# 類神經網路基礎應用1

王豐緒
銘傳大學資工系

# 學習目標

- 理解Perceptron感知器類神經網路的限制
- 藉由Pima數據案例了解非線性可分的問題
- 理解數據前置(預)處理的重要性
- 具備數據前置(預)處理的實務能力

# 大綱

- Pima 案例
- 資料集描述
- 顯示 Pima 資料
- Perceptron網路結構
- 資料前處理

# PIMA 案例

- 資料集
  - The American Pima Indians
  - Downloaded from (http://archive.ics.uci.edu/ml)
  - 分類問題 (糖尿病， diabetes)

# 資料集描述

- 資料欄位介紹 (全部數值型資料)
  - 1.懷孕次數
  - 2. 口服葡萄糖耐受試驗中2小時的血漿葡萄糖濃度
  - 3.舒張壓（毫米汞柱）
  - 4.肱三頭肌皮褶厚度（mm）
  - 5. 2小時血清胰島素（mu U/ml）
  - 6. 體重指數（體重公斤/（身高公尺）^2）
  - 7. 糖尿病譜系功能
  - 8. 年齡（歲）
  - 9. 類別變數（0或1）
- 案例數: 768

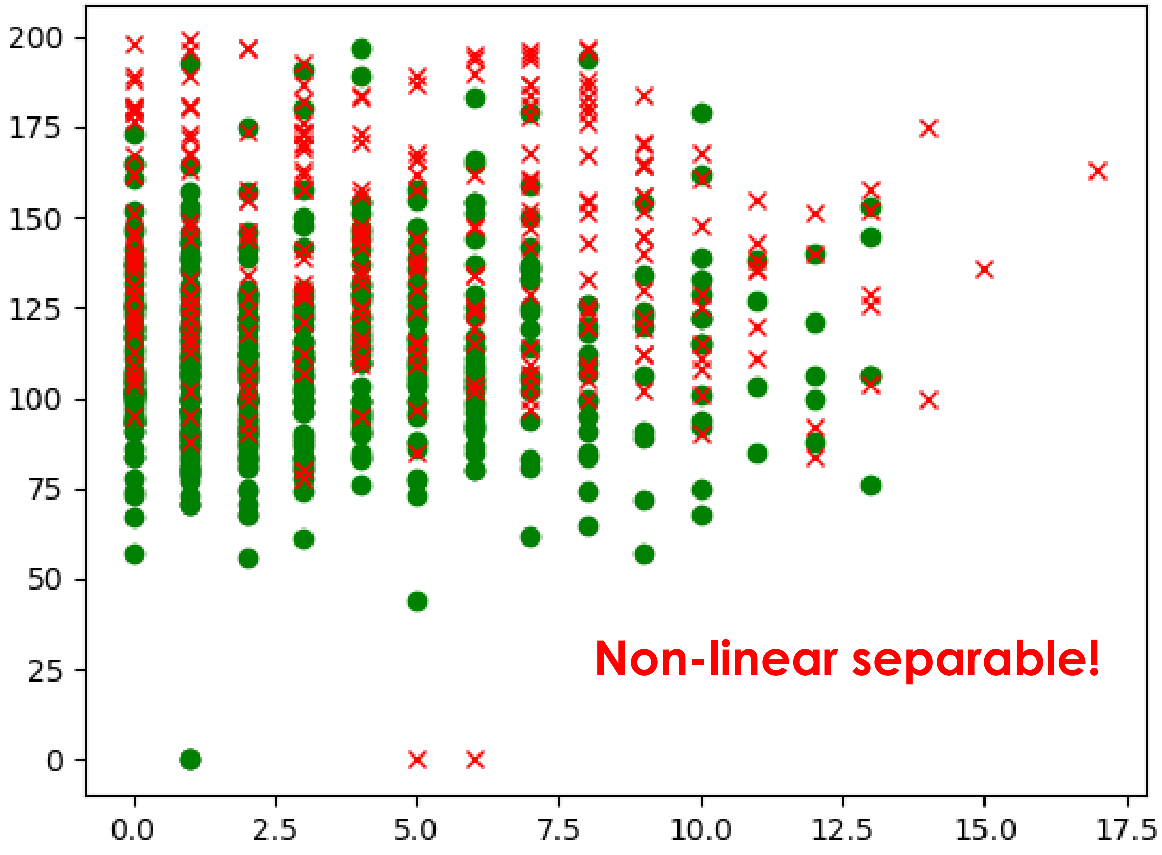
# 顯示 PIMA 資料(程式碼)

```
import pylab as pl
import numpy as np

pima = np.loadtxt('./Pima/pima-indians-diabetes.data',delimiter=',')
pima.shape #dimensions of the array (768, 9)(前8個欄位是輸入，最後一個欄位是輸出)

# Plot the first and second values for the two classes
indices0 = np.where(pima[:,8]==0) #no diabetes, a tuple of row indices as a single ndarray
indices1 = np.where(pima[:,8]==1) #have diabetes a tuple of row indices a single ndarray

pl.ion()
pl.plot(pima[indices0,0],pima[indices0,1],'go')  #as green circles (show scatter plot)
pl.plot(pima[indices1,0],pima[indices1,1],'rx')  #as red cross  (show scatter plot)
pl.show()
```

# 顯示 PIMA 資料(執行結果)

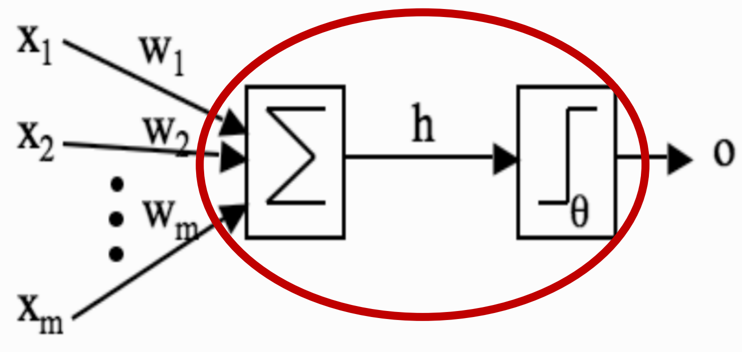
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| 3 | 78 | 50 | 32 | 88 | 31 | 0.248 | 26 | 1 |
| 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 | 1 |
| 4 | 110 | 92 | 0 | 0 | 37.6 | 0.191 | 30 | 0 |
| 10 | 139 | 80 | 0 | 0 | 27.1 | 1.441 | 57 | 0 |
| 5 | 166 | 72 | 19 | 175 | 25.8 | 0.587 | 51 | 1 |
| 0 | 118 | 84 | 47 | 230 | 45.8 | 0.551 | 31 | 1 |
| 1 | 103 | 30 | 38 | 83 | 43.3 | 0.183 | 33 | 0 |

X: **Positive test**



# PERCEPTRON網路結構(程式碼)

```python
# Custom activation function
from keras.layers import Activation
from keras import backend as K
from tensorflow.keras.utils import get_custom_objects
import tensorflow as tf
import numpy as np

def custom_activation(x):
   #1 if x>0 else 0
   result1 = tf.cast(tf.math.greater(x, 0), tf.float32)
   return result1

get_custom_objects().update({'custom_activation':
Activation(custom_activation)})
```




$$h = \sum_i W_i X_i - \theta$$

$$o = g(h) = \begin{cases} 1 & if\ h > 0 \\ 0 & if\ h \leq 0 \end{cases}$$

$g$：激活函數 (activation function)

# PERCEPTRON網路結構(程式碼)

```
class CustomModel(tf.keras.Model):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.loss_tracker = tf.keras.metrics.Mean(name="loss")
        self.mse_metric = tf.keras.metrics.MeanSquaredError(name="mse")
    def train_step(self, data):
      # Unpack the data. Its structure depends on your model and
      # on what you pass to `fit()`.
      x, y = data #x: input  y:label
      with tf.GradientTape() as tape:
          y_pred = self(x, training=True)  # Forward pass
          # Compute the loss value
          # (the loss function is configured in `compile()`)
          loss = tf.keras.losses.mean_squared_error(y, y_pred)
      …
```

# PERCEPTRON網路結構(程式碼)

$\triangle w_{ij} = -\eta(y_j - t_j) \cdot x_i$

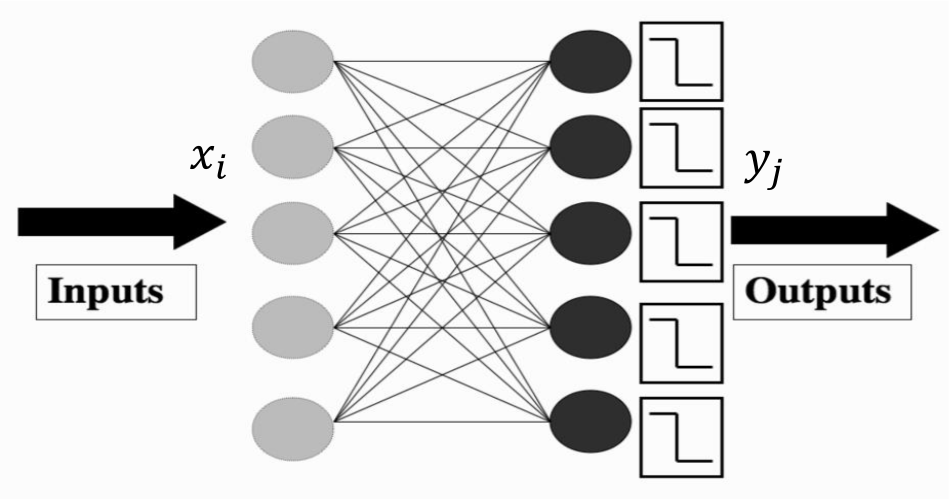
```
class CustomModel(tf.keras.Model):
      …
      # Compute gradients
      Xupdates = -eta*K.dot(K.transpose(x),y_pred-y)
      Thetaupdates = -eta*K.dot(K.transpose(Theta),y_pred-y)
      Thetaupdates = tf.reshape(Thetaupdates, shape=(1,))
      self.trainable_variables[0].assign( self.trainable_variables[0]+Xupdates)
      self.trainable_variables[1].assign( self.trainable_variables[1]+Thetaupdates)

      # Compute our own metrics
      self.loss_tracker.update_state(loss)
      self.mse_metric.update_state(y, y_pred)
      return {"loss": self.loss_tracker.result(), "mse": self.mse_metric.result()}
```

# PERCEPTRON網路結構(程式碼)

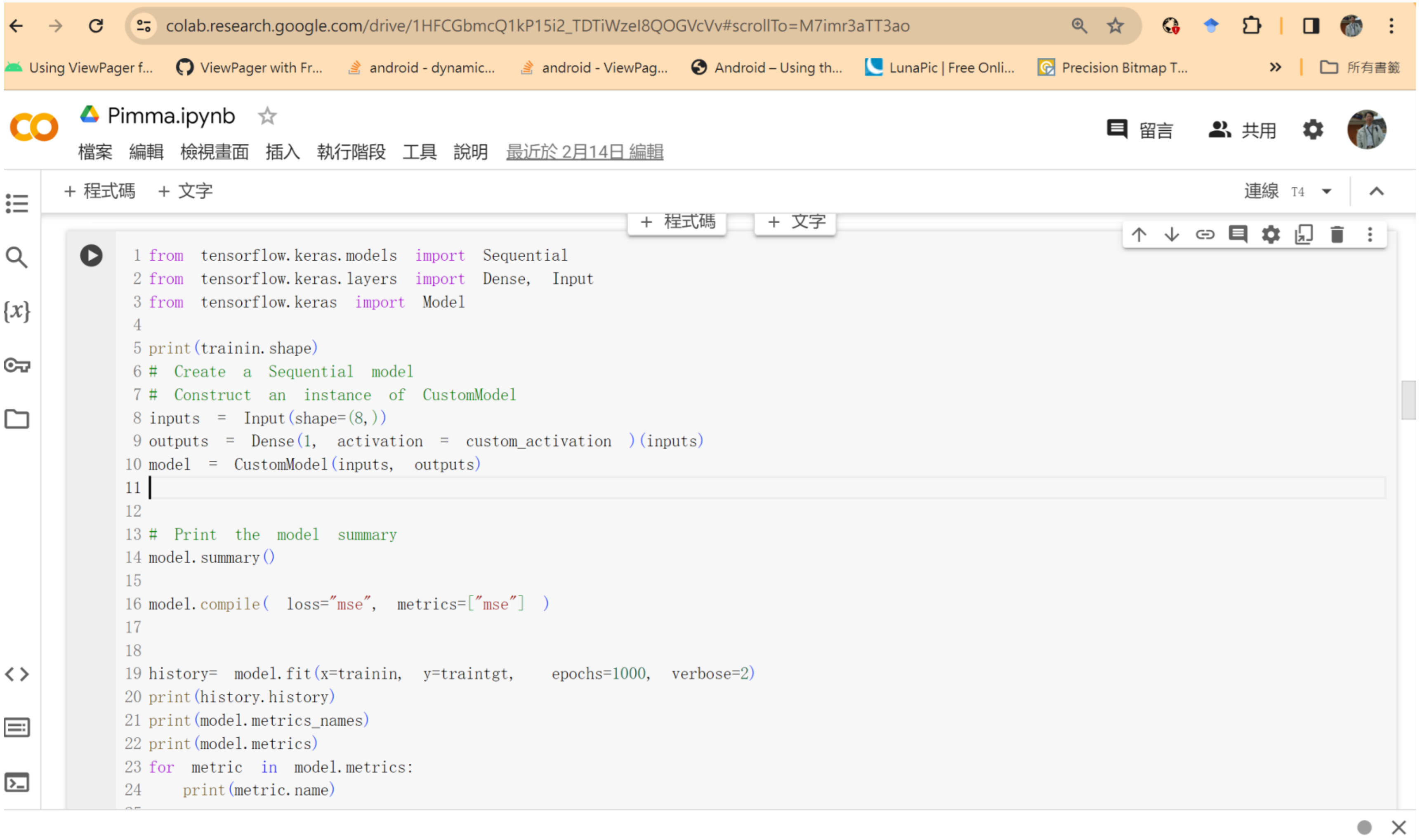
```
class CustomModel(tf.keras.Model):
    …
    @property
    def metrics(self):
        # We list our `Metric` objects here so that `reset_states()` can be
        # called automatically at the start of each epoch
        # or at the start of `evaluate()`.
        # If you don't implement this property, you have to call
        # `reset_states()` yourself at the time of your choosing.
        return [self.loss_tracker, self.mse_metric]
```

# PERCEPTRON網路結構(程式碼)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input

# Create a Sequential model
# Construct an instance of CustomModel
inputs = Input(shape=(8,))
outputs = Dense(1, activation = custom_activation )(inputs)
model = CustomModel(inputs, outputs)
# Print the model summary
model.summary()
model.compile( loss="mse", metrics=["mse"] )
history= model.fit(x=trainin, y=traintgt, epochs=1000,
verbose=2)
```



# COLAB執行範例

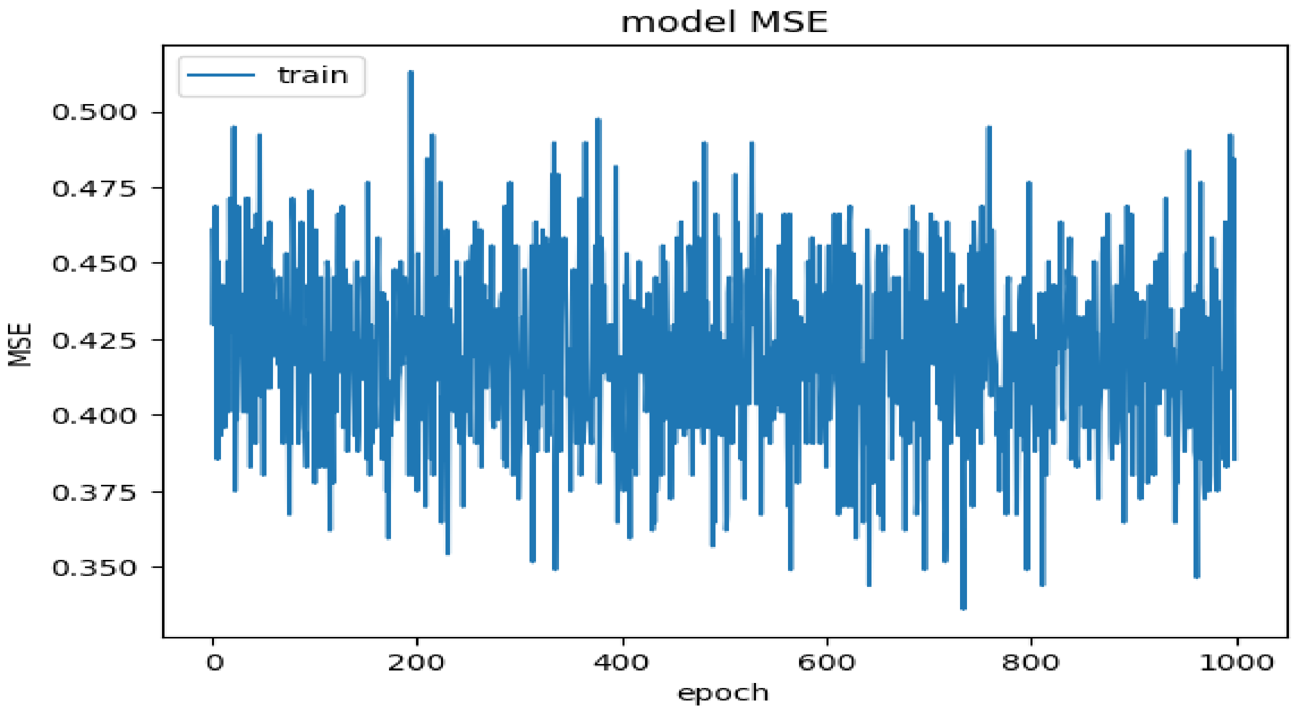

colab.research.google.com/drive/1HFCGbmcQ1kP15i2_TDTiWzel8QOGVcVv#scrollTo=M7imr3aTT3ao
Pimma.ipynb
檔案 編輯 檢視畫面 插入 執行階段 工具 說明 最近於 2月14日 編輯
留言
共用
+ 程式碼 + 文字
連線 T4
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras import Model
print(trainin.shape)
# Create a Sequential model
# Construct an instance of CustomModel
inputs = Input(shape=(8,))
outputs = Dense(1, activation = custom_activation )(inputs)
model = CustomModel(inputs, outputs)
# Print the model summary
model.summary()
model.compile( loss="mse", metrics=["mse"] )
history= model.fit(x=trainin, y=traintgt, epochs=1000, verbose=2)
print(history.history)
print(model.metrics_names)
print(model.metrics)
for metric in model.metrics:
print(metric.name)

# PERCEPTRON (未經前置處理)

```
Model: "custom_model_7"
_________________________________________________________________
Layer    (type)          Output Shape          Param #
=================================================================
input_9 (InputLayer) [(None, 8)]                 0
dense_28 (Dense)         (None, 1)               9
=================================================================
Total params: 13 (52.00 Byte)
Trainable params: 9 (36.00 Byte)
Non-trainable params: 4 (16.00 Byte)
_________________________________________________________________
Epoch 1/1000 12/12 - 0s - loss: 0.4609 - mse: 0.4609 - 155ms/epoch - 13ms/step
Epoch 2/1000 12/12 - 0s - loss: 0.4297 - mse: 0.4297 - 25ms/epoch - 2ms/step
Epoch 3/1000 12/12 - 0s - loss: 0.4531 - mse: 0.4531 - 26ms/epoch - 2ms/step
Epoch 4/1000 12/12 - 0s - loss: 0.4688 - mse: 0.4688 - 24ms/epoch - 2ms/step
Epoch 5/1000 12/12 - 0s - loss: 0.4505 - mse: 0.4505 - 24ms/epoch - 2ms/step
…
```

# PERCEPTRON (未經前置處理)

```
# summarize history for accuracy
import matplotlib.pyplot as plt
plt.plot(history.history['mse'])
plt.title('model mse')
plt.ylabel(MSE')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```



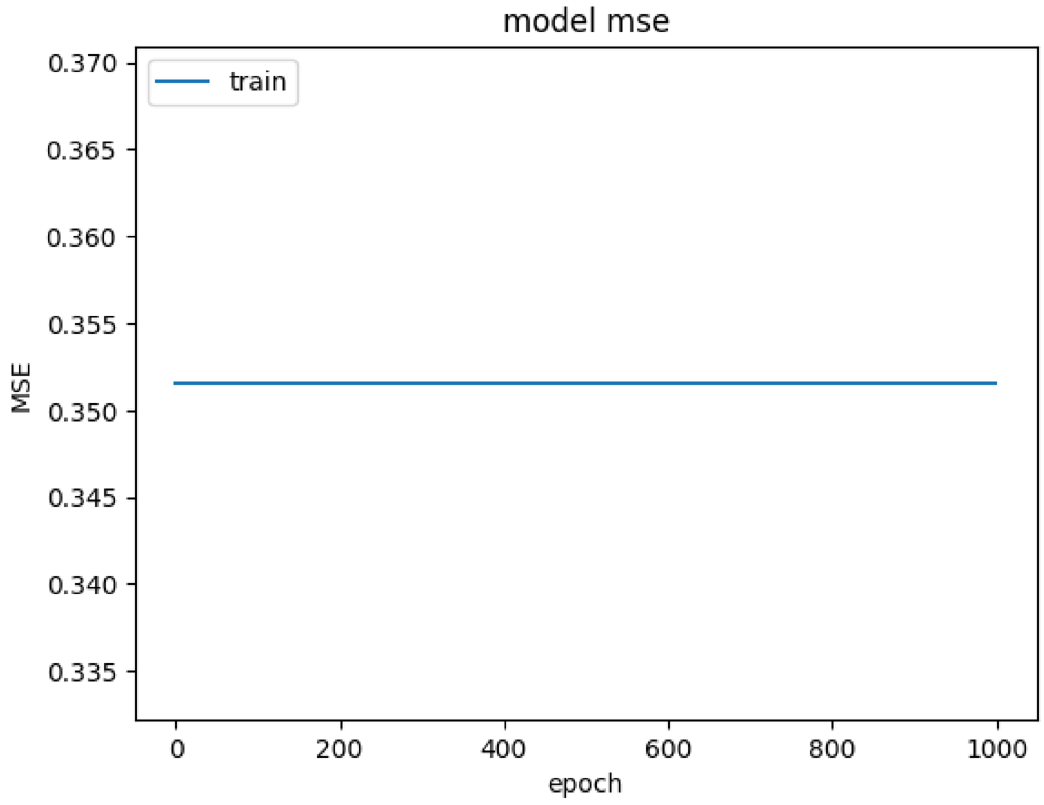
# PERCEPTRON (經前置處理)

```
# Various preprocessing steps
pima[np.where(pima[:,0]>8),0] = 8 #set the first column with more than 8 pregnant times as 8
pima[np.where(pima[:,7]<=30),7] = 1 #set the 7'th column (age) into intervals (1)
pima[np.where((pima[:,7]>30) & (pima[:,7]<=40)),7] = 2 #set the 7'th column (age) into intervals (2)
pima[np.where((pima[:,7]>40) & (pima[:,7]<=50)),7] = 3 #set the 7'th column (age) into intervals (3)
pima[np.where((pima[:,7]>50) & (pima[:,7]<=60)),7] = 4 #set the 7'th column (age) into intervals (4)
pima[np.where(pima[:,7]>60),7] = 5 #set the 7'th column (age) into intervals (5)
pima[:,:8] = pima[:,:8]-pima[:,:8].mean(axis=0) #normalize the first 8 columns, mean
pima[:,:8] = pima[:,:8]/pima[:,:8].var(axis=0) #normalize the first 8 columns, var
trainin = pima[::2,:8] #even rows as training data
testin = pima[1::2,:8] #odd rows as testing data
traintgt = pima[::2,8:9] #even rows as training label
testtgt = pima[1::2,8:9] #odd rows as testing label
# Perceptron training on the preprocessed dataset
print ("Output after preprocessing of data")
```

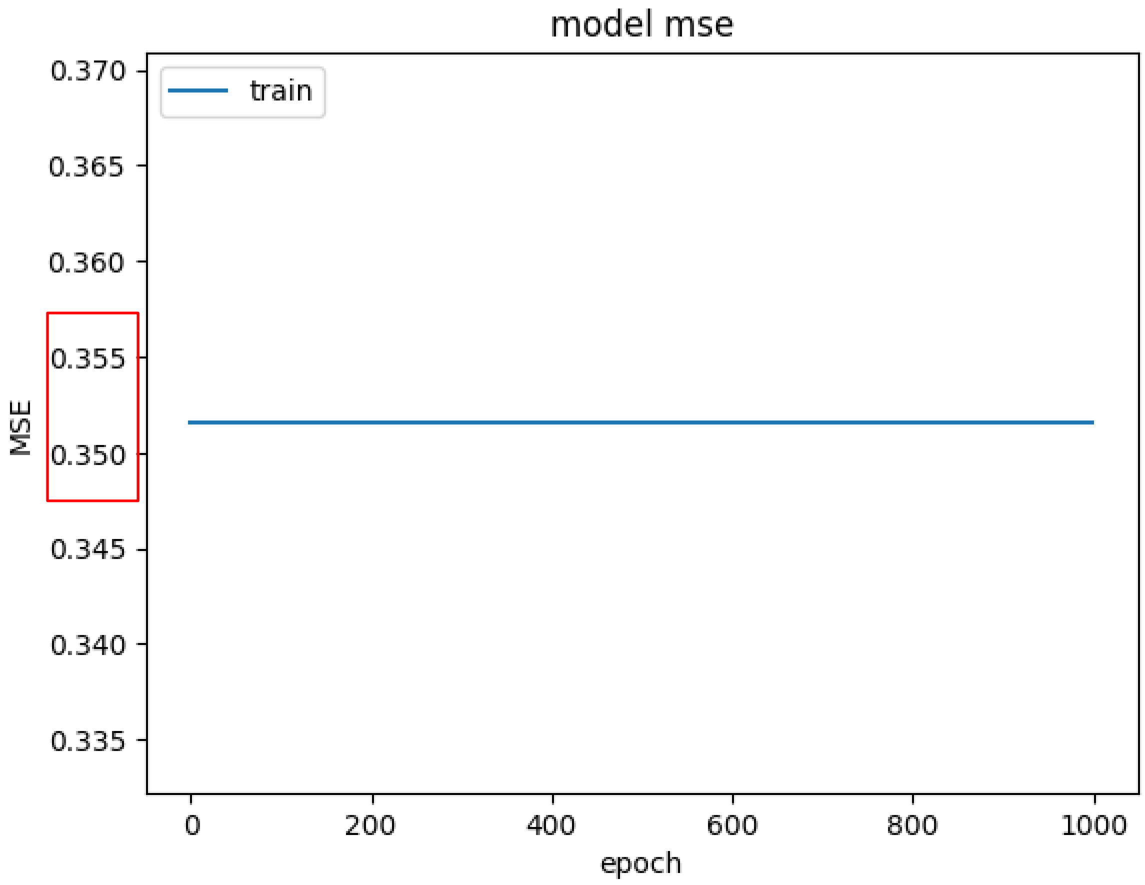
**pima[*r1*:*rn*, *c1*:*cm*]:** 存取行列資料

# PERCEPTRON (經前置處理)

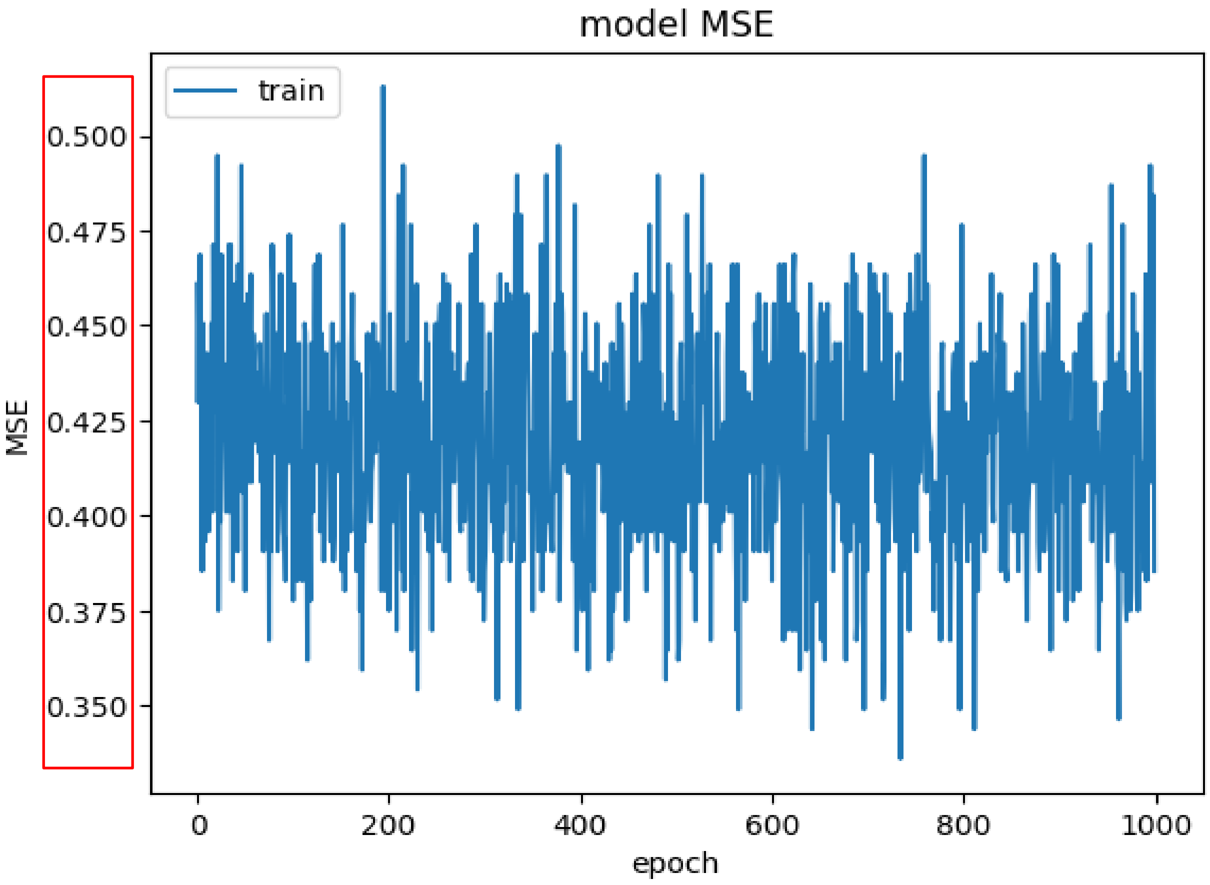
```python
# summarize history for accuracy
import matplotlib.pyplot as plt
plt.plot(history.history['mse'])
plt.title('model mse')
plt.ylabel(MSE')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```



# PERCEPTRON (經前置處理)




with Normalization




without Normalization

# 小結

- Pima數據集不是線性可分的
  - 感知器表現不佳
- 數據預處理可能有助於提高性能
  - 數據正規化
  - 離散化

# Q&A