Task1 Kuzushiji Kanji Classification

ข้อมูลเป็นรูปภาพตัวหนังสือคันจิที่เขียนด้วยมือ ข้อมูลบันทึกในรูปแบบ NumPy's uint8 (unsigned integers of 8-bit) โดยแต่ละรูปภาพจะมีขนาดกว้าง 64 พิกเซลและสูง 64 พิกเซลรวมทั้งหมด 4,096 พิกเซล

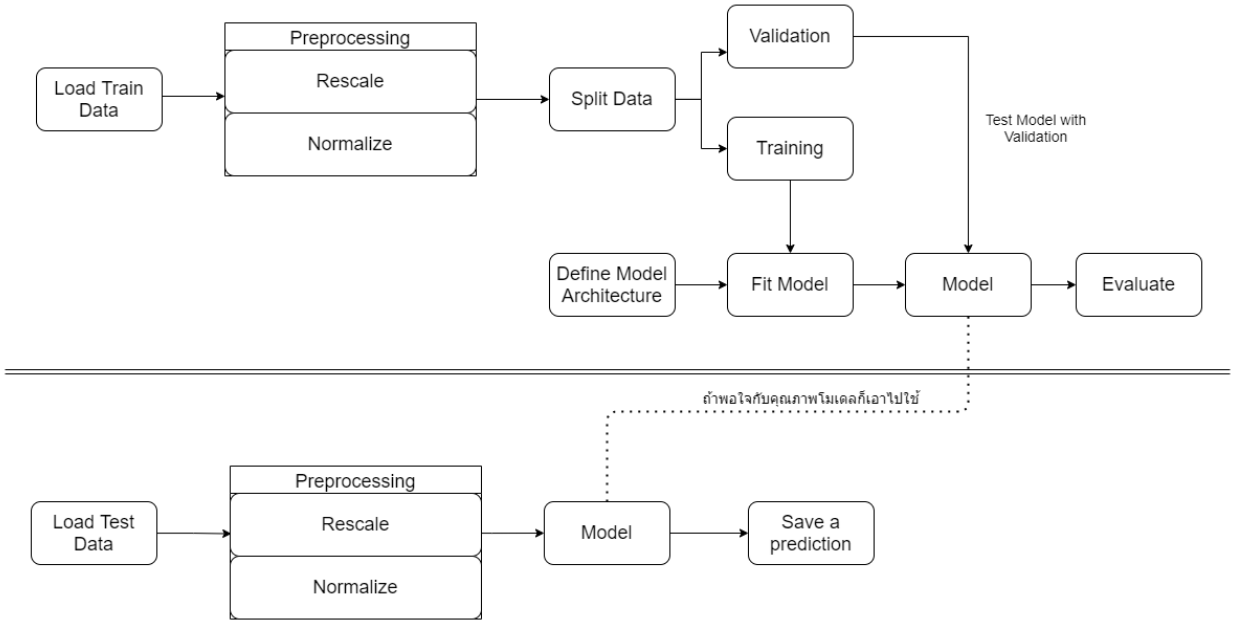
ข้อมูลมีจำนวนทั้งหมด 101,376 รูป และมีจำนวนตัวอักษรทั้งหมด 1,080 ตัว ซึ่งข้อมูลนั้นมีความไม่สมดุลอย่างมาก ดังนั้นวิธีการประเมิณจะไม่สามารถใช้ accuracy โดยตรงได้ เพราะถึงแม้จะมีค่า accuracy แต่ก็ไม่ได้หมายความว่าโมเดลมีคุณภาพ เพราะอาจจะทำนายถูกเพียงแค่คลาสส่วนใหญ่ (majority class) แต่คลาสอื่นไม่ถูกเลย ดังนั้นจึงจะใช้วิธีการประเมิณด้วย f1-score แทน

F1-score คือค่าเฉลี่ยแบบ harmonic ของ precision และ recall ซึ่ง f1-score ถูกสร้างขึ้นมาเพื่อเป็น single metric ที่วัดความสามารถของโมเดล ทำให้ไม่ต้องเลือกใช้ระหว่าง precision หรือ recall เพราะทำการเฉลี่ยให้เรียบร้อยแล้ว โดย f1-score จะมีสมการดังนี้

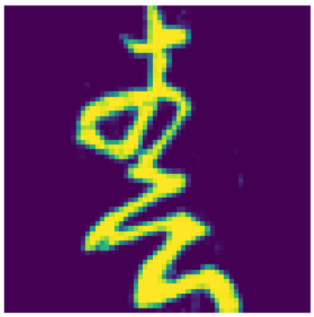
$$F1 = 2\frac{p \cdot r}{p + r} \quad \text{where} \quad p = \frac{tp}{tp + fp}, \quad r = \frac{tp}{tp + fn}$$

ขั้นตอนการทำงาน

1. CNN



1.1 นำเข้าชุดข้อมูล Train และลองทำการแสดงข้อมูลบางส่วน



1.2 ในการเตรียมข้อมูลจะมีการ reshape ข้อมูลเพื่อให้ง่ายต่อการเตรียมข้อมูลแล้วจึงทำการปรับ scale ใหม่เพื่อให้คอมพิวเตอร์ทำงานได้รวดเร็วขึ้น และหลังจากนั้นจะทำการ reshape ข้อมูลให้อยู่ในรูปแบบที่เหมาะสมสำหรับอัลกอริทึม

```
(101376, 64, 64, 1)
```

1.3 แบ่งข้อมูลออกเป็น 2 ส่วนได้แก่ Training และ Validation โดยแบ่งเป็นขนาด 90% และ 10% จากชุดข้อมูลหลักตามลำดับ

1.4 สร้างโมเดล

- activation จะเลือกใช้เป็น relu เพราะเป็นที่นิยมและมีคุณภาพ
- padding กำหนดให้เป็น same ซึ่งหมายถึงเราจะไม่เปลี่ยนขนาดของภาพ
- MaxPooling2D ลดขนาดอินพุต
- ใส่ dropout เพื่อไม่ให้โมเดล overfitting
- Batch Normalization จะทำให้อินพุตที่มุ่งหน้าไปยังเลเยอร์ถัดไปเป็นปกติ เพื่อให้แน่ใจว่าเครือข่ายจะมีการกระจายเดียวกันกับที่เราต้องการเสมอ
- Flatten เลเยอร์สุดท้ายของ CNN ต้องการให้ข้อมูลอยู่ในรูปของเวกเตอร์เพื่อประมวลผล ด้วยเหตุนี้ข้อมูลจึงต้องใช้ flatten เพื่อที่ค่าจะได้ถูกบีบอัดเป็นเวกเตอร์
- Dense อันสุดท้ายจะกำหนดให้มีจำนวนเท่ากับ Label ของเรา

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 256)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 4, 4, 256)         1024
```

batch_normalization_3 (Batch	(None, 4, 4, 256)	1024
conv2d_4 (Conv2D)	(None, 4, 4, 512)	1180160
max_pooling2d_4 (MaxPooling2	(None, 2, 2, 512)	0
dropout_4 (Dropout)	(None, 2, 2, 512)	0
batch_normalization_4 (Batch	(None, 2, 2, 512)	2048
flatten (Flatten)	(None, 2048)	0
dropout_5 (Dropout)	(None, 2048)	0
dense (Dense)	(None, 512)	1049088
dropout_6 (Dropout)	(None, 512)	0
batch_normalization_5 (Batch	(None, 512)	2048
flatten_1 (Flatten)	(None, 512)	0
dropout_7 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_8 (Dropout)	(None, 256)	0
batch_normalization_6 (Batch	(None, 256)	1024
dense_2 (Dense)	(None, 1080)	277560

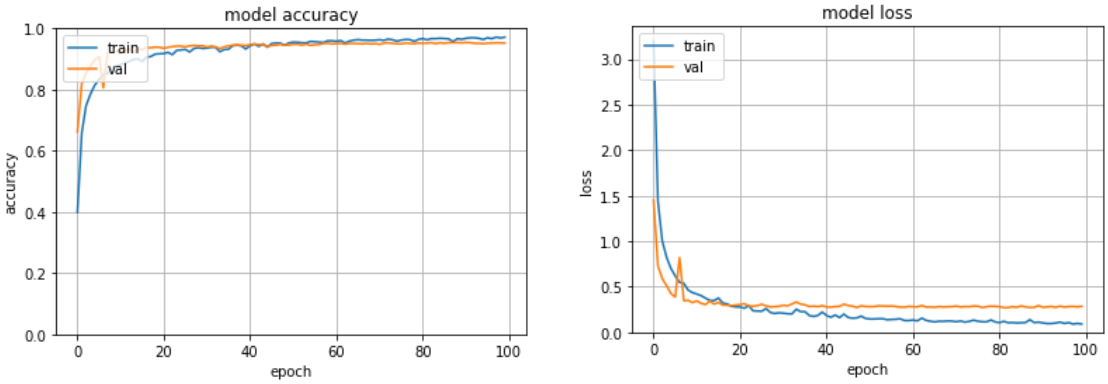
Total params: 3,033,016
Trainable params: 3,029,496
Non-trainable params: 3,520

1.5 นำชุดข้อมูล Training มา train โมเดล

- ผลลัพธ์เมื่อ epoch ที่ 100

```
Epoch 100/100
82112/82114 [============================>.] - ETA: 0s - loss: 0.0900 - acc: 0.9714
Epoch 00100: val_loss did not improve from 0.27010
82114/82114 [==============================] - 46s 560us/sample - loss: 0.0900 - acc: 0.9714 - val_loss: 0.2845 - val_acc: 0.95
24
```

- นำค่า accuracy และ loss มา plot เพื่อดูการเปลี่ยนไปในแต่ละ epoch



1.6 เมื่อได้โมเดลที่ได้จากการ train แล้วจึงนำโมเดลมาทดสอบกับชุดข้อมูล Validation

1.7 นำผลลัพธ์ที่ได้มาวิเคราะห์ ถ้าพึ่งพอใจก็จะนำโมเดลไปใช้กับชุดข้อมูล test ซึ่งในการประเมินผลจะใช้ค่า f1-score จะใช้ average แบบ micro เพราะเป็นการคำนวณเมตริกทั้งหมดโดยการนับผลบวกจริงทั้งหมด (total true positives) ผลลบเท็จ (false negatives) และผลบวกเท็จ (false positives)

```
accuracy: 0.95867
precision: 0.95867
recall: 0.95867
f1_score: 0.95867
```

	precision	recall	f1-score	support
0	0.90909	1.00000	0.95238	30
1	1.00000	1.00000	1.00000	2
2	0.90000	0.90000	0.90000	10
...				
1077	0.00000	0.00000	0.00000	2
1078	1.00000	1.00000	1.00000	2
1079	1.00000	1.00000	1.00000	2
accuracy			0.95867	10138
macro avg	0.93551	0.91715	0.91752	10138
weighted avg	0.96185	0.95867	0.95711	10138

1.8 นำเข้าชุดข้อมูล Test

1.9 ทำการเตรียมข้อมูลให้เหมือนกับที่ทำกับชุดข้อมูล Train

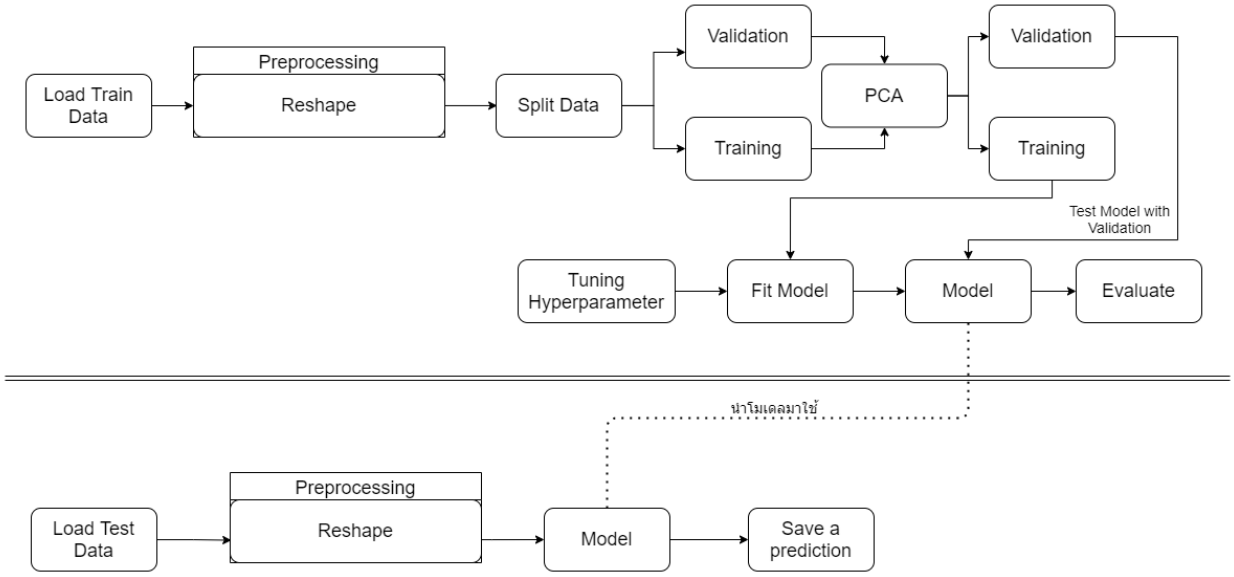
1.10 นำชุดข้อมูล Test ที่ได้หลังจากการเตรียมไปใช้กับโมเดล

1.11 นำผลลัพธ์ที่ได้บันทึกลงไฟล์ csv

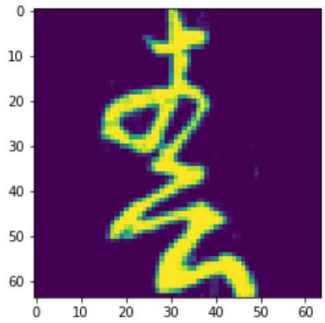
- ตัวอย่างผลการทำนาย 10 รูปแรก

	ImageId	ClassId
0	1	412
1	2	20
2	3	241
3	4	185
4	5	557
5	6	984
6	7	915
7	8	79
8	9	332
9	10	28

2. KNN



2.1 นำเข้าชุดข้อมูล Train และลองทำการแสดงข้อมูลบางส่วน



2.2 ในการเตรียมข้อมูลจะมีการ reshape ข้อมูลเพื่อให้ง่ายต่อการเตรียมข้อมูลและอยู่ในรูปแบบที่เหมาะสมสำหรับอัลกอริทึม

2.3 แบ่งข้อมูลออกเป็น 2 ส่วนได้แก่ Training และ Validation โดยแบ่งเป็นขนาด 80% และ 20% จากชุดข้อมูลหลักตามลำดับ

2.4 นำข้อมูลที่ได้หลังจากการแบ่งมาทำ PCA เพื่อลดขนาดของข้อมูลและหาแกนหลักสำคัญ

2.5 ใช้ Grid search ในการจูนโมเดลเพื่อหา hyperparameter ที่ดีที่สุด โดย hyperparameter ที่จะค้นหามี 2 ตัวได้แก่

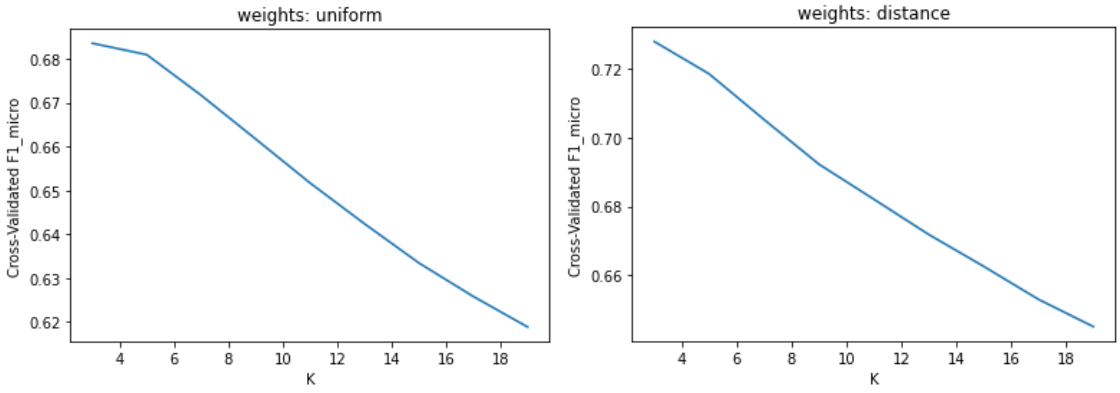
- จำนวน n โดยจะหาตั้งแต่ 3, 5, 7, 9, 11, 13, 15, 17, 19 ที่ค้นหาเฉพาะเลขคี่เพราะว่า n ที่เป็นเลขคู่ไม่สามารถให้ค่า majority vote ได้ทุกครั้ง

- Weights โดยจะหา uniform และ distance

ซึ่งจะให้ความสำคัญกับ hyperparameter ที่ให้ค่า f1-score ที่สูงกว่า

	mean_test_score	std_test_score	params
0	0.683514	0.001629	{'n_neighbors': 3, 'weights': 'uniform'}
1	0.728002	0.002024	{'n_neighbors': 3, 'weights': 'distance'}
2	0.680912	0.002700	{'n_neighbors': 5, 'weights': 'uniform'}
3	0.718582	0.002378	{'n_neighbors': 5, 'weights': 'distance'}
4	0.671652	0.004364	{'n_neighbors': 7, 'weights': 'uniform'}
5	0.705302	0.003635	{'n_neighbors': 7, 'weights': 'distance'}
6	0.661702	0.003730	{'n_neighbors': 9, 'weights': 'uniform'}
7	0.692306	0.003240	{'n_neighbors': 9, 'weights': 'distance'}
8	0.651677	0.004363	{'n_neighbors': 11, 'weights': 'uniform'}
9	0.682121	0.003355	{'n_neighbors': 11, 'weights': 'distance'}
10	0.642380	0.003698	{'n_neighbors': 13, 'weights': 'uniform'}
11	0.671862	0.003584	{'n_neighbors': 13, 'weights': 'distance'}
12	0.633453	0.004389	{'n_neighbors': 15, 'weights': 'uniform'}
13	0.662602	0.004437	{'n_neighbors': 15, 'weights': 'distance'}
14	0.625808	0.004351	{'n_neighbors': 17, 'weights': 'uniform'}
15	0.652984	0.004186	{'n_neighbors': 17, 'weights': 'distance'}
16	0.618841	0.004885	{'n_neighbors': 19, 'weights': 'uniform'}
17	0.645043	0.004547	{'n_neighbors': 19, 'weights': 'distance'}

2.6 ทำการ plot ค่า f1-score ที่ได้จากการเปลี่ยน n ไปในแต่ละครั้ง โดยจะแบ่งเป็นทั้งหมด 2 รูปได้ weights ที่เป็น uniform และ distance



จากตารางข้างต้นจะเห็นว่า hyperparameter ที่ได้คะแนนเยอะที่สุดคือ

```
{'n_neighbors': 3, 'weights': 'distance'}
```

2.7 นำ hyperparameter ที่ได้จากการจูนไปสร้างโมเดล

2.8 นำโมเดลมาทดสอบกับชุดข้อมูล Validation

2.9 นำผลลัพธ์ที่ได้มาวิเคราะห์ ซึ่งในการประเมินผลจะใช้ค่า f1-score

```
accuracy: 0.74162
precision: 0.74162
recall: 0.74162
f1_score: 0.74162
```

	precision	recall	f1-score	support
0	0.69841	0.84615	0.76522	52
1	0.00000	0.00000	0.00000	3
2	0.80000	0.70588	0.75000	17
		...		
1077	0.00000	0.00000	0.00000	4
1078	0.50000	0.20000	0.28571	5
1079	1.00000	0.66667	0.80000	3
accuracy			0.74162	20276
macro avg	0.68379	0.55557	0.58441	20276
weighted avg	0.75681	0.74162	0.73140	20276

2.10 นำเข้าชุดข้อมูล Test

2.11 ทำการเตรียมข้อมูลให้เหมือนกับที่ทำกับชุดข้อมูล Train

2.12 นำชุดข้อมูล Test ที่ได้หลังจากการเตรียมไปใช้กับโมเดล

2.13 นำผลลัพธ์ที่ได้บันทึกลงไฟล์ csv

- ตัวอย่างผลการทำนาย 5 รูปแรก

	ImageId	ClassId
0	1	972
1	2	20
2	3	241
3	4	52
4	5	350

สรุปผลการทดลอง

จากการทดลองจะเห็นว่ากระบวนการทำงานของทั้ง 2 วิธีจะไม่ต่างกันมากแต่ผลลัพธ์ที่ได้นั้นแตกต่างกันอย่างชัดเจน ซึ่งวิธี CNN จะให้ผลลัพธ์ที่ดีกว่าวิธี KNN โดยวิธี CNN นั้นผลลัพธ์ที่ได้ก็จะแตกต่างกันออกไปตามวิธีการสร้างโมเดลและวิธีนี้มีโอกาสเกิด overfitting ที่สูง ดังนั้นจำเป็นจะต้องออกแบบโมเดลให้ดี มีคุณภาพและต้องหาวิธีที่จะลดการเกิด overfitting เช่นการใช้ dropout ส่วนวิธี KNN นั้นอาจจะยังไม่มีคุณภาพพอเพราะแม้กระทั่งใช้ PCA หรือใช้ Grid search เข้ามาช่วยก็ยังไม่สามารถให้ผลลัพธ์ที่ดีเทียบเท่า CNN ได้ ดังนั้นถ้าต้องการจะใช้ KNN อาจจะต้องเตรียมข้อมูลให้ดีมากยิ่งขึ้นและ จากการทดลองจะเห็นว่าเราอาจจะไม่จำเป็นจะต้องกำหนด n ให้มีจำนวนเยอะเพราะเมื่อ n เพิ่มขึ้น ค่า f1-sccore ที่ได้ก็ยิ่งลดลง

บรรณานุกรม

- Stackabuse.//(2564).// Image Recognition in Python with TensorFlow and Keras.//สืบค้นเมื่อ 30 เมษายน 2564,/จาก/https://stackabuse.com/image-recognition-in-python-with-tensorflow-and-keras/
- Matplotlib.//(2564).// Visualization with Python.//สืบค้นเมื่อ 29 เมษายน 2564,/จาก/ https://matplotlib.org/
- Numpy.//(2564).//Scientific computing with Python.//สืบค้นเมื่อ 29 เมษายน 2564,/จาก/ https:// https://numpy.org/
- Pandas.//(2564).//manipulation tool.//สืบค้นเมื่อ 29 เมษายน 2564,/จาก/ https://pandas.pydata.org/
- Tensorflow.//(2564).//Convolutional Neural Network (CNN).//สืบค้นเมื่อ 29 เมษายน 2564,/จาก/ https://www.tensorflow.org/tutorials/images/cnn
- Keras.//(2564).//Convolutional layers.//สืบค้นเมื่อ 29 เมษายน 2564,/จาก/ https://keras.io/api/layers/convolution_layers/
- Keras.//(2564).// Callbacks API.//สืบค้นเมื่อ 28 เมษายน 2564,/จาก/ https://keras.io/api/callbacks/
- Sklearn.//(2564).//Model Selection.//สืบค้นเมื่อ 28 เมษายน 2564,/จาก/ https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection
- Sklearn.//(2564).//Metrics and scoring: quantifying the quality of predictions.//สืบค้นเมื่อ 28 เมษายน 2564,/จาก/https://scikit-learn.org/stable/modules/model_evaluation.html
- Sklearn.//(2564).//Preprocessing data.//สืบค้นเมื่อ 28 เมษายน 2564,/จาก/ https://scikit-learn.org/stable/modules/preprocessing.html
- Sklearn.//(2564).//PCA.//สืบค้นเมื่อ 26 เมษายน 2564,/จาก/ https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
- Sklearn.//(2564).//KNeighborsClassifier.//สืบค้นเมื่อ 25 เมษายน 2564,/จาก/ https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

- Medium.//(2564).//k-Neighbors Classifier with GridSearchCV Basics.//สืบค้นเมื่อ 26 เมษายน 2564,/จาก/https://medium.com/@erikgreenj/k-neighbors-classifier-with-gridsearchcv-basics-3c445ddeb657
- Datarockie.//(2564).//รีวิวเทคนิค Normalization และ Standardization.//สืบค้นเมื่อ 30 เมษายน 2564,/จาก/ https://datarockie.com/2019/11/07/comparison-normalization-standardization/
- Datarockie.//(2564).//อธิบาย 10 Metrics พื้นฐานสำหรับวัดผลโมเดล Machine Learning.//สืบค้นเมื่อ 26 เมษายน 2564,/จาก/ https://datarockie.com/2019/03/30/top-ten-machine-learning-metrics/#:~:text=18)%20%3D%2052.6%25-,F1%2DScore,recall%20%E0%B9%80%E0%B8%9E%E0%B8%A3%E0%B8%B2%E0%B8%B0%E0%B9%80%E0%B8%89%E0%B8%A5%E0%B8%B5%E0%B9%88%E0%B8%A2%E0%B9%83%E0%B8%AB%E0%B9%89%E0%B9%81%E0%B8%A5%E0%B9%89%E0%B8%A7)