Overview

The purpose of this analysis is to use data from a non-profit, Alphabet Soup, that provides funding to other organizations. This data will train a model that can be used to determine potential successful applications in the future.

Results

Preprocessing

- What variable(s) are the target(s) for your model?
    - The target variable for my model is the “IS_SUCCESSFUL” column, which indicates whether or not a given group achieved its funding goal.
- What variable(s) are the features for your model?
    - The features are “APPLICATION_TYPE,” “AFFILIATION,” “CLASSIFICATION,” “USE_CASE,” “ORGANIZATION,” “STATUS,” “INCOME_AMT,” “SPECIAL_CONSIDERATIONS,” and “ASK_AMT.”
- What variable(s) should be removed from the input data because they are neither targets nor features?
    - “EIN” and “NAME” are purely descriptive and were removed.

Compiling, Training, and Evaluating the Model

- How many neurons, layers, and activation functions did you select for your neural network model, and why?
    - In the original model, I used two hidden layers and one output layer.
    - The first layer contained eight nodes and the second layer contained six.
    - I used the in-class activities as a reference for these. I also do some research on Stack Overflow and Stack Exchange and I got the impression that the number of neurons chosen for a model is somewhat arbitrary. However, I kept in mind the possibility of overfitting the model to the data. I felt eight and six were sufficient.
- Were you able to achieve the target model performance?
    - I did not achieve the target model performance. My model only reached 72.5% accuracy.
- What steps did you take in your attempts to increase model performance?
    - I created two other models to see if I could break the 75% threshold.
    - Adjustments made in the “optimized” model:
        - “APPLICATION_TYPE” value
            - I changed the threshold for which application types were assigned “other” from counts of 200 to counts of 1000. This means that only application types that occurred more 1000 times would be included, the rest were changed to “other.”
        - “CLASSIFICATION” value

        - I changed the threshold for which classifications were assigned “other” from counts of 1750 to counts of 2000. This means that only classification values that occurred more 2000 times would be included, the rest were changed to “other.”
        - Moved it from 1750 to 2000
    - “ASK_AMT” value
        - I noticed there were 8747 different values in the “ASK_AMT” column, but the count for an ask amount of $5000 was 25398. I was curious if the model would be more accurate if I considered just the funding goals of $5000. So, I set all other ask amounts to “other.”
    - 200 epochs
        - Instead of 100 epochs, I used 200 epochs. This doubles the training time and would hopefully increase the accuracy.
    - Added more neurons, 10 to each layer
        - I added 10 neurons to each layer to see if that would boost accuracy.
    - Addded a third hidden layer
        - I added a third hidden layer to see if that would help with accuracy.
    - Did these adjustments work?
        - No, the model accuracy was 72.2%. This is marginally less than the original

Images

Original Model

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[0])
hidden_nodes_layer1 =  8
hidden_nodes_layer2 = 6

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()
```

Optimized Model

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[0])
hidden_nodes_layer1 =  10
hidden_nodes_layer2 = 10
#adding an extra node to see if the model performs better
hidden_nodes_layer3 =  10

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"))


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

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

#updated the cutoff number from 200 to 1000 to make consolidated more of the lesser-used categories
cutoff = 1000
application_types_to_replace = []

for type, count in all_types.items():
    if count < cutoff:
        application_types_to_replace.append(type)

# 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 replacement was successful
application_df['APPLICATION_TYPE'].value_counts()
```

```
APPLICATION_TYPE
T3       27037
Other     2266
T4        1542
T6        1216
T5        1173
T19       1065
Name: count, dtype: int64
```

```
# Choose a cutoff value and create a list of classifications to be replaced
# use the variable name `classifications_to_replace`

#Again, updated the cutoff here to 2000 from 1750
cutoff = 2000
classifications_to_replace = []

for item, count in all_class.items():
    if count < cutoff:
        classifications_to_replace.append(item)

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

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

```
CLASSIFICATION
C1000    17326
C2000     6074
Other     6062
C1200     4837
Name: count, dtype: int64
```

```python
# I notice a lot of campaigns with 5000 as the ask amount. I'm going to set this as the only value and the rest to other and see what the mode
# use the variable name `amts_to_replace`

cutoff = 5000
# Get all unique values in the column
amts_to_replace = application_df['ASK_AMT'].unique()

# Keep values that are not equal to cutoff
amts_to_replace = [amt for amt in amts_to_replace if amt != cutoff]

# Display values to be replaced for debugging
#print("Amounts to replace:", amts_to_replace)

# Replace values not equal to cutoff with 'Other'
application_df['ASK_AMT'] = application_df['ASK_AMT'].apply(lambda x: 'Other' if x != cutoff else x)

# Check to make sure replacement was successful
print(application_df['ASK_AMT'].value_counts())
```

```
ASK_AMT
5000     25398
Other     8901
Name: count, dtype: int64
```

Summary

In both cases, the models I created did not meet the 75% threshold. This means both models are unreliable and should not be used by Alphabet Soup. Neither model will provide a strong indicator of what types of organizations or campaigns will succeed.

Instead of a neural network, a decision tree could be useful in achieving a more reliable model. Decision trees are versatile in that they can handle both numerical and categorical data. This data set has both of these types of values. Similarly, decision trees can capture non-linear relationships between features and target variables. Another advantage is that decision trees can provide insight into the importance of certain features. Such a characteristic could help Alphabet Soup in understanding what factors lead to successful campaigns.