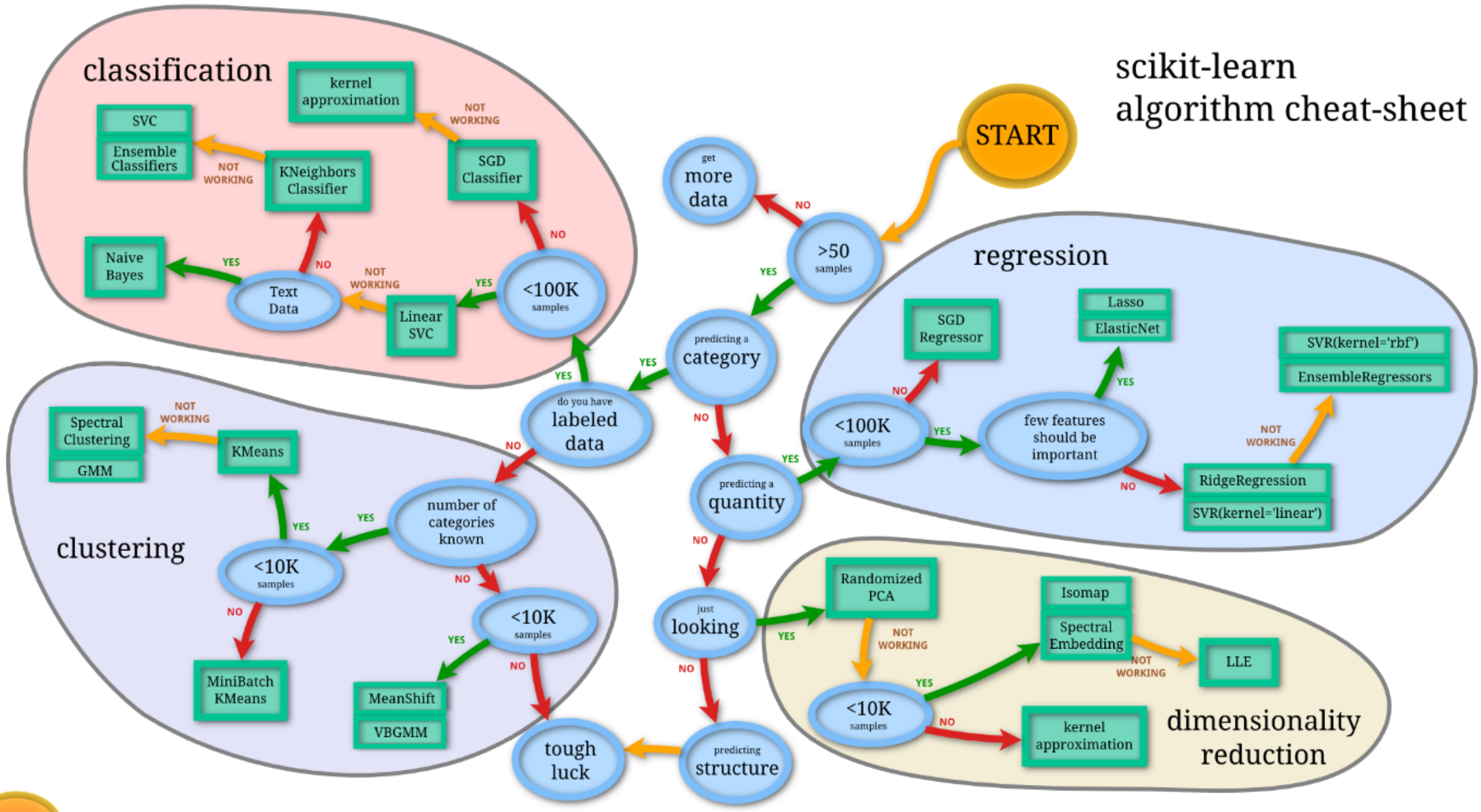
Machine Learning Workshops

< Scikit-Learn >

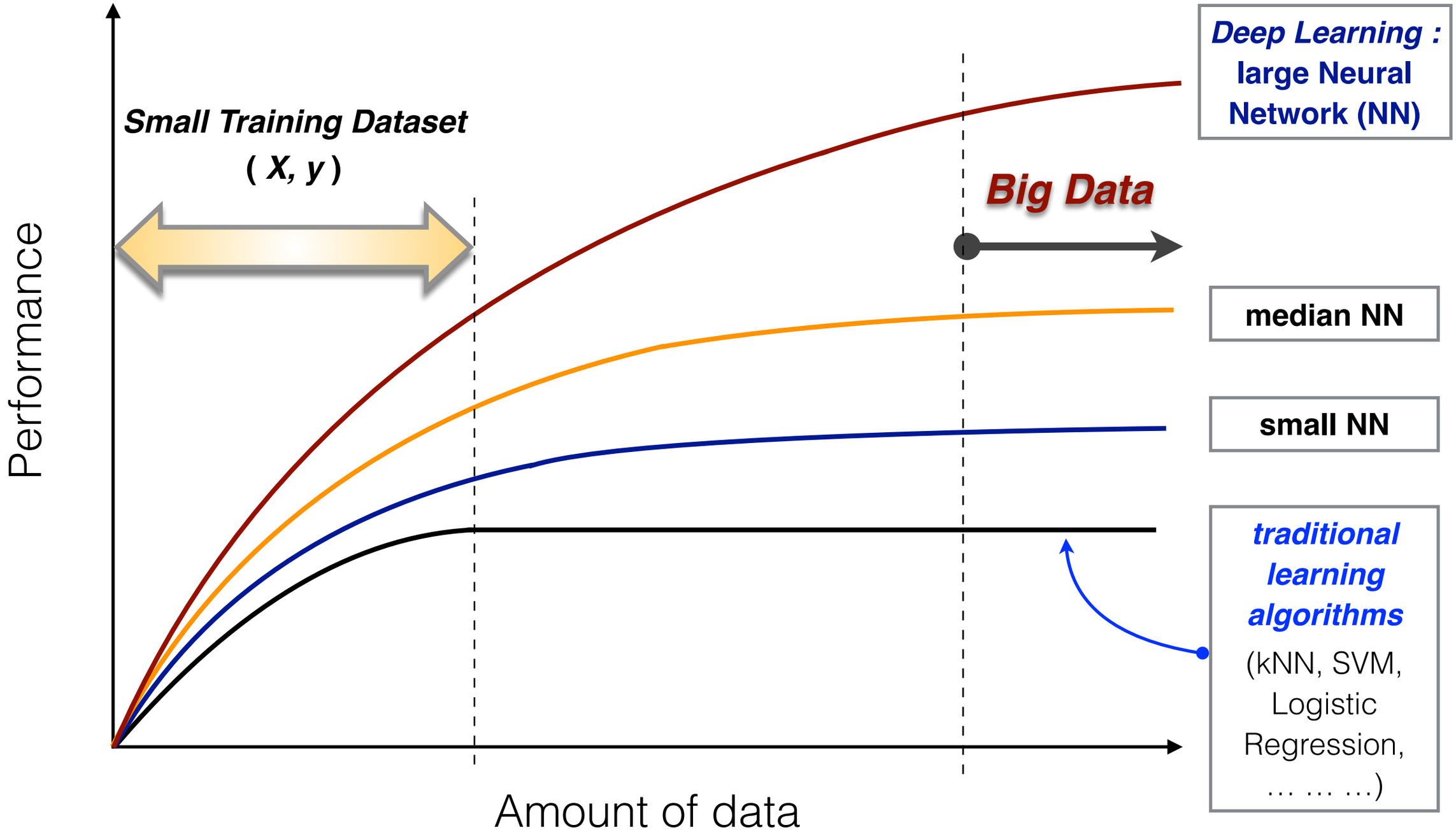
C. Alex Hu, PhD

Choosing the right estimator (選擇適當的估測器或演算法)



Scikit-Learn official site : **Choosing the right estimator**
(http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

After Scikit-Learn … Next, ***Deep Learning.***

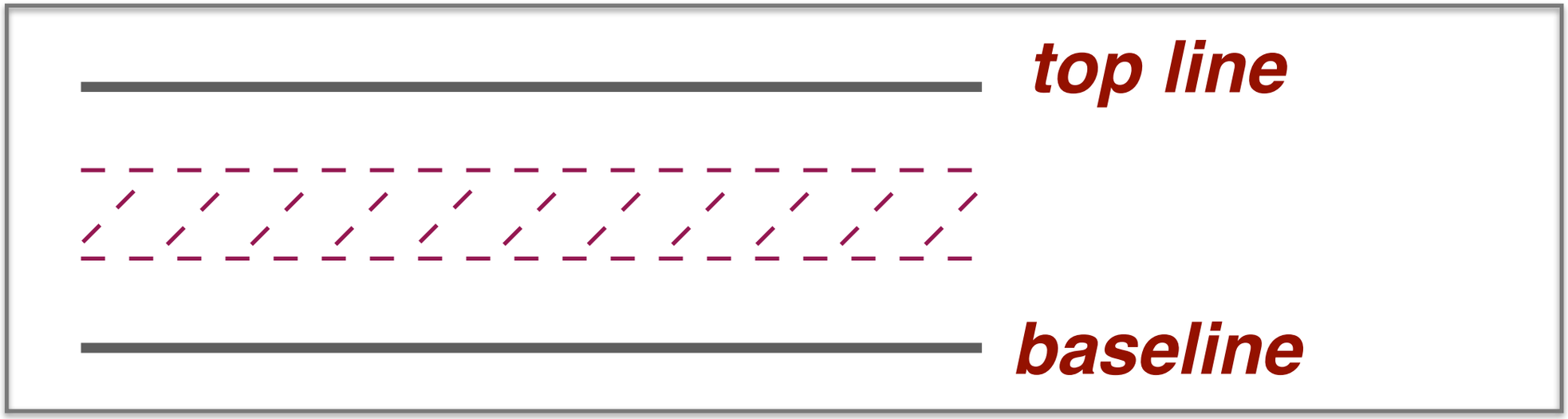


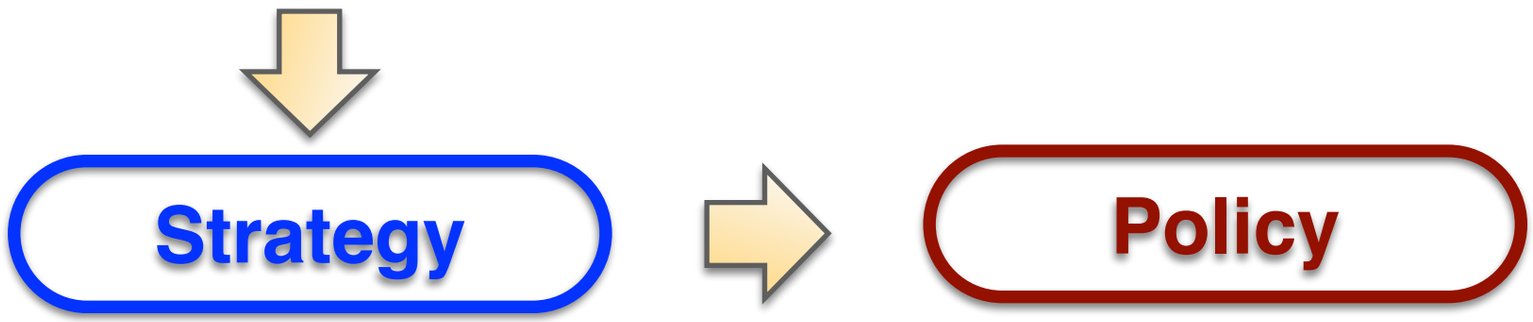
Deep Learning by Andrew Ng (吳恩達) [Full Course] — YouTube Video :
4. Drivers Behind the Rise of Deep Learning (https://youtu.be/j4-QFpTVCtM)

Data Analysis for Finding **Strategy to Resolving Problems**

1. Small Dataset => Sklearn Models => Baseline

2. Big Data => Predictive Models => Risk Analysis



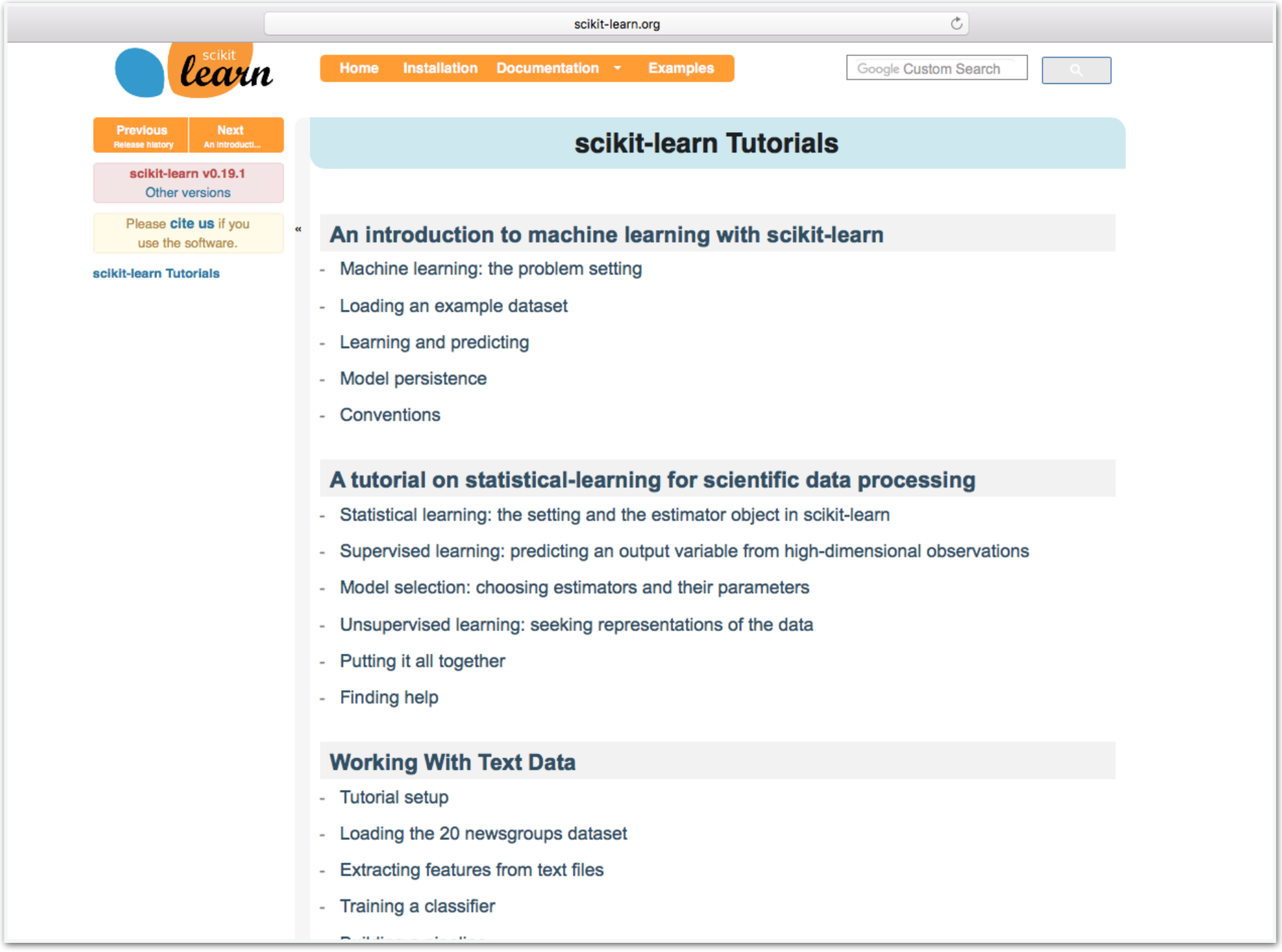


3. Policy => Execution => Risk Management

scikit-learn Tutorials

Sklearn Tutorials : http://scikit-learn.org/stable/tutorial/index.html

Code downloading : https://github.com/scikit-learn/scikit-learn



Scikit-Learn Workshops

[Scikit-Learn Workshop 1] : 監督式學習 — 迴歸模型

[Scikit-Learn Workshop 2] : 監督式學習 — 分類模型

[Scikit-Learn Workshop 3] : Validation Curves & Learning Curves

[Scikit-Learn Workshop 4] : 非監督式學習 — 集群分析

[Scikit-Learn Workshop 5] : 支持向量機 **(for Both)**

[Scikit-Learn Workshop 6] : Neural Networks **(for Both)**

[Scikit-Learn Workshop 7] : Meta-Learner - Random Forests

Chapter 11

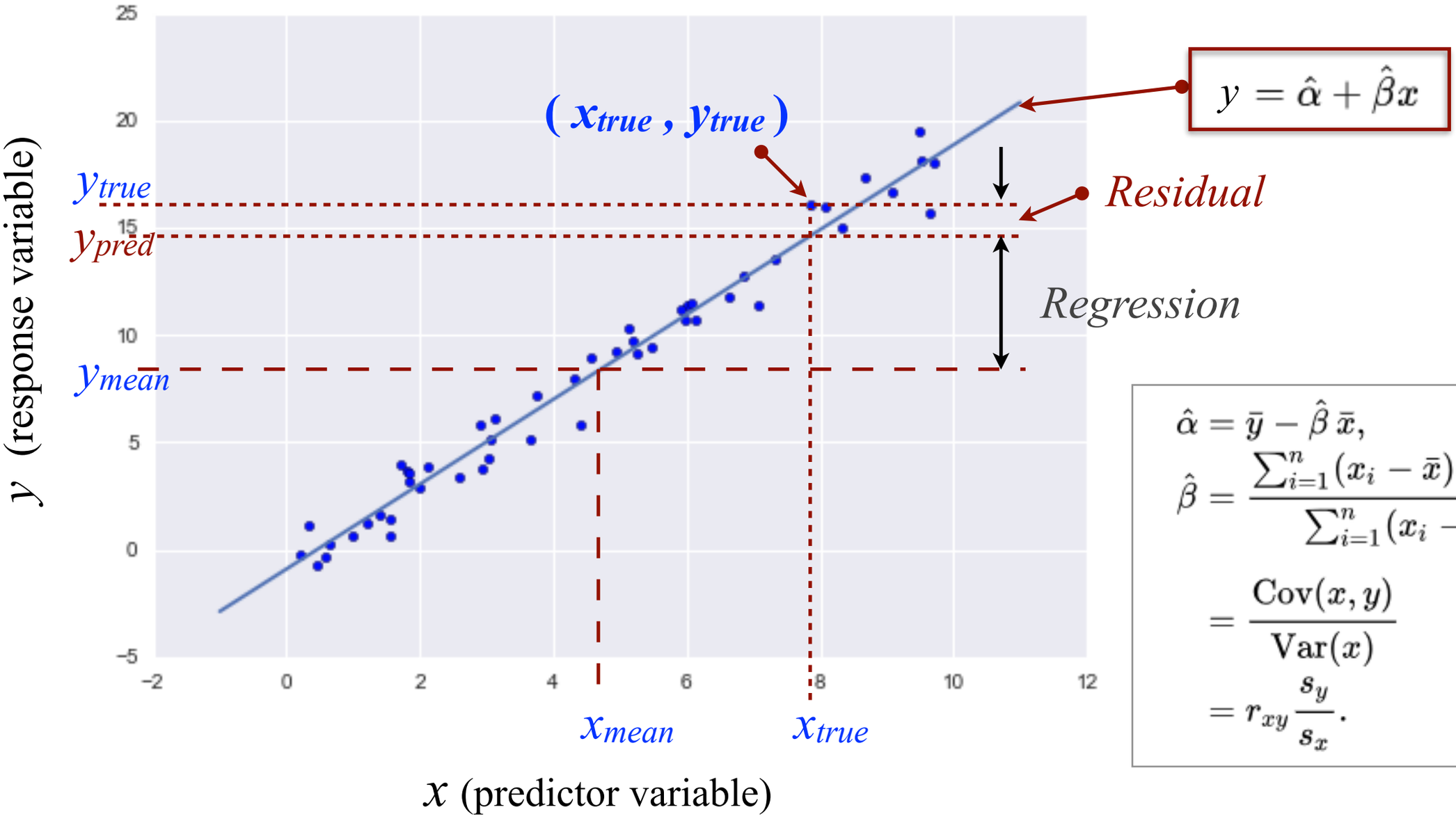
Scikit-Learn Workshop 1 :

監督式學習 — 迴歸模型

Scikit-Learn Workshop 1：監督式學習 — 迴歸模型

A. Simple Linear Regression (https://en.wikipedia.org/wiki/Simple_linear_regression)

from sklearn.**linear_model** import **LinearRegression**



$$\hat{\alpha} = \bar{y} - \hat{\beta}\,\bar{x},$$

$$\hat{\beta} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$= \frac{\operatorname{Cov}(x, y)}{\operatorname{Var}(x)}$$

$$= r_{xy}\frac{s_y}{s_x}.$$

Scikit-Learn Workshop 1：監督式學習 — 迴歸模型 (cont'd)

B. Basis Function Regression

(Ref : "05.06-Linear-Regression" from the Python Data Science Handbook by Jake VanderPlas)

— One trick you can use to adapt linear regression to nonlinear relationships between variables is to transform the data according to ***basis functions***.

— **The idea** is to take our multidimensional linear model:

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + \cdots$$

and build the x_1 , x_2 , x_3 , and so on, from our single-dimensional input x . That is, we let $x_n = f_n(x)$, where $f_n()$ is some function that transforms our data; for examples,

1. **Polynomial basis functions**
2. **Gaussian basis functions**

Scikit-Learn Workshop 1：監督式學習 — 迴歸模型 (cont'd)

1. Polynomial basis functions

(Ref : "05.06-Linear-Regression" from the Python Data Science Handbook by Jake VanderPlas)

If $f_n(x) = x^n$, our model becomes a ***polynomial regression*** :

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots$$

Notice that this is still a linear model — the ***linearity*** refers to the fact that the coefficients a_n never multiply or divide each other.

What we have effectively done is taken our one-dimensional x values and projected them into a higher dimension, so that a linear fit can fit more complicated relationships between x and y.

Scikit-Learn Workshop 1：監督式學習 — 迴歸模型 (cont'd)

2. Gaussian basis functions

(Ref： "05.06-Linear-Regression" from the Python Data Science Handbook by Jake VanderPlas)

Other basis functions are possible. For example, one useful pattern is to fit a model that is not a sum of polynomial bases, but a sum of Gaussian bases.

If $$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$,

our model becomes a ***Gaussian regression*** :

$$y = a_0 + a_1 f_1(x) + a_2 f_2(x) + a_3 f_3(x) + \cdots$$

Scikit-Learn Workshop 1：監督式學習 — 迴歸模型 (cont'd)

C. The Issue of "Regularization"

(Ref : "05.06-Linear-Regression" from the Python Data Science Handbook by Jake VanderPlas)

— The introduction of basis functions into our linear regression makes the model much more flexible, but it also can very quickly lead to ***over-fitting*.**

— For example, if we choose too many Gaussian basis functions, we end up with results that don't look so good.

— This is typical over-fitting behavior when basis functions overlap: the coefficients of adjacent basis functions blow up and cancel each other out.

— We could limit such spikes explicitly in the model by ***penalizing*** *large values of the model parameters*.

- **Ridge regression (L_2 regularization)**
- **Lasso regression (L_1 regularization)**

Scikit-Learn Workshop 1：監督式學習 — 迴歸模型 (cont'd)

[Ref. "**A Complete Tutorial on Ridge and Lasso Regression in Python**," AARSHAY JAIN, JANUARY 28, 2016
https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python/]

- **Linear Regression**

$$\hat{y}_i = \sum_{j=0}^{M} w_j * x_{ij}$$

$$Cost\ (W) = RSS(W) = \sum_{i=1}^{N} \{y_i - \hat{y}_i\}^2 = \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} w_j\, x_{ij} \right\}^2$$

Q : Why Penalize the Magnitude of Coefficients?
=> *Over-fitting* !!

- **Ridge regression**
  (L_2 regularization)

$$Cost\ (W) = RSS(W) + \lambda * (sum\ of\ squares\ of\ weights)$$

$$= \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} w_j\, x_{ij} \right\}^2 + \lambda \sum_{j=0}^{M} w_j^2$$

- **Lasso regression**
  (L_1 regularization)

usually for sparse data

$$Cost\ (W) = RSS(W) + \lambda * (sum\ of\ absolute\ value\ of\ weights)$$

$$= \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} w_j\, x_{ij} \right\}^2 + \lambda \sum_{j=0}^{M} |w_j|$$

Scikit-Learn Workshop 1：監督式學習 — 迴歸模型 (cont'd)

[EXAMPLE] : Predicting Bicycle Traffic

(Ref： "05.06-Linear-Regression" from the Python Data Science Handbook by Jake VanderPlas)

— We can ***predict*** *the number of bicycle trips across Seattle's Fremont Bridge based on weather, season, and other factors*.

— Join the bike data with another dataset, and try to determine the extent to which weather and seasonal factors—temperature, precipitation, and daylight hours—affect the volume of bicycle traffic through this corridor.