customer-staying-or-not.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

Share

[3] 0s
```
import pandas as pd
```

[4] 0s
```
# load customer dataset from gh

url = "https://raw.githubusercontent.com/joneikholmkea/machine-learning/main/spring26/lektion2/customer_staying_or_not.csv"
df = pd.read_csv(url)
```

[5] 0s
```
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	NaN	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

Next steps: Generate code with df | New interactive sheet

[6] 0s
```
# drop irrelevant columns
df = df.drop(columns=['RowNumber', 'CustomerId', 'Surname'])
X = df.drop(columns=["Exited"])
```

[7] 0s
```
# set y to "Exited" (true or false)

y = df["Exited"]
```

[8] 0s
```
# convert categorical variables (Geography, Gender) to numerical varables

X = pd.get_dummies(X, drop_first=True)
```

[9] 0s
```
# saves column names for later use

feature_columns = X.columns
```

[10] 2s
```
from sklearn.model_selection import train_test_split

# split into training (80%) and test (20%) sets

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42
)
```

[11] 0s
```
from sklearn.preprocessing import StandardScaler

# scale features to mean=0 and std=1
# meaning that if we have age = 20-60 and salary 7000-12000 the model will be too
# biased towards the larger numbers

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[12] 4s
```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Activation

# build the model

model = tf.keras.Sequential([
    Dense(64, activation="tanh", input_shape = (X_train.shape[1],)), # first hidden layer
    Dense(32, activation="tanh"), # second hidden layer
    Dense(16, activation="tanh"), # third hidden layer
    Dense(1, activation="sigmoid") # output layer - we can use sigmoid because it is a binary classification
])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, pre
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[13] 0s
```
#compile the model

model.compile(
    optimizer='adam',
    loss='binary_crossentropy', # again, binary classification
    metrics=['accuracy']
)
```

[14] 0s
```
# displays the architecture of my neural network
# it shows: Layertype and name, output shape and number of trainable parameters (weight + biases )
# Total params: Sum of all trainable parameters in the model
# more parameters = more complex, slower training
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	768
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 16)	528
dense_3 (Dense)	(None, 1)	17

Total params: 3,393 (13.25 KB)
Trainable params: 3,393 (13.25 KB)
Non-trainable params: 0 (0.00 B)

[15]
```
# train model
```

```
model.fit(
    X_train,
    y_train,
    epochs=50,
    validation_split=0.2,
    verbose=1
)
```

```
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8786 - loss: 0.3043 - val_accuracy: 0.8500 - val_loss: 0.3503
Epoch 23/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8666 - loss: 0.3187 - val_accuracy: 0.8575 - val_loss: 0.3525
Epoch 24/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8711 - loss: 0.3095 - val_accuracy: 0.8537 - val_loss: 0.3506
Epoch 25/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8800 - loss: 0.2983 - val_accuracy: 0.8525 - val_loss: 0.3536
Epoch 26/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8736 - loss: 0.3082 - val_accuracy: 0.8531 - val_loss: 0.3503
Epoch 27/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8739 - loss: 0.3061 - val_accuracy: 0.8519 - val_loss: 0.3508
Epoch 28/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8731 - loss: 0.3124 - val_accuracy: 0.8600 - val_loss: 0.3487
Epoch 29/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8744 - loss: 0.3038 - val_accuracy: 0.8531 - val_loss: 0.3527
Epoch 30/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8776 - loss: 0.2993 - val_accuracy: 0.8506 - val_loss: 0.3539
Epoch 31/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8775 - loss: 0.2955 - val_accuracy: 0.8544 - val_loss: 0.3535
Epoch 32/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8749 - loss: 0.3081 - val_accuracy: 0.8537 - val_loss: 0.3526
Epoch 33/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8828 - loss: 0.2948 - val_accuracy: 0.8569 - val_loss: 0.3565
Epoch 34/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8771 - loss: 0.2983 - val_accuracy: 0.8562 - val_loss: 0.3552
Epoch 35/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8750 - loss: 0.3030 - val_accuracy: 0.8512 - val_loss: 0.3578
Epoch 36/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8794 - loss: 0.2983 - val_accuracy: 0.8550 - val_loss: 0.3573
Epoch 37/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8733 - loss: 0.3009 - val_accuracy: 0.8531 - val_loss: 0.3591
Epoch 38/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8840 - loss: 0.2897 - val_accuracy: 0.8500 - val_loss: 0.3613
Epoch 39/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8742 - loss: 0.2932 - val_accuracy: 0.8525 - val_loss: 0.3610
Epoch 40/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8801 - loss: 0.2932 - val_accuracy: 0.8525 - val_loss: 0.3581
Epoch 41/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8780 - loss: 0.2864 - val_accuracy: 0.8475 - val_loss: 0.3636
Epoch 42/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8799 - loss: 0.2806 - val_accuracy: 0.8431 - val_loss: 0.3671
Epoch 43/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8874 - loss: 0.2846 - val_accuracy: 0.8487 - val_loss: 0.3630
Epoch 44/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8890 - loss: 0.2815 - val_accuracy: 0.8438 - val_loss: 0.3685
Epoch 45/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8899 - loss: 0.2750 - val_accuracy: 0.8494 - val_loss: 0.3648
Epoch 46/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8825 - loss: 0.2801 - val_accuracy: 0.8431 - val_loss: 0.3688
Epoch 47/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8876 - loss: 0.2729 - val_accuracy: 0.8462 - val_loss: 0.3690
Epoch 48/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8863 - loss: 0.2736 - val_accuracy: 0.8469 - val_loss: 0.3672
Epoch 49/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8834 - loss: 0.2767 - val_accuracy: 0.8481 - val_loss: 0.3746
Epoch 50/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8833 - loss: 0.2795 - val_accuracy: 0.8462 - val_loss: 0.3771
<keras.src.callbacks.history.History at 0x7edaa4da9e80>
```

[16]

```
import numpy as np

# prediction for new customer

new_customer = {
    'CreditScore' : 600,
    'Geography' : "France",
    'Gender' : "Male",
    'Age' : 40,
    'Tenure' : 3,
    'Balance' : 60000,
    'NumOfProducts' : 2,
    'HasCrCard' : 1,
    'IsActiveMember' : 1,
    'EstimatedSalary' : 50000
}
```

[17]

```
# we need to encode categiorical variables, match training columns, apply the same scaling again

new_customer_df = pd.DataFrame([new_customer])

new_customer_encoded = pd.get_dummies(new_customer_df, drop_first=True)

new_customer_encoded = new_customer_encoded.reindex(columns=feature_columns, fill_value=0)

new_customer_scaled = scaler.transform(new_customer_encoded)
```

[19]

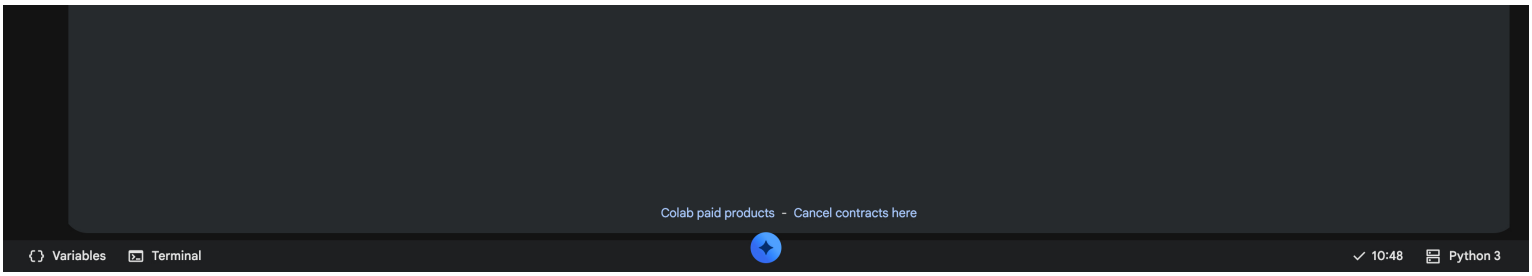
```
# get probability

prediction = model.predict(new_customer_scaled)[0][0]

if prediction > 0.5:
    print("The customer will leave the bank")
    print(f"Probability: {prediction}")
else:
    print("The customer will not leave the bank")
    print(f"Probability: {prediction}")

model.save("model.keras")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
The customer will not leave the bank
Probability: 0.06198442354798317
```


Colab paid products - Cancel contracts here
Variables
Terminal
10:48
Python 3