

K-Nearest Neighbor Classifier

อ. ปรัชญ์ ปิยะวงศ์วิศาล

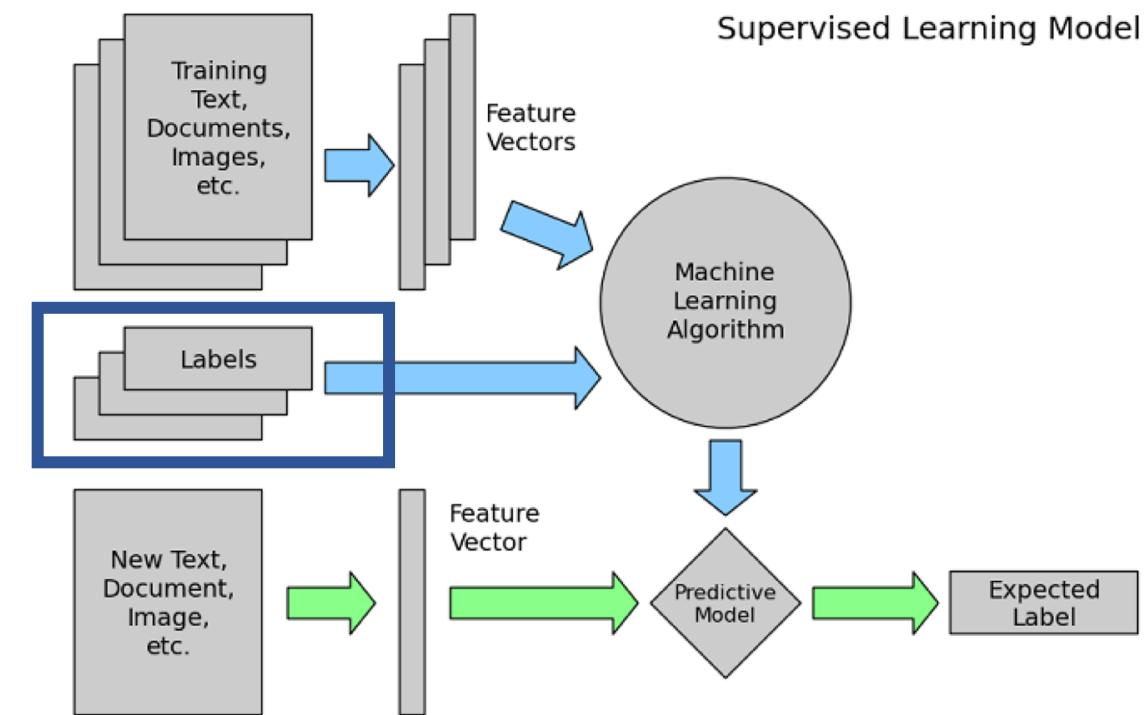
Pratch Piyawongwisal

Today

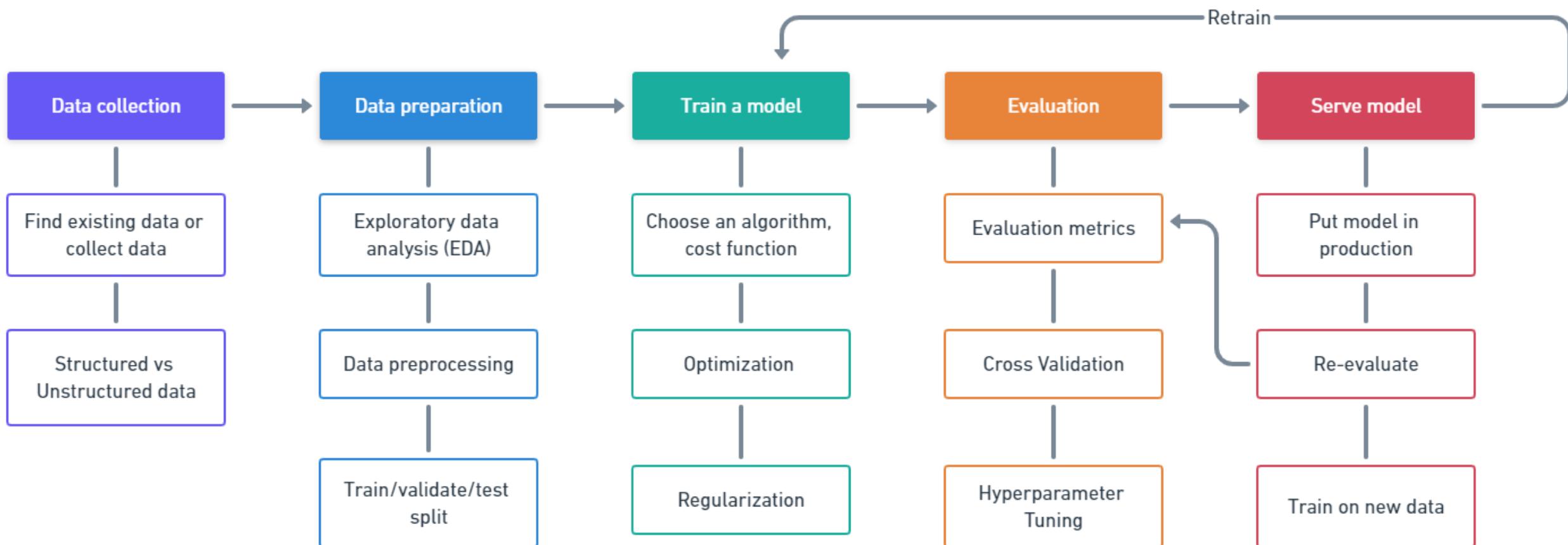
- Recap – Supervised Learning
- ML Process Overview
- Image Classification Problem
- k-NN classifier
- Scikit-learn tutorial with MNIST dataset

ML Basics

- **Supervised vs Unsupervised Learning**
- **Classification vs Regression**
- **Training Set vs Test Set**



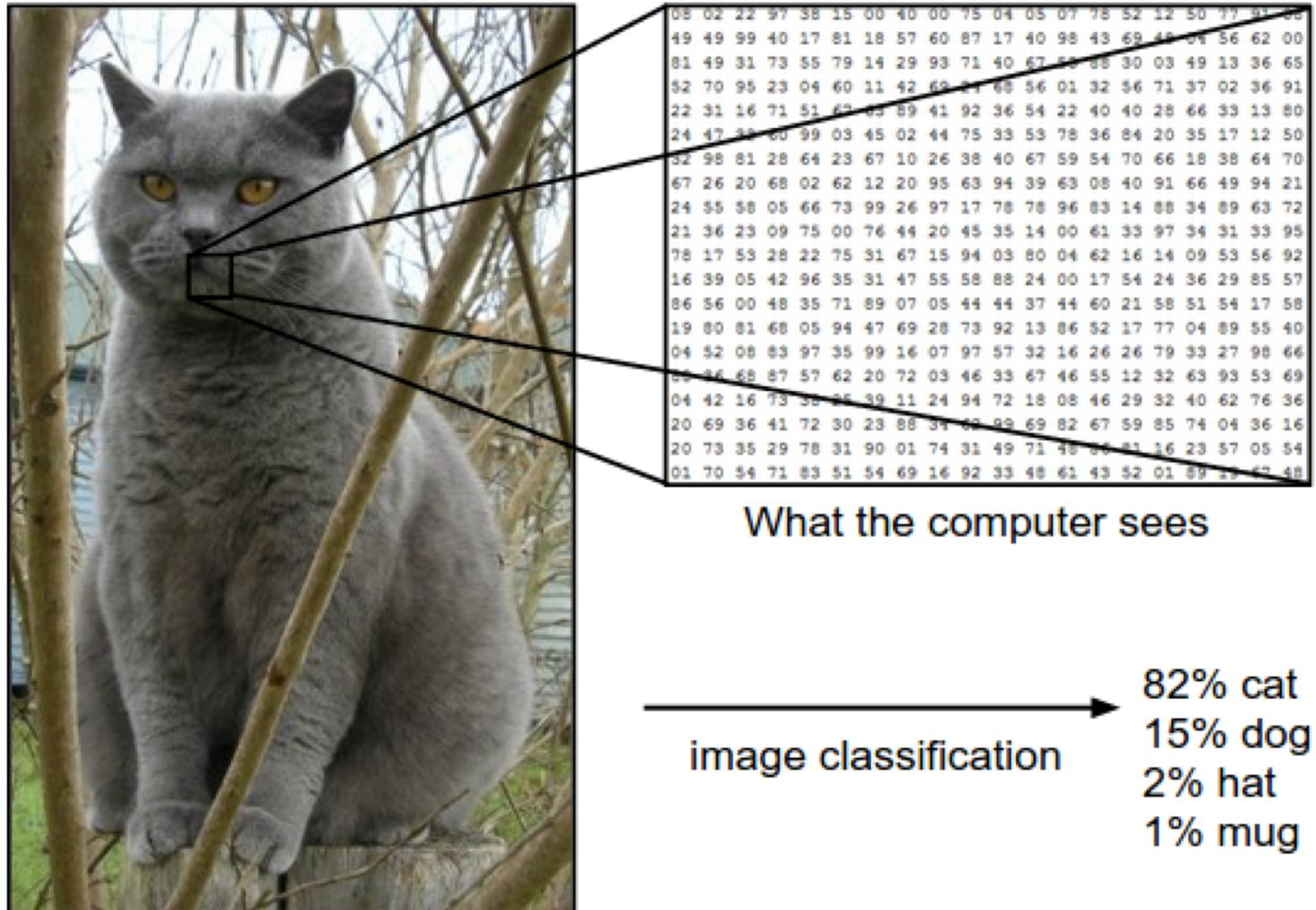
ML Process Overview



Supervised Learning Algorithms for Classification (Classifiers)

- k-Nearest Neighbors (k-NN)
- Logistic Regression
- Support Vector Machines (SVM)
- Decision Trees
- Neural Network

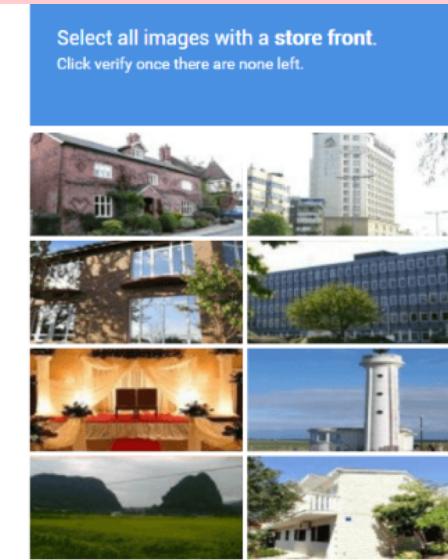
Example: Image Classification Problem



Source: <http://cs231n.github.io/classification/>

- การจำแนกภาพเป็น task หลักในงานด้าน Computer Vision

Q: ทำไมการจำแนกภาพจึงยากสำหรับคอมพิวเตอร์ ทั้งๆ ที่สำหรับคนนั้นเป็นเรื่องง่ายมาก?



Please also check the new images.

ทำไมปัญหาการจำแนกภาพจึงยากสำหรับคอมพิวเตอร์ ?

Viewpoint Variation



Deformation



Occlusion



Illumination



Intraclass Variation



Background Clutter



Solution:

Data-Driven Approach (Machine Learning)

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

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

Example training set

airplane



automobile



bird



cat



deer



CIFAR-10 Dataset

Nearest Neighbor Classifier



?

```
def train(train_images, labels):  
    # memorize training data  
    return model
```

```
def predict(model, test_images):  
    # compute distance and find  
    # nearest neighbors  
    return test_labels
```

แค่จดจำทุก **training data**
(รูปภาพ + label) ไว้



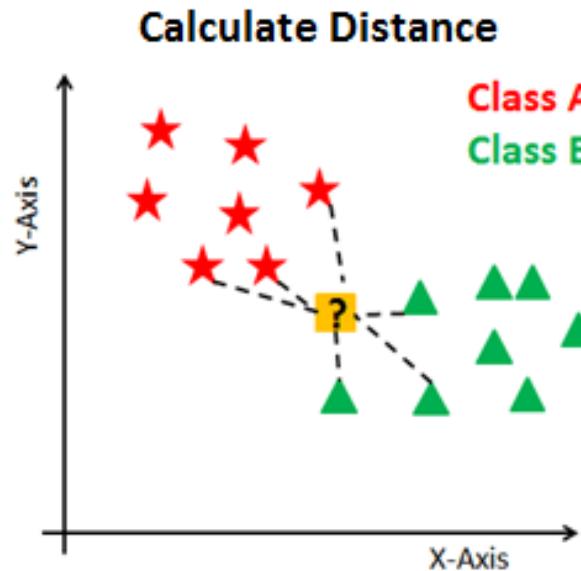
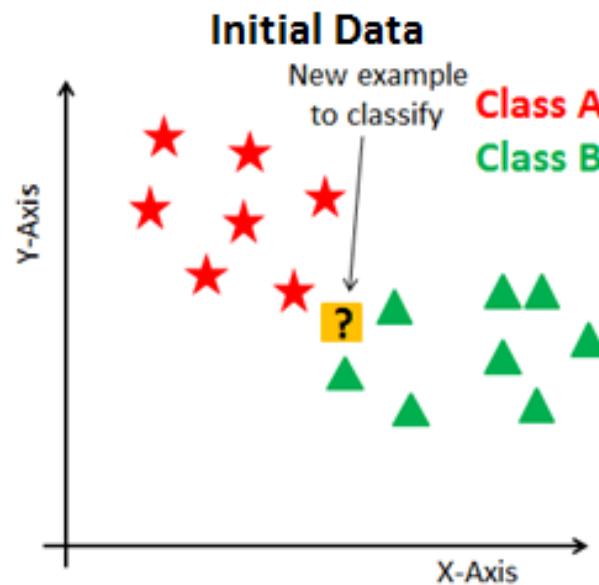
ทำนาย **label** ของภาพปริศนาตาม
ภาพใน **training data** ที่มีหน้าตา
ใกล้เคียงกับภาพปริศนามากที่สุด

นั่นคือในตัวอย่างนี้ เราจะหาภาพใน **training data** ที่หน้าตาคล้ายภาพ
ปริศนาที่สุด และถ้าภาพนั้นเป็น **CAT** ก็ตอบว่าภาพปริศนาน่าจะเป็น **CAT** ด้วย

K-NN is Instance-based Learning

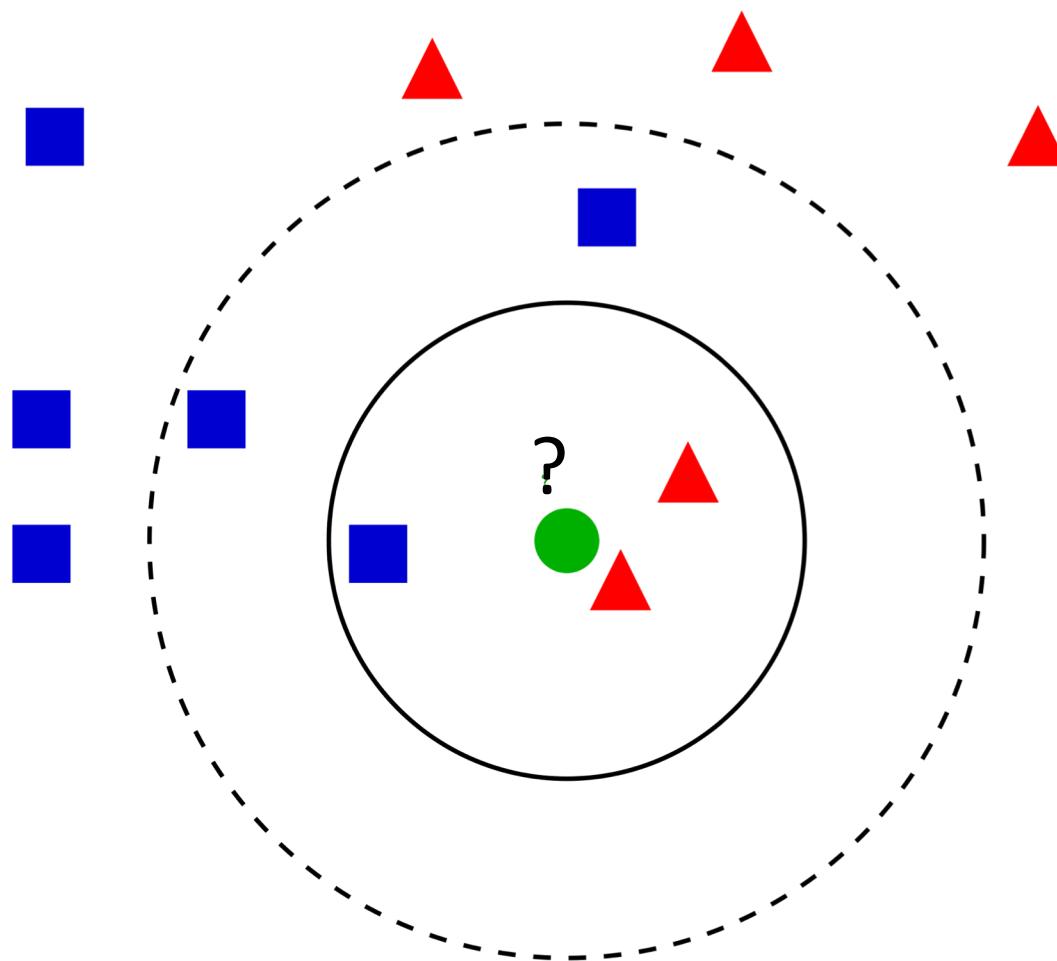
- Instance-based learning เป็นวิธีการเรียนรู้ที่หาคำตอบของข้อมูลใหม่โดยการ เปรียบเทียบกับทุกๆ instance ของข้อมูลชุดฝึก เช่น
 - เป้าหมาย: ต้องการหา label y_{test} ของข้อมูล x_{test}
 - วิธีการ:
 - นำ x_{test} ไปเปรียบเทียบกับข้อมูลในชุดฝึก x_{train} ทุกๆ ตัว
 - หา x_{train} ที่คล้ายกับข้อมูลนี้ x_{test} ที่สุด k อันดับ
 - แล้วตอบ y_{test} เลียนแบบตามข้อมูลที่คล้ายสุดนั้น
- Instance-based learning มีจุดเด่นที่
 - ไม่มี model (นั่นคือไม่ assume ว่าข้อมูลมีการเรียงตัวเป็นโครงสร้างตามสมการใดๆ)
 - ต้องจำข้อมูลชุดฝึกไว้ทั้งหมด เพื่อใช้ในการทำนาย (ถือเป็นข้อเสียใหญ่ เพราะเปลืองหน่วยความจำ)
 - มีการคำนวณ **distance metric** => ระยะห่างระหว่าง instance

How k-NN works?



<https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn>

k-NN example (3-NN vs 5-NN)



Distance Metric

- ในการหาเพื่อนบ้านที่ใกล้ที่สุด เราจะใช้อะไรเป็นตัววัด ระยะห่างระหว่างภาพ / ความคล้าย ?

Distance Metric

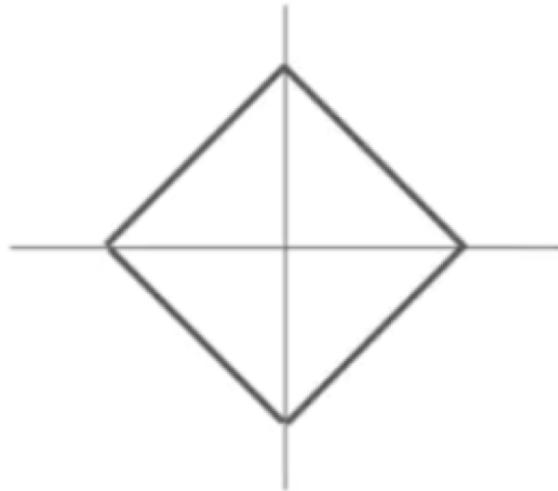
- ในการหาเพื่อนบ้านที่ใกล้ที่สุด เราจะใช้อะไรเป็นตัววัด ระยะห่างระหว่างภาพ / ความคล้าย ?
- อาจใช้ L1 distance: $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$ (Manhattan)
- หรือ L2 distance: $d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$ (Euclidean)

test image				training image				pixel-wise absolute value differences				
56	32	10	18	10	20	24	17	46	12	14	1	
90	23	128	133	8	10	89	100	82	13	39	33	→ 456
24	26	178	200	12	16	178	170	12	10	0	30	
2	0	255	220	4	32	233	112	2	32	22	108	

Distance Metric: L1 vs L2

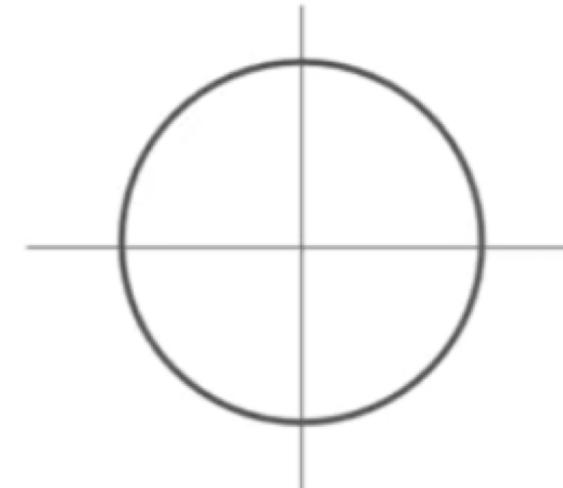
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

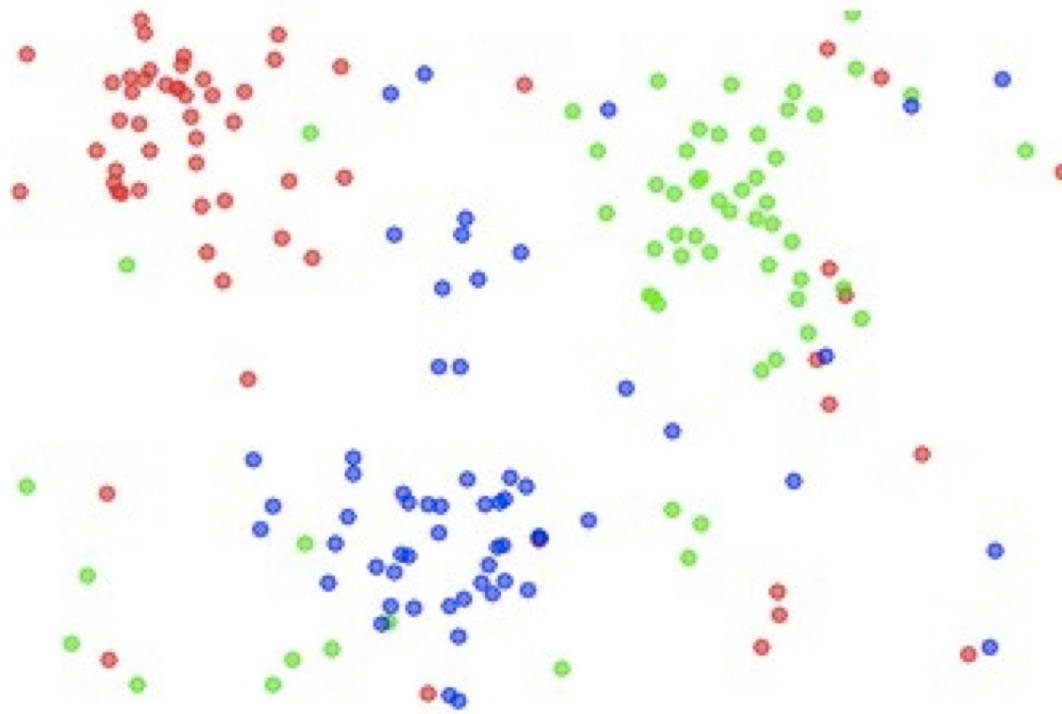


L2 (Euclidean) distance

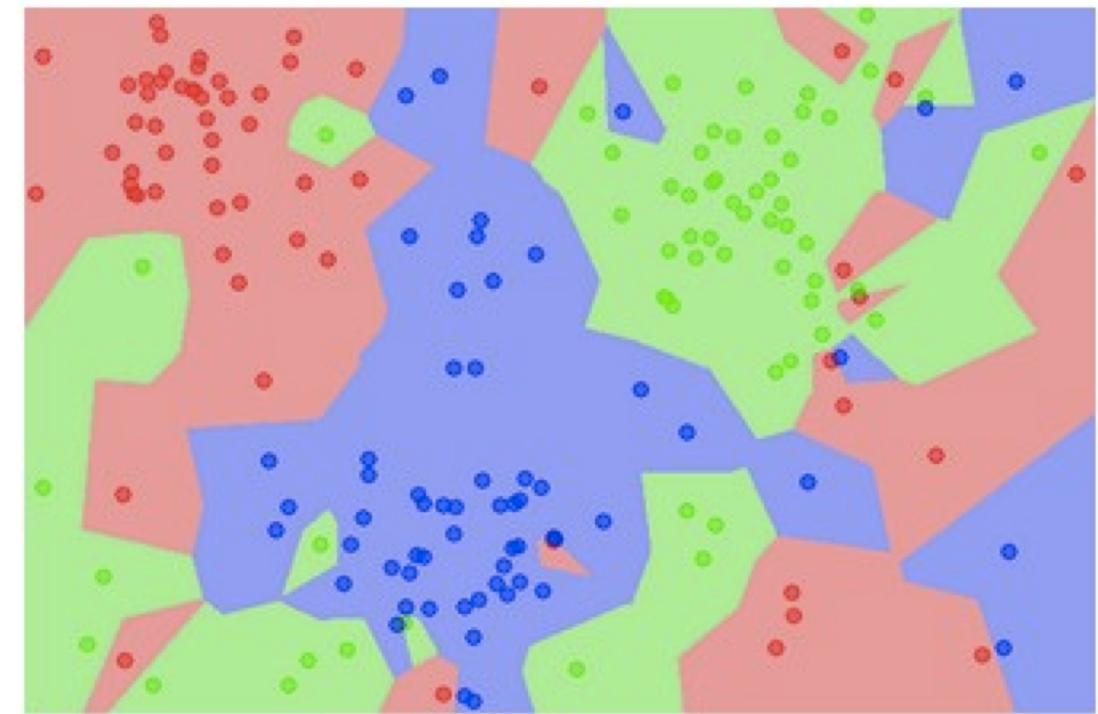
$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



Visualize เส้นแบ่งแทนของ Nearest Neighbor กับข้อมูลขนาด 2 มิติ

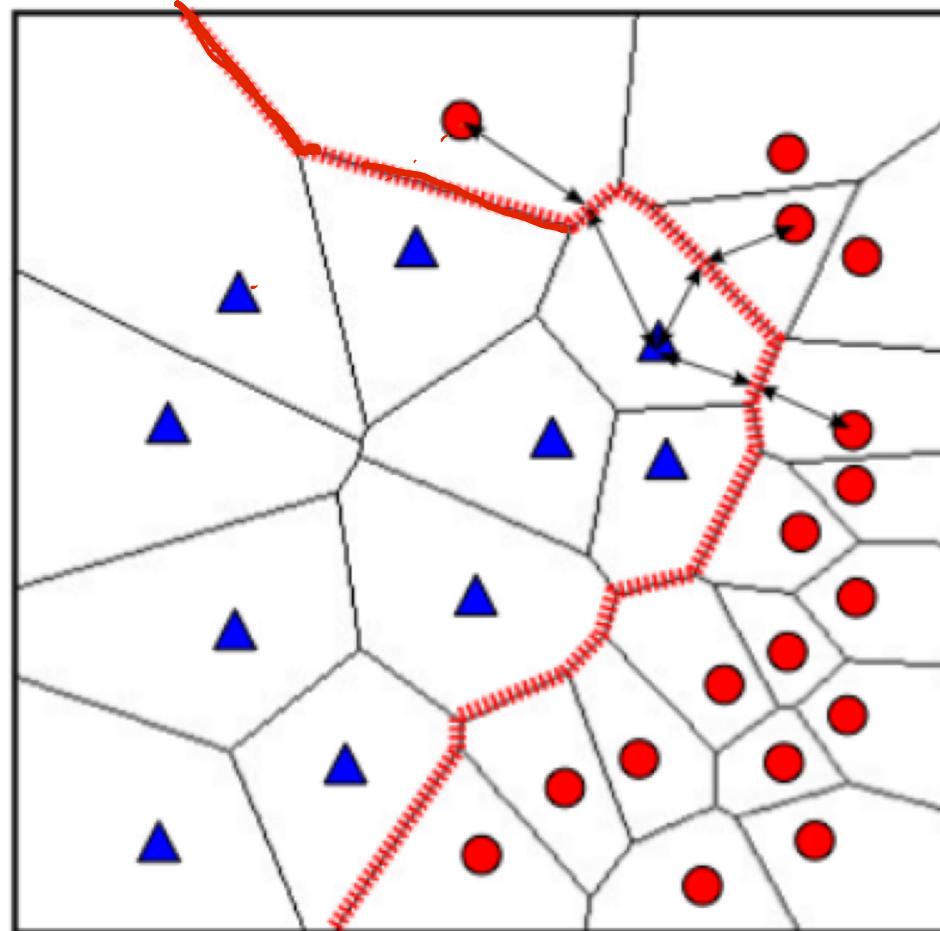


ข้อมูล training



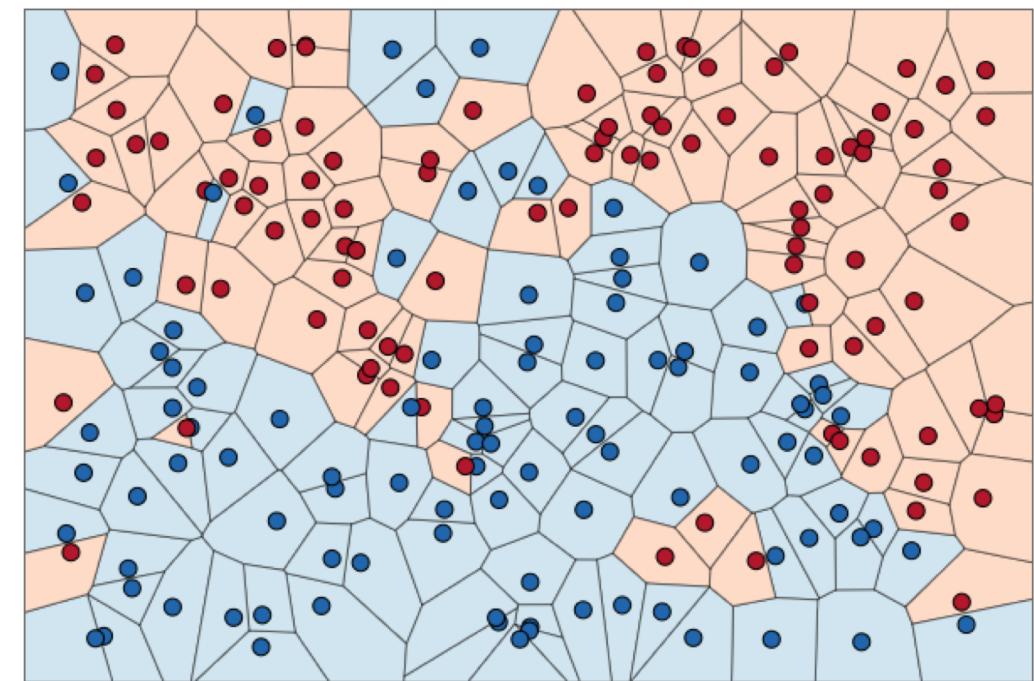
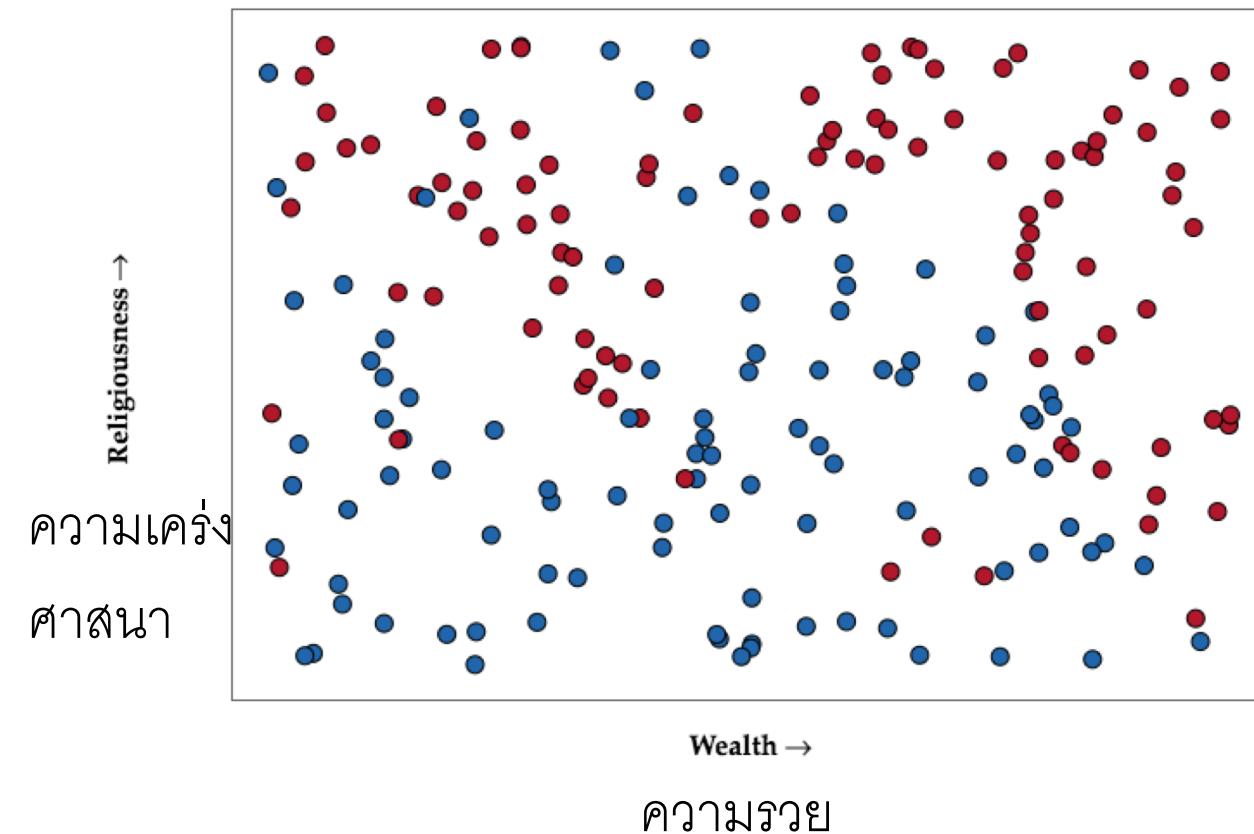
การแบ่งแทนข้อมูลโดย NN classifier

Visualize เส้นแบ่งแดนของ Nearest Neighbor กับข้อมูลขนาด 2 มิติ (Voronoi Diagram)



Visualize เส้นแบ่งแดนของ Nearest Neighbor กับข้อมูลขนาด 2 มิติ (Voronoi Diagram)

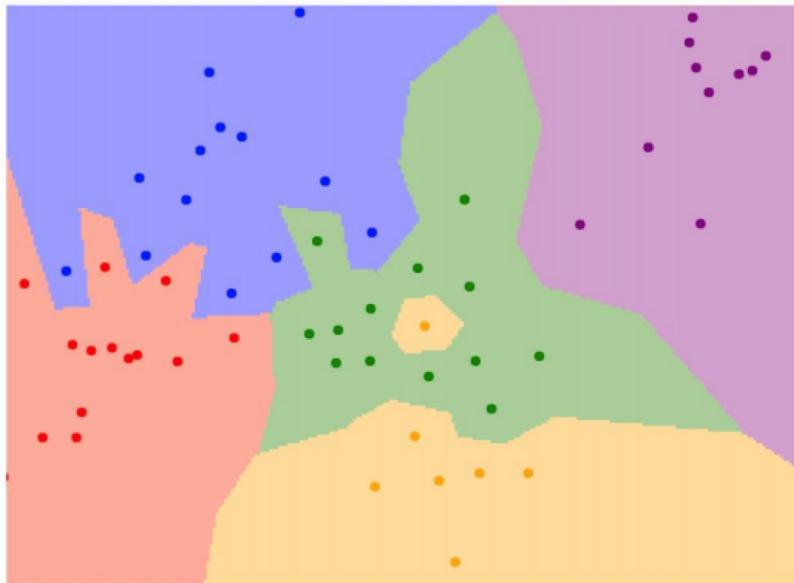
ข้อมูลสมมุติ ผู้ให้คะแนน **Republican** vs **Democrat**



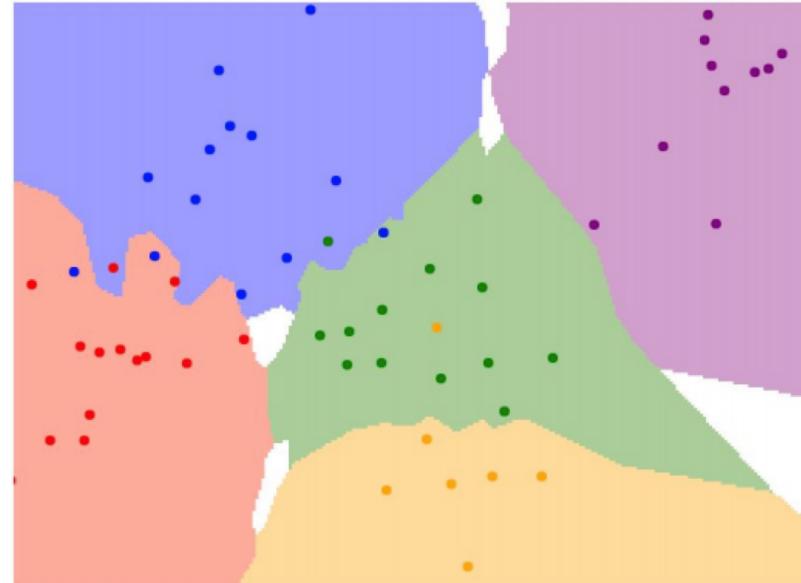
แผนภาพ Voronoi แสดงเขตแดนของ Nearest Neighbor

K-Nearest Neighbors

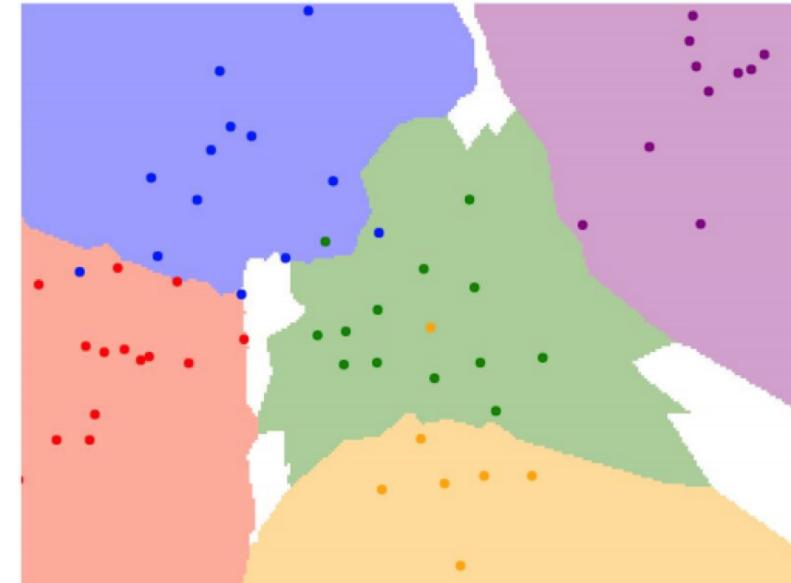
- ใช้วิธีหัวใจจากเพื่อนบ้านที่ใกล้สุด K จุด แทนการตอบตามเพื่อนบ้านที่ใกล้ที่สุดเพียงจุดเดียว



$K = 1$



$K = 3$



$K = 5$

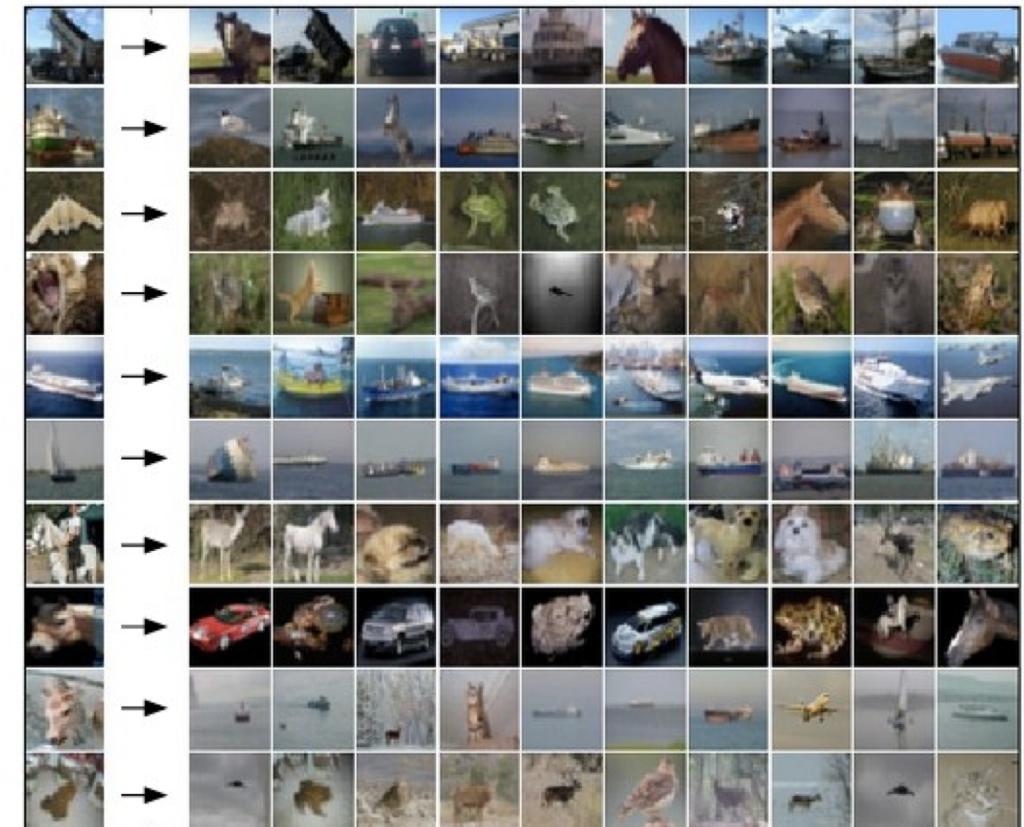
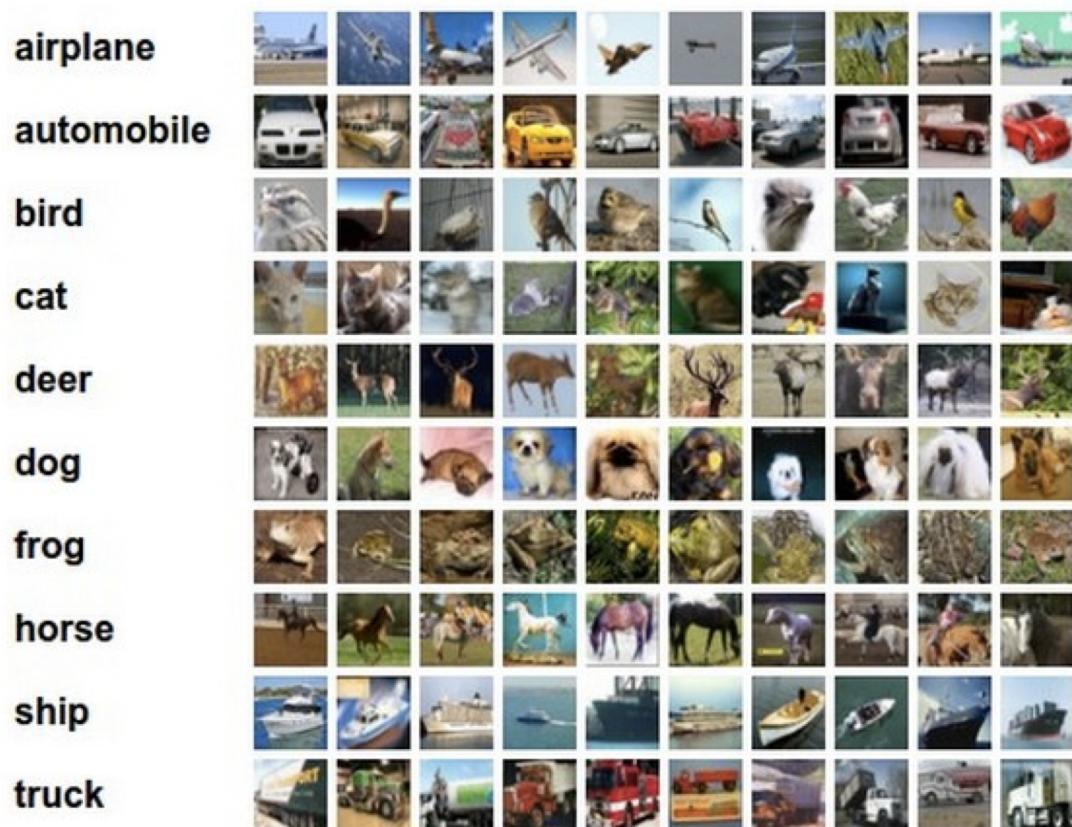
Hyperparameters

- ผู้ใช้อัลกอริทึม **k-NN** สามารถเลือกใช้
 - ค่า **k** ที่เหมาะสม ($k=1,2,3,\dots$)
 - วิธีการวัดระยะห่างที่เหมาะสม (**L1 vs L2**)
- เราเรียก ตัวแปรเกี่ยวกับอัลกอริทึม **Machine Learning** ที่ผู้ใช้สามารถปรับตั้งค่าได้ นี้ว่า “**Hyperparameter**”
- จะเลือก **Hyperparameter** ที่ดีได้อย่างไร?
 - ต่อไปนี้คือหน้าเรื่อง **Model Selection**

ระวัง ! **Hyperparameter** คือคนละอย่างกับ **Parameter**
(เช่น Parameter ของ Linear Regression Model ซึ่งเรา
จะเรียนในคลาสฯ ไป)

Example: k-NN on CIFAR-10 Dataset

- 10 classes | 50,000 training images | 10,000 testing images



Test Images --> nearest neighbors

Implementing k-NN

<https://qianjun.github.io/2019f-UVA-CS6316-MachineLearning//Lectures/L09-KNN.pdf>

	\vec{x}_1	\vec{x}_2	\dots	\vec{x}_P	y
\vec{x}_1					
\vec{x}_2					
x					
$\vec{x}_{n_{tr}}$					

→ Step 1: $O(n_{tr}P)$

$$\left\{ \begin{array}{l} d(\vec{x}_?, \vec{x}_1) \\ d(\vec{x}_?, \vec{x}_2) \\ \vdots \\ d(\vec{x}_?, \vec{x}_{n_{tr}}) \end{array} \right.$$

$\vec{x}_?$ []

→ Step 2:
pick top k from n_{tr}

$$O(n_{tr})$$

① $d(\vec{x}_i, \vec{x}_j)$

② k

→ Step 3
 $y_? = \frac{1}{k} \sum_{j \in NN(x_?)} y_j$

ตัวอย่างโค้ด Nearest Neighbor ใน Python

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

ตัวอย่างโค้ด Nearest Neighbor ใน Python

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

จดจำ **training data** ไว้

X = รูปภาพ

y = label

ตัวอย่างโค้ด Nearest Neighbor ใน Python

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

For แต่ละรูป **test_image**:

หาภาพที่ใกล้เคียงที่สุด

ทำนายเป็น **label** ของภาพที่ใกล้สุด

ตัวอย่างโค้ด Nearest Neighbor ใน Python

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

Q: หากมีข้อมูลชุดฝึก N ภาพ
อัลกอริทึมดังกล่าวจะใช้เวลาในการ **train**
และ **predict** เท่าใด ?

ตัวอย่างโค้ด Nearest Neighbor ใน Python

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

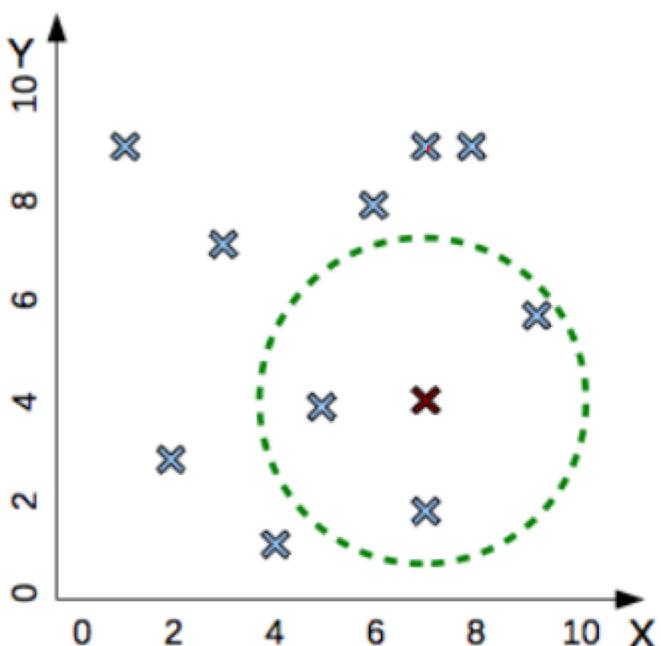
Q: หากมีข้อมูลชุดฝึก N ภาพ
อัลกอริทึมดังกล่าวจะใช้เวลาในการ **train**
และ **predict** เท่าใด ?

A: Train: $O(1)$
Predict: $O(N)$

ไม่ดี เพรา...

Why is k-NN slow?

What you see



What algorithm sees

- Training set:
 $\{(1,9), (2,3), (4,1), (3,7), (5,4), (6,8), (7,2), (8,8), (7,9), (9,6)\}$
- Testing instance:
 $(7,4)$
- Nearest neighbors?
compare one-by-one
to each training instance
- n comparisons
- each takes d operations

More on kNN...

- Refer to Stanford's CS231n slides
 - http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture02.pdf
- Visualize kNN classifier's boundary
 - http://wittawat.com/posts/knn_boundary.html
 - <http://vision.stanford.edu/teaching/cs231n-demos/knn/>

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

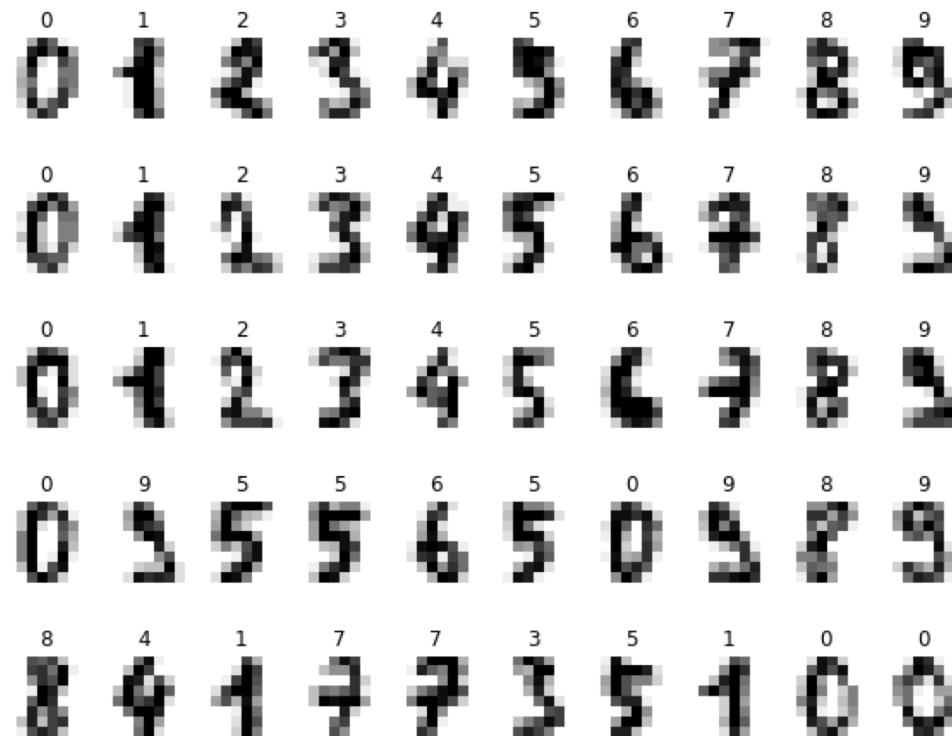
ข้อเสีย

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

Exercise:
Handwritten Digit Recognition with k-NN
(Scikit-learn)

Another famous dataset: MNIST

- Handwritten digits ("Hello World" of Machine Learning)
- 10 classes: 0, 1, 2, ..., 9



Handwritten Digit Recognition with k-NN

1. นำเข้าข้อมูล MNIST

```
from sklearn import datasets  
digits = datasets.load_digits()
```

2. กด SHIFT-Tab ตรง load_digits() เพื่ออ่าน docstring

```
digits = datasets.load_digits()  
Signature: datasets.load_digits(n_class=10, return_X_y=False)  
Docstring:  
Load and return the digits dataset (classification).  
  
Each datapoint is a 8x8 image of a digit.  
  
===== =====  
Classes 10  
Samples per class ~180  
Samples total 1797  
Dimensionality 64
```

Handwritten Digit Recognition with k-NN

3. สำรวจลักษณะของข้อมูลใน digits dataset

```
digits
```

```
digits.images.shape
```

```
digits.images[0].shape
```

```
digits.images[0]
```

```
digits.target
```

```
digits.target.shape
```

Handwritten Digit Recognition with k-NN

4. ทดลองแสดงภาพใน `images` ด้วย `matplotlib.pyplot`

```
%matplotlib inline
import matplotlib.pyplot as plt

plt.subplot(1,2,1)
plt.imshow(digits.images[66], cmap='gray_r')
plt.subplot(1,2,2)
plt.imshow(digits.images[121], cmap='gray_r')
```

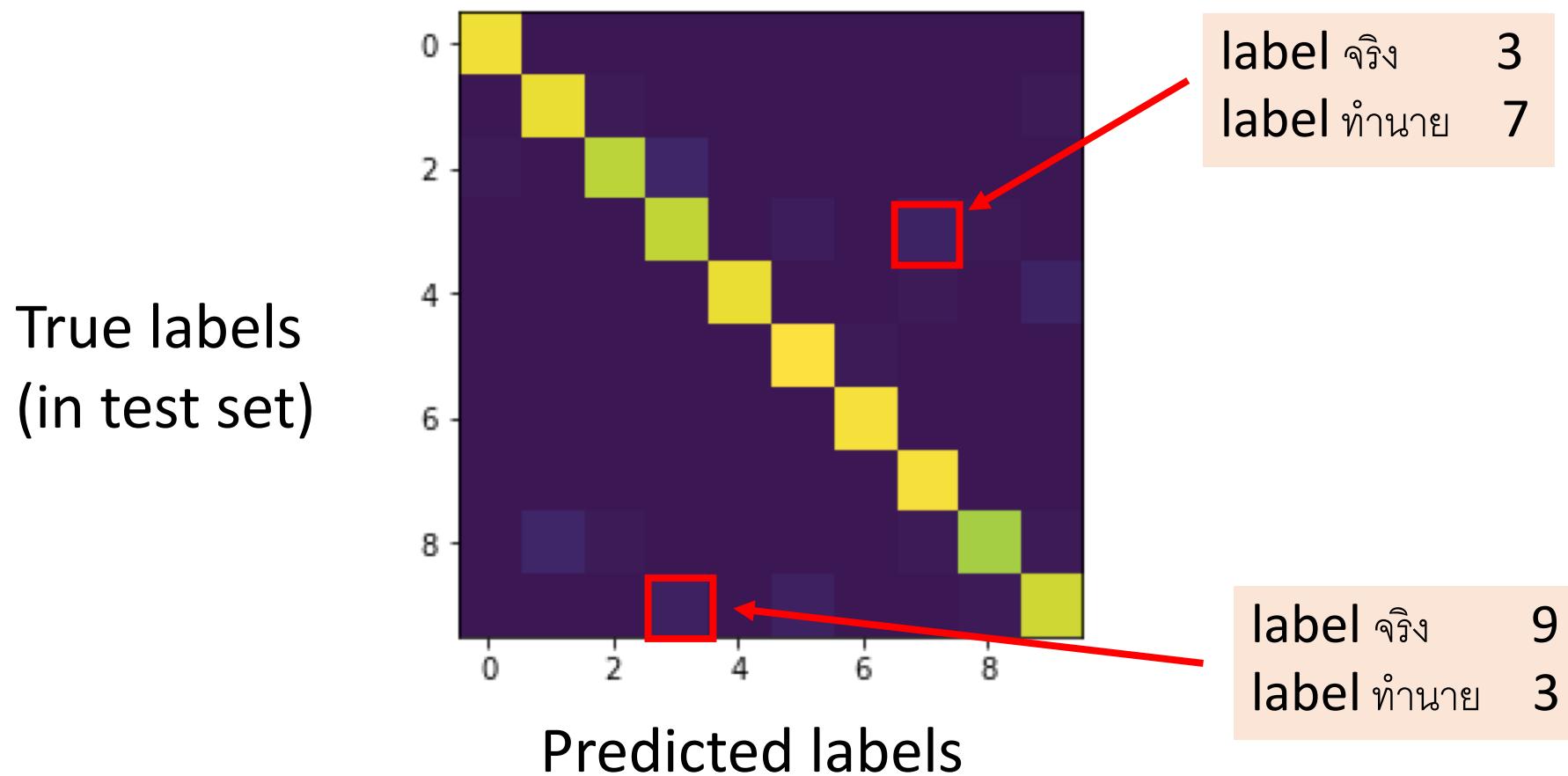
Handwritten Digit Recognition with k-NN

5. ลองใช้ `sklearn.neighbors.KNeighborsClassifier` ฝึกให้คอมจำแนกภาพตัวเลข

- แบ่งข้อมูลออกเป็นชุด Train (1000 ภาพแรก) และชุด Test (797 ภาพที่เหลือ)
- ประกาศ `classifier` ชนิด `KNeighborsClassifier` โดยให้ $k = 5$
- ทำการ `train classifier` ด้วยคำสั่ง `classifier.fit(ข้อมูลชุด train)`
- ทำการ `test classifier` ด้วยคำสั่ง `classifier.predict(ข้อมูลชุด test)`
- เปรียบเทียบผลการ `predict` กับเฉลย แล้วประเมินประสิทธิภาพ
 - $\text{accuracy} = \frac{\text{จำนวนชุด test ที่ถูกต้อง}}{\text{จำนวนชุด test}} / \text{จำนวนชุด test}$
 - ใช้ `metrics.confusion_matrix(เฉลย, ทาย)` เพื่อหาว่าทำนายผิดอย่างไร

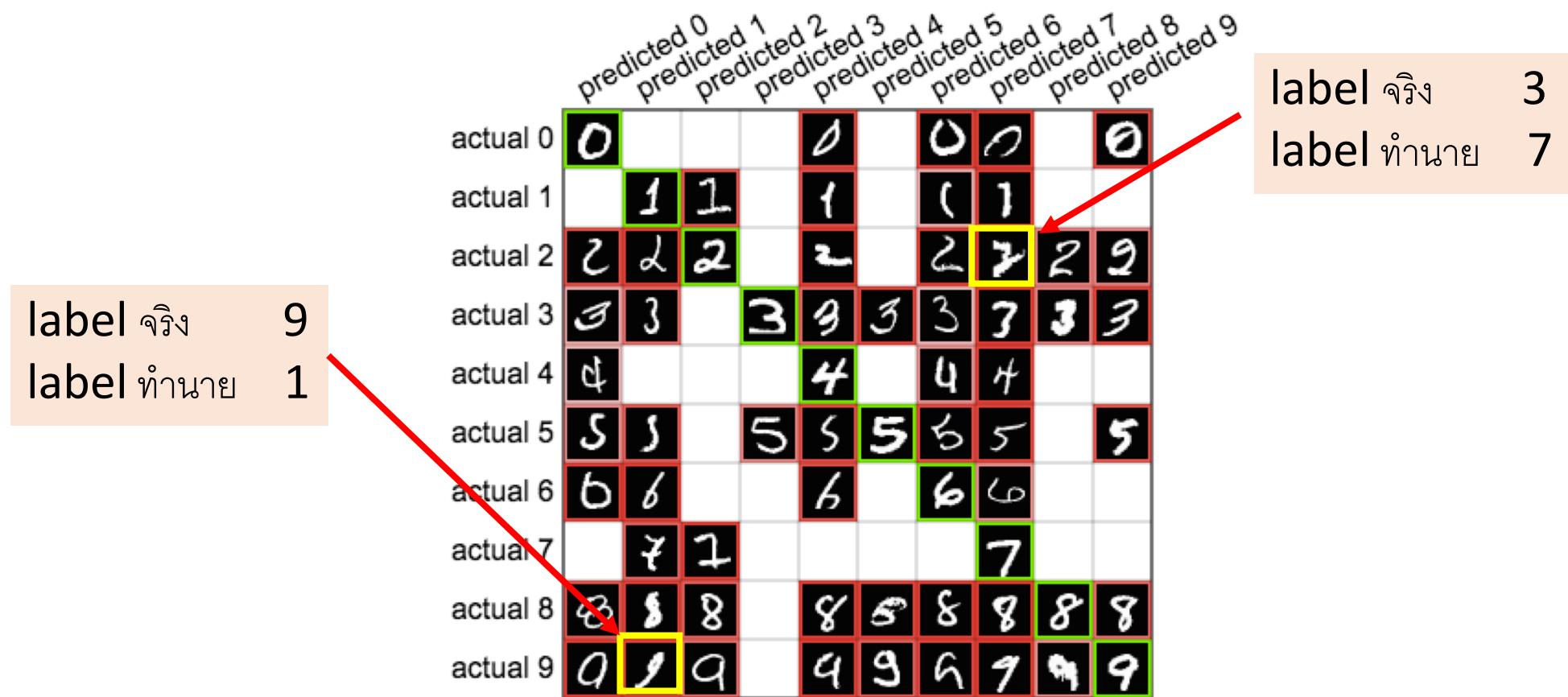
Confusion Matrix

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



Confusion Matrix

- ตัวอย่างภาพจาก MNIST dataset



Next week

- Lab: Implementing k-NN from scratch