LAPORAN CASE BASED 1

SUPERVISED LEARNING

Mata Kuliah: Pembelajaran Mesin

Dosen Pengampu: Bedy Purnama, S.Si, M.T, Doctor of Philosophy

Kode Dosen: BDP

Disusun Oleh :
Johannes Raphael Nandaputra (1301204243)
IF-44-01

PROGRAM STUDI INFORMATIKA

FAKULTAS INFORMATIKA

2022

Tugas ini dikerjakan dengan cara yang

tidak melanggar aturan perkuliahan dan kode etik akademisi

Kata Pengantar

Puji syukur penulis ucapkan kepada Tuhan Yang Maha Esa atas berkat-Nya penulis dapat menyelesaikan Laporan Case Based 1 ini sebagai hasil dari pembelajaran supervised learning dimana penulis dapat menjelaskan, mengimplementasikan, menganalisis, dan mendesain teknik pembelajaran mesin supervised learning.

Dalam pengerjaan laporan ini, penulis mengerjakan tugas ini dengan cara yang tidak melanggar aturan perkuliahan dan kode etik akademisi. Meski begitu, penulis merasa masih banyak kekurangan-kekurangan baik pada penulisan maupun materi, mengingat kememampuan yang dimiliki penulis. Untuk itu, kritik dan saran dari semua pihak sangat penulis harapkan demi penyempurnaan laporan ini.

Bandung, 31 Oktober 2022

Johanes Raphael Nandaputra

A. Ikthisar Kumpulan Data yang Dipilih

Data yang digunakan dalam pengerjaan tugas ini adalah data *audit_risk.csv,* dimana data ini merupakan data non-rahasia satu tahun yang lengkap pada tahun 2015 hingga 2016 dari perusahaan yang dikumpulkan dari Kantor Auditor India untuk membangun prediktor untuk mengklasifikasikan perusahaan yang mencurigakan. Berikut informasi data yang digunakan:

- Lima data teratas:

	Sector_score	LOCATION_ID	PARA_A	Score_A	Risk_A	PARA_B	Score_B	Risk_B	\
0	3.89	23	4.18	0.6	2.508	2.50	0.2	0.500	
1	3.89	6	0.00	0.2	0.000	4.83	0.2	0.966	
2	3.89	6	0.51	0.2	0.102	0.23	0.2	0.046	
3	3.89	6	0.00	0.2	0.000	10.80	0.6	6.480	
4	3.89	6	0.00	0.2	0.000	0.08	0.2	0.016	

	TOTAL	numbers	Score_B.1	Risk_C	Money_Value	Score_MV	Risk_D	\
0	6.68	5.0	0.2	1.0	3.38	0.2	0.676	
1	4.83	5.0	0.2	1.0	0.94	0.2	0.188	
2	0.74	5.0	0.2	1.0	0.00	0.2	0.000	
3	10.80	6.0	0.6	3.6	11.75	0.6	7.050	
4	0.08	5.0	0.2	1.0	0.00	0.2	0.000	

	District_Loss	PROB	RiSk_E	History	Prob	Risk_F	Score	Inherent_Risk	\
0	2	0.2	0.4	0	0.2	0.0	2.4	8.574	
1	2	0.2	0.4	0	0.2	0.0	2.0	2.554	
2	2	0.2	0.4	0	0.2	0.0	2.0	1.548	
3	2	0.2	0.4	0	0.2	0.0	4.4	17.530	
4	2	0.2	0.4	0	0.2	0.0	2.0	1.416	

	CONTROL_RISK	Detection_Risk	Audit_Risk	Risk
0	0.4	0.5	1.7148	1
1	0.4	0.5	0.5108	0
2	0.4	0.5	0.3096	0
3	0.4	0.5	3.5060	1
4	0.4	0.5	0.2832	0

- Informasi dataset:

```
RangeIndex: 776 entries, 0 to 775
Data columns (total 27 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sector_score    776 non-null    float64
 1   LOCATION_ID     776 non-null    object
 2   PARA_A          776 non-null    float64
 3   Score_A         776 non-null    float64
 4   Risk_A          776 non-null    float64
 5   PARA_B          776 non-null    float64
 6   Score_B         776 non-null    float64
 7   Risk_B          776 non-null    float64
 8   TOTAL           776 non-null    float64
 9   numbers         776 non-null    float64
 10  Score_B.1       776 non-null    float64
 11  Risk_C          776 non-null    float64
 12  Money_Value     775 non-null    float64
 13  Score_MV        776 non-null    float64
 14  Risk_D          776 non-null    float64
 15  District_Loss   776 non-null    int64
 16  PROB            776 non-null    float64
 17  RiSk_E          776 non-null    float64
 18  History         776 non-null    int64
 19  Prob            776 non-null    float64
 20  Risk_F          776 non-null    float64
 21  Score           776 non-null    float64
 22  Inherent_Risk   776 non-null    float64
 23  CONTROL_RISK    776 non-null    float64
 24  Detection_Risk  776 non-null    float64
 25  Audit_Risk      776 non-null    float64
 26  Risk            776 non-null    int64
dtypes: float64(23), int64(3), object(1)
memory usage: 163.8+ KB
None
```

- Statistik deskriptif:

	Sector_score	PARA_A	Score_A	Risk_A	PARA_B
count	776.000000	776.000000	776.000000	776.000000	776.000000
mean	20.184536	2.450194	0.351289	1.351029	10.799988
std	24.319017	5.678870	0.174055	3.440447	50.083624
min	1.850000	0.000000	0.200000	0.000000	0.000000
25%	2.370000	0.210000	0.200000	0.042000	0.000000
50%	3.890000	0.875000	0.200000	0.175000	0.405000
75%	55.570000	2.480000	0.600000	1.488000	4.160000
max	59.850000	85.000000	0.600000	51.000000	1264.630000

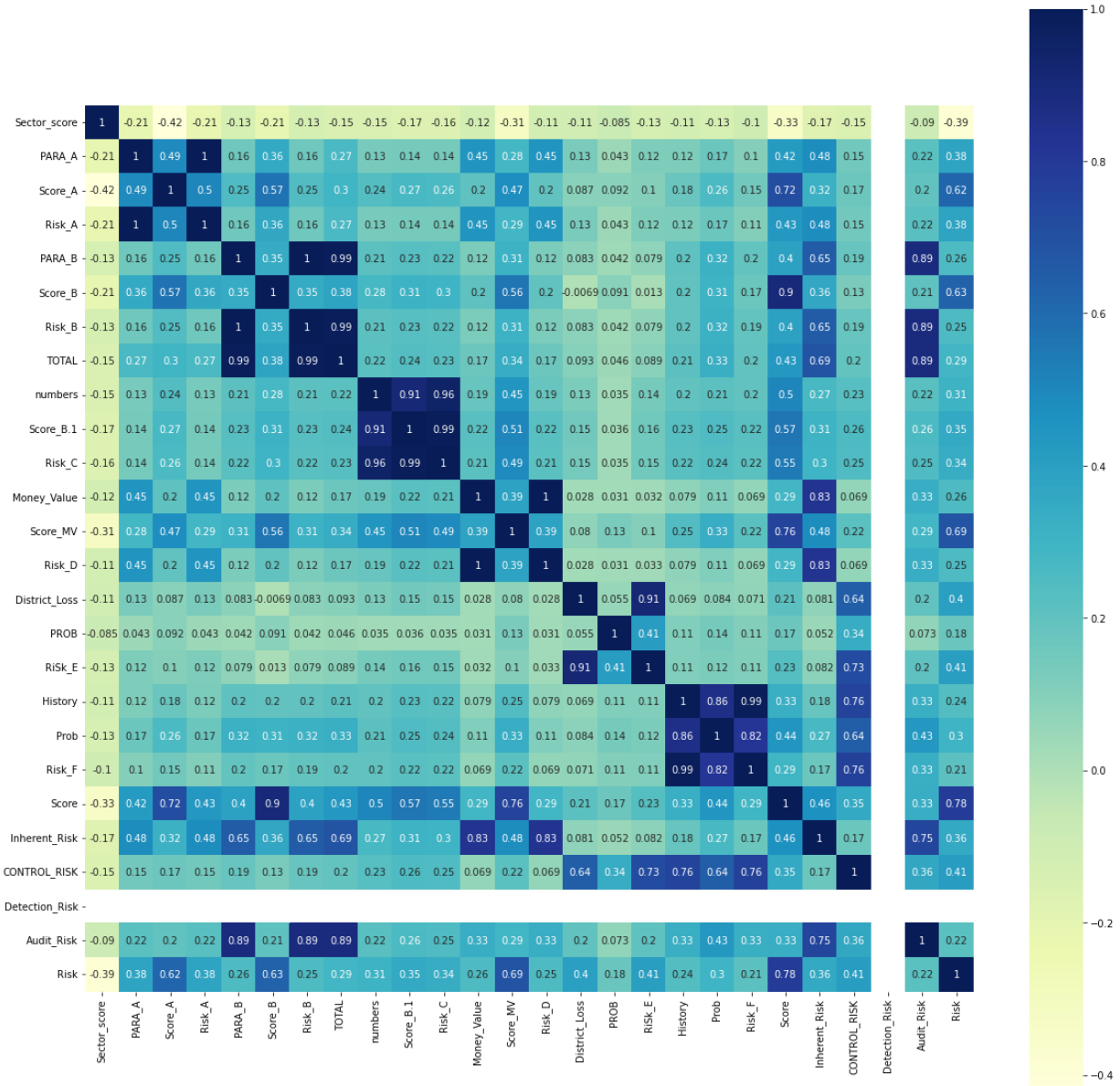
	Score_B	Risk_B	TOTAL	numbers	Score_B.1
count	776.000000	776.000000	776.000000	776.000000	776.000000
mean	0.313144	6.334008	13.218481	5.067655	0.223711
std	0.169804	30.072845	51.312829	0.264449	0.080352
min	0.200000	0.000000	0.000000	5.000000	0.200000
25%	0.200000	0.000000	0.537500	5.000000	0.200000
50%	0.200000	0.081000	1.370000	5.000000	0.200000
75%	0.400000	1.840500	7.707500	5.000000	0.200000
max	0.600000	758.778000	1268.910000	9.000000	0.600000

	Risk_C	Money_Value	Score_MV	Risk_D	District_Loss
count	776.000000	775.000000	776.000000	776.000000	776.000000
mean	1.152964	14.137631	0.290979	8.265434	2.505155
std	0.537417	66.606519	0.159745	39.970849	1.228678
min	1.000000	0.000000	0.200000	0.000000	2.000000
25%	1.000000	0.000000	0.200000	0.000000	2.000000
50%	1.000000	0.090000	0.200000	0.018000	2.000000
75%	1.000000	5.595000	0.400000	2.235000	2.000000
max	5.400000	935.030000	0.600000	561.018000	6.000000

	PROB	RiSk_E	History	Prob	Risk_F	Score
count	776.000000	776.000000	776.000000	776.000000	776.000000	776.000000
mean	0.206186	0.519072	0.104381	0.216753	0.053608	2.702577
std	0.037508	0.290312	0.531031	0.067987	0.305835	0.858923
min	0.200000	0.400000	0.000000	0.200000	0.000000	2.000000
25%	0.200000	0.400000	0.000000	0.200000	0.000000	2.000000
50%	0.200000	0.400000	0.000000	0.200000	0.000000	2.400000
75%	0.200000	0.400000	0.000000	0.200000	0.000000	3.250000
max	0.600000	2.400000	9.000000	0.600000	5.400000	5.200000

	Inherent_Risk	CONTROL_RISK	Detection_Risk	Audit_Risk	Risk
count	776.000000	776.000000	776.0	776.000000	776.000000
mean	17.680612	0.572680	0.5	7.168158	0.393041
std	54.740244	0.444581	0.0	38.667494	0.488741
min	1.400000	0.400000	0.5	0.280000	0.000000
25%	1.583500	0.400000	0.5	0.316700	0.000000
50%	2.214000	0.400000	0.5	0.555600	0.000000
75%	10.663500	0.400000	0.5	3.249900	1.000000
max	801.262000	5.800000	0.5	961.514400	1.000000

- Korelasi:



B. Ringkasan Pra-Pemrosesan Data yang Diusulkan

1. Handling Missing Value

Pada *handling missing value*, ditemukan 1 datum dengan tipe data NaN pada kolom "Money_Value". Langkah yang dilakukan adalah menghapus/mendrop baris dimana datum tersebut berada supaya saat kita menerapkan algoritma yang dipilih untuk membuat model prediksi tidak terjadi error saat prosesnya,seperti gambar dan kode yang ditunjukkan berikut:

```
Sector_score      0
LOCATION_ID       0
PARA_A            0
Score_A           0
Risk_A            0
PARA_B            0
Score_B           0
Risk_B            0
TOTAL             0
numbers           0
Score_B.1         0
Risk_C            0
Money_Value       1
Score_MV          0
Risk_D            0
District_Loss     0
PROB              0
RiSk_E            0
History           0
Prob              0
Risk_F            0
Score             0
Inherent_Risk     0
CONTROL_RISK      0
Detection_Risk    0
Audit_Risk        0
Risk              0
```

```
# menghilangkan row dari missing value
dataset = dataset.dropna()
```

2. Redundant Data

Pada *redundant data* akan didrop tiap baris data yang *redundant*. Meskipun *redundant data* tidak terlalu berpengaruh jika dihilangkan, tetapi jika banyak data yang *redundant* tentunya berpengaruh dengan dimensi data dan kecepatan dalam proses prediksi modelnya juga. Didapat 13 baris data yang redundant seperti yang ditunjukkan gambar dan kode sebagai berikut:

	Sector_score	LOCATION_ID	PARA_A	Score_A	Risk_A	PARA_B	Score_B	Risk_B	TOTAL	numbers	Score_B.1	Risk_C	Money_Value	Score_MV	Risk_D	District_Loss	PROB	RISK_E	History	Prob	Risk_F	Score	Inherent_Risk	CONTROL_RISK	Detection_Risk	Audit_Risk	Risk
265	2.72	15	0.00	0.2	0.000	0.0	0.2	0.00	0.00	5.0	0.2	1.0	0.0	0.2	0.0	2	0.2	0.4	0	0.2	0.0	2.0	1.400	0.4	0.5	0.2800	0
474	1.85	16	0.00	0.2	0.000	0.0	0.2	0.00	0.00	5.0	0.2	1.0	0.0	0.2	0.0	2	0.2	0.4	0	0.2	0.0	2.0	1.400	0.4	0.5	0.2800	0
600	55.57	8	0.00	0.2	0.000	0.0	0.2	0.00	0.00	5.0	0.2	1.0	0.0	0.2	0.0	2	0.2	0.4	0	0.2	0.0	2.0	1.400	0.4	0.5	0.2800	0
659	55.57	5	0.00	0.2	0.000	0.0	0.2	0.00	0.00	5.0	0.2	1.0	0.0	0.2	0.0	2	0.2	0.4	0	0.2	0.0	2.0	1.400	0.4	0.5	0.2800	0
679	55.57	9	0.00	0.2	0.000	0.0	0.2	0.00	0.00	5.0	0.2	1.0	0.0	0.2	0.0	2	0.2	0.4	0	0.2	0.0	2.0	1.400	0.4	0.5	0.2800	0
695	55.57	12	0.00	0.2	0.000	0.0	0.2	0.00	0.00	5.0	0.2	1.0	0.0	0.2	0.0	2	0.2	0.4	0	0.2	0.0	2.0	1.400	0.4	0.5	0.2800	0
696	55.57	12	0.00	0.2	0.000	0.0	0.2	0.00	0.00	5.0	0.2	1.0	0.0	0.2	0.0	2	0.2	0.4	0	0.2	0.0	2.0	1.400	0.4	0.5	0.2800	0
704	55.57	8	0.00	0.2	0.000	0.0	0.2	0.00	0.00	5.0	0.2	1.0	0.0	0.2	0.0	2	0.2	0.4	0	0.2	0.0	2.0	1.400	0.4	0.5	0.2800	0
715	55.57	27	0.00	0.2	0.000	0.0	0.2	0.00	0.00	5.0	0.2	1.0	0.0	0.2	0.0	2	0.2	0.4	0	0.2	0.0	2.0	1.400	0.4	0.5	0.2800	0
716	55.57	2	0.00	0.2	0.000	0.0	0.2	0.00	0.00	5.0	0.2	1.0	0.0	0.2	0.0	6	0.2	1.2	0	0.2	0.0	2.4	2.200	1.2	0.5	1.3200	1
742	55.57	15	0.00	0.2	0.000	0.0	0.2	0.00	0.00	5.0	0.2	1.0	0.0	0.2	0.0	2	0.2	0.4	0	0.2	0.0	2.0	1.400	0.4	0.5	0.2800	0
761	55.57	12	0.00	0.2	0.000	0.0	0.2	0.00	0.00	5.0	0.2	1.0	0.0	0.2	0.0	2	0.2	0.4	0	0.2	0.0	2.0	1.400	0.4	0.5	0.2800	0
771	55.57	9	0.49	0.2	0.098	0.4	0.2	0.08	0.89	5.0	0.2	1.0	0.0	0.2	0.0	2	0.2	0.4	0	0.2	0.0	2.0	1.578	0.4	0.5	0.3156	0

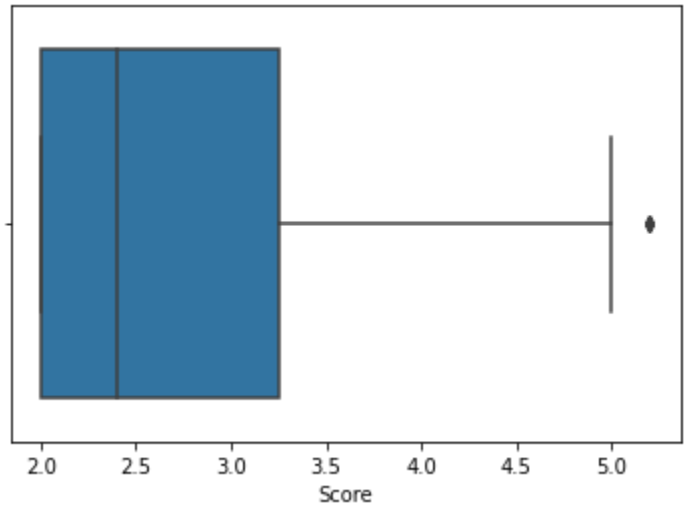
```
# menghilangkan data redundant
dataset = dataset.drop_duplicates()
```

3. Outliers

Berdasarkan korelasi yang telah dicari, terdapat 4 kolom data yang korelasinya tinggi, yaitu:

- Score_A (0.619726)
- Score_B (0.635768)
- Score_MV (0.688367)
- Score (0.785995)

Dari ke-4 kolom diatas, ditemukan *outliers* di kolom Score dengan visualisasi boxplot dan daftar *outliers* pada kolom *Score* sebagai berikut:

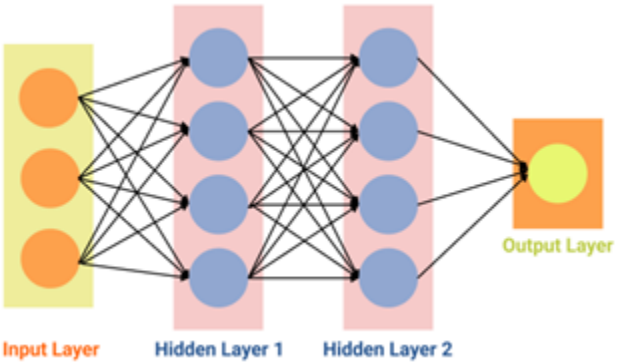



```
93     5.2
190    5.2
241    5.2
495    5.2
Name: Score, dtype: float64
```

```
# mendrop row outliers berada
dataset = dataset.drop([93, 190, 241, 495])
```

C. Menerapkan Algoritma yang Dipilih

Algoritma yang digunakan pada tugas ini adalah algoritma ANN (Artificial Neural Network) yang merupakan bagian dari Supervised Learning dimana kita akan memiliki input serta output yang sesuai yang ada di dalam *dataset*. Tujuan disini adalah mencari cara untuk memetakan input ke output masing-masing dan ANN dapat digunakan untuk memecahkan masalah regresi dan klasifikasi, seperti pada data yang diberikan pada tugas ini. Berikut ilustrasi algoritma ANN yang digunakan:



1. Import Library

```
# import library
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.pyplot as mp
import seaborn as sns
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Diimport 7 library untuk mendukung pra-pemrosesan dan ANN yang akan dilakukan.

2. Import Dataset

```
# import dataset
dataset = pd.read_csv(
    'https://github.com/johanesraphaeln/case-based-1-supervised-learning/blob/main/dataset/audit_risk.csv?raw=true'
    )
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

Dataset yang telah disimpan di github (dalam bentuk link) akan diimport/dibaca dengan bantuan library pandas dan disimpan ke dalam variabel *dataset*.

3. **Split Dataset into Training and Testing Dataset**

```
# split dataset
X = dataset.iloc[:, 2:26].values
Y = dataset.iloc[:, -1].values
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state=0)
```

Menggunakan method iloc dari pandas untuk mengambil value dari kolom yang diinginkan di dataset. Variabel X diisi dengan *independent* variabel dari kolom PARA_A hingga kolom Audit_Risk, lalu variabel Y diisi dengan *dependent* variabel yaitu kolom Risk. Setelah itu, akan displit menggunakan bantuan library sklearn dengan konfigurasi 80% data train dan 20% data test.

4. **Performing Feature Scaling**

```
# feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

Ada kemungkinan bahwa selama pembuatan model, variabel yang memiliki nilai sangat tinggi mendominasi variabel yang memiliki nilai sangat rendah. Karena itu, ada kemungkinan bahwa variabel-variabel dengan nilai rendah tersebut dapat diabaikan oleh mode;, dan karenanya *feature scaling* diperlukan. Saat melakukan *feature scaling*, kita menggunakan teknik *Standardization* dengan bantuan libary sklearn pada class StandardScaler.

5. **Initialize ANN**

```
# init ANN
ann = tf.keras.models.Sequential()
```

Disini kita akan membuat objek ANN dengan bantuan kelas Keras bernama Sequential dari library tensorflow.

6. **Create Hidden Layer**

```
# first hidden layer
ann.add(tf.keras.layers.Dense(
    units=6,
    activation="relu"
))
```

```
# second hidden layer
ann.add(tf.keras.layers.Dense(
    units=6,
    activation="relu"
))
```

Setelah menginisiasi ANN, akan dibuat 2 hidden layer dengan bantuan kelas Dense dari library tensorflow yang menerima 2 input, yaitu:

- units: jumlah neuron di masing lapisan
- activation: fungsi aktivasi yang akan digunakan

Optimal value dari units yang ditemukan berdasarkan pengalaman adalah 6, dan untuk activation akan selalu menggunakan relu (rectified linear unit) sebagai fungsi aktivasi untuk hidden layer.

7. **Create Output Layer**

```
# output layer
ann.add(tf.keras.layers.Dense(
    units=1,
    activation="sigmoid"
))
```

Disini kita akan membuat output layer untuk ANN dengan bantuan kelas Dense dari library tensorflow. Pada units diberi 1 karena pada binary classification dimana kita hanya punya 2 kelas sebagai output (1 dan 0) kita akan mengalokasi hanya 1 neuron ke output layer. Dan pada fungsi aktivasi selalu menggunakan sigmoid untuk masalah binary classification.

8. Compile ANN

```
# compile ANN
ann.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=['accuracy']
)
```

Pada compilling ANN akan menerima input sebagai berikut:

- optimizer: menentukan pengoptimal yang akan digunakan untuk melakukan *stochastic gradient descent*. Optimizer 'adam' dapat digunakan dengan *neural network* manapun.
- loss: menentukan *loss function* yang akan digunakan. Untuk *binary classification* menggunakan *binary_crossentropy.*
- metrics: *performance metrics* mana yang akan digunakan untuk menghitung performa. Disini menggunakan 'accuracy' sebagai *performance metrics*-nya.

9. Fitting ANN

```
# fitting ANN
ann.fit(
    X_train,
    Y_train,
    batch_size = 32,
    epochs = 100
)
```

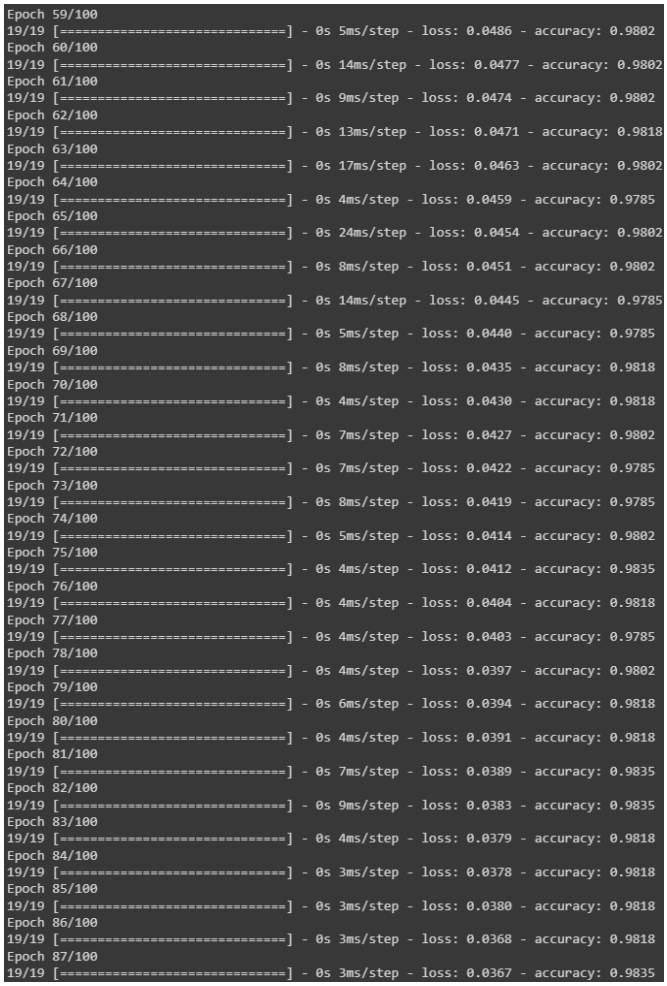
Untuk melatih ANN pada training dataset akan digunakan fit method yang menerima 4 input, yaitu:

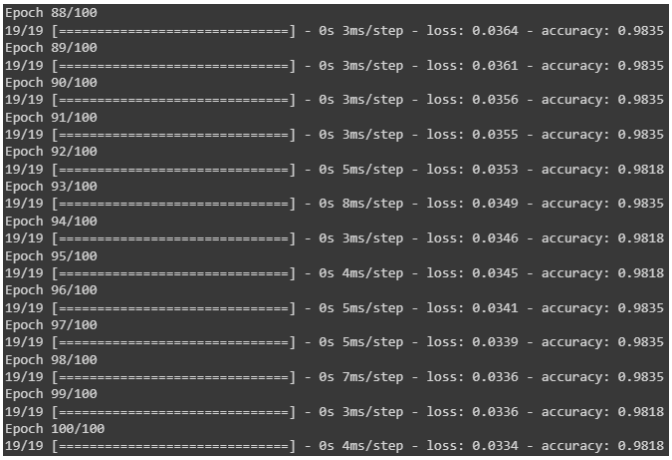
- X_train: matriks fitur untuk dataset train
- Y_train: vektor vairable dependent untuk dataset train
- batch_size: berapa banyak pengamatan yang harus ada dalam batch, umumnya 32 observasi.
- epochs: berapa kali neural networks akan dilatih, optimalnya 100.

D. Evaluasi Hasil

Setelah fitting ANN dijalankan, berikut output yang diberikan dengan keterangan loss dan accuracy sebagai berikut:

```
Epoch 1/100
19/19 [==============================] - 2s 3ms/step - loss: 0.7814 - accuracy: 0.6419
Epoch 2/100
19/19 [==============================] - 0s 4ms/step - loss: 0.7129 - accuracy: 0.6799
Epoch 3/100
19/19 [==============================] - 0s 4ms/step - loss: 0.6742 - accuracy: 0.7756
Epoch 4/100
19/19 [==============================] - 0s 6ms/step - loss: 0.6427 - accuracy: 0.8185
Epoch 5/100
19/19 [==============================] - 0s 5ms/step - loss: 0.6156 - accuracy: 0.8366
Epoch 6/100
19/19 [==============================] - 0s 4ms/step - loss: 0.5900 - accuracy: 0.8581
Epoch 7/100
19/19 [==============================] - 0s 4ms/step - loss: 0.5624 - accuracy: 0.8680
Epoch 8/100
19/19 [==============================] - 0s 3ms/step - loss: 0.5339 - accuracy: 0.8795
Epoch 9/100
19/19 [==============================] - 0s 4ms/step - loss: 0.5020 - accuracy: 0.8845
Epoch 10/100
19/19 [==============================] - 0s 3ms/step - loss: 0.4690 - accuracy: 0.8845
Epoch 11/100
19/19 [==============================] - 0s 4ms/step - loss: 0.4341 - accuracy: 0.8861
Epoch 12/100
19/19 [==============================] - 0s 3ms/step - loss: 0.3981 - accuracy: 0.9224
Epoch 13/100
19/19 [==============================] - 0s 3ms/step - loss: 0.3623 - accuracy: 0.9307
Epoch 14/100
19/19 [==============================] - 0s 3ms/step - loss: 0.3277 - accuracy: 0.9340
Epoch 15/100
19/19 [==============================] - 0s 4ms/step - loss: 0.2937 - accuracy: 0.9455
Epoch 16/100
19/19 [==============================] - 0s 3ms/step - loss: 0.2614 - accuracy: 0.9505
Epoch 17/100
19/19 [==============================] - 0s 4ms/step - loss: 0.2329 - accuracy: 0.9587
Epoch 18/100
19/19 [==============================] - 0s 4ms/step - loss: 0.2074 - accuracy: 0.9653
Epoch 19/100
19/19 [==============================] - 0s 3ms/step - loss: 0.1863 - accuracy: 0.9653
Epoch 20/100
19/19 [==============================] - 0s 4ms/step - loss: 0.1682 - accuracy: 0.9653
Epoch 21/100
19/19 [==============================] - 0s 4ms/step - loss: 0.1533 - accuracy: 0.9670
Epoch 22/100
19/19 [==============================] - 0s 8ms/step - loss: 0.1406 - accuracy: 0.9686
Epoch 23/100
19/19 [==============================] - 0s 10ms/step - loss: 0.1298 - accuracy: 0.9670
Epoch 24/100
19/19 [==============================] - 0s 15ms/step - loss: 0.1209 - accuracy: 0.9703
Epoch 25/100
19/19 [==============================] - 0s 13ms/step - loss: 0.1133 - accuracy: 0.9719
Epoch 26/100
19/19 [==============================] - 0s 10ms/step - loss: 0.1066 - accuracy: 0.9719
Epoch 27/100
19/19 [==============================] - 0s 16ms/step - loss: 0.1013 - accuracy: 0.9719
Epoch 28/100
19/19 [==============================] - 0s 14ms/step - loss: 0.0963 - accuracy: 0.9736
Epoch 29/100
19/19 [==============================] - 0s 11ms/step - loss: 0.0918 - accuracy: 0.9736
Epoch 30/100
19/19 [==============================] - 0s 9ms/step - loss: 0.0883 - accuracy: 0.9736
Epoch 31/100
19/19 [==============================] - 0s 10ms/step - loss: 0.0847 - accuracy: 0.9736
Epoch 32/100
19/19 [==============================] - 0s 13ms/step - loss: 0.0815 - accuracy: 0.9769
Epoch 33/100
19/19 [==============================] - 0s 5ms/step - loss: 0.0790 - accuracy: 0.9752
Epoch 34/100
19/19 [==============================] - 0s 9ms/step - loss: 0.0765 - accuracy: 0.9719
Epoch 35/100
19/19 [==============================] - 0s 5ms/step - loss: 0.0744 - accuracy: 0.9769
Epoch 36/100
19/19 [==============================] - 0s 5ms/step - loss: 0.0723 - accuracy: 0.9769
Epoch 37/100
19/19 [==============================] - 0s 7ms/step - loss: 0.0704 - accuracy: 0.9785
Epoch 38/100
19/19 [==============================] - 0s 13ms/step - loss: 0.0688 - accuracy: 0.9769
Epoch 39/100
19/19 [==============================] - 0s 10ms/step - loss: 0.0673 - accuracy: 0.9785
Epoch 40/100
19/19 [==============================] - 0s 11ms/step - loss: 0.0658 - accuracy: 0.9785
Epoch 41/100
19/19 [==============================] - 0s 12ms/step - loss: 0.0646 - accuracy: 0.9785
Epoch 42/100
19/19 [==============================] - 0s 8ms/step - loss: 0.0630 - accuracy: 0.9785
Epoch 43/100
19/19 [==============================] - 0s 7ms/step - loss: 0.0619 - accuracy: 0.9802
Epoch 44/100
19/19 [==============================] - 0s 6ms/step - loss: 0.0604 - accuracy: 0.9785
Epoch 45/100
19/19 [==============================] - 0s 17ms/step - loss: 0.0588 - accuracy: 0.9785
Epoch 46/100
19/19 [==============================] - 0s 8ms/step - loss: 0.0576 - accuracy: 0.9785
Epoch 47/100
19/19 [==============================] - 0s 7ms/step - loss: 0.0570 - accuracy: 0.9785
Epoch 48/100
19/19 [==============================] - 0s 15ms/step - loss: 0.0561 - accuracy: 0.9785
Epoch 49/100
19/19 [==============================] - 0s 8ms/step - loss: 0.0557 - accuracy: 0.9802
Epoch 50/100
19/19 [==============================] - 0s 14ms/step - loss: 0.0541 - accuracy: 0.9802
Epoch 51/100
19/19 [==============================] - 0s 9ms/step - loss: 0.0534 - accuracy: 0.9802
Epoch 52/100
19/19 [==============================] - 0s 10ms/step - loss: 0.0526 - accuracy: 0.9802
Epoch 53/100
19/19 [==============================] - 0s 6ms/step - loss: 0.0519 - accuracy: 0.9802
Epoch 54/100
19/19 [==============================] - 0s 14ms/step - loss: 0.0513 - accuracy: 0.9802
Epoch 55/100
19/19 [==============================] - 0s 11ms/step - loss: 0.0505 - accuracy: 0.9802
Epoch 56/100
19/19 [==============================] - 0s 8ms/step - loss: 0.0500 - accuracy: 0.9802
Epoch 57/100
19/19 [==============================] - 0s 17ms/step - loss: 0.0496 - accuracy: 0.9802
Epoch 58/100
19/19 [==============================] - 0s 4ms/step - loss: 0.0490 - accuracy: 0.9818
```





Berdasarkan hasil fitting tersebut, bisa dilihat tiap epoch bahwa loss berkurang dan accuracy bertambah. Dan terlihat bahwa final accuracy yang didapat adalah 98.18 yang dimana cukup luar biasa untuk neural network dengan kesederhanaan ini.

Dengan Confusion Matrix yang didapat sebagai berikut:

		Actual Values	
		Positive	Negative
Predicted Values	Positive	91	1
	Negative	4	56

E. Link dan Referensi

Link Laporan:

https://docs.google.com/document/d/1SBfjyQwP3tECx1F93gmLLDIyhQS-UJDuCxKBlNngI00/edit?usp=sharing

Link Slide:

https://www.canva.com/design/DAFRafKVnsU/J0OGtYnR0fUNoXr72Q7Ogw/view?utm_content=DAFRafKVnsU&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Link Video Presentasi:

https://youtu.be/0BtG6XtesHE

Link Code:

https://colab.research.google.com/drive/1lUn4xSF4x7Wz6NaGV9wUgQ4gOZ4HGliy?usp=sharing

Referensi:

Slide Perkuliahan Pembelajaran Mesin

https://www.analyticsvidhya.com/blog/2021/10/implementing-artificial-neural-networkclassification-in-python-from-scratch/#respond

https://medium.com/@bayysp/predictive-modelling-using-ann-with-python-part-3-a0cbdefbec2