Brian Morrissey

Module 21 Report

Overview

The purpose of this analysis is to determine whether a binary classifier can predict the success of applicants funded by Alphabet Soup.

Results

- The target variable for the model is the IS_SUCCESSFUL column
- The variables featured in the model are EIN, NAME, APPLICATION_TYPE, AFFILIATION, CLASSIFICATION, USE_CASE, ORGANIZATION, STATUS, INCOME_AMT, SPECIAL_CONSIDERATIONS, and ASK_AMT
- The variables that were removed are EIN and NAME
- I used 2 hidden layers with 100 neurons each with a ReLU activation function. I found that adding more layers did not have a positive impact on the model. I used the ReLU activation function because research showed that it achieves better performance than other activation functions.
- I was not able to achieve the target model performance. The closest I could get was 73%
- In my attempts to increase model performance, I tried increasing and decreasing the hidden layers and number of neurons. I found that the model did not improve, and in some cases performed worse.

ATTEMPT 1:

I changed the activation functions to “Sigmoid”. The outcome was .726

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

# First hidden layer
nn.add(tf.keras.layers.Dense(units=100, activation="sigmoid", input_dim=len(converted_data.columns)-1))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=100, activation="sigmoid"))

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

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

```
268/268 - 0s - loss: 0.5595 - accuracy: 0.7263 - 461ms/epoch - 2ms/step
Loss: 0.5595492720603943, Accuracy: 0.7262973785400391
```

ATTEMPT 2:

I added two more hidden layers. The outcome was .724

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

# First hidden layer
nn.add(tf.keras.layers.Dense(units=100, activation="relu", input_dim=len(converted_data.columns)-1))

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

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

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

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

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

```
268/268 - 0s - loss: 0.5636 - accuracy: 0.7248 - 483ms/epoch - 2ms/step
Loss: 0.5635740756988525, Accuracy: 0.724781334400177
```

ATTEMPT 3:

For my final attempt I removed a hidden layer and adjusted the unit numbers. The outcome was .729

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

# First hidden layer
nn.add(tf.keras.layers.Dense(units=60, activation="relu", input_dim=len(converted_data.columns)-1))

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

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

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

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

 # Summary

The deep learning model proved to not be as successful as I had hoped it would be. To improve the model I attempted to use keras_tuner, but this led to issues. I would suggest using a different type of classification model altogether.