Deep Learning Challenge

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 your knowledge of machine learning and neural networks, you'll use the features in the provided dataset to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup.

Results

STARTER CODE DATA

- In the beginning, we started by looking at the precision of our models' capability to predict whether applicants will be successful if funded by binning the data by the columns "APPLICATION_TYPE" and "CLASSIFICATION". Binning is what allows us to reduce the complexity of continuous data or helps to group similar values together to simplify our data. After successfully binning, splitting, and scaling the data, everything was then compiled, trained, and evaluated to see how good our model was at predicting the success of the funding. For the starter code data, the model was around 72.42% accurate.

OPTIMIZATION

The goal of optimizing the model was to reach an accuracy rate of at least 75%. After six attempts by restructuring the data, the highest accuracy rate I was able to attain was 72.65%. I will explain my sixth attempt in more detail in that section of the analysis.

Attempt 1:

Dropped EIN and NAME columns, then binned and restructured USE_CASE and OPTIMIZATION columns.

Accuracy rate: 72.65%

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn_model.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 1s - loss: 0.5592 - accuracy: 0.7265 - 1s/epoch - 4ms/step
Loss: 0.559190571308136, Accuracy: 0.7265306115150452
```

Attempt 2:

Dropped EIN and NAME columns, then binned and restructured AFFILIATION and INCOME_AMT columns.

Accuracy rate: 72.46%

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn_model.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 0s - loss: 0.5627 - accuracy: 0.7247 - 444ms/epoch - 2ms/step
Loss: 0.5626847147941589, Accuracy: 0.7246647477149963
```

Attempt 3:

Dropped EIN and NAME columns, then binned and restructured AFFILIATION, ORGANIZATION, and INCOME_AMT columns.

Accuracy rate: 72.53%

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn_model.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 0s - loss: 0.5617 - accuracy: 0.7254 - 456ms/epoch - 2ms/step
Loss: 0.5616775155067444, Accuracy: 0.7253644466400146
```

Attempt 4:

Dropped EIN and NAME columns, then binned and restructured AFFILIATION and ORGANIZATION columns. I tried to up the neurons to 100, 50, 1 to see if this would increase my accuracy by more than one percent, to no avail.

Accuracy rate: 72.6%

Attempt 5:

Dropped EIN and NAME columns, then binned and restructured APPLICATION_TYPE, CLASSIFICATION, and INCOME_AMT columns.

Accuracy rate: 72.49%

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn_model.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 0s - loss: 0.5602 - accuracy: 0.7249 - 438ms/epoch - 2ms/step
Loss: 0.5601752400398254, Accuracy: 0.7248979806900024
```

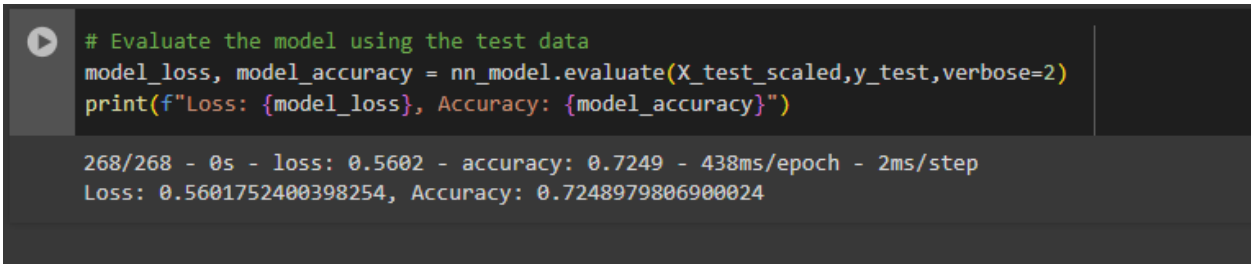
Attempt 6:

After the fifth attempt, I decided to try the original code (binning and restructuring by the APPLICATION_TYPE and CLASSIFICATION columns) and to also bin by USE_CASE, since that was one of my attempts highest accuracy percentages.

- Target variable: IS_SUCCESSFUL
- Feature variables: All columns except IS_SUCCESSFUL
- Variables removed: EIN and NAMES columns were removed because they weren't beneficial to the data that I was looking at

After binning and restructuring the data, I created the predictor model to try and get the accuracy rating of 75% or higher. To do so, I included the following:

- The neuron quantities were kept the same (80, 30, 1) to prevent the model from overfitting the data. An extra hidden layer was added – the function used for this layer was tanh. Tanh was chosen to see if there could be underlying complex, nonlinear patterns in the data.
- I was unsuccessful after six attempts to reach the goal of 75% or more accuracy.
- It can be seen through the snips provided that I tried many different ways to increase the accuracy of the model. I changed the neurons, hidden layers, binning of columns, and activation functions in order to try and obtain the 75% accuracy, but I was never able to obtain a higher rating than 72.65% in the first optimization chosen.



```
# Evaluate the model using the test data
model_loss, model_accuracy = nn_model.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 0s - loss: 0.5602 - accuracy: 0.7249 - 438ms/epoch - 2ms/step
Loss: 0.5601752400398254, Accuracy: 0.7248979806900024
```

Summary

Over all, the model was consistently working within a 72% accuracy, three percent lower than what we wanted. If I had more time, I would continue to try and add layers, change neurons, and bin different columns in order to see if the accuracy rating would go up. Adding these layers helps the model be able to predict information by adding different filtering and layering techniques.