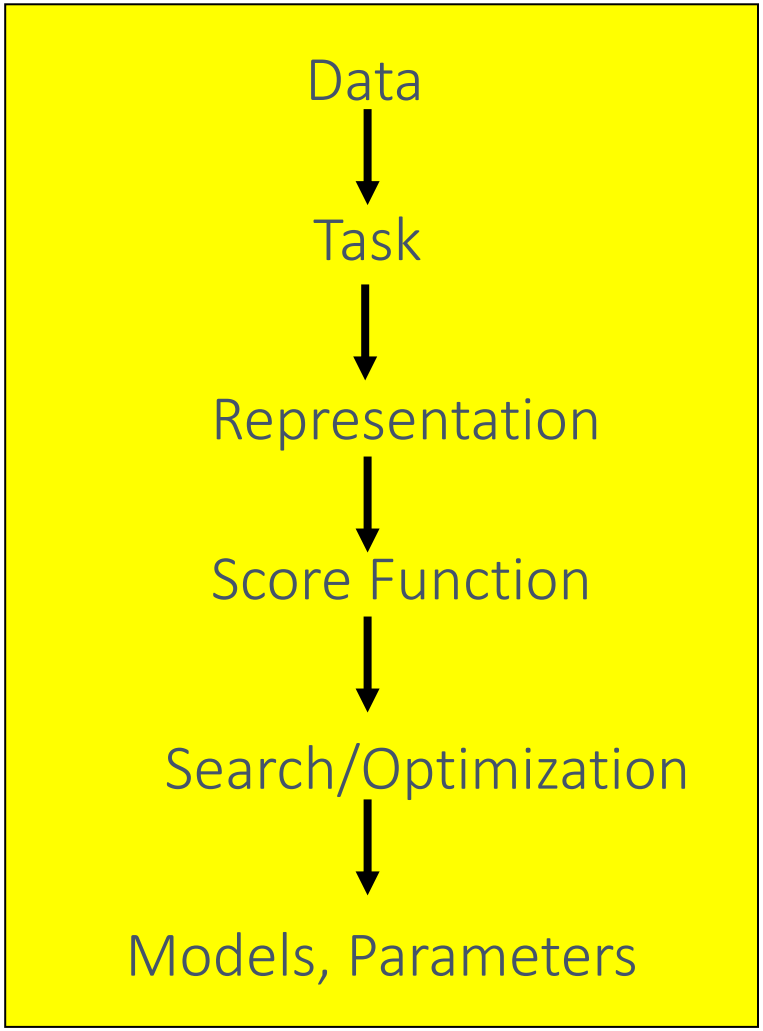
UVA CS 4774: Machine Learning

Lecture 3: Linear Regression Basics

Dr. Yanjun Qi

University of Virginia
Department of Computer Science

Machine Learning in a Nutshell



Data

Task

Representation

Score Function

Search/Optimization

Models, Parameters

ML grew out of work in AI

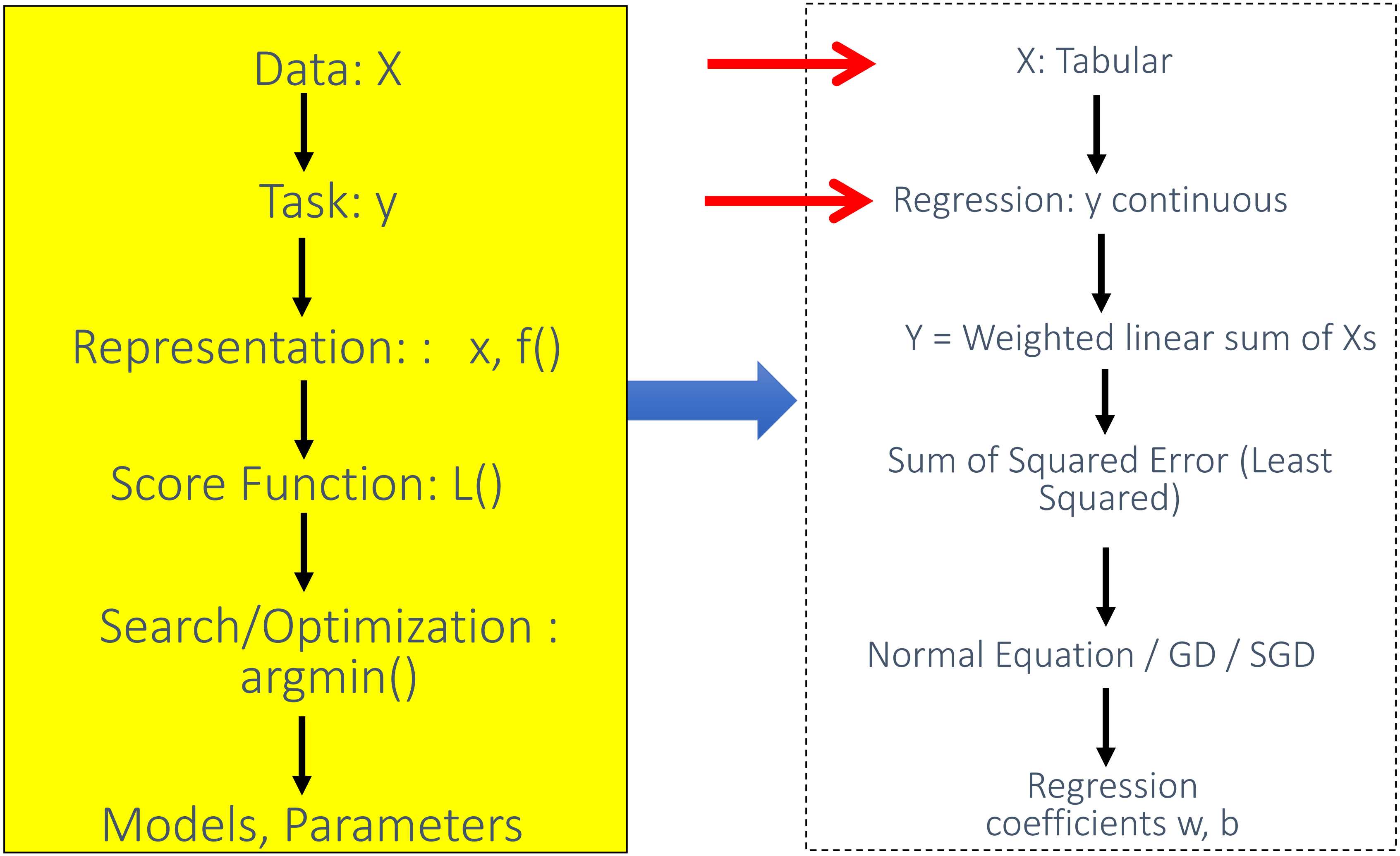
Optimize a performance criterion using example data or past experience,

Aiming to generalize to unseen data

Rough Sectioning of this Course

- 1. Basic Supervised Regression + on Tabular Data
- 2. Basic Deep Learning + on 2D Imaging Data
- 3. Generative and Deep + on 1D Sequence Text Data
- 4. Advanced Supervised learning + on Tabular Data
- 5. Not Supervised + Mostly on Tabular Data

Today: Multivariate Linear Regression in a Nutshell



Data: X

Task: y

Representation: : x, f()

Score Function: L()

Search/Optimization : argmin()

Models, Parameters

X: Tabular

Regression: y continuous

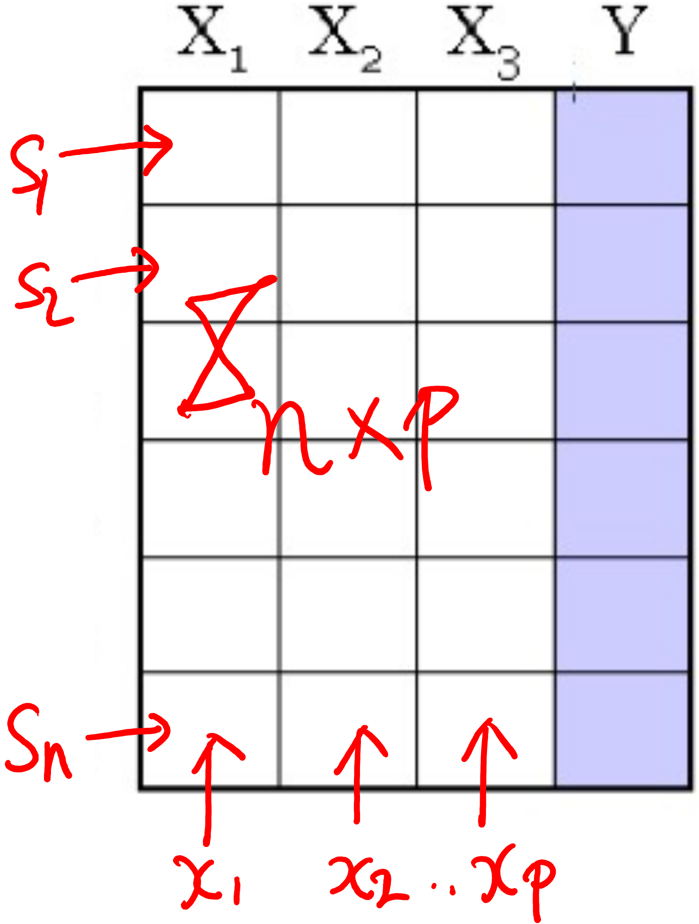
Y = Weighted linear sum of Xs

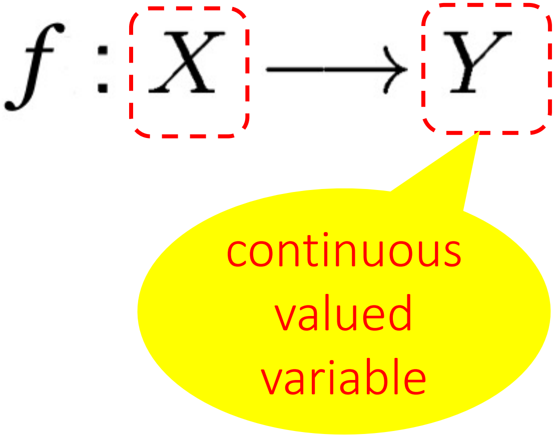
Sum of Squared Error (Least Squared)

Normal Equation / GD / SGD

Regression coefficients w, b

Tabular Dataset for regression

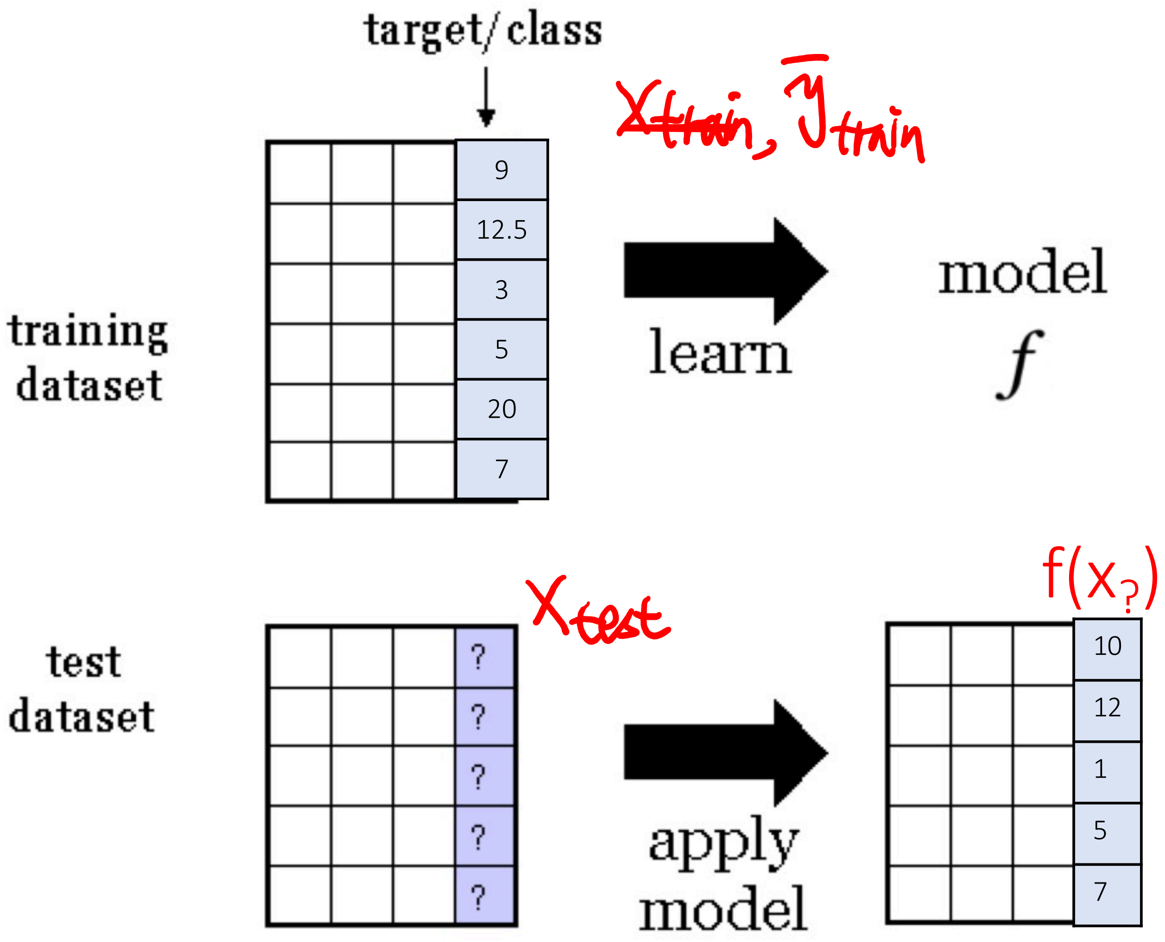




$$f : \boxed{X} \longrightarrow \boxed{Y}$$

- **Data**/*points/instances/examples/samples/records*: [rows]
- **Features**/*attributes/dimensions/independent variables/covariates/predictors/regressors*: [columns, except the last]
- **Target**/*outcome/response/label/dependent variable*: special column to be predicted [last column]

SUPERVISED Regression



Training dataset consists of input-output

pairs

- Target Y: continuous target variable

$$\mathbf{X}_{train} = \begin{bmatrix} -- & \mathbf{x}_1^T & -- \\ -- & \mathbf{x}_2^T & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n^T & -- \end{bmatrix} \qquad \vec{y}_{train} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

training dataset

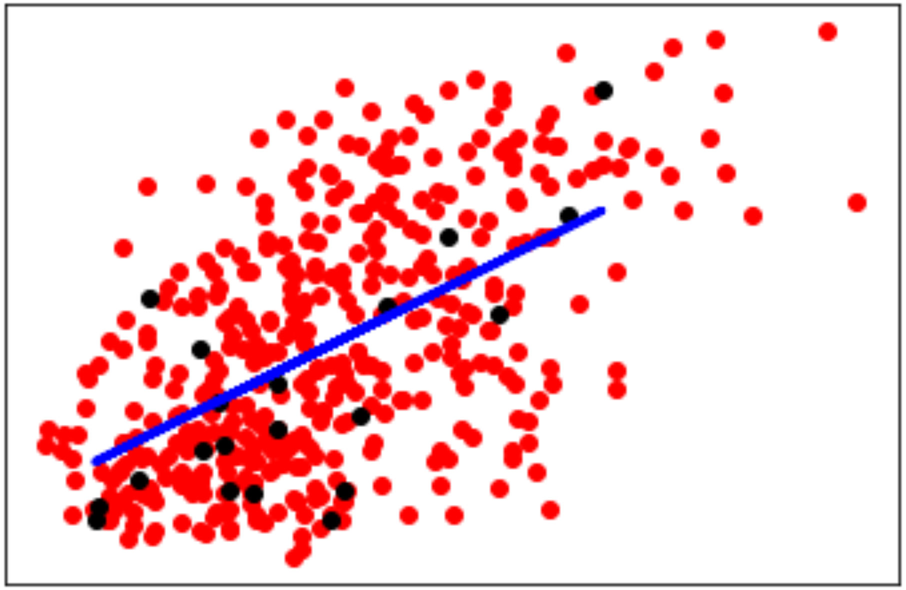
test dataset

$$\mathbf{X}_{test} = \begin{bmatrix} -- & \mathbf{x}_{n+1}^T & -- \\ -- & \mathbf{x}_{n+2}^T & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_{n+m}^T & -- \end{bmatrix} \quad \left\{ \begin{array}{l} f(X_{test}) \\ \vec{y}_{test} \end{array} \right. = \begin{bmatrix} y_{n+1} \\ y_{n+2} \\ \vdots \\ y_{n+m} \end{bmatrix}$$

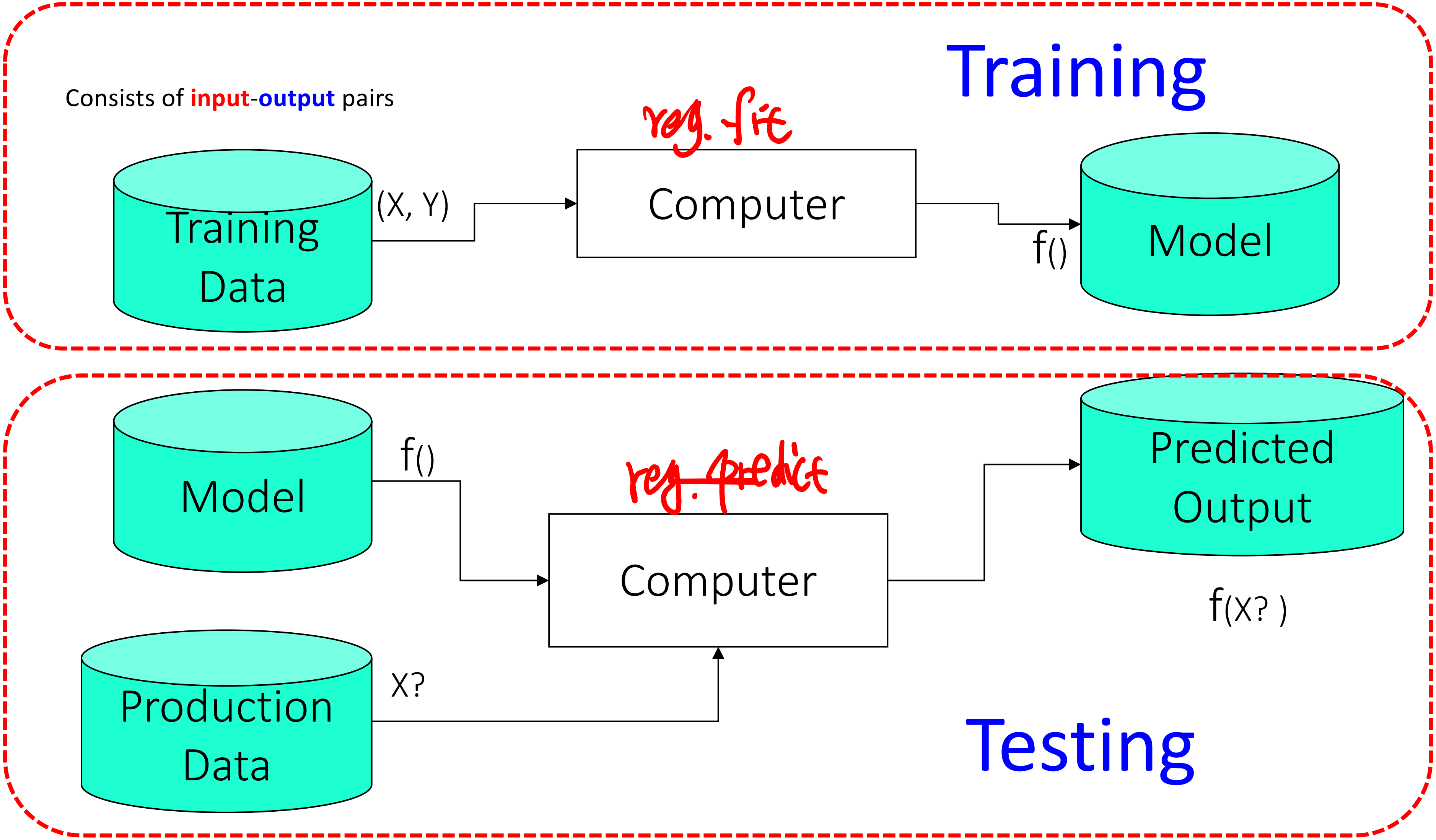
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

https://colab.research.google.com/drive/1mI-PFB2J1UVIp6JjLhQKEPirPZlQnDH7?usp=sharing

I will code-run through: Scikit-Learn Linear Regression Example
Adapted from: https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html



Two Modes of Machine Learning



Training

Consists of **input**-**output** pairs

Training Data

(X, Y)

reg.fit

Computer

f()

Model

Model

f()

reg.predict

Computer

Predicted Output

f(X?)

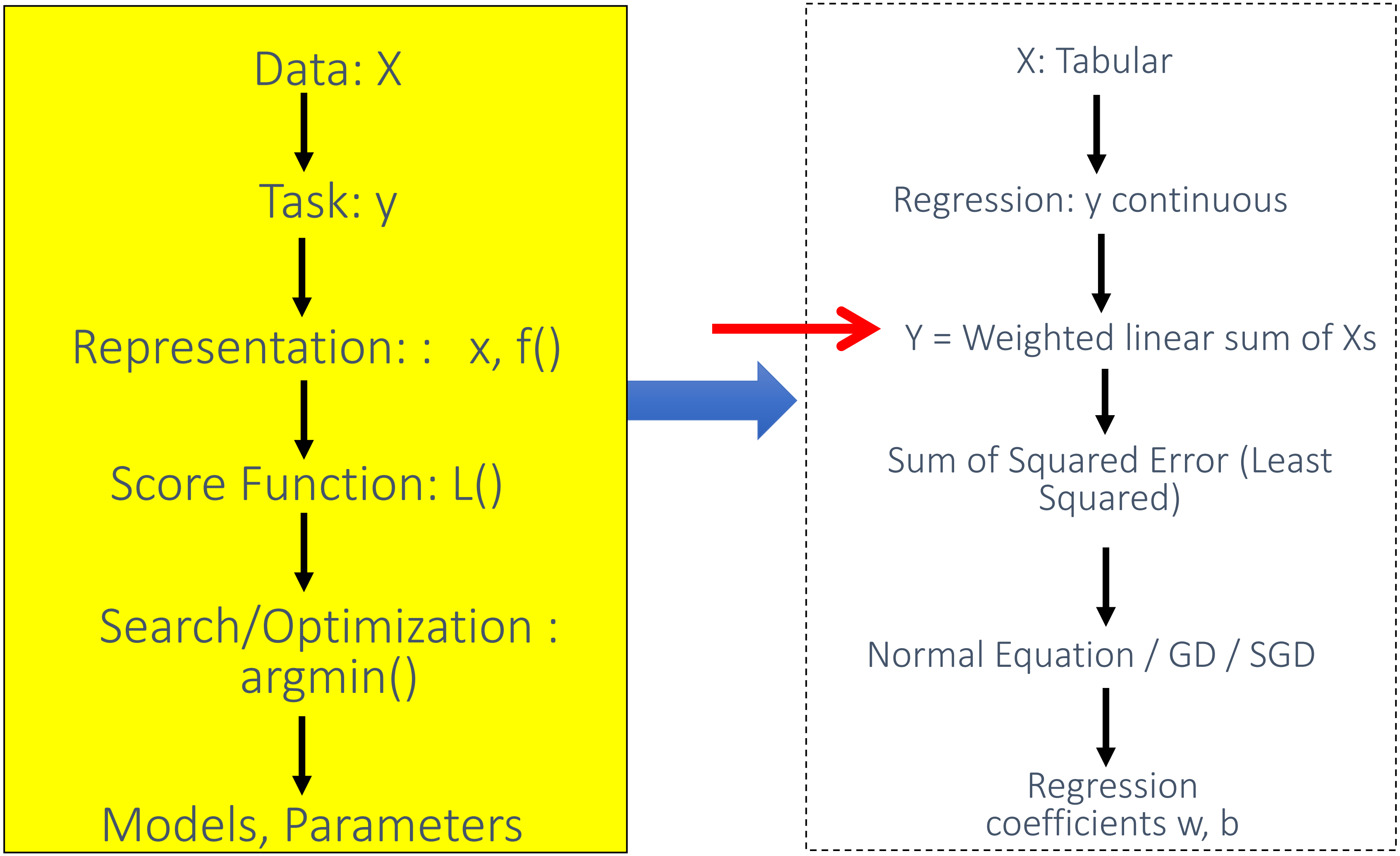
Production Data

X?

Testing

Thank You

Today: Multivariate Linear Regression in a Nutshell



Data: X

Task: y

Representation: : x, f()

Score Function: L()

Search/Optimization : argmin()

Models, Parameters

X: Tabular

Regression: y continuous

Y = Weighted linear sum of Xs

Sum of Squared Error (Least Squared)

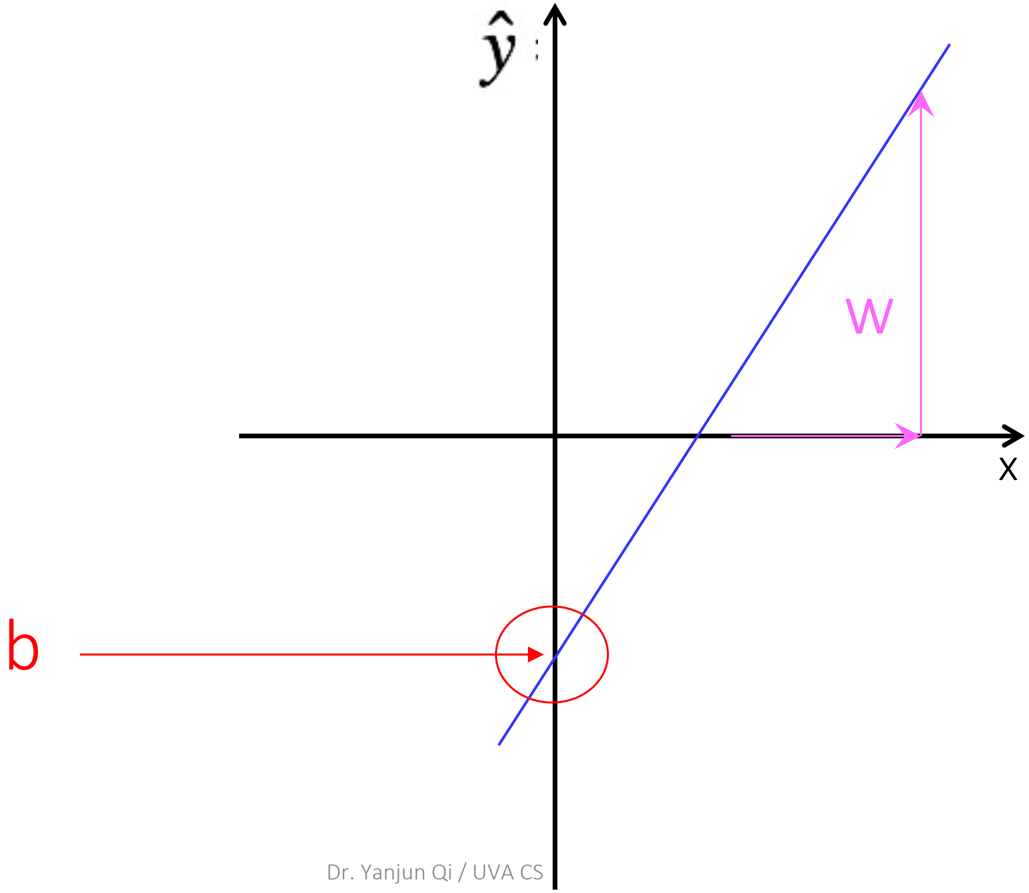
Normal Equation / GD / SGD

Regression coefficients w, b

Review: f(x) is Linear when X is single variable

- *f(x)=wx+b?*

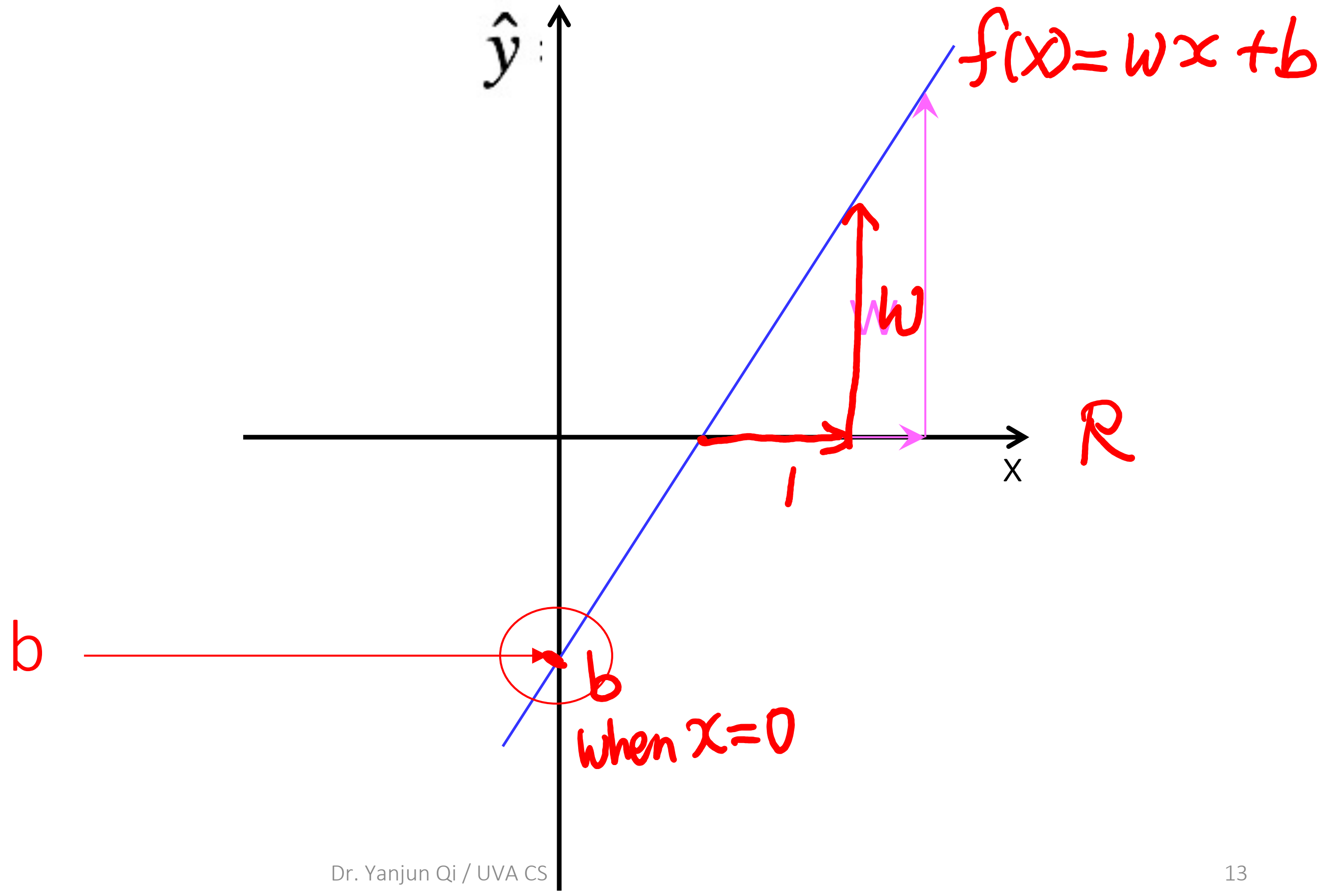
A slope of 2 (i.e. w=2) means that every 1-unit change in X yields a 2-unit change in Y.



Review: f(x) is Linear when X is single variable

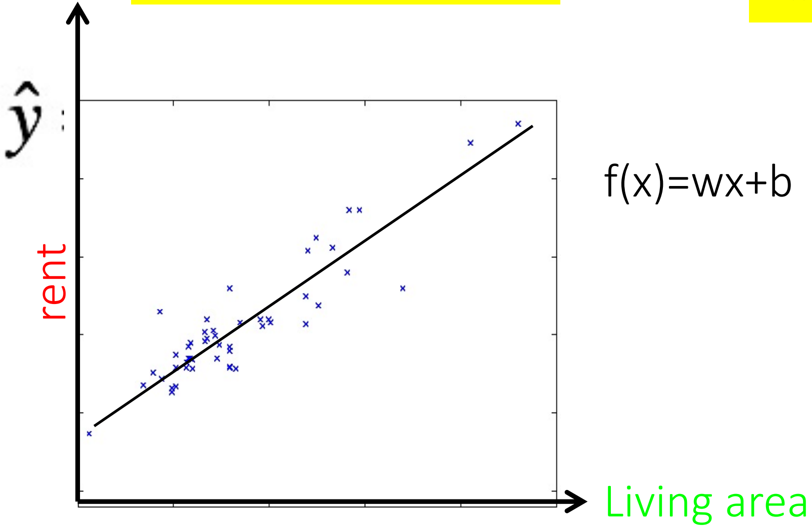
- *f(x)=wx+b?*

A slope of 2 (i.e. w=2) means that every 1-unit change in X yields a 2-unit change in Y.



Linear Regression: Y as Weighted linear sum of Xs

e.g., Living area as X



f(x)=wx+b

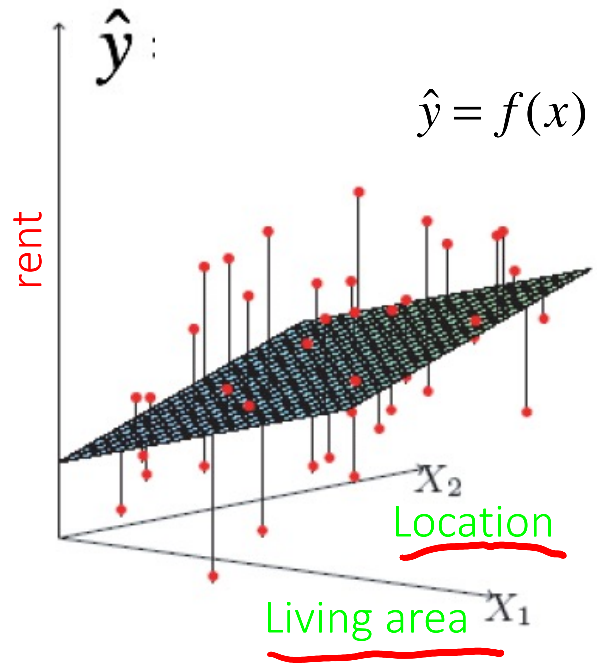
1D case ($\mathcal{X} = \mathbb{R}$): a line

---

e.g., (Living area, Location) as X

Weighted sum of x_i

$$\hat{y} = f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$



$\mathcal{X} = \mathbb{R}^2$: a plane

Linear Supervised Regression

$$\boxed{f} : X \longrightarrow Y$$

$$f(x) = \theta_0 + \theta_1 x_1 + \dots\dots + \theta_p x_p$$

Linear Regression Models

Linear Regression: Y as Weighted linear sum of Xs

$$\hat{y} = f(x) = \theta_0 + \theta_1 \underline{x_1} + \theta_2 \underline{x_2}$$

=> Features x_i: $\vec{x} \in R^p$;

e.g., Living area, distance to campus, # bedroom ...

=> Target y:

e.g., Rent (a continuous variable)

A Concise Notation: via Vector/Matrix Product

- Represent each data sample **x** as a column vector, plus a pseudo feature
  - We add a pseudo "feature" $x_0=1$ (this is the intercept term), and RE-define the feature vector to be:

$$\vec{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}$$

  - The parameter vector θ is also a column vector

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{bmatrix}_{(p+1)\times 1}$$

$$\hat{y} = f(\mathbf{x})$$

$$\underbrace{= \vec{\mathbf{x}}^T \vec{\theta}}_{\| \; \begin{bmatrix} 1 & x_1 & x_2 & .. & x_p \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{bmatrix}} = \theta^T \mathbf{x}$$

$$\hat{y} = f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_p x_p = \begin{bmatrix} 1 & x_1 & x_2 & .. & x_p \end{bmatrix}$$

A Concise Notation: via Vector/Matrix Product

- Represent each data sample **x** as a column vector, plus a pseudo feature
  - We add a pseudo "feature" $x_0=1$ (this is the intercept term), and RE-define the feature vector to be:

$$x^T=[\underline{(x_0=1)}, x_1, x_2, \dots ,x_p]$$

  - The parameter vector θ is also a column vector

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{bmatrix} \Rightarrow \hat{y} = f(\mathbf{x}) = \mathbf{x}^T\theta = \theta^T\mathbf{x}$$

$$= \theta_0 + \theta_1 x_1 + \theta_2 x + \cdots \theta_p x_p$$

$$\hat{y} = f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Review:

- Dot (or Inner) Product of two vectors <x, a>

is the sum of products of elements in similar positions for the two vectors

$$<x, \Theta> = <\Theta, x>$$

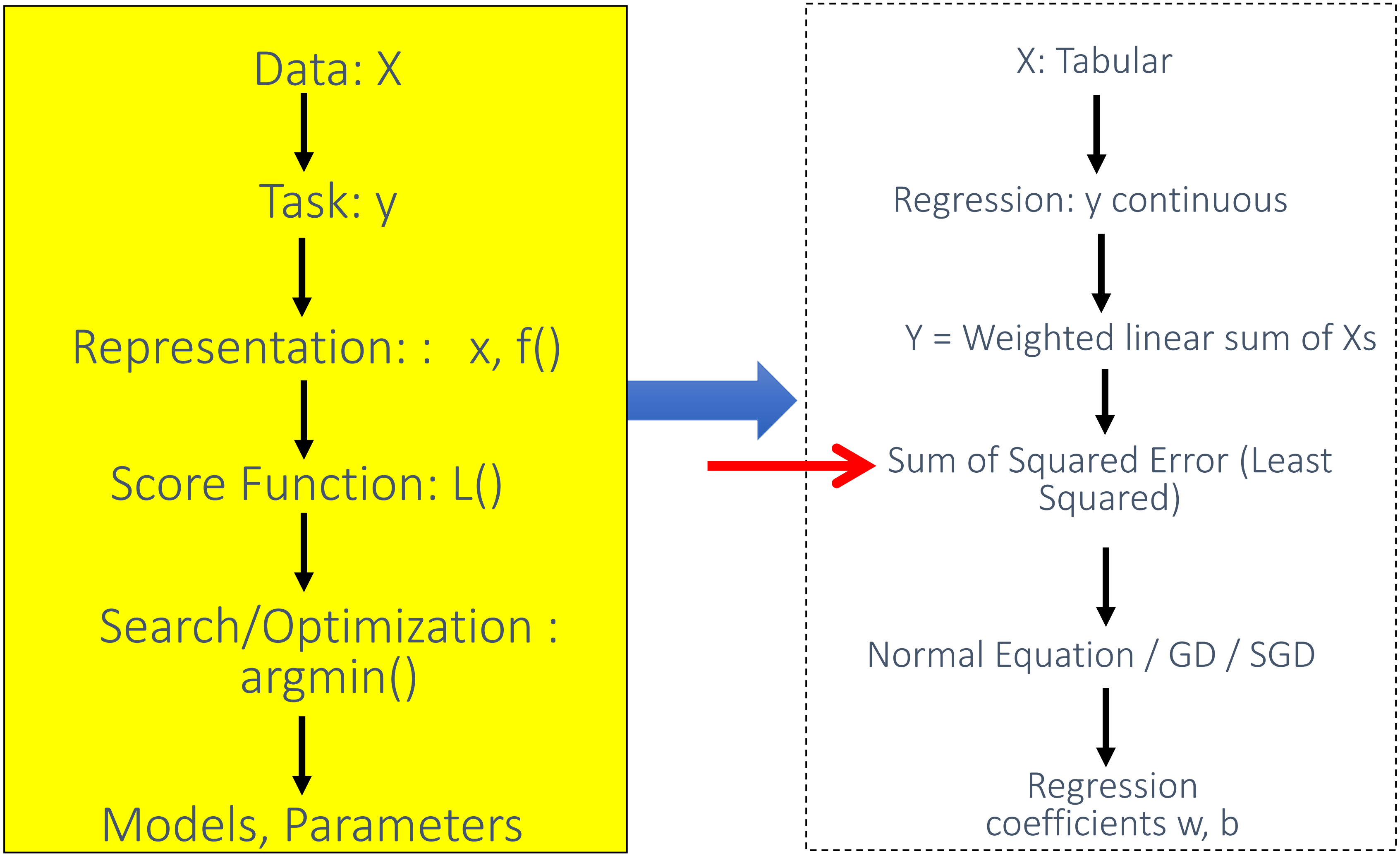
$$x^T\theta = \theta^T x$$

Linear Regression: Y as Weighted linear sum of Xs

$$\hat{y} = f(\mathbf{x})$$

$$= \mathbf{x}^T\theta = \theta^T\mathbf{x}$$

Today: Multivariate Linear Regression in a Nutshell



Data: X

Task: y

Representation: : x, f()

Score Function: L()

Search/Optimization : argmin()

Models, Parameters

X: Tabular

Regression: y continuous

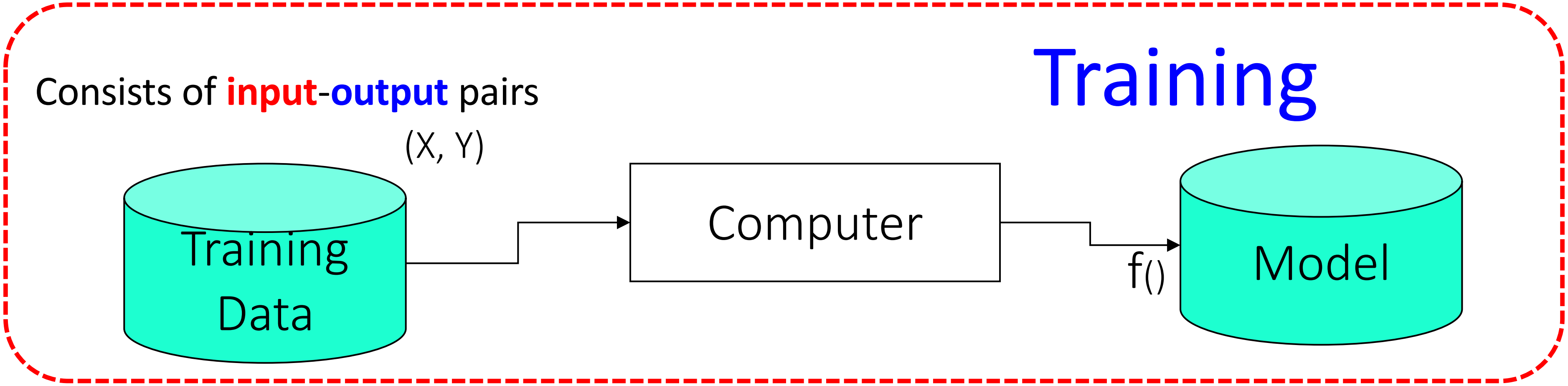
Y = Weighted linear sum of Xs

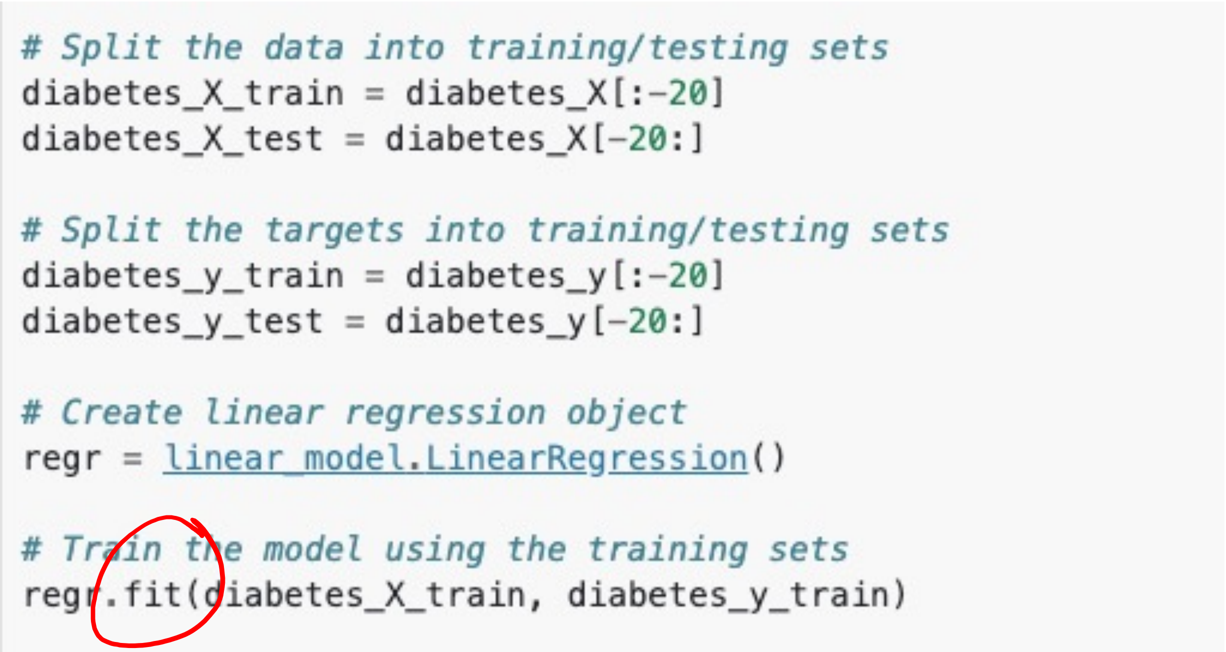
Sum of Squared Error (Least Squared)

Normal Equation / GD / SGD

Regression coefficients w, b

Training Modes of Machine Learning





```
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
```

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Basic Concepts

- Training (i.e. learning parameters w,b)
  - Training set includes
    - available examples $x_1,\ldots,x_n$
    - available corresponding labels $y_1,\ldots,y_n$

$$X_{train} \qquad \xrightarrow{\quad} p$$

$$\begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vdots \\ \vec{x}_n \end{bmatrix} \qquad \vec{Y}_{train} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

n x 1

- Find (w,b) by minimizing loss / Cost function $\underline{L()}$
  - (i.e. difference between y and f(x) on available examples in training set)

$$(W, b) = \underset{W,\, b}{\operatorname{argmin}} \underbrace{\sum_{i=1}^{\overset{n}{L}} \ell(f(\underline{x_i}), \underline{y_i})}_{??}$$

Loss/Cost function for Regression L()

SSE: Sum of squared error

- Our goal is to search for the optimal θ that minimize the following lost /cost function:

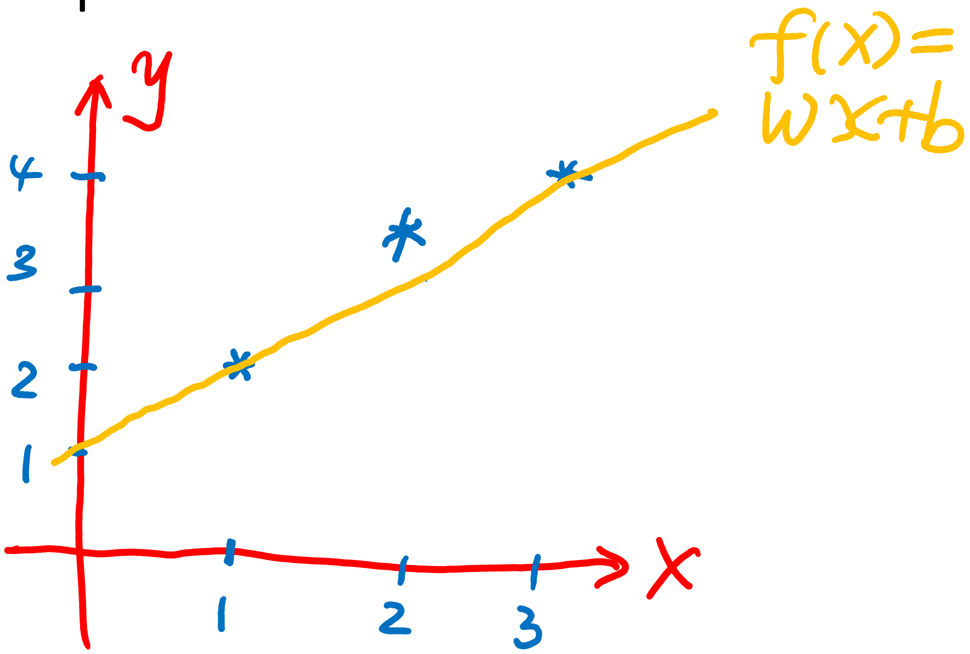
trainig : $$\underline{J(\theta)} = \frac{1}{2}\underbrace{\sum_{i=1}^{n}}(f(\mathbf{x}_i) - y_i)^2 \Rightarrow \theta^* = \underset{\theta}{\operatorname{argmin}} J(\theta)$$

testig : $$\hat{y} = f(\mathbf{x})$$

$$= \mathbf{x}^T\theta^* = \theta^{*T}\mathbf{x}$$

One concrete example

	x	y
S_1	1	2
S_2	2	3.3
S_3	3	4



$$\Downarrow$$

$$f(x)=wx+b$$

$$\Downarrow$$

$$(w^*, b^*) = \underset{w,b}{\operatorname{argmin}} \; J(w,b) = \sum_{i=1}^{3} (w x_i + b - y_i)^2$$

One concrete example

$n=3$

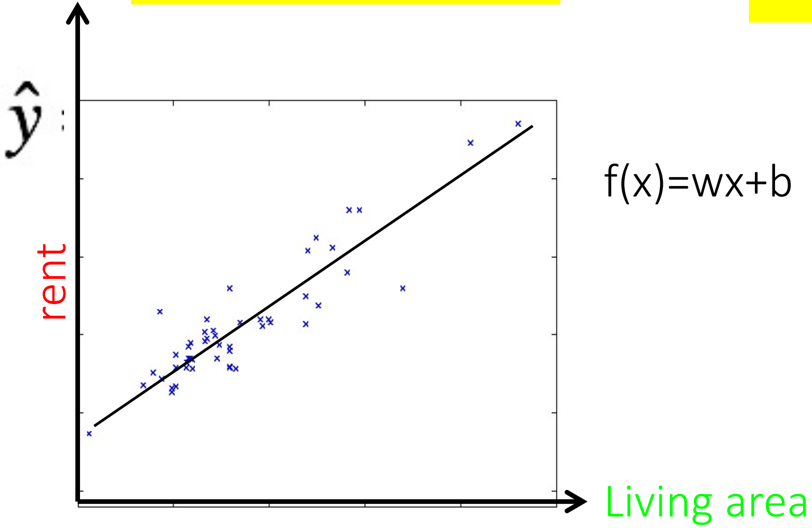
x_i	y_i	$f(x_i)=\hat{y}_i = wx+b$	$(wx_i+b-y_i)^2$
1	2	$w+b$	$(w+b-2)^2$
2	3	$2w+b$	$(2w+b-3)^2$
3	4	$3w+b$	$(3w+b-4)^2$

$$J(\overline{w,b}) = \frac{1}{2}\left\{(w+b-2)^2+(2w+b-3)^2+(3w+b-4)^2\right\}$$

$$\Rightarrow ?? \quad \underset{w,b}{\operatorname{argmin}}\ J(w,b)$$

Linear Regression: Y as Weighted linear sum of Xs

e.g., Living area as X

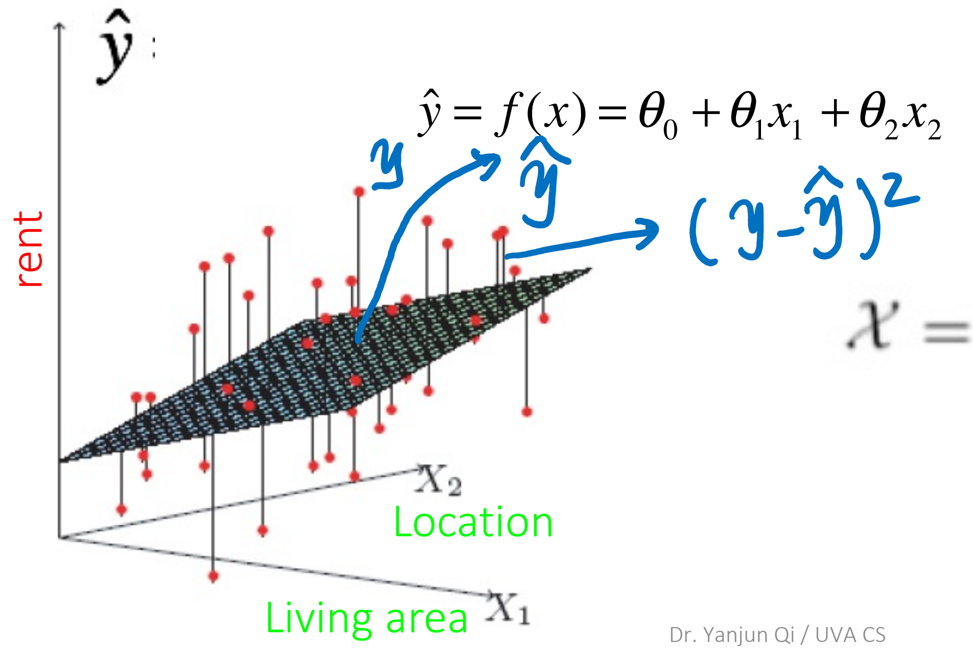


$f(x)=wx+b$

1D case ($\mathcal{X} = \mathbb{R}$): a line

---

e.g., (Living area, Location) as X



$\hat{y} = f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

$\mathcal{X} = \mathbb{R}^2$: a plane

Now the loss function: via A Concise Notation

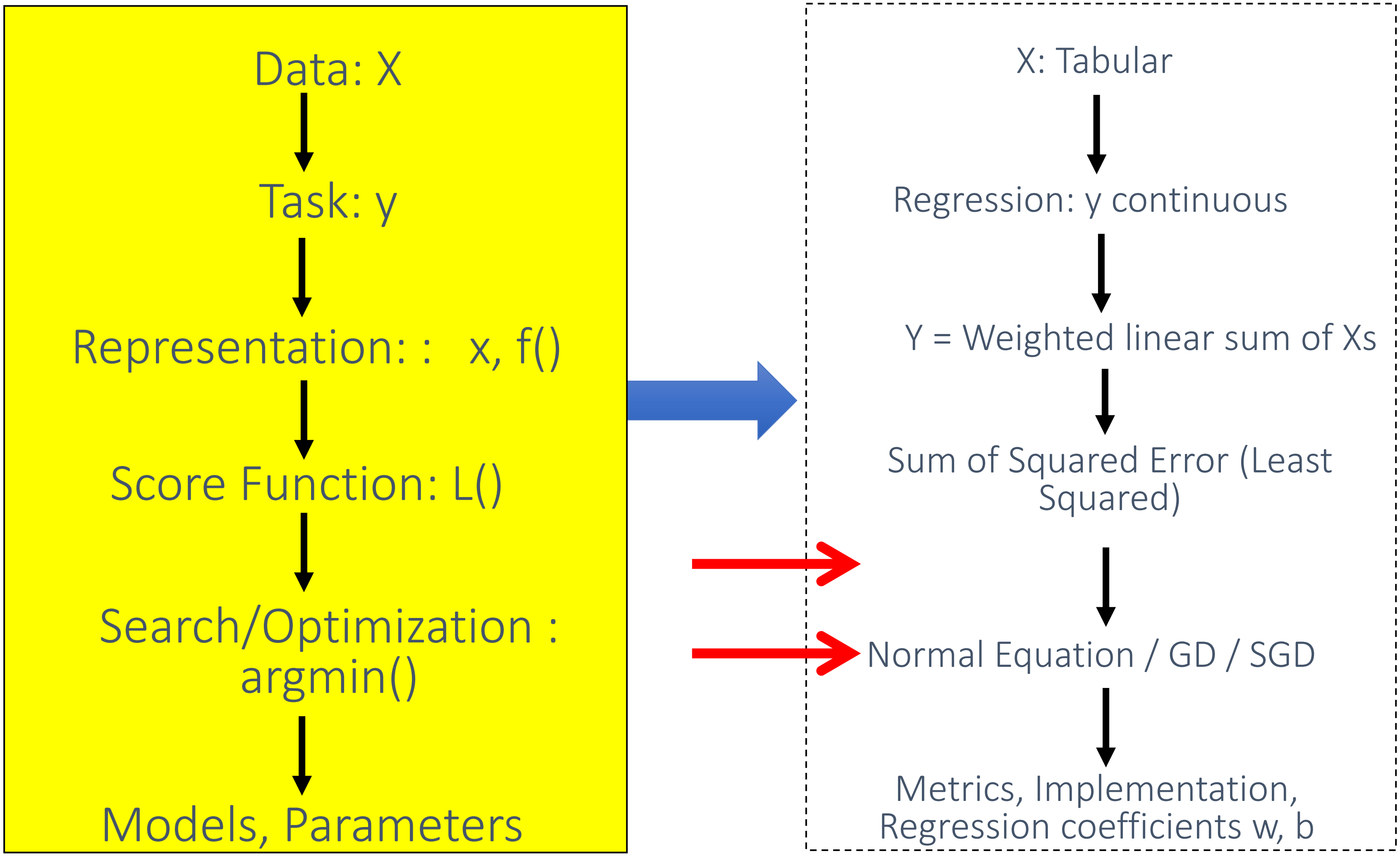
- Using matrix form, we get the following general representation of the linear regression function:

$$\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1^T & -- \\ -- & \mathbf{x}_2^T & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n^T & -- \end{bmatrix}$$

$$\begin{aligned} J(\theta) &= \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i{}^T\theta - y_i)^2 \\ &= \frac{1}{2}\left(X\theta - \vec{y}\right)^T\left(X\theta - \vec{y}\right) \\ &= \underline{\frac{1}{2}\left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T \vec{y}\right)} \end{aligned}$$

Thank You

Today: Multivariate Linear Regression in a Nutshell



Data: X

Task: y

Representation: : x, f()

Score Function: L()

Search/Optimization : argmin()

Models, Parameters

X: Tabular

Regression: y continuous

Y = Weighted linear sum of Xs

Sum of Squared Error (Least Squared)

Normal Equation / GD / SGD

Metrics, Implementation, Regression coefficients w, b

Default Vector Form is the Column Form

$$\vec{X}_1 = \begin{bmatrix} 1 \\ x_{1,1} \\ x_{1,2} \\ x_{1,3} \\ \vdots \\ x_{1,p} \end{bmatrix}_{(p+1)\times 1}$$

$$\vec{x}_1^{\,T} = \left[1, x_{1,1}, x_{1,2}, \cdots, x_{1,p} \right]$$

$$\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1^T & -- \\ -- & \mathbf{x}_2^T & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n^T & -- \end{bmatrix}$$

Training Set in Matrix Form

- the whole Training set (with n samples) as matrix form :

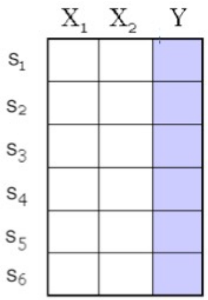
$$\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1^T & -- \\ -- & \mathbf{x}_2^T & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n^T & -- \end{bmatrix} = \begin{bmatrix} x_{1,0} & x_{1,1} & \ldots & x_{1,p} \\ x_{2,0} & x_{2,1} & \ldots & x_{2,p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,0} & x_{n,1} & \ldots & x_{n,p} \end{bmatrix}$$

p

n

nxP

$$\vec{y}_{train} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

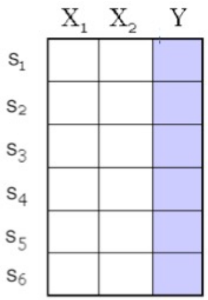


Training Set in Matrix Form

- the whole Training set (with n samples) as matrix form :

$$\underset{n\times(p+1)}{X}\,\underset{(p+1)\times 1}{\theta} = \begin{bmatrix} x_1^T\theta \\ \vdots \\ x_n^T\theta \end{bmatrix}$$

$$\mathbf{X}\theta = \begin{bmatrix} -- & \mathbf{x}_1^T & -- \\ -- & \mathbf{x}_2^T & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n^T & -- \end{bmatrix}\theta = \begin{bmatrix} x_{1,0} & x_{1,1} & \dots & x_{1,p} \\ x_{2,0} & x_{2,1} & \dots & x_{2,p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,0} & x_{n,1} & \dots & x_{n,p} \end{bmatrix} \qquad \vec{y}_{train} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$



Cont: Loss function: via A Concise Notation

- Using matrix form, we get the following general representation of the linear regression function:

$$\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1^T & -- \\ -- & \mathbf{x}_2^T & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n^T & -- \end{bmatrix}$$

$$\begin{aligned} J(\theta) &= \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i{}^T\theta - y_i)^2 \\ &= \frac{1}{2}\left(X\theta - \vec{y}\right)^T\left(X\theta - \vec{y}\right) \\ &= \frac{1}{2}\left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T \vec{y}\right) \end{aligned}$$

Cont: Loss function: via A Concise Notation

- Using matrix form, we get the following general representation of the linear regression function:

$$\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1^T & -- \\ -- & \mathbf{x}_2^T & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n^T & -- \end{bmatrix} \theta$$

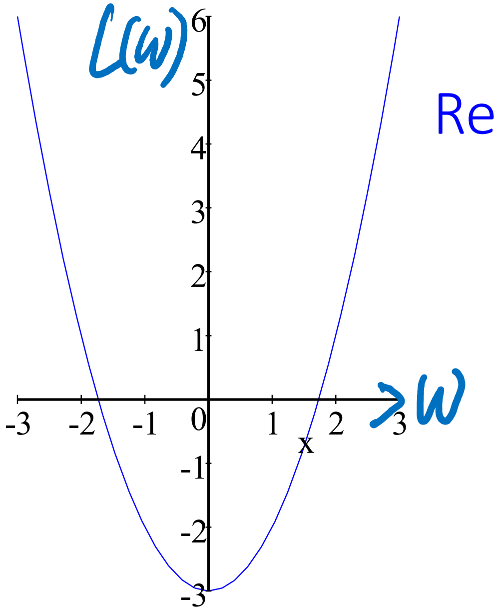
$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^{\,T}\theta - y_i)^2$$

$$= \frac{1}{2}\underline{(X\theta - \vec{y})^T (X\theta - \vec{y})}$$

$$= \frac{1}{2}\left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T \vec{y}\right)$$

$$a = (\underbrace{\underset{n\times p}{X}\,\underset{p\times 1}{\theta}}_{n\times 1} - \underbrace{\underset{n\times 1}{y}}_{}) = \begin{bmatrix} x_1^T\theta - y_1 \\ x_2^T\theta - y_2 \\ \vdots \\ x_n^T\theta - y_n \end{bmatrix}$$

$$\vec{a}^T\vec{a} = \sum_{i=1}^{n} a_i^2$$



Review (IV): Derivative of a Quadratic Function

$$\underline{L(w)} = w^2 \underset{\times}{\underline{-\,3}}$$

$$\downarrow$$

$$L'(w) = 2w = 0$$

$$argmin_w L(w) \quad = (L'(w)=0)$$

$$\|$$

$$w^*$$

This quadrative (convex) function is minimized @ the unique point whose derivative (slope) is zero.

➔ When we find zeros of the derivative of this function, we also find the minima (or maxima) of that function.

Critical Points

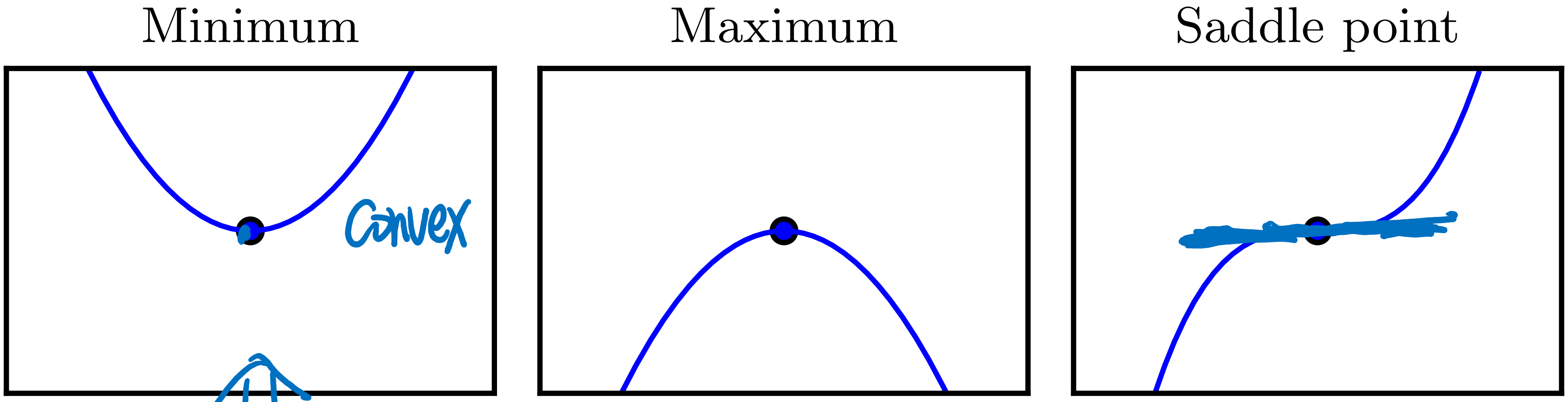


Figure 4.2

⇑

$$J(\theta) = \sum_{i=1}^{n} (\vec{x}_i^T \vec{\theta} - y_i)^2$$

(Goodfellow 2017)

Find best θ via Solving $\nabla_\theta J(\theta) = 0$

- Write the cost function in matrix form:

$$
\begin{aligned}
J(\theta) &= \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i{}^T\theta - y_i)^2 \\
&= \frac{1}{2}\left(X\theta - \vec{y}\right)^T\left(X\theta - \vec{y}\right) \\
&= \frac{1}{2}\left(\theta^T X^T X\theta - \theta^T X^T\vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right)
\end{aligned}
$$

To minimize J(θ), take derivative (gradient) and set to zero:

$$\Rightarrow \quad X^T X\theta = X^T\vec{y}$$

The normal equations

$$\Downarrow$$

Closed form solution

$$\theta^* = \left(X^T X\right)^{-1} X^T\vec{y}$$

See handout 4.1 + 4.3 ⇒ matrix calculus, partial deri ⇒ Gradient

$$\nabla_\theta(\theta^T X^T X \theta) = 2X^T X \theta \qquad (P24)$$

$$\nabla_\theta(-2\theta^T X^T y) = -2X^T Y \qquad (P24)$$

$$\nabla_\theta(y^T y) = 0$$

This Slide is Optional / Extra

$$\Rightarrow \nabla_\theta J(\theta) = \boxed{X^T X \theta - X^T Y}$$

$$\Rightarrow \text{Hessian } H(J(\theta)) = \frac{\partial \nabla_\theta J(\theta)}{\partial \theta} = \frac{\partial (X^T X \theta)}{\partial \theta} =$$

Advanced / Optional / More in Extra Slides

Gram matrix $X^T X$

Method I: normal equations to minimize the loss

- Write the cost function in matrix form:

$$J(\theta)=\frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta-y_i)^2$$

$$=\frac{1}{2}(X\theta-\vec{y})^T(X\theta-\vec{y})$$

$$\theta^T X^T \bar{y}$$

//

$$=\frac{1}{2}\left(\underbrace{\theta^TX^TX\theta}_{2X^TX\theta}-\theta^TX^T\vec{y}-\underbrace{\vec{y}^T}_{1\times n}\underbrace{X\theta}_{n\times 1}+\vec{y}^T\vec{y}\right)$$

$$\nabla_\theta J(\theta)=\frac{\partial J(\theta)}{\partial\theta}=\frac{1}{2}\left(2X^TX\theta-X^Ty-X^T\bar{y}\right)$$

To minimize J(θ), take derivative and set to zero:

$$\nabla_\theta J(\theta)=0 \quad\Rightarrow\quad \boxed{X^TX\theta=X^T\vec{y}}$$

The normal equations

$$\Downarrow$$

Closed form solution

$$\theta^*=\left(X^TX\right)^{-1}X^T\vec{y}$$

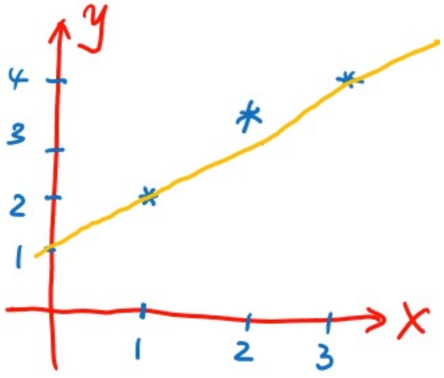
Detailed Derivation is Optional / Extra

One concrete example

$$\Rightarrow \quad J(w,b) = \frac{1}{2}\left((w+b-2)^2 + (2w+b-3)^2\right)$$

$$\Rightarrow \begin{cases} \dfrac{\partial J(w,b)}{\partial w} = (w+b-2) + (2w+b-3)\cdot 2 = 0 \\ \dfrac{\partial J(w,b)}{\partial b} = w+b-2 + (2w+b-3) = 0 \end{cases}$$

$$\begin{cases} 5w + \underbrace{3b} - 8 = 0 \\ 3w + \underbrace{2b} - 5 = 0 \end{cases} \Rightarrow \begin{cases} 10w + 6b - 16 = 0 \\ 9w + 6b - 15 = 0 \end{cases}$$

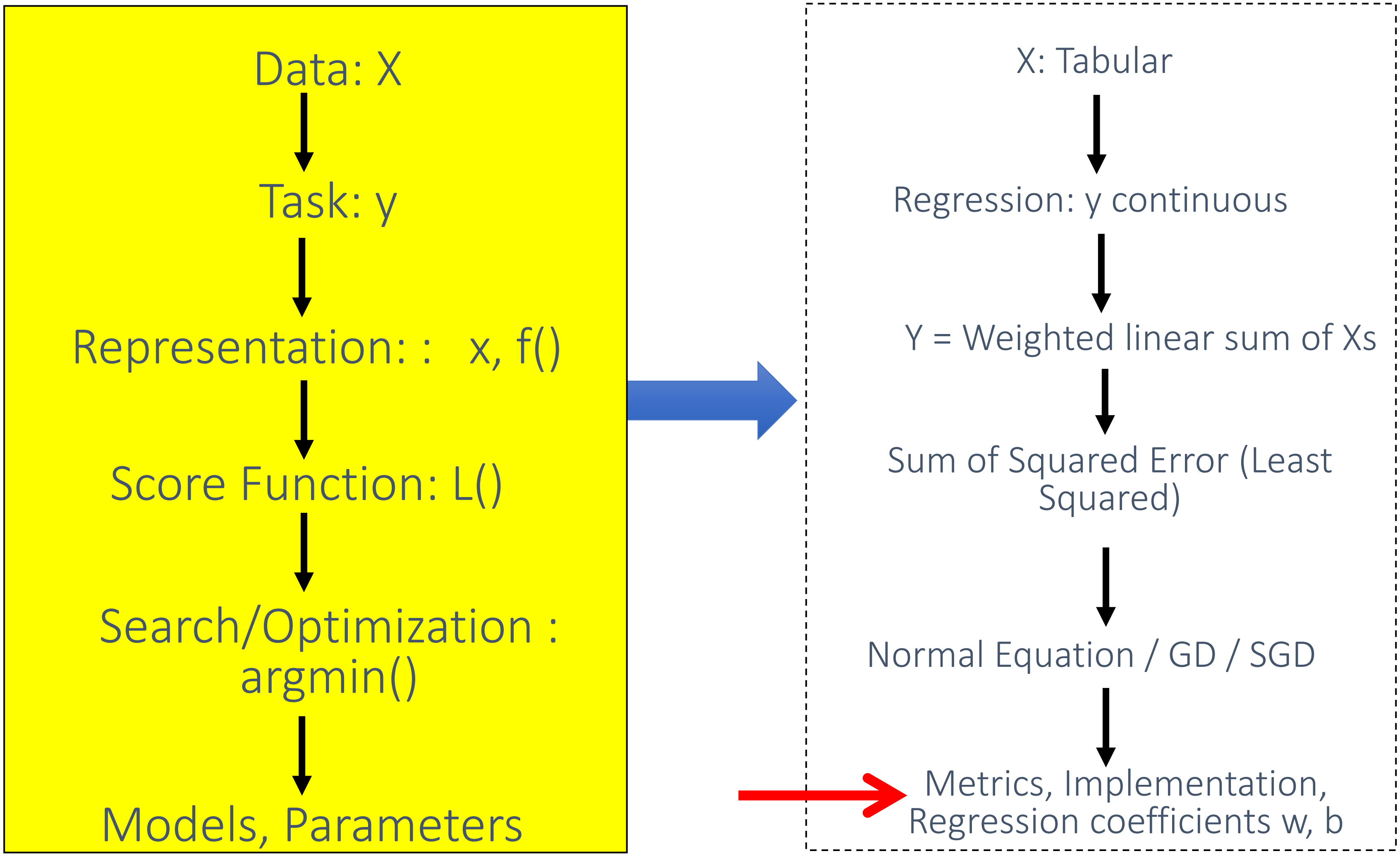


$$\Rightarrow w = 1, \quad b = \frac{1}{2}(5-3w) = 1$$

we solve the matrix equation via Gaussian Elimination

Thank You

Today: Multivariate Linear Regression in a Nutshell



Data: X

Task: y

Representation: : x, f()

Score Function: L()

Search/Optimization : argmin()

Models, Parameters

X: Tabular

Regression: y continuous

Y = Weighted linear sum of Xs

Sum of Squared Error (Least Squared)

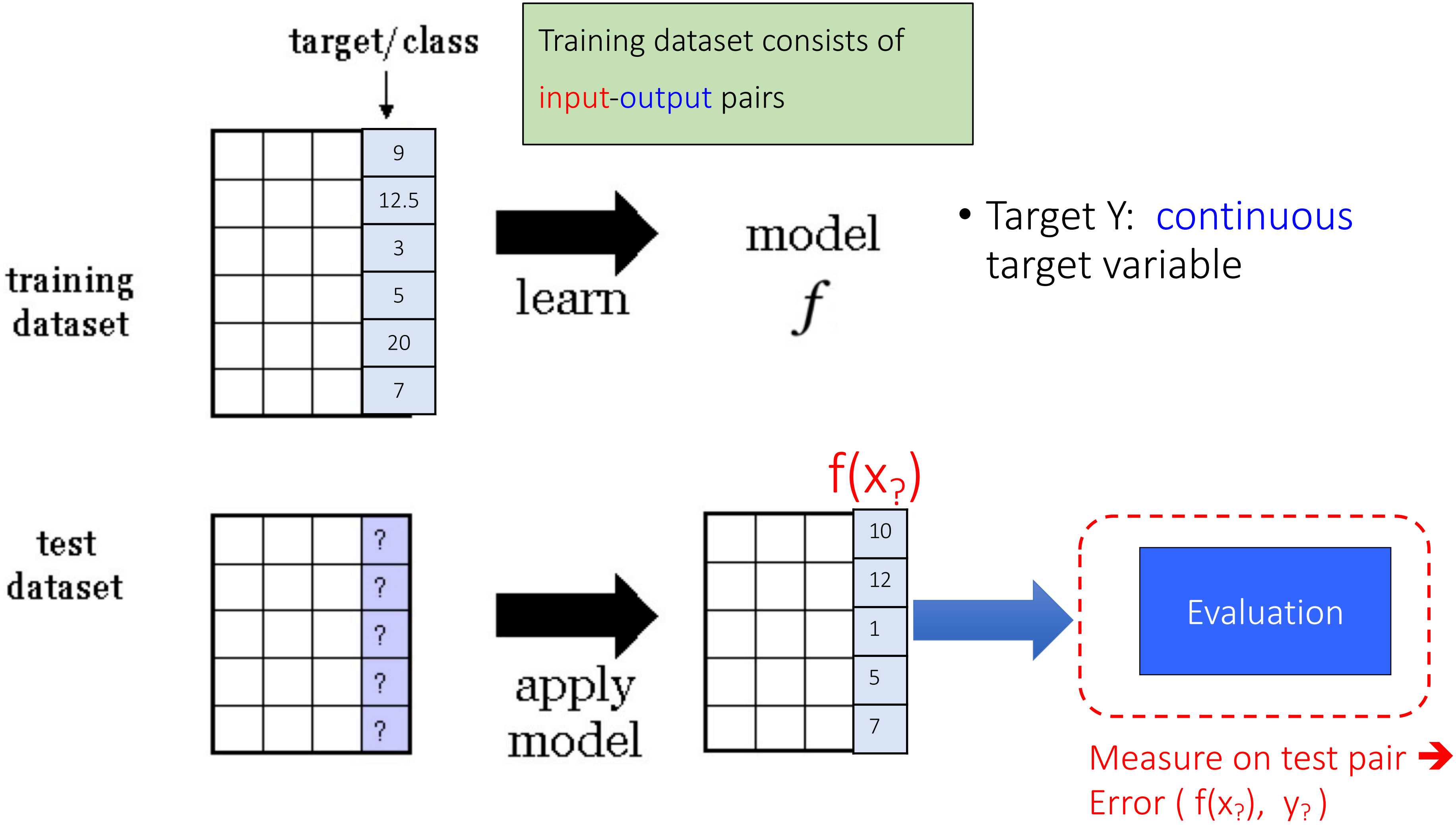
Normal Equation / GD / SGD

Metrics, Implementation, Regression coefficients w, b

We aim to make the learned model

- 1. Generalize Well
- 2. Computational Scalable and Efficient
- 3. Robust / Trustworthy / Interpretable
  - Especially for some domains, it is about trust!

How to know the regression program works well: Metrics on Regression Predictions

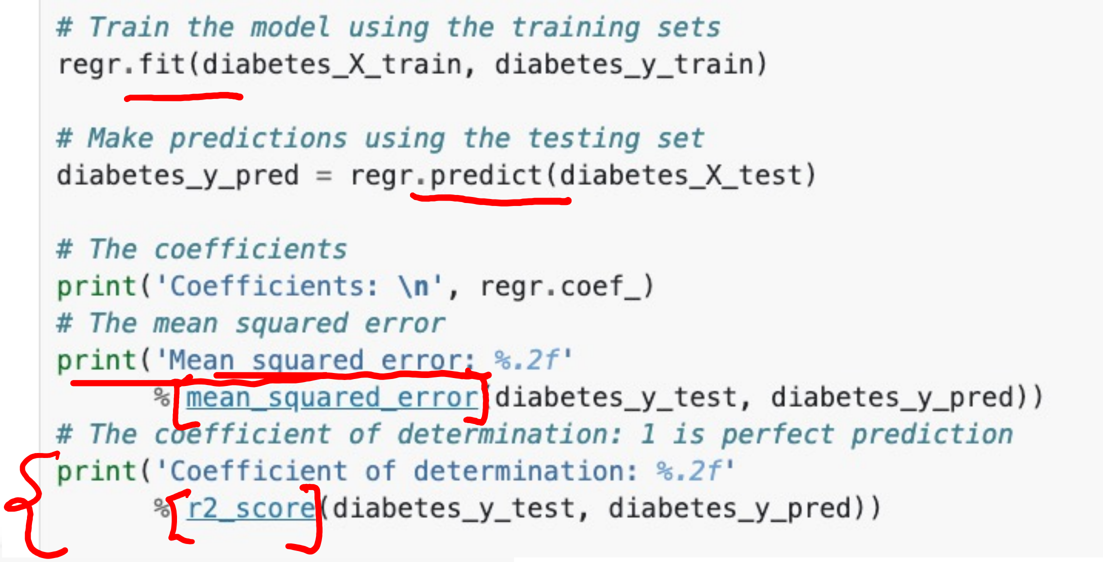


Training dataset consists of

input-output pairs

- Target Y: continuous target variable

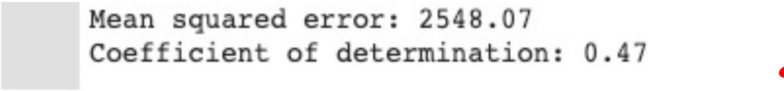
Measure on test pair → Error ($f(x_?)$, $y_?$)



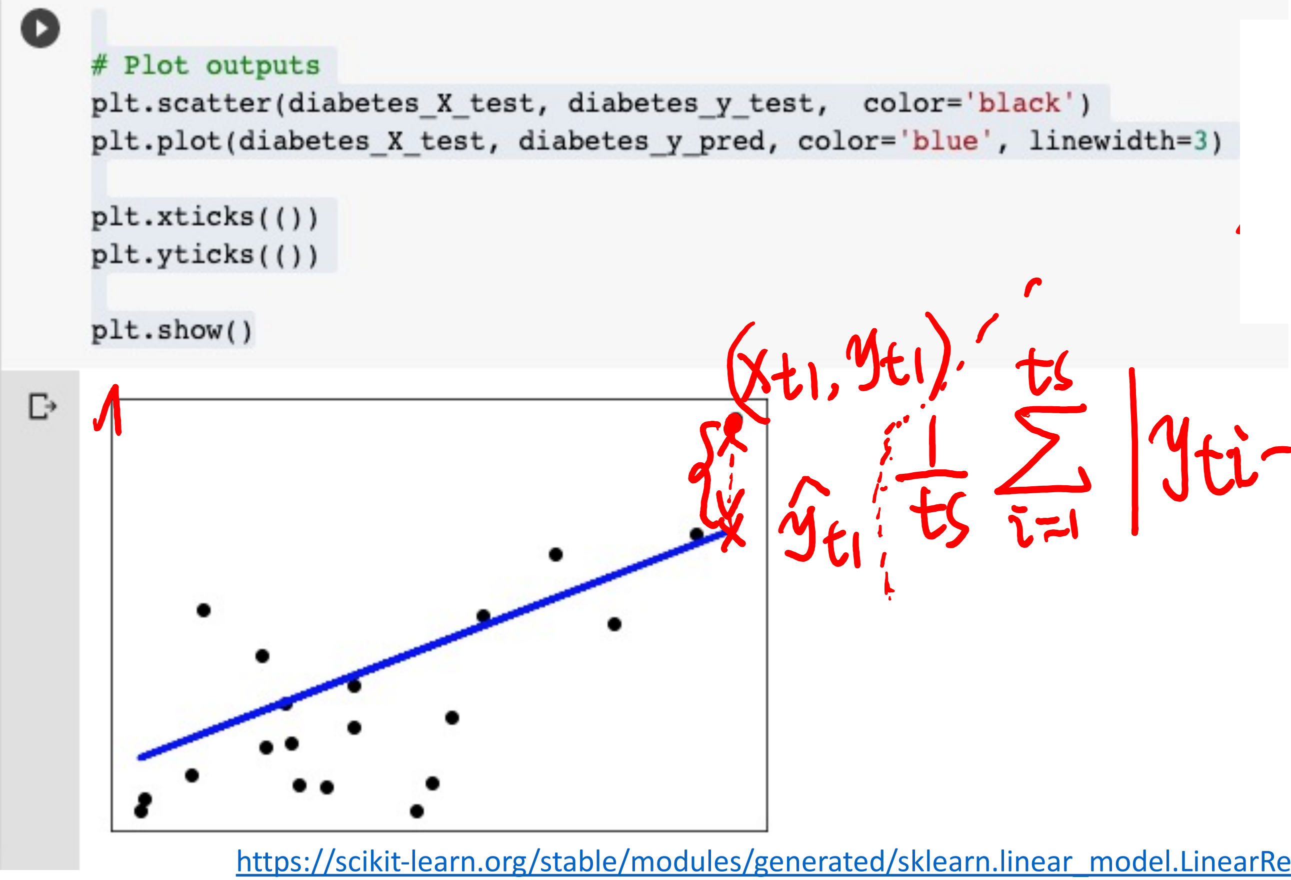
```
# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(diabetes_y_test, diabetes_y_pred))
```



```
Mean squared error: 2548.07
Coefficient of determination: 0.47
```



```
# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```

(x_{t1}, y_{t1})

$\hat{y}_{t1}$

$$\frac{1}{ts}\sum_{i=1}^{ts}\left|y_{ti}-\hat{y}_{ti}\right|$$

mean_absolute
_error

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

- Test MSE Error to report:

$$J_{test\_MSE} = \frac{1}{m}\sum_{i=\mathrm{n+1}}^{n+m} \left(\mathbf{x}_i{}^T\theta^{\mathrm{t}} - y_i\right)^2$$

training dataset

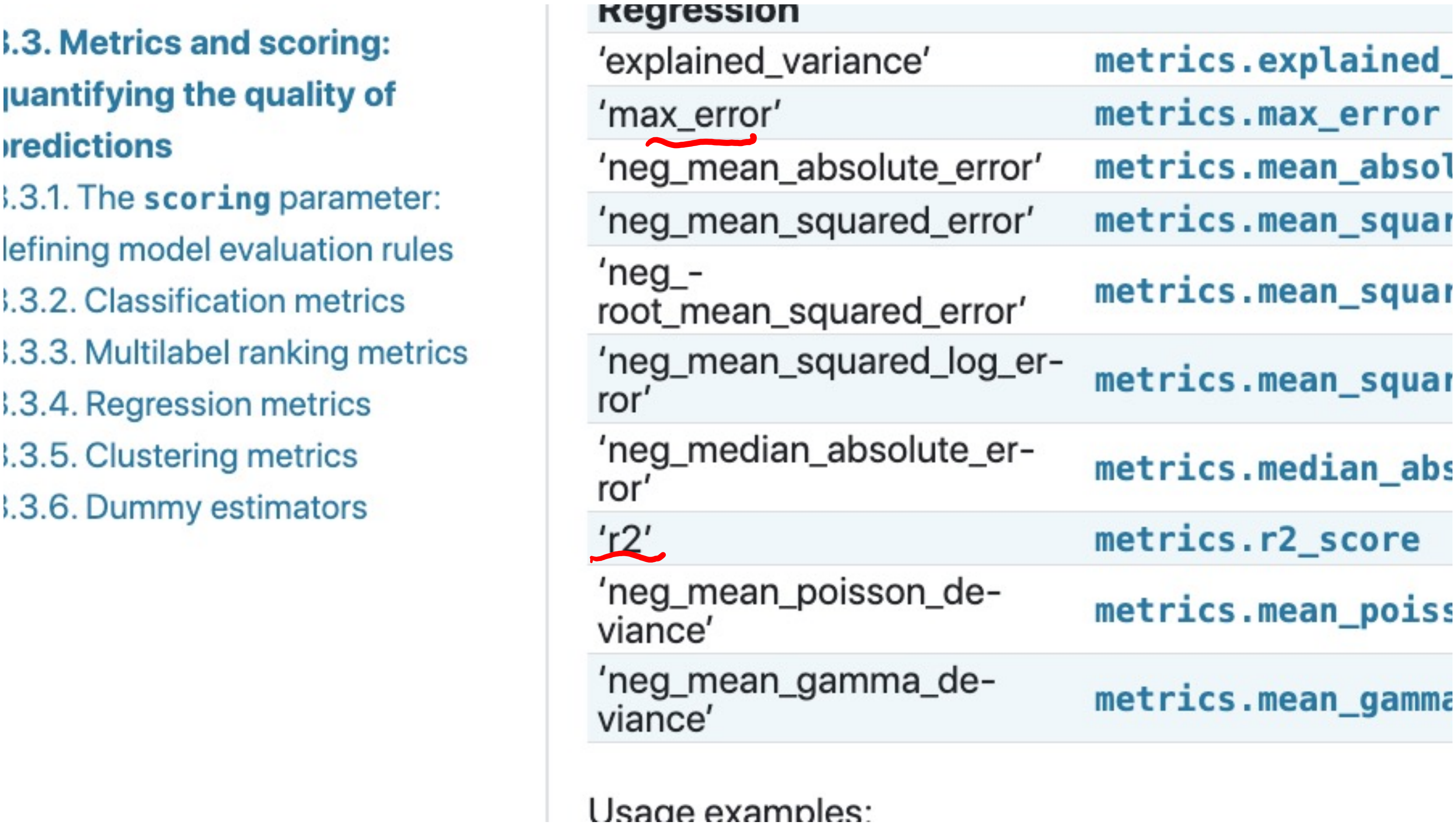
$$\mathbf{X}_{train} = \begin{bmatrix} -\,- & \mathbf{x}_1^T & -\,- \\ -\,- & \mathbf{x}_2^T & -\,- \\ \vdots & \vdots & \vdots \\ -\,- & \mathbf{x}_n^T & -\,- \end{bmatrix} \qquad \vec{y}_{train} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

test dataset

$$\mathbf{X}_{test} = \begin{bmatrix} -\,- & \mathbf{x}_{n+1}^T & -\,- \\ -\,- & \mathbf{x}_{n+2}^T & -\,- \\ \vdots & \vdots & \vdots \\ -\,- & \mathbf{x}_{n+m}^T & -\,- \end{bmatrix} \qquad \vec{y}_{test} = \begin{bmatrix} y_{n+1} \\ y_{n+2} \\ \vdots \\ y_{n+m} \end{bmatrix}$$

Many other possible Metrics for Regression

https://scikit-learn.org/stable/modules/model_evaluation.html



.3. Metrics and scoring: uantifying the quality of redictions

- .3.1. The `scoring` parameter: efining model evaluation rules
- .3.2. Classification metrics
- .3.3. Multilabel ranking metrics
- .3.4. Regression metrics
- .3.5. Clustering metrics
- .3.6. Dummy estimators

Regression	
'explained_variance'	`metrics.explained_`
'max_error'	`metrics.max_error`
'neg_mean_absolute_error'	`metrics.mean_absol`
'neg_mean_squared_error'	`metrics.mean_squar`
'neg_root_mean_squared_error'	`metrics.mean_squar`
'neg_mean_squared_log_error'	`metrics.mean_squar`
'neg_median_absolute_error'	`metrics.median_abs`
'r2'	`metrics.r2_score`
'neg_mean_poisson_deviance'	`metrics.mean_poiss`
'neg_mean_gamma_deviance'	`metrics.mean_gamma`

Usage examples:

Question 3.1. Linear Regression+ Train-Test Split

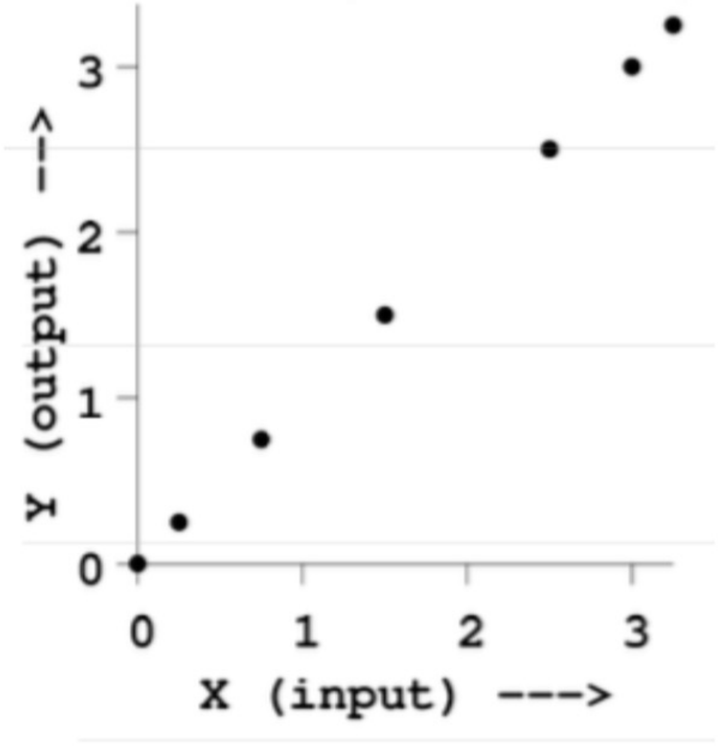


Figure 1: A reference dataset for regression with one real-valued input (x as horizontal axis) and one real-valued output (y as vertical axis).

What is the mean squared training error when running linear regression to fit the data ? (i.e., the model is $y = \beta_0 + \beta_1 x$). Assuming the rightmost three points are in the test set, and the others are in the training set. (you can eyeball the answers.)

We aim to make the learned model

- 1. Generalize Well
- → 2. Computational Scalable and Efficient
- 3. Robust / Trustworthy / Interpretable
  - Especially for some domains, this is about trust!

Why we Prefer Concise Vector/Matrix Form?

Training: Closed form solution

fit

$$\theta^* = (X_{train}^T X_{train})^{-1}(X_{train}^T \vec{y}_{train})$$

Testing: on multiple Test Inputs

predict

$$\widehat{\vec{y}}_{test} = X_{test}\theta^*$$

Extra: Naïve Matrix Multiply

$A_{n\times n}, B_{n\times n}, C_{n\times n}$

```
{implements C = C + A*B}
for i = 1 to n
    for j = 1 to n
        for k = 1 to n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

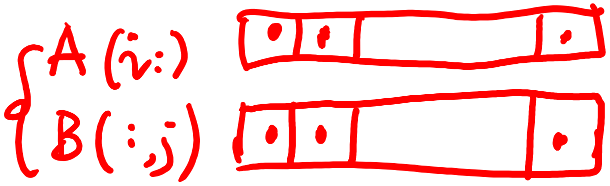
nxn

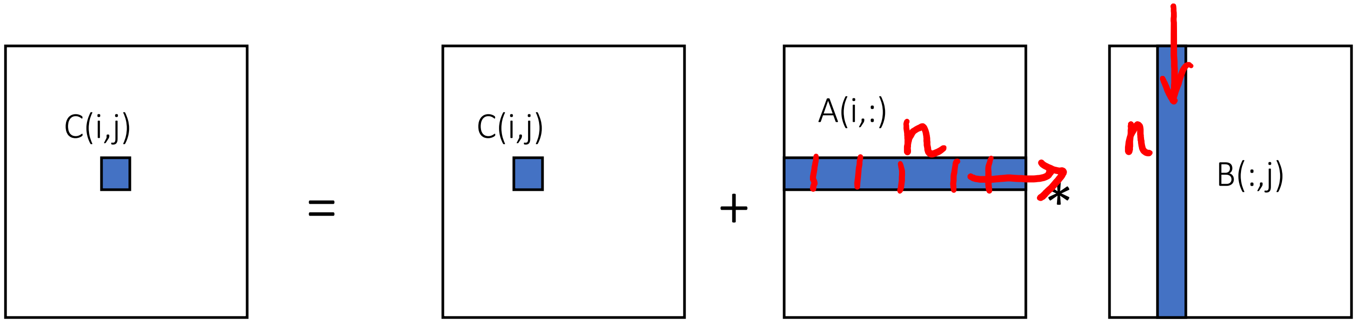
O(n)

$O(n^3)$ time

Extra:

Algorithm has $2*n^3 = O(n^3)$ Flops and operates on $\underbrace{3*n^2}_{\text{space cost}}$ words of memory





http://www.cs.berkeley.edu/~demmel/cs267_Spr12/

Computational Cost (Naïve Way)

$$\underset{n \times p}{X_{train}} \rightarrow \underset{p \times n}{X^T}$$

$$\vec{\theta}^* = (\underbrace{\underset{p \times n}{X^T} \underset{n \times p}{X}}_{p \times p})^{-1} X^T \vec{y}$$

Extra:

X^TX : $O(p^2 n)$

$(X^TX)^{-1}$: $O(p^3)$

$O(\underline{np^2} + p^3)$

when $n >> p$, matrix multi slower than inversion

Many architecture details and Algorithm details to consider

- (1): Data parallelization through CPU SIMD / Multithreading/ GPU parallelization / ….

- (2): Memory hierarchical / locality

- (3): Better algorithms, like Strassen’s Matrix Multiply and many others

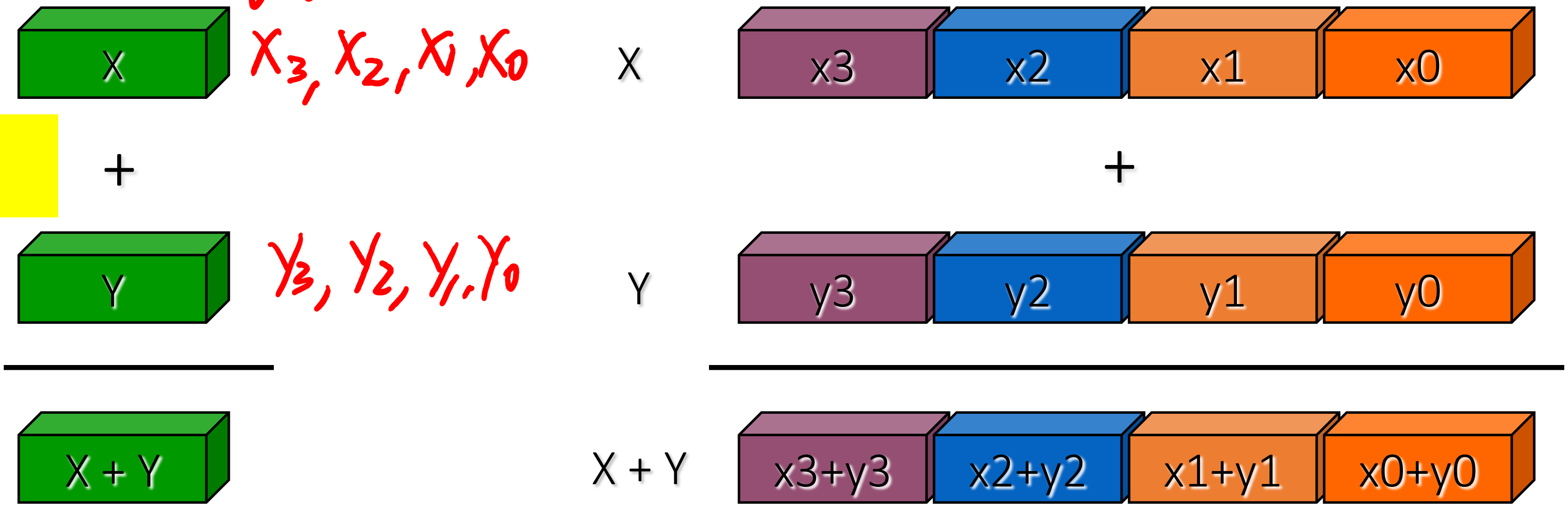
(1): SIMD: Single Instruction, Multiple Data

- Scalar processing
  - traditional mode
  - one operation produces one result

$$= \sum_{i=0}^{3} (x_i + y_i)$$

- SIMD processing
  - with SSE / SSE2
  - SSE = streaming SIMD extensions
  - one operation produces multiple results

Extra:



X

x_3, x_2, x_1, x_0

\+

Y

y_3, y_2, y_1, y_0

X + Y

X

x3 | x2 | x1 | x0

\+

Y

y3 | y2 | y1 | y0

X + Y

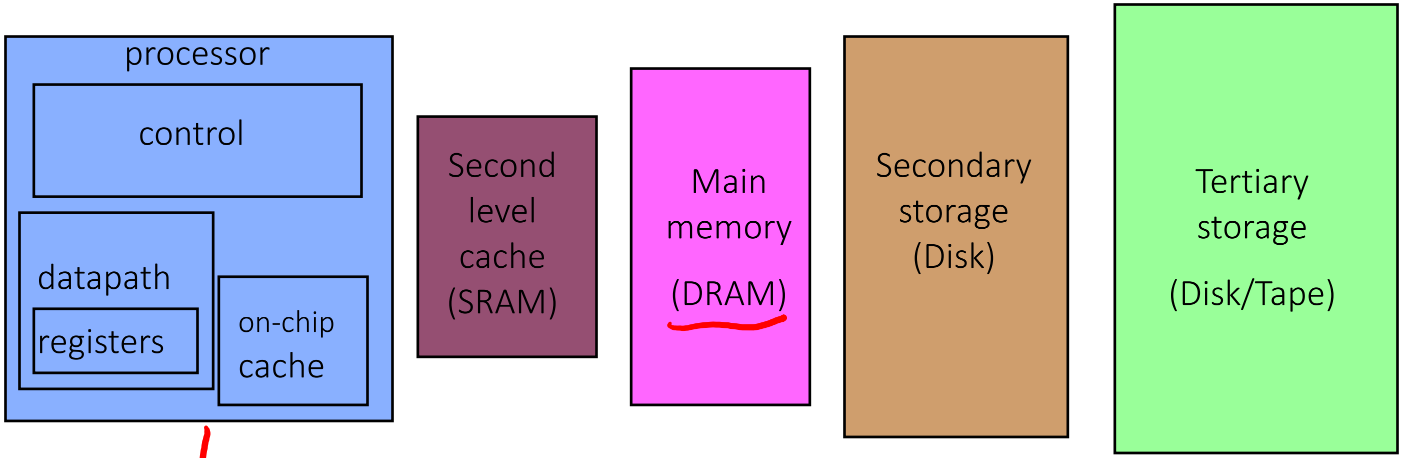
x3+y3 | x2+y2 | x1+y1 | x0+y0

Slide Source: Alex Klimovitski & Dean Macri, Intel Corporation

(2): Memory Hierarchy

- Most programs have a high degree of locality in their accesses
  - **spatial locality:** accessing things nearby previous accesses
  - **temporal locality:** reusing an item that was previously accessed
- Memory hierarchy tries to exploit locality to improve average

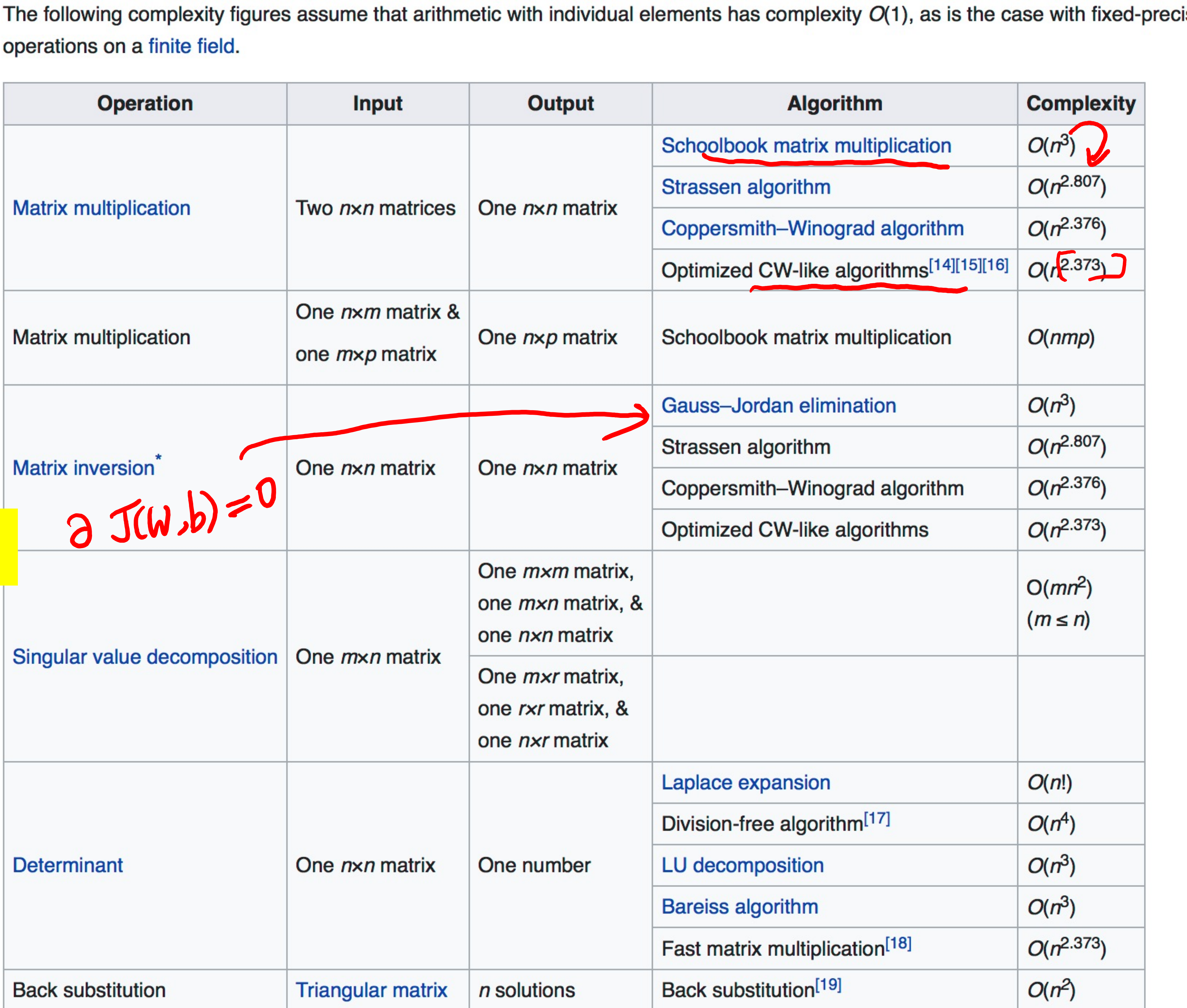
Extra:



Speed	1ns	10ns	100ns	10ms	10sec
Size	KB	MB	GB	TB	PB

http://www.cs.berkeley.edu/~demmel/cs267_Spr12/

(3)



The following complexity figures assume that arithmetic with individual elements has complexity $O(1)$, as is the case with fixed-preci operations on a finite field.

Operation	Input	Output	Algorithm	Complexity
Matrix multiplication	Two $n \times n$ matrices	One $n \times n$ matrix	Schoolbook matrix multiplication	$O(n^3)$
			Strassen algorithm	$O(n^{2.807})$
			Coppersmith–Winograd algorithm	$O(n^{2.376})$
			Optimized CW-like algorithms[14][15][16]	$O(n^{2.373})$
Matrix multiplication	One $n \times m$ matrix & one $m \times p$ matrix	One $n \times p$ matrix	Schoolbook matrix multiplication	$O(nmp)$
Matrix inversion*	One $n \times n$ matrix	One $n \times n$ matrix	Gauss–Jordan elimination	$O(n^3)$
			Strassen algorithm	$O(n^{2.807})$
			Coppersmith–Winograd algorithm	$O(n^{2.376})$
			Optimized CW-like algorithms	$O(n^{2.373})$
Singular value decomposition	One $m \times n$ matrix	One $m \times m$ matrix, one $m \times n$ matrix, & one $n \times n$ matrix		$O(mn^2)$ $(m \leq n)$
		One $m \times r$ matrix, one $r \times r$ matrix, & one $n \times r$ matrix		
Determinant	One $n \times n$ matrix	One number	Laplace expansion	$O(n!)$
			Division-free algorithm[17]	$O(n^4)$
			LU decomposition	$O(n^3)$
			Bareiss algorithm	$O(n^3)$
			Fast matrix multiplication[18]	$O(n^{2.373})$
Back substitution	Triangular matrix	n solutions	Back substitution[19]	$O(n^2)$

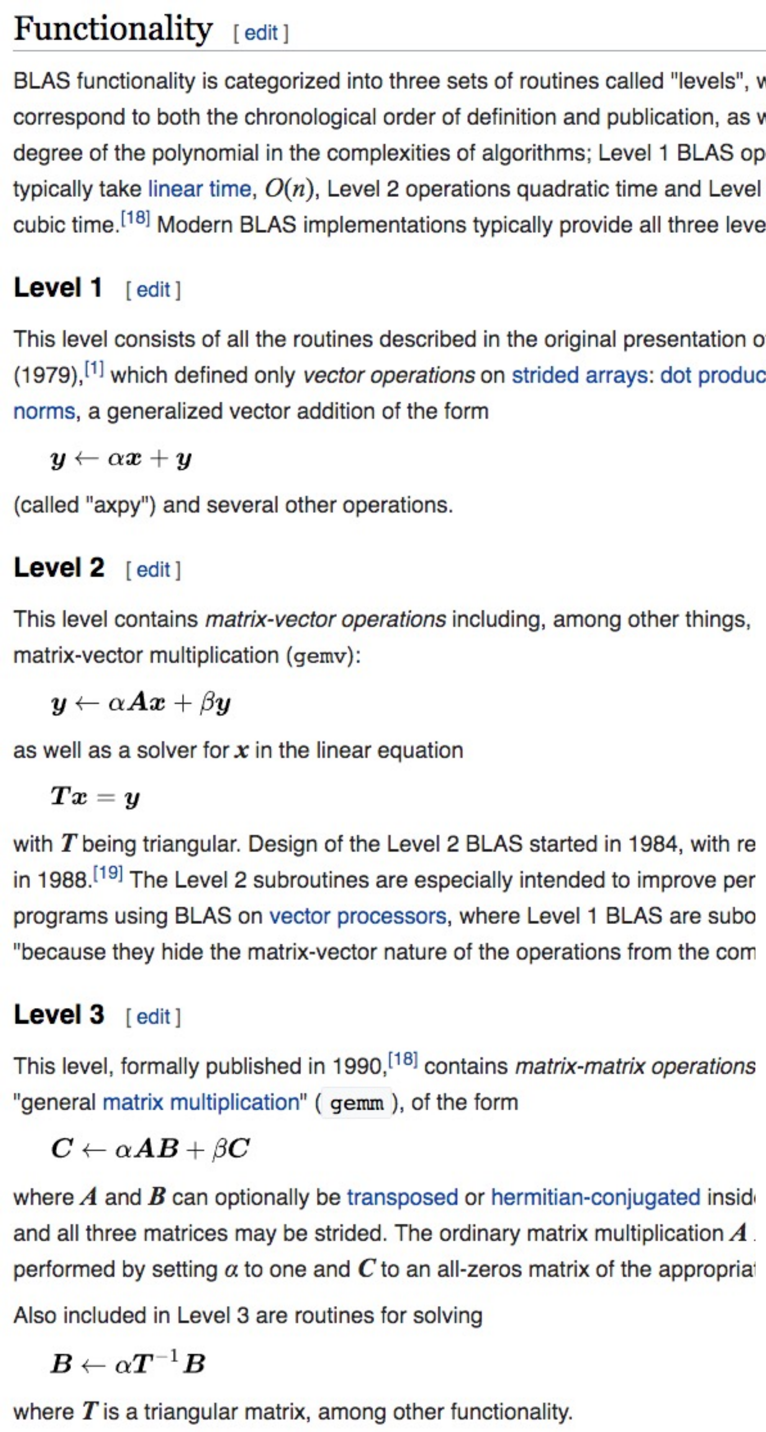
Extra:

$\partial J(w,b)=0$

Basic Linear Algebra Subroutines (BLAS) ➔ numpy: a wrapper library of BLAS

- Industry standard interface (evolving)
  - www.netlib.org/blas, www.netlib.org/blas/blast--forum
- Vendors, others supply optimized implementations
- History
  - BLAS1 (1970s):
    - vector operations: dot product, saxpy (y=a*x+y), etc
    - m=2*n, f=2*n, q = f/m = computational intensity ~1 or less
  - BLAS2 (mid 1980s)
    - matrix-vector operations: matrix vector multiply, etc
    - m=n^2, f=2*n^2, q~2, less overhead
    - somewhat faster than BLAS1
  - BLAS3 (late 1980s)
    - matrix-matrix operations: matrix matrix multiply, etc
    - m <= 3n^2, f=O(n^3), so q=f/m can possibly be as large as n, so BLAS3 is potentially much faster than BLAS2
- Good algorithms use BLAS3 when possible (LAPACK & ScaLAPACK)
  - See www.netlib.org/{lapack,scalapack}

source: Stanford Optim EE course



Functionality [edit]

BLAS functionality is categorized into three sets of routines called "levels", which correspond to both the chronological order of definition and publication, as well as the degree of the polynomial in the complexities of algorithms; Level 1 BLAS operations typically take linear time, $O(n)$, Level 2 operations quadratic time and Level 3 operations cubic time.[18] Modern BLAS implementations typically provide all three levels.

Level 1 [edit]

This level consists of all the routines described in the original presentation of BLAS (1979),[1] which defined only *vector operations* on strided arrays: dot products, vector norms, a generalized vector addition of the form

$$\boldsymbol{y} \leftarrow \alpha \boldsymbol{x} + \boldsymbol{y}$$

(called "axpy") and several other operations.

Level 2 [edit]

This level contains *matrix-vector operations* including, among other things, matrix-vector multiplication (`gemv`):

$$\boldsymbol{y} \leftarrow \alpha \boldsymbol{A} \boldsymbol{x} + \beta \boldsymbol{y}$$

as well as a solver for $\boldsymbol{x}$ in the linear equation

$$\boldsymbol{T} \boldsymbol{x} = \boldsymbol{y}$$

with $\boldsymbol{T}$ being triangular. Design of the Level 2 BLAS started in 1984, with re in 1988.[19] The Level 2 subroutines are especially intended to improve per programs using BLAS on vector processors, where Level 1 BLAS are subo "because they hide the matrix-vector nature of the operations from the com

Level 3 [edit]

This level, formally published in 1990,[18] contains *matrix-matrix operations* "general matrix multiplication" (`gemm`), of the form

$$\boldsymbol{C} \leftarrow \alpha \boldsymbol{A} \boldsymbol{B} + \beta \boldsymbol{C}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ can optionally be transposed or hermitian-conjugated insid and all three matrices may be strided. The ordinary matrix multiplication $\boldsymbol{A}$ performed by setting α to one and $\boldsymbol{C}$ to an all-zeros matrix of the appropria

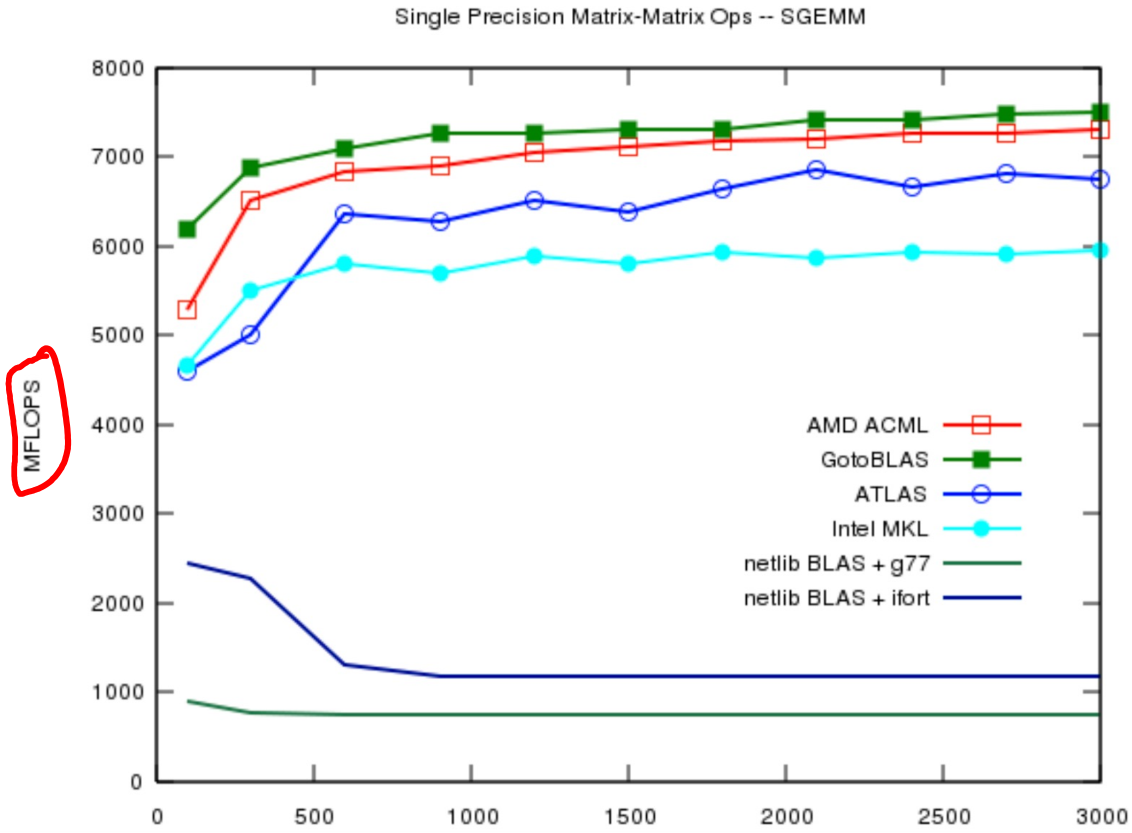
Also included in Level 3 are routines for solving

$$\boldsymbol{B} \leftarrow \alpha \boldsymbol{T}^{-1} \boldsymbol{B}$$

where $\boldsymbol{T}$ is a triangular matrix, among other functionality.

BLAS performance is very much system dependent, e.g., https://www.hoffman2.idre.ucla.edu/blas_benchmark/

Versions of BLAS compared: BLAS library from the Netlib Repository, ATLAS library, Intel-MKL library, AMD ACML Library and Goto BLAS.



Recap : Multivariate Linear Regression in a Nutshell

Data: X

Task: y

Representation: : x, f()

Score Function: L()

Search/Optimization : argmin()

Models, Parameters

X: Tabular

Regression: y continuous

Y = Weighted linear sum of Xs

Sum of Squared Error (Least Squared)

Normal Equation / GD / SGD

Regression coefficients w, b

Thank You

References

- Big thanks to Prof. Eric Xing @ CMU for allowing me to reuse some of his slides

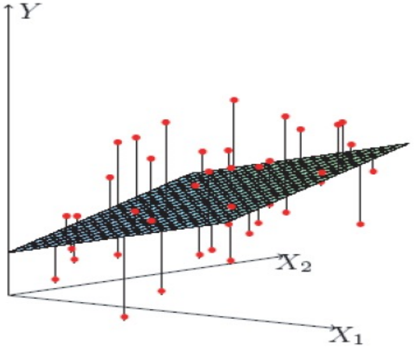
❑ http://www.cs.cmu.edu/~zkolter/course/15-884/linalg-review.pdf

❑ Prof. Alexander Gray's slides

EXTRA

In Case you are interested in more advanced details!

Probabilistic Interpretation of Linear Regression (Extra)



- Let us assume that the target variable and the inputs are related by the equation:

$$y_i = \theta^T \mathbf{x}_i + \varepsilon_i$$

error data on each point

where ε is an error term of unmodeled effects or random noise

- Now assume that ε follows a Gaussian N(0,σ), then we have:

$$p(y_i \mid x_i;\theta) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(y_i - \theta^T \mathbf{x}_i)^2}{2\sigma^2} \right)$$

- By iid (among samples) assumption:

$$L(\theta) = \prod_{i=1}^{n} p(y_i \mid x_i;\theta) = \left(\frac{1}{\sqrt{2\pi\sigma}} \right)^n \exp\left(-\frac{\sum_{i=1}^{n} (y_i - \theta^T \mathbf{x}_i)^2}{2\sigma^2} \right)$$

Many more variations of LinearR from this perspective, e.g. binomial / poisson (LATER)

Review (I):

- Sum the Squared Elements of a Vector equals Vector dot product to itself

$\vec{a} = X\theta - y$

$$\mathbf{a} = \begin{bmatrix} 5 \\ 2 \\ 8 \end{bmatrix}$$

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta - y_i)^2$$

$$a^T = \begin{bmatrix} 5 & 2 & 8 \end{bmatrix}$$

$$\vec{a}^T\vec{a} = \sum_{i=1}^{n} a_i^2$$

$$a^T\mathbf{a} = \begin{bmatrix} 5 & 2 & 8 \end{bmatrix}\begin{bmatrix} 5 \\ 2 \\ 8 \end{bmatrix} = 5^2 + 2^2 + 8^2 = 93$$

Review(I) :

$$a = \begin{bmatrix} \mathbf{x}_1^T\theta - y_1 \\ \mathbf{x}_2^T\theta - y_2 \\ \vdots \\ \mathbf{x}_n^T\theta - y_n \end{bmatrix} = X\theta - \vec{y}$$

$$\mathbf{a}^T\mathbf{a} = \qquad 2J(\theta) = \sum_{i=1}^{n} (\mathbf{x}_i^T\theta - y_i)^2$$

$$J(\theta) = (X\theta - y)^T (X\theta - y)\frac{1}{2}$$

$$= ((X\theta)^T - y^T)(X\theta - y)\frac{1}{2}$$

$$= (\theta^T X^T - y^T)(X\theta - y)\frac{1}{2}$$

$$= \left(\theta^T X^T X\theta \underbrace{- \theta^T X^T y - y^T X\theta}_{} + y^T y\right)\frac{1}{2}$$

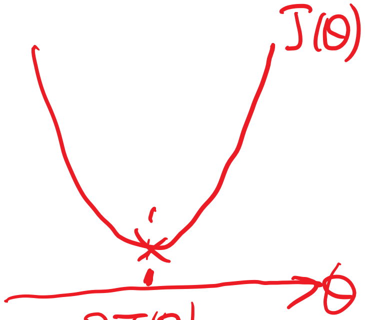
Since $\theta^T X^T y = y^T X\theta$

$\langle X\theta, y\rangle \quad \langle y, X\theta\rangle$

$$= \left(\theta^T X^T X\theta - 2\theta^T X^T y + y^T y\right)\frac{1}{2}$$

$\Rightarrow J(\theta)$ quadratic func of θ;

1d case



$$\frac{\partial J(\theta)}{\partial \theta} = 0$$

Review (II): gradient of linear form

$$\frac{\partial(\theta^T X^T y)}{\partial\theta} = X^T y$$

One Concrete Example

$$f(w) = w^T a = \left[w_1, w_2, w_3\right]\begin{pmatrix}1\\2\\3\end{pmatrix} = w_1 + 2w_2 + 3w_3$$

$$\frac{\partial f}{\partial w_1} = 1$$

$$\frac{\partial f}{\partial w_2} = 2$$

$$\frac{\partial f}{\partial w_3} = 3$$

$$\frac{\partial f}{\partial w} = \frac{\partial w^T a}{\partial w} = a = \begin{pmatrix}1\\2\\3\end{pmatrix}$$

Review (III): Gradient of Quadratic Form

- See L2-note.pdf -> Page 17, Page 23-24
- See white board

$$\frac{\partial(\theta^T X^T X\theta)}{\partial\theta} = \frac{\partial(\theta^T G\theta)}{\partial\theta} = 2G\theta = 2X^T X\theta$$

Review (III): Single Var-Func to Multivariate

Single **Var-Function**	Multivariate Calculus
Derivative Second-order derivative	Partial Derivative Gradient Directional Partial Derivative Vector Field Contour map of a function Surface map of a function Hessian matrix Jacobian matrix (vector in / vector out)

Review (IV) : Definitions of gradient (Matrix_calculus / Scalar-by-vector)

- Size of gradient is always the same as the size of variable

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n \quad \text{if } x \in \mathbb{R}^n$$

In principle, gradients are a natural extension of partial derivatives to functions of

multiple variables.

Review (V): Rank of a Matrix

- rank(A) (the rank of a m-by-n matrix A) is

  = The maximal number of linearly independent columns

  =The maximal number of linearly independent rows

- If A is n by m, then
  - rank(A)<= min(m,n)
  - If n=rank(A), then A has full row rank
  - If m=rank(A), then A has full column rank

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Rank=?

$$\begin{pmatrix} 2 & 1 \\ 4 & 2 \end{pmatrix}$$

Rank=?

If A is n*n, rank(A)=n iff A is invertible

rank(AB) <= min(rank(A), rank(B))

Extra: Loss J() is Convex

$$\Rightarrow J(\theta) = \frac{1}{2}\left(\theta^T X^T X \theta - 2\theta^T X^T y + y^T y\right)$$

Gram matrix

$\Downarrow$

$$\Rightarrow \text{Hessan}\,(J(\theta)) = X^T X \quad [\text{PSD}]$$

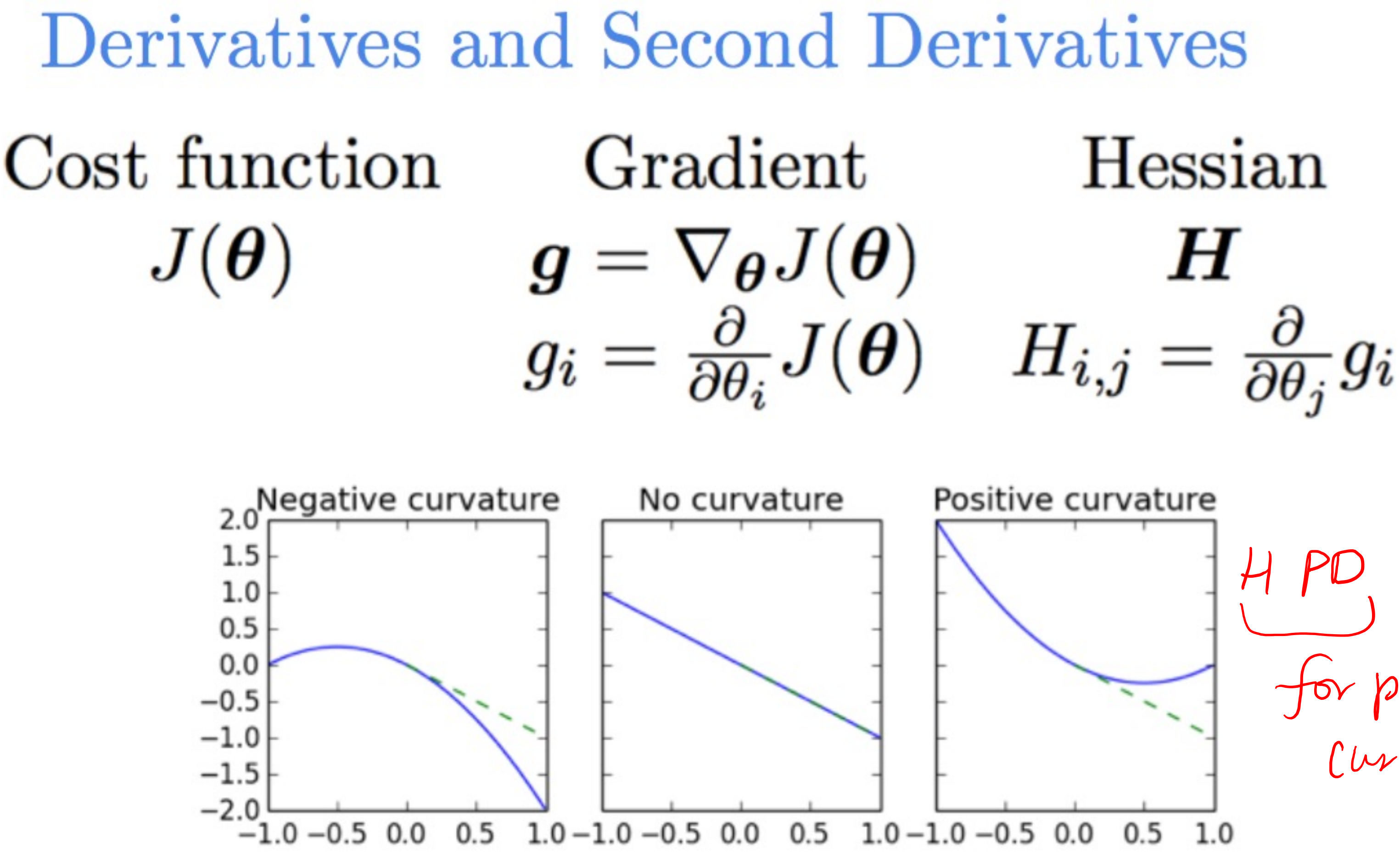
$\Downarrow$

$J(\theta)$ is convex

$\Downarrow$

If $\nabla J(\theta^*) = 0$, $J(\theta)$ is minimized @ θ^*

Review: Hessian Matrix



Derivatives and Second Derivatives

Cost function

$$J(\boldsymbol{\theta})$$

Gradient

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$g_i = \frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta})$$

Hessian

$$\boldsymbol{H}$$

$$H_{i,j} = \frac{\partial}{\partial \theta_j} g_i$$

$\underbrace{H \text{ PD}}_{\text{for positive curvature}}$

Positive Definite Hessian

Extra: Convex function

- Intuitively, a convex function (1D case) has a single point at which the derivative goes to zero, and this point is a minimum.
- Intuitively, a function f (1D case) is convex on the range [a,b] if a function’s second derivative is positive every-where in that range.

- Intuitively, if a multivariate function's Hessians is pd (positive definite!), this (multivariate) function is Convex
  - Intuitively, we can think “Positive definite” matrices as analogy to positive numbers in matrix case

Our loss function J() ’s Hessian is Positive Semi-definite - PSD

$H = X^T X$

is always

PSD

Intuitively, a convex function with PD hessian is minimized @ point whose

✓ derivative (slope) is zero

✓ gradient is zero vector (multivariate case)

Gram Matrix $\mathrm{H}= X^{T}X$

when X is full rank, H is positive definite!

Extra: Gram Matrix H= $X^T X$ is always positive semi-definite!

Because for any vector a

$$a^T X^T Xa =| Xa |_2^2 \geq 0$$

Besides, when X is full rank,
H is Positive Definite (PD) and invertible

Comments on the normal equation

when X full rank

$$\theta^* = \left(X^T X\right)^{-1} X^T \vec{y}$$

- In most situations of practical interest, the number of data points n is larger than the dimensionality p of the input space and the matrix X is of full column rank. If this condition holds, then it is easy to verify that X^TX is necessarily invertible.

$n \gg p$

- The assumption that X^TX is invertible implies that it is positive definite, thus the critical point (by solving gradient to zero) we have found is a minimum.

- What if X has less than full column rank? → regularization (later).

when X not full rank, e.g.

e.g. when $p > n \Rightarrow \text{rank}(\underset{n\times p}{X}) < p \Rightarrow \text{rank}(\underbrace{X^T X}_{p\times p}) \leq \min\{\text{rank}(X)\} < p$

$\Rightarrow$ not invertible

Extra: positive semi-definite!

L2-Note: Page 17

$A \in \mathbb{R}^{n\times n}$, $\forall x \in \mathbb{R}^n$

If $\underset{1\times n}{x^T}\ \underset{n\times n}{A}\ \underset{n\times 1}{x} \geq 0$

$\Rightarrow$ A is positive semi-definite (PSD)

If $X^T A X > 0$

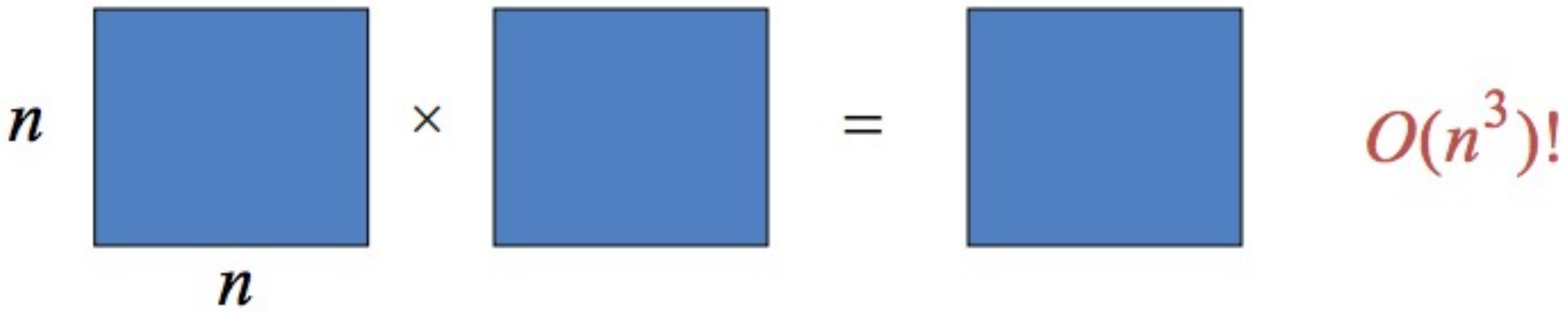
$\Rightarrow$ A is PD $\Rightarrow$ full rank / invertible

See proof on L2-Note: Page 18

Extra: Scalability to big data?

- Traditional CS view: Polynomial time algorithm, Wow!
- Large-scale learning: Sometimes even O(n) is bad! => Many state-of-the-art solutions (e.g., low rank, sparse, hardware, sampling, randomized…)

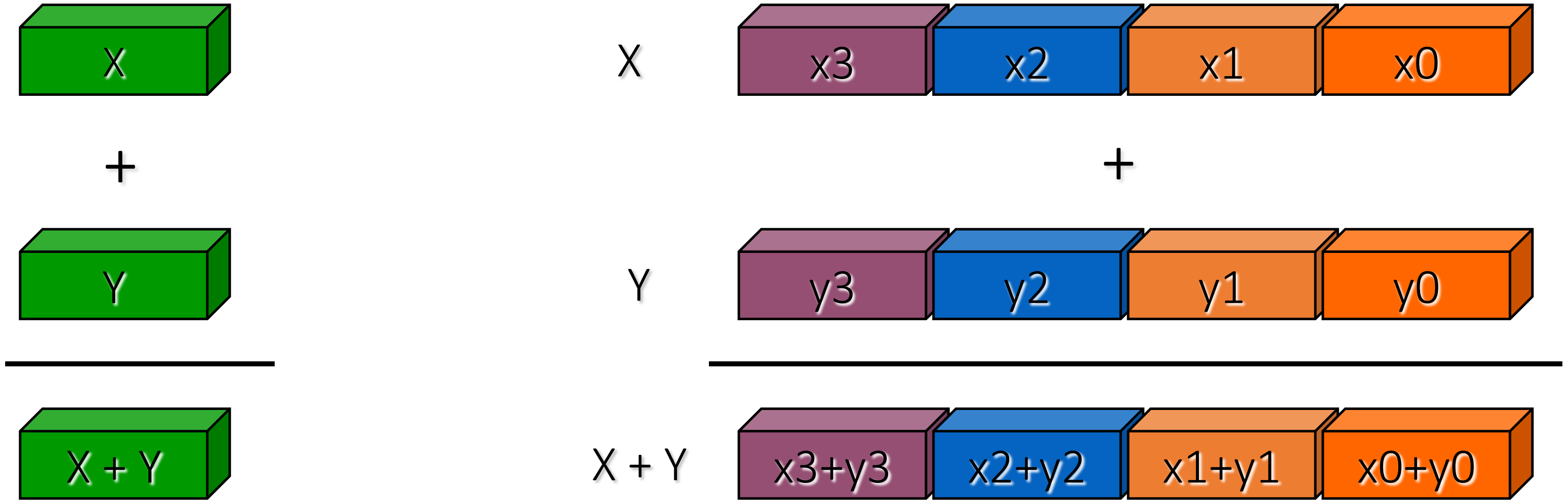
Simple example: Matrix multiplication



SIMD: Single Instruction, Multiple Data

- Scalar processing
  - traditional mode
  - one operation produces one result

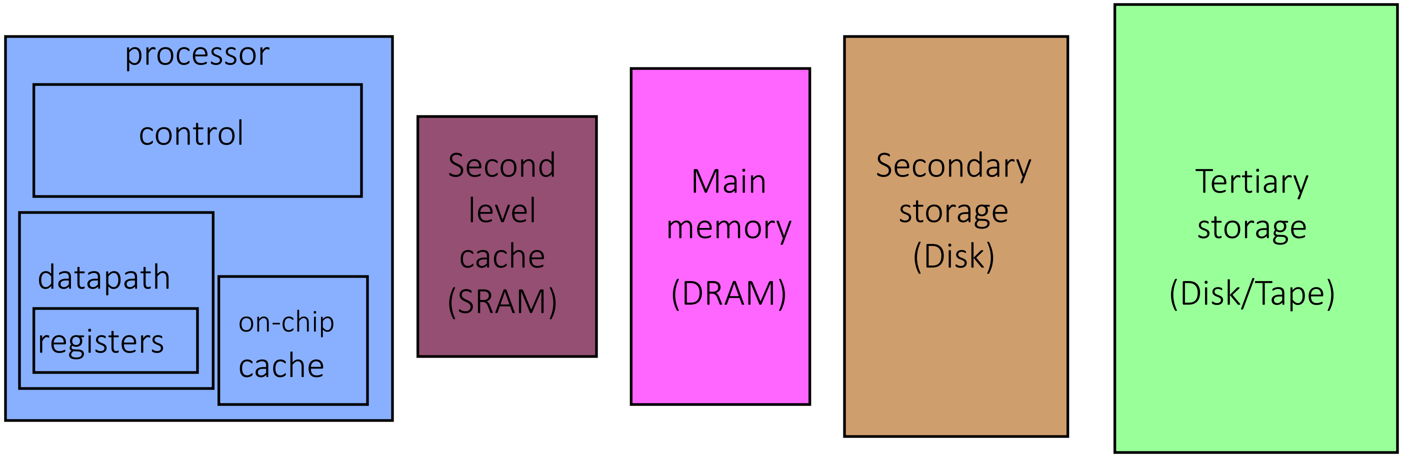
- SIMD processing
  - with SSE / SSE2
  - SSE = streaming SIMD extensions
  - one operation produces multiple results



Slide Source: Alex Klimovitski & Dean Macri, Intel Corporation

Memory Hierarchy

- Most programs have a high degree of locality in their accesses
  - **spatial locality:** accessing things nearby previous accesses
  - **temporal locality:** reusing an item that was previously accessed
- Memory hierarchy tries to exploit locality to improve average



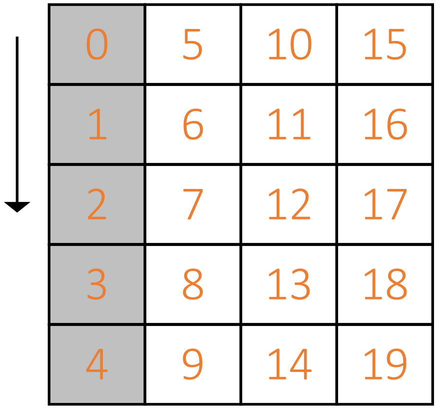
Speed	1ns	10ns	100ns	10ms	10sec
Size	KB	MB	GB	TB	PB

http://www.cs.berkeley.edu/~demmel/cs267_Spr12/

Note on Matrix Storage

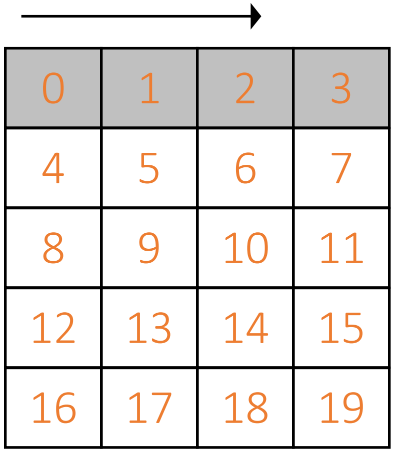
- A matrix is a 2-D array of elements, but memory addresses are “1-D”
- Conventions for matrix layout
  - by column, or “column major” (Fortran default); A(i,j) at A+i+j*n
  - by row, or “row major” (C default) A(i,j) at A+i*n+j
  - recursive (later)

Column major



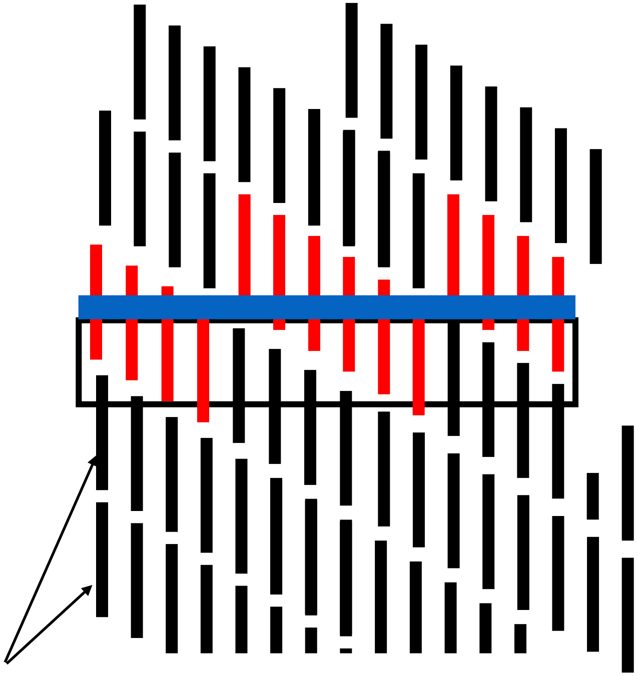
0	5	10	15
1	6	11	16
2	7	12	17
3	8	13	18
4	9	14	19

Row major



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

Column major matrix in memory



cachelines

Blue row of matrix is stored in red cachelines

- Column major (for now)

http://www.cs.berkeley.edu/~demmel/cs267_Spr12/

Strassen's Matrix Multiply

- The traditional algorithm (with or without tiling) has $O(n^3)$ flops
- Strassen discovered an algorithm with asymptotically lower flops
  - $O(n^{2.81})$
- Consider a 2x2 matrix multiply, normally takes 8 multiplies, 4 adds
  - Strassen does it with 7 multiplies and 18 adds

Let M = m11 m12 = a11 a12 b11 b12
m21 m22 a21 a22 b21 b22

Let p1 = (a12 - a22) * (b21 + b22)
p2 = (a11 + a22) * (b11 + b22)
p3 = (a11 - a21) * (b11 + b12)
p4 = (a11 + a12) * b22

p5 = a11 * (b12 - b22)
p6 = a22 * (b21 - b11)
p7 = (a21 + a22) * b11

Then m11 = p1 + p2 - p4 + p6
m12 = p4 + p5
m21 = p6 + p7
m22 = p2 - p3 + p5 - p7

Extends to nxn by divide&conquer

http://www.cs.berkeley.edu/~demmel/cs267_Spr12/

Strassen (continued)

$$
\begin{aligned}
\mathbf{T(n)} &= \textbf{Cost of multiplying nxn matrices} \\
&= 7^*T(n/2) + 18^*(n/2)^2 \\
&= O(n^{\log_2 7}) \\
&= O(n^{2.81})
\end{aligned}
$$

- Asymptotically faster
  - Several times faster for large n in practice
  - Cross-over depends on machine
  - "Tuning Strassen's Matrix Multiplication for Memory Efficiency", M. S. Thottethodi, S. Chatterjee, and A. Lebeck, in Proceedings of Supercomputing '98

- Possible to extend communication lower bound to Strassen
  - #words moved between fast and slow memory $= \Omega(n^{\log 2\ 7} / M^{(\log 2\ 7)/2 - 1}) \sim \Omega(n^{2.81} / M^{0.4})$
  - (Ballard, D., Holtz, Schwartz, 2011)
  - Attainable too

http://www.cs.berkeley.edu/~demmel/cs267_Spr12/