hw 5.ipynb

File Edit View Insert Runtime Tools Help

Share

Files

Analyze your files with code written by Gemini Upload

..
sample_data
dataset.csv
my_model.keras

Disk 76.26 GB available

+ Code + Text

RAM Disk

[20]
```
import tensorflow as tf
import keras
import pandas as pd
from keras import layers
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

[21]
```
data = pd.read_csv("dataset.csv")
print(data.shape)
```

```
(400, 11)
```

[22]
```
data.head()
```

	gender	age	hypertension	heart_disease	Marriage	work_type	Living_type	avg_gluco
0	Male	67.0	0	1	Yes	Private	Urban	228.(
1	Male	80.0	0	1	Yes	Private	Rural	105.9
2	Female	49.0	0	0	Yes	Private	Urban	171.2
3	Female	79.0	1	0	Yes	Self-employed	Rural	174.7
4	Male	81.0	0	0	Yes	Private	Urban	186.2

Next steps: Generate code with data View recommended plots New interactive sheet

[23]
```
data.describe()
```

	age	hypertension	heart_disease	avg_glucose	bmi	illness
count	400.00000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	55.26780	0.180000	0.132500	119.391950	29.481750	0.500000
std	22.51279	0.384669	0.339458	54.377459	6.488354	0.500626
min	0.80000	0.000000	0.000000	56.070000	15.600000	0.000000
25%	44.00000	0.000000	0.000000	80.460000	25.575000	0.000000
50%	59.00000	0.000000	0.000000	97.665000	28.600000	0.500000
75%	74.25000	0.000000	0.000000	144.345000	33.025000	1.000000
max	82.00000	1.000000	1.000000	271.740000	48.900000	1.000000

[24]
```
scaler = StandardScaler()
data[['age', 'hypertension', 'heart_disease', 'avg_glucose', 'bmi', 'illness']] = scaler
```

[25]
```
label_encoder = LabelEncoder()
data['gender'] = label_encoder.fit_transform(data['gender'])
data['Marriage'] = label_encoder.fit_transform(data['Marriage'])
data['work_type'] = label_encoder.fit_transform(data['work_type'])
data['Living_type'] = label_encoder.fit_transform(data['Living_type'])
data['smoking_status'] = label_encoder.fit_transform(data['smoking_status'])
```

[26]
```
data.head()
```

	gender	age	hypertension	heart_disease	Marriage	work_type	Living_type	avg_g
0	1	0.521788	-0.468521	2.558744	1	1	1	2.
1	1	1.099960	-0.468521	2.558744	1	1	0	-0.
2	0	-0.278759	-0.468521	-0.390817	1	1	1	0.
3	0	1.055485	2.134375	-0.390817	1	2	0	1.
4	1	1.144435	-0.468521	-0.390817	1	1	1	1.

Next steps: Generate code with data View recommended plots New interactive sheet

[27]
```
y=data['illness']
x=data.drop(columns=['illness'])
```

```
y = y.values.reshape(-1, 1)
```

[29]
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.05, random_state=
```

Double-click (or enter) to edit

[30]
```
# design architecture of model
#model = keras.Sequential()

#model.add(keras.Input(shape=(400,11)))
#model.add(keras.layers.Dense(1, activation="relu"))
#model.add(keras.layers.Dense(1, activation="sigmoid"))

#model.summary()
```

Resources

You are not subscribed. Learn more
You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.
At your current usage level, this runtime may last up to 7 hours 10 minutes.

Manage sessions

Want more memory and disk space?
Upgrade to Colab Pro

Python 3 Google Compute Engine backend
Showing resources from 10:28 PM to 10:46 PM

System RAM
2.2 / 12.7 GB

Disk
31.5 / 107.7 GB

Change runtime type

```
[31] model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
[32] X_train.shape
```

```
(380, 10)
```

```
[33]
     y_train.shape
```

```
(380, 1)
```

```
[34] import numpy as np
     import pandas as pd
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, Dropout, Flatten
     from tensorflow.keras.optimizers import Adam
     from tensorflow.keras.callbacks import EarlyStopping
```

```
[35] y_train = np.where(y_train == -1, 0, y_train)
     y_test = np.where(y_test == -1, 0, y_test)
```

```
[36] from sklearn.utils.class_weight import compute_class_weight
     import numpy as np

     y_train_flat = y_train.ravel()  # แปลงเป็น 1 มิติ
     classes = np.unique(y_train_flat)  # ดึงค่าคลาสที่มีอยู่

     # คำนวณค่า class weight
     class_weights = compute_class_weight(class_weight='balanced', classes=classes, y=y_trai

     # แปลงเป็น dictionary โดยใช้ enumerate เพื่อจับคู่ index กับ class
     class_weight_dict = {classes[i]: class_weights[i] for i in range(len(classes))}

     print(class_weight_dict)  # ดูค่า class weight
```

```
{0.0: 1.0, 1.0: 1.0}
```

```
[50] # สร้างโมเดล
     model = Sequential([
         Flatten(),
         Dense(128, input_dim=10, activation='relu'),
         Dropout(0.35),

         Dense(64, activation='relu'),
         Dropout(0.35),

         Dense(32, activation='relu'),
         Dropout(0.35),

         Dense(16, activation='relu'),
         Dropout(0.3),

         Dense(1, activation='sigmoid')
     ])

     optimizer = Adam(learning_rate=0.0001)
     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

     early_stopping = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=T
     history = model.fit(X_train, y_train, epochs=100, validation_data=(X_test, y_test),
                         batch_size=64, callbacks=[early_stopping], class_weight=class_weig
     model.evaluate(X_test, y_test)

     model.save('my_model.keras')

     loss, accuracy = model.evaluate(X_test, y_test)
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8079 - loss: 0.4168 - val_accuracy
Epoch 74/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7844 - loss: 0.4907 - val_accuracy
Epoch 75/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8055 - loss: 0.4428 - val_accuracy
Epoch 76/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7962 - loss: 0.4341 - val_accuracy
Epoch 77/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7921 - loss: 0.4338 - val_accuracy
Epoch 78/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7551 - loss: 0.4435 - val_accuracy
Epoch 79/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7882 - loss: 0.4293 - val_accuracy
Epoch 80/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7927 - loss: 0.4324 - val_accuracy
Epoch 81/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8034 - loss: 0.4549 - val_accuracy
Epoch 82/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8054 - loss: 0.4218 - val_accuracy
Epoch 83/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8051 - loss: 0.4334 - val_accuracy
Epoch 84/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8320 - loss: 0.4086 - val_accuracy
Epoch 85/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8192 - loss: 0.4348 - val_accuracy
Epoch 86/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8084 - loss: 0.4266 - val_accuracy
Epoch 87/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8320 - loss: 0.4501 - val_accuracy
Epoch 88/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8303 - loss: 0.3951 - val_accuracy
Epoch 89/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8084 - loss: 0.4208 - val_accuracy
Epoch 90/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8623 - loss: 0.3619 - val_accuracy
```

```
Epoch 91/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8122 - loss: 0.4008 - val_accuracy
Epoch 92/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7974 - loss: 0.4289 - val_accuracy
Epoch 93/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8176 - loss: 0.3929 - val_accuracy
Epoch 94/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7985 - loss: 0.4147 - val_accuracy
Epoch 95/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8173 - loss: 0.4072 - val_accuracy
Epoch 96/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8354 - loss: 0.3896 - val_accuracy
Epoch 97/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8031 - loss: 0.4226 - val_accuracy
Epoch 98/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8368 - loss: 0.3873 - val_accuracy
Epoch 99/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.7954 - loss: 0.4348 - val_accuracy
Epoch 100/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.8472 - loss: 0.3779 - val_accuracy
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.9000 - loss: 0.2736
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - accuracy: 0.9000 - loss: 0.2736
```

```
print(f"Loss: {loss}")
print(f"Accuracy: {accuracy:.2f}%")
```

```
Loss: 0.2735774517059326
Accuracy: 0.90%
```

Colab paid products - Cancel contracts here

✓ 0s completed at 10:44 PM