

Alphabet Soup Funding Predictor

Module 21 Challenge

Zachary Mason

Background and Overview

The nonprofit foundation Alphabet Soup wants a tool that can help it select the applicants for funding with the best chance of success in their ventures. With my knowledge of machine learning and neural networks, I used the features in the provided dataset to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup.

From Alphabet Soup's business team, we have received a CSV containing more than 34,000 organizations that have received funding from Alphabet Soup 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

Results

The first step of the process involved reading in the data from the csv file and beginning to process it to get it ready for the machine learning model. After processing the data, I compiled it together and developed a neural network, or deep learning model, to create a binary classification model that can predict if a funded organization will be successful based on the features in the dataset.

Data Preprocessing:

- Target Variable:
 - 'IS_SUCCESSFUL'

- Indicates whether the funding was used successfully
- Feature Variables
 - 'APPLICATION_TYPE'
 - 'AFFILIATION'
 - 'CLASSIFICATION'
 - 'USE_CASE'
 - 'ORGANIZATION'
 - 'STATUS'
 - 'INCOME_AMT'
 - 'SPECIAL_CONSIDERATIONS'
 - 'ASK_AMT'
- Variable(s) to be removed from the input data because they are neither targets nor features:
 - The 'EIN' and 'NAME' columns were removed from the input data initially as they serve as identification columns and do not provide predictive value.
- The data was split into training and testing sets and scaled using 'StandardScaler'
- Variables were encoded using 'pd.get_dummies'

Compiling, Training, and Evaluating the Model

- The initial model did not achieve the target accuracy of 75%. In the initial model, the number of neurons, layers, and activation function were utilized to get a base sense of how well the model would perform. In the next section, the selection of those factors were manipulated in or to reach our target goal.

Optimization Steps

1. Increased, decreased the number of neurons on each layer.
2. Added and then later removed additional hidden layers after evaluation.
3. Experimented with different activation functions for the hidden layers.
4. Adjusted the number of epochs to ensure the model was trained adequately.

Summary

After the first few attempts at optimizing the model failed to reach the target goal of 75%, only adjusting the neurons, activator, and layers, I decided to go another route. This time around ('AlphabetSoupCharity_Optimization_4'), in the preprocessing stage I decided to keep the 'NAME' variable as a feature and altered/binning the name of any company with less than 5 instances as "Other." Once that was added back in, we were able to achieve an accuracy of 79% and loss of 45% where the accuracy was roughly 4% higher than the goal.

One option to further improve the model would be to use Keras Tuner to automate the optimization of the deep learning model. This will help one select the best model configuration by searching through for the most ideal combination of parameters (i.e. layers, neurons, activation functions, etc..).

AlphabetSoupCharity_Optimization_4

```
[ ] # value counts of names
    name_count = application_df['NAME'].value_counts()
    name_count
```

```
[ ] # Choose a cutoff value and create a list of names to be replaced
    # variable name names_replace

    cutoff_class = 4
    names_replace = list(name_count[name_count <= cutoff_class].index)

    # Replace in dataframe
    for name in names_replace:
        application_df['NAME'] = application_df['NAME'].replace(name, "Other")

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

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
# activation type change, add a layer, adjust nodes
number_input_features = len(X_train_scaled[0])
hidden_nodes_layer1 = 32
hidden_nodes_layer2 = 16


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()
```

```
 # Train the model
fit_model = nn.fit(X_train_scaled, y_train, epochs=100, batch_size=32, callbacks=[checkpoint_callback])
```

 Show hidden output

```
 # 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 - 2ms/step - accuracy: 0.7909 - loss: 0.4508
Loss: 0.45075860619544983, Accuracy: 0.7909038066864014