```
# model 21 - deep learning ----SMU DS----Raj Agrawal SEP/2023
%matplotlib inline
!pip install keras-tuner --upgrade
# Import our dependencies
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
# from keras.utils import np_utils
import warnings
warnings.filterwarnings('ignore')
```

```
Requirement already satisfied: keras-tuner in /usr/local/lib/python3.10/dist-packages (1.3.5)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from keras-tuner) (23.1)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from keras-tuner) (2.31.0)
Requirement already satisfied: kt-legacy in /usr/local/lib/python3.10/dist-packages (from keras-tuner) (1.0.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kera
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (3.
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tune
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tune
```

```
import csv
import numpy as np
import pandas as pd
import os
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remou
```

```
import pandas as pd
df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/charity_data.csv')
```

```
df.head()
```

	EIN	NAME	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_C
0	10520599	BLUE KNIGHTS MOTORCYCLE CLUB	T10	Independent	C1000	Productl
1	10531628	AMERICAN CHESAPEAKE CLUB CHARITABLE TR	T3	Independent	C2000	Preserva
2	10547893	ST CLOUD PROFESSIONAL FIREFIGHTERS	T5	CompanySponsored	C3000	Productl

```
application_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/charity_data.csv')


# Drop the non-beneficial ID columns, 'EIN'
application_df = application_df.drop(columns=['EIN'], axis=1)
application_df.head()
```

	NAME	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGAN
0	BLUE KNIGHTS MOTORCYCLE CLUB	T10	Independent	C1000	ProductDev	Ass
1	AMERICAN CHESAPEAKE CLUB CHARITABLE TR	T3	Independent	C2000	Preservation	Co-c
2	ST CLOUD PROFESSIONAL FIREFIGHTERS	T5	CompanySponsored	C3000	ProductDev	Ass
3	SOUTHSIDE ATHLETIC ASSOCIATION	T3	CompanySponsored	C2000	Preservation	
4	GENETIC RESEARCH INSTITUTE OF THE DESERT	T3	Independent	C1000	Heathcare	

```
# Determine the number of unique values in each column.
application_df.nunique()
```

```
NAME                      19568
APPLICATION_TYPE             17
AFFILIATION                   6
CLASSIFICATION               71
USE_CASE                      5
ORGANIZATION                  4
STATUS                        2
INCOME_AMT                    9
SPECIAL_CONSIDERATIONS        2
ASK_AMT                    8747
IS_SUCCESSFUL                 2
dtype: int64
```

```
# Look at APPLICATION_TYPE value counts for binning
app_counts = application_df['APPLICATION_TYPE'].value_counts()
app_counts
```

```
T3     27037
T4      1542
T6      1216
T5      1173
T19     1065
T8       737
T7       725
T10      528
T9       156
T13       66
T12       27
```

```
T2        16
T25        3
T14        3
T29        2
T15        2
T17        1
Name: APPLICATION_TYPE, dtype: int64
```

```
# Choose a cutoff value and create a list of application types to be replaced
application_types_to_replace = list(app_counts[app_counts < 500].index)

# Replace in dataframe
for app in application_types_to_replace:
    application_df['APPLICATION_TYPE'] = application_df['APPLICATION_TYPE'].replace(
        app, "Other")

# Check to make sure binning was successful
application_df['APPLICATION_TYPE'].value_counts()
```

```
T3       27037
T4        1542
T6        1216
T5        1173
T19       1065
T8         737
T7         725
T10        528
Other      276
Name: APPLICATION_TYPE, dtype: int64
```

```
# Look at CLASSIFICATION value counts for binning
class_counts = application_df['CLASSIFICATION'].value_counts()
class_counts
```

```
C1000    17326
C2000     6074
C1200     4837
C3000     1918
C2100     1883
          ...
C4120        1
C8210        1
C2561        1
C4500        1
C2150        1
Name: CLASSIFICATION, Length: 71, dtype: int64
```

```
# You may find it helpful to look at CLASSIFICATION value counts >1
class_counts_gt1 = class_counts.loc[class_counts > 1]
class_counts_gt1.head()
```

```
C1000    17326
C2000     6074
C1200     4837
C3000     1918
C2100     1883
Name: CLASSIFICATION, dtype: int64
```

```
# Choose a cutoff value and create a list of classifications to be replaced
classifications_to_replace = list(class_counts[class_counts < 1000].index)

# Replace in dataframe
for cls in classifications_to_replace:
    application_df['CLASSIFICATION'] = application_df['CLASSIFICATION'].replace(
        cls, "Other")

# Check to make sure binning was successful
application_df['CLASSIFICATION'].value_counts()
```

```
C1000    17326
C2000     6074
C1200     4837
Other     2261
C3000     1918
C2100     1883
Name: CLASSIFICATION, dtype: int64
```

```
# Look at NAME value counts for binning
name_counts = application_df['NAME'].value_counts()
name_counts
```

```
PARENT BOOSTER USA INC                                                  1260
TOPS CLUB INC                                                            765
UNITED STATES BOWLING CONGRESS INC                                       700
WASHINGTON STATE UNIVERSITY                                              492
AMATEUR ATHLETIC UNION OF THE UNITED STATES INC                          408
                                                                        ...
ST LOUIS SLAM WOMENS FOOTBALL                                              1
AIESEC ALUMNI IBEROAMERICA CORP                                            1
WEALLBLEEDRED ORG INC                                                      1
AMERICAN SOCIETY FOR STANDARDS IN MEDIUMSHIP & PSYCHICAL INVESTIGATI       1
WATERHOUSE CHARITABLE TR                                                   1
Name: NAME, Length: 19568, dtype: int64
```

```
# Choose a cutoff value and create a list of names to be replaced
names_to_replace = list(name_counts[name_counts < 100].index)

# Replace in dataframe
for name in names_to_replace:
    application_df['NAME'] = application_df['NAME'].replace(
        name, "Other")

# Check to make sure binning was successful
application_df['NAME'].value_counts()
```

```
Other                                                                 25987
PARENT BOOSTER USA INC                                                 1260
TOPS CLUB INC                                                           765
UNITED STATES BOWLING CONGRESS INC                                      700
WASHINGTON STATE UNIVERSITY                                             492
AMATEUR ATHLETIC UNION OF THE UNITED STATES INC                         408
PTA TEXAS CONGRESS                                                      368
SOROPTIMIST INTERNATIONAL OF THE AMERICAS INC                           331
ALPHA PHI SIGMA                                                         313
TOASTMASTERS INTERNATIONAL                                              293
MOST WORSHIPFUL STRINGER FREE AND ACCEPTED MASONS                       287
LITTLE LEAGUE BASEBALL INC                                              277
INTERNATIONAL ASSOCIATION OF LIONS CLUBS                                266
MOMS CLUB                                                               210
INTERNATIONAL ASSOCIATION OF SHEET METAL AIR RAIL & TRANSPORTATION      206
AMERICAN ASSOCIATION OF UNIVERSITY WOMEN                                197
FARMERS EDUCATIONAL AND COOPERATIVE UNION OF AMERICA                    166
KNIGHTS OF COLUMBUS                                                     158
HABITAT FOR HUMANITY INTERNATIONAL INC                                  154
TENNESSEE ORDER OF THE EASTERN STAR                                     151
VETERANS OF FOREIGN WARS OF THE UNITED STATES AUXILIARY                 144
PTA UTAH CONGRESS                                                       140
THE UNITED STATES PONY CLUBS INC                                        136
CIVITAN INTERNATIONAL                                                   131
SIGMA BETA DELTA INC                                                    127
HONOR SOCIETY OF PHI KAPPA PHI                                          107
MONTANA 4-H FOUNDATION INC                                              107
WASHINGTON STATE GRANGE                                                 106
UNIVERSITY OF WYOMING                                                   105
DEMOLAY INTERNATIONAL                                                   104
SERTOMA INC                                                             103
Name: NAME, dtype: int64
```

```
# Convert categorical data to numeric with `pd.get_dummies`
```

```
application_numeric = pd.get_dummies(application_df)
application_numeric.head()
```

	STATUS	ASK_AMT	IS_SUCCESSFUL	NAME_ALPHA PHI SIGMA	NAME_AMATEUR ATHLETIC UNION OF THE UNITED STATES INC	NAME_AMERICAN ASSOCIATION OF UNIVERSITY WOMEN	NAME_CIVITAN INTERNATIONAL	NAME_DEMOLAY INTERNATIONAL	NAME_FARMERS EDUCATIONAL AND COOPERATIVE UNION OF AMERICA
0	1	5000	1	0	0	0	0	0	0
1	1	108590	1	0	0	0	0	0	0
2	1	5000	0	0	0	0	0	0	0
3	1	6692	1	0	0	0	0	0	0
4	1	142590	1	0	0	0	0	0	0

5 rows × 75 columns

```
# Split our preprocessed data into our features and target arrays
X = application_numeric.drop(['IS_SUCCESSFUL'], axis=1)
y = application_numeric['IS_SUCCESSFUL']

# Split the preprocessed data into a training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=58)


# Create a StandardScaler instances
scaler = StandardScaler()

# Fit the StandardScaler
X_scaler = scaler.fit(X_train)

# Scale the data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)


## Compile, Train and Evaluate the Model


# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len(X_train_scaled[0])
hidden_nodes_layer1 = 10
hidden_nodes_layer2 = 8
hidden_nodes_layer3= 6

nn_model2 = tf.keras.models.Sequential()

# First hidden layer
nn_model2.add(tf.keras.layers.Dense(units=hidden_nodes_layer1,
             input_dim=number_input_features, activation="relu"))

# Second hidden layer
nn_model2.add(tf.keras.layers.Dense(
    units=hidden_nodes_layer2, activation="sigmoid"))

# Third hidden layer
nn_model2.add(tf.keras.layers.Dense(
    units=hidden_nodes_layer3, activation="sigmoid"))

# Output layer
nn_model2.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))


# Check the structure of the model
```

```
nn_model2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                750

 dense_1 (Dense)             (None, 8)                 88

 dense_2 (Dense)             (None, 6)                 54

 dense_3 (Dense)             (None, 1)                 7

=================================================================
Total params: 899 (3.51 KB)
Trainable params: 899 (3.51 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Compile the model
nn_model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
# Train the model
fit_model2 = nn_model2.fit(X_train_scaled, y_train, epochs=30)
```

```
Epoch 2/30
804/804 [==============================] - 3s 4ms/step - loss: 0.5208 - accuracy: 0.7519
Epoch 3/30
804/804 [==============================] - 2s 3ms/step - loss: 0.5112 - accuracy: 0.7553
Epoch 4/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5072 - accuracy: 0.7561
Epoch 5/30
804/804 [==============================] - 2s 3ms/step - loss: 0.5020 - accuracy: 0.7572
Epoch 6/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4995 - accuracy: 0.7586
Epoch 7/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4977 - accuracy: 0.7584
Epoch 8/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4966 - accuracy: 0.7579
Epoch 9/30
804/804 [==============================] - 1s 2ms/step - loss: 0.4953 - accuracy: 0.7581
Epoch 10/30
804/804 [==============================] - 1s 2ms/step - loss: 0.4943 - accuracy: 0.7576
Epoch 11/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4939 - accuracy: 0.7579
Epoch 12/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4932 - accuracy: 0.7567
Epoch 13/30
804/804 [==============================] - 2s 3ms/step - loss: 0.4926 - accuracy: 0.7579
Epoch 14/30
804/804 [==============================] - 2s 3ms/step - loss: 0.4921 - accuracy: 0.7577
Epoch 15/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4917 - accuracy: 0.7584
Epoch 16/30
804/804 [==============================] - 1s 2ms/step - loss: 0.4915 - accuracy: 0.7599
Epoch 17/30
804/804 [==============================] - 1s 2ms/step - loss: 0.4912 - accuracy: 0.7584
Epoch 18/30
804/804 [==============================] - 1s 2ms/step - loss: 0.4908 - accuracy: 0.7582
Epoch 19/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4905 - accuracy: 0.7598
Epoch 20/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4904 - accuracy: 0.7598
Epoch 21/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4899 - accuracy: 0.7593
Epoch 22/30
804/804 [==============================] - 3s 4ms/step - loss: 0.4897 - accuracy: 0.7589
Epoch 23/30
```

```
804/804 [==============================] - 2s 2ms/step - loss: 0.4891 - accuracy: 0.7598
Epoch 25/30
804/804 [==============================] - 1s 2ms/step - loss: 0.4889 - accuracy: 0.7592
Epoch 26/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4887 - accuracy: 0.7592
Epoch 27/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4884 - accuracy: 0.7607
Epoch 28/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4884 - accuracy: 0.7598
Epoch 29/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4877 - accuracy: 0.7600
Epoch 30/30
804/804 [==============================] - 2s 2ms/step - loss: 0.4881 - accuracy: 0.7599
```

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn_model2.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 1s - loss: 0.4908 - accuracy: 0.7562 - 752ms/epoch - 3ms/step
Loss: 0.49077799916267395, Accuracy: 0.7561516165733337
```

```
# Export our model to HDF5 file
# Define the filename
# filename = '/content/drive/MyDrive/UTSA_Homework/H5_Files/AlphabetSoupCharity_Optimization2.h5'
filename = '/content/drive/My Drive/Colab Notebooks/AlphabetSoupCharity_Optimization2.h5'
# Save the model to a HDF5 file
nn_model2.save(filename)
```

```
filename = '/content/drive/My Drive/Colab Notebooks/AlphabetSoupCharity_Optimization2.keras'
```

```
#END
```

✓ 0s completed at 4:53 PM