear model is not good? How can we create a more complicated model?
- We can create a more complicated model by defining input variables that are combinations of components of **x**
- Example: an M-th order polynomial function of one dimensional feature x:

$$y(x, \mathbf{w}) = w_0 + \sum_{j=1}^{M} w_j x^j$$

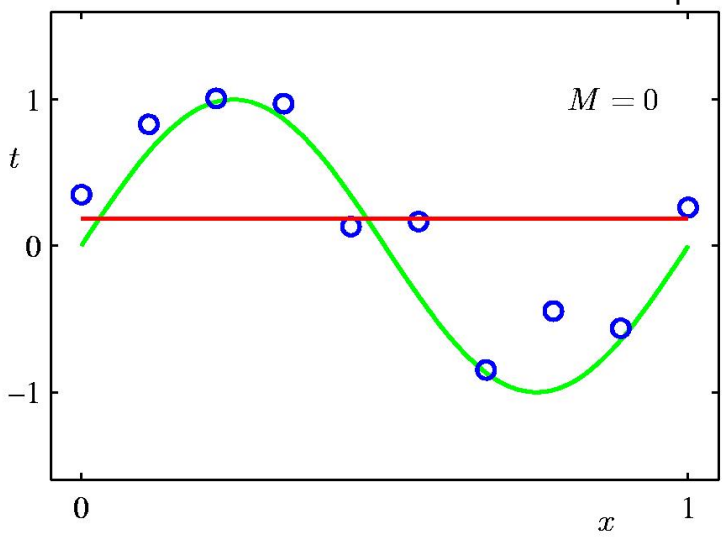
where x^j is the j-th power of x

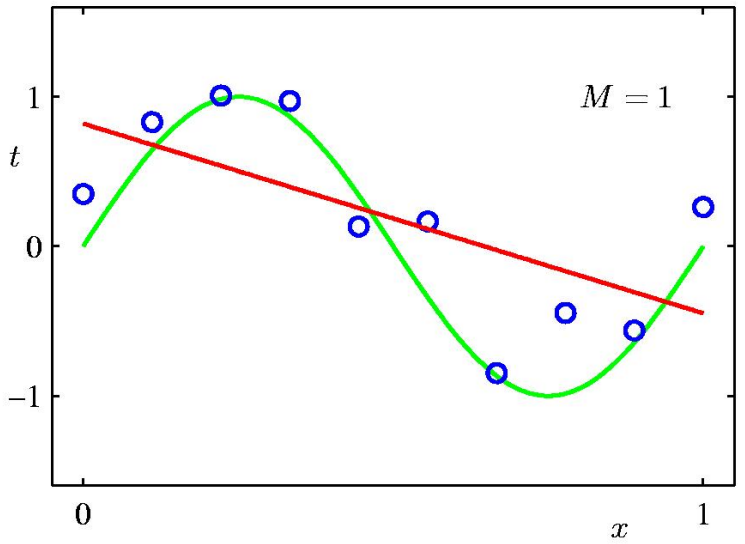
- We can use the same approach to optimize for the weights **w**
- How do we do that?

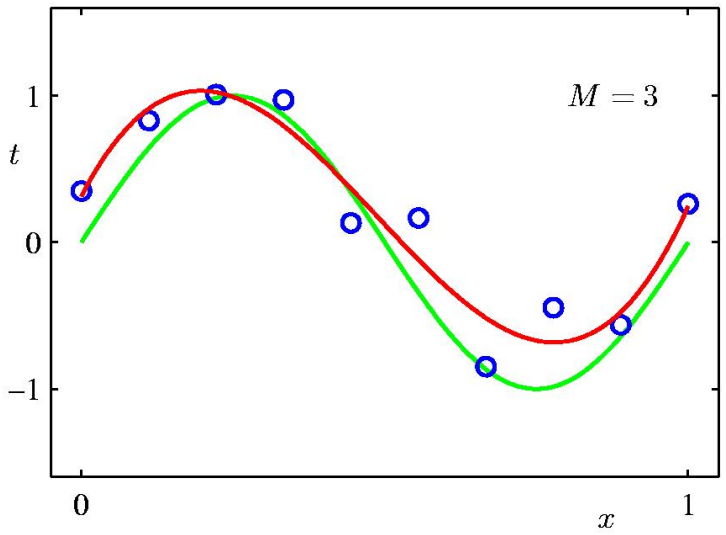
$$w = w - \lambda \frac{\partial l(w)}{\partial w} = w + 2\lambda[t - y(x, w)] \frac{\partial y}{\partial w}$$

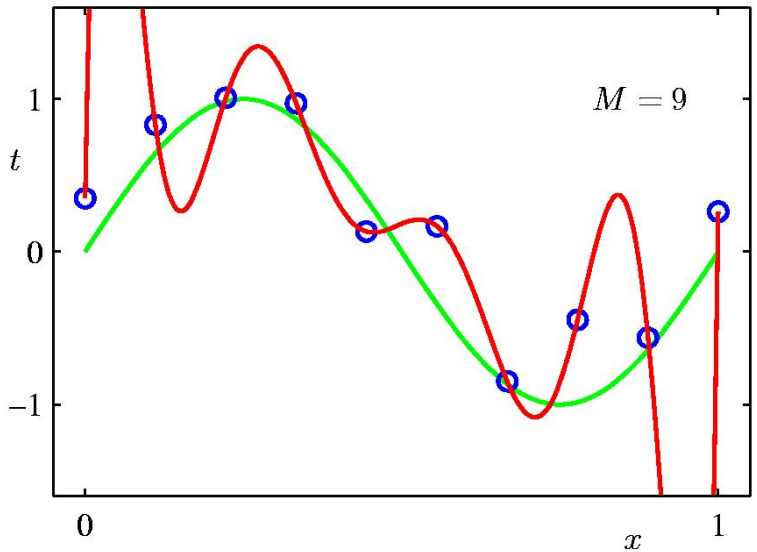
Which Fit is Best?

from Bishop



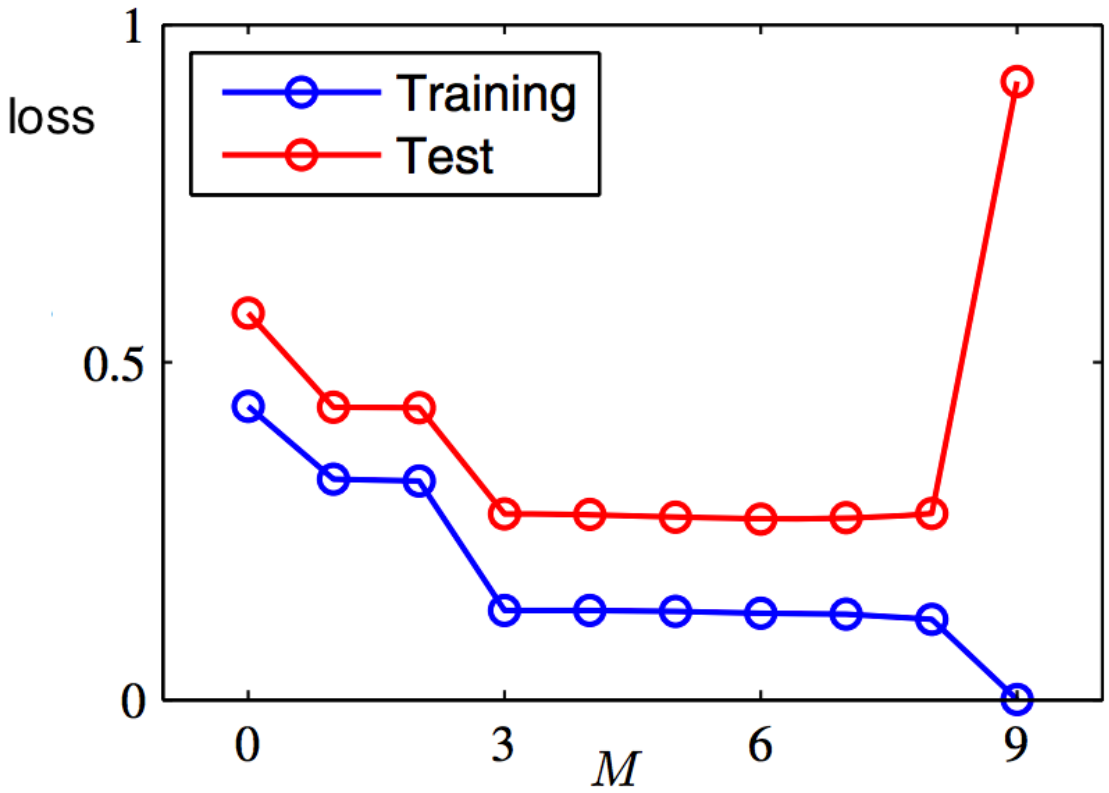






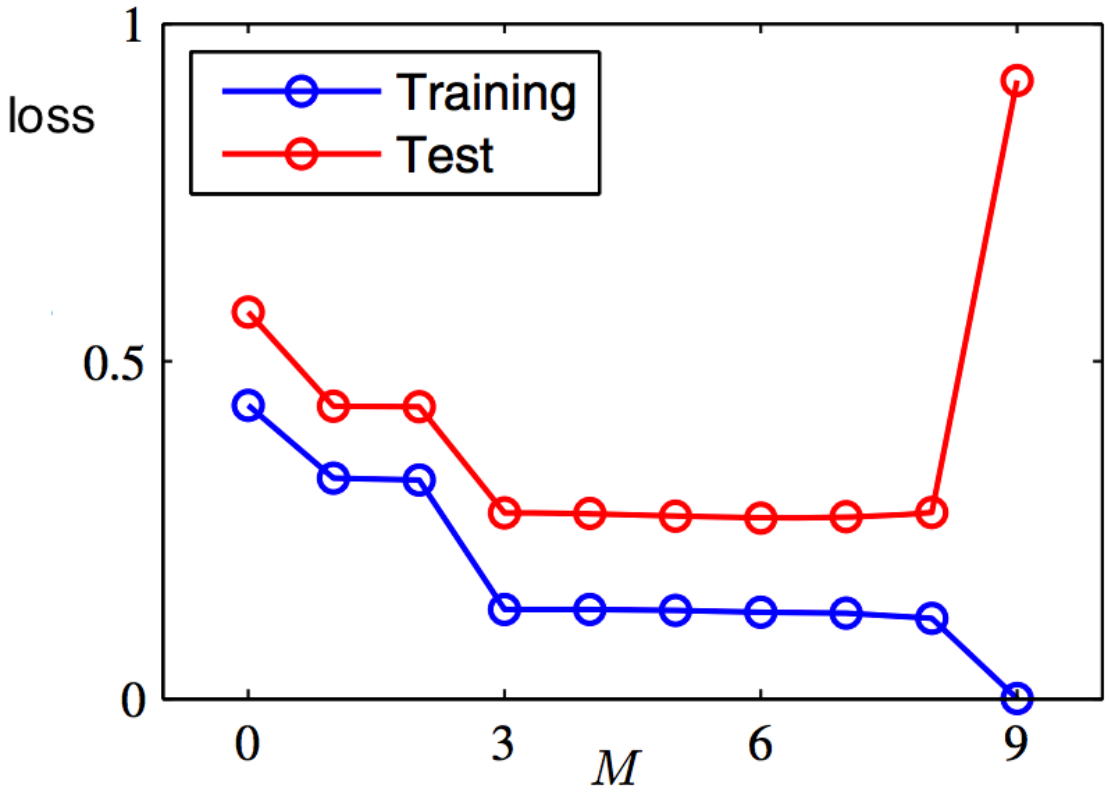
Generalization

- Generalization = model's ability to predict the held out data
- What is happening?



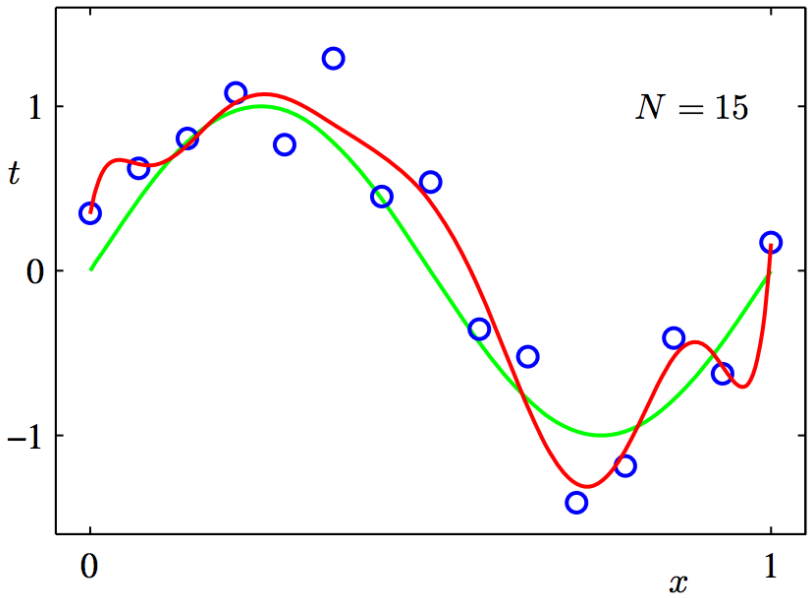
Generalization

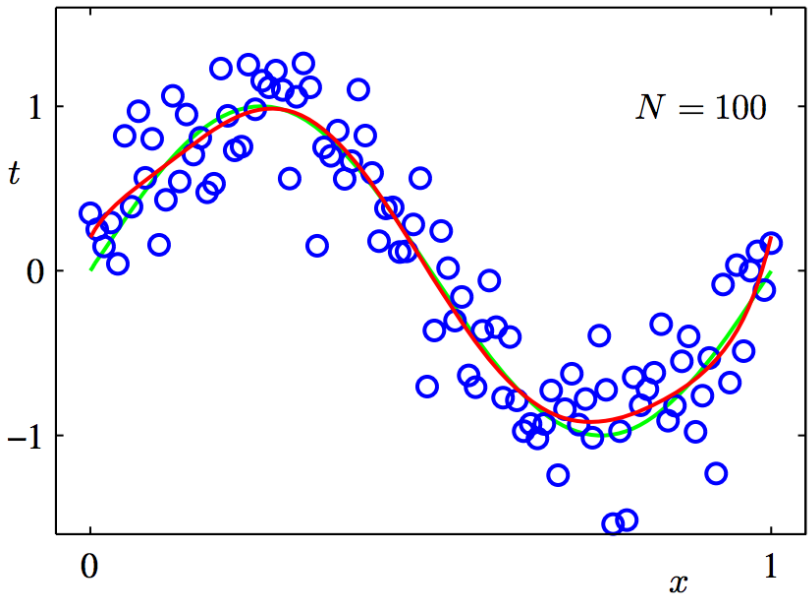
- Generalization = model's ability to predict the held out data
- What is happening?
- Our model with $M = 9$ overfits the data (it models also noise)



Generalization

- Generalization = model's ability to predict the held out data
- What is happening?
- Our model with $M = 9$ overfits the data (it models also noise)
- Not a problem if we have lots of training examples





Generalization

- Generalization = model's ability to predict the held out data
- What is happening?
- Our model with $M = 9$ overfits the data (it models also noise)
- Let's look at the estimated weights for various M in the case of fewer examples

	$M = 0$	$M = 1$	$M = 6$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Generalization

- Generalization = model's ability to predict the held out data
- What is happening?
- Our model with $M = 9$ overfits the data (it models also noise)
- Let's look at the estimated weights for various M in the case of fewer examples
- The weights are becoming huge to compensate for the noise

Generalization

- Generalization = model's ability to predict the held out data
- What is happening?
- Our model with $M = 9$ overfits the data (it models also noise)
- Let's look at the estimated weights for various M in the case of fewer examples
- The weights are becoming huge to compensate for the noise
- One way of dealing with this is to encourage the weights to be small (this way no input will have too much influence on prediction). This is called regularization

Regularized Least Squares

- Increasing the input features this way can complicate the model considerably

Regularized Least Squares

- Increasing the input features this way can complicate the model considerably
- Goal: select the appropriate model complexity automatically

Regularized Least Squares

- Increasing the input features this way can complicate the model considerably
- Goal: select the appropriate model complexity automatically
- Standard approach: regularization

$$\tilde{\ell}(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2 + \alpha \mathbf{w}^T \mathbf{w}$$

Regularized Least Squares

- Increasing the input features this way can complicate the model considerably
- Goal: select the appropriate model complexity automatically
- Standard approach: regularization

$$\tilde{\ell}(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2 + \alpha \mathbf{w}^T \mathbf{w}$$

- Intuition: Since we are minimizing the loss, the second term will encourage smaller values in $\mathbf{w}$

Regularized Least Squares

- Increasing the input features this way can complicate the model considerably
- Goal: select the appropriate model complexity automatically
- Standard approach: regularization

$$\tilde{\ell}(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2 + \alpha \mathbf{w}^T \mathbf{w}$$

- Intuition: Since we are minimizing the loss, the second term will encourage smaller values in $\mathbf{w}$
- When we use the penalty on the squared weights we have ridge regression in statistics

Regularized Least Squares

- Increasing the input features this way can complicate the model considerably
- Goal: select the appropriate model complexity automatically
- Standard approach: regularization

$$\tilde{\ell}(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2 + \alpha \mathbf{w}^T \mathbf{w}$$

- Intuition: Since we are minimizing the loss, the second term will encourage smaller values in $\mathbf{w}$
- When we use the penalty on the squared weights we have ridge regression in statistics
- Leads to a modified update rule for gradient descent:

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda [\sum_{n=1}^{N} (t^{(n)} - y(x^{(n)})) \begin{bmatrix} 1 \\ x^{(n)} \end{bmatrix} - \alpha \mathbf{w}]$$

Regularized Least Squares

- Increasing the input features this way can complicate the model considerably
- Goal: select the appropriate model complexity automatically
- Standard approach: regularization

$$\tilde{\ell}(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2 + \alpha \mathbf{w}^T \mathbf{w}$$

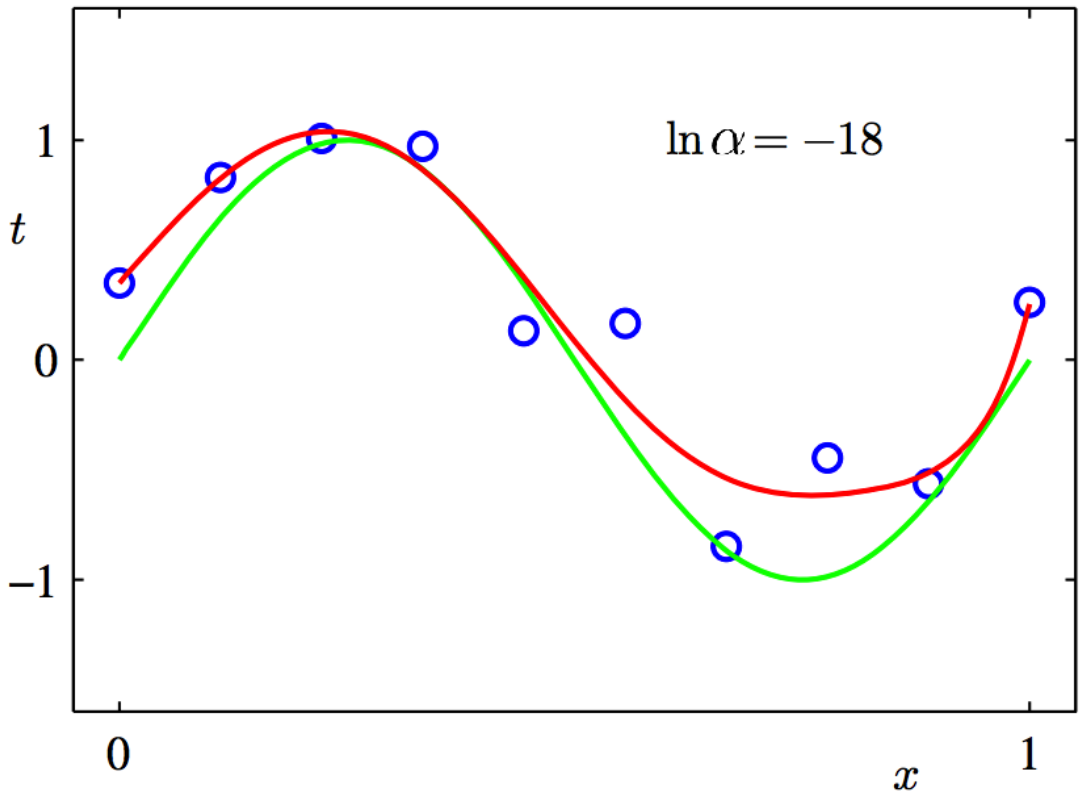
- Intuition: Since we are minimizing the loss, the second term will encourage smaller values in $\mathbf{w}$
- When we use the penalty on the squared weights we have ridge regression in statistics
- Leads to a modified update rule for gradient descent:

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda [\sum_{n=1}^{N} (t^{(n)} - y(x^{(n)})) \begin{bmatrix} 1 \\ x^{(n)} \end{bmatrix} - \alpha \mathbf{w}]$$

- Also has an analytical solution: $\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \alpha \mathbf{I})^{-1} \mathbf{X}^T \mathbf{t}$ (verify!)

Regularized least squares

- Better generalization
- Choose α carefully



1-D regression illustrates key concepts

- Data fits – is linear model best (model selection)?

1-D regression illustrates key concepts

- Data fits – is linear model best (model selection)?
  - Simple models may not capture all the important variations (signal) in the data: underfit

1-D regression illustrates key concepts

- Data fits – is linear model best (model selection)?
  - Simple models may not capture all the important variations (signal) in the data: underfit
  - More complex models may overfit the training data (fit not only the signal but also the noise in the data), especially if not enough data to constrain model

1-D regression illustrates key concepts

- Data fits – is linear model best (model selection)?
  - Simple models may not capture all the important variations (signal) in the data: underfit
  - More complex models may overfit the training data (fit not only the signal but also the noise in the data), especially if not enough data to constrain model
- One method of assessing fit: test generalization = model's ability to predict the held out data

Training set vs. Test set

1-D regression illustrates key concepts

- Data fits – is linear model best (model selection)?
  - Simple models may not capture all the important variations (signal) in the data: underfit
  - More complex models may overfit the training data (fit not only the signal but also the noise in the data), especially if not enough data to constrain model
- One method of assessing fit: test generalization = model's ability to predict the held out data
- Optimization is essential: stochastic and batch iterative approaches; analytic when available

Recap: Regression

- Regression: to predict continuous outputs *t*
  - Consider proper features (inputs): *x* (or **x** if vectors)
  - Training examples, many $x(\mathrm{i})$ for which *t(i)* is known (labeled)
  - A model, a function that represents the relationship between *x* and t

$$y = f(x, w)$$

  - A loss or a cost or an objective function, which tells us how well our model approximates the training examples
  - Optimization, a way of finding the parameters w of our model that minimizes the loss function

Cat or dog?

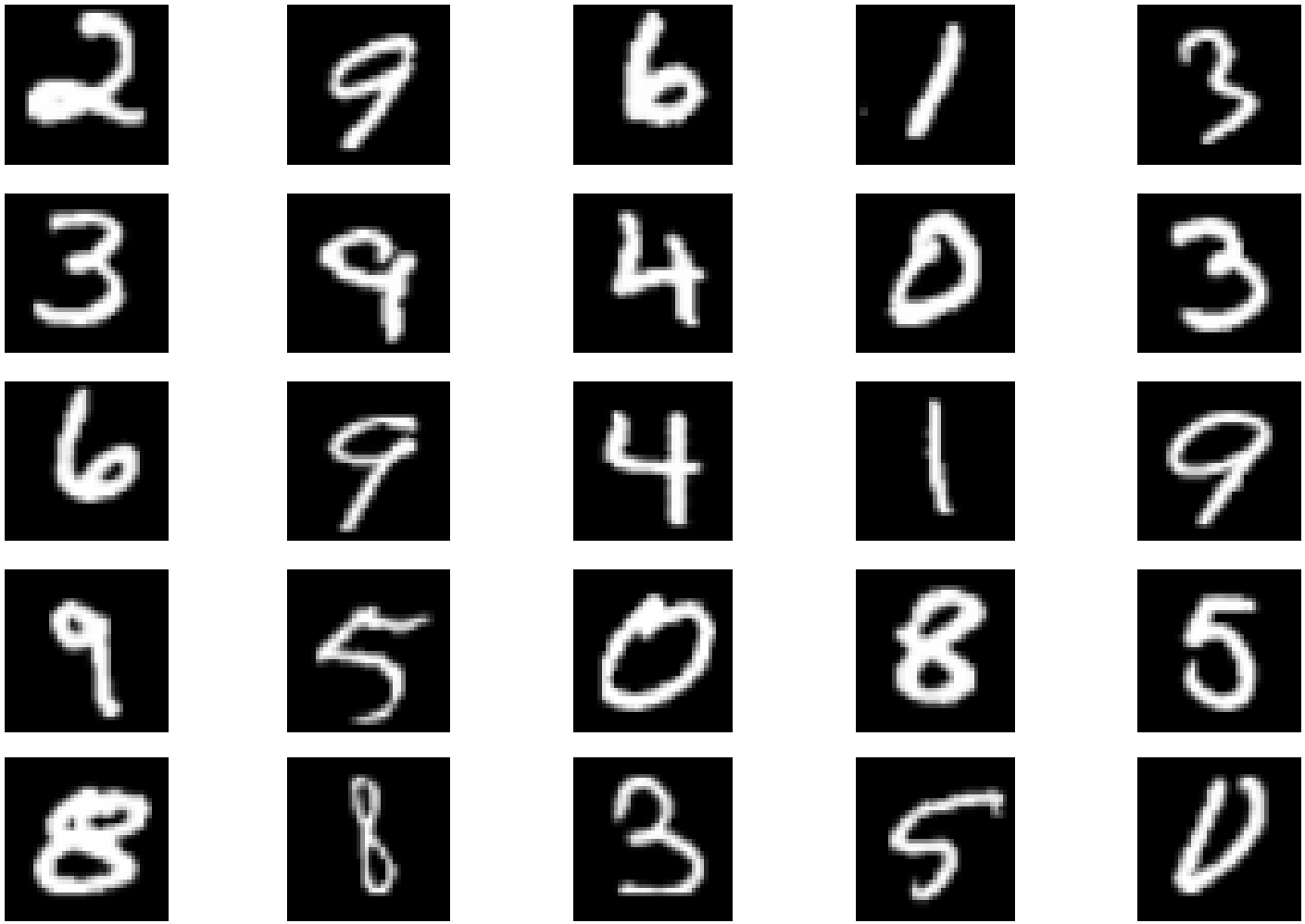
- ☐ Machine learn to classify a cat or dog
- ✓ Offer a large number of pictures with cat or dog labels and train a model
- ✓ Use the trained model to predict where there is a cat or a dog

Is this a dog?

Am I going to pass the exam?

Am I going to pass the exam?

Examples of Problems



What digit is this?
How can I predict this? What are my input features?

Classification

What do all these problems have in common?

Categorical outputs, called labels
(eg, yes/no, dog/cat/person/other)

Assigning each input vector to one of a finite number of labels is called classification

Binary classification: two possible labels (eg, yes/no, 0/1, cat/dog)

Multi-class classification: multiple possible labels

We will first look at binary problems, and discuss multi-class problems later in class

Today

Linear Classification (binary)

Key Concepts:

- Classification as regression
- Decision boundary
- Loss functions
- Metrics to evaluate classification

Classification vs Regression

We are interested in mapping the input $\mathbf{x} \in \mathcal{X}$ to a *label* $t \in \mathcal{Y}$

In regression typically $\mathcal{Y} = \Re$

Now $\mathcal{Y}$ is categorical

Classification as Regression

Can we do this task using what we have learned in previous lectures?

Simple hack: Ignore that the output is categorical!

Suppose we have a binary problem, $t \in \{-1, 1\}$

Assuming the standard model used for (linear) regression

$$y(\mathbf{x}) = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

How can we obtain $\mathbf{w}$?

Use least squares, $\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$. How is $\mathbf{X}$ computed? and $\mathbf{t}$?

Which loss are we minimizing? Does it make sense?

$$\ell_{square}(\mathbf{w}, t) = \frac{1}{N} \sum_{n=1}^{N} (t^{(n)} - \mathbf{w}^T \mathbf{x}^{(n)})^2$$

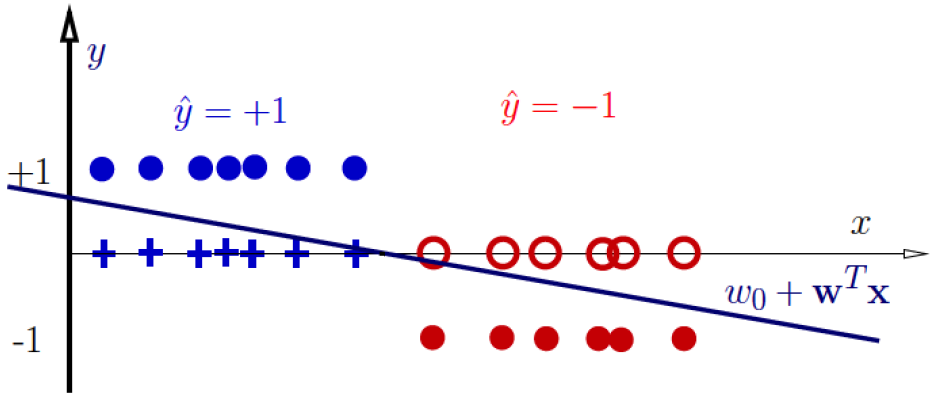
How do I compute a label for a new example? Let's see an example

Classification as Regression

One dimensional example (input x is 1-dim)

The colors indicate labels (a blue plus denotes that $t^{(i)}$ is from the first class, red circle that $t^{(i)}$ is from the second class)

Decision Rules



Our classifier has the form

$$f(\mathbf{x}, \mathbf{w}) = w_o + \mathbf{w}^T\mathbf{x}$$

A reasonable decision rule is

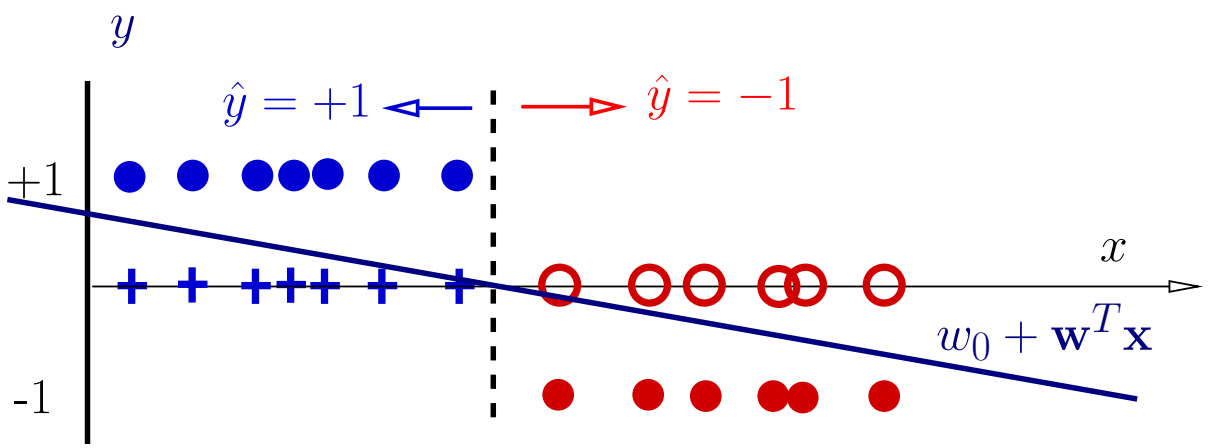
$$y = \begin{cases} 1 & \text{if } f(\mathbf{x}, \mathbf{w}) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

How can I mathematically write this rule?

$$y(\mathbf{x}) = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

What does this function look like?

Decision Rules



How can I mathematically write this rule?

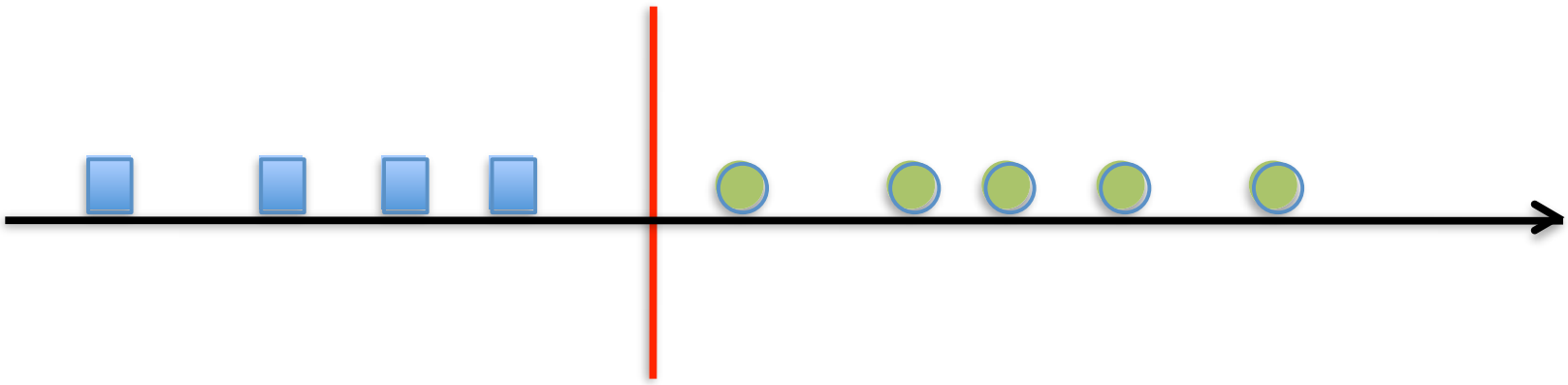
$$y(\mathbf{x}) = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

This specifies a linear classifier: it has a linear boundary (hyperplane)

$$w_0 + \mathbf{w}^T\mathbf{x} = 0$$

which separates the space into two "half-spaces"

Example in 1D



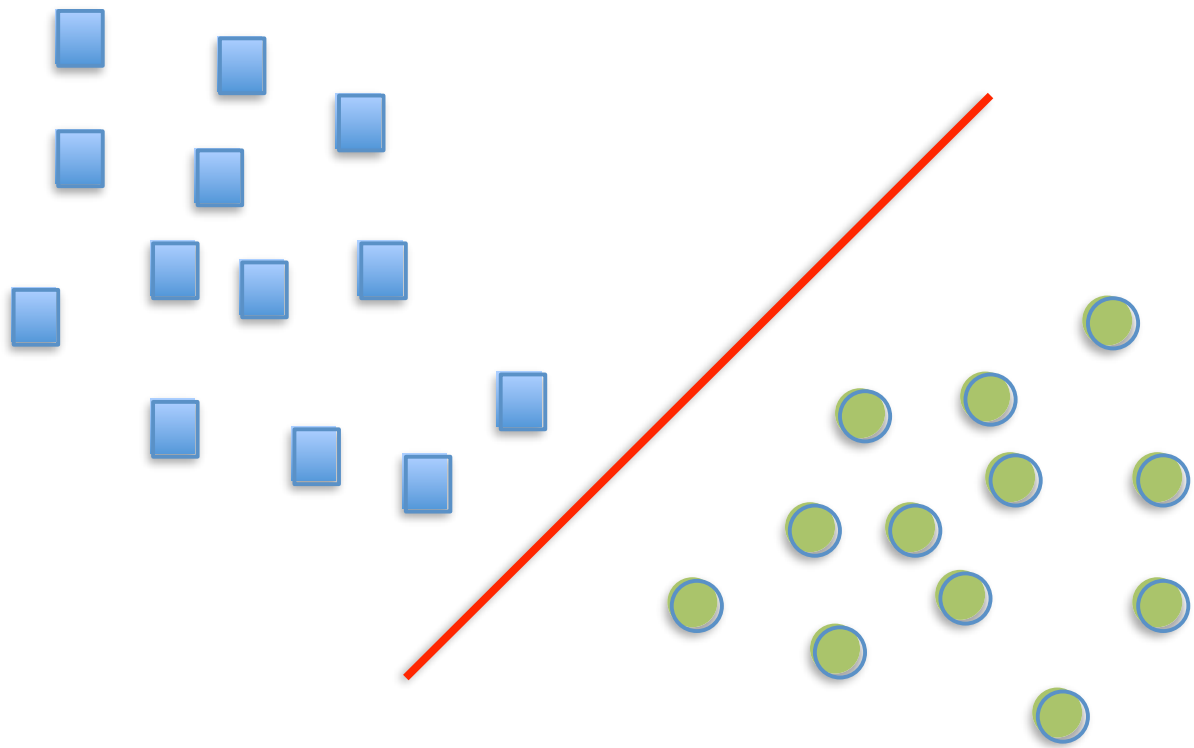
The linear classifier has a linear boundary (hyperplane)

$$w_0 + \mathbf{w}^T \mathbf{x} = 0$$

which separates the space into two "half-spaces"

In 1D this is simply a threshold

Example in 2D



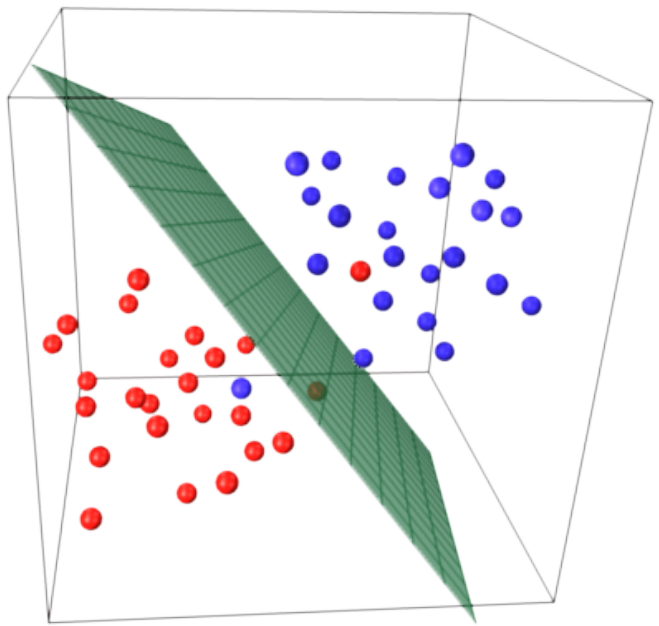
The linear classifier has a linear boundary (hyperplane)

$$w_0 + \mathbf{w}^T \mathbf{x} = 0$$

which separates the space into two "half-spaces"

In 2D this is a line

Example in 3D



The linear classifier has a linear boundary (hyperplane)

$$w_0 + \mathbf{w}^T \mathbf{x} = 0$$

which separates the space into two "half-spaces"

In 3D this is a plane

What about higher-dimensional spaces?

Geometry

$\mathbf{w}^T\mathbf{x} = 0$ a line passing though the origin and orthogonal to $\mathbf{w}$
$\mathbf{w}^T\mathbf{x} + w_0 = 0$ shifts it by w_0

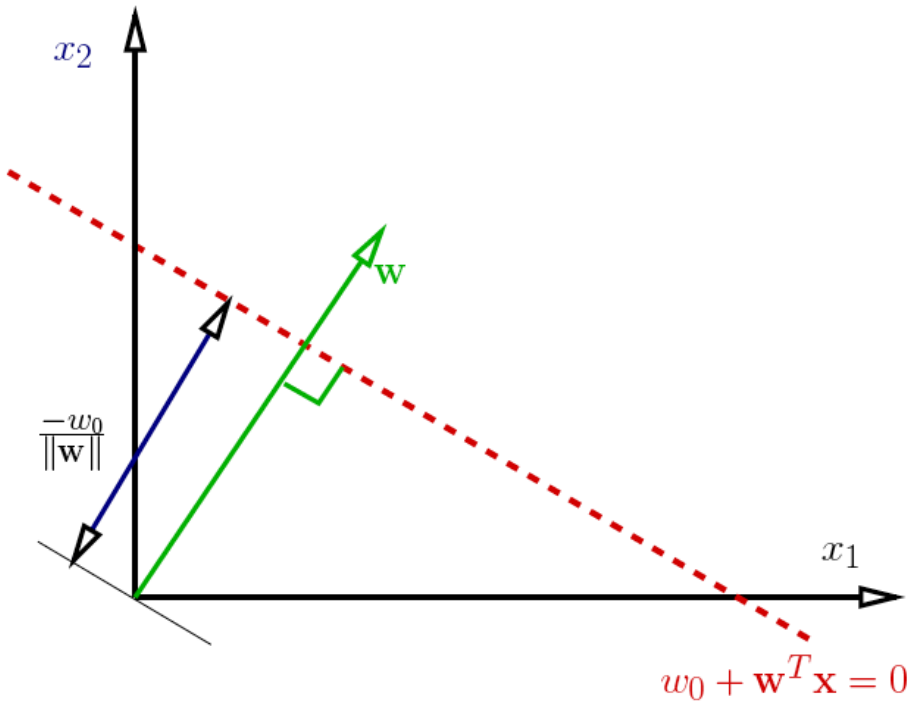


Figure from G. Shakhnarovich

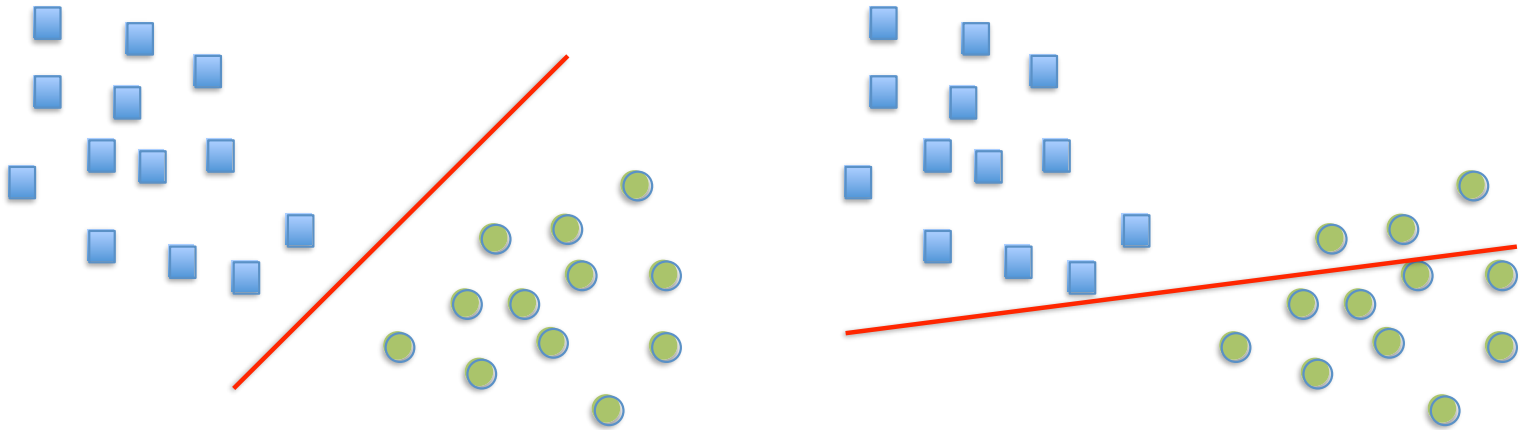
Learning Linear Classifiers

Learning consists in estimating a "good" decision boundary

We need to find $\mathbf{w}$ (direction) and w_0 (location) of the boundary

What does "good" mean?

Is this boundary good?



We need a criteria that tell us how to select the parameters

Do you know any?

Loss functions

Classifying using a linear decision boundary reduces the data dimension to 1

$$y(\mathbf{x}) = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

What is the cost of being wrong?

Loss function: $L(y, t)$ is the loss incurred for predicting y when correct answer is t

Loss functions

A possible loss to minimize is the zero/one loss

$$L(y(\mathbf{x}), t) = \begin{cases} 0 & \text{if } y(\mathbf{x}) = t \\ 1 & \text{if } y(\mathbf{x}) \neq t \end{cases}$$

Is this minimization easy to do? Why?

Other Loss functions

Zero/one loss for a classifier

$$L_{0-1}(y(\mathbf{x}), t) = \begin{cases} 0 & \text{if } y(\mathbf{x}) = t \\ 1 & \text{if } y(\mathbf{x}) \neq t \end{cases}$$

Asymmetric Binary Loss

$$L_{ABL}(y(\mathbf{x}), t) = \begin{cases} \alpha & \text{if } y(\mathbf{x}) = 1 \wedge t = 0 \\ \beta & \text{if } y(\mathbf{x}) = 0 \wedge t = 1 \\ 0 & \text{if } y(\mathbf{x}) = t \end{cases}$$

Squared (quadratic) loss

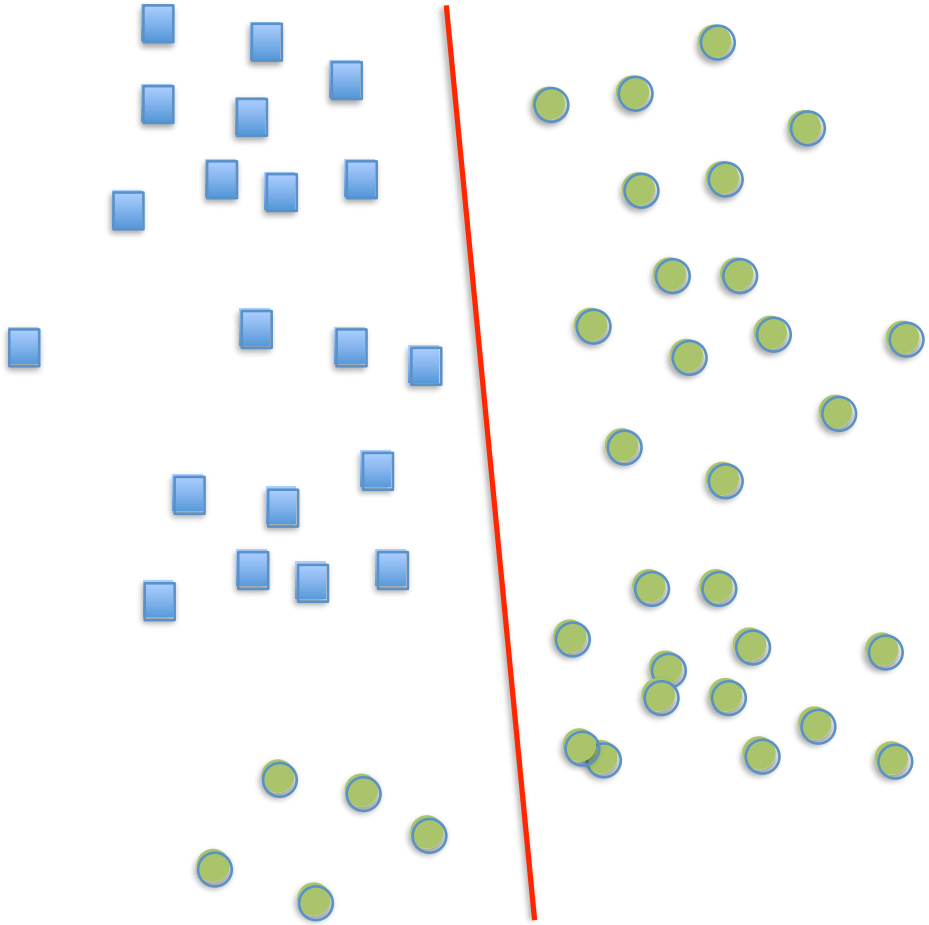
$$L_{squared}(y(\mathbf{x}), t) = (t - y(\mathbf{x}))^2$$

Absolute Error

$$L_{absolute}(y(\mathbf{x}), t) = |t - y(\mathbf{x})|$$

Can we always separate the classes?

If we can separate the classes, the problem is linearly separable



Can we always separate the classes?

Causes of non perfect separation:

- Model is too simple
- Noise in the inputs (i.e., data attributes)
- Simple features that do not account for all variations
- Errors in data targets (mis-labelings)

Should we make the model complex enough to have perfect separation in the training data?

Metrics

How to evaluate how good my classifier is? How is it doing on dog vs no-dog?

TP (True Positive) FP (False Positive) FN (False Negative)

Metrics

How to evaluate how good my classifier is?

Recall: is the fraction of relevant instances that are retrieved

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{all groundtruth instances}}$$

Precision: is the fraction of retrieved instances that are relevant

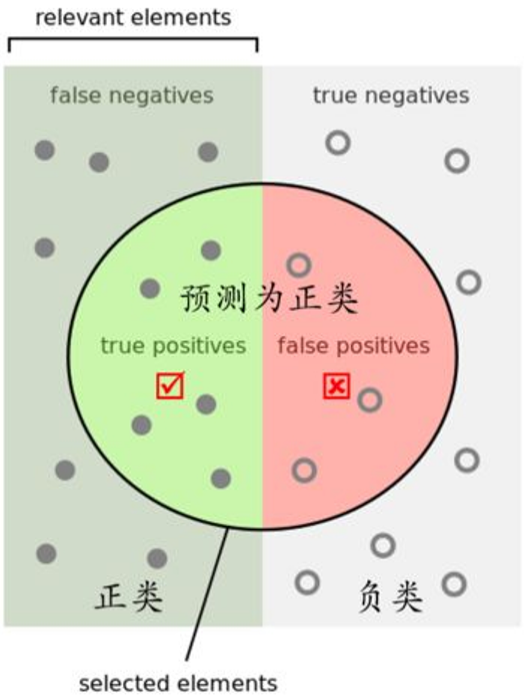
$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{all predicted}}$$

F1 score: harmonic mean of precision and recall

$$F1 = 2\frac{P \cdot R}{P + R}$$

More on Metrics

How to evaluate how good my classifier is?



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{selected elements}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{relevant elements}}$$

TP: positive examples classified as positive	FP: negative examples classified as positive
FN: positive examples cllasified as negative	TN:negative examples classified as negative

Metrics vs Loss

Metrics on a dataset is what we care about (performance)

We typically cannot directly optimize for the metrics

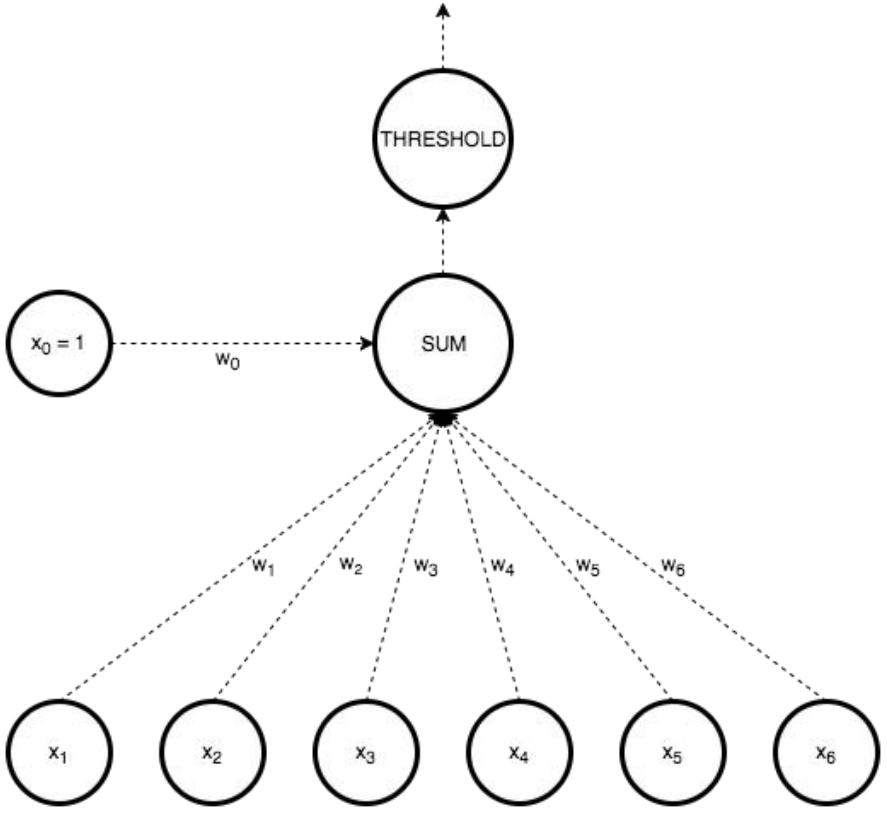
Our loss function should reflect the problem we are solving. We then hope it will yield models that will do well on our dataset

Linear Classifiers

- Linear classifiers classify data into labels based on a linear combination of input features. Therefore, these classifiers separate data using a line or plane or a hyperplane (a plane in more than 2 dimensions).

- Perceptron

In Perceptron, we take weighted linear combination of input features and pass it through a threshold function which outputs 1 or 0.



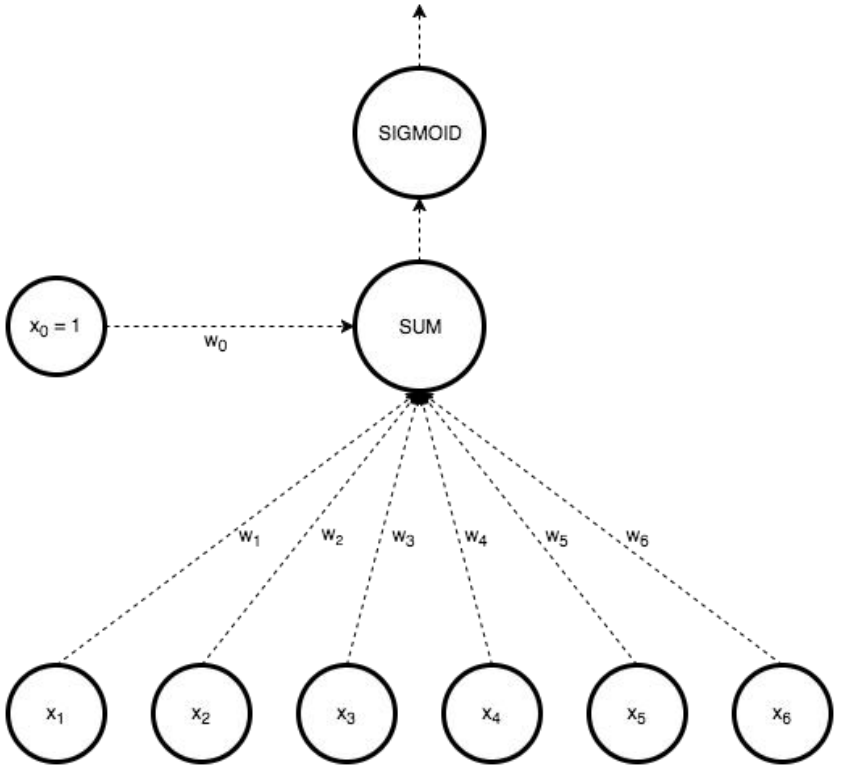
Linear Classifiers

Logistic Regression

- take weighted linear combination of input features and pass it through a sigmoid function, which outputs a number between 1 and 0.

- Unlike perceptron, which just tells us which side of the plane the point lies on, logistic regression gives a probability of a point lying on a particular side of the plane.

- The probability of classification will be very close to 1 or 0 as the point goes far away from the plane. The probability of classification of points very close to the plane is close to 0.5



SVM

There can be multiple hyperplanes that separate linearly separable data. SVM calculates the optimal separating hyperplane using concepts of geometry

Summary of Perceptron

- Algorithm
  - Let the input feature vector be

$$x = [x_1, x_2, \ldots, x_n]^T, \quad \bar{x} = [1, x_1, x_2, \ldots, x_n]^T,$$

  - Let the weight vector be

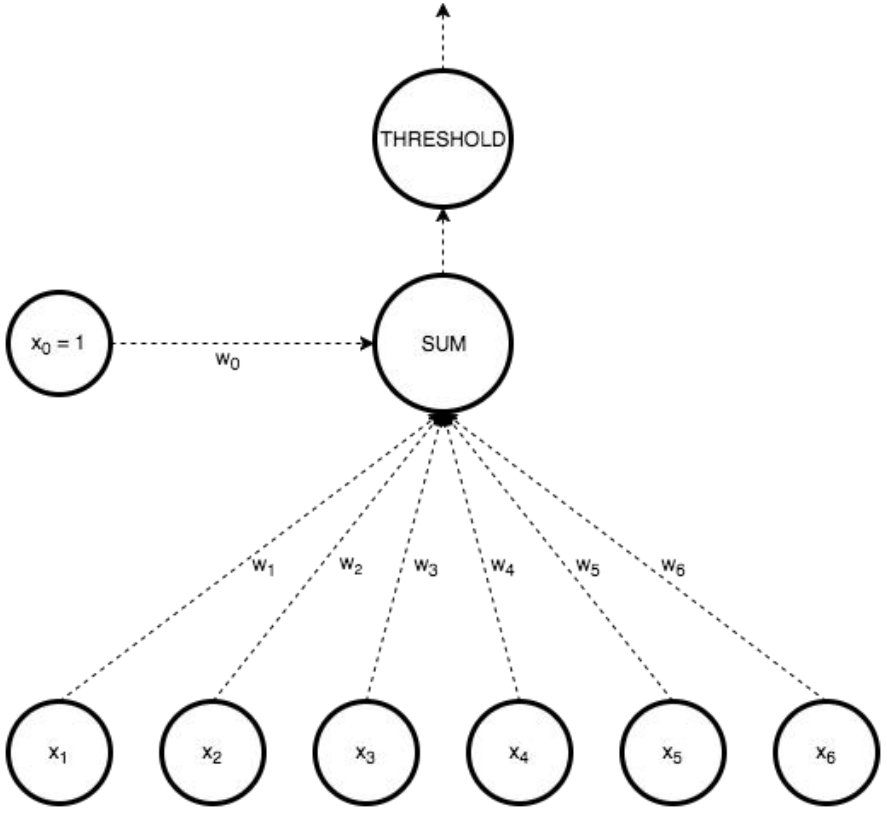
$$w = [w_0, w_1, w_2, \ldots, x_n]^T$$

  - Perceptron
  - ✓ Transfer function

$$f(x, w) = \sum_{i=0}^{n} w_i x_i = w^T \bar{x}$$

  - ✓ Activation function

$$output = \begin{cases} 1, & f(x, w) \geq 0 \\ 0, & f(x, w) < 0 \end{cases}$$



Summary of Perceptron

Learning Weights

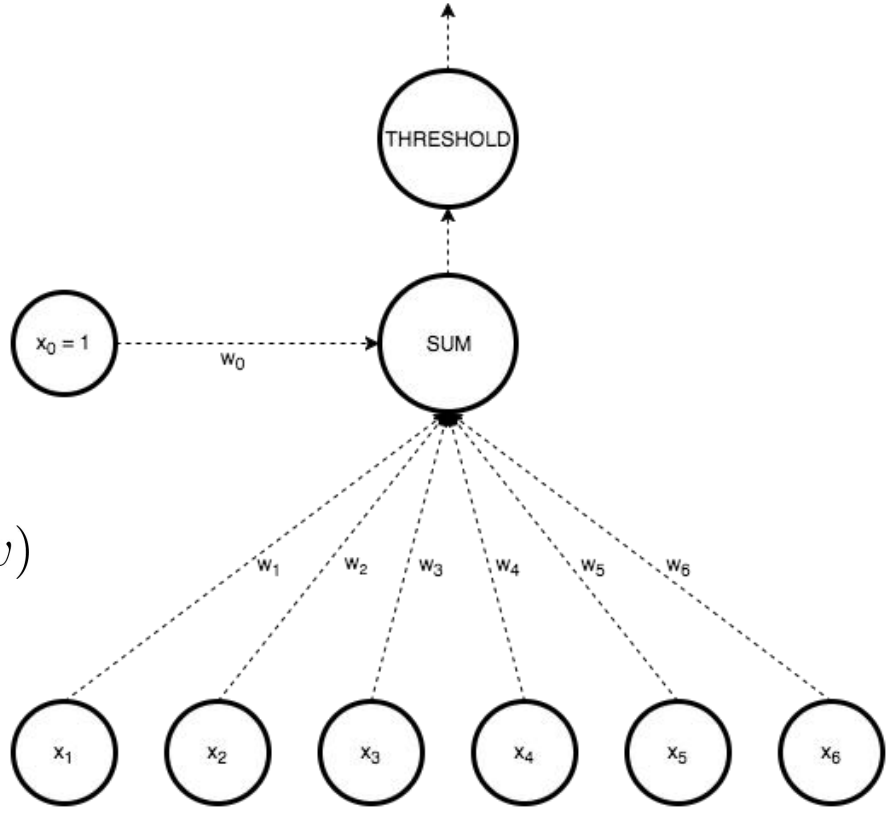
- Perceptron is an example of online learning algorithm

Until convergence:

1. Randomly initialize weights w
2. Calculate $f(x, w)$ for first sample $x^{(1)}$
3. Suppose the actual label of sample is $t^{(1)}$

(which can be 1 or 0), calculate the loss $L(w)$

4. Update weights by gradient descent
5. Move to next sample



Intuition of Perceptron

	Friend 1's opinion	Friend 2's opinion	Your Decision
week 1	Good	Good	You watch the movie and like it
week 2	Good	Bad	You watch the movie and do not like it
week 3	Bad	Good	Would you watch the movie

- Suppose your opinions about movies do not match with friend 1 multiple times, you are unlikely to ever consider his opinion to decide whether to watch a movie or not

- Thus every week you update the weights that you have assigned to each of your friend's opinion. The Perceptron algorithm works in a similar fashion

Linear regression vs. linear classification

- Linear regression
  - Linear regression model
  - MSE (loss function)
  - Gradient descent optimization (Batch update and stochastic update)
  - From single input to multiple inputs
  - From linear to polynomial model
  - Regularization (why and how?)
- Linear classification
  - Classification vs. regression
  - Decision boundary for binary classification
  - Loss function
  - Metric for evaluation (Recall, Precision, and F1 score)
  - Binary linear classification is essentially a perceptron

Homework Assignment

- Write a program of a perceptron to learn an OR function.

OR Function Using A Perceptron

x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$