Alphabet Soup Charity

Predicting Successful Candidates for Funding using Neural Networks

Overview

Report is based on the analysis using Machine Learning and Neural Networks for the Alphabet Soup which is a nonprofit foundation. The main aim of this analysis is to prepare the tool for the foundation which can help it to select the candidates for funding with the higher chances of success in their ventures.

Data Source

The analysis is performed on the dataset of 34,000 samples which is from the CSV file. CSV file contains the history of the funding organizations have received from the foundations over the year. Below is the list of columns in the obtained dataset:

- EIN and NAME – Identification columns
- APPLICATION_TYPE – Application type of Alphabet Soup
- AFFILIATION – Affiliated sector of industry
- USE_CASE – Use case for funding
- CLASSIFICATION – Classification of Government organization
- ORGANIZATION – Type of Organization
- STATUS – Active Status
- INCOME_AMT – Classification of Income
- SPECIAL_CONSIDERATION – Special Considerations for Application
- ASK_AMT – Requested Funding Amount
- IS_SUCCESSFUL – Is the funded amount was frequently used

These features from the dataset are used to train a neural network which can perform the binary classification, to predict whether applicants will be successful if funded by the foundation.

Neural Network Models

Two variants of neural networks are trained on the dataset, to predict the candidates for funding. The first variant of the neural network was able to achieve the accuracy of 72.8% which is not high, and second neural network was able to achieve the accuracy of 77.8% by changing some parameters of the model. Below are the steps performed for training the neural networks:

Original Model

The neural network is trained by performing the below steps:

Data Preprocessing

During the data preprocessing different techniques are applied on the dataset to make data ready for training the neural network.

- **Dropping Variables:** EIN and NAME variables were removed from the dataset.

```
# 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_
0	T10	Independent	C1000	ProductDev	Association	1	0	N	5000	
1	T3	Independent	C2000	Preservation	Co-operative	1	1-9999	N	108590	
2	T5	CompanySponsored	C3000	ProductDev	Association	1	0	N	5000	
3	T3	CompanySponsored	C2000	Preservation	Trust	1	10000-24999	N	6692	
4	T3	Independent	C1000	Heathcare	Trust	1	100000-499999	N	142590	

- **Binning:** Binning is applied on the APPLICATION_TYPE and CLASSIFICATION columns.

```
# 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()
```

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

```
# Choose a cutoff value and create a list of classifications to be replaced
# use the variable name `classifications_to_replace`
#  YOUR CODE GOES HERE
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()
```

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

- **Data Split:** Independent columns and dependent columns are separated and dataset is divided into training and testing. To train the neural network training set is used and for the evaluation testing set is used.

```
# Split our preprocessed data into our features and target arrays
y = application_numeric_df['IS_SUCCESSFUL']
X = application_numeric_df.drop(columns=['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=42)
```

- **Standard Scaling:** Standard Scaling is applied on the dataset to scale all the columns in the dataset into the same size, standard scaler will allow neural network to not get biased towards columns with higher values.

```
# 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)
```

Compiling Training and Evaluating Model

Once the data processing is applied on the dataset, model training and evaluation is performed by following the below steps:

- **Neuron Layers:** Neural network is composed of one input layer, two hidden layers and one output layer. First hidden layer has 80 neurons and second hidden layer has 30 neurons.
- **Activation Function:** For capturing the non-linear relationships in the dataset ReLU activation function was used and for the output layer Sigmoid activation function was used.

```
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"))

# Check the structure of the model
nn.summary()
```

- **Parameters:** The model is trained on total 5981 parameters.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 80)                3520      
                                                                 
 dense_1 (Dense)             (None, 30)                2430      
                                                                 
 dense_2 (Dense)             (None, 1)                 31        
                                                                 
=================================================================
Total params: 5981 (23.36 KB)
Trainable params: 5981 (23.36 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- **Epochs:** The model is trained on 100 epochs. 100 epoch means 100 times dataset will be passed to the model for training.

```
Epoch 95/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5340 - accuracy: 0.7397
Epoch 96/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5343 - accuracy: 0.7399
Epoch 97/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5343 - accuracy: 0.7394
Epoch 98/100
804/804 [==============================] - 3s 3ms/step - loss: 0.5341 - accuracy: 0.7389
Epoch 99/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5340 - accuracy: 0.7394
Epoch 100/100
804/804 [==============================] - 3s 3ms/step - loss: 0.5341 - accuracy: 0.7395
```

- Model Performance: After training the model is evaluated using the testing dataset and model has the accuracy of the 72.8%.

```
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: 0.5637 - accuracy: 0.7287 - 841ms/epoch - 3ms/step
Loss: 0.5636910796165466, Accuracy: 0.7287463545799255
```

Optimized Model

To improve the accuracy of the neural network below are the steps applied:

- Dropping lesser variables
- Preprocessing the data in more detailed
- Include more hidden layer.
- Decrease the number of neurons in each hidden layer.

Below are the steps take to train the optimized model:

Data Preprocessing

During the data preprocessing different techniques are applied on the dataset to make data ready for training the neural network.

- **Dropping Variable:** EIN variable was removed from the dataset.

```
application_df.drop(['EIN'], axis=1, inplace=True)
application_df.head()
```

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

- **Binning:** Binning is applied on the NAME, APPLICATION_TYPE and CLASSIFICATION columns.

```
names_to_replace = list(name_count[name_count < 10].index)
for name in names_to_replace:
    application_df['NAME'] = application_df['NAME'].replace(name,"Other")
application_df['NAME'].value_counts()
```

```
NAME
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: count, Length: 223, dtype: int64
```

```
application_types_to_replace = list(application_type_count[application_type_count < 500].index)

for app in application_types_to_replace:
    application_df['APPLICATION_TYPE'] = application_df['APPLICATION_TYPE'].replace(app,"Other")
application_df['APPLICATION_TYPE'].value_counts()
```

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

```
classifications_to_replace = list(classification_counts[classification_counts < 500].index)
for cls in classifications_to_replace:
    application_df['CLASSIFICATION'] = application_df['CLASSIFICATION'].replace(cls,"Other")
application_df['CLASSIFICATION'].value_counts()
```

```
CLASSIFICATION
C1000    17326
C2000     6074
C1200     4837
C3000     1918
C2100     1883
Other     1484
C7000      777
Name: count, dtype: int64
```

Compiling Training and Evaluating Model

Once the data processing is applied on the dataset, model training and evaluation is performed by following the below steps:

- **Neuron Layers:** Neural network is composed of one input layer, three hidden layers and one output layer. First hidden layer has 30 neurons, second hidden layer has 27 neurons and third layer has 21 neurons.
- **Activation Function:** For capturing the non-linear relationships in the dataset ReLU activation function was used and for the output layer Sigmoid activation function was used.

```
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()
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu"))
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation="relu"))
nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))
nn.summary()
```

- **Parameters:** The model is trained on total 9487 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: 9487 (37.06 KB)
Trainable params: 9487 (37.06 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- **Epochs:** The model is trained on 100 epochs. 100 epoch means 100 times dataset will be passed to the model for training.

```
Epoch 95/100
804/804 [==============================] - 2s 2ms/step - loss: 0.4140 - accuracy: 0.7988
Epoch 96/100
804/804 [==============================] - 2s 3ms/step - loss: 0.4138 - accuracy: 0.7997
Epoch 97/100
804/804 [==============================] - 1s 2ms/step - loss: 0.4135 - accuracy: 0.7995
Epoch 98/100
804/804 [==============================] - 1s 2ms/step - loss: 0.4137 - accuracy: 0.7996
Epoch 99/100
804/804 [==============================] - 1s 2ms/step - loss: 0.4137 - accuracy: 0.7993
Epoch 100/100
804/804 [==============================] - 1s 2ms/step - loss: 0.4138 - accuracy: 0.7993
```

- Model Performance: After training the model is evaluated using the testing dataset and model has the accuracy of the 77.8%.

```
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: 0.4772 - accuracy: 0.7782 - 447ms/epoch - 2ms/step
Loss: 0.4772453010082245, Accuracy: 0.778192400932312
```

Summary

Two variants of the neural networks are trained on the dataset of the Alphabet Soup to predict the successful candidates for the funding. Among two trained models optimized model is giving high accuracy as compare to the simple neural network with no optimization. Optimized model could be utilized for further predicting the right candidates for the charity funding. Optimized model is trained on one more feature as compare to simple model and also additional bins created for one of the variables compared to the original model. Addition of the bins seems to have a good addition in the model. Optimized model has one additional hidden layer, which makes the more in-depth processing of the model, with less number of neurons as compare to the simple model.

For the prediction of candidates for charity funding the optimized could be the best fit, it can easily predict the successful candidates for the charity funding.