Alphabet Soup Analysis

MODULE 21 CHALLENGE

RAE ANN GREGG

3.11.24

Alphabet Soup Overview

The purpose of this project is to create a tool for Alphabet Soup to help select applicants for funding with the best chance of success in their ventures.
The provided dataset was used to create a binary classifier using a machine learning program to predict whether applicants will be successful if funded by Alphabet Soup.
The provided dataset included more than 34,000 organizations that have received funding from Alphabet Soup over the years.
The following information was provided in the dataset for use in the program to predict probability of success with at least 75% accuracy:

- **EIN** and **NAME**—Applicants
- **APPLICATION_TYPE**—Alphabet Soup application type
- **AFFILIATION**—Affiliated sector of industry such as Independent or Company Sponsored
- **CLASSIFICATION**—Government organization classification
- **USE_CASE**—Use case for funding such as Product Development, Preservation, Healthcare, etc.
- **ORGANIZATION**—Organization type such as Association, Co-operative or Trust
- **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

Alphabet Soup Analysis Results

Data Preprocessing

- The focus of this model is to predict the probability of success for an applicant if they receive funding.
- The IS_SUCCESSFUL data was the target variable used for the model.
- The other columns were used as feature variables for the model except the EIN column. I originally also removed the NAME column as suggested in the instructions. However, the accuracy increased when I included the NAME column and binned the NAME data.
- The EIN column was removed since it was only introducing noise into the model and was not needed to predict the probability of success for funding.
- I used binning/bucketing with the APPLICATION_TYPE, CLASSIFICATION, and NAME columns to combine the rare occurrence categories to optimize the model.
- The categorical data was converted into numeric data using get_dummies.
- The data was split 80/20 into training and testing data and scaled for a more balanced distribution.

Alphabet Soup Analysis Results

Compiling, Training, and Evaluating the Model

- I did 5 optimizations to get to the target of over 75% accuracy
  - **Original Mode**
    - Dropped EIN and Name Columns
    - 1st layer – 80 neurons with ReLU activation
    - 2nd layer – 30 neurons with ReLU activation
    - Output layer – 1 neuron with Sigmoid activation
    - 100 epochs
    - 51.43% Accuracy

```
Epoch 96/100
804/804 [==============================] - 1s 1ms/step - loss: 0.5347 - accuracy: 0.7390
Epoch 97/100
804/804 [==============================] - 1s 1ms/step - loss: 0.5349 - accuracy: 0.7399
Epoch 98/100
804/804 [==============================] - 1s 1ms/step - loss: 0.5347 - accuracy: 0.7402
Epoch 99/100
804/804 [==============================] - 1s 1ms/step - loss: 0.5350 - accuracy: 0.7390
Epoch 100/100
804/804 [==============================] - 1s 1ms/step - loss: 0.5344 - accuracy: 0.7391
<keras.src.callbacks.History at 0x7ebb75936680>
```

```
[ ] # 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 - 2s - loss: 0.7857 - accuracy: 0.5143 - 2s/epoch - 7ms/step
    Loss: 0.7856943607330322, Accuracy: 0.5142857432365417
```

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.
    nn = tf.keras.models.Sequential()

    input_dims = len(X_train[0])

    # First hidden layer
    nn.add(tf.keras.layers.Dense(units=80, activation="relu", input_dim = input_dims))

    # Second hidden layer
    nn.add(tf.keras.layers.Dense(units=30, activation="relu"))

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

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

    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)
    _________________________________________________________________
```

Alphabet Soup Analysis Results

Compiling, Training, and Evaluating the Model

- I did 5 optimizations to get to the target of over 75% accuracy
  - **Optimization 1** – added a hidden layer and increased the nodes
    - Dropped EIN and Name Columns
    - 1st layer – 80 neurons with ReLU activation
    - 2nd layer – 40 neurons with ReLU activation
    - 3rd layer – 20 neurons with ReLU activation
    - Output layer – 1 neuron with Sigmoid activation
    - 100 epochs
    - 72.92% Accuracy – increased by over 20%

```
Epoch 96/100
804/804 [==============================] - 3s 4ms/step - loss: 0.5332 - accuracy: 0.7406
Epoch 97/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5330 - accuracy: 0.7400
Epoch 98/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5335 - accuracy: 0.7404
Epoch 99/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5331 - accuracy: 0.7404
Epoch 100/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5341 - accuracy: 0.7404
<keras.src.callbacks.History at 0x7a17cc55bdf0>
```

```
# 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 - 2s - loss: 0.5704 - accuracy: 0.7292 - 2s/epoch - 9ms/step
Loss: 0.5704286694526672, Accuracy: 0.7292128205299377
```

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.
nn = tf.keras.models.Sequential()

input_dims = len(X_train[0])

# First hidden layer
nn.add(tf.keras.layers.Dense(units=80, activation="relu", input_dim = input_dims))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=40, activation="relu"))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=20, activation="relu"))

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

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

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

 dense_1 (Dense)             (None, 40)                3240

 dense_2 (Dense)             (None, 20)                820

 dense_3 (Dense)             (None, 1)                 21

=================================================================
Total params: 7601 (29.69 KB)
Trainable params: 7601 (29.69 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Alphabet Soup Analysis Results

Compiling, Training, and Evaluating the Model

- I did 5 optimizations to get to the target of over 75% accuracy
  - **Optimization 2** – increased the nodes and the epochs
    - Dropped EIN and Name Columns
    - 1st layer – 100 neurons with ReLU activation
    - 2nd layer – 50 neurons with ReLU activation
    - 3rd layer – 25 neurons with ReLU activation
    - Output layer – 1 neuron with Sigmoid activation
    - 150 epochs
    - 72.96% Accuracy – slightly more accurate

```
Epoch 146/150
804/804 [==============================] - 2s 2ms/step - loss: 0.5313 - accuracy: 0.7411
Epoch 147/150
804/804 [==============================] - 3s 4ms/step - loss: 0.5314 - accuracy: 0.7410
Epoch 148/150
804/804 [==============================] - 3s 4ms/step - loss: 0.5312 - accuracy: 0.7406
Epoch 149/150
804/804 [==============================] - 3s 3ms/step - loss: 0.5309 - accuracy: 0.7406
Epoch 150/150
804/804 [==============================] - 2s 3ms/step - loss: 0.5314 - accuracy: 0.7415
<keras.src.callbacks.History at 0x796f0b6a4df0>
```

```
[ ]  # 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: 0.5792 - accuracy: 0.7296 - 519ms/epoch - 2ms/step
     Loss: 0.5792485475540161, Accuracy: 0.7295626997947693
```

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.
     nn = tf.keras.models.Sequential()

     input_dims = len(X_train[0])

     # First hidden layer
     nn.add(tf.keras.layers.Dense(units=100, activation="relu", input_dim = input_dims))

     # Second hidden layer
     nn.add(tf.keras.layers.Dense(units=50, activation="relu"))

     # Third hidden layer
     nn.add(tf.keras.layers.Dense(units=25, activation="relu"))

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

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               4400

 dense_1 (Dense)             (None, 50)                5050

 dense_2 (Dense)             (None, 25)                1275

 dense_3 (Dense)             (None, 1)                 26

=================================================================
Total params: 10751 (42.00 KB)
Trainable params: 10751 (42.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Alphabet Soup Analysis Results

Compiling, Training, and Evaluating the Model

- I did 5 optimizations to get to the target of over 75% accuracy
  - **Optimization 3** – adjusted to fewer bins and reduced the epochs back to 100
    - Dropped EIN and Name Columns
    - Changed app count cut off from 500 to 50
    - Changed class count cut off from 1000 to 100
    - 1st layer – 100 neurons with ReLU activation
    - 2nd layer – 50 neurons with ReLU activation
    - 3rd layer – 25 neurons with ReLU activation
    - Output layer – 1 neuron with Sigmoid activation
    - 100 epochs
    - 72.91% Accuracy - slightly less than Optimization 2

```
Epoch 96/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5292 - accuracy: 0.7411
Epoch 97/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5290 - accuracy: 0.7409
Epoch 98/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5294 - accuracy: 0.7416
Epoch 99/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5288 - accuracy: 0.7416
Epoch 100/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5290 - accuracy: 0.7414
<keras.src.callbacks.History at 0x7e8894147640>
```

```
[ ]  # 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: 0.5770 - accuracy: 0.7291 - 532ms/epoch - 2ms/step
     Loss: 0.5769652724266052, Accuracy: 0.7290962338447571
```

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.
     nn = tf.keras.models.Sequential()

     input_dims = len(X_train[0])

     # First hidden layer
     nn.add(tf.keras.layers.Dense(units=100, activation="relu", input_dim = input_dims))

     # Second hidden layer
     nn.add(tf.keras.layers.Dense(units=50, activation="relu"))

     # Third hidden layer
     nn.add(tf.keras.layers.Dense(units=25, activation="relu"))

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

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               5200

 dense_1 (Dense)             (None, 50)                5050

 dense_2 (Dense)             (None, 25)                1275

 dense_3 (Dense)             (None, 1)                 26

=================================================================
Total params: 11551 (45.12 KB)
Trainable params: 11551 (45.12 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Alphabet Soup Analysis Results

Compiling, Training, and Evaluating the Model

- I did 5 optimizations to get to the target of over 75% accuracy
  - **Optimization 4** – created more bins
    - Dropped EIN and Name Columns
    - Changed app count cut-off from 500 to 1000
    - Changed class count cut-off from 1000 to 2000
    - 1st layer – 100 neurons with ReLU activation
    - 2nd layer – 50 neurons with ReLU activation
    - 3rd layer – 25 neurons with ReLU activation
    - Output layer – 1 neuron with Sigmoid activation
    - 100 epochs
    - 72.75% Accuracy – slightly less accuracy

```
Epoch 96/100
804/804 [==============================] - 3s 3ms/step - loss: 0.5423 - accuracy: 0.7371
Epoch 97/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5417 - accuracy: 0.7368
Epoch 98/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5417 - accuracy: 0.7373
Epoch 99/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5416 - accuracy: 0.7376
Epoch 100/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5419 - accuracy: 0.7376
<keras.src.callbacks.History at 0x787dc6de24d0>
```

```
[ ] # 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: 0.5826 - accuracy: 0.7275 - 523ms/epoch - 2ms/step
Loss: 0.5825673937797546, Accuracy: 0.7274635434150696
```

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.
    nn = tf.keras.models.Sequential()

    input_dims = len(X_train[0])

    # First hidden layer
    nn.add(tf.keras.layers.Dense(units=100, activation="relu", input_dim = input_dims))

    # Second hidden layer
    nn.add(tf.keras.layers.Dense(units=50, activation="relu"))

    # Third hidden layer
    nn.add(tf.keras.layers.Dense(units=25, activation="relu"))

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

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

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               3900

 dense_1 (Dense)             (None, 50)                5050

 dense_2 (Dense)             (None, 25)                1275

 dense_3 (Dense)             (None, 1)                 26

=================================================================
Total params: 10251 (40.04 KB)
Trainable params: 10251 (40.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Alphabet Soup Analysis Results

Compiling, Training, and Evaluating the Model

- The 5th Optimization was successful with almost 79% accuracy
- Adding an additional layer and increasing the neurons improved the accuracy over 20% from 51% to almost 73%.
- Including the NAME data and binning this data improved the accuracy from almost 73% to almost 79% to reach the target of over 75% accuracy.
    - **Optimization 5** – Did not drop the NAME column and binned the NAME data
        - Dropped EIN Column
        - Changed app count cut-off back to 500
        - Changed class count cut-off back to 1000
        - 1st layer – 100 neurons with ReLU activation
        - 2nd layer – 50 neurons with ReLU activation
        - 3rd layer – 25 neurons with ReLU activation
        - Output layer – 1 neuron with Sigmoid activation
        - 100 epochs
        - 78.95% Accuracy achieved to reach the target accuracy

```
Epoch 96/100
804/804 [==============================] - 3s 3ms/step - loss: 0.3993 - accuracy: 0.8082
Epoch 97/100
804/804 [==============================] - 4s 4ms/step - loss: 0.3997 - accuracy: 0.8080
Epoch 98/100
804/804 [==============================] - 2s 3ms/step - loss: 0.3995 - accuracy: 0.8083
Epoch 99/100
804/804 [==============================] - 2s 3ms/step - loss: 0.4000 - accuracy: 0.8093
Epoch 100/100
```

```
[21] # 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: 0.5024 - accuracy: 0.7895 - 729ms/epoch - 3ms/step
     Loss: 0.5024434328079224, Accuracy: 0.78950434923172
```

Compile, Train and Evaluate the Model

```
[18] # Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
     nn = tf.keras.models.Sequential()

     input_dims = len(X_train[0])

     # First hidden layer
     nn.add(tf.keras.layers.Dense(units=100, activation="relu", input_dim = input_dims))

     # Second hidden layer
     nn.add(tf.keras.layers.Dense(units=50, activation="relu"))

     # Third hidden layer
     nn.add(tf.keras.layers.Dense(units=25, activation="relu"))

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

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               39900

 dense_1 (Dense)             (None, 50)                5050

 dense_2 (Dense)             (None, 25)                1275

 dense_3 (Dense)             (None, 1)                 26

=================================================================
Total params: 46251 (180.67 KB)
Trainable params: 46251 (180.67 KB)
Non-trainable params: 0 (0.00 Byte)
```

0s completed at 6:5

Alphabet Soup Analysis Results Summary

Overall, by using machine learning, we can predict with almost 79% accuracy which campaigns will be successful. This will help us fund campaigns more efficiently.
It would be ideal to continue to try other models to improve the accuracy even more.
The greatest gains in accuracy were gained from adding layers and neurons and including the NAME data. Continuing to adjust the layers, neurons and binning could further increase the prediction accuracy.
The Tanh model could also be used to try to increase the accuracy since it will normalize the data.
The Random Forest Model can also be used to create a decision tree to find the predictive variables. Explain why…