0.1 JOBSHEET 9 Perceptron dan ANN

Nama : Elis Nurhidayati

NIM : 2241720035

Kelas : TI-3C

Link Google Colab :
https://colab.research.google.com/drive/19zovycD6od58yahSbHDQjI5uBn-pMsAg?usp=sharing

1 Praktikum 1

Klasifikasi Iris dengan Perceptron

1.1 Deskripsi

Pada pratikum ini, Anda diminta untuk melakukan klasifikasi bunga iris dengan menggunakan model Perceptron. Anda dapat menggunakan dataset iris pada praktikum sebelumnya.

Untuk menambah pemahaman Anda terkait dengan model Perceptron, pada pratkikum ini Anda akan membuat model Perceptron tanpa menggunakan library.

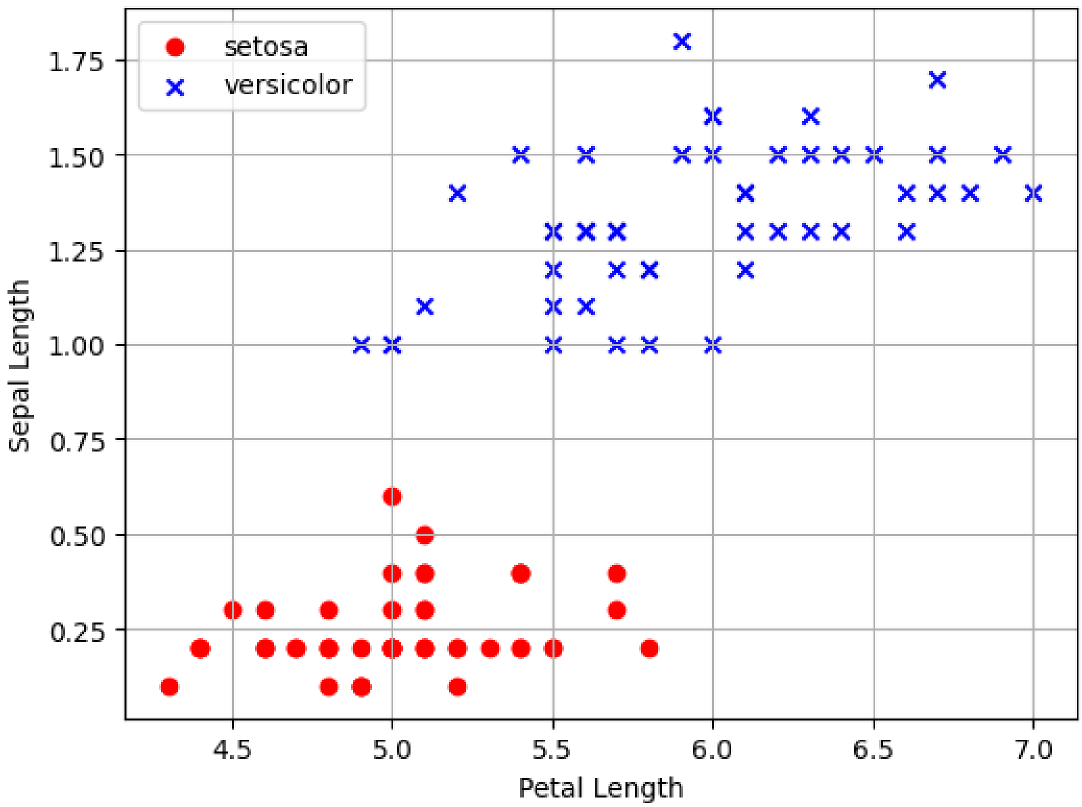
1.2 Langkah 1 - Import Library

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

1.3 Langkah 2 - Load Data dan Visualisasi

[13]:
```
# Membaca data iris dari file CSV
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
df = pd.read_csv(url, header=None)

# Memisahkan data berdasarkan kelas
setosa = df[df[4] == 'Iris-setosa']
versicolor = df[df[4] == 'Iris-versicolor']
virginica = df[df[4] == 'Iris-virginica']
```

```python
# Indeks kolom yang akan diplot (Petal Length dan Sepal Length)
a, b = 0, 3

# Membuat scatter plot
plt.scatter(setosa.iloc[:, a], setosa.iloc[:, b], color='red', marker='o',␣
↪label='setosa')
plt.scatter(versicolor.iloc[:, a], versicolor.iloc[:, b], color='blue',␣
↪marker='x', label='versicolor')

# Mengatur label sumbu x dan y, serta legenda
plt.xlabel('Petal Length')
plt.ylabel('Sepal Length')
plt.legend(loc='upper left')

# Menampilkan plot
plt.grid()
plt.show()
```



1.4 Langkah 3 - Membuat Kelas Perceptron

[15]:
```
class Perceptron(object):
    def __init__(self, eta=0.01, n_iter=10):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):

        self.w_ = np.zeros(1 + X.shape[1])
        self.errors_ = []

        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[0] += update
                self.w_[1:] += update * xi
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

1.5 Langkah 4 - Pilih Data dan Encoding Label

[16]:
```
y = df.iloc[0:100, 4].values # pilih 100 data awal
y = np.where(y == 'Iris-setosa', -1, 1) # ganti coding label
X = df.iloc[0:100, [0, 3]].values # slice data latih
```

1.6 Langkah 5 - Fitting Model

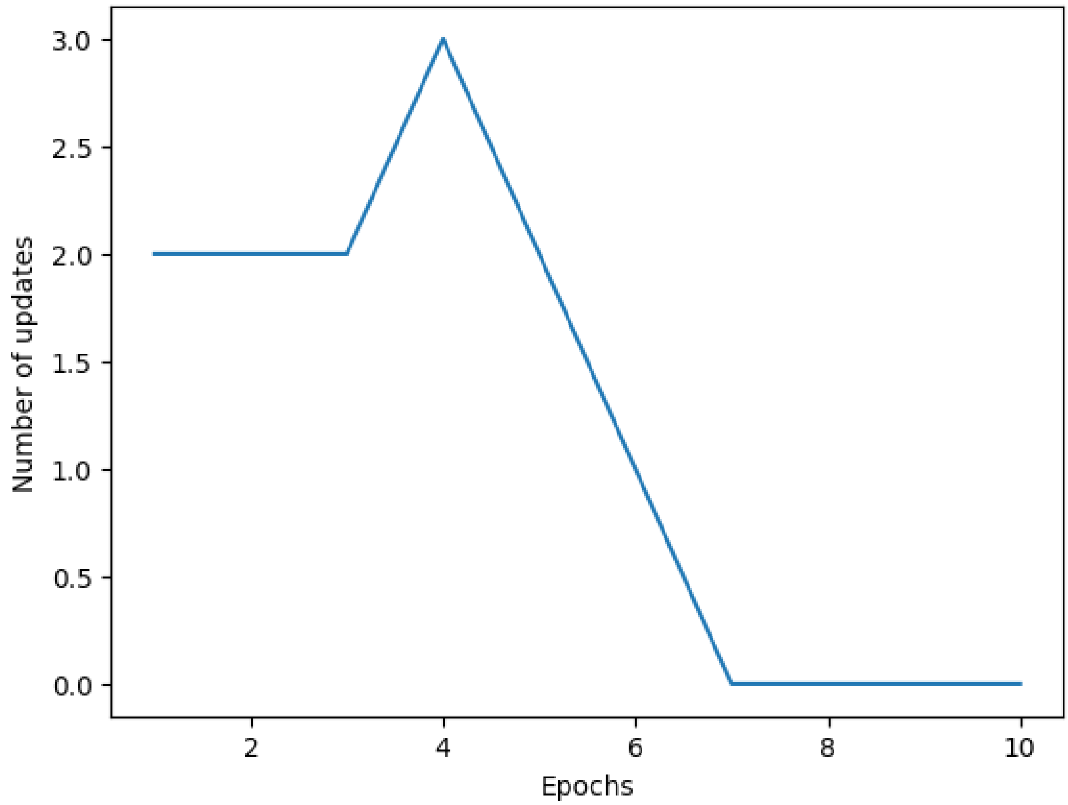
[17]:
```
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
```

[17]:
```
<__main__.Perceptron at 0x785bbec58b50>
```

1.7 Langkah 6 - Visualisasi Nilai Error Per Epoch

[18]:
```
plt.plot(range(1, len(ppn.errors_)+1), ppn.errors_)
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
```

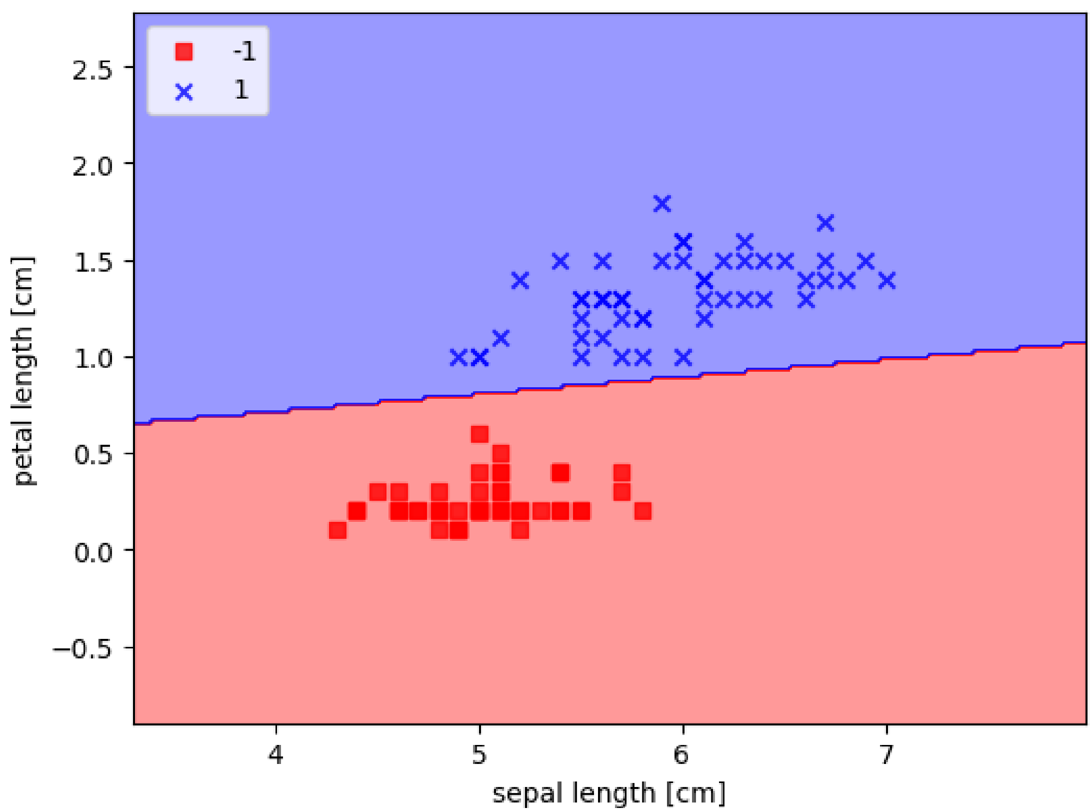
```
plt.show()
```



1.8 Langkah 7 - Visualiasasi Decision Boundary

[24]:
```
# buat fungsi untuk plot decision region

from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('r', 'b', 'g', 'k', 'grey')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision regions by creating a pair of grid arrays xx1 and xx2␣
 ↪via meshgrid function in Numpy
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution), np.
 ↪arange(x2_min, x2_max, resolution))
```

```
    # use predict method to predict the class labels z of the grid points
    Z = classifier.predict(np.array([xx1.ravel(),xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)

    # draw the contour using matplotlib
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # plot class samples
    for i, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y==cl, 0], y=X[y==cl, 1], alpha=0.8, color=cmap(i),␣
 ↪marker=markers[i], label=cl)

    plt.xlabel('sepal length [cm]')
    plt.ylabel('petal length [cm]')
    plt.legend(loc='upper left')
    plt.show()

# Plot decision boundary
plot_decision_regions(X, y, classifier=ppn)
```



2 Praktikum 2

Klasifikasi Berita dengan Perceptron

2.1 Deskripsi

Dalam kasus ini, Anda akan melakukan klasifiaksi berita berdasarkan 3 kategori, yaitu **Sport Hockey**, **Sport Baseball**, dan **Otomotif**. Proses klasifikasi akan menggunakan model Perceptron.

2.2 Langkah 1 - Import Library

[25]:
```python
from sklearn.datasets import fetch_20newsgroups # download dataset
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Perceptron
from sklearn.metrics import f1_score, classification_report
```

2.3 Langkah 2 - Pilih Label dan Split Data

[26]:
```python
categories = ['rec.sport.hockey', 'rec.sport.baseball', 'rec.autos']
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories,␣
 ↪remove=('headers', 'footers', 'quotes'))
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories,␣
 ↪remove=('headers', 'footers', 'quotes'))
```

2.4 Langkah 3 - Ekstrak Fitur dan Buat Model Perceptron

[27]:
```python
# Ekstrak Fitur
vectorizer = TfidfVectorizer()

# Fit fitur
X_train = vectorizer.fit_transform(newsgroups_train.data)
X_test = vectorizer.transform(newsgroups_test.data)

# Fit Model
clf = Perceptron(random_state=11)
clf.fit(X_train, newsgroups_train.target)

# Prediksi
predictions = clf.predict(X_test)
print(classification_report(newsgroups_test.target, predictions))
```

```
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       396
```

```
           1       0.82      0.83      0.83       397
           2       0.88      0.87      0.87       399

    accuracy                           0.86      1192
   macro avg       0.86      0.86      0.86      1192
weighted avg       0.86      0.86      0.86      1192
```

2.5 Penjelasan

Dataset yang digunakan pada kode program diatas adalah `20newsgroup` yang terdiri dari sekitar 20.000 dokumen. Scikit-learn bahkan menyediakan fungsi yang memberikan kemudahan untuk mengunduh dan membaca kumpulan dataset dengan menggunakan sklearn.datasets. pada kode program diatas Perceptron mampu melakukan klasifikasi multikelas; strategi yang digunakan adalah one-versus-all untuk melakukan pelatihan untuk setiap kelas dalam data training. Dokumen teks memerlukan ekstraksi fitur salah satunya adalah bobot tf-idf pada kodeprogram diatas digunakan `tfidf-vectorizer`.

3 Praktikum 3

Nilai Logika XOR dengan MLP

3.1 Deskripsi

Pada kasus sederhana ini, Anda akan menggunakan MLP untuk mendapatkan nilai biner yang dioperasikan dengan logika XOR. Perlu diingat bahwa nilai XOR berbeda dengan OR, XOR hanya akan bernilai benar jika salah satu nilai yang benar, bukan keduanya atau tidak sama sekali.

3.2 Langkah 1 - Import Library

[28]:
```
from sklearn.neural_network import MLPClassifier
```

3.3 Langah 2 - Buat Data

[29]:
```
y = [0, 1, 1, 0] # label
X = [[0, 0], [0, 1], [1, 0], [1, 1]] # data
```

3.4 Langkah 3 - Fit Model

[30]:
```
# Fit model
clf = MLPClassifier(solver='lbfgs', activation='logistic',␣
 ↪hidden_layer_sizes=(2,), max_iter=100, random_state=20)
clf.fit(X, y)
```

[30]:
```
MLPClassifier(activation='logistic', hidden_layer_sizes=(2,), max_iter=100,
              random_state=20, solver='lbfgs')
```

3.5 Langkah 4 - Prediksi

[31]:
```python
pred = clf.predict(X)
print('Accuracy: %s' % clf.score(X, y))
for i,p in enumerate(pred[:10]):
    print('True: %s, Predicted: %s' % (y[i], p))
```

```
Accuracy: 1.0
True: 0, Predicted: 0
True: 1, Predicted: 1
True: 1, Predicted: 1
True: 0, Predicted: 0
```

4 Praktikum 4

Klasifikasi dengan ANN

4.1 Deskripsi

Pada praktikum kali ini, Anda diminta untuk membuat model ANN untuk mengklasifikasi potensi seorang customer akan meninggalkan perusahaan Anda atau tidak. Istirlah populer dari fenomena ini disebut sebagai 'churn'. Tingkat churn yang tinggi (chrun rate) akan berdampak tidak baik bagi perusahaan.

4.2 Dataset

Churn_Modelling.csv

Perhatian! Pada praktikum ini, Anda akan menggunakan library `tensorflow` dari google. Oleh karena itu, Anda diharuskan untuk menginstal `tensorflow` terlebih dahulu.

Anda juga perlu menyesuaikan instalasi tensorflow yang Anda gunakan pada komputer lokal, apakah komputasi pada,

1. CPU
2. GPU (GPU support CUDA)
3. Apple Silicon (M1/M2)

Panduan instalasi,

[https://www.tensorflow.org/install

https://developer.apple.com/metal/tensorflow-plugin/

https://caffeinedev.medium.com/how-to-install-tensorflow-on-m1-mac-8e9b91d93706]

4.3 Pra Pengolahan Data

4.3.1 Langkah 1 - Import Library

[32]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
```

4.3.2 Langkah 2 - Load Data

[35]:
```
dataset = pd.read_csv('/content/drive/MyDrive/Elis-3C/SMT 5/ML/Jobsheet 9/
↪dataset/Churn_Modelling.csv')
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values

print(X)
```

```
[[619 'France' 'Female' … 1 1 101348.88]
 [608 'Spain' 'Female' … 0 1 112542.58]
 [502 'France' 'Female' … 1 0 113931.57]
 …
 [709 'France' 'Female' … 0 1 42085.58]
 [772 'Germany' 'Male' … 1 0 92888.52]
 [792 'France' 'Female' … 1 0 38190.78]]
```

4.3.3 Langkah 3 - Encoding Data Kategorikal

[37]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])

print(X)
```

```
[[619 'France' 0 … 1 1 101348.88]
 [608 'Spain' 0 … 0 1 112542.58]
 [502 'France' 0 … 1 0 113931.57]
 …
 [709 'France' 0 … 0 1 42085.58]
 [772 'Germany' 1 … 1 0 92888.52]
 [792 'France' 0 … 1 0 38190.78]]
```

4.3.4 Langkah 4 - Encoding Kolom "Geography" dengan One Hot Encoder

[38]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])],␣
↪remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

```
print(X)
```

```
[[1.0 0.0 0.0 … 1 1 101348.88]
 [0.0 0.0 1.0 … 0 1 112542.58]
 [1.0 0.0 0.0 … 1 0 113931.57]
 …
 [1.0 0.0 0.0 … 0 1 42085.58]
 [0.0 1.0 0.0 … 1 0 92888.52]
 [1.0 0.0 0.0 … 1 0 38190.78]]
```

4.3.5 Langkah 5 - Split Data

[40]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

4.3.6 Langkah 6 - Scaling Fitur

[41]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

4.4 Membuat Model ANN

4.4.1 Langkah 1 - Inisiasi Model ANN

[42]:
```
ann = tf.keras.models.Sequential()
```

4.4.2 Langkah 2 - Membuat Input Layer dan Hidden Layer Pertama

[43]:
```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

4.4.3 Langkah 3 - Membuat Hidden Layer Kedua

[44]:
```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

4.4.4 Langkah 4 - Membuat Output Layer

[45]:
```
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

4.5 Training Model

4.5.1 Langkah 1 - Compile Model (Menyatukan Arsitektur) ANN

[46]:
```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =␣
↪['accuracy'])
```

4.5.2 Langkah 2 - Fitting Model

[47]:
```
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
250/250              2s 2ms/step -
accuracy: 0.8019 - loss: 0.5477
Epoch 2/100
250/250              1s 2ms/step -
accuracy: 0.7985 - loss: 0.4711
Epoch 3/100
250/250              1s 2ms/step -
accuracy: 0.7939 - loss: 0.4522
Epoch 4/100
250/250              1s 2ms/step -
accuracy: 0.7951 - loss: 0.4420
Epoch 5/100
250/250              1s 2ms/step -
accuracy: 0.8023 - loss: 0.4308
Epoch 6/100
250/250              1s 2ms/step -
accuracy: 0.7986 - loss: 0.4332
Epoch 7/100
250/250              1s 1ms/step -
accuracy: 0.8121 - loss: 0.4136
Epoch 8/100
250/250              1s 1ms/step -
accuracy: 0.8055 - loss: 0.4268
Epoch 9/100
250/250              1s 1ms/step -
accuracy: 0.8229 - loss: 0.4100
Epoch 10/100
250/250              1s 1ms/step -
accuracy: 0.8152 - loss: 0.4289
Epoch 11/100
250/250              1s 1ms/step -
accuracy: 0.8206 - loss: 0.4232
Epoch 12/100
250/250              1s 1ms/step -
accuracy: 0.8339 - loss: 0.4077
Epoch 13/100
```

```
250/250              1s 1ms/step -
accuracy: 0.8236 - loss: 0.4105
Epoch 14/100
250/250              1s 1ms/step -
accuracy: 0.8242 - loss: 0.4117
Epoch 15/100
250/250              1s 1ms/step -
accuracy: 0.8324 - loss: 0.4001
Epoch 16/100
250/250              1s 1ms/step -
accuracy: 0.8319 - loss: 0.4089
Epoch 17/100
250/250              1s 1ms/step -
accuracy: 0.8360 - loss: 0.3976
Epoch 18/100
250/250              1s 1ms/step -
accuracy: 0.8347 - loss: 0.3969
Epoch 19/100
250/250              1s 1ms/step -
accuracy: 0.8349 - loss: 0.3995
Epoch 20/100
250/250              1s 1ms/step -
accuracy: 0.8337 - loss: 0.3951
Epoch 21/100
250/250              1s 1ms/step -
accuracy: 0.8401 - loss: 0.3887
Epoch 22/100
250/250              1s 1ms/step -
accuracy: 0.8411 - loss: 0.3843
Epoch 23/100
250/250              1s 2ms/step -
accuracy: 0.8435 - loss: 0.3773
Epoch 24/100
250/250              1s 2ms/step -
accuracy: 0.8421 - loss: 0.3760
Epoch 25/100
250/250              1s 2ms/step -
accuracy: 0.8419 - loss: 0.3744
Epoch 26/100
250/250              1s 2ms/step -
accuracy: 0.8452 - loss: 0.3661
Epoch 27/100
250/250              1s 2ms/step -
accuracy: 0.8430 - loss: 0.3751
Epoch 28/100
250/250              1s 1ms/step -
accuracy: 0.8482 - loss: 0.3598
Epoch 29/100
```

```
250/250              1s 1ms/step -
accuracy: 0.8594 - loss: 0.3494
Epoch 30/100
250/250              1s 1ms/step -
accuracy: 0.8474 - loss: 0.3646
Epoch 31/100
250/250              1s 1ms/step -
accuracy: 0.8468 - loss: 0.3670
Epoch 32/100
250/250              1s 1ms/step -
accuracy: 0.8568 - loss: 0.3491
Epoch 33/100
250/250              1s 1ms/step -
accuracy: 0.8557 - loss: 0.3488
Epoch 34/100
250/250              1s 1ms/step -
accuracy: 0.8580 - loss: 0.3453
Epoch 35/100
250/250              1s 1ms/step -
accuracy: 0.8647 - loss: 0.3415
Epoch 36/100
250/250              1s 1ms/step -
accuracy: 0.8646 - loss: 0.3357
Epoch 37/100
250/250              1s 1ms/step -
accuracy: 0.8506 - loss: 0.3560
Epoch 38/100
250/250              1s 1ms/step -
accuracy: 0.8550 - loss: 0.3439
Epoch 39/100
250/250              1s 1ms/step -
accuracy: 0.8628 - loss: 0.3349
Epoch 40/100
250/250              1s 1ms/step -
accuracy: 0.8575 - loss: 0.3393
Epoch 41/100
250/250              1s 1ms/step -
accuracy: 0.8651 - loss: 0.3389
Epoch 42/100
250/250              1s 1ms/step -
accuracy: 0.8609 - loss: 0.3376
Epoch 43/100
250/250              0s 1ms/step -
accuracy: 0.8621 - loss: 0.3348
Epoch 44/100
250/250              0s 1ms/step -
accuracy: 0.8619 - loss: 0.3375
Epoch 45/100
```

```
250/250              1s 2ms/step -
accuracy: 0.8661 - loss: 0.3260
Epoch 46/100
250/250              1s 2ms/step -
accuracy: 0.8578 - loss: 0.3370
Epoch 47/100
250/250              1s 2ms/step -
accuracy: 0.8622 - loss: 0.3335
Epoch 48/100
250/250              1s 2ms/step -
accuracy: 0.8700 - loss: 0.3239
Epoch 49/100
250/250              1s 2ms/step -
accuracy: 0.8624 - loss: 0.3340
Epoch 50/100
250/250              1s 2ms/step -
accuracy: 0.8568 - loss: 0.3365
Epoch 51/100
250/250              1s 2ms/step -
accuracy: 0.8527 - loss: 0.3418
Epoch 52/100
250/250              1s 2ms/step -
accuracy: 0.8673 - loss: 0.3296
Epoch 53/100
250/250              0s 1ms/step -
accuracy: 0.8685 - loss: 0.3220
Epoch 54/100
250/250              1s 1ms/step -
accuracy: 0.8612 - loss: 0.3305
Epoch 55/100
250/250              1s 1ms/step -
accuracy: 0.8641 - loss: 0.3311
Epoch 56/100
250/250              0s 1ms/step -
accuracy: 0.8589 - loss: 0.3428
Epoch 57/100
250/250              1s 1ms/step -
accuracy: 0.8666 - loss: 0.3321
Epoch 58/100
250/250              1s 1ms/step -
accuracy: 0.8663 - loss: 0.3273
Epoch 59/100
250/250              0s 1ms/step -
accuracy: 0.8617 - loss: 0.3347
Epoch 60/100
250/250              1s 1ms/step -
accuracy: 0.8674 - loss: 0.3232
Epoch 61/100
```

```
250/250              0s 1ms/step -
accuracy: 0.8669 - loss: 0.3346
Epoch 62/100
250/250              1s 1ms/step -
accuracy: 0.8684 - loss: 0.3246
Epoch 63/100
250/250              1s 1ms/step -
accuracy: 0.8580 - loss: 0.3366
Epoch 64/100
250/250              1s 1ms/step -
accuracy: 0.8602 - loss: 0.3384
Epoch 65/100
250/250              1s 1ms/step -
accuracy: 0.8672 - loss: 0.3286
Epoch 66/100
250/250              1s 1ms/step -
accuracy: 0.8653 - loss: 0.3341
Epoch 67/100
250/250              1s 1ms/step -
accuracy: 0.8634 - loss: 0.3293
Epoch 68/100
250/250              1s 1ms/step -
accuracy: 0.8640 - loss: 0.3283
Epoch 69/100
250/250              1s 1ms/step -
accuracy: 0.8607 - loss: 0.3356
Epoch 70/100
250/250              1s 2ms/step -
accuracy: 0.8708 - loss: 0.3229
Epoch 71/100
250/250              1s 2ms/step -
accuracy: 0.8598 - loss: 0.3412
Epoch 72/100
250/250              1s 2ms/step -
accuracy: 0.8563 - loss: 0.3411
Epoch 73/100
250/250              1s 2ms/step -
accuracy: 0.8634 - loss: 0.3444
Epoch 74/100
250/250              1s 2ms/step -
accuracy: 0.8633 - loss: 0.3318
Epoch 75/100
250/250              1s 2ms/step -
accuracy: 0.8645 - loss: 0.3324
Epoch 76/100
250/250              1s 2ms/step -
accuracy: 0.8629 - loss: 0.3371
Epoch 77/100
```

```
250/250              1s 2ms/step -
accuracy: 0.8689 - loss: 0.3136
Epoch 78/100
250/250              1s 1ms/step -
accuracy: 0.8650 - loss: 0.3311
Epoch 79/100
250/250              0s 1ms/step -
accuracy: 0.8660 - loss: 0.3281
Epoch 80/100
250/250              1s 1ms/step -
accuracy: 0.8617 - loss: 0.3390
Epoch 81/100
250/250              1s 1ms/step -
accuracy: 0.8660 - loss: 0.3356
Epoch 82/100
250/250              0s 1ms/step -
accuracy: 0.8655 - loss: 0.3305
Epoch 83/100
250/250              1s 1ms/step -
accuracy: 0.8678 - loss: 0.3229
Epoch 84/100
250/250              1s 1ms/step -
accuracy: 0.8577 - loss: 0.3370
Epoch 85/100
250/250              1s 1ms/step -
accuracy: 0.8688 - loss: 0.3260
Epoch 86/100
250/250              0s 1ms/step -
accuracy: 0.8675 - loss: 0.3246
Epoch 87/100
250/250              0s 1ms/step -
accuracy: 0.8601 - loss: 0.3343
Epoch 88/100
250/250              1s 1ms/step -
accuracy: 0.8666 - loss: 0.3336
Epoch 89/100
250/250              1s 1ms/step -
accuracy: 0.8617 - loss: 0.3376
Epoch 90/100
250/250              0s 1ms/step -
accuracy: 0.8600 - loss: 0.3362
Epoch 91/100
250/250              1s 1ms/step -
accuracy: 0.8624 - loss: 0.3379
Epoch 92/100
250/250              1s 1ms/step -
accuracy: 0.8580 - loss: 0.3444
Epoch 93/100
```

```
250/250              1s 1ms/step -
accuracy: 0.8678 - loss: 0.3258
Epoch 94/100
250/250              0s 1ms/step -
accuracy: 0.8636 - loss: 0.3307
Epoch 95/100
250/250              0s 1ms/step -
accuracy: 0.8680 - loss: 0.3231
Epoch 96/100
250/250              0s 2ms/step -
accuracy: 0.8696 - loss: 0.3262
Epoch 97/100
250/250              1s 2ms/step -
accuracy: 0.8665 - loss: 0.3244
Epoch 98/100
250/250              1s 2ms/step -
accuracy: 0.8682 - loss: 0.3303
Epoch 99/100
250/250              1s 2ms/step -
accuracy: 0.8730 - loss: 0.3195
Epoch 100/100
250/250              1s 2ms/step -
accuracy: 0.8648 - loss: 0.3304
```

[47]:
```
<keras.src.callbacks.history.History at 0x785b66036440>
```

4.6 Membuat Prediksi

Diberikan informasi sebagai berikut,

- Geography: France
- Credit Score: 600
- Gender: Male
- Age: 40 years old
- Tenure: 3 years
- Balance: $ 60000
- Number of Products: 2
- Does this customer have a credit card ? Yes
- Is this customer an Active Member: Yes
- Estimated Salary: $ 50000

Apakah customer tersebut perlu dipertahankan?

4.6.1 Modelkan Data Baru dan Buat Prediksi

[48]:
```
print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1,␣
↪50000]])) > 0.5)
```

```
1/1              0s 224ms/step
[[False]]
```

Apakah hasilnya False?

Jawab: benar, hasil yang ditampilkan adalah false

4.6.2 Prediksi Dengan Data Testing

[49]:
```
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.
 ↪reshape(len(y_test),1)),1))
```

```
63/63               0s 1ms/step
[[0 0]
 [0 1]
 [0 0]
 …
 [0 0]
 [0 0]
 [0 0]]
```

4.6.3 Cek Akurasi dan Confusion Matrix

[50]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[1510   85]
 [ 192  213]]
```

[50]: 0.8615

Hasil (bisa jadi berbeda) setiap kali kode dijalankan karena adanya variasi dalam pembagian data, inisialisasi acak pada algoritma, preprocessing data, dan parameter model yang digunakan.