QBUS6850
Lecture 2
Python Machine Learning

BUSINESS SCHOOL

Professor Junbin Gao

THE UNIVERSITY OF SYDNEY

Topics

- Topics covered
  - Machine learning model representation
  - Cost/loss function
  - Linear regression with single and multiple features
  - Optimisation algorithm: gradient descent
  - Model and feature selection techniques
  - Learning curves
  - Training, validation and test sets
  - Cross validation

- References
  - Bishop (2006), Chapters 1.3 - 1.5; 3.2
  - Friedman et al., (2001), Chapters 2.3.1, 3.1 - 3.2, 7.1 - 7.6, 7.10
  - James et al., (2014), Chapters 2.1 - 2.2
  - James et al., (2014) Chapter 5.1 (Cross Validation)

Learning Objectives

- ❑ Understand model representation and cost function
- ❑ Understand the matrix representation of linear regression with single and multiple features
- ❑ Understand how gradient descent algorithm works
- ❑ Understand overfitting and underfitting
- ❑ Understand bias and variance decomposition and be able to diagnose them
- ❑ Be able to interpret the learning curves
- ❑ Be able to do the polynomial order selection
- ❑ Understand the reason and process of Cross Validation

ML Basic Concepts and Workflow

THE UNIVERSITY OF SYDNEY

Terminology in ML

- **Input/Feature Supervised learning:**
  - An object is usually characterized by a *feature* scalar or vector
  - Denote by $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$ where each component x_i is a specified feature for the object
  - Don't confuse x_i (the variable component – notation) and $\mathbf{x}_n$ (the n-th case of data with $\mathbf{x}_n = (x_{n1}, x_{n2}, \dots, x_{nd})^T$
  - Each component x_i may be a quantitative value from $\mathbb{R}$ (the set of all real numbers) or one of finite categorical values.
- **Outcome/Target:**
  - An unknown system (to be learnt) which generates an *output/outcome/target* denoted by $\mathbf{t} = (t_1, t_2, \dots, t_m)^T$ for each object feature $\mathbf{x}$
  - Each component t_j may be a quantitative value from $\mathbb{R}$ (the set of all real numbers) or one of finite categorical values
  - In most cases, we assume $m = 1$. We may assume $m = 1$ in this course. Thus t is a scalar
  - As a measurement value, we always suppose there are some noises ϵ in t, i.e., the measurement is $t = y + \epsilon$ where y is the true target.

Terminology in ML

➢ Training/Testing Dataset:

- ❖ A pair of observed $(\mathbf{x}, t)$ is called a training/testing datum.
- ❖ In unsupervised learning case, there is no target observation t.
- ❖ All the available training data are collected together by a set of *training/testing data*, denoted $\mathcal{D}$ with or without target observation

$$\mathcal{D} = \{(\mathbf{x}_n, t_n)\}_{n=1}^{N} \quad \text{or } \{\mathbf{x}_n\}_{n=1}^{N}$$

➢ Learner or Model:

- ❖ Use this dataset $\mathcal{D}$ to build a prediction model, or *learner*; which will enable us to predict the outcome for new unseen objects or characterize them if without outcomes.
- ❖ A good learner is one that accurately predicts such an outcome or make a right characterization.

Data Representation

- Machine learning algorithms are built up data. There exist different types of data. Although the numeric data are widely seen in scientific world, the categorical data are more common in business world

- When we have a data set with N observations $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, we will organise them into a matrix of size $N \times d$ such that each row corresponds to an observation (or a case). If we have the target/output for N observation $\{t_1, t_2, \dots, t_N\}$, we also organise them into a column (simulating a row corresponding to a case or an observation), denote by as

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{21} & \dots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Nd} \end{pmatrix} \in \mathbb{R}^{N\times d}, \qquad \mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix} \in \mathbb{R}^N$$

Machine Learning Flow

- Learning is the process of estimating an unknown dependency between the input and output or structure of a system using a limited number of observations.

- A typical learning procedure consists of the following steps:
  1. Statement of the Problem
  2. Data Generation/Experiment Design
  3. Data Collection and Pre-processing
  4. Hypothesis Formulation and Objectives
  5. Model Estimation and Assessment
  6. Interpretation of the Model and Drawing Conclusions

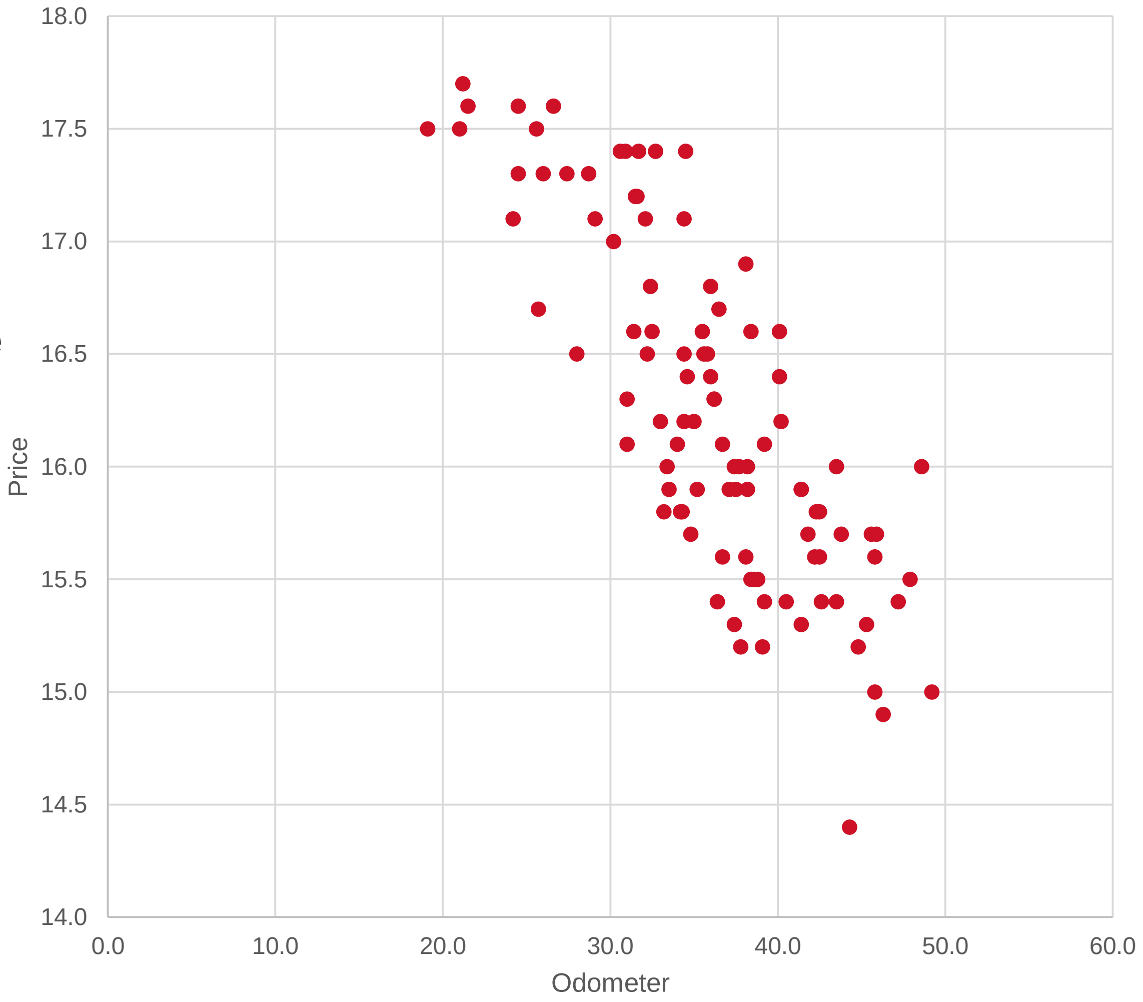
- Many recent application studies tend to focus on the learning methods used (i.e., a neural network).

THE UNIVERSITY OF SYDNEY

Example: Linear Regression

Supervised learning:

Step 1: Regression
'Problem': Predict car price
Steps 2&3: Completed
(Data collected and
labelled)



Example: Linear Regression

Step 4a: Linear Model Hypothesis

$$y = f(\mathbf{x}; \boldsymbol{\beta}) = \beta_0 + \beta_1 x$$

$f()$: simple (univariate) linear regression model;

$\boldsymbol{\beta} = (\beta_0, \beta_1)^T$: model parameters.

Training set

N: number of training examples

$\mathbf{x}$ $(= x)$: "input" variable; **one feature here**

t: "output" variable; "target" variable which is a noised version of model output y, *i.e.*,

$$t = y + \varepsilon$$

(x_n, t_n) n_{th} training example

Odometer (x)	Price (t)
37.4	16.0
44.8	15.2
45.8	15.0
30.9	17.4
31.7	17.4
34.0	16.1
45.9	15.7
…	…
41.4	15.3

$(x_1, t_1) = (37.4, 16.0)$

$(x_3, t_3) = \,??$

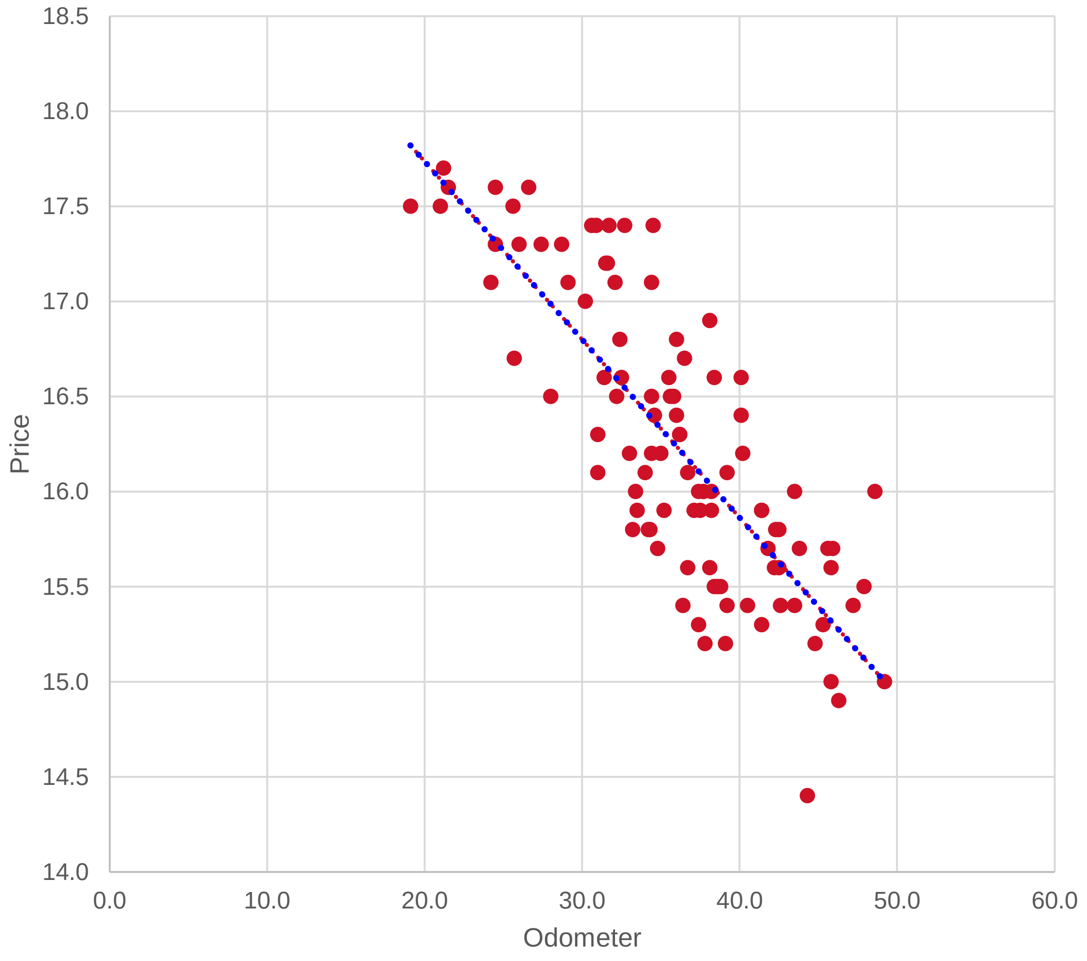
Note: According to our general notation, as there is only one feature, we shall write it as (x_{n1}, t_n)

f() representation?

$$y = f(\mathbf{x}; \boldsymbol{\beta}) = \beta_0 + \beta_1 x$$

How to choose $\boldsymbol{\beta}$?
Or how can we estimate $\boldsymbol{\beta}$?

This is the task in Step 5a



Loss/cost function

Step 4b: Define an appropriate objective for the task in hand;

Purpose: to measure the error between the observed and the model. Loss function, also called a cost function, which is a single, overall measure of loss incurred in taking any of the available decisions or actions. (Bishop, C.M., 2006.)

Predicted y values

Observed t values

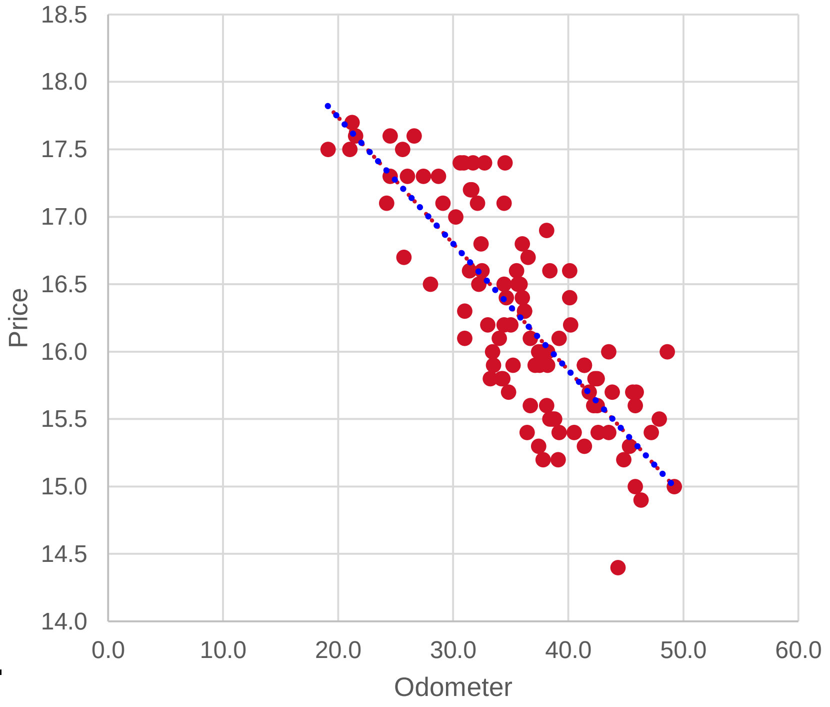
$$L(\beta_0, \beta_1) = \boxed{\frac{1}{2N}} \sum_{n=1}^{N} (f(\mathbf{x}_n, \boldsymbol{\beta}) - t_n)^2$$

For computational convenience

where $f(\mathbf{x}; \boldsymbol{\beta}) = \beta_0 + \beta_1 x$

$$\min_{\beta_0, \beta_1} \quad L(\beta_0, \beta_1)$$

Choose parameters so that estimated linear regression line is close to our training examples. This is also call argmin



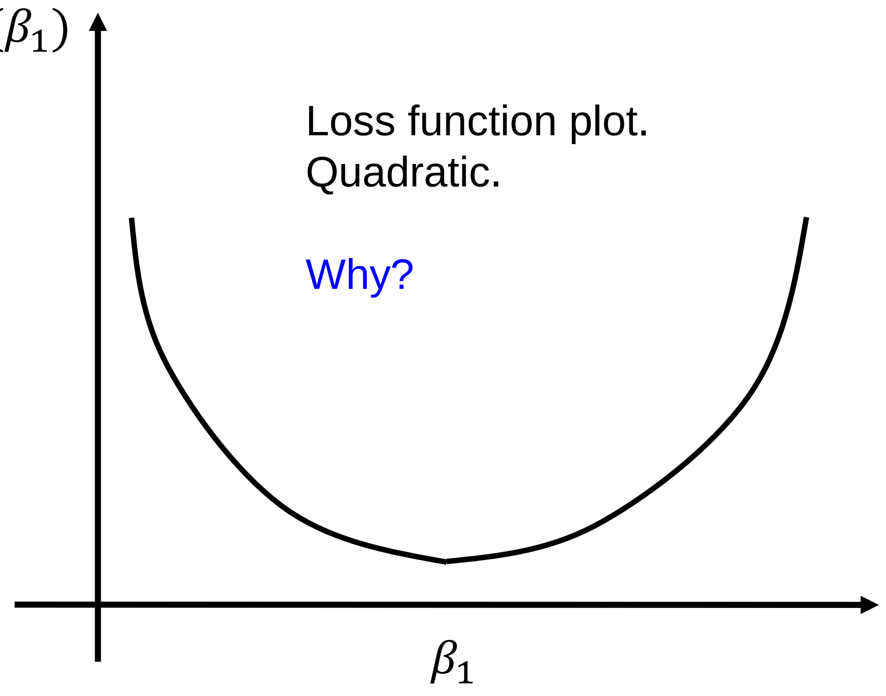
Optimisation Algorithm

Step 5a: Find the model parameters;

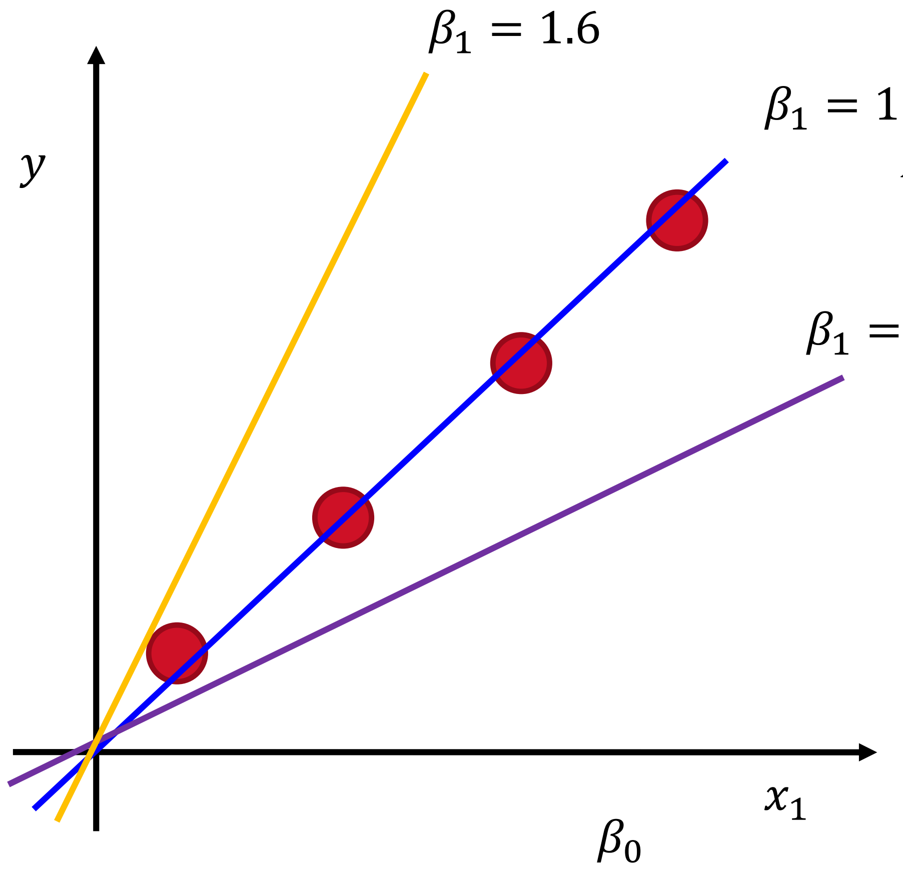
For demo, we assume $\beta_0 = 0$, hence the loss function becomes

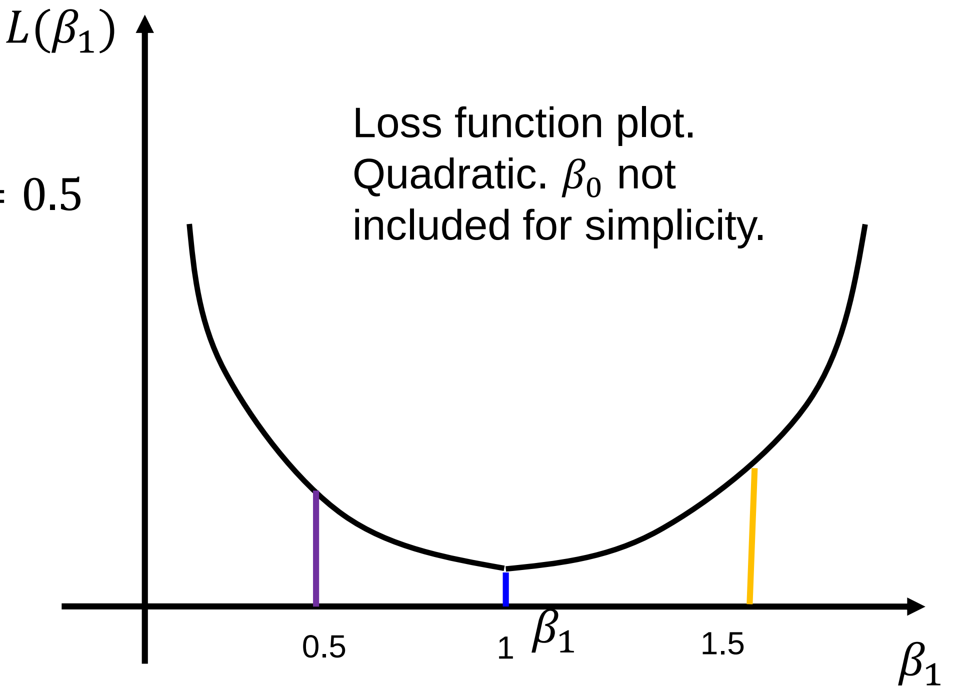
$$L(\beta_1) = \boxed{\frac{1}{2N}} \sum_{n=1}^{N} (f(\boldsymbol{x}_n, \boldsymbol{\beta}) - t_n)^2$$

Sometime this term is added for computational convenience, but will not change the estimation results of parameters



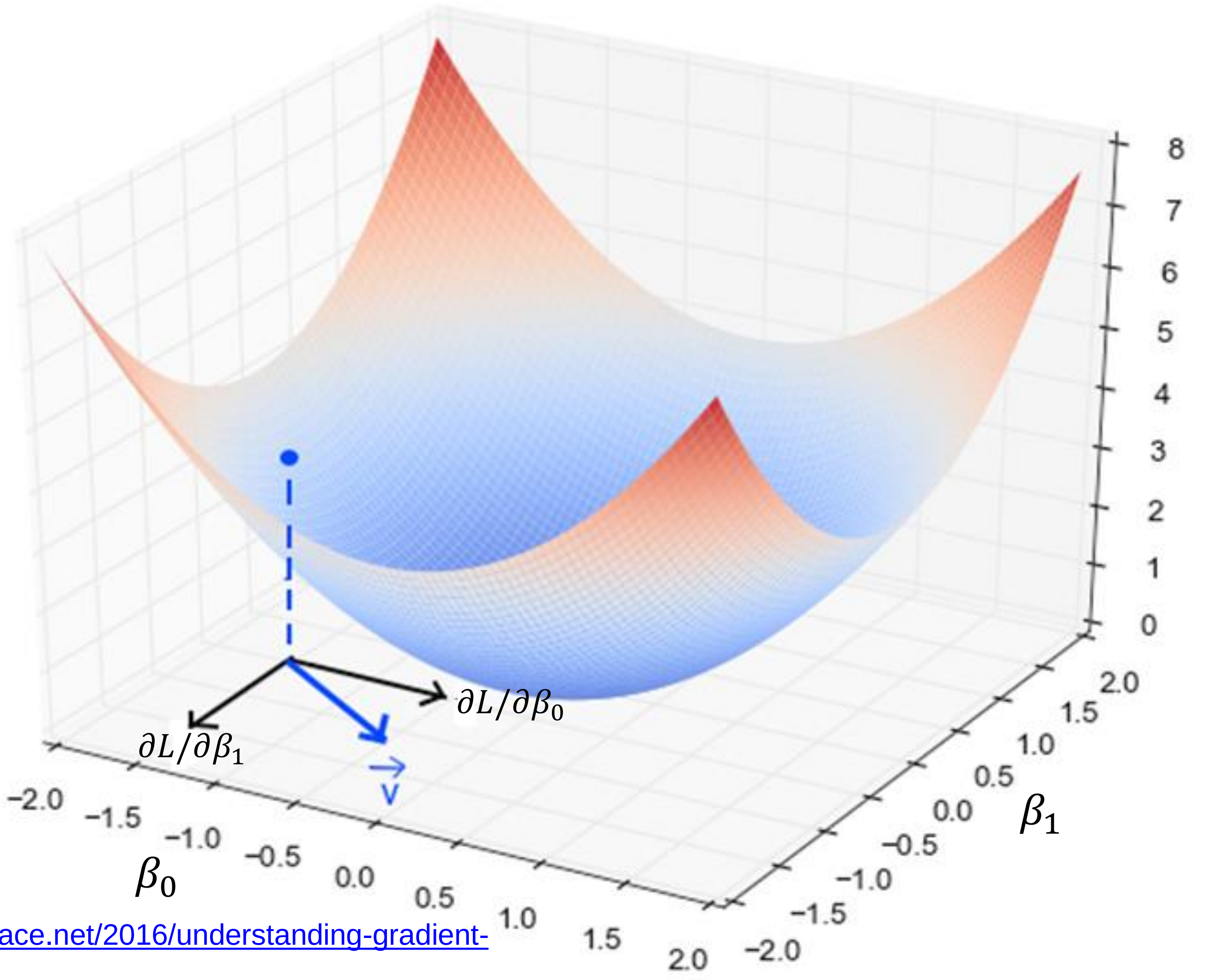
β_0 not included for simplicity





Loss function plot. Quadratic. β_0 not included for simplicity.

If β_0 is included, how will the loss function $L(\beta_0, \beta_1)$ plot look like?



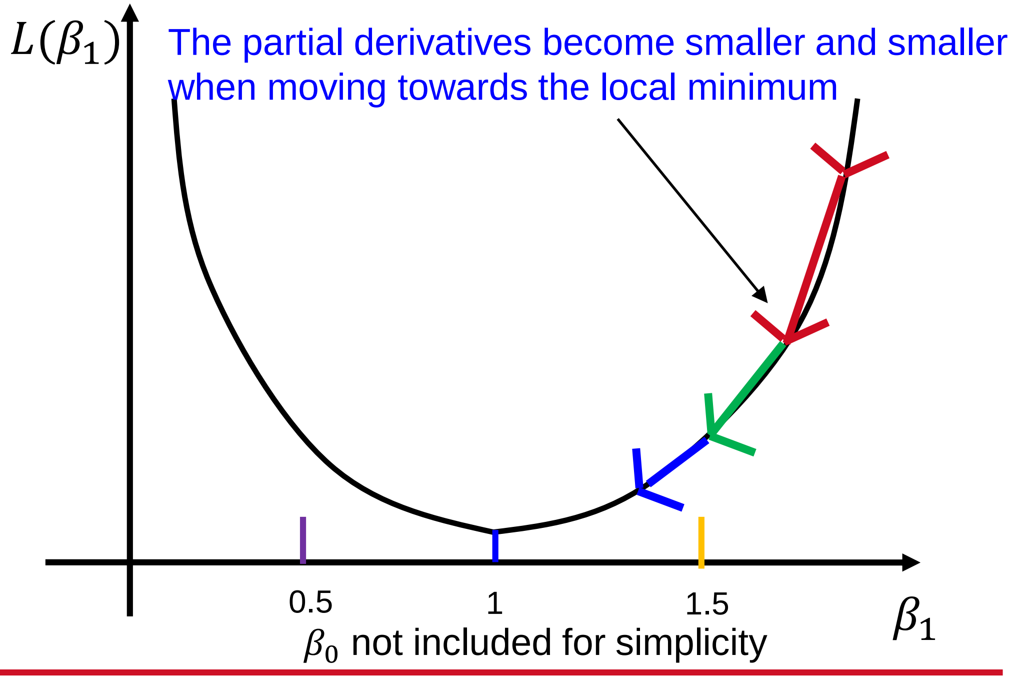
https://eli.thegreenplace.net/2016/understanding-gradient-descent

Gradient descent

- Have some random starting point for β_1;
- Keep updating β_1 to decrease the loss function $L(\beta_1)$ value;
- Repeat until achieving minimum (convergence).
- $\alpha > 0$ is called the learning rate: in empirical study, we can try many α values, and select the one generates least $L(\beta_1)$
- Gradient descent can converge to a **local minimum**

$$\beta_1 \boxed{:=} \beta_1 - \alpha \frac{\partial L(\beta_1)}{\partial \beta_1}$$

Assignment notation: keep updating β_1 based on calculations to the right hand side of this notation

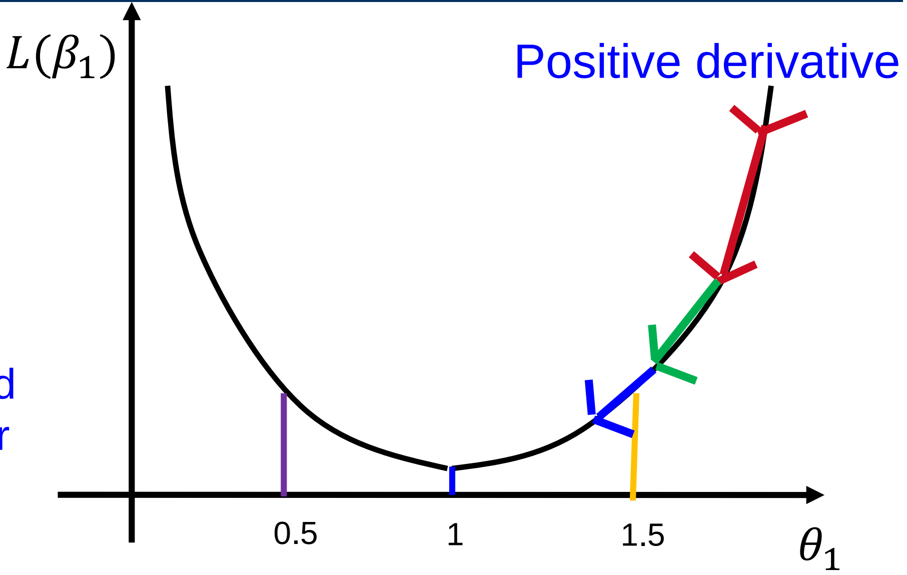

$L(\beta_1)$
The partial derivatives become smaller and smaller when moving towards the local minimum
0.5
1
1.5
β_1
β_0 not included for simplicity

If starting point of β_1 is to the right of the local minimum:

$$\frac{\partial(\mathrm{L}(\beta_1))}{\partial\beta_1} > 0$$

$$\beta_1 := \beta_1 - \alpha\frac{\partial(\mathrm{L}(\beta_1))}{\partial\beta_1}$$

β_1 is updated to be smaller and smaller

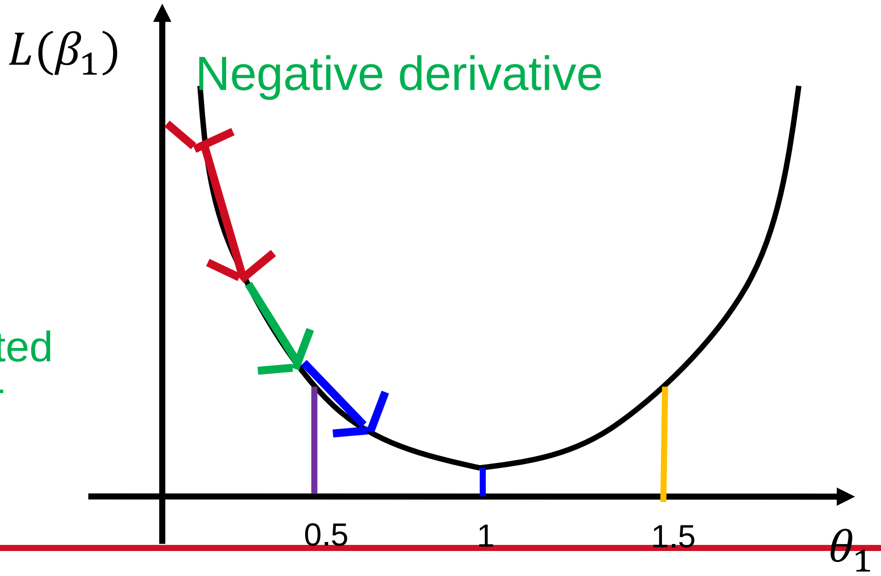


If starting point of β_1 is to the left of the local minimum:

$$\frac{\partial(\mathrm{L}(\beta_1))}{\partial\beta_1} < 0$$

$$\beta_1 := \beta_1 - \alpha\frac{\partial(\mathrm{L}(\beta_1))}{\partial\beta_1}$$

β_1 is updated to be lager and larger



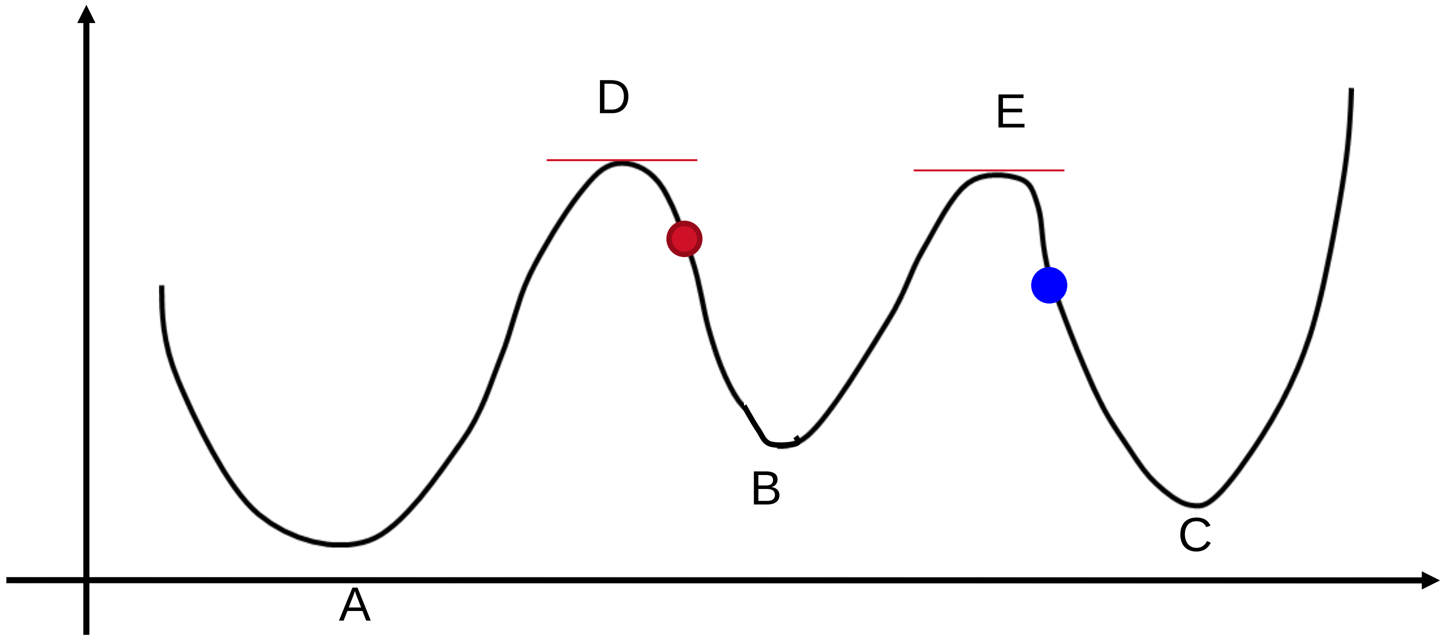
Gradient descent of linear regression

- Have some random starting points for β_0 and β_1;
- Keep updating β_0 and β_1 (simultaneously) to decrease the loss function $L(\beta_0, \beta_1)$ value;
- Repeat until achieving minimum (convergence).

$$L(\beta_0, \beta_1) = \frac{1}{2N}\sum_{n=1}^{N}(f(\boldsymbol{x}_n, \boldsymbol{\beta}) - t_n)^2 = \frac{1}{2N}\sum_{n=1}^{N}(\beta_0 + \beta_1 x_{n1} - t_n)^2$$

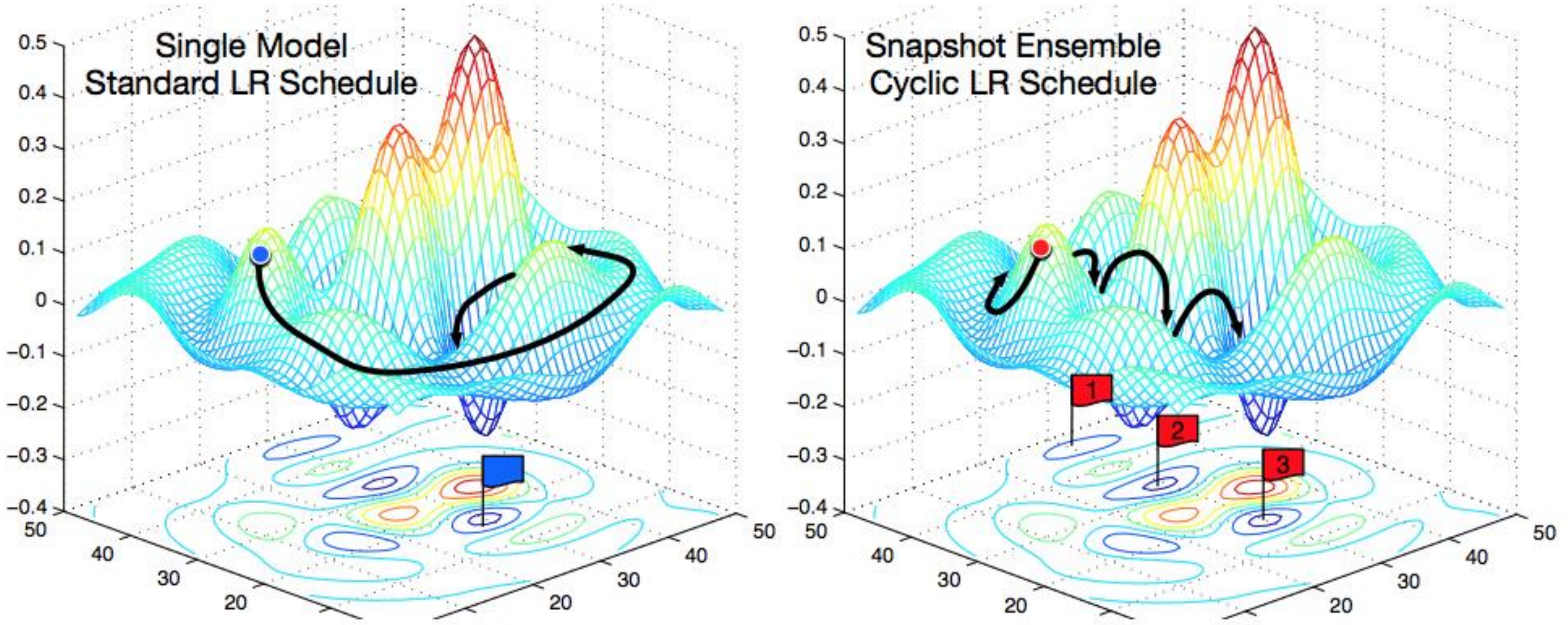
$$\left.\begin{aligned} \beta_0 &\coloneqq \beta_0 - \alpha\frac{\partial L(\beta_0, \beta_1)}{\partial \beta_0} = \beta_0 - \alpha\frac{1}{N}\sum_{n=1}^{N}(\beta_0 + \beta_1 x_{n1} - t_n) \\ \beta_1 &\coloneqq \beta_1 - \alpha\frac{\partial L(\beta_0, \beta_1)}{\partial \beta_1} = \beta_1 - \alpha\frac{1}{N}\sum_{n=1}^{N}(\beta_0 + \beta_1 x_{n1} - t_n)x_{n1} \end{aligned}\right\} \text{Update simultaneously}$$

Why local minimum?



- If the starting point is the red dot, then gradient descent can only converge to local minimum B
- If the starting point is the blue dot, then gradient descent can only converge to local minimum C
- The derivatives at D and E are 0
- The derivatives at A, B or C are also 0

Why local minimum?



https://hackernoon.com/@muhia

Linear Regression with Multiple Features

THE UNIVERSITY OF SYDNEY

Multiple Features

Number (x_1)	Nearest (x_2)	Office (x_3)	Enrolment (x_4)	Income (x_5)	Distance (x_6)	Margin (t)
3203	4.2	54.9	8.0	40	4.3	55.5
2810	2.8	49.6	17.5	38	23.0	33.8
2890	2.4	25.4	20.0	38	4.2	49.0
3422	3.3	43.4	15.5	41	19.4	31.9
2687	0.9	67.8	15.5	46	11.0	57.4
3759	2.9	63.5	19.0	36	17.3	49.0
2341	2.3	58	23.0	31	11.8	46.0
3021	1.7	57.2	8.5	45	8.8	50.2

N: number of training examples

$\boldsymbol{d = 6}$

d: number of features

$\mathbf{x}$**:** "input" variable; d **features** $\mathbf{x} = (x_1, x_2, \dots, x_d)^T \in \mathbb{R}^{\boldsymbol{d}}$

t**:** "output" variable; "target" variable. We consider a single output

For data set, we use notation

$x_{nj} \longrightarrow$ n_{th} training example of j_{th} feature $\longrightarrow$ $x_{32} = 2.4$

For this example, the linear model is

$$f(\mathbf{x}, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6$$

Matrix Representation

In general with d features

$$f(\mathbf{x}, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_d\, x_d$$

Define a special feature $x_0 = 1$, always taking value 1

So, new feature variable of a vector of d dimension

$$\mathbf{x} = (\boxed{x_0}, x_1, x_2, \dots, x_d)^T \in \mathbb{R}^{d+1}$$

Think about why this T

Collect all the parameter into a vector as $\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_d \end{bmatrix} \Longrightarrow f(\mathbf{x}, \boldsymbol{\beta}) = \mathbf{x}^T \boldsymbol{\beta}$

THE UNIVERSITY OF SYDNEY

Multiple features loss function

$$f(\mathbf{x}, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_d \, x_d$$

$$f(\mathbf{x}, \boldsymbol{\beta}) = \mathbf{x}^T \boldsymbol{\beta}$$

Consider an input feature data $\mathbf{x}_n = (x_{n0}, x_{n1}, x_{n2}, \dots, x_{nd})$ and its corresponding output (or target) t_n. The squared model error is

$$e_n^2 = (t_n - f(\mathbf{x}_n, \boldsymbol{\beta}))^2 = (t_n - \mathbf{x}_n^T \boldsymbol{\beta})^{\mathbf{2}}$$

The overall “mean” error is

$$L(\boldsymbol{\beta}) = \frac{1}{2N} \sum_{n=1}^{N} e_n^2 = \frac{1}{2N} \sum_{n=1}^{N} (t_n - \mathbf{x}_n^T \boldsymbol{\beta})^{\mathbf{2}}$$

Note $\frac{1}{2}$ here is for mathematical convenience

THE UNIVERSITY OF SYDNEY

Multiple features loss function

Another way to write the loss function

Denote

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{pmatrix} = \begin{pmatrix} x_{10} & x_{11} & x_{12} & \cdots & x_{1d} \\ x_{20} & x_{21} & x_{21} & \cdots & x_{2d} \\ \vdots & & \vdots & \ddots & \vdots \\ x_{N0} & x_{N1} & x_{N2} & \cdots & x_{Nd} \end{pmatrix} \in \mathbb{R}^{N\times(d+1)}, \mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix} \in \mathbb{R}^N$$

It is easy to prove that

$$L(\boldsymbol{\beta}) = \frac{1}{2N}(\mathbf{t} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{t} - \mathbf{X}\boldsymbol{\beta}) = ?$$

Question: What is the meaning of, for example, the second column vector of data matrix **X**, called the design matrix? What is the 10th row of **X**?

Closed-Form Solution: Normal equation

Vector differentiation rules: Let $\boldsymbol{x}$ and $\boldsymbol{a}$ be vectors of equal dimension and $\boldsymbol{A}$ is a matrix with column dimension the same as number of rows in $\boldsymbol{x}$.

$$\frac{d(\boldsymbol{x}^T\boldsymbol{a})}{d(\boldsymbol{x})} = \boldsymbol{a}$$

$$\frac{d(\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x})}{d(\boldsymbol{x})} = (\boldsymbol{A} + \boldsymbol{A}^T)\boldsymbol{x}$$

We need to first calculate the below 1st derivative of the loss function:

$$\frac{d\big(L(\boldsymbol{\beta})\big)}{d(\boldsymbol{\beta})} = ?$$

Then by first order condition, solving $\boldsymbol{\beta}$ when $\frac{d(L(\boldsymbol{\beta}))}{d(\boldsymbol{\beta})}$=0

Closed-Form Solution: Normal equation

Normal equation is an analytical solution:

- Compared with the gradient descent: no need to choose learning rate α and do not have run a loop
- Can be slow when d is large

$$\underset{(d+1)\times 1}{\boldsymbol{\beta}} = (\overset{(d+1)\times N}{\mathbf{X}^T}\overset{N\times(d+1)}{\mathbf{X}})^{-1}\underset{(d+1)\times N}{\mathbf{X}^T}\underset{N\times 1}{\mathbf{t}}$$

$$\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)$$

Non-invertable?

$\mathbf{X}^{\mathbf{T}}\mathbf{X}$ non-invertable?

Reason: Multiconlinearity problem or redundant features.

Rank and determinant of $\mathbf{X}^{\mathbf{T}}\mathbf{X} =$?

Solution: Drop one or more highly correlated features from the model or collect more data

Reason: The number of features is too large, e.g. ($N \ll d$).

Rank and determinant of $\mathbf{X}^{\mathbf{T}}\mathbf{X} =$?

Solution: Drop some features or collect more data;
Add "regularization“ term into the model

For real matrices $\mathbf{X}$, rank($\mathbf{X}^{\mathbf{T}}\mathbf{X}$) =rank($\mathbf{X}\mathbf{X}^{\mathbf{T}}$)=rank($\mathbf{X}$)=rank($\mathbf{X}^{\mathbf{T}}$)

Gradient descent of linear regression with multiple features

- Have some random starting points for all β_i;
- Keep updating all β_i (simultaneously) to decrease the loss function $L(\boldsymbol{\beta})$ value;
- Repeat until achieving minimum (convergence).

$$L(\boldsymbol{\beta}) = \frac{1}{2N}\sum_{n=1}^{N}(t_n - f(\mathbf{x}_n, \boldsymbol{\beta}))^2 = \frac{1}{2N}\sum_{n=1}^{N}(\beta_0 + \beta_1 x_{n1} + \beta_2\, x_{n2} + \cdots + \beta_d\, x_{nd} - t_n)^2$$

$$\left.\begin{aligned}
\beta_0 &\coloneqq \beta_0 - \alpha\frac{\partial L(\boldsymbol{\beta})}{\partial \beta_0} = \beta_0 - \alpha\frac{1}{N}\sum_{n=1}^{N}(\beta_0 + \beta_1 x_{n1} + \beta_2\, x_{n2} + \cdots + \beta_d\, x_{nd} - t_n)\color{red}{x_{n0}} \\
\beta_1 &\coloneqq \beta_1 - \alpha\frac{\partial L(\boldsymbol{\beta})}{\partial \beta_1} = \beta_1 - \alpha\frac{1}{N}\sum_{n=1}^{N}(\beta_0 + \beta_1 x_{n1} + \beta_2\, x_{n2} + \cdots + \beta_d\, x_{nd} - t_n)x_{n1} \\
&\cdots \\
\beta_d &\coloneqq \beta_d - \alpha\frac{\partial L(\boldsymbol{\beta})}{\partial \beta_d} = \beta_d - \alpha\frac{1}{N}\sum_{n=1}^{N}(\beta_0 + \beta_1 x_{n1} + \beta_2\, x_{n2} + \cdots + \beta_d\, x_{nd} - t_n)x_{nd}
\end{aligned}\right\}$$

Update simultaneously

Gradient Descent in Matrix Form

- We can write the Gradient Descent for linear regression with multiple features in a matrix form
- The matrix form looks much more simple. First define

$$\mathbf{f}(\mathbf{X},\boldsymbol{\beta}) := \begin{bmatrix} f(\mathbf{x}_1,\boldsymbol{\beta}) \\ f(\mathbf{x}_2,\boldsymbol{\beta}) \\ \vdots \\ f(\mathbf{x}_N,\boldsymbol{\beta}) \end{bmatrix}; \quad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}; \quad \text{and} \quad \frac{\partial L(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} := \begin{bmatrix} \frac{\partial L(\boldsymbol{\beta})}{\partial \beta_0} \\ \frac{\partial L(\boldsymbol{\beta})}{\partial \beta_1} \\ \vdots \\ \frac{\partial L(\boldsymbol{\beta})}{\partial \beta_d} \end{bmatrix}$$

Then it can be proved that

$$\frac{\partial L(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \frac{1}{N}\mathbf{X}^T(\mathbf{f}(\mathbf{X},\boldsymbol{\beta}) - \mathbf{t})$$

- Hence gradient descent is

$$\boldsymbol{\beta} := \boldsymbol{\beta} - \alpha\frac{\partial L(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta} - \frac{\alpha}{N}\mathbf{X}^T(\mathbf{f}(\mathbf{X},\boldsymbol{\beta}) - \mathbf{t})$$

THE UNIVERSITY OF SYDNEY

Feature Scaling

Target: transform features to be on a similar scale.

Results: faster convergence of the optimisation algorithm

Number (x_1)	Nearest (x_2)	Office (x_3)	Enrolment (x_4)	Income (x_5)	Distance (x6)	Margin (t)
3203	4.2	54.9	8.0	40	4.3	55.5
2810	2.8	49.6	17.5	38	23.0	33.8
2890	2.4	25.4	20.0	38	4.2	49.0
3422	3.3	43.4	15.5	41	19.4	31.9
2687	0.9	67.8	15.5	46	11.0	57.4
3759	2.9	63.5	19.0	36	17.3	49.0
2341	2.3	58	23.0	31	11.8	46.0
3021	1.7	57.2	8.5	45	8.8	50.2

Number (x_1): 1613 to 4214
Nearest (x_2): 0.1 to 4.2
...

Mean Normalization

$$x_{nj} = \frac{x_{nj} - \bar{x}_j}{s_j}$$

Goal: have all the features to be approximately 0 mean and 1 variance

Polynomial Regression

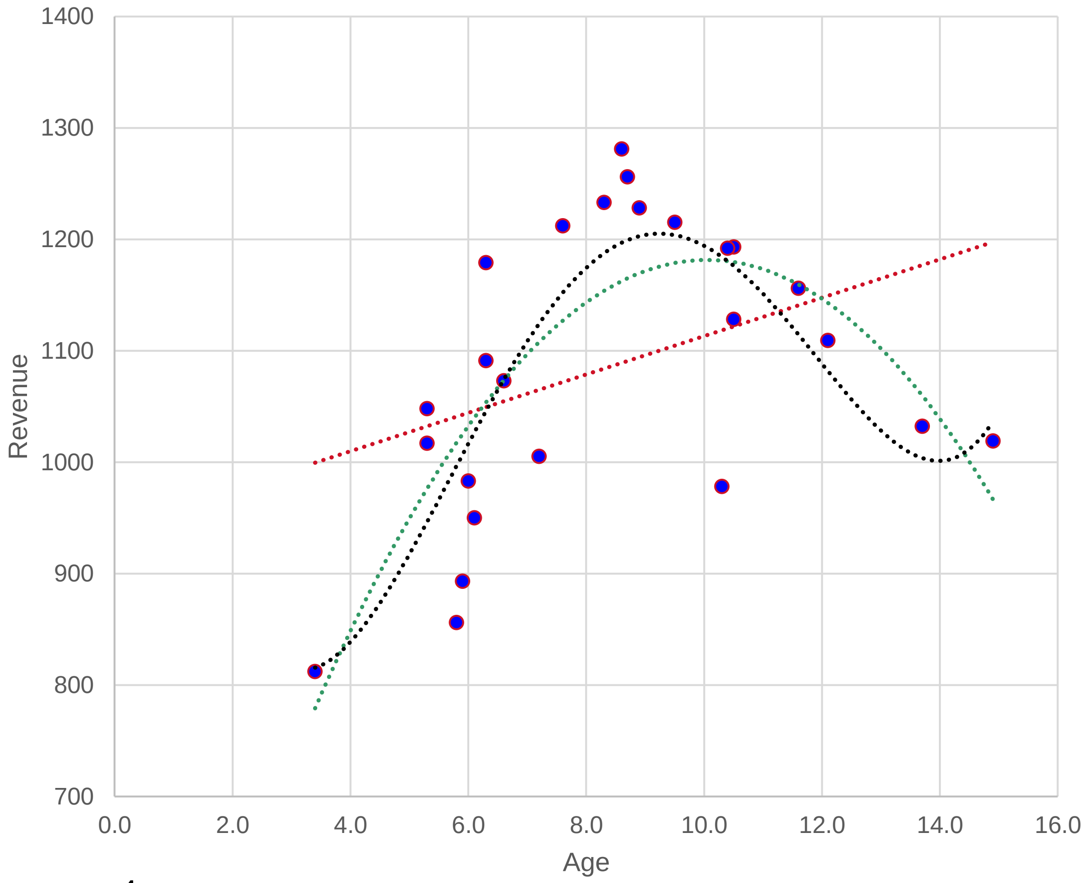
Question:
Which model is the best model?

$$f(\mathbf{x}, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2$$

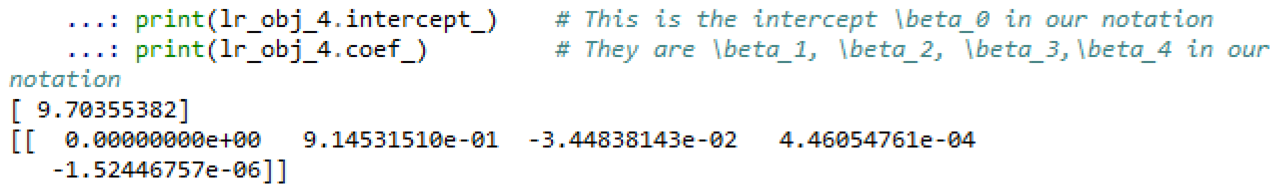
We added a quadratic feature as $x_2 = x_1^2$ constructed from the first feature; Similarly

$$f(\mathbf{x}, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3$$

$$f(\mathbf{x}, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3 + \beta_4 x_1^4$$

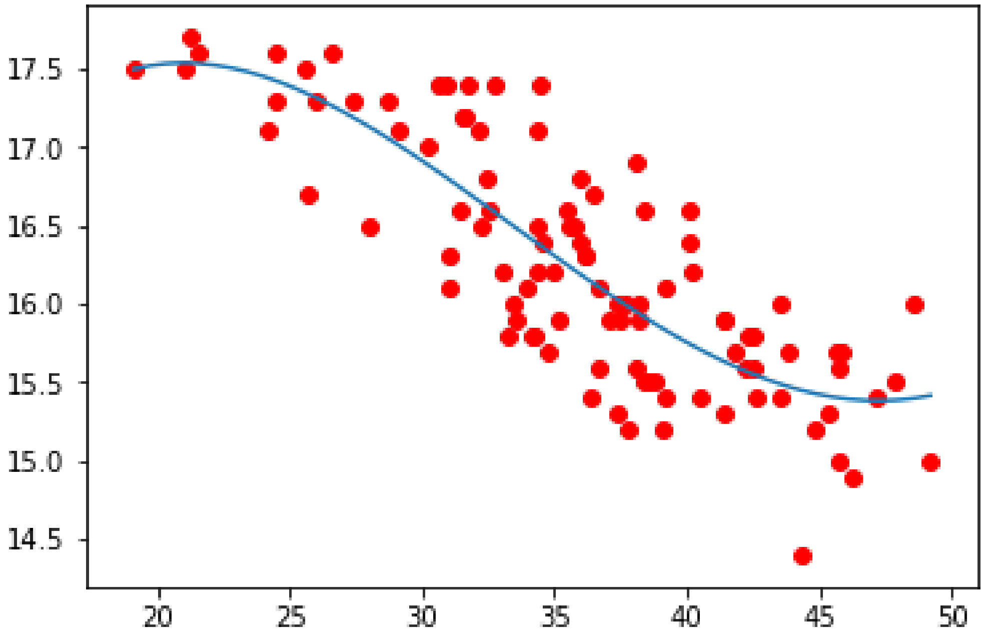


```
# fit a 4th order polynomial regression model
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(4)
poly_4 = poly.fit_transform(np.reshape(x_data, (100,1)))
poly_4 = np.asmatrix(poly_4)
lr_obj_4 = LinearRegression()
lr_obj_4.fit(poly_4, y_data)
print(lr_obj_4.intercept_)    # This is the intercept \beta_0 in our notation
print(lr_obj_4.coef_)
```



```
    ...: print(lr_obj_4.intercept_)    # This is the intercept \beta_0 in our notation
    ...: print(lr_obj_4.coef_)         # They are \beta_1, \beta_2, \beta_3,\beta_4 in our
notation
[ 9.70355382]
[[  0.00000000e+00   9.14531510e-01  -3.44838143e-02   4.46054761e-04
   -1.52446757e-06]]
```

```
# plot the fitted 4th order polynomial regression line
x_temp = np.reshape(np.linspace(np.min(x_data), np.max(x_data), 50), (50,1))
poly_temp0_4 = poly.fit_transform(np.reshape(x_temp, (50,1)))
y_temp = lr_obj.predict(poly_temp0_4)

plt.plot(x_temp,y_temp)
plt.scatter(odometer,car_price,label = "Observed Points", color = "red")
```



Is this a better model?

Model Selection

Model Selection and Assessment

Step 5b

Model Selection:

estimate the performance of different models in order to choose the (approximate) best one

Model Assessment:

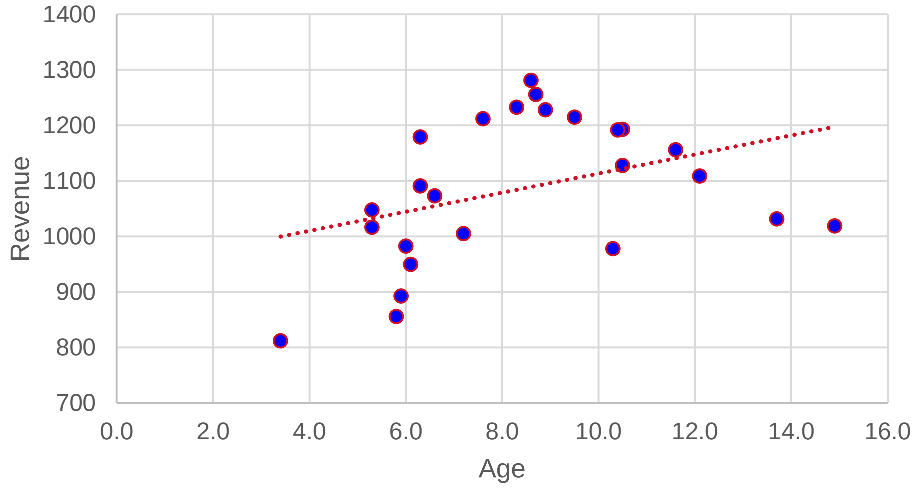
after chosen the “best” model, estimate its prediction error (generalization error) on new data. (Friedman et al., 2001).

In general, we shall divide the given dataset into **three** parts:

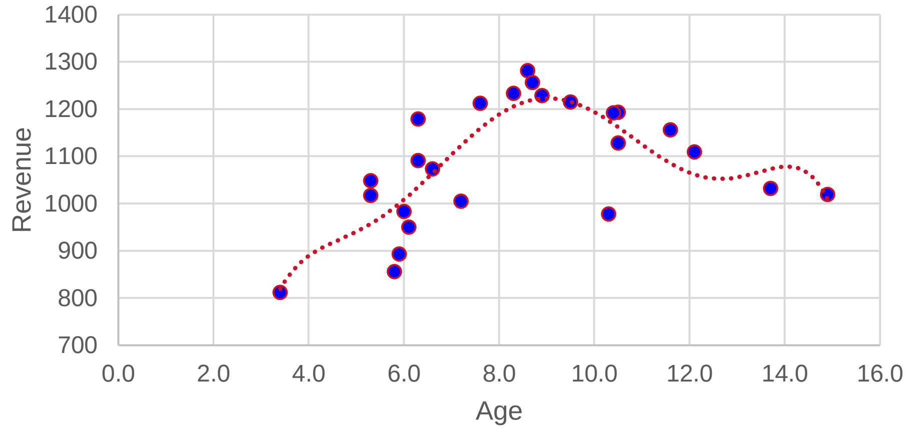
- **Training dataset** (60%): used to estimate a model (or models)
- **Validation dataset** (20%): used to select an appropriate model
- **Test dataset** (20%): used to assess the performance of the selected model. This set of data should be hidden from training and validating process.

Some academics use 50%, 25%, 25% split

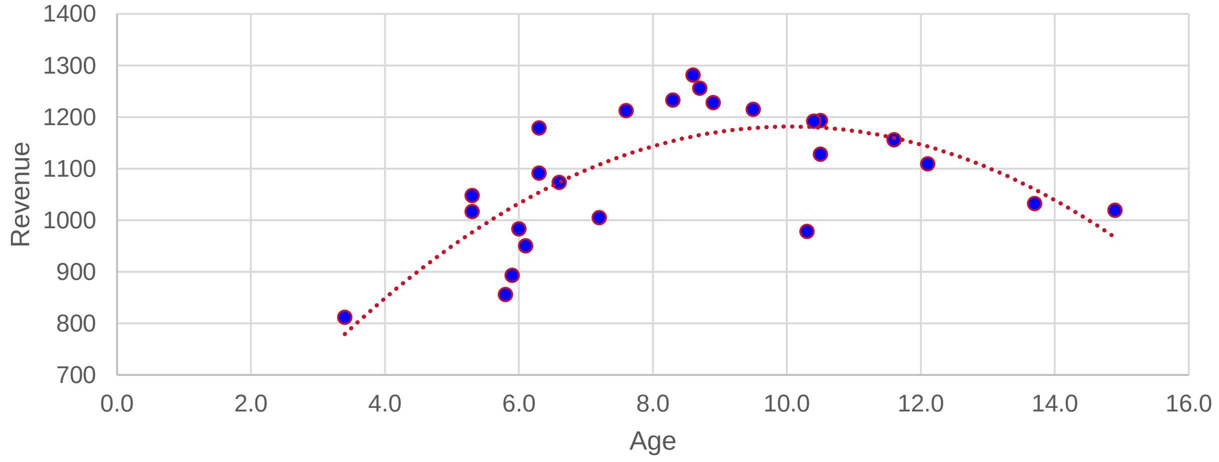
Underfitting & Overfitting



Underfitting: High bias, low variance



Overfitting: High variance, low bias



Best model

THE UNIVERSITY OF SYDNEY

Underfitting and Overfitting

Why is overfitting bad?

- Low training error, high generalization error
- Poor predictive performance
- Overreacts to minor fluctuations in training data

How to address overfitting:

❑ **Drop some features**

  ➢ Model selection algorithm

  ➢ Manually select features to keep

❑ **Regularization**

  ➢ Keep all features, reduce the magnitude/values of parameters

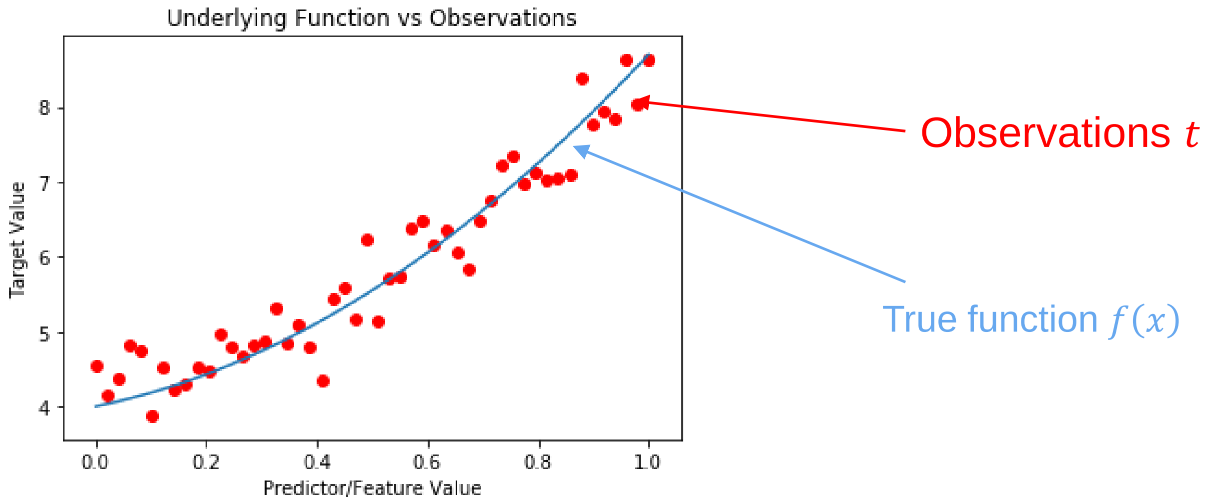
Bias and Variance

Mean Squared Error (MSE)= (Bias)^2 +Variance+Irreducible Error   Why?

Suppose that there is a function with noise $t = f(\mathbf{x}) + \varepsilon$,
Where $f(\mathbf{x})$ is our true function, and ε has zero mean and variance σ^2

Target: we want to find a function $f(\mathbf{x}, \boldsymbol{\beta})$ that **approximates** the true function $f(\mathbf{x})$ as well as possible, by means of some learning algorithm, e.g. linear regression by minimizing $(t - f(\mathbf{x}, \boldsymbol{\beta}))^2$

In reality, we can observe t, while cannot observe true function $f(\mathbf{x})$



THE UNIVERSITY OF SYDNEY

Bias and Variance Decomposition

Can we **approximates** the true function $f(\mathbf{x})$ perfectly? No.

Since the t contain noise ε, which means we must be prepared to accept an irreducible error in any algorithm/function we implemented.

$$E[(t - f(\mathbf{x}, \boldsymbol{\beta}))^2] = \text{Bias}^2[f(\mathbf{x}, \boldsymbol{\beta})] + \text{Var}[f(\mathbf{x}, \boldsymbol{\beta})] + \sigma^2$$

True function $f(\mathbf{x})$

How to calculate?

Bias reflects the error between average of our estimate differs from the true function

$$\text{Bias}^2[f(\mathbf{x}, \boldsymbol{\beta})] = (\boxed{f(\mathbf{x})} - \boxed{\mathbb{E}[f(\mathbf{x}, \boldsymbol{\beta})]})^2$$

Variance reflects the expected squared deviation of $f(\mathbf{x}, \boldsymbol{\beta})$ around its mean

$$\text{Var}[f(\mathbf{x}, \boldsymbol{\beta})] = \mathbb{E}[(f(\mathbf{x}, \boldsymbol{\beta}) - \mathbb{E}[f(\mathbf{x}, \boldsymbol{\beta})])^2]$$

All expections ($\mathbb{E}$) are taken with respect to data set from the model parameter is calculated

Training, Validation and Test Sets

$$L(\boldsymbol{\beta}) = \frac{1}{2N}\sum_{n=1}^{N}(t_n - f(\mathbf{x}_n, \boldsymbol{\beta}))^2$$

This loss function issued for training, validation and test sets respectively

Odometer (x)	Price (t)
37.4	16.0
44.8	15.2
45.8	15.0
30.9	17.4
31.7	17.4
34.0	16.1
45.9	15.7
19.1	17.5
40.1	16.6
40.2	16.2

	Cost function
Training set: 60%	$L_{train}(\boldsymbol{\beta})$
Validation set: 20%	$L_v(\boldsymbol{\beta})$
Test set: 20%	$L_{test}(\boldsymbol{\beta})$

Polynomial Order Selection

Training set	Validation set	Test
Estimate the parameters	Select the best model	Estimate the generalization error

Which one to use?

$$f(\mathbf{x},\boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2$$

$$f(\mathbf{x},\boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3$$

$$f(\mathbf{x},\boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3 + \beta_4 x_1^4$$

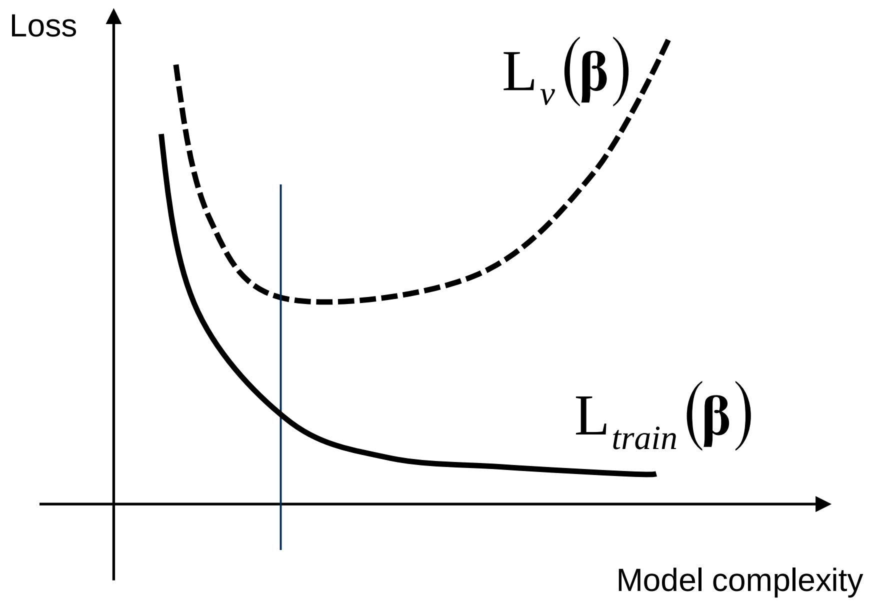
Cannot use training set to select the best model. Not a fair competition.

Since the model minimize this training data set, not necessarily minimize the new date sets.

- ❑ Optimize the parameters $\boldsymbol{\theta}$ employing the training set for each polynomial degree
- ❑ Find the polynomial degree d with the smallest error using the validation set
- ❑ Estimate the generalization error using the test set.

Diagnosing Learning Curve

Learning Curve



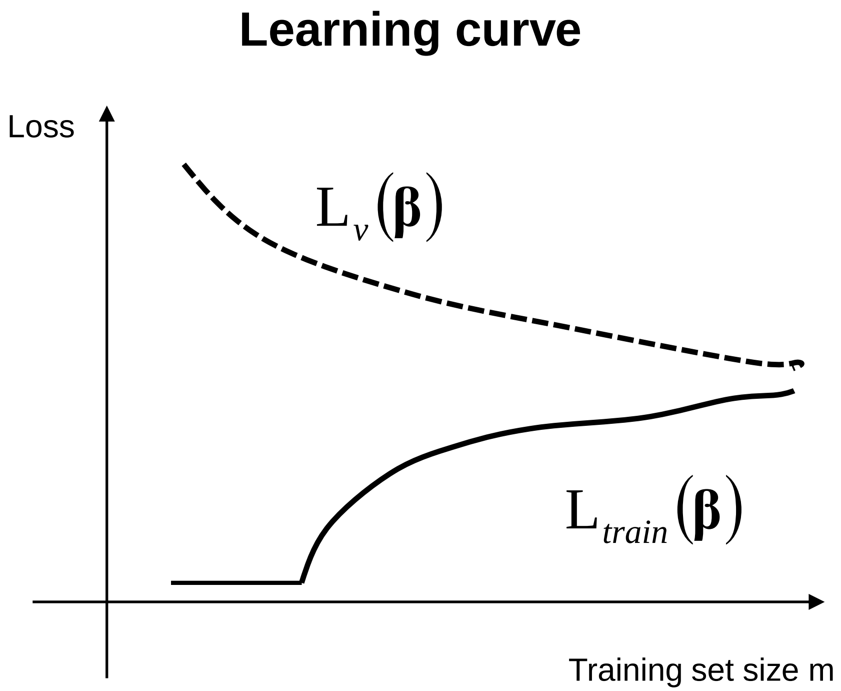
- ❑ Suppose your training error is low, while validation/test error is high.
  Underfitting or overfitting problem?

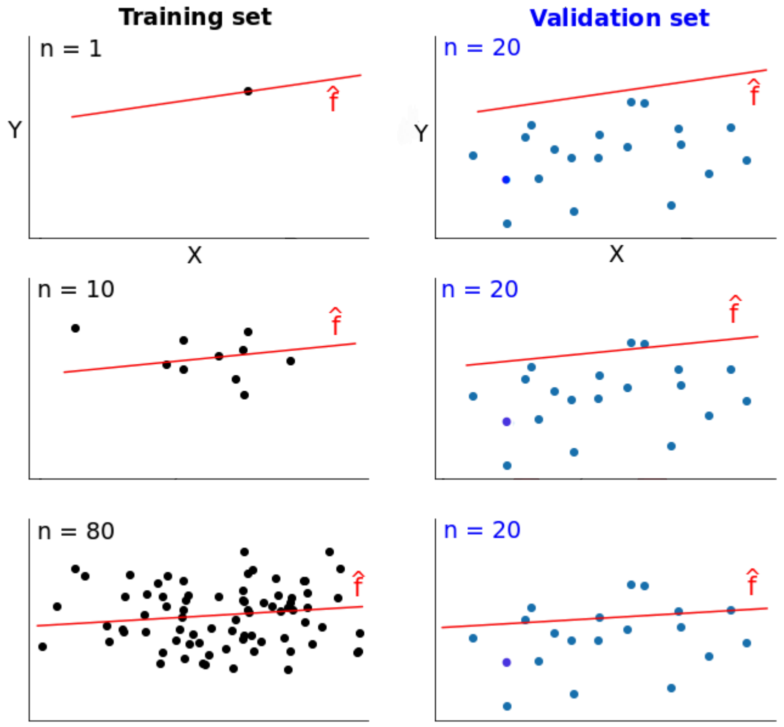
- ❑ Suppose your training error is high, while validation/test error is also high.
  Underfitting or overfitting problem?

Underfitting: training error is high, validation error is slightly > training error;
Overfitting: training error is low, validation error is significantly > training error.

THE UNIVERSITY OF SYDNEY

Diagnosing Learning Curve





Why is this?

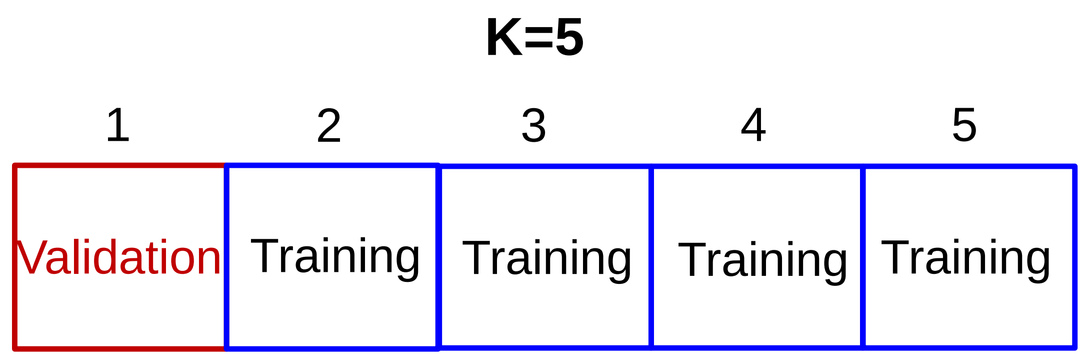
The impact of training set size on loss function

Cross Validation

K-Fold Cross-Validation

- ❑ If we had enough data, we would set aside a validation set and use it to assess the performance of our prediction model
- ❑ However, data are often scarce, this is usually not possible
- ❑ Particularly when we do not have sufficient labelled data
- ❑ K-fold cross-validation (CV) uses part of the available data to fit the model, and a different part to test it, then iterate/repeat this process

5-fold Cross-Validation

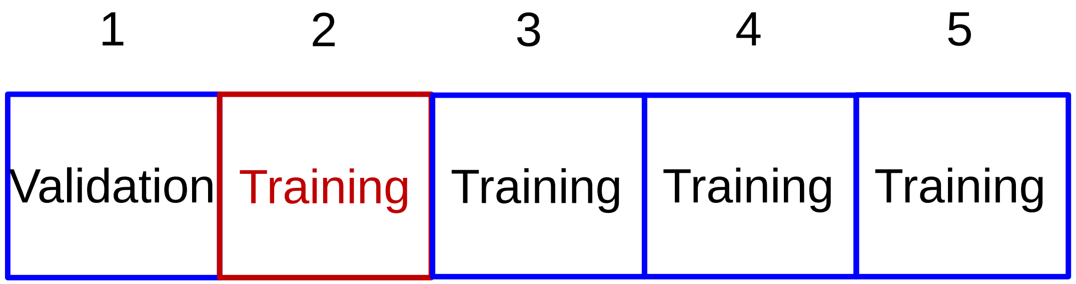


The original sample is **randomly** partitioned into K equal size subsamples; For each iteration, of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining K-1 subsamples are used as training data.

Specifically, **at iteration 1**:
Data set **1** is chosen as validation set, and **2, 3, 4, 5** are chosen as training sets. Estimate the parameters $\boldsymbol{\beta}_1$ using training sets and calculate the validation error $\mathrm{L}_{\mathrm{v}}(\boldsymbol{\beta}_1)$

At iteration 2:

Data set **2** is chosen as validation set, and **1, 3, 4, 5** (K-1 sets) are chosen as training sets. Estimate the parameters $\boldsymbol{\beta}_2$ using training sets and calculate the validation error $\mathrm{L_v}(\boldsymbol{\beta}_2)$

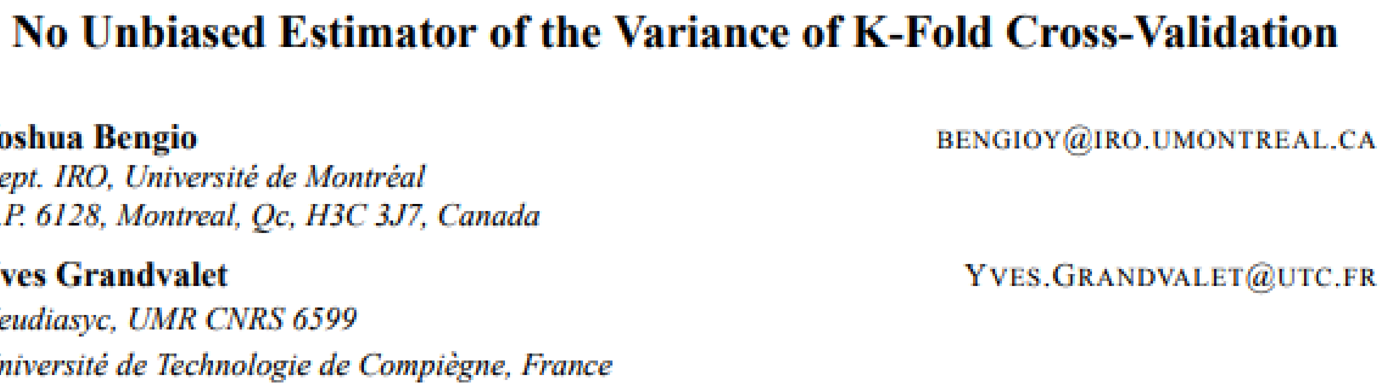


Repeat until iteration K=5. Estimate the parameters $\boldsymbol{\beta}_5$ using training sets and calculate the validation error $\mathrm{L_v}(\boldsymbol{\beta}_5)$

Output mean validation error $(\mathrm{L_v}(\boldsymbol{\beta}_1) + \mathrm{L_v}(\boldsymbol{\beta}_2)+\ldots+\mathrm{L_v}(\boldsymbol{\beta}_5))/5$ on validation sets and select the model that generates the least error.

Cross-Validation potential issues:

- ❑ Computational cost
  - ➢ you must train each model K times.
  - ➢ The K training sets (and hence the trained models) are highly correlated (see, e.g., Bengio Y & Grandvalet Y (2004)



No Unbiased Estimator of the Variance of K-Fold Cross-Validation

Yoshua Bengio BENGIOY@IRO.UMONTREAL.CA
Dept. IRO, Université de Montréal
C.P. 6128, Montreal, Qc, H3C 3J7, Canada

Yves Grandvalet YVES.GRANDVALET@UTC.FR
Heudiasyc, UMR CNRS 6599
Université de Technologie de Compiègne, France

Scikit-learn Workflow

See `Lecture02_Example01.py` and `Lecture02_Example02.py`

- Python `scikit-learn` package provides facilities for most popular machine learning algorithms
- The best way to learn how to use `scikit-learn` functionalities to learn from examples and read user guide
- Workflow:
  - A typical machine learning task starts with data preparation: For example, loading data from database/files (e.g. using pandas); data cleaning; feature extraction, feature scaling and dimensionality reduction etc; some of these can be done with `scikit-learn`, some rely on other packages
  - Following data preparation there will be a step to define a machine learning model, for example, linear regression etc.
  - `scikit-learn` introduces the concept of pipeline that chains all the steps in a linear sequence and automates the cross-validation
  - Read examples here https://machinelearningmastery.com/automate-machine-learning-workflows-pipelines-python-scikit-learn/