

Alphabet Soup Charity Funding Predictor Analysis

Overview: The nonprofit foundation Alphabet Soup wants to develop an algorithm that predicts whether a grant applicant will be accepted. Armed with our knowledge of machine learning and neural networks, we leveraged the capabilities of the datasets provided to develop a binary classifier that could predict whether an applicant would pass if funded by Alphabet Soup. must be created.

Results: I removed all irrelevant information to start processing the data. After omitting EIN and NAME, the remaining columns are considered features of the model. This name was reinserted into his second test for binning purposes. The data was then split into a training set and a test set. The model's target variable is labeled "IS_SUCCESSFUL" and has a value of 1 for yes and 0 for no. APPLICATION data was analyzed and CLASSIFICATION values were used for classification. Multiple data points were used as a cutoff, and 'rare' variables were grouped by unique value with the new value 'other'. Categorical variables were encoded by `get_dummies()` after checking if the classification was successful.

Compiling, Training, and Evaluating the Model: After applying the neural network, each model now has a total of three layers. The number of hidden nodes is determined by the number of features.

```
# 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_scaled[0])
hidden_nodes_layer1=7
hidden_nodes_layer2=14
hidden_nodes_layer3=21
nn = tf.keras.models.Sequential()

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()
```

The 477 parameters were created by a 3-layer training model. The first trial was slightly above 73% accuracy and below the target of 75%, but not too far.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 7)	3171
dense_1 (Dense)	(None, 14)	112
dense_2 (Dense)	(None, 1)	15
Total params: 3,298		
Trainable params: 3,298		
Non-trainable params: 0		

```
# 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 - loss: 0.4677 - accuracy: 0.7846 - 351ms/epoch - 1ms/step
Loss: 0.4677017331123352, Accuracy: 0.784606397151947
```

Optimization: A second trial on the 'NAME' column of the dataset achieved an accuracy of almost 79%. This is 4% above the target of 75% for 3,298 parameters.

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 7)	3171
dense_4 (Dense)	(None, 14)	112
dense_5 (Dense)	(None, 1)	15
Total params: 3,298		
Trainable params: 3,298		
Non-trainable params: 0		

Deep learning models must use multiple layers to learn to predict and classify information based on computer filtering between layers.

```
[22] # 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 - loss: 0.4677 - accuracy: 0.7846 - 351ms/epoch - 1ms/step
Loss: 0.4677017331123352, Accuracy: 0.784606397151947
```