

Model Selection

อ. ปรัชญ์ ปิยะวงศ์วิศาล

Pratch Piyawongwisal

Today

- Recap – kNN
- Evaluation Metrics
- Cross-Validation
- Model Complexity
- Bias-Variance Tradeoff
- Lab: Cross-Validation for kNN

Recap: K-Nearest Neighbor Classifier

```
def train(train_images, labels):  
    # ML  
    return model
```

แอดจ์ดจำกทุก training data
(รูปภาพ + label) ไว้

```
def predict(model, test_images):  
    # use model to predict labels  
    return test_labels
```

ทำนายว่า test_image น่าจะมี label
เดียวกันกับภาพ train_image
ที่ดูคล้ายภาพนั้นที่สุด (top K matches)

ขั้นการ test/predict ใน kNN

X คือ feature ของข้อมูล (เช่น น้ำหนัก อายุ)
Y คือ label (เช่น เป็นมะเร็ง/สุขภาพดี)

- ในการ **test** เราจะหาเพื่อนบ้าน k คน ที่มีลักษณะใกล้เคียงที่สุด และดูว่าเพื่อนบ้านเป็นมะเร็ง (Y_1) หรือไม่เป็น (Y_0) มา ก ก ว ก ก น เป็นต้น
- ในการหาเพื่อนบ้านที่ใกล้ เรายังต้องคำนึงถึงวิธีการวัดระยะทางจากอื่นๆ -> อาจใช้ Euclidean (L2) Distance

$$d(x, x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \dots + (x_n - x'_n)^2}$$

- formally, เราต้องการหาพังก์ชัน $h: X \rightarrow Y$ ซึ่ง $h(x) = P(y = \text{มะเร็ง?} | X=x)$

$$P(y = j | X = x) = \frac{1}{K} \sum_{i \in \mathcal{A}} I(y^{(i)} = j)$$

โดยที่ $I()$ เป็น indicator จะเท่ากับ 1 ถ้า $y_i = j$ หรือเท่ากับ 0 ถ้า $y_i \neq j$

สรุปเกี่ยวกับ k-NN

- เป็นอัลกอริทึมแบบ
 - supervised
 - non-parametric
 - instance-based
- ใช้สำหรับทำ **classification** เป็นหลัก

X คือ **feature** ของข้อมูล (เช่น น้ำหนัก อายุ)
Y คือ **label** (เช่น เป็นมะเร็ง/สุขภาพดี)

มีผู้ช่วย (ใช้ข้อมูล **train** ที่มี **label** เฉลย ในการหา $h: X \rightarrow Y$)
ไม่มี **assumption** เกี่ยวกับหน้าตาของฟังก์ชัน $h: X \rightarrow Y$
ไม่มีโมเดลทางคณิตศาสตร์ที่ประกอบด้วยตัวแปร (parameter)
เรียนรู้โดยพิจารณาข้อมูลชุด **test** แต่ละ **instance** เทียบกับ
instance ที่เคยพบในข้อมูลชุด **train** โดยการจำ
สามารถจำแนกหมวดหมู่ (**classify**) ของข้อมูล
* แต่ใช้ประยุกต์ทำ **regression** ก็ได้

ข้อดี

- เข้าใจง่าย **implement** ง่าย
- ไม่ต้องเสียเวลา **train**
- มีความแม่นยำดีพอใช้

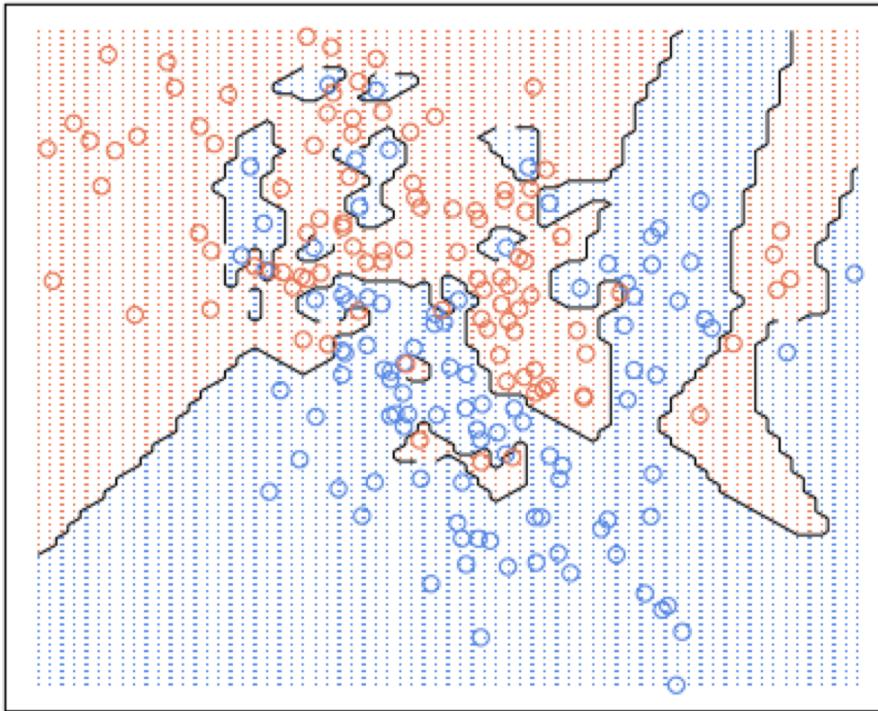
ข้อเสีย

- เปลือง **RAM** เนื่องจากต้องจำข้อมูลชุดฝึกทั้งหมด
- ใช้เวลา **predict** นาน -- $O(N)$ ขึ้นกับขนาดข้อมูลชุดฝึก
- sensitive** ต่อข้อมูลที่ผิดปกติ (**outlier**)

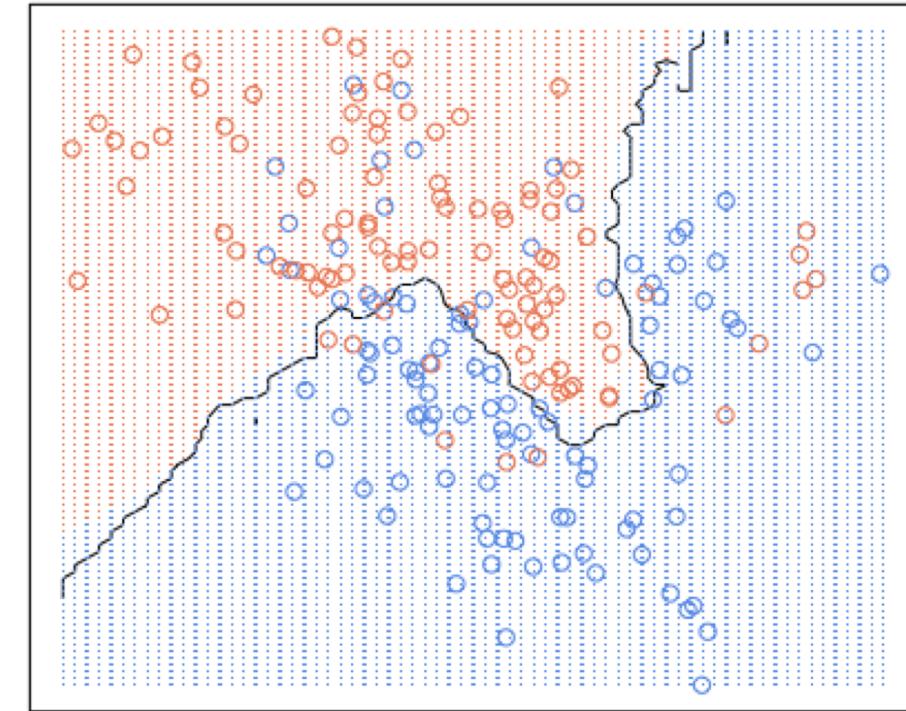
How to choose the best k?

- ค่าของ k เป็น hyperparameter ที่เราเลือกปรับได้ แต่เราจะเลือกค่าที่ดีสุดได้อย่างไร?

nearest neighbour ($k = 1$)



20-nearest neighbour



Evaluation Metrics

- ตัวชี้วัดประสิทธิภาพการทำงานของ Machine Learning algorithm
 - Accuracy
 - Training Error
 - Testing Error
 - Confusion Matrix
 - Logarithmic Loss
 - Precision, Recall, F1 score (ความถูกต้อง แม่นยำ)
 - Mean squared error
 - ROC/AUC

Evaluation Metrics

- ตัวชี้วัดประสิทธิภาพการทำงานของ **Machine Learning algorithm**
 - Accuracy = $\# \text{ครั้งที่预言ถูก} / \# \text{ครั้งที่预言ทั้งหมด}$ (ตรงข้ามกับ error)
 - Training Error = $\# \text{ข้อมูลชุด train ที่预言ผิด} / \# \text{ข้อมูลชุด train} \text{ ทั้งหมด}$
 - Testing Error = $\# \text{ข้อมูลชุด test ที่预言ผิด} / \# \text{ข้อมูลชุด test} \text{ ทั้งหมด}$
- Q: เราให้ความสำคัญกับ training error หรือ testing error 多少 กัน?

Choosing k

- **goal:** เรายังต้องการค่า k ที่ทำให้ test error ต่ำสุด
- **solution 1:**
ทำการ train, test กับข้อมูลชุดเดิมซ้ำๆ โดยเปลี่ยนค่า k ไปเรื่อยๆ แล้วหา k ที่ทำให้ test error ต่ำสุด

Problem?

```
# ลอง k ตั้งแต่ 1...10
for k in range(1,11):
    classifier.train(X_train, y_train)
    dists = classifier.compute_distances(X_test)
    y_test_pred = classifier.predict_labels(dists, k)
    accuracy = np.sum(y_test_pred == y_test)/len(y_test)
    print(accuracy) # เลือก k ที่ให้ค่า test accuracy สูงสุด
```

Choosing k

- **goal:** เรายังต้องการค่า k ที่ทำให้ **test error** ต่ำสุด
 - **solution 1:**
ทำการ **train**, **test** กับข้อมูลชุดเดิมซ้ำๆ โดยเปลี่ยนค่า k ไปเรื่อยๆ แล้วหา k ที่ทำให้ **test error** ต่ำสุด
 - **problem:** โมเดลเราอาจจะ **overfit** กับข้อมูลชุด **test** นั้น
- ❖ **Overfitting** คือการที่โมเดล **fit** กับข้อมูลชุดหนึ่งมากเกินไป ทำให้ **generalize** กับข้อมูลชุดอื่นๆ ไม่ได้

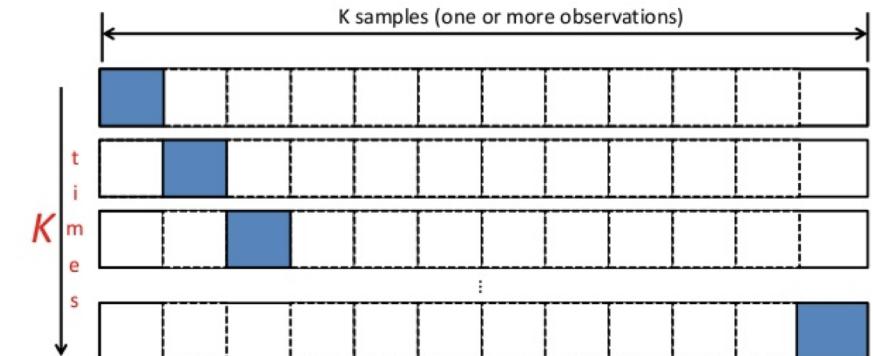
Cross-Validation

- **solution 2:**

- แบ่งข้อมูลเป็น 3 ส่วน คือ **train, validate, test**
- แยกชุด **test** ออกไปเก็บไว้ในกรุ ห้ามใช้จนกว่าจะ **train** เสร็จ
- ใช้ชุด **train, validate** ในการหาค่า **hyperparam** ที่ดีที่สุด
- ใช้ชุด **test** ในการประเมินสุดท้าย

Cross-validation: How it works?

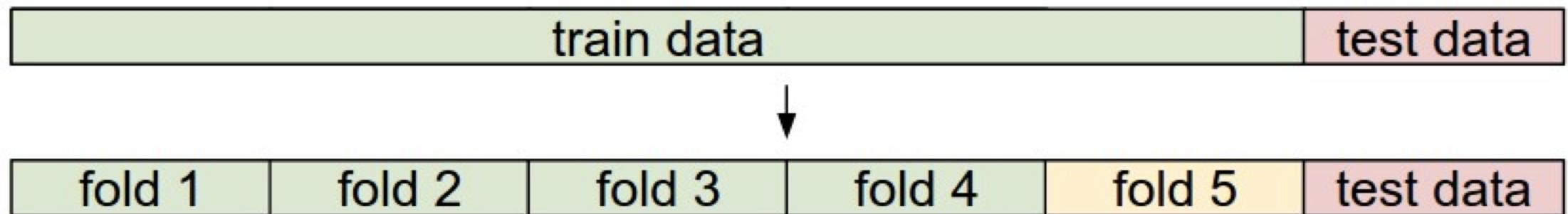
- **K-fold cross-validation:**



- สามารถเลือกจำนวน **fold** ในการแบ่งข้อมูลได้
 - เช่น **leave-one-out, 3-fold, 5-fold**

K-fold Cross-Validation

- 5-fold example



Fold 1-5 ผลักกันเป็น validate set (สีเหลือง)

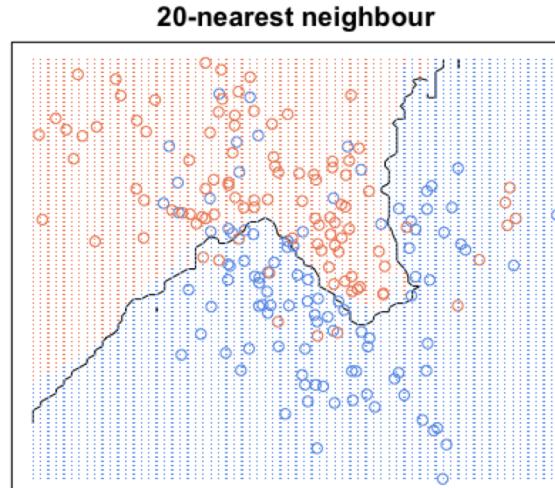
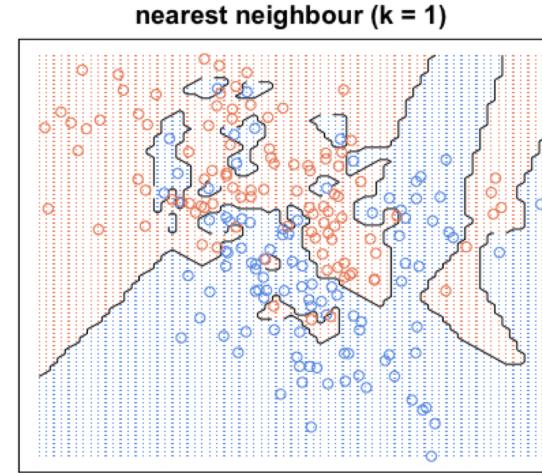
ตัวอย่าง: วิธีทำ 5-fold CV กับ MNIST, k-NN

- ชุดข้อมูล **digits** ใน **sklearn** ประกอบด้วยภาพ + label จำนวน **1797** ภาพ
 - แยกข้อมูล **297** ภาพไปเป็น **test set** (เก็บเข้ากรุ ใช้ทดสอบครั้งสุดท้ายครั้งเดียว)
 - นำข้อมูลที่เหลือ **1500** ภาพไปใช้ **train k-NN**
- กระบวนการ **train k-NN** โดยละเอียด
 - วนลูปค่า **k** ที่ลูปค่า เช่น จากเซ็ต $\{1, 3, 5, 7, 9, 11\}$
 - เริ่มจาก **k = 1**
 - ตัดแบ่งข้อมูล 1500 ภาพเป็น 5 ชุด (fold) ชุดละ 300 ภาพ
 - เลือก 1 ชุด (300 ภาพ) เป็น **validation set** และให้ที่เหลืออีก 4 ชุด (1200 ภาพ) เป็น **train set**
 - หาค่า **accuracy** จากการใช้ k-NN คำนายนายชุด validation set
 - ทำซ้ำโดยผลัดให้แต่ละชุดได้เป็น validation set ==> จะได้ accuracy มา 5 ค่า
 - หาค่า **accuracy** เฉลี่ย
 - ทำซ้ำสำหรับ **k=3, 5, ..., 11** ต่อไปจนครบ แล้วเราจะเลือก **k** ที่ให้ค่า **mean accuracy** สูงที่สุด
 - ใช้ค่า **k** นั้นในการคำนายนาย **test set** (เปิดกรุ) และหา **accuracy** ท้ายสุด

300	300	300	300	300	297
Train			Validate		Test

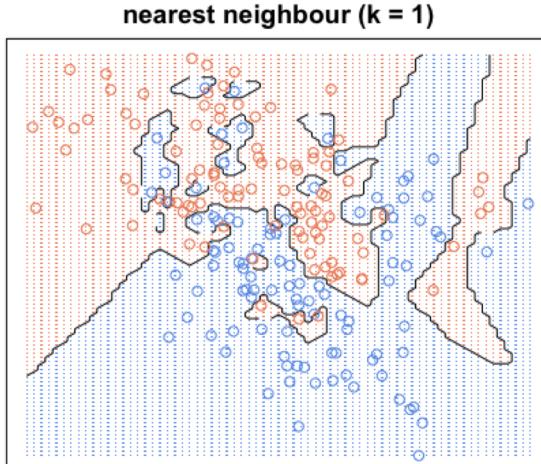
Model Complexity

- การปรับจูนค่า **hyperparameter** อาจส่งผลต่อ **model complexity**
 - model ที่มี **complexity** สูง จะมี **assumption** เกี่ยวกับข้อมูลที่มาก
 - เช่น 1-NN
 - model ที่มี **complexity** ต่ำ จะมี **assumption** ที่ผ่อนปรนกว่า
 - เช่น 20-NN
- ❖ สังเกตได้จากความซับซ้อนยุ่งเหยิงของ **decision boundary**
- ❖ **model complexity** มีผลต่อความสามารถในการ **generalize** ของโมเดล

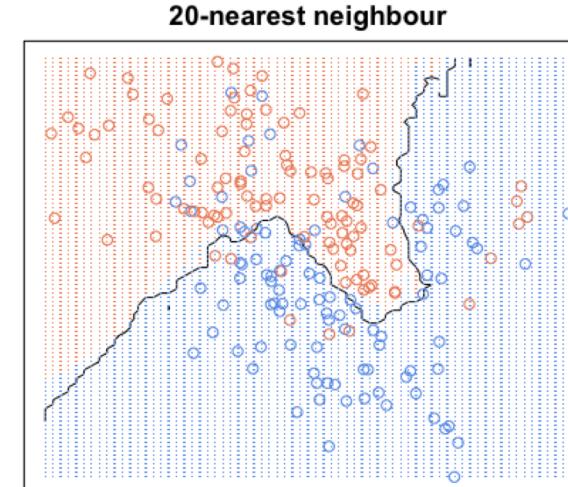


Bias-Variance Tradeoff

- ด้วยทฤษฎีทางสถิติ พบว่าเราสามารถแทรก generalization error (expected test error) ของโมเดลออกเป็นจาก 3 แหล่ง ได้แก่
 - bias = ทำนายผิดเพราะ model มี assumption เกี่ยวกับข้อมูลที่ผิด เรียบง่ายไป (complexity ต่ำ) ทำให้เกิดการ underfit (ภาพขวา)
 - variance = ทำนายผิดเพราะ model อ่อนไหวต่อ variation ในข้อมูลเกินไป จึงเกิด overfit (ภาพซ้าย)
 - irreducible error = ทำนายผิดเพราะ noise ในข้อมูลตามธรรมชาติ จะปรับโมเดลอีกต่อไปไม่ได้



low bias 😊
high variance 😞



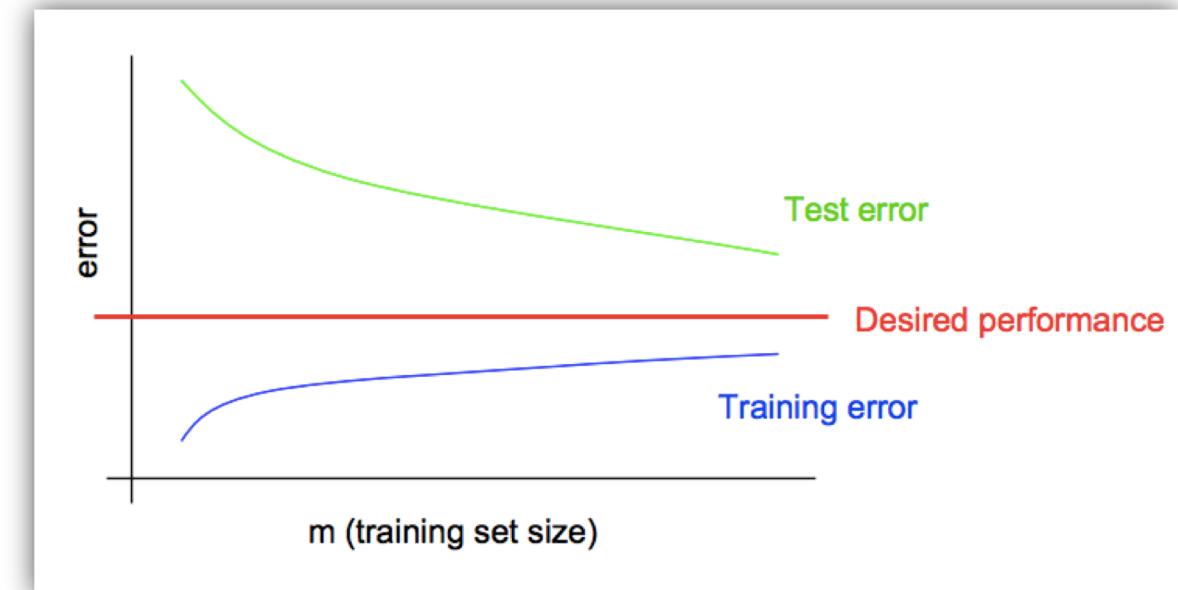
high bias 😞
low variance 😊

Learning curve

- Accuracy vs Training size plot



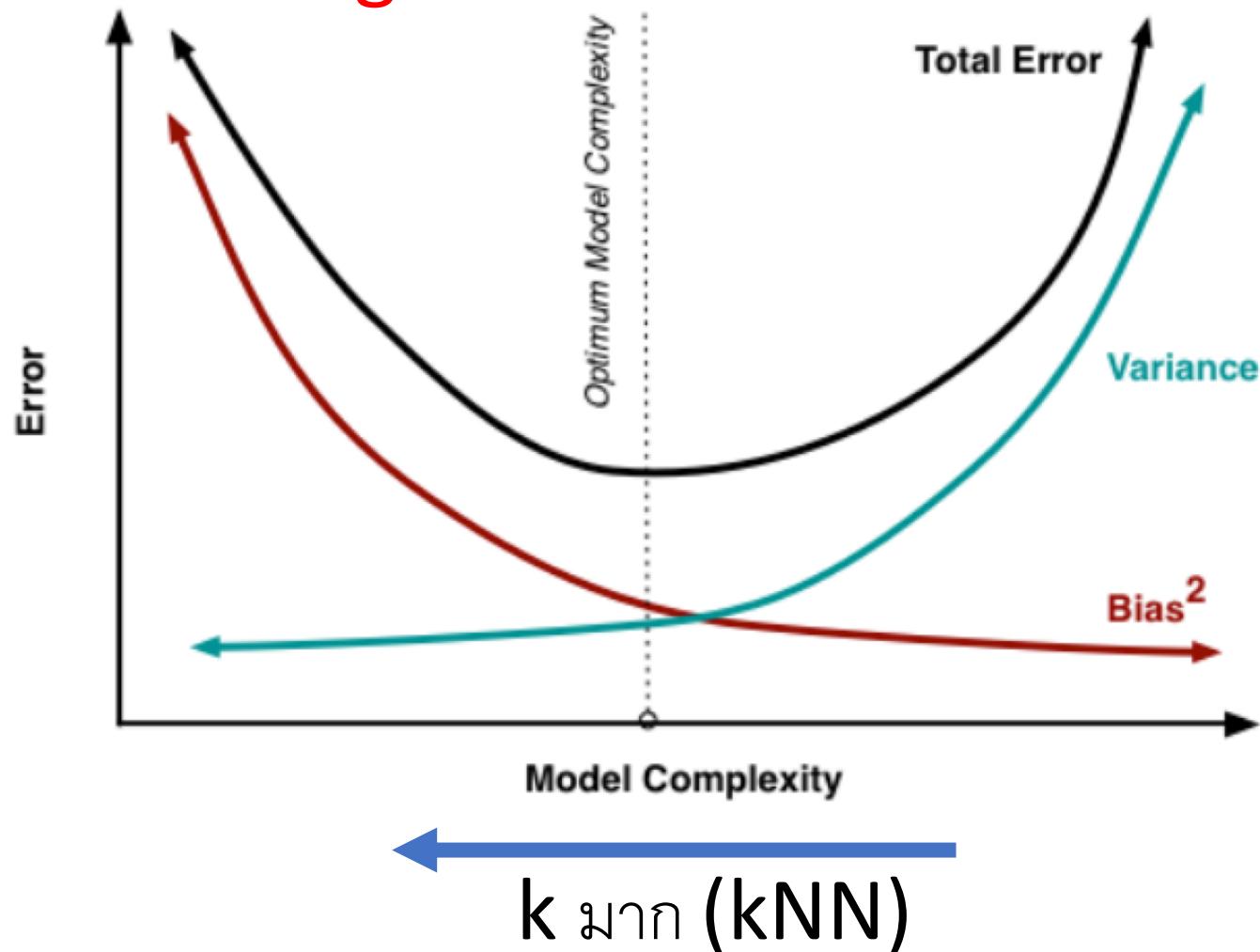
High Bias

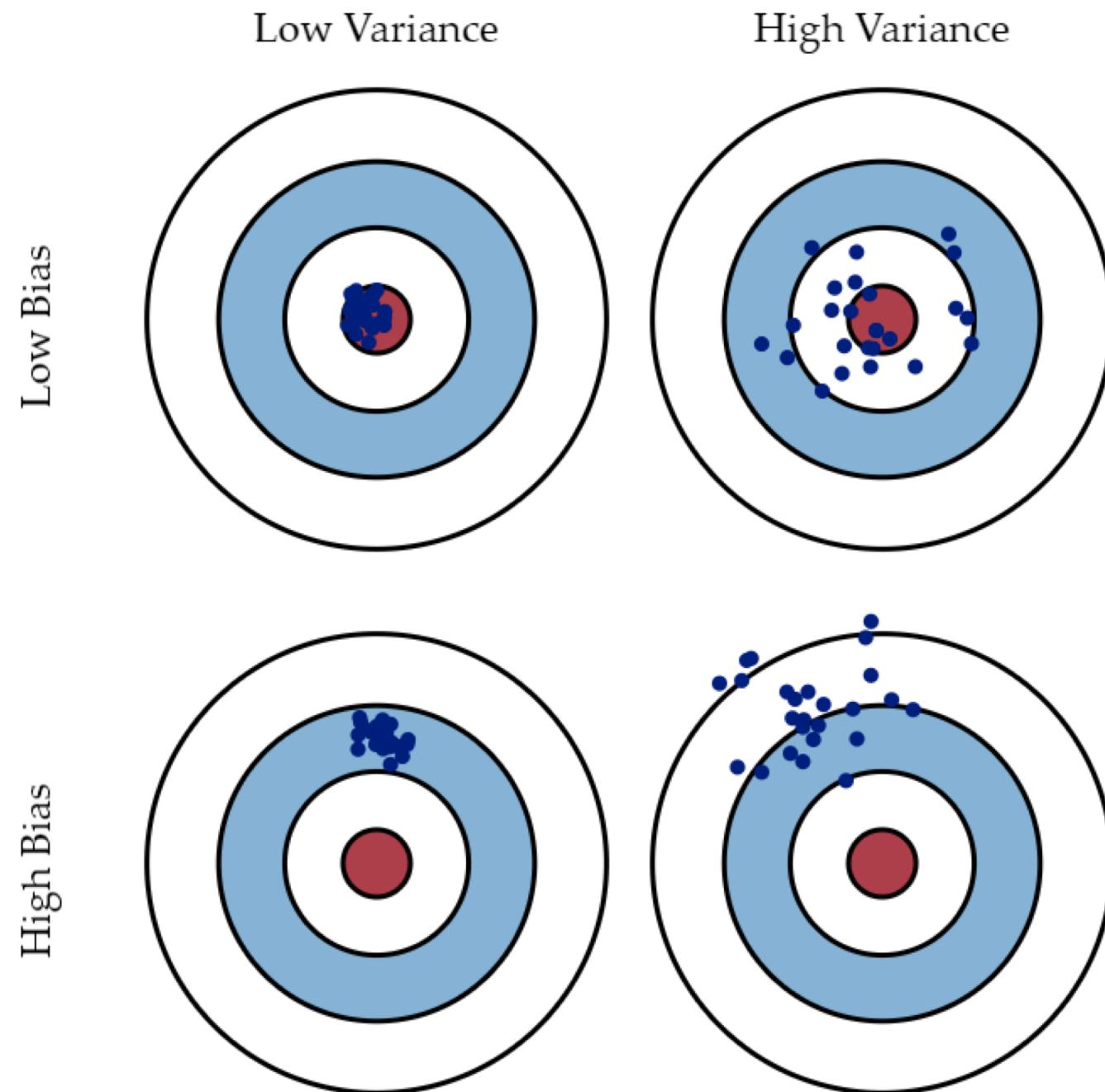


High Variance

Bias-Variance Tradeoff

underfitting overfitting





Bias-Variance Tradeoff (Math)

- Assuming true relationship is $Y = f(x) + \varepsilon$
 - where noise $\varepsilon \sim N(0, \sigma^2)$

- We want to make a model $\hat{f}(x)$ of f

- Expected squared error is $Err(x) = E[(Y - \hat{f}(x))^2]$

- which can be decomposed as

$$Err(x) = \left(E[\hat{f}(x)] - f(x) \right)^2 + E \left[(\hat{f}(x) - E[\hat{f}(x)])^2 \right] + \sigma^2$$

Bias

Variance

Irreducible
Error

Lab: Cross-validation for kNN

- ทำการ implement cross-validation เพื่อหาค่า k ที่เหมาะสม
- ลอง $k = 1, 3, 5, 8, 10, 12, 15, 20, 50, 100$
- กำหนด `num_folds` เป็นจำนวน fold
- ให้ `num_folds = 5`

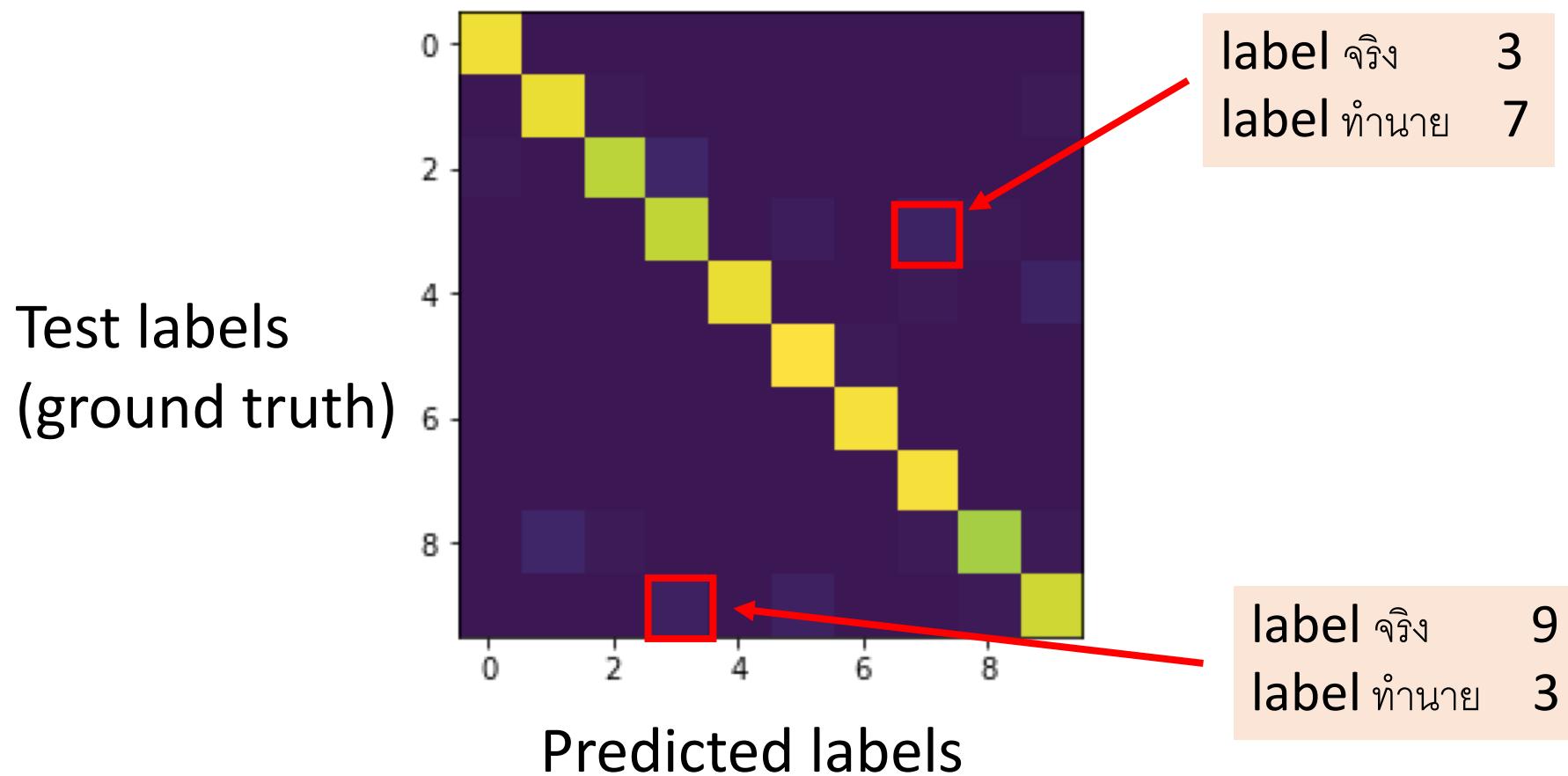
300	300	300	300	300	297
Train			Validate	Test	

Extra: การ implement kNN ด้วย scikit-learn

- โหลดข้อมูล MNIST ซึ่งประกอบด้วย
 - images ภาพตัวเลข
 - target เฉลย
- แบ่งข้อมูลเป็นชุด train ชุด test
- สร้าง KNeighborsClassifier
- ทำการ train โดยเรียก `classifier.fit(ชุด train)`
- ทำการ test โดยเรียก `classifier.predict(ชุด test)`
- เปรียบเทียบผลการ `predict` กับเฉลยใน `target` และประเมินประสิทธิภาพ
 - $\text{accuracy} = \frac{\text{จำนวนชุด test ที่ถูกต้อง}}{\text{จำนวนชุด test ทั้งหมด}}$
 - ใช้ `metrics.confusion_matrix(เฉลย, ทาย)` เพื่อหาว่าทำนายผิดอย่างไร

Confusion Matrix

- ใช้แสดงอัตราทำนายถูกผิดของแต่ละค่า **label** ระหว่างทุกคู่ **prediction** กับ **ground truth**



Confusion Matrix

- ตัวอย่างภาพจาก MNIST dataset

	predicted 0	predicted 1	predicted 2	predicted 3	predicted 4	predicted 5	predicted 6	predicted 7	predicted 8	predicted 9
actual 0	0			0	0	0				0
actual 1		1 1	1		1 1					
actual 2	2	2 2	2		2 2	2	2	2	2	2
actual 3	3	3	3 3	3 3	3 3	3 3	3 3	3 3	3 3	3
actual 4	4		4		4 4	4	4			
actual 5	5	5	5 5	5 5	5 5	5 5	5			
actual 6	6	6	6		6 6	6	6			
actual 7		7 1					7			
actual 8	8	8 8	8 5	8 8	8 8	8 8	8 8			
actual 9	9	9 9	9 9	9 9	9 9	9 9	9 9	9 9	9 9	9

Next week

- Linear Regression
- Gradient Descent
- Logistic Regression