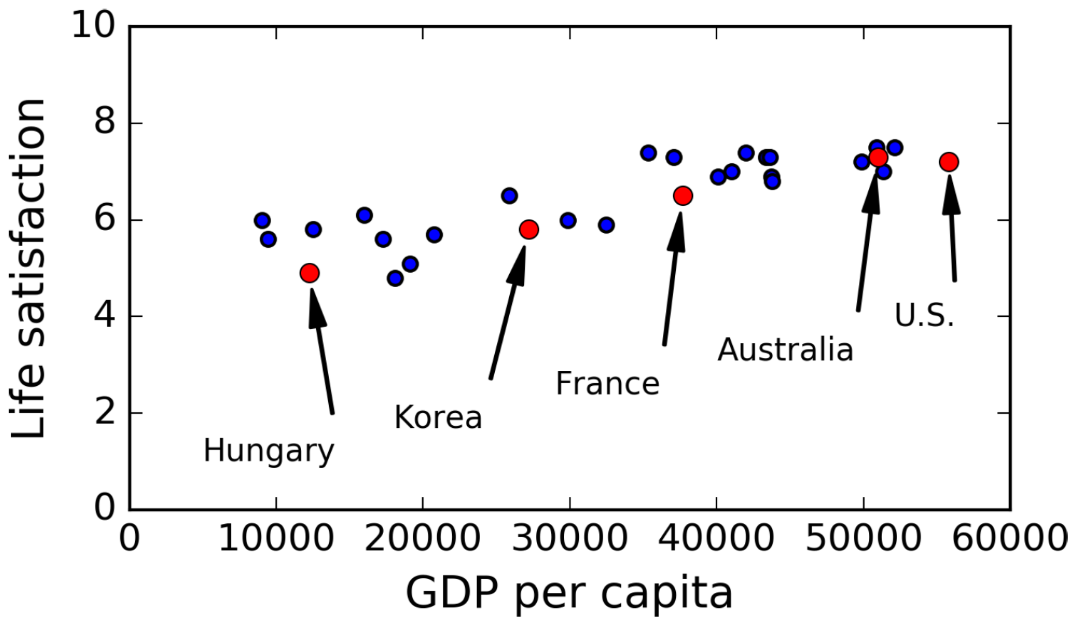
Regression

COEN140

Santa Clara University

Regression Problem

- Purpose: to predict values from some inputs
- Example: predict the life satisfaction by the GDP per capita
- Objective: find a relation between the input and the output

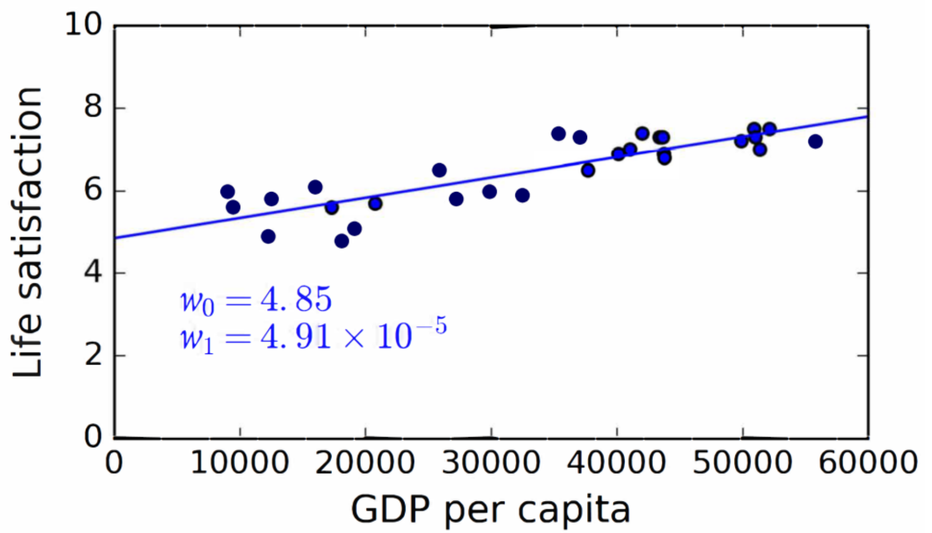


Regression Problem

- Define a model
  - A simple model: the input-output relation is a straight line

$$life\_satisfaction = w_0 + w_1 \times GDP\_per\_capita$$

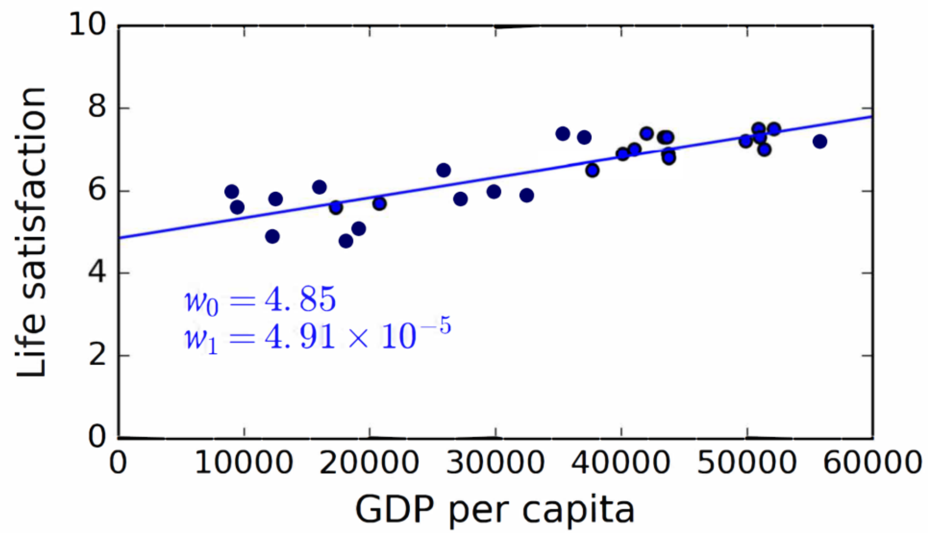
  - Model parameters: w_0, w_1



Regression Problem

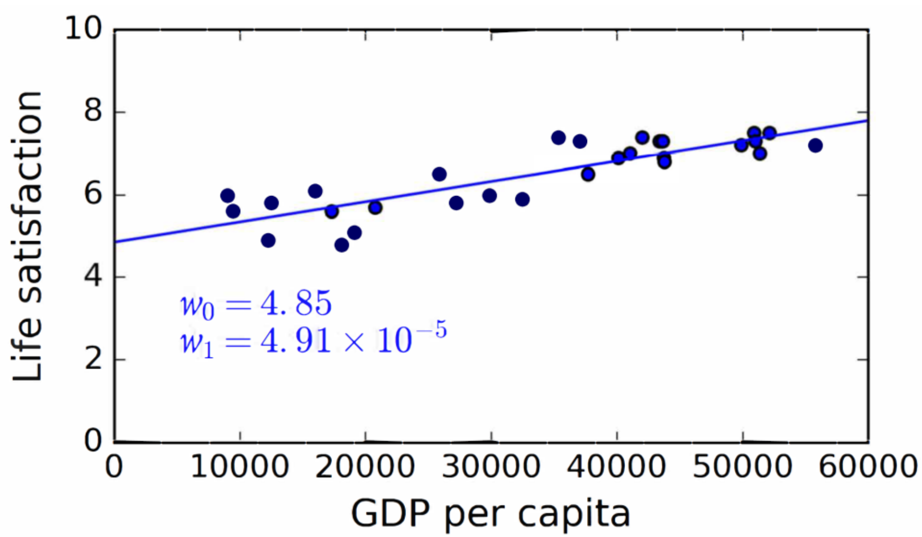
- Goal: find w_0, w_1
- Why? If you are given a new GDP_per_capita, you can predict the corresponding $life\_satisfaction$

$$life\_satisfaction = w_0 + w_1 \times GDP\_per\_capita$$



Regression Problem

- How to find w_0, w_1 ?
- N training samples: $(x_n, t_n), \quad n = 1, \ldots, N$
  - x_n is the input, t_n is the target/true value
  - Prediction model: $y_n = w_0 + w_1 x_n$
  - y_n is the predicted output



Regression Problem

- Prediction Error

$$y_n - t_n = w_0 + w_1 x_n - t_n$$

- Error Function: sum of the squared error between the predicted value $y_n(x_n, w_0, w_1)$ and the true target value t_n

$$E(w_0, w_1) = \frac{1}{2} \sum_{n=1}^{N} \{w_0 + w_1 x_n - t_n\}^2$$

Regression Problem

- Minimize the error function:

$$E(w_0, w_1) = \frac{1}{2}\sum_{n=1}^{N}\{w_0 + w_1 x_n - t_n\}^2$$

  - Take the partial derivative of $E(w_0, w_1)$ with respect to (w.r.t.) w_0, and let it be 0

$$\frac{\partial E(w_0, w_1)}{\partial w_0} = 0$$

$$\frac{1}{2}\sum_{n=1}^{N} 2(w_0 + w_1 x_n - t_n) = 0$$

$$\sum_{n=1}^{N} (w_0 + w_1 x_n - t_n) = 0$$

$$N w_0 + w_1 \sum_{n=1}^{N} x_n = \sum_{n=1}^{N} t_n \qquad (1)$$

Regression Problem

- Minimize the error function:

$$E(w_0, w_1) = \frac{1}{2}\sum_{n=1}^{N}\{w_0 + w_1 x_n - t_n\}^2$$

  - Take the partial derivative of $E(w_0, w_1)$ w.r.t. w_1, and let it be 0

$$\frac{\partial E(w_0, w_1)}{\partial w_1} = 0$$

$$\frac{1}{2}\sum_{n=1}^{N} 2(w_0 + w_1 x_n - t_n)x_n = 0$$

$$\sum_{n=1}^{N} (w_0 + w_1 x_n - t_n)x_n = 0$$

$$w_0 \sum_{n=1}^{N} x_n + w_1 \sum_{n=1}^{N} x_n^2 = \sum_{n=1}^{N} t_n x_n \qquad (2)$$

Regression Problem

- $$N w_0 + w_1 \sum_{n=1}^{N} x_n = \sum_{n=1}^{N} t_n \qquad (1)$$

- $$w_0 \sum_{n=1}^{N} x_n + w_1 \sum_{n=1}^{N} x_n^2 = \sum_{n=1}^{N} t_n x_n \qquad (2)$$

- Solution

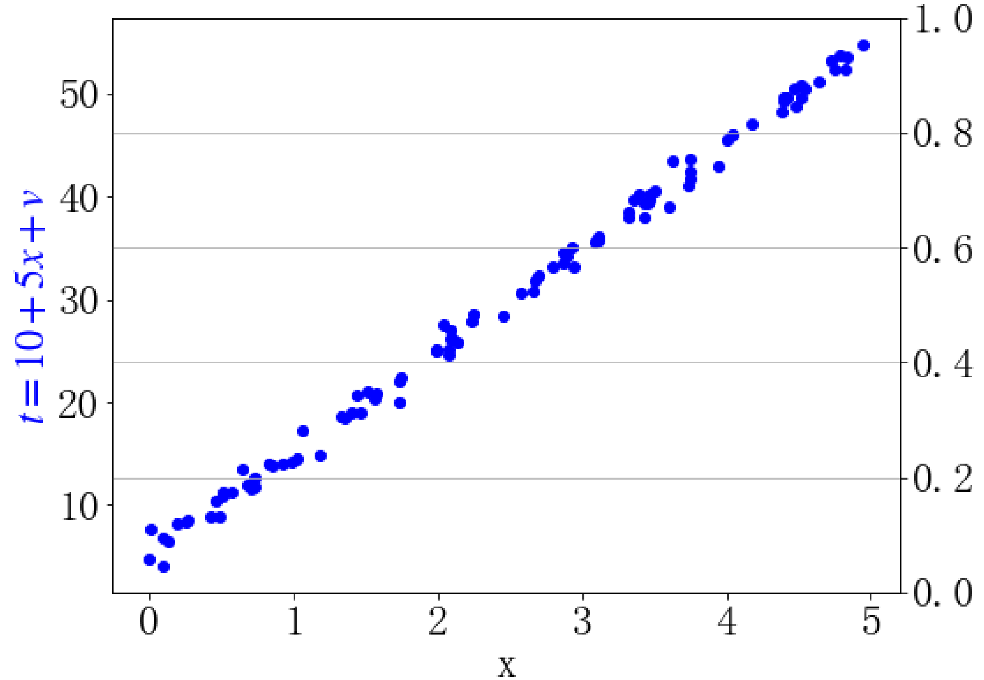
$$w_0 = \frac{\left(\sum_{n=1}^{N} x_n\right) \times \left(\sum_{n=1}^{N} t_n x_n\right) - \left(\sum_{n=1}^{N} t_n\right) \times \left(\sum_{n=1}^{N} x_n^2\right)}{\left(\sum_{n=1}^{N} x_n\right)^2 - N\left(\sum_{n=1}^{N} x_n^2\right)}$$

$$w_1 = \frac{\left(\sum_{n=1}^{N} t_n\right) - N w_0}{\sum_{n=1}^{N} x_n}$$

Example

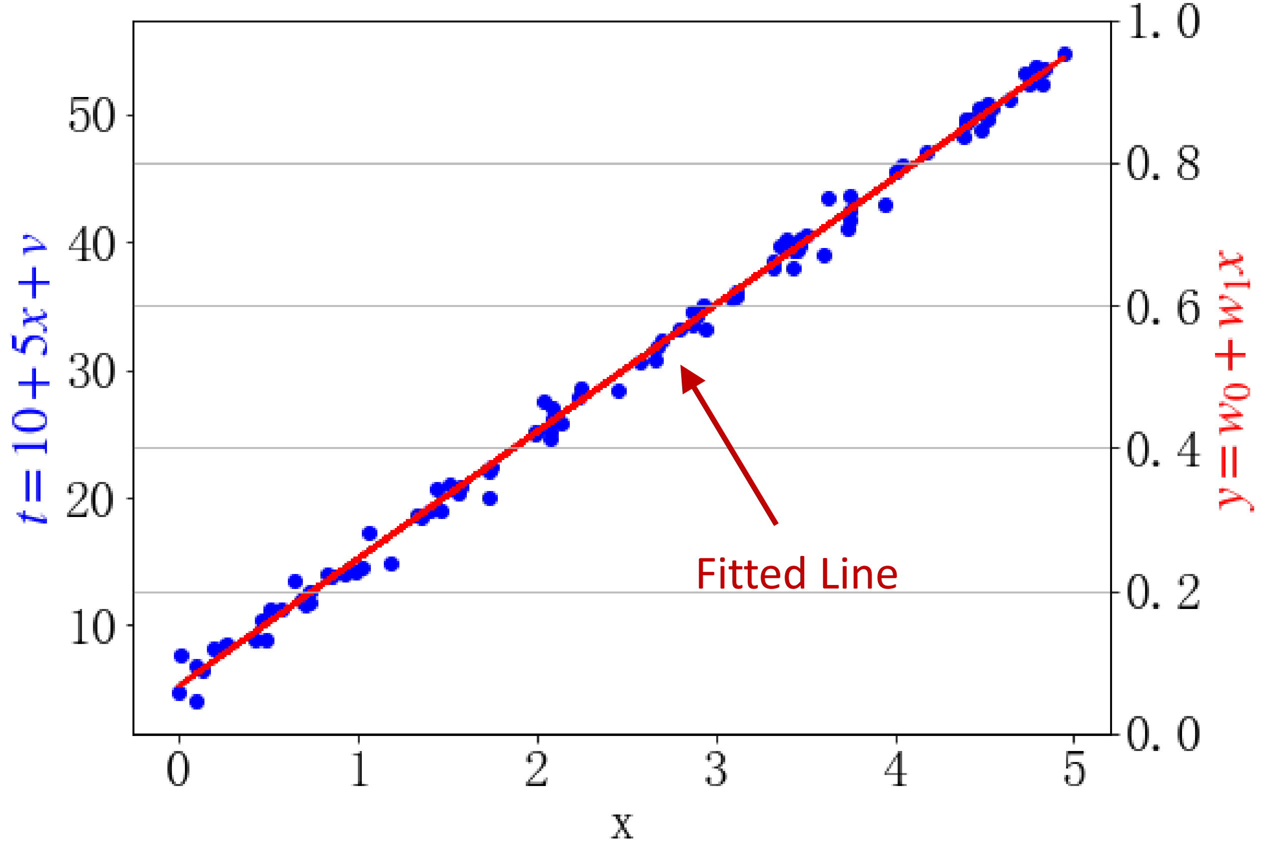
- Ground-truths: $t = 10 + 5x + v$
- $v \sim \mathcal{N}(0,1)$

  Normal distribution with mean 0 and variance 1
- Collect $N = 100$ training samples
- (x_n, t_n),
- $n = 1, 2, \dots, 100$



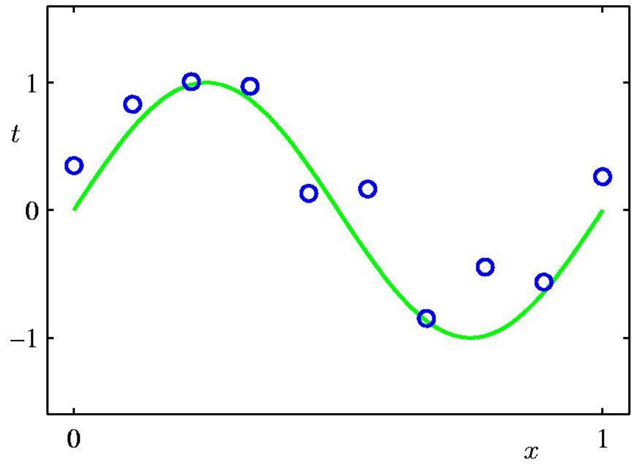
Example

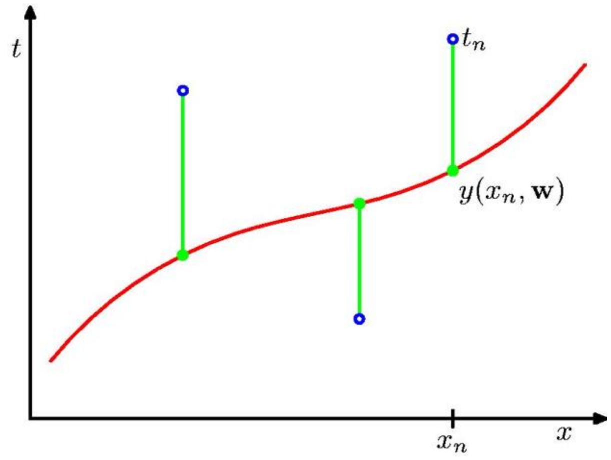
- Use linear regression to find a model:
- $y = w_0 + w_1 x$



Polynomial Curve Fitting

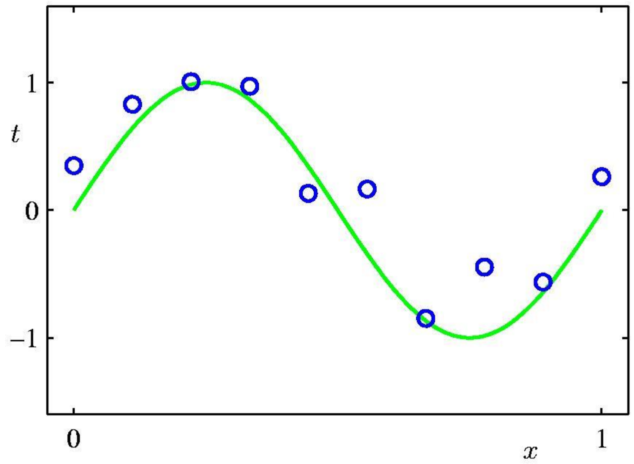
- Real-valued input: x
- True function: $\sin(2\pi x)$
- Observations
  - $t = \sin(2\pi x)$+Gaussian Noise
- Training set: N samples
  - $(x_n, t_n), \;\; n = 1, \ldots, N$





Polynomial Curve Fitting

- We are given $N = 10$ data points
- $x_1, x_{2,\ldots,} x_N$
- Observations of the values of t
- $\mathbf{t} = [t_1, t_{2,\ldots,} t_N]^T$
- Objective: predict the target value t for some new input x

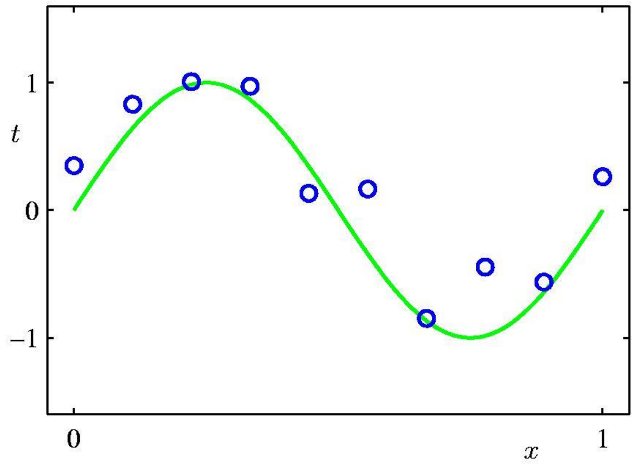


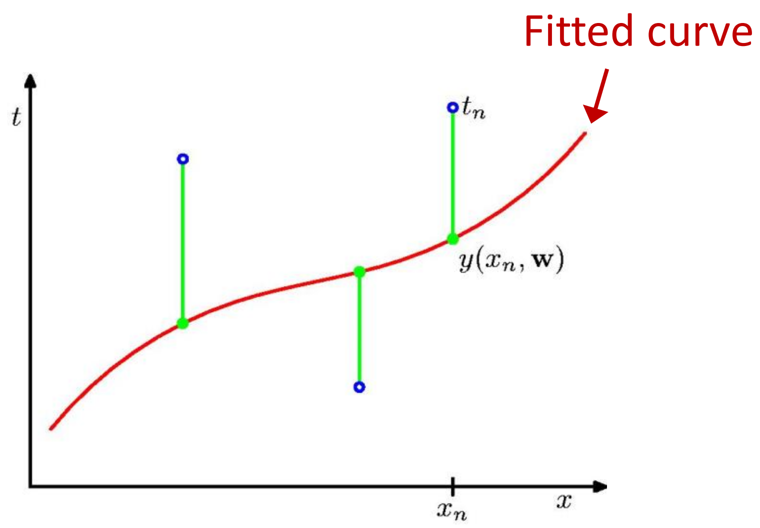
Polynomial Curve Fitting

- Method: Fit the data using a polynomial function

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

  - M: the order of the polynomial
  - x^j: x raised to the power of j
  - $\mathbf{w} = [w_0, w_1, \ldots, w_M]^T$: model parameters





Polynomial Curve Fitting

- A simple version:

$$y(x, \mathbf{w}) = w_0 + w_2 x^2$$

  - Nonlinear in x

$$\frac{d\,[y(x)]}{dx} = 2w_2 x$$

Not a constant!

Polynomial Curve Fitting

- A simple version:

$$y(x, \mathbf{w}) = w_0 + w_2 x^2$$

  - Linear in $\mathbf{w}$

$$\frac{d\,[y(\mathbf{w})]}{dw_0} = 1, \qquad \frac{d\,[y(\mathbf{w})]}{dw_2} = x^2$$

    They are constants!

- Linear Regression: the model $y(x, \mathbf{w})$ is linear in the model parameter $\mathbf{w}$

Polynomial Curve Fitting

- How do we find $\mathbf{w}$?

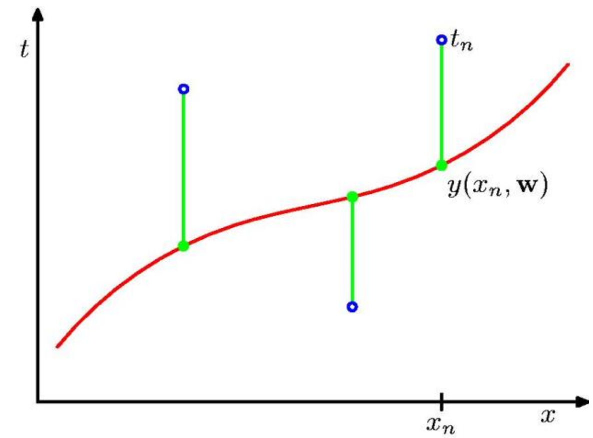
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

$$\mathbf{w} = [w_0, w_1, \ldots, w_M]^T$$

The n-th data sample $\mathbf{x}_n = \begin{bmatrix} 1 \\ x_n^1 \\ \vdots \\ x_n^M \end{bmatrix} = [1, x_n^1, \ldots, x_n^M]^T$

power

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

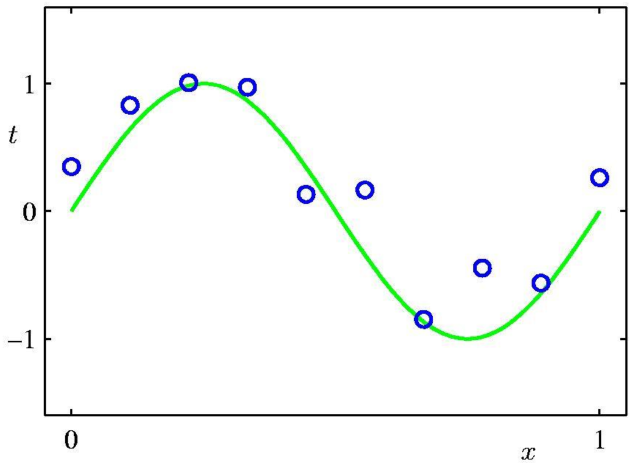


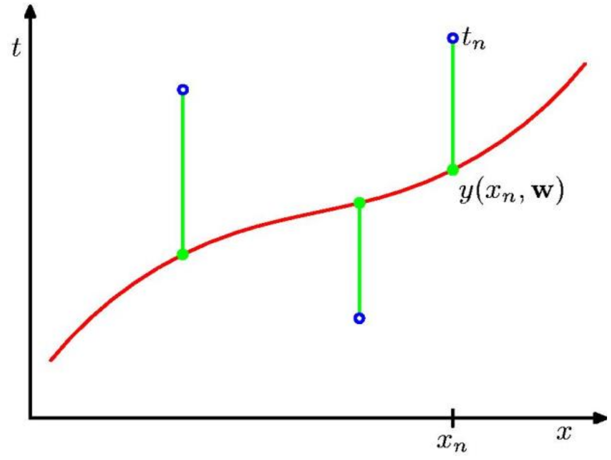
Polynomial Curve Fitting

- Minimize the error function:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\mathbf{x}_n^T\mathbf{w} - t_n\}^2$$





Polynomial Curve Fitting

- The optimization problem:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w})$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{2} \sum_{n=1}^{N} \{\mathbf{x}_n^T \mathbf{w} - t_n \}^2$$

- How to solve for $\mathbf{w}^*$? Take the derivative of $E(\mathbf{w})$ w.r.t $\mathbf{w}$ and set it as the **0** vector

Least Squares Solution

- Solution

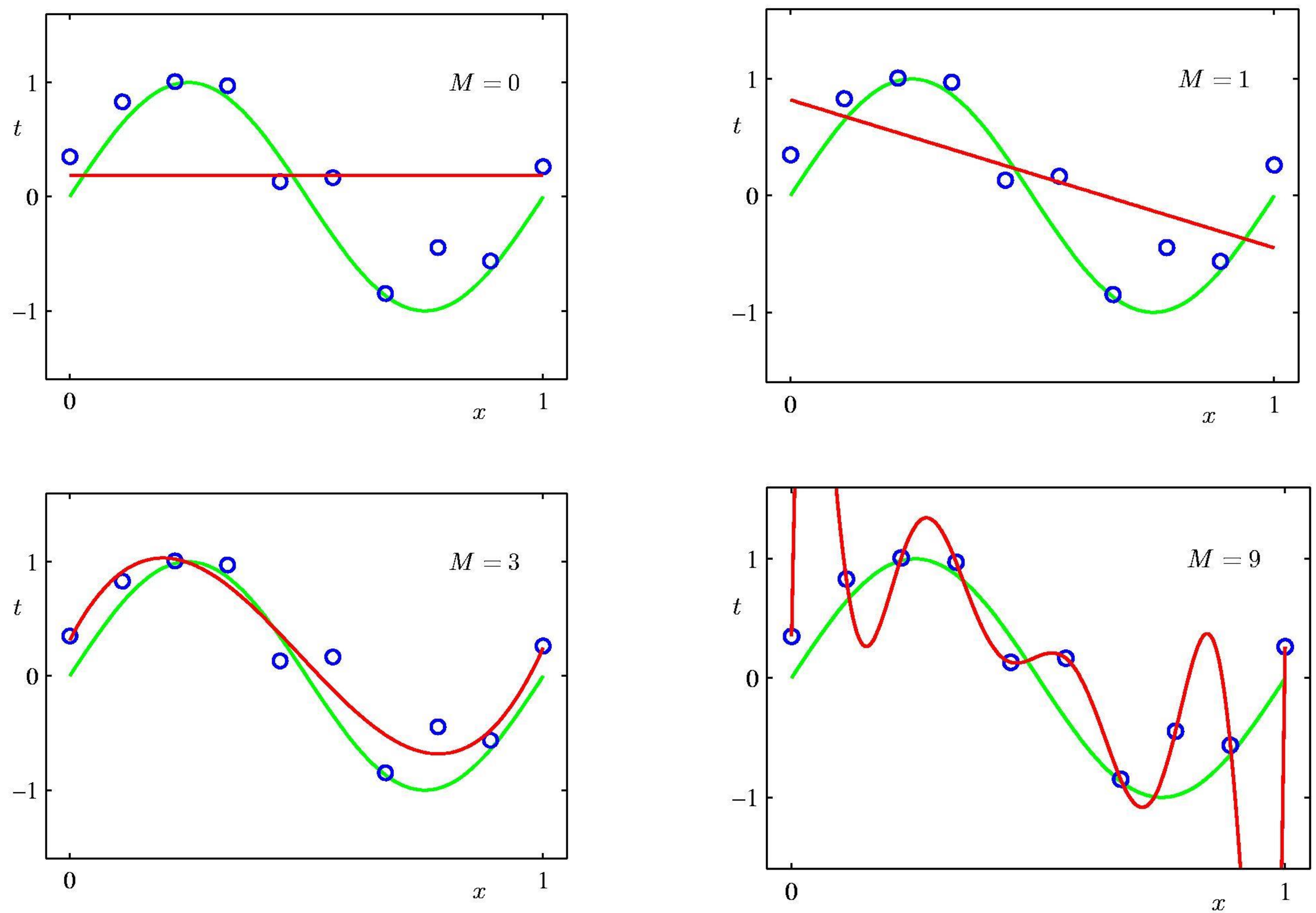
optimal weights → $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$ ← vector of target values

Matrix $\mathbf{X}$ has one input vector per row

- $\mathbf{t} = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix}, \mathbf{x}_n = \begin{bmatrix} 1 \\ x_n^1 \\ \vdots \\ x_n^M \end{bmatrix}, \quad n = 1, 2, \dots, N$ (power)
- $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}, \qquad \mathbf{X}^T = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]$

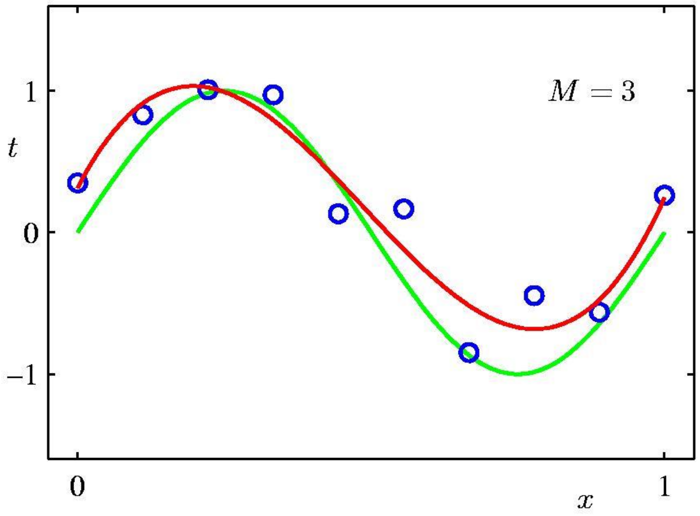
Model Selection: how to choose M?

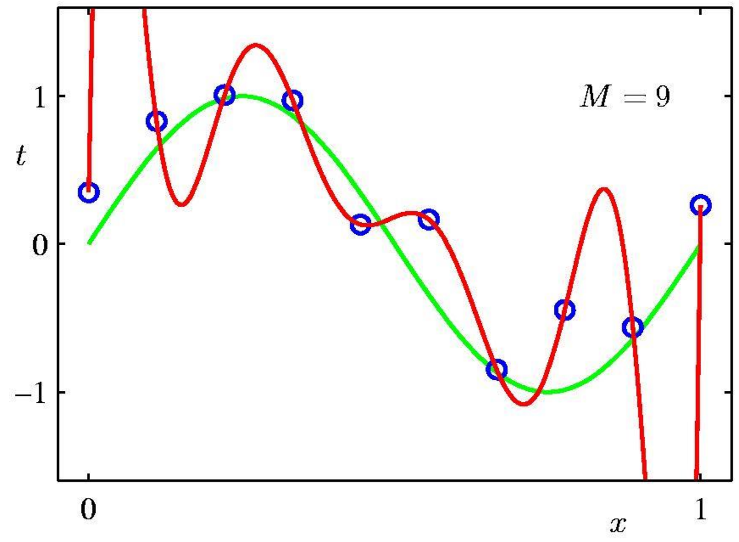
$$y = w_0 + w_1 x$$



Model Selection

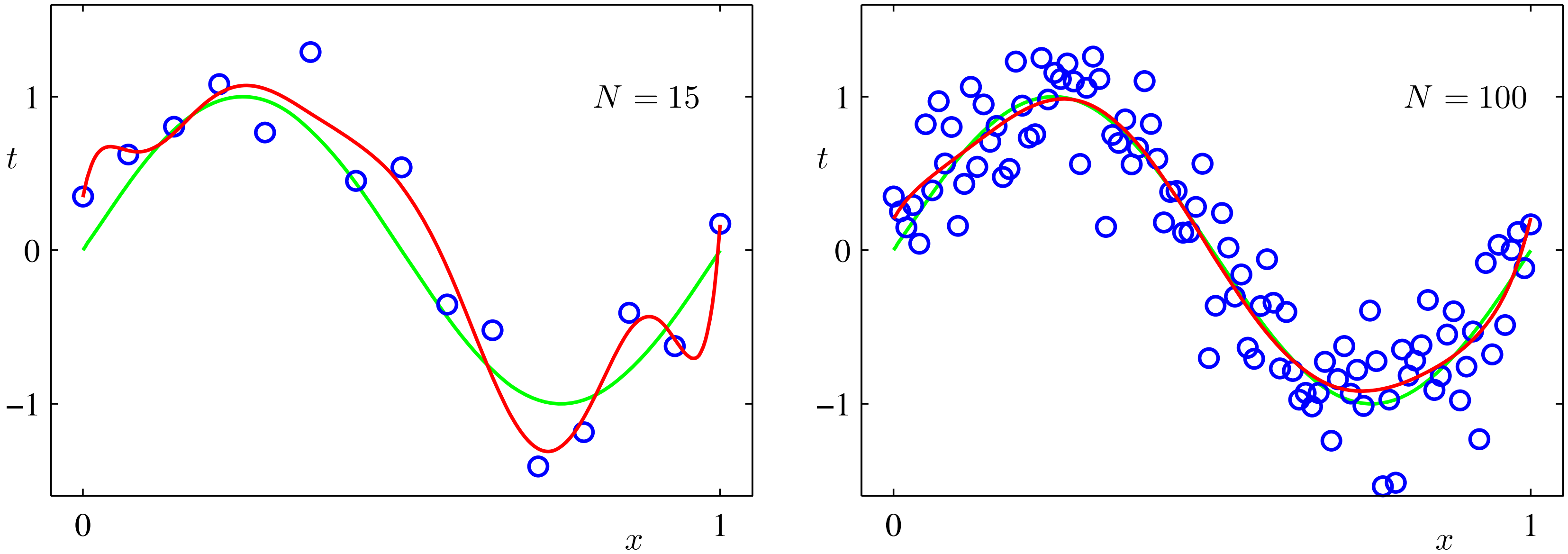
- Rule of Thumb
  - The number of training samples N should be no less than some multiple (say 5 or 10) of the number of adaptive parameters $(M + 1)$
  - Here: $N = 10$
  - $M = 9$: Over-fitting!





Model Selection

- Rule of Thumb
  - $M = 9$
  - Increasing the size of the training set reduces the over-fitting problem.



Magnitudes of the Weights

- The magnitude of the weights increases dramatically as M increases (for $N = 10$)

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	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

Magnitudes of the Weights

- The more flexible polynomials with larger values of M are becoming increasingly tuned to the random noise on the target values.

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	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

Ridge Regression (Regularized Least Squares)

- To address the over-fitting problem
- Add a penalty term to the error function
- Discourage the weights from reaching large magnitudes

penalized error function

regularization parameter

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

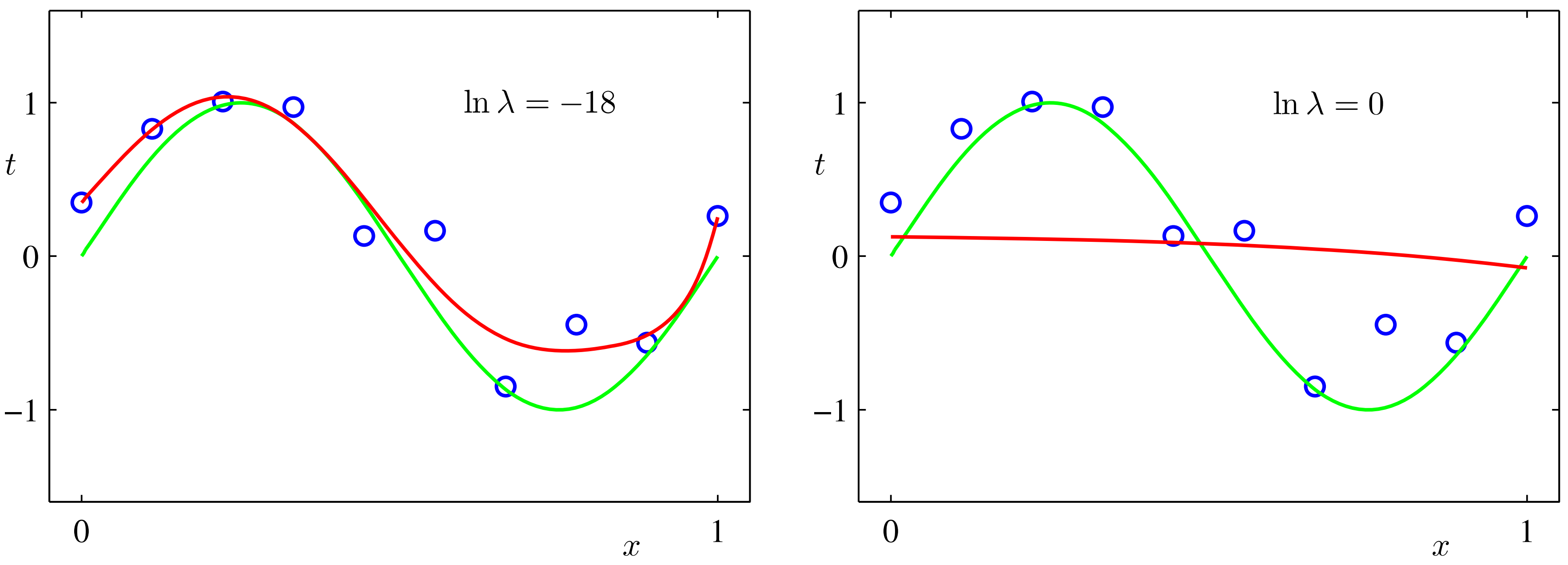
target value

Polynomial Coefficients/Weights

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Polynomial Coefficients

- λ cannot be too large (e.g. $\ln\lambda = 0$)
- λ cannot be too small (e.g. $\ln\lambda = -\infty$)
- The results for $N = 10, M = 9$



Ridge Regression (Regularized Least Squares)

- Setting the derivative w.r.t. $\mathbf{w}$ to $\mathbf{0}$ and solving for $\mathbf{w}$

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

$$\mathbf{w}^* = (\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$$

$(M+1) \times (M+1)$

identity matrix

- Recall $\mathbf{x}_n = \begin{bmatrix} 1 \\ x_n^1 \\ \vdots \\ x_n^M \end{bmatrix}, \quad n = 1, 2, \dots, N, \quad \mathbf{X}^T = [\mathbf{x}_1, \dots, \mathbf{x}_N]$

Housing price prediction

- Data set: California housing dataset
- $M = 8$ attributes/features for the n-th input
  - $x_{n1},\ x_{n2},\ \dots,\ x_{n8}$
  - MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup, Latitude, Longitude
  - $n = 1, \dots, N$

- Total number of data samples: $N = 20{,}640$

- Target output: the price of the house
  - $t_n, n = 1, \dots, N$

General Case

- The input data sample has multiple attributes: $x_{n1}, x_{n2}, \dots, x_{nM}$, we can form the nth data sample as

$$\mathbf{x}_n = [1, x_{n1}, x_{n2}, \dots, x_{nM}]^T$$

This is to let $\mathbf{w}$ have a bias term w_0

- Build a linear regression model

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

- Then find $\mathbf{w}$ in the same way as the polynomial curve fitting example

Performance Evaluation

- How to know if your regression model works well or not?
- Try it on the Test Set!
  - N_{test} test data samples
  - $(\mathbf{x}_n, t_n), \quad n = 1, \ldots, N_{test}$
- Performance Metric?
  - Mean Squared Error (MSE)
  - $$MSE = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \{ y(\mathbf{x}_n, \mathbf{w}) - t_n \}^2$$
    $$= \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \{ \mathbf{w}^T \mathbf{x}_n - t_n \}^2$$
  - The smaller the MSE, the better the performance.

Example: housing price prediction

housing_price_v3.py

```
# -*- coding: utf-8 -*-
"""
% linear regression example
% California house price prediction
"""
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split

housing = fetch_california_housing()
# N = total number of samples; M = number of attributes/features
N,M = housing.data.shape

# Data matrix: currently, each row is one data sample
housing_data_plus_bias = np.c_[np.ones((N,1)), housing.data]
target_val = housing.target.reshape(-1,1) # reshape it as a column vector t

X_train, X_test, t_train, t_test = \
train_test_split(housing_data_plus_bias, target_val, test_size=0.2, random_state=42)


# Ntrain = number of training samples
Ntrain=X_train.shape[0]
# Ntest = number of test samples
Ntest=X_test.shape[0]
```

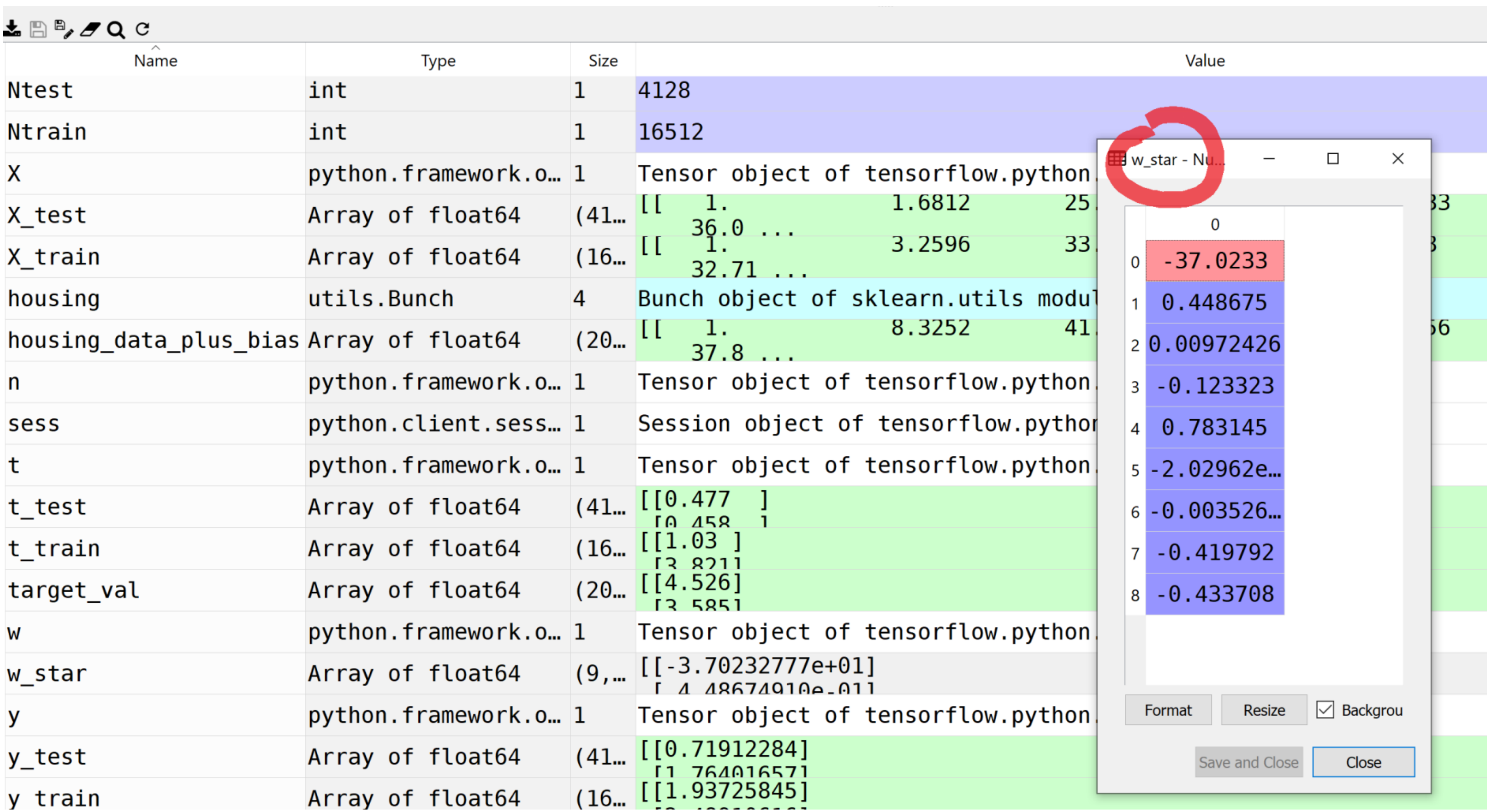
Example: housing price prediction

```
# define the tensors
X = tf.placeholder(tf.float64, shape = (None,M+1), name = 'X') # rows as samples
t = tf.placeholder(tf.float64, shape = (None, 1), name = 't') # target values: t vector
n = tf.placeholder(tf.float64, name='n') # number of samples
XT = tf.transpose(X)
w = tf.matmul(tf.matmul(tf.matrix_inverse(tf.matmul(XT,X)),XT),t) # w=inv(X'*X)*X'*t

# predicted values: a column vector y=[y1, y2, ..., yn]', where yn=xn'*w
y = tf.matmul(X,w)
# mean-squared error of the prediction
MSE = tf.div(tf.matmul(tf.transpose(y-t),y-t),n)



with tf.Session() as sess:
    MSE_train, w_star,y_train = \
    sess.run([MSE,w,y], feed_dict={X: X_train,t: t_train, n: Ntrain})

    MSE_test,y_test = \
    sess.run([MSE,y],feed_dict={X:X_test,t:t_test,n:Ntest,w:w_star})

```

Example: housing price prediction

Name	Type	Size	Value
M	int	1	8
MSE_test	Array of float64	(1,…	[[0.5558916]]
MSE_train	Array of float64	(1,…	[[0.51793313]]
N	int	1	20640
Ntest	int	1	4128
Ntrain	int	1	16512
X	python.framework.o…	1	Tensor object of tensorflow.python.framework.ops module
X_test	Array of float64	(41…	[[1. 1.6812 25. ... 3.87743733 36.0 ...
X_train	Array of float64	(16…	[[1. 3.2596 33. ... 3.6918138 32.71 ...
housing	utils.Bunch	4	Bunch object of sklearn.utils module
housing_data_plus_bias	Array of float64	(20…	[[1. 8.3252 41. ... 2.55555556 37.8 ...
n	python.framework.o…	1	Tensor object of tensorflow.python.framework.ops module
sess	python.client.sess…	1	Session object of tensorflow.python.client.session module
t	python.framework.o…	1	Tensor object of tensorflow.python.framework.ops module
t_test	Array of float64	(41…	[[0.477] [0.458]
t_train	Array of float64	(16…	[[1.03] [3.821]
target_val	Array of float64	(20…	[[4.526] [3.585]

Example: housing price prediction



Remark: Line Fitting

- Notation:
- $\mathbf{w} \triangleq [w_0, w_1]^T$
- $\mathbf{x}_n \triangleq [1, x_n]^T$
- Inner product:
- $y_n = w_0 + w_1 x_n = [w_0, w_1] \times \begin{bmatrix} 1 \\ x_n \end{bmatrix} = \mathbf{w}^T \mathbf{x}_n$
- Prediction Error

$$y_n - t_n = \mathbf{w}^T \mathbf{x}_n - t_n$$

- Sum of the squared errors

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{\mathbf{w}^T \mathbf{x}_n - t_n\}^2$$

Testing and Validation – Case 1

- When you only need to learn the model parameters:
- Split your data into two sets: the **training set** and the **test set**.
  - It is common to use 80% of the data for training and *hold out* 20% for testing.
- You train your model using the **training set**, and you test it using the **test set**.
  - The error rate on the test set is called the generalization error.

Testing and Validation – Case 2

- When you need to learn the model parameters and some hyperparameters, such as the λ in Ridge Regression
- Split your data set into three sets: **training set**, **validation set**, and **test set**
- You train multiple models with various hyperparameters using the **training set**, you select the model and hyperparameters that perform best on the **validation set**
- With the selected model, you run a single final test against the **test set** to get an estimate of the generalization error.