A large, stylized blue water splash graphic on the left side of the slide, containing the university's seal.

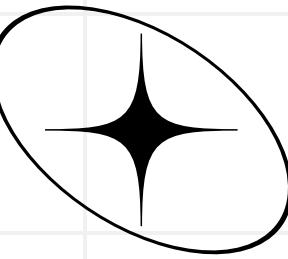
WATER



# WATER QUALITY WITH BINARY CLASSIFICATION

---

01418362 Introduction to Machine Learning



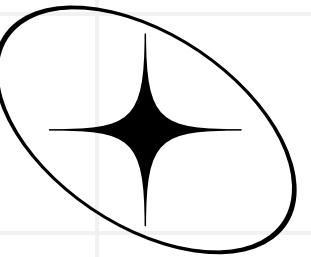
5/4/2567

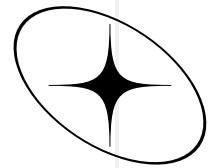


# MEMBER



นาย พิรสิษฐ์ โลயอร่าม  
6410451237



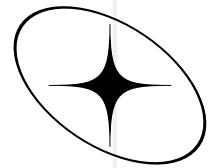


# WATER QUALITY DATASET

- เป็น DataSet กี่บอกข้อมูลตัวอย่างน้ำว่ามีความสะอาด หรือ มีความปล่อยกั้ยหรือไม่
- โดยมี Features กี่ใช้ในการจำแนกทั้งหมด 20 Feature และ Label เลย
- มีจำนวนข้อมูลตัวอย่างทั้งหมด 7999 ตัวอย่าง

aluminium	ammonia	arsenic	barium	cadmium	chloramine	chromium	copper	flouride	bacteria	viruses
1.65	9.08	0.04	2.85	0.007	0.35	0.83	0.17	0.05	0.2	0
2.32	21.16	0.01	3.31	0.002	5.28	0.68	0.66	0.9	0.65	0.65
1.01	14.02	0.04	0.58	0.008	4.24	0.53	0.02	0.99	0.05	0.003
1.36	11.33	0.04	2.96	0.001	7.23	0.03	1.66	1.08	0.71	0.71
0.92	24.33	0.03	0.2	0.006	2.67	0.69	0.57	0.61	0.13	0.001
lead	nitrates	nitrites	mercury	perchlorate	radium	selenium	silver	uranium	is_safe	
0	0	0	0	0	0	0	0	0	1	
0.034	4.06	0	0.006	0.01	0	0.02	0.02	0	1	
0.014	16.44	0.94	0.01	0.31	0	0.07	0.1	0.02	0	
0.151	13.03	0.82	0.009	0.53	0	0.1	0.04	0.03	0	
0.164	3.17	1.9	0	0.87	0	0.01	0.02	0.01	0	

แหล่งที่มาของ DataSet: <https://www.kaggle.com/datasets/mssmartypants/water-quality/data>



# Support Vector Machine: SVM

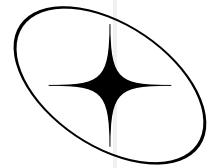
- SVM ต้องการที่จะหา Optimal Hyperplane ที่สามารถแบ่งชุดข้อมูลที่มี Class ต่างกัน ออกจากกันได้

$$y_i(x_i^T w + b) \geq 1 - \xi_i \quad \xi_i \geq 0$$

- โดย Hyperplane ที่ได้ต้องมีระยะห่าง (Margin) มากที่สุด

$$\begin{aligned} \min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) &= \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{N} \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) \\ &= \frac{1}{N} \sum_i^N \left( \frac{\lambda}{2} \|\mathbf{w}\|^2 + \underbrace{\max(0, 1 - y_i f(\mathbf{x}_i))}_{\text{loss function}} \right) \end{aligned}$$

- L2 Regularization



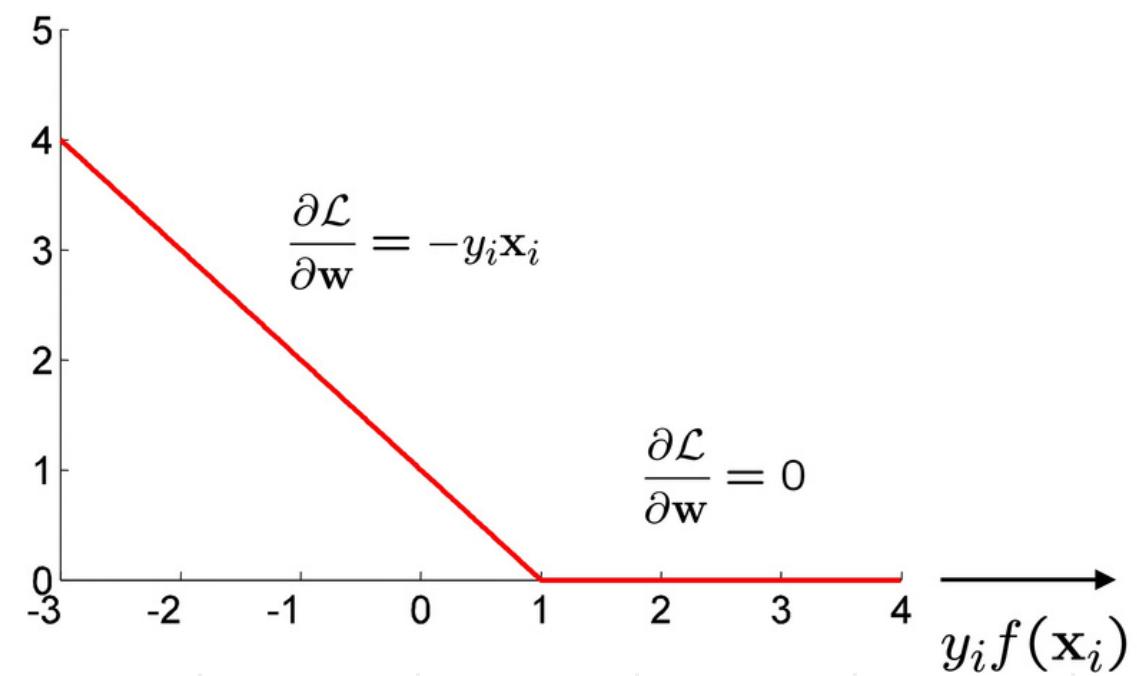
# Support Vector Machine: SVM

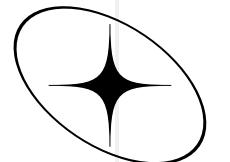
- SVM ใช้ “Hinge Loss”  $\max(0, 1 - y_i f(\mathbf{x}_i))$
- Gradient Descent:  $C(w)$  โดยในแต่ละครั้งที่ Update  $w$  เมื่อ  $y_i(\mathbf{w}^T \mathbf{x}_i + b) < 1$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta(\lambda \mathbf{w}_t - y_i \mathbf{x}_i) \quad \text{if } y_i f(\mathbf{x}_i) < 1$$

เมื่อ  $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

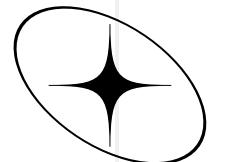
$$\leftarrow \mathbf{w}_t - \eta \lambda \mathbf{w}_t$$





# Support Vector Machine: SVM

```
class SVM:  
    def __init__(self,kernel='linear', learning_rate=0.001,lambda_param=0.01 ,epoch=1000, debug=False, verbose=False)--> None:  
        self.kernel = kernel  
        self.learningRate = learning_rate  
        self.lambda_param = lambda_param  
        self.epoch = epoch          # 1 epoch for n sample  
        self.w = None                # Weight  
  
        self.debug = debug          # True / False  
        self.gradient_round = 0      # Gradient step  
        self.verbose = verbose  
  
        # history for plot graph  
        self.cost_function = []  
        self.zero_one_loss = []  
  
    if self.debug:  
        print("-- Parameter --\nLearning Rate: {}\nLambda Param: {}\nN_Iters: {}-----".format(self.learningRate, self.lambda_param, self.epoch))
```



# Support Vector Machine: SVM

```
def bias(self, features):                                     # add b to w for not compute b
    n_samples = len(features)
    return np.concatenate( (np.ones((n_samples, 1)), features), axis=1 )  # [1 , x1, x2, ..., xn]

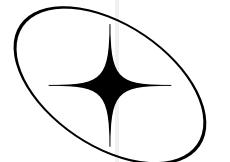
def gradient(self, sample, label):                                # Slack higeloss
    slack = label * np.dot(self.w, sample)

    n_feature = sample.shape[0]
    gradient = np.zeros(n_feature)                               # Set array w for feature + b

    if slack >= 1:                                              # if >=1 yi * (w * xi)
        gradient = (self.lambda_param * self.w)

    else:                                                       # if <1 1 - yi * (w * xi)
        gradient = (self.lambda_param * self.w) - (sample * label)

    self.gradient_round += 1      # Count round
    return gradient
```



# Support Vector Machine: SVM

```
def fit(self, x_train, y_train):
    x_train = self.bias(x_train)           # Add b in w
    x_samples = x_train.shape[1]          # get Index of sample, sample
    self.w = np.zeros(x_samples)          # Set w size feature x

    for epoch in range(self.epoch):       # Train Epoch round
        zr_oe_loss = 0
        for index, x_sample in enumerate(x_train):
            # Compute Gradient
            gradient = self.gradient(x_sample, y_train[index]) # Find gradient

            # Update Weight
            self.w -= self.learningRate * gradient             # Update new weight

            # Save History Loss
            predict = np.sign(np.dot(self.w, x_sample))
            if predict != y_train[index]:
                zr_oe_loss += 1

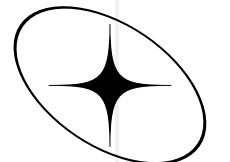
        avr_gradient = sum(gradient) / len(x_sample)          # Find average gradient of array gradient

        cost_w = 1 / 2 * np.dot(self.w, self.w) + (self.lambda_param * avr_gradient) # Cal Cost(w) of regularization

        if self.verbose == True:
            print(cost_w)

        self.cost_function.append(cost_w)
        self.zero_one_loss.append(zr_oe_loss)

    if self.debug:
        print("Gradient {} steps.".format(self.gradient_round))
```



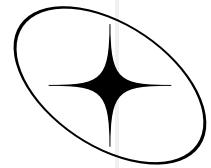
# Support Vector Machine: SVM

```
def predict(self, test_features):
    test_features = self.bias(test_features)      # Add b to w
    buffer = []                                    # List for y predict

    for x_sample in test_features:                # Predict from list
        predict = np.dot(self.w, x_sample)          # Predict
        if predict < 0:
            buffer.append(-1)
        elif predict > 0:
            buffer.append(1)
        else:
            buffer.append(0)
    return buffer

def score(self, x_test, y_test):
    counter = 0                                     # Counter correct
    predict = self.predict(x_test)                  # Predict

    for index, sample_test in enumerate(predict):
        if sample_test == y_test[index]:             # If equal +1
            counter += 1
    return counter / len(x_test)                    # Return Accuracy 0 - 1
```

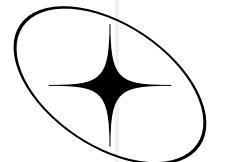


# Preprocessing Data

- เนื่องจากข้อมูลตัวอย่างบางตัวมี **#NUM!** อยู่
- ลบข้อมูลตัวอย่างออกที่มี **#NUM!** ออก
- เปลี่ยนชนิดข้อมูล Column **ammonia** และ Column **is\_safe** ให้เป็น Type Float

alumin...	ammonia	arsenic	barium	cadmium	chloram...	chromium
0.03	#NUM!	0.08	0.79	0.07	0.08	0.05
0.1	8.76	0.1	0.66	0.07	0.03	0
0.04	0.26	0.08	1.58	0.08	0.01	0.08

✓ 0.0s



# แบ่งชุดข้อมูล

- แบ่งข้อมูลออกเป็น x และ y
- เปลี่ยน Label เวลาจะจาก 0, 1 เป็น -1, 1 เพื่อให้สามารถนำไปคำนวณใน model ได้

- แบ่ง x, y ออกเป็น Train, Validate และ Test โดยใช้อัตราส่วน 5 : 3 : 2
  - Train: 5 ส่วน
  - Validate: 3 ส่วน
  - Test: 2 ส่วน

```
from model_selection import train_test_split

buff_x, x_test, buff_y, y_test = train_test_split(x,y, test_size=0.2, random_state=42)
x_train, x_validate, y_train, y_validate = train_test_split(buff_x,buff_y, test_size=0.3, random_state=13)
# x_train

✓ 0.0s
```

- โดยใช้ Train, Validate ในการ Training และปรับ Parameter และใช้ Test ในการวัดประสิทธิภาพตอนสุดท้าย

```
x = np.array(df.drop(['is_safe'], axis=1))
y = np.array(df['is_safe'].copy())

from model_selection import positive_negative_check
p,s = positive_negative_check(y)

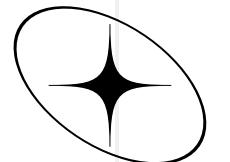
✓ 0.0s

Total 7996 Samples.
Positive Class [1.0]: 912 sample.
Negative Class [0.0]: 7084 sample.
```

```
y = np.where(y <= 0, -1, 1)
p, s = positive_negative_check(y)

✓ 0.0s

Total 7996 Samples.
Positive Class [1]: 912 sample.
Negative Class [-1]: 7084 sample.
```



# ทดลองโดยใช้ SVM

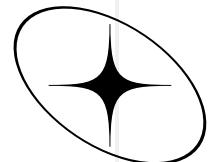
- นำ x\_train และ y\_train ไปใช้ใน SVM โดยใช้ Default Parameter
- ทดสอบโดยใช้ x\_validate, y\_validate โดยได้ประสิทธิภาพอยู่ที่ **0.8905**
- เมื่อลองแสดงค่า Predict ที่ได้ออกมา ผลที่ได้คือ model predict -1 เป็นจำนวนมาก
- คาดการณ์ได้ว่า model นี้เรียนรู้ด้วยชุดข้อมูลที่มีจำนวน Positive, Negative แตกต่างกันเกินไป หรือ เกิดปัญหา (Imbalance DataSet)

```
from model import SVM
model1 = SVM(kernel='linear', learning_rate=0.001, epoch=1000, debug=True, lambda_param=0.01)
model1.fit(x_train, y_train)

predict_test = model1.predict(x_test)
predict_validate = model1.predict(x_validate)

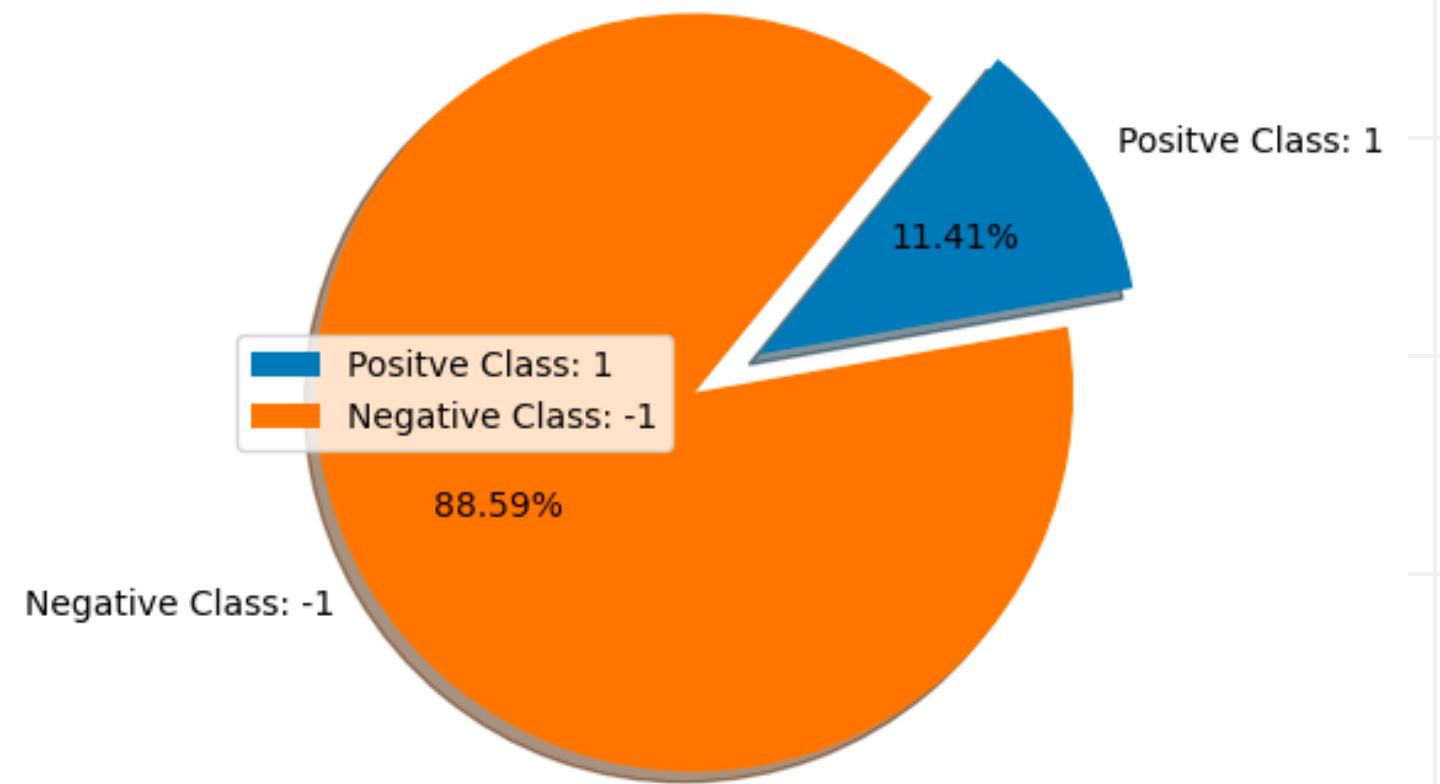
print("Validate Accuracy:",model1.score(x_validate, y_validate))
print("y_validate:",len(y_validate), "sample")
for i in range(20):
    print(predict_validate[i], y_validate[i])
✓ 14.3s
```

Parameter	Value
Learning Rate	0.001
Lambda Param	0.01
N_Iters	1000
-----	
Gradient	4478000 steps.
Validate Accuracy	0.890568004168838
y_validate	1919 sample
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	1

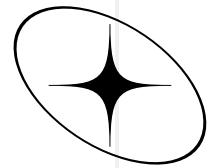


# ปัญหา Imbalance DataSet

- จำนวนของ Negative Class มีมากกว่า Positive Class
- ทำให้ Model ที่ใช้ มีการเรียนรู้โดยปรับ weight ไปในทาง Negative Class มากกว่า Positive Class
- ส่งผลให้ model ทำนาย Negative Class ได้ดีกว่า Positive Class
- โดยมี Positive Class [1]: 912 sample  
Negative Class [-1]: 7084 sample



```
from model_selection import positive_negative_check
p,s = positive_negative_check(y)
✓ 0.0s
Total 7996 Samples.
Positive Class [1]: 912 sample.
Negative Class [-1]: 7084 sample.
```



# แก้ปัญหา Imbalance DataSet

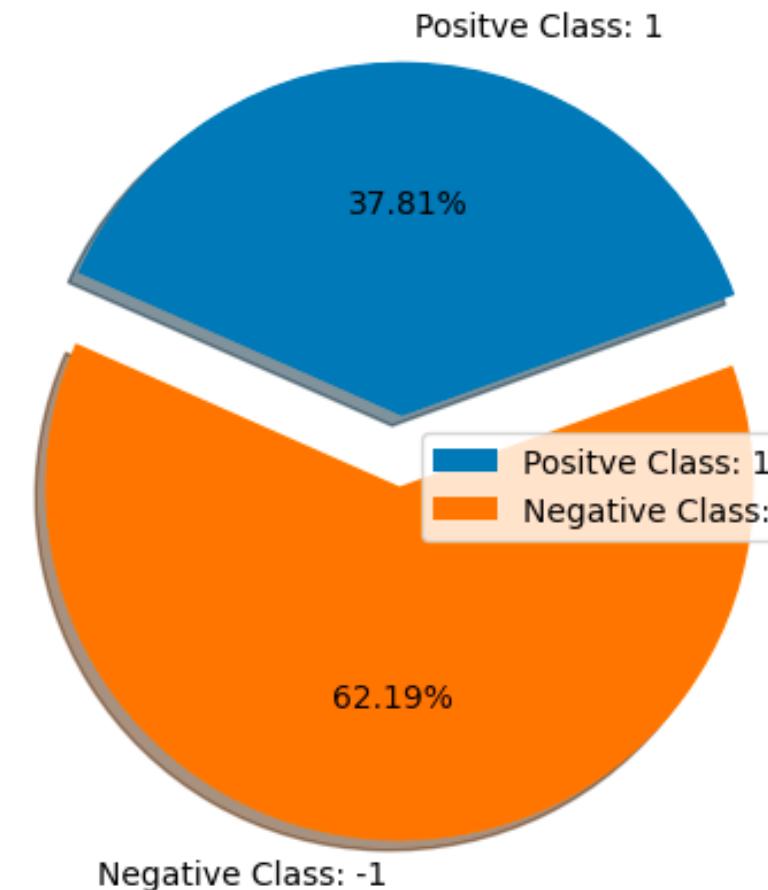
- เรียก Class ที่มีจำนวนมาก ว่า Majority Class และ Class ที่มีจำนวนน้อยกว่าคือ Minority Class
- แก้ปัญหาด้วยการทำ **Random Under Sampling** คือ การลดจำนวน Majority Class ลง โดยใช้วิธีสุ่มลบข้อมูลตัวอย่างออก
- โดยลดจำนวน Majority Class หรือ Negative Class ลงให้เหลือ 1500 sample
- หลังจากที่แก้ม Positive Class [1]: 912 sample Negative Class [-1]: 1500 sample

```
df_majority = df[df.is_safe == 0]
df_minority = df[df.is_safe == 1]
✓ 0.0s
```

```
from model_selection import random_under_sampling
df_majority_downsampled = random_under_sampling(df_majority, 1500, random_state=32)
```

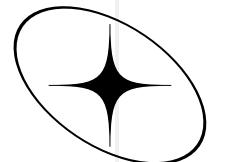
```
df_downsampled = pd.concat([df_majority_downsampled, df_minority])
df_downsampled
✓ 0.0s
```

Current Size: 7084  
Delete Size Target: 5584



```
y = np.where(y <= 0, -1, 1)
p, s = positive_negative_check(y)
✓ 0.0s
```

Total 2412 Samples.  
Positive Class [1]: 912 sample.  
Negative Class [-1]: 1500 sample.



# ทดลองโดยใช้ SVM

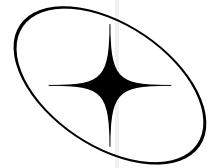
- นำ x\_train และ y\_train ไปใช้ใน SVM โดยใช้ Default Parameter
- ทดสอบโดยใช้ x\_validate, y\_validate โดยได้ประสิทธิภาพอยู่ที่ **0.7737**
- ประสิทธิภาพที่ได้ลดลงเนื่องจาก ลดจำนวน Negative Class ให้ความแม่นยำลดน้อยลง

```
from model import SVM
model1 = SVM(kernel='linear', learning_rate=0.001, epoch=1000, debug=True, lambda_param=0.01)
model1.fit(x_train, y_train)

predict = model1.predict(x_test)
predict_validate = model1.predict(x_validate)

print("Validate Accuracy:", model1.score(x_validate, y_validate))
✓ 4.7s

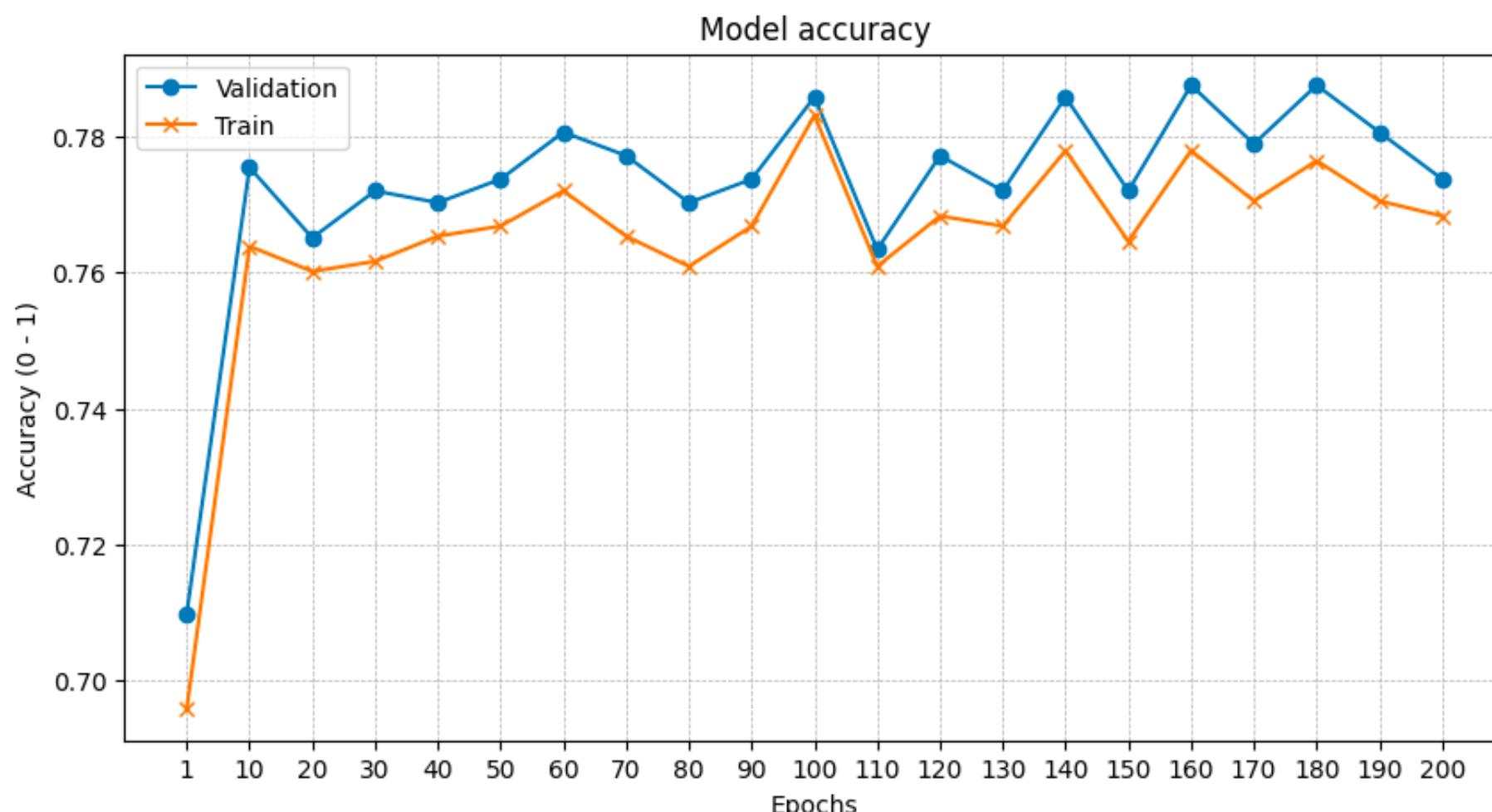
-- Parameter --
Learning Rate: 0.001
Lambda Param: 0.01
N_Iters: 1000
-----
Gradient 1351000 steps.
Validate Accuracy: 0.7737478411053541
```



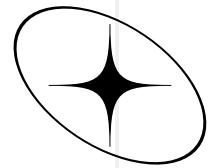
# Parameter SVM: Epoch

- Epoch คือ จำนวนครั้งที่ให้ model Train ครบทุก samples
- หา parameter Epoch ที่ดีที่สุดผ่าน Validate
- ทดสอบโดยการหาผ่าน Validate โดยได้ประสิทธิภาพดีที่สุดคือ **0.7875** เมื่อใช้ Epoch: **160**

```
epoch = [1, 10 , 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180 ,190 ,200]
model1.plot_accuracy(x_train,y_train, x_validate, y_validate, epoch, verbose=True)
```



Epoch 150  
Validate Accuracy is: 0.772020725388601  
Train Accuracy is: 0.764618800888231  
Epoch 160  
Validate Accuracy is: 0.7875647668393783  
Train Accuracy is: 0.7779422649888971

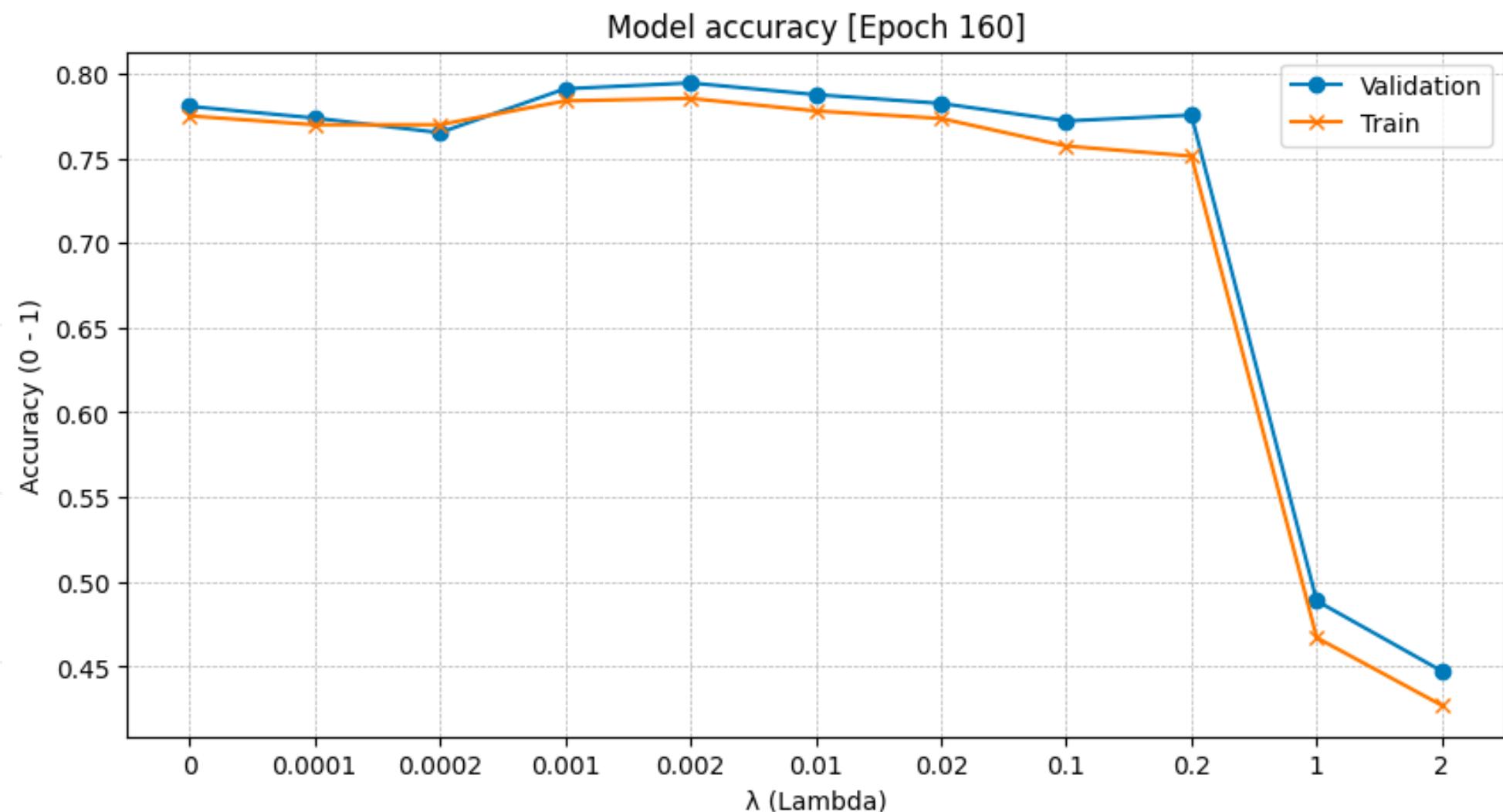


# Parameter SVM: $\lambda$ (Lambda)

- หา Lambda จาก Parameter Epoch: 160
- Lambda  $\rightarrow$  0 (Original SVM)  
Lambda  $\rightarrow$  1 (เกิด Loss เพิ่มขึ้น)
- หา parameter Lambda ที่ดีที่สุดผ่าน Parameter Epoch และ Validate
- ทดสอบโดยการหาผ่าน Validate โดยได้ประสิทธิภาพดีที่สุดคือ **0.7944** เมื่อใช้ Epoch: 160 และ Lambda: 0.002

```
lambda_param = [0, 0.0001, 0.0002, 0.001, 0.002, 0.01, 0.02, 0.1, 0.2, 1, 2]
model1.plot_lambda(x_train,y_train, x_validate, y_validate, lambda_param,160, verbose=True)
```

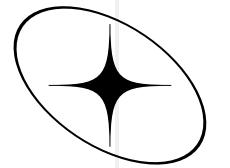
✓ 8.4s



Lambda 0.002

Validate Accuracy is: 0.7944732297063903

Train Accuracy is: 0.7853441894892672



# SVM Classifier

- นำ x\_train และ y\_train ไปใช้ใน SVM โดยใช้ Parameter ที่ได้มา คือ **Epoch: 160, Lambda: 0.002**
- ทดสอบโดยใช้ x\_validate, y\_validate โดยได้ประสิทธิภาพอยู่ที่ **0.7944** และ ทดสอบด้วย x\_test, y\_test โดยได้ประสิทธิภาพอยู่ที่ **0.8278**

```
from model import SVM
model2 = SVM(kernel='linear', learning_rate=0.001, epoch=160, debug=True, lambda_param=0.002)
model2.fit(x_train, y_train)

predict_test = model2.predict(x_test)
predict_validate = model2.predict(x_validate)

print("Validate Accuracy:", model2.score(x_validate, y_validate))
print("Test Accuracy:", model2.score(x_test, y_test))
model2.plot_loss()

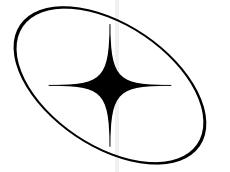
✓ 0.8s

-- Parameter --
Learning Rate: 0.001
Lambda Param: 0.002
N_Iters: 160
-----
Gradient 216160 steps.
Validate Accuracy: 0.7944732297063903
Test Accuracy: 0.8278008298755186
```

```
from model_selection import confusion_matrix
confusion_matrix(predict_test, y_test)

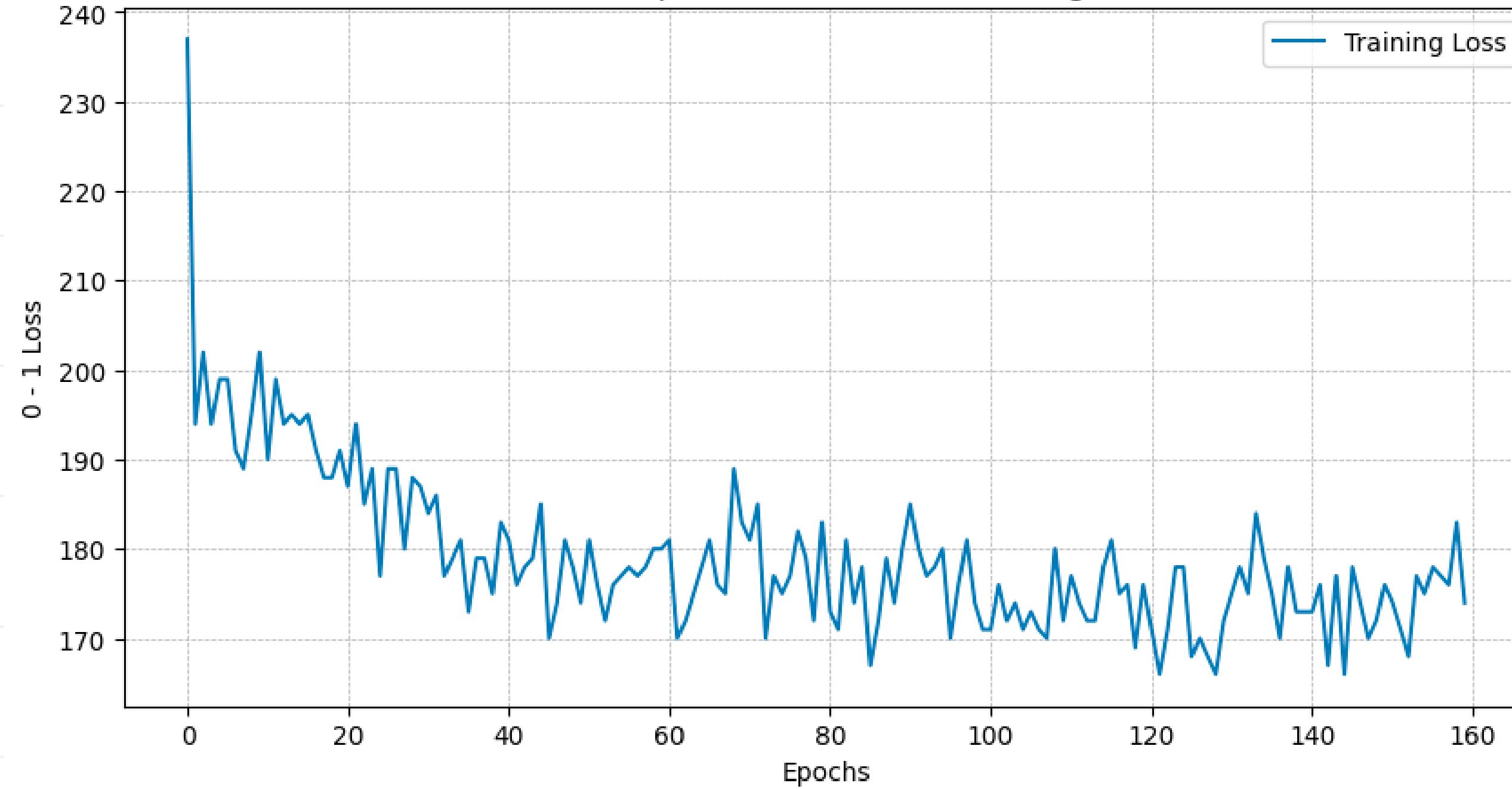
✓ 0.0s

Precision: 0.7647
Recall: 0.7514
Accuracy: 0.8278
F1-score: 0.7580
```

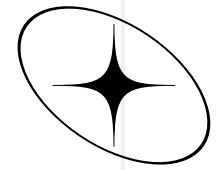


# SVM Classifier

SVM with Epoch[160],  $\lambda[0.002]$  Training Loss



# ຕາரາງເປົ້າຍບເທິຍບ SVM

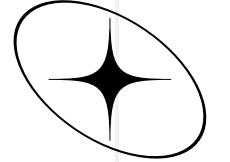


Epoch	Lambda	Validate accuracy	Test accuracy
60	0.0002	0.7772	0.7946
100	0.0002	0.7875	0.8153
140	0.0002	0.7737	0.7925
160	0.0002	0.7651	0.7966
180	0.0002	0.7875	0.8049

Epoch	Lambda	Validate accuracy	Test accuracy
60	0.02	0.7702	0.7925
100	0.02	0.7823	0.8008
140	0.02	0.7582	0.7883
160	0.02	0.7823	0.8029
180	0.02	0.7841	0.8132

Epoch	Lambda	Validate accuracy	Test accuracy
60	0.002	0.7875	0.8029
100	0.002	0.7858	0.8008
140	0.002	0.7823	0.8215
160	0.002	0.7944	0.8278
180	0.002	0.7806	0.7966

Epoch	Lambda	Validate accuracy	Test accuracy
60	1	0.5025	0.4875
100	1	0.6511	0.6037
140	1	0.6666	0.6431
160	1	0.4887	0.4730
180	1	0.6338	0.5933



# Conclusion

- จากการทดลองโดยใช้ Support Vector Machine: SVM กับชุดข้อมูล Water Quality ในการจำแนกน้ำว่ามีความสะอาด หรือ มีความปลดออกซิเจนต่ำ ได้ประสิทธิภาพดีที่สุด คือ **0.8278** เมื่อปรับ

**Paramter Epoch: 160, Lambda: 0.002 และ Learning Rate: 0.001**

- โดยได้ **Precision: 0.7647**

**Recall: 0.7514**

**Accuracy: 0.8278**

และ **F1-score: 0.7580**

```
from model_selection import confusion_matrix
confusion_matrix(predict_test, y_test)
✓ 0.0s
Precision: 0.7647
Recall: 0.7514
Accuracy: 0.8278
F1-score: 0.7580
```



WATER

**THANK YOU**

