Charity Funding Predictor

Overview

In this project I will create an algorithm to help The nonprofit foundation Alphabet Soup that predicts whether or not applicants for funding will be successful. In order to do that I used the dataset of 34,000 organizations receiving fundings and created a binary classifier for predictions.

Data Processing

EIN and NAME have been removed from the model. CLASSIFICATION value count have been used for binning. I have used “IS_SUCCESSFUL” as target variable. Then data have been split into features and target arrays

```
[ ]  # Split our preprocessed data into our features and target arrays
     y = application_df['IS_SUCCESSFUL'].values
     y

     array([1, 1, 0, ..., 0, 1, 0])
```

```
[ ]  # drop 'IS_SUCCESSFUL' column
     X = application_df.drop('IS_SUCCESSFUL', axis=1).values
     X

     array([[1.0000000e+00, 5.0000000e+03, 0.0000000e+00, ..., 0.0000000e+00,
             1.0000000e+00, 0.0000000e+00],
            [1.0000000e+00, 1.0859000e+05, 0.0000000e+00, ..., 0.0000000e+00,
             1.0000000e+00, 0.0000000e+00],
            [1.0000000e+00, 5.0000000e+03, 0.0000000e+00, ..., 0.0000000e+00,
             1.0000000e+00, 0.0000000e+00],
            ...,
            [1.0000000e+00, 5.0000000e+03, 0.0000000e+00, ..., 0.0000000e+00,
             1.0000000e+00, 0.0000000e+00],
            [1.0000000e+00, 5.0000000e+03, 0.0000000e+00, ..., 0.0000000e+00,
             1.0000000e+00, 0.0000000e+00],
            [1.0000000e+00, 3.6500179e+07, 0.0000000e+00, ..., 0.0000000e+00,
             1.0000000e+00, 0.0000000e+00]])
```

Compile, Train and Evaluate the Model

```
nn.summary()

Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 7)                 504

 dense_7 (Dense)             (None, 14)                112

 dense_8 (Dense)             (None, 1)                 15

=================================================================
Total params: 631
Trainable params: 631
Non-trainable params: 0
```

```
# Train the model
fit_model = nn.fit(X_train_scaled,y_train,validation_split=0.15, epochs=100)
```

```
Epoch 1/100
684/684 [==============================] - 2s 2ms/step - loss: 0.6160 - accuracy: 0.6820 - val_loss: 0.5627 - val_accuracy: 0.7357
Epoch 2/100
684/684 [==============================] - 1s 2ms/step - loss: 0.5688 - accuracy: 0.7225 - val_loss: 0.5491 - val_accuracy: 0.7388
Epoch 3/100
684/684 [==============================] - 1s 2ms/step - loss: 0.5595 - accuracy: 0.7254 - val_loss: 0.5468 - val_accuracy: 0.7396
Epoch 4/100
684/684 [==============================] - 1s 2ms/step - loss: 0.5566 - accuracy: 0.7263 - val_loss: 0.5450 - val_accuracy: 0.7396
Epoch 5/100
684/684 [==============================] - 1s 2ms/step - loss: 0.5545 - accuracy: 0.7282 - val_loss: 0.5453 - val_accuracy: 0.7406
Epoch 6/100
```

```
# 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: 0.5550 - accuracy: 0.7297 - 268ms/epoch - 1000us/step
Loss: 0.5549996495246887, Accuracy: 0.72967928647995
```

As we see accuracy is 72% which is not desirable. Therefore we need to take another step and do optimization.

This time, I have kept “NAME” column and only dropped “EIN”. I have used “NAME” value count for binning. I then used Application type value count for binning.

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

```
T3       27037
T4        1542
T6        1216
T5        1173
T19       1065
T8         737
T7         725
T10        528
Other      276
Name: APPLICATION_TYPE, dtype: int64
```

```
[ ] # You may find it helpful to look at CLASSIFICATION value counts >1
    c_counts = application_df['CLASSIFICATION'].value_counts()

    countsclassification = c_counts[c_counts>1]
    countsclassification
```

```
C1000    17326
C2000     6074
C1200     4837
C3000     1918
C2100     1883
C7000      777
C1700      287
C4000      194
```

Then I followed the same steps that I took previously

```python
# Split our preprocessed data into our features and target arrays
y = application_df['IS_SUCCESSFUL'].values
y
```

```
array([1, 1, 0, ..., 0, 1, 0])
```

```python
# drop 'IS_SUCCESSFUL' column
X = application_df.drop('IS_SUCCESSFUL', axis=1).values
X
```

```
array([[1.0000000e+00, 5.0000000e+03, 0.0000000e+00, ..., 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       [1.0000000e+00, 1.0859000e+05, 0.0000000e+00, ..., 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       [1.0000000e+00, 5.0000000e+03, 0.0000000e+00, ..., 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       ...,
       [1.0000000e+00, 5.0000000e+03, 0.0000000e+00, ..., 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       [1.0000000e+00, 5.0000000e+03, 0.0000000e+00, ..., 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       [1.0000000e+00, 3.6500179e+07, 0.0000000e+00, ..., 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00]])
```

```python
# Split the preprocessed data into a training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state = 42)
```

And after compile, and train,

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 1911

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 1)                 15

=================================================================
Total params: 2,038
Trainable params: 2,038
Non-trainable params: 0
```

I got the following evaluation:

```python
# 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: 0.4731 - accuracy: 0.7789 - 314ms/epoch - 1ms/step
Loss: 0.47311681509017944, Accuracy: 0.7788920998573303
```

Optimized data gave us 77% accuracy.