Alphabet Soup Charity

Predicting Successful Candidates for Funding using Neural Networks

Overview:

This report is prepared for Alphabet Soup which is a nonprofit foundation. The report is based on an analysis performed using Machine Learning and Neural Network. The purpose is to help prepare a tool for the foundation which can help it select the applicants for funding with the best chance of success in their ventures.

Data Source:

The analysis is based on the CSV provided by the business team at Alphabet Soup. The CSV contains more than 34,000 organizations that have received funding from the foundation over the years. Within this dataset are a number of columns that capture metadata about each organization, such as:

- **EIN** and **NAME** – Identification columns
- **APPLICATION_TYPE** – Alphabet Soup application type
- **AFFILIATION** – Affiliated sector of industry
- **CLASSIFICATION** – Government organization classification
- **USE_CASE** – Use case for funding
- **ORGANIZATION** – Organization type
- **STATUS** – Active status
- **INCOME_AMT** – Income classification
- **SPECIAL_CONSIDERATIONS** – Special considerations for application
- **ASK_AMT** – Funding amount requested
- **IS_SUCCESSFUL** – Was the money used effectively

I have used the features in the provided dataset to create a binary classifier that can predict whether applicants will be successful if funded by the foundation.

Neural Network Models:

There are two neural network models created to predict the successful candidates. The first model is created by applying some optimization methods but it did not achieve the 75% accuracy. A second optimized model was then created by adjusting the optimization methods to try and achieve 75% or greater accuracy.

Following are the steps taken to create each of the models:

ORIGINAL MODEL:

This neural network model was created by completing the following steps:

Data Preprocessing:

The data was preprocessed by reviewing the available variables.

- **Target Variable**
  The target variable in this model is:
  - IS_SUCCESSFUL

- **Feature Variables**
  The feature variables used for the model include:
  - APPLICATION_TYPE
  - AFFILIATION
  - CLASSIFICATION
  - USE_CASE
  - ORGANIZATION
  - STATUS
  - INCOME_AMT
  - SPECIAL_CONSIDERATIONS
  - ASK_AMT

- **Removed Variables**
  Following variables were removed from the input data as these are neither targets nor features:
  - EIN
  - NAME

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

# Display the dataset to confirm the drop
application_df.head()
```

	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZATION	STATUS	INCOME_AMT	SPECIAL_CONSIDERATIONS	ASK_AMT	IS_SUCCESSFUL
0	T10	Independent	C1000	ProductDev	Association	1	0	N	5000	1
1	T3	Independent	C2000	Preservation	Co-operative	1	1-9999	N	108590	1
2	T5	CompanySponsored	C3000	ProductDev	Association	1	0	N	5000	0
3	T3	CompanySponsored	C2000	Preservation	Trust	1	10000-24999	N	6692	1
4	T3	Independent	C1000	Heathcare	Trust	1	100000-499999	N	142590	1

- **Binning**
  The data preprocessing also included creating bins for two specific variables by creating cutoff values for value counts of the underlying categories, and creating a bin called ***Other*** to store the categories with a value count of less than the cutoff value.
    - APPLICATION_TYPE
      This variable was binned into 9 categories by creating a cutoff value of 500.

```
# Choose a cutoff value and create a list of application types to be replaced
# use the variable name `application_types_to_replace`
application_types_to_replace = list(application_type_count[application_type_count < 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
```

    - CLASSIFICATION
      This variable was binned into 6 categories by creating a cutoff value of 1000.

```
# Choose a cutoff value and create a list of classifications to be replaced
# use the variable name `classifications_to_replace`
classifications_to_replace = list(classification_counts[classification_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
```

Compiling, Training, and Evaluating the Model:

After preprocessing the data, the model was compiled, trained and evaluated by defining the following details:

- **Neuron Layers**
  There were two hidden layers and an output layer created in this model. The first hidden layer contained 80 neurons and the second layer had 30 neurons.

- **Activation Functions**
  Relu activation function was used for the hidden layers, whereas ***Sigmoid*** activation function was used for the output layer.

```
# 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 =  80
hidden_nodes_layer2 = 30

nn = tf.keras.models.Sequential()

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

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

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

- **Parameters**
  The model was trained based on a total of 5,981 parameters.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 80)                3520

 dense_4 (Dense)             (None, 30)                2430

 dense_5 (Dense)             (None, 1)                 31

=================================================================
Total params: 5,981
Trainable params: 5,981
Non-trainable params: 0
_________________________________________________________________
```

- **Epochs**
  The model was then fitted and trained by using 100 epochs.

```
# Train the model
fit_model = nn.fit(X_train_scaled,y_train,epochs=100)
```

- **Model Performance**
  This model was evaluated to have an accuracy score of 72.87% which is below the target performance of 75%.

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

```
268/268 - 0s - loss: 0.5649 - accuracy: 0.7287 - 426ms/epoch - 2ms/step
Loss: 0.5649230480194092, Accuracy: 0.7287463545799255
```

OPTIMIZED MODEL:

In order to improve the accuracy of the original model, following steps were taken:

1. Dropping lesser columns/ variables
2. Creating bins for an additional variable
3. Decreasing the number of values for the *Other* bin for one of the variables
4. Adding one more hidden layer
5. Changing the number of neurons in each hidden layer

This optimized neural network model was created by completing the following steps:

Data Preprocessing:

The data was preprocessed by reviewing the available variables.

- **Target Variable**
  The target variable in this model is:
  - IS_SUCCESSFUL

- **Feature Variables**
  The feature variables used for the model include:
  - NAME
  - APPLICATION_TYPE
  - AFFILIATION
  - CLASSIFICATION
  - USE_CASE
  - ORGANIZATION
  - STATUS
  - INCOME_AMT
  - SPECIAL_CONSIDERATIONS
  - ASK_AMT

- **Removed Variables**
  Only one variable was removed (as compared to two in the original model) from the input data as it is neither target nor feature:
    - EIN

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

# Display the dataset to confirm the drop
application_df.head()
```

	NAME	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZATION	STATUS	INCOME_AMT	SPECIAL_CONSIDERATIONS	ASK_AMT	IS_SUCCESSFUL
0	BLUE KNIGHTS MOTORCYCLE CLUB	T10	Independent	C1000	ProductDev	Association	1	0	N	5000	1
1	AMERICAN CHESAPEAKE CLUB CHARITABLE TR	T3	Independent	C2000	Preservation	Co-operative	1	1-9999	N	108590	1
2	ST CLOUD PROFESSIONAL FIREFIGHTERS	T5	CompanySponsored	C3000	ProductDev	Association	1	0	N	5000	0
3	SOUTHSIDE ATHLETIC ASSOCIATION	T3	CompanySponsored	C2000	Preservation	Trust	1	10000-24999	N	6692	1
4	GENETIC RESEARCH INSTITUTE OF THE DESERT	T3	Independent	C1000	Heathcare	Trust	1	100000-499999	N	142590	1

- **Binning**
  The data preprocessing also included creating bins for three specific variables (as compared to two variables in the original data) by creating cutoff values for value counts of the underlying categories, and creating a bin called ***Other*** to store the categories with a value count of less than the cutoff value.
    - NAME
      This variable was binned into various categories by creating a cutoff value of 10. Please note that this was not binned in the original model.

```
# Choose a cutoff value and create a list of names to be replaced
# use the variable name `names_to_replace`
names_to_replace = list(name_count[name_count < 10].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                                  21022
PARENT BOOSTER USA INC                  1260
TOPS CLUB INC                            765
UNITED STATES BOWLING CONGRESS INC       700
WASHINGTON STATE UNIVERSITY              492
                                        ...
CASCADE 4-H FOUNDATION                    10
FREE & ACCEPTED MASONS OF WASHINGTON      10
NEW MEXICO GARDEN CLUBS INC               10
NATIONAL ASSOCIATION OF HISPANIC NURSES   10
UNION OF CALIFORNIA STATE WORKERS         10
Name: NAME, Length: 223, dtype: int64
```

- APPLICATION_TYPE
  This variable was binned into 9 categories by creating a cutoff value of 500 which is similar to the original model.

```
# Choose a cutoff value and create a list of application types to be replaced
# use the variable name `application_types_to_replace`
application_types_to_replace = list(application_type_count[application_type_count < 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
```

- CLASSIFICATION
  This variable was binned into 7 categories by creating a cutoff value of 500. The original model had 6 categories based on cutoff value of 1000.

```
# Choose a cutoff value and create a list of classifications to be replaced
# use the variable name `classifications_to_replace`
classifications_to_replace = list(classification_counts[classification_counts < 500].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
C3000     1918
C2100     1883
Other     1484
C7000      777
Name: CLASSIFICATION, dtype: int64
```

Compiling, Training, and Evaluating the Model:

After preprocessing the data, the model was compiled, trained and evaluated by defining the following details:

- **Neuron Layers**
  There were three hidden layers (as compared to two in the original model) and an output layer created in this model. The first hidden layer contained 30 neurons, the second layer had 27 and the third layer had 21 neurons.

- **Activation Functions**
  Relu activation function was used for the hidden layers, whereas ***Sigmoid*** activation function was used for the output layer.

```
# 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 =  30
hidden_nodes_layer2 = 27
hidden_nodes_layer3 = 21

nn = tf.keras.models.Sequential()

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

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

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

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

- **Parameters**
  The model was trained based on a total of 9,487 parameters.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 30)                8040

 dense_1 (Dense)             (None, 27)                837

 dense_2 (Dense)             (None, 21)                588

 dense_3 (Dense)             (None, 1)                 22

=================================================================
Total params: 9,487
Trainable params: 9,487
Non-trainable params: 0
_________________________________________________________________
```

- **Epochs**
  The model was then fitted and trained by using 100 epochs.

```
# Train the model
fit_model = nn.fit(X_train_scaled,y_train,epochs=100)
```

- **Model Performance**
  This optimized model was evaluated to have an accuracy score of 77.87% which is above the target performance of 75% and 5% higher than the original model.

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

```
268/268 - 1s - loss: 0.4746 - accuracy: 0.7787 - 803ms/epoch - 3ms/step
Loss: 0.4745773375034332, Accuracy: 0.7786588668823242
```

Summary:

Based on the evaluation of the two models described in the sections above, my recommendation is to use the optimized model to help predict the successful candidates for charity funding at Alphabet Soup. The optimized model yielded an overall better accuracy score of 77.87% compared to the original model at 72.87% and it was also met the target accuracy of 75% or higher. This model uses one additional feature variable from the input dataset to predict the target variable. It also includes bins created for an additional variable as well as additional bins created for one of the variables compared to the original model. Adding the bins seemed to have improved the outcome of this model.

The optimized model also includes additional hidden layer by making it a total of three hidden layers. It does, however, has lesser number of neurons for the first two hidden layers. This implies that although the overall number of neurons were lesser in the optimized model, however, the increased number of layers ended up increasing the number of trained parameters which eventually affected the accuracy of this model in a positive manner.

The recommended model in this case is definitely the second optimized model as it was more accurately able to predict the successful candidates for charity funding based on the change in optimization methods.