

ANN PRACTICAL

FOR CLASSIFICATION

Tensorflow → library for Deep learning by Google

Keras → API on top of Tensorflow

Important; we can use either Tensorflow / Pytorch.

STEP I; Setting up Environment

Bash → Conda create -p venv python == 3.13

conda activate venv

pip install -r requirements.txt

install ipynbkernel

PREPROCESSING

1. Remove unnecessary columns
2. Use Label Encoder → Binary Categories
3. Use OHE for → Multi Categories

Imp If we use label Encoder on multi class, it will assign $a=1, b=2, c=0$; which will mean for DL $b > a \& c$, so OHE is preferred.

4. Save Encoders in pickle file.
5. Split the Data into train-test
6. Scale the Data (Standard Scaler)
→ Save this also as pickle

Imp In Tensorflow, we call ANN as sequential model

Useful parameters; Steps

1. sequential
2. Dense → 64 means 64 neuron
3. Activation fn → Sigmoid, softmax → O/p
Relu, Tanh → Hidden
4. Optimizers →

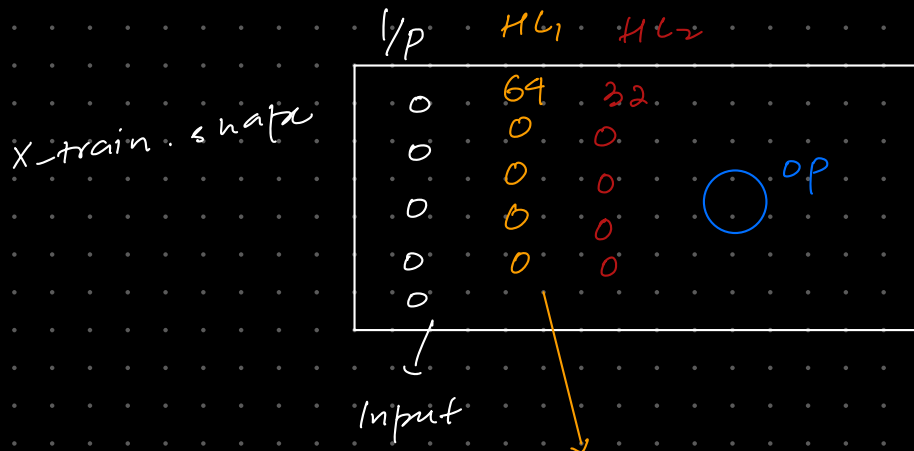
5. Loss \rightarrow GD, SGD

6. Evaluation metrics

7. Training \rightarrow logs \rightarrow folders
 \downarrow
Tensorboard

Libraries

\rightarrow tensorflow.keras.models import Sequential
tensorflow.keras.layers import Dense \rightarrow Nodes
tensorflow.keras.callbacks import EarlyStopping
TensorBoard
 \hookrightarrow logs



Build ANN Model

```
# Build ANN Model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],))) ## 1st Input layer connected to 64 neurons - HL1
model.add(Dense(32, activation='relu')) # no shape needed here as it is a hidden layer - Hidden layer with 32 neurons - HL2
model.add(Dense(1, activation='sigmoid')) # Output layer with 1 neuron for binary classification
```

\rightarrow Setup Optimizer, Loss function, metrics

```
# compile the model with optimizer, loss function and metrics
from tensorflow.keras.optimizers import Adam
model.compile(optimizer=Adam(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])
```

\rightarrow Setup Tensorboard & Early stopping

- ✓ TensorBoard: For real-time monitoring of training (loss, accuracy, etc.)
- ✓ EarlyStopping: To automatically stop training when performance stops improving (prevents overfitting and saves time) \rightarrow stops when model reaches convergence

\rightarrow mention epochs

```
# Set up the tensorboard callback
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)

# Set up early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

\downarrow
Ruko para sabar kro till 5 epochs

→ Train the model file

```
history = model.fit(X_train_scaled, y_train,
                    validation_data=(X_test_scaled, y_test),
                    epochs=100,
                    callbacks=[tensorboard_callback, early_stopping])
```

✓ 16.5s

Epoch 1/100
250/250 [=====] - 2s 7ms/step - loss: 0.5535 - accuracy: 0.7806 - val_loss: 0.5473 - val_accuracy: 0.8030
Epoch 2/100
250/250 [=====] - 1s 6ms/step - loss: 1.2733 - accuracy: 0.7465 - val_loss: 1.9880 - val_accuracy: 0.6815
Epoch 3/100
250/250 [=====] - 1s 6ms/step - loss: 2.8616 - accuracy: 0.7368 - val_loss: 3.8312 - val_accuracy: 0.7565
Epoch 4/100
250/250 [=====] - 1s 6ms/step - loss: 3.3271 - accuracy: 0.7303 - val_loss: 17.8678 - val_accuracy: 0.6560
Epoch 5/100
250/250 [=====] - 1s 6ms/step - loss: 6.7133 - accuracy: 0.7289 - val_loss: 3.3616 - val_accuracy: 0.8045
Epoch 6/100
250/250 [=====] - 1s 6ms/step - loss: 8.6875 - accuracy: 0.7306 - val_loss: 9.0012 - val_accuracy: 0.7670
Epoch 7/100
250/250 [=====] - 1s 6ms/step - loss: 13.4958 - accuracy: 0.7394 - val_loss: 7.7190 - val_accuracy: 0.7380
Epoch 8/100
250/250 [=====] - 1s 6ms/step - loss: 24.6641 - accuracy: 0.7144 - val_loss: 17.9428 - val_accuracy: 0.7920
Epoch 9/100
250/250 [=====] - 1s 6ms/step - loss: 18.2670 - accuracy: 0.7521 - val_loss: 6.6817 - val_accuracy: 0.8015
Epoch 10/100
250/250 [=====] - 1s 6ms/step - loss: 19.5793 - accuracy: 0.7377 - val_loss: 43.4841 - val_accuracy: 0.6100
Epoch 11/100
250/250 [=====] - 1s 6ms/step - loss: 25.3717 - accuracy: 0.7340 - val_loss: 17.2696 - val_accuracy: 0.7635

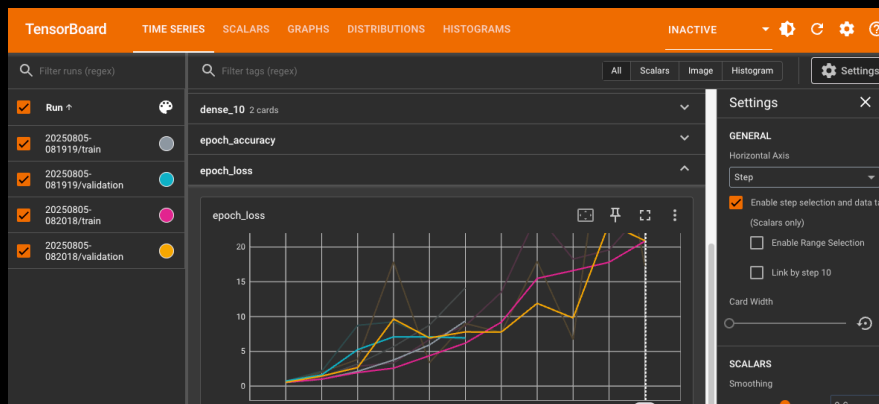
→ Save model as .h5

```
# Save model file as .h5 is compatible with Keras
model.save('customer_churn_model.h5')
```

→ Monitor logs in Tensorboard.

```
# Load TensorBoard Extension
%load_ext tensorboard

# Launch TensorBoard
%tensorboard --logdir logs/fit
```



Finally Get predictions

→ Import pickle files & .h5 file

→ Get prediction → Ready for production.

→ Create app.py which runs all the trained models

FOR REGRESSION

```
### Load the trained model, scaler pickle, onehot
model=load_model('customer_churn_model.h5')

## load the encoder and scaler
with open('ohe_geo.pkl','rb') as file:
    label_encoder_geo=pickle.load(file)

with open('label_encoder_gender.pkl', 'rb') as file:
    label_encoder_gender = pickle.load(file)

with open('scaler.pkl', 'rb') as file:
    scaler = pickle.load(file)
```

```
In [65]: ## Predict churn
         prediction=model.predict(input_scaled)
         prediction
```

WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x33227f560> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x33227f560> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

1/1 [=====] - 0s 37ms/step

```
Out[65]: array([[0.01600381]], dtype=float32)
```

```
In [66]: prediction_proba = prediction[0][0]
```

```
In [67]: prediction_proba
```

```
Out[67]: 0.016003806
```

```
In [68]: if prediction_proba > 0.5:
         print('The customer is likely to churn.')
         else:
         print('The customer is not likely to churn.')
```

The customer is not likely to churn.