Lecture 3 – Nonlinear Regression

Bulat Ibragimov

bulat@di.ku.dk

Department of Computer Science
University of Copenhagen

Outline

1. Recap: Linear Regression
2. Non-Linear Response, Overfitting, and Cross-Validation
3. Regularisation
4. Summary & Outlook

Recap: Multivariate Linear Regression

- Given: Pairs of the form $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \in \mathbb{R}^D \times \mathbb{R}$.
- Let's "augment" all data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$. This yields an augmented data matrix $\boldsymbol{X} \in \mathbb{R}^{N\times(D+1)}$ and an associated target vector $\boldsymbol{t} \in \mathbb{R}^N$:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \ldots & x_{1,D} \\ 1 & x_{2,1} & x_{2,2} & \ldots & x_{2,D} \\ \vdots & & & & \\ 1 & x_{N,1} & x_{N,2} & \ldots & x_{N,D} \end{bmatrix} \quad \text{and} \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

- As before, we can write the overall loss in the following form:

Overall Loss

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(f(\mathbf{x}_n;\mathbf{w}) - t_n)^2 = \frac{1}{N}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})$$

Recap: Multivariate Linear Regression

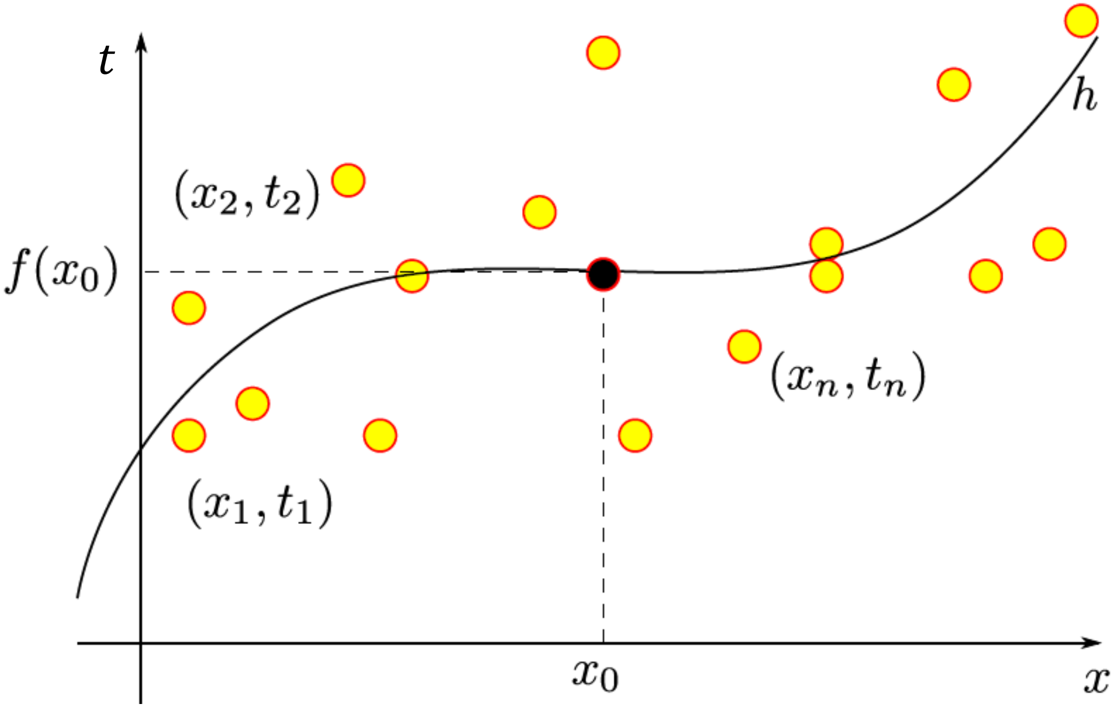
- Given: Pairs of the form $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \in \mathbb{R}^D \times \mathbb{R}$.
- Goal: Find $(D+1)$-dimensional weight vector $\hat{\mathbf{w}} = [\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_D]^T$ that minimizes $\mathcal{L}(\mathbf{w}) = \frac{1}{N}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})$, i.e., which is a solution for

$$\begin{aligned} \nabla \mathcal{L}(\mathbf{w}) &= \mathbf{0} \\ \Leftrightarrow \quad \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} &= \boldsymbol{X}^T\boldsymbol{t} \end{aligned} \tag{1}$$

Computation in Practice

1. Definition of data matrix $\mathbf{X} \in \mathbb{R}^{N\times(D+1)}$ (make use of Numpy arrays and functions!)
2. There are different ways to compute an optimal weight vector $\hat{\mathbf{w}}$:
    1. Compute $\hat{\mathbf{w}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}$ (e.g., via `numpy.linalg.inv`)
    2. Directly solve system of equations (1) (e.g., via `numpy.linalg.solve`)
    3. ...
3. For new point $\mathbf{x}_{new} \in \mathbb{R}^D$: Compute $t_{new} = [1, \mathbf{x}_{new}^T]\hat{\mathbf{w}}$

Non-Linear Regression



The same idea as for linear regression, we want to find a non-linear representation $z = g(x)$, so that loss $\boldsymbol{L}(g(x), t)$ is minimal

Can we use the ideas from multivariate regression for non-linear regression?

Quadratic model

- Let's focus again on $D = 1$, i.e., on input data of the form $x_n \in \mathbb{R}$.
- Now, let's "augment" all data points $x_1, x_2, \ldots, x_N$, now with an additional column containing x_n^2. This yields an augmented data matrix $\boldsymbol{X} \in \mathbb{R}^{N \times 3}$ and an associated target vector $\boldsymbol{t} \in \mathbb{R}^N$:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_N & x_N^2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

- Note: Basically as before, just a "new" input feature/variable.
  (i.e., we have generated a new column based on an existing column)
- As before: $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}^T \boldsymbol{w}$ with $\boldsymbol{x} = [1, x, x^2]^T$ and $\boldsymbol{w} = [w_0, w_1, w_2]^T$
- Our model is still linear in the parameters, but the actual function that is fitted is now quadratic:

$$f(\boldsymbol{x}; \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2$$

Polynomial model

- We can continue adding columns of this form ...

$$\boldsymbol{X} = \begin{bmatrix} x_1^0 & x_1^1 & x_1^2 & \dots & x_1^K \\ x_2^0 & x_2^1 & x_2^2 & \dots & x_2^K \\ \vdots & & & & \\ x_N^0 & x_N^1 & x_N^2 & \dots & x_N^K \end{bmatrix} \quad \text{and} \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

- Our model function can then be written as $f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{k=0}^{K} w_k x^k$

Demo

jupyter L8_Non_Linear_Regression Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [1]:
```
import numpy
import matplotlib.pyplot as plt
import linreg
```

In [2]:
```
# number of data points
n_points = 15

# maximum degree
max_degree = 2
```

In [3]:
```
# set a seed here to initialize the random number generator
# (such that we get the same dataset each time this cell is executed)
numpy.random.seed(1)

# let's generate some "non-linear" data; note
# that the sorting step is done for visualization
# purposes only (to plot the models as connected lines)
X = numpy.random.uniform(-10,10, n_points)
t = X**2 + numpy.random.random(n_points) * 25

# reshape both arrays to make sure that we deal with
# N-dimensional Numpy arrays
t = t.reshape((len(t), 1))
X = X.reshape((len(X),1))
print("Shape of our data matrix: %s" % str(X.shape))
print("Shape of our target vector: %s" % str(t.shape))
```

```
Shape of our data matrix: (15, 1)
Shape of our target vector: (15, 1)
```

In [4]:
```
# instantiate the regression model
model = linreg.LinearRegression()

# fit the model
model.fit(X,t)
```

Arbitrary 'basis' functions

- We can basically resort to arbitrary functions ...

$$\boldsymbol{X} = \begin{bmatrix} h_1(x_1) & h_2(x_1) & h_3(x_1) & \ldots & h_K(x_1) \\ h_1(x_2) & h_2(x_2) & h_3(x_2) & \ldots & h_K(x_2) \\ \vdots & & & & \\ h_1(x_N) & h_2(x_N) & h_3(x_N) & \ldots & h_K(x_N) \end{bmatrix} \quad \text{and} \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

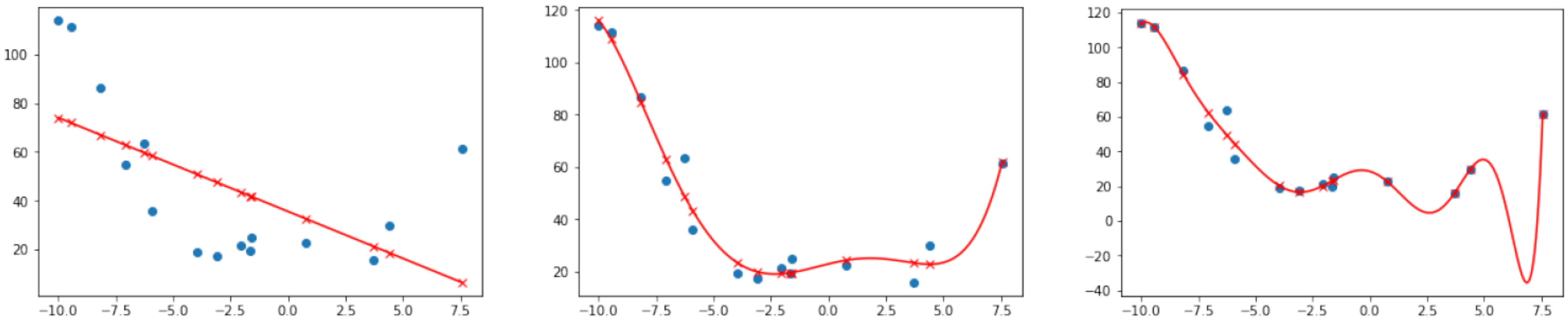
- Our model function can then be written as $f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{k=0}^{K} w_k h_k(x)$

General Case

Also, given more input variables ($D > 1$), we can simply

- transform each input variable/column ...
- combine different input variables (e.g., difference between columns) ...
- combine and transform input variables ...
- ...

Which model is optimal?

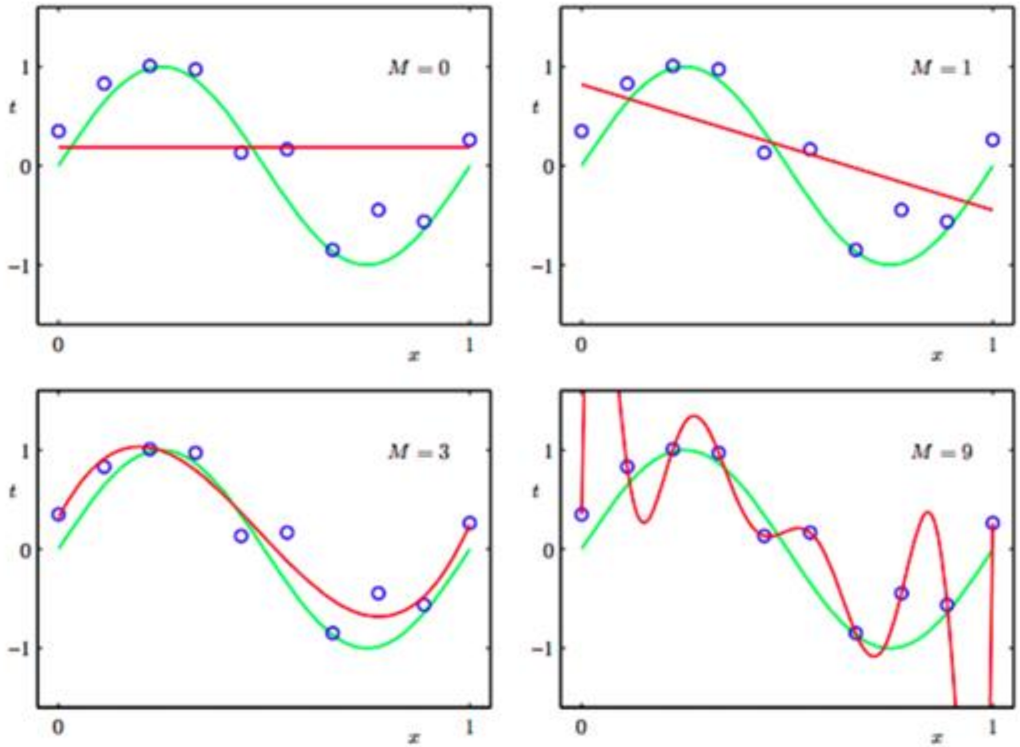


- We are given a so-called training set $T = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\} \subset \mathbb{R}^D \times \mathbb{R}$.
- Given the additional flexibility, we now have to choose a “good” model . . .
  1. Which non-linear functions should we choose?
  2. How many additional columns should be generated?
  3. . . .
- We would like to choose a model that performs well on new, unseen data!

Question: How can we select such a model?

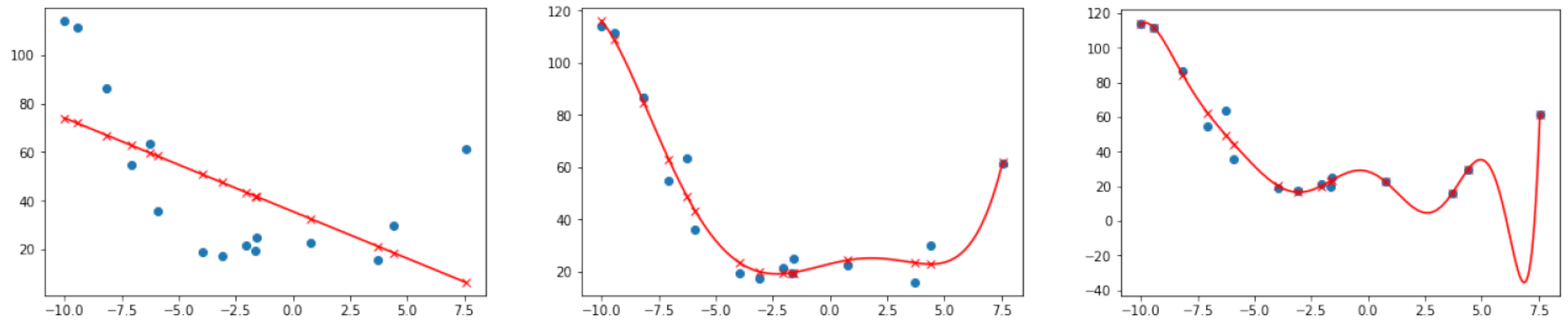
Overfitting

A model that performs best on training samples does not necessarily performs well on test samples



Polynomial Curve Fitting: Polynomials having various order M (in red), fitted to the data (in blue) coming from the true underlying curve shown in green.

Regularization



- The simple model $f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{x}^T\mathbf{w}$ with $\mathbf{w} = [0, \ldots, 0]^T$ always predicts 0.
- Consider the following 5-th order polynomial:

$$f(x; \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5$$

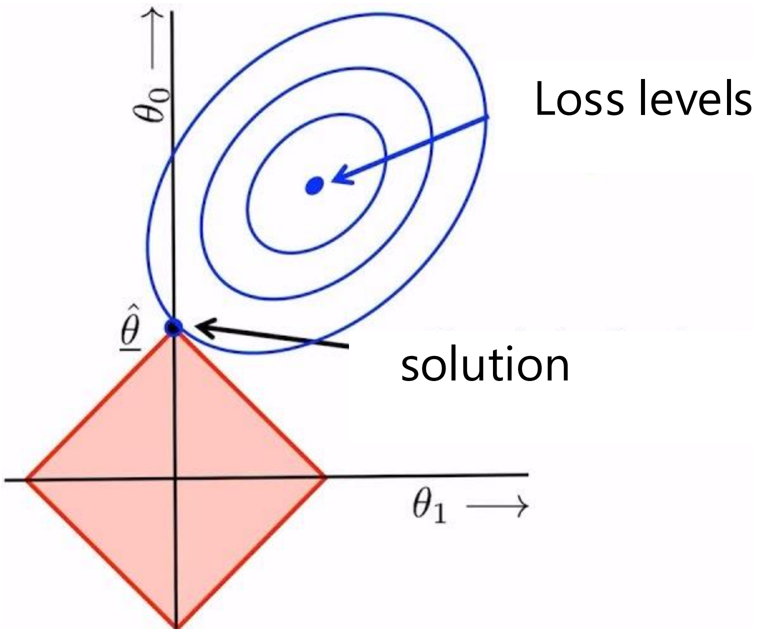
Let's make the model "pay" for every non-zero w_i

Regularization

We can augment the loss function:

$$Loss = \sum_i (f(\mathbf{x};\mathbf{w}) - \boldsymbol{t})^2 + \mu \sum_j |w_j|$$

What will this additional terms do?



Regularization

Let’s consider two sets of parameters:

- $\breve{W} = [0, 0, 1, 10, 0]$
- $\bar{W} = [0, 0, 1, 1, 1]$

Which set will result in more reliable, robust model?

$\breve{W}$ uses fewer coefficients

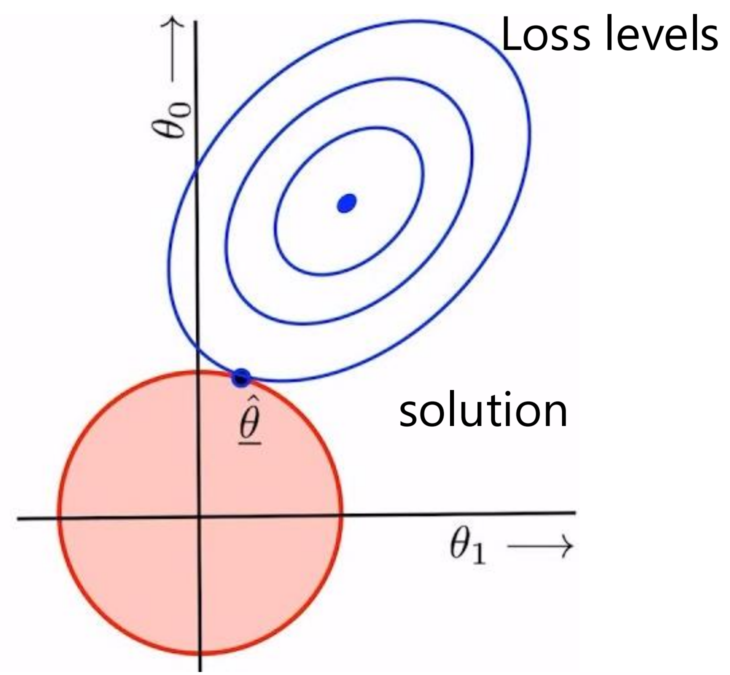
$\bar{W}$ does not put all of its trust to one coefficient

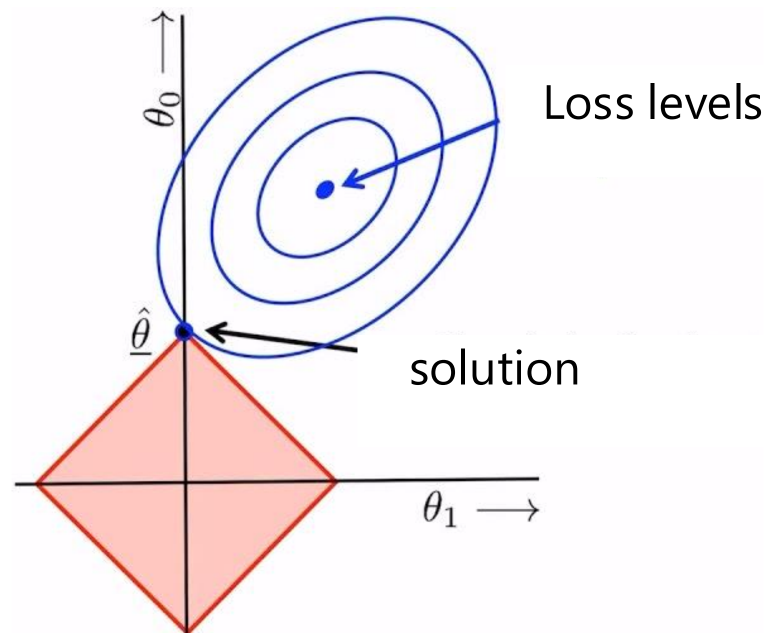
Regularization

We can augment the loss function:

$$Loss = \sum_i (f(\mathbf{x}; \mathbf{w}) - \boldsymbol{t})^2 + \mu \sum_j (w_j)^2$$

What will this additional term do?





Gradient

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}) + \lambda\mathbf{w}^T\mathbf{w},$$

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \frac{2}{N}\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{t} + \frac{1}{N}\boldsymbol{t}^T\boldsymbol{t} + \lambda\mathbf{w}^T\mathbf{w}$$

Toolbox (Table 1.4 in Rogers & Girolami)

1. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{x} \Rightarrow \nabla f(\boldsymbol{w}) = \boldsymbol{x}$
2. $f(\mathbf{w}) = \boldsymbol{x}^T\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = \boldsymbol{x}$
3. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = 2\boldsymbol{w}$
4. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{C}\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = 2\boldsymbol{C}\boldsymbol{w}$

Multivariate Regularized Linear Regression

- Given: Pairs of the form $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \in \mathbb{R}^D \times \mathbb{R}$ and parameter $\lambda > 0$.
- Goal: Find $(D+1)$-dimensional weight vector $\hat{\mathbf{w}} = [\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_D]^T$ that minimizes $\mathcal{L}(\mathbf{w}) = \frac{1}{N}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}) + \lambda \mathbf{w}^T\mathbf{w}$, i.e., which is a solution to

$$\begin{aligned} \nabla \mathcal{L}(\mathbf{w}) &= \mathbf{0} \\ \Leftrightarrow \quad (\boldsymbol{X}^T\boldsymbol{X} + N\lambda\mathbf{I})\boldsymbol{w} &= \boldsymbol{X}^T\boldsymbol{t} \end{aligned} \tag{2}$$

Solve (2) to find the optimal multivariate solution

Logistic regression

Let’s say we have the following data:

	Tumor Size	Recurrence
Case1	0.5	0
Case2	2.11	0
Case3	2.9	1
Case4	2.8	1
Case5	2.1	1
Case6	1.9	0

Can we predict cancer recurrence with linear regression?

Minimizer for Linear Regression

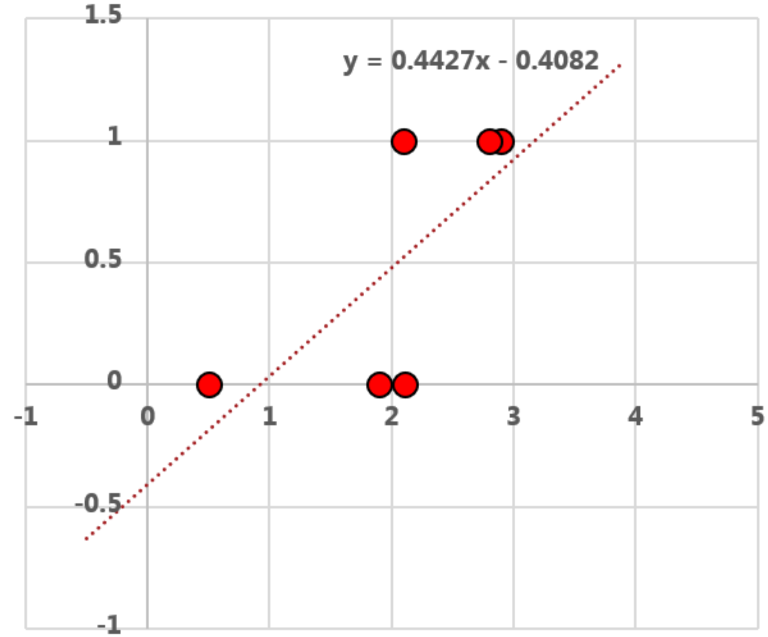
$$\hat{\mathbf{w}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}$$

Logistic regression

The linear regression solutions is:

$$w_0 = -0.4082; w_1 = 0.4427$$

	Tumor Size	Recurrence	Predicted
Case1	0.5	0	-0.19
Case2	2.11	0	0.53
Case3	2.9	1	0.88
Case4	2.8	1	0.83
Case5	2.1	1	0.52
Case6	1.9	0	0.43



$$L = (-0.19)^2 + (0.53)^2 + (1-0.88)^2 + (1-0.83)^2 + (1-0.52)^2 + (0.43)^2 = 0.7718$$

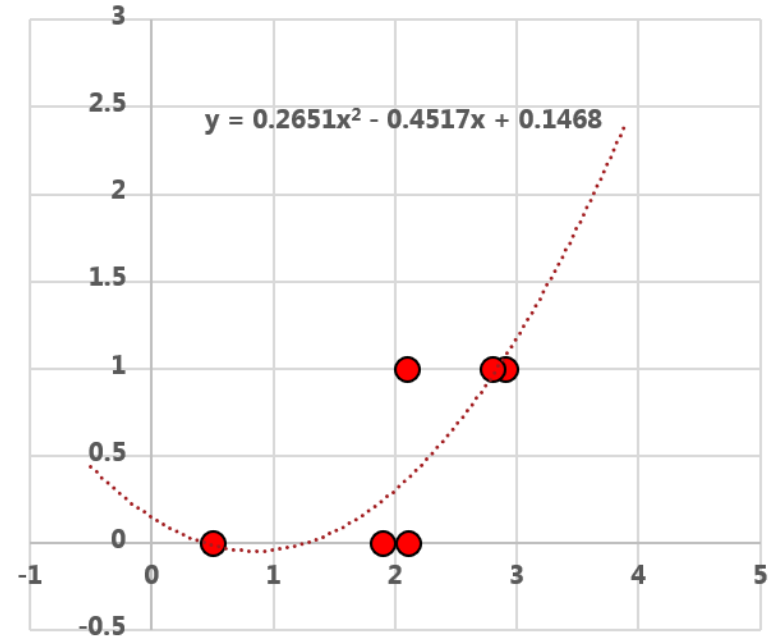
Let's say we add a patient with tumor size 20, and recurrence 1.
Will linear regression work properly?

Logistic regression

The nonlinear (2d polynomial) regression solutions is:

$$w_0 = 0.147; w_1 = -0.452; w_1 = 0.265$$

	Tumor Size	Recurrence	Predicted
Case1	0.5	0	-0.01
Case2	2.11	0	0.37
Case3	2.9	1	1.06
Case4	2.8	1	0.96
Case5	2.1	1	0.37
Case6	1.9	0	0.24



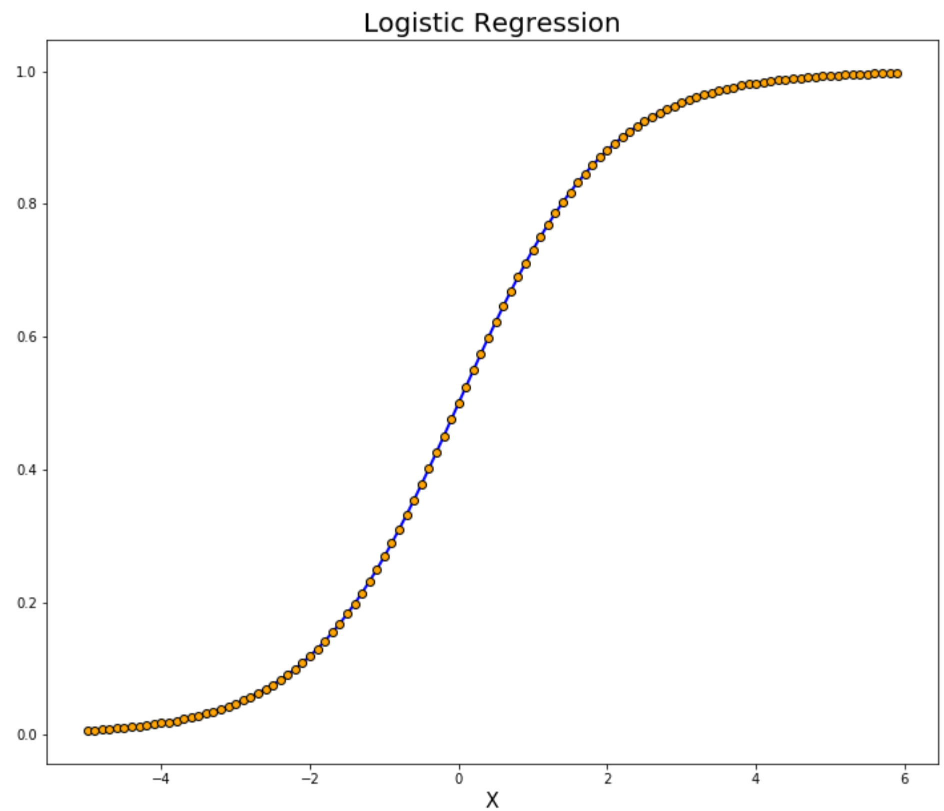
$$L = (-0.01)^2 + (0.37)^2 + (1 - 1.06)^2 + (1 - 0.96)^2 + (1 - 0.37)^2 + (0.24)^2 = 0.6066$$

Logistic regression

The regressions we tried are not perfectly suitable. We would like:

- Model to saturate and not go above 1
- Model to saturate and not go below 0

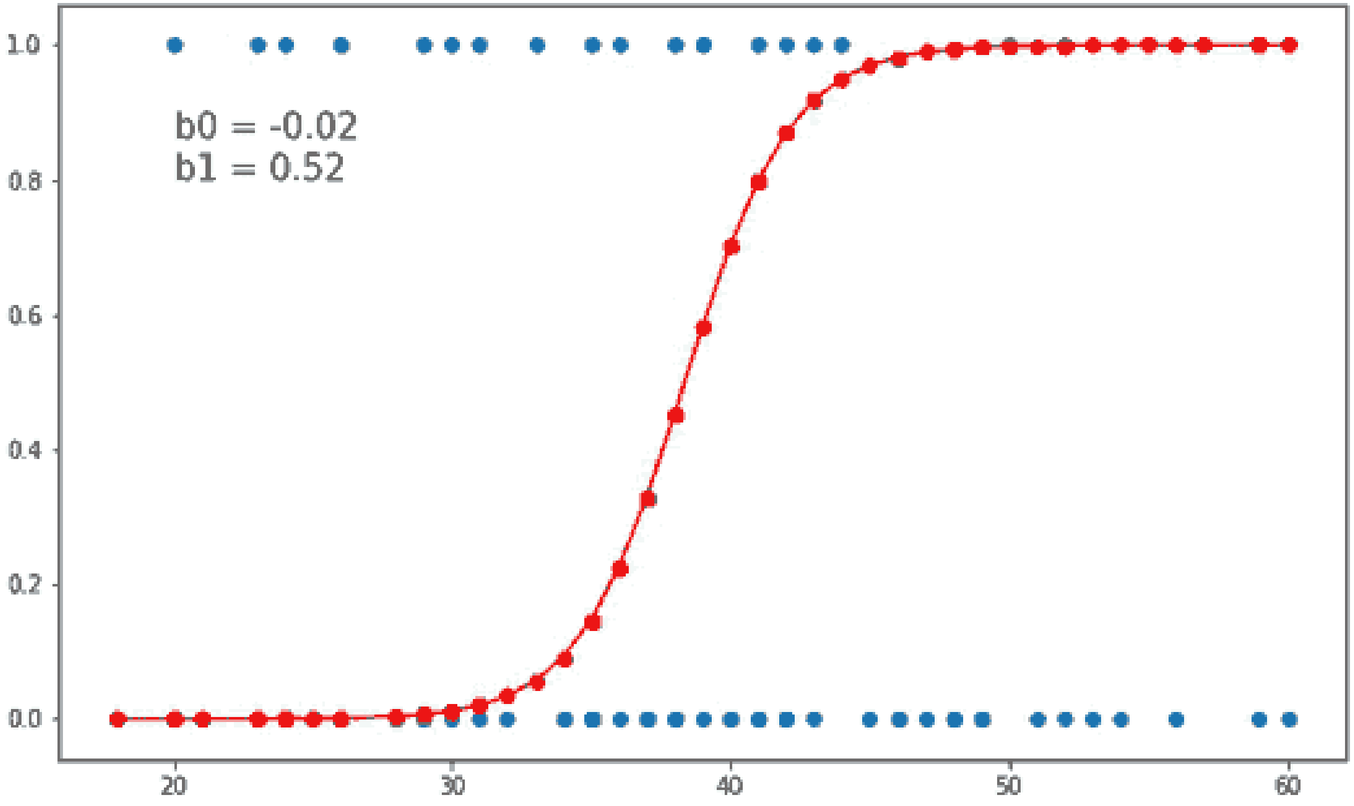
$$F(\boldsymbol{w}, \boldsymbol{x}) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$



Logistic regression

The regressions we tried are not perfectly suitable. We would like:

- Model to saturate and not go above 1
- Model to saturate and not go below 0



Logistic regression: derivatives

Loss function:

$$L(w_0, w_1) = \sum_i (F(\boldsymbol{w}, x_i) - t_i)^2$$

Derivatives:

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial F}\frac{\partial F}{\partial w_0} =$$

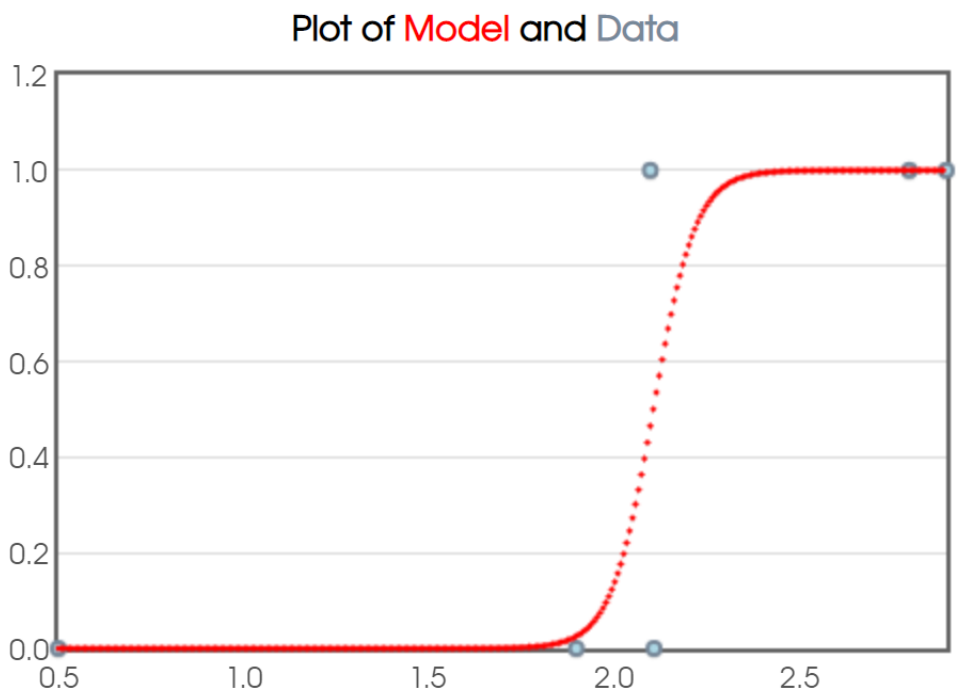
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial F}\frac{\partial F}{\partial w_1} =$$

Logistic regression

	Tumor Size	Recurrence	Predicted
Case1	0.5	0	0
Case2	2.11	0	0.51
Case3	2.9	1	1
Case4	2.8	1	1
Case5	2.1	1	0.47
Case6	1.9	0	0.03

Model: $$P = \frac{1}{1 + e^{-(-36.9639 + 17.5358 x_1)}}$$

$$L = (0)^2 + (0.51)^2 + (1-1)^2 + (1-1)^2 + (1-0.47)^2 + (0.03)^2 = 0.5457$$



Problem with derivatives

Linear, polynomial and logistic regression models are very simple, but their derivatives are already complex:

Linear regression

$$\frac{\partial \mathcal{L}}{\partial w_0} = 2w_0 + 2w_1 \frac{1}{N}\left(\sum_{n=1}^{N} x_n\right) - \frac{2}{N}\left(\sum_{n=1}^{N} t_n\right)$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = 2w_1 \frac{1}{N}\left(\sum_{n=1}^{N} {x_n}^2\right) + \frac{2}{N}\left(\sum_{n=1}^{N} x_n(w_0 - t_n)\right)$$

Regression with regularization

$$\frac{2}{N}\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{N}\mathbf{X}^T\mathbf{t} + 2\lambda\mathbf{w} = \mathbf{0}$$

Coding

jupyter Multivariate Linear Regression Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

copy selected cells

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy
```

We shall work with the dataset found in the file 'murderdata.txt', which is a 20 x 5 data matrix where the columns correspond to

Index (not for use in analysis)

Number of inhabitants

Percent with incomes below $5000

Percent unemployed

Murders per annum per 1,000,000 inhabitants

Reference:

Helmut Spaeth, Mathematical Algorithms for Linear Regression, Academic Press, 1991, ISBN 0-12-656460-4.

D G Kleinbaum and L L Kupper, Applied Regression Analysis and Other Multivariable Methods, Duxbury Press, 1978, page 150.

http://people.sc.fsu.edu/~jburkardt/datasets/regression

What to do?

We start by loading the data; today we will study how the number of murders relates to the percentage of unemployment.

In [ ]:
```
data = numpy.loadtxt('murderdata.txt')
N, d = data.shape
```

We consider all both features simulaneously.

In [ ]:
```
t = data[:,4]
X = data[:,2:4]
print("Number of training instances: %i" % X.shape[0])
print("Number of features: %i" % X.shape[1])
```

In [ ]:
```
# NOTE: This template makes use of Python classes. If
# you are not yet familiar with this concept, you can
# find a short introduction here:
# http://introtopython.org/classes.html

class LinearRegression():
    """
    Linear regression implementation.
    """

    def __init__(self):
```

Coding

Computation in Practice

1. Definition of data matrix $\mathbf{X} \in \mathbb{R}^{N \times (D+1)}$ (make use of Numpy arrays and functions!)
2. There are different ways to compute an optimal weight vector $\hat{\mathbf{w}}$:
   1. Compute $\hat{\mathbf{w}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}$ (e.g., via `numpy.linalg.inv`)
   2. Directly solve the system of equations (e.g., via `numpy.linalg.solve`):

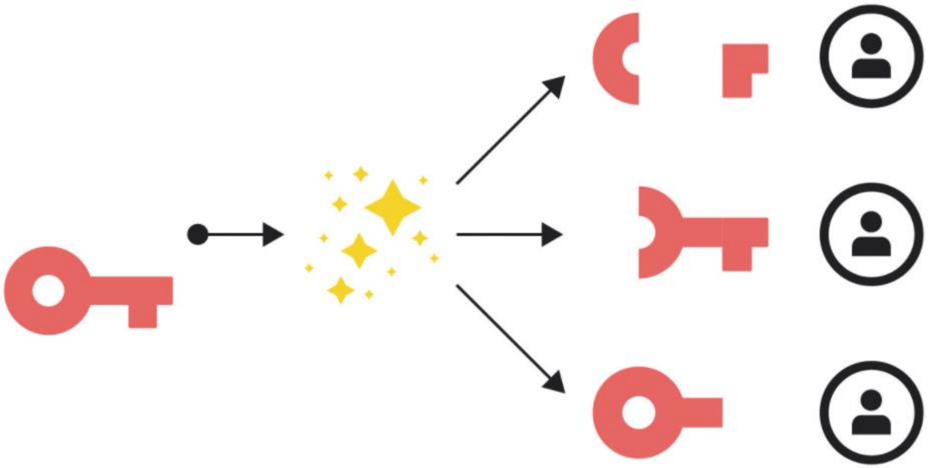
$$\begin{aligned} \nabla\mathcal{L}(\mathbf{w}) &= \mathbf{0} \\ \Leftrightarrow \quad \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} &= \boldsymbol{X}^T\boldsymbol{t} \end{aligned}$$

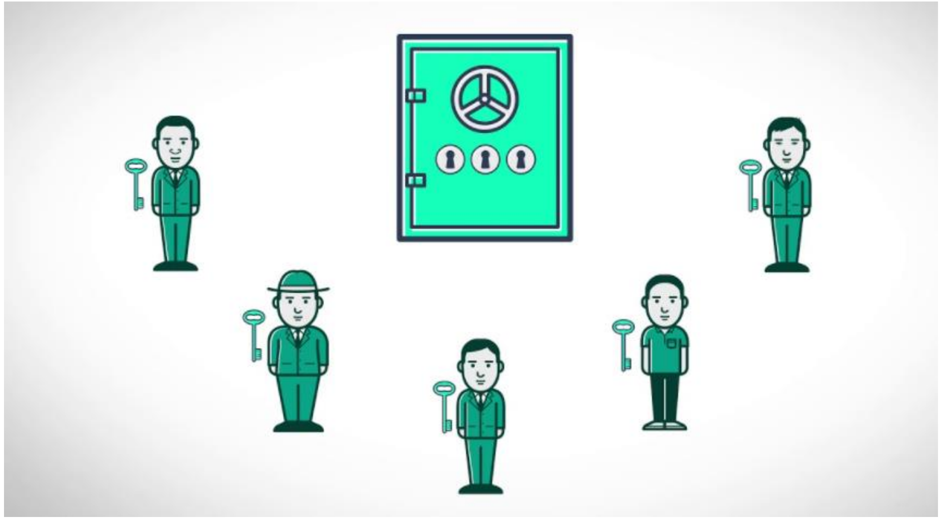
   3. ...
3. For new point $\mathbf{x}_{new} \in \mathbb{R}^D$: Compute $t_{new} = [1, \mathbf{x}_{new}^T]\hat{\mathbf{w}}$

```
In [ ]: t = data[:,4]
        X = data[:,2:4]
        print("Number of training instances: %i" % X.shape[0])
        print("Number of features: %i" % X.shape[1])

In [ ]: # NOTE: This template makes use of Python classes. If
        # you are not yet familiar with this concept, you can
        # find a short introduction here:
        # http://introtopython.org/classes.html

        class LinearRegression():
```

Shamir's secret sharing

$$f(\mathbf{x};\mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots.$$



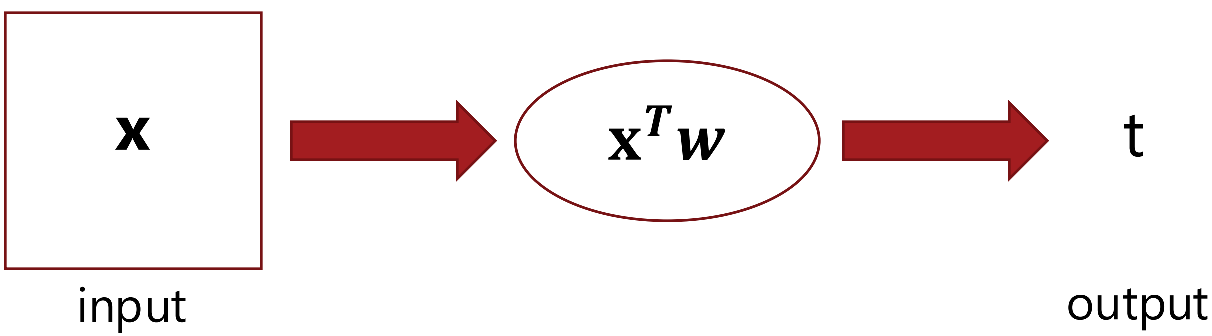


Questions?

Iterative optimization

Linear regression:

- Input x is 1-dimensional (tumor size)
- Output y is 1-dimensional (survival time)



Iterative optimization: initialization

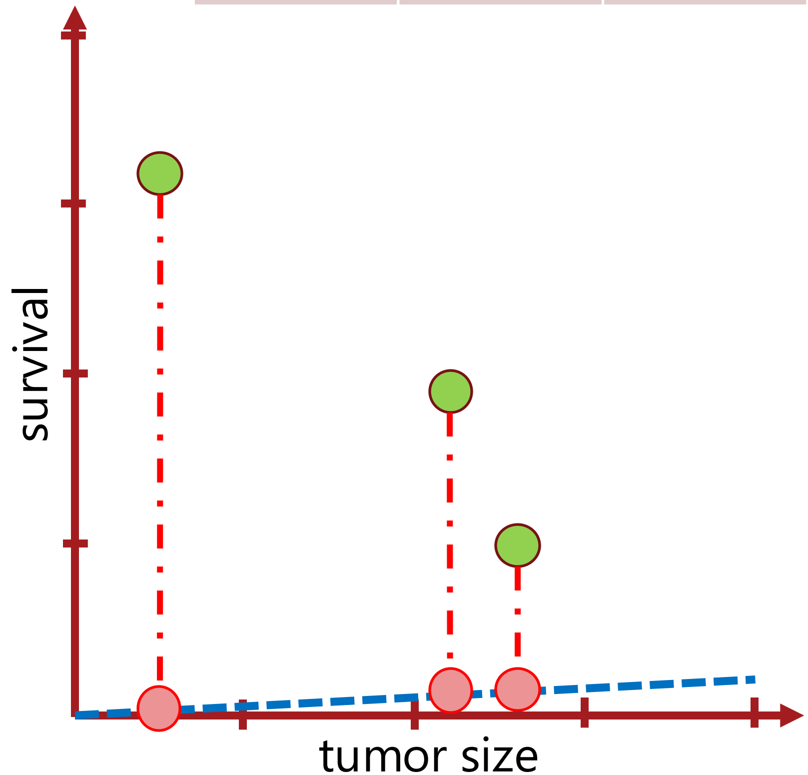
	Tumor Size	Survival
Case1	0.5	3.2
Case2	2.3	1.9
Case3	2.9	1.0

Let's initialize our model:

- $w_0 = 0.01; w_1 = 0.01$

The performance of the model is low:

- $y_1 = w_0 + w_1 x_1 = 0.01 + 0.01 * 0.5 = 0.015$
- $y_2 = w_0 + w_1 x_2 = 0.01 + 0.01 * 2.3 = 0.033$
- $y_3 = w_0 + w_1 x_3 = 0.01 + 0.01 * 2.9 = 0.039$



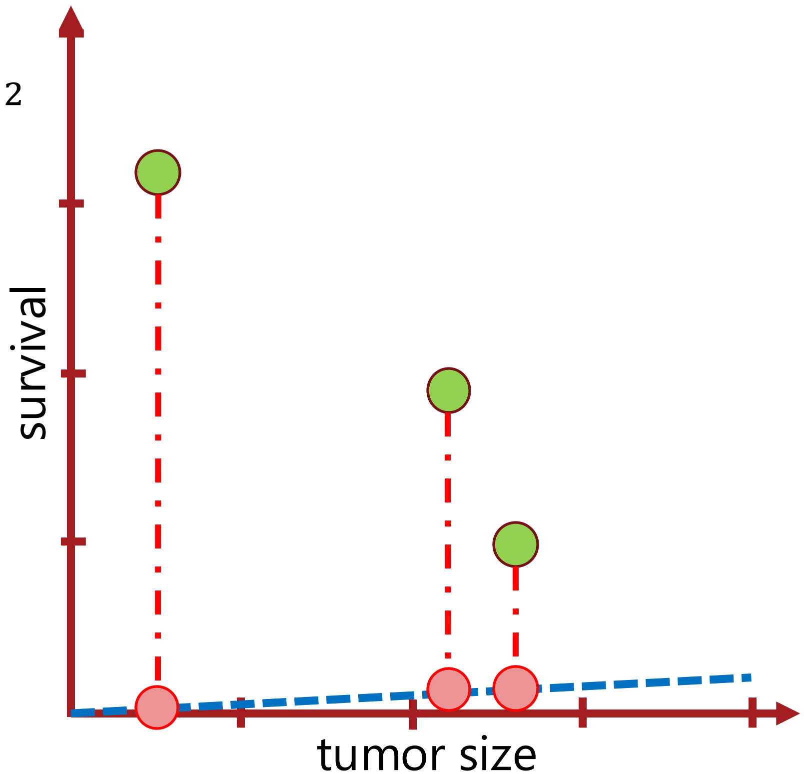
Iterative optimization: loss function

	Tumor Size	Survival
Case1	0.5	3.2
Case2	2.3	1.9
Case3	2.9	1.0

The performance of the model is low, but how low?

Mean squared error:

- $Loss = \sum_i (f(\mathbf{x}_i) - t_i)^2 = \sum_i \left((\mathbf{x}_i^T \mathbf{w}) - t_i \right)^2$
- $Loss =$
  $(0.015 - 3.2)^2 +$
  $(0.033 - 1.9)^2 +$
  $(0.039 - 1.0)^2 = 14.55$

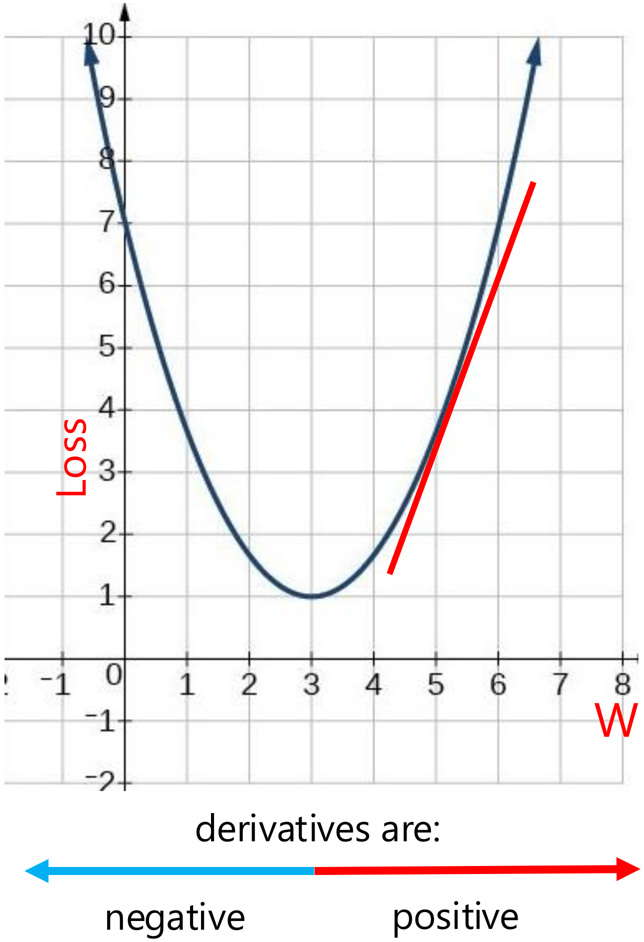


Iterative optimization: derivatives

We want the loss to be as small as possible, i.e. find its minimum.

We use derivatives to find minima/maxima of a function:

- How fast function changes
- Will it increase or decrease



- Chain rule:
  - $\frac{\partial}{\partial x} f(g(x)) = \frac{\partial}{\partial g} f(g(x)) \cdot \frac{\partial}{\partial x} g(x)$

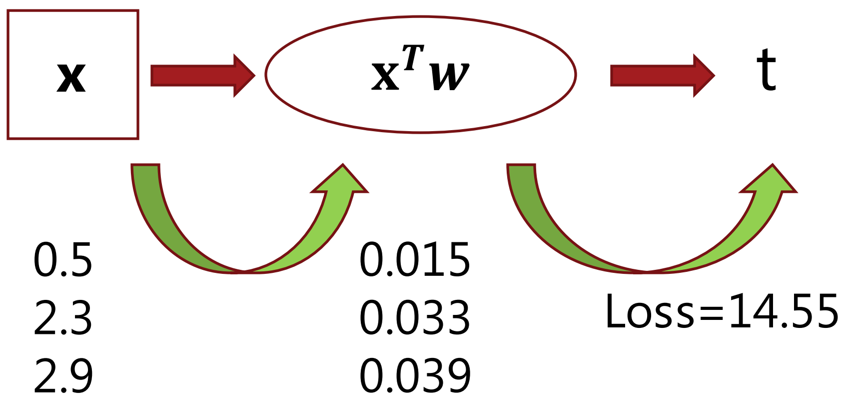
Iterative optimization

	Tumor Size	Survival
Case1	0.5	3.2
Case2	2.3	1.9
Case3	2.9	1.0

Initialization:

- $\boldsymbol{w} = [0.01, 0.01]$

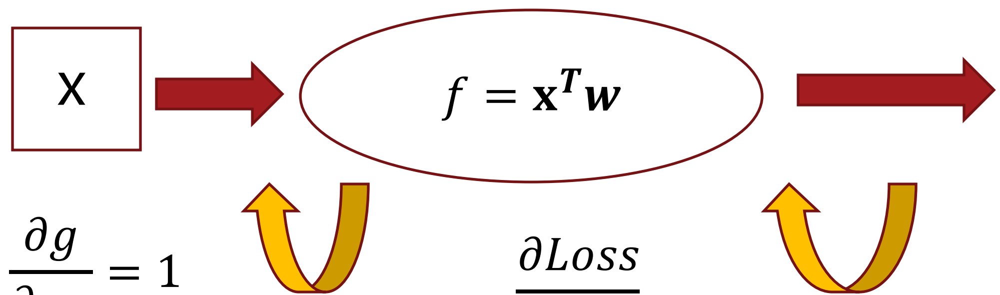
Forward pass:



Iterative optimization

Compute the derivatives:

$$Loss = \sum_i (f - t)^2$$



- How changes in $Wx + b$ affect the loss

$$\frac{\partial g}{\partial w_0} = 1 \qquad \frac{\partial Loss}{\partial f}$$

14.55

- How changes in W and b affect g

$$\frac{\partial f}{\partial w_1} = x \qquad = 2 \sum_i (f(x_i, \boldsymbol{w}) - t_i)$$

- $\frac{\partial Loss}{\partial w_0}$ and $\frac{\partial Loss}{\partial w_1}$ will be computed using the chain rule:
  - $\frac{\partial Loss}{\partial w_0} = 2 \sum_i \left((f(x_i) - t_i) \cdot 1 \right)$
  - $\frac{\partial Loss}{\partial w_1} = 2 \sum_i \left((f(x_i) - t_i) \cdot x_i \right)$

Iterative optimization: optimization step

Update W and b according to the derivatives:

- $w_0 \leftarrow w_0 - \lambda \frac{\partial Loss}{\partial w_0}$; $w_1 \leftarrow w_1 - \lambda \frac{\partial Loss}{\partial w_1}$

Learning rate

The results of the optimization step:

- $w_0 \leftarrow w_0 - 0.1 \cdot 2 \sum_i \left((f(x_i) - t_i) \right) = 0.61$

- $w_1 \leftarrow w_1 - 0.1 \cdot 2 \sum_i \left((f(x_i) - t_i) \cdot x_i \right) = 0.01 - 0.1 \cdot 2 \cdot \left((0.015 - 3.2)0.5 + (0.033 - 1.9)2.3 + (0.039 - 1.0)2.9 \right) = 0.01 - 0.1 \cdot -8.67 = 0.88$

w_0	w_1
0.01	0.01

	Tumor Size	Survival	First solution
Case1	0.5	3.2	0.015
Case2	2.3	1.9	0.033
Case3	2.9	1.0	0.039

Iterative optimization: optimization step

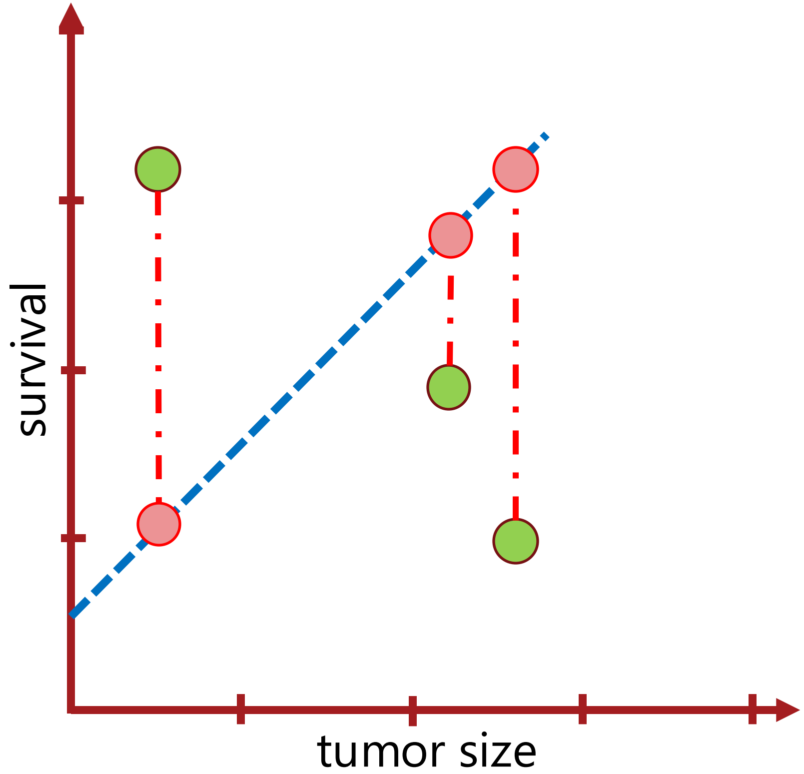
	Tumor Size	Survival
Case1	0.5	3.2
Case2	2.3	1.9
Case3	2.9	1.0

After parameter update:

- $w_0 = 0.61; w_1 = 0.88;$

The performance of the model is better:

- $t_1 = w_0 + w_1 x_1 = 0.61 + 0.88 * 0.5 = 1.05$
- $t_2 = w_0 + w_1 x_2 = 0.61 + 0.88 * 2.3 = 2.63$
- $t_3 = w_0 + w_1 x_3 = 0.61 + 0.88 * 2.9 = 3.16$



Loss improves $Loss\ = 9.8$

Iterative optimization: optimization step

	Tumor Size	Survival
Case1	0.5	3.2
Case2	2.3	1.9
Case3	2.9	1.0

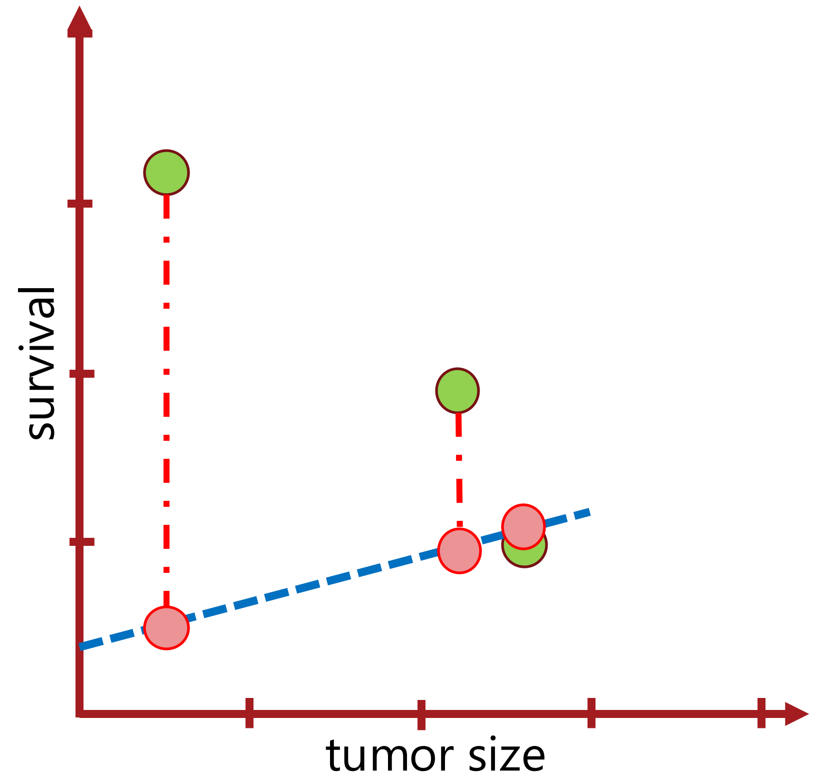
After next parameter update:

- $w_0 = 0.54; w_1 = 0.19$

The performance of the model is better:

- $t_1 = w_0 + w_1 x_1 = 0.54 + 0.19 * 0.5 = 0.63$
- $t_2 = w_0 + w_1 x_2 = 0.54 + 0.19 * 2.3 = 0.98$
- $t_3 = w_0 + w_1 x_3 = 0.54 + 0.19 * 2.9 = 1.10$

Loss improves $Loss \ = 7.46$

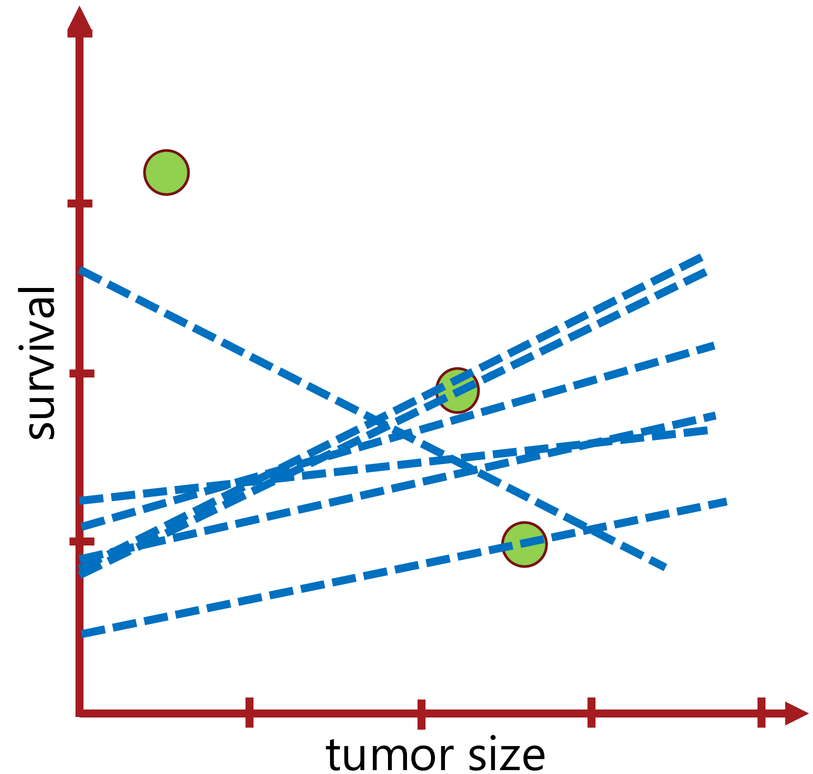


Iterative optimization: optimization step

After next parameter update:

- $w_0 = 0.88; w_1 = 0.50, Loss = 6.07$
- $w_0 = 0.50; w_1 = 0.19, Loss = 7.75$
- $w_0 = 0.85; w_1 = 0.53, Loss = 6.29$
- $w_0 = 0.90; w_1 = 0.19, Loss = 5.37$
- $w_0 = 1.13; w_1 = 0.29, Loss = 4.67$
- $w_0 = 1.23; w_1 = 0.12, Loss = 4.14$
- …
- $w_0 = 1.51; w_1 = 0.04, Loss = 3.27$
- $w_0 = 1.98; w_1 = -0.15, Loss = 2.06$
- $w_0 = 2.18; w_1 = -0.22, Loss = 1.63$
- $w_0 = 2.35; w_1 = -0.29, Loss = 1.31$
- $w_0 = 2.50; w_1 = -0.36, Loss = 1.04$
- $w_0 = 2.63; w_1 = -0.43, Loss = 0.82$

	Tumor Size	Survival
Case1	0.5	3.2
Case2	2.3	1.9
Case3	2.9	1.0



Iterative optimization

Instead of explicitly calculating the global derivative, we can calculate derivatives step by step:

- Eventually, we will calculate how each node is affects the next one
- By using the chain rule, we can estimate how change of w_0 and w_1 will affect the loss L
- We can then slightly modify values of w_0 and w_1 to reduce the loss L, and then recompute the new loss and the repeat the procedure

Derivatives: chain rule

Let's look closely at the derivatives for logistic regression:

$$\frac{\partial L}{\partial b_0} = \sum_i 2(F(x_i) - t_i) \cdot (e^{-(b_0+b_1x)}(1 + e^{-(b_0+b_1x)})^2)$$

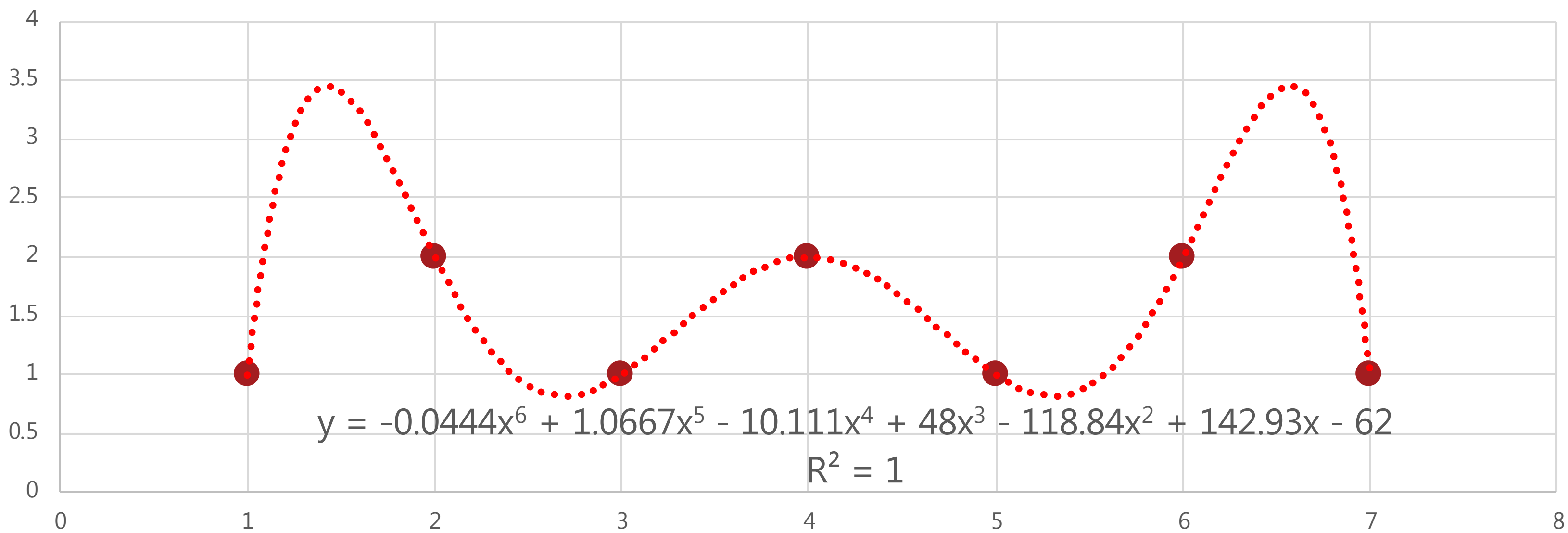
$$\frac{\partial L}{\partial b_1} = \sum_i 2(F(x_i) - t_i) \cdot (e^{-(b_0+b_1x)}(1 + e^{-(b_0+b_1x)})^2)\boxed{x_i}$$

This components comes
from the square loss

This components comes from
the logistic regression derivative

The only difference that comes
from internal linear function

Lagrange interpolation



Lagrange Polynomials

$$L_2(x) = f(x_1)\frac{x-x_2}{x_1-x_2}\cdot\frac{x-x_3}{x_1-x_3} + f(x_2)\frac{x-x_1}{x_2-x_1}\cdot\frac{x-x_3}{x_2-x_3} + f(x_3)\frac{x-x_1}{x_3-x_1}\cdot\frac{x-x_2}{x_3-x_2}$$