Report on the Neural Network Model

Charity Funding Predictor

Overview:

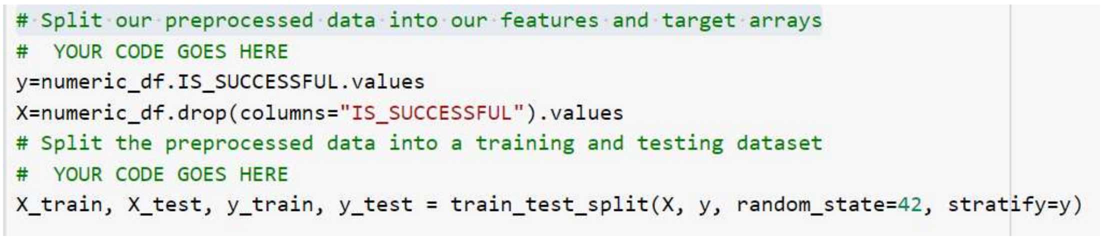
The purpose of this analysis is to create a tool to help the nonprofit foundation Alphabet Soup select the applicants for funding with the best chance of success in their ventures. With the knowledge of machine learning and neural networks, I have used the features in the provided dataset to create a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

From Alphabet Soup's business team, I 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 consideration for application
- **ASK_AMT**—Funding amount requested
- **IS_SUCCESSFUL**—Was the money used effectively

Results

- **Data Processing**
  - Drop the non-beneficial ID columns, 'EIN' and 'NAME'. These two variables should be removed because they are neither targets nor features.
  - Check the number of unique values in each column: application_df.nunique()
  - Choose a cutoff value and create a list of application types /classifications to be replaced
  - Convert categorical data to numeric with `pd.get_dummies`
  - Split our preprocessed data into our features and target arrays: the variable "IS_SUCCESSFUl" is the target, and other variables are the features.



```
# Split our preprocessed data into our features and target arrays
#  YOUR CODE GOES HERE
y=numeric_df.IS_SUCCESSFUL.values
X=numeric_df.drop(columns="IS_SUCCESSFUL").values
# Split the preprocessed data into a training and testing dataset
#  YOUR CODE GOES HERE
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, stratify=y)
```

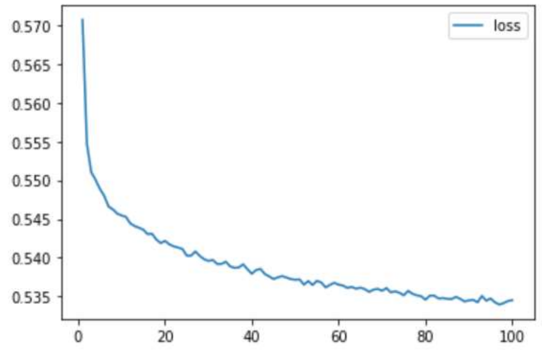
- **Compiling, Training, and Evaluating the Model**
  - **The first model** I built uses the following parameters with low computation time in mind:
    - ❖ Two hidden layers with 80 and 30 neurons, the activation function of relu for the two hidden layers
    - ❖
    - ❖ The output layer activation function of sigmoid as the model output with 1 node is binary classification between 0 and 1.
    - ❖ The model performance is with the accuracy of 72.68%. The loss value of 56% indicates that the model can be further optimized.

```
# 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: 0.5600 - accuracy: 0.7268 - 772ms/epoch - 3ms/step
Loss: 0.5600056648254395, Accuracy: 0.7267638444900513
```

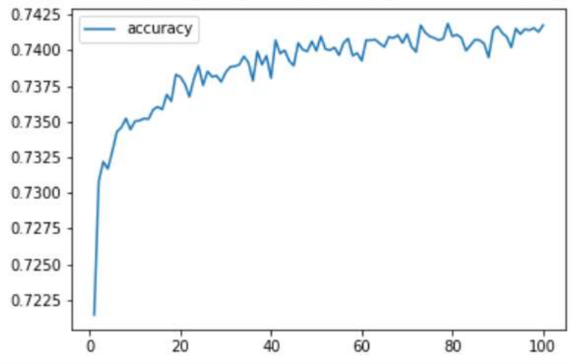
```
# Plot the loss
history_df.plot(y="loss")
```

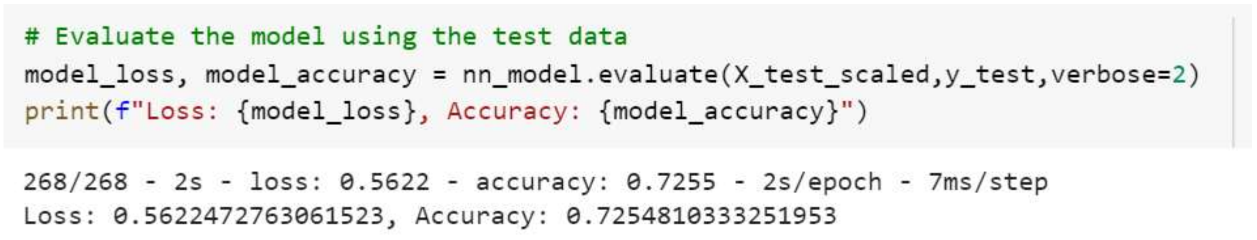
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe3ab6b3650>
```



```
# Plot the accuracy
history_df.plot(y="accuracy")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe3ab52ba10>
```

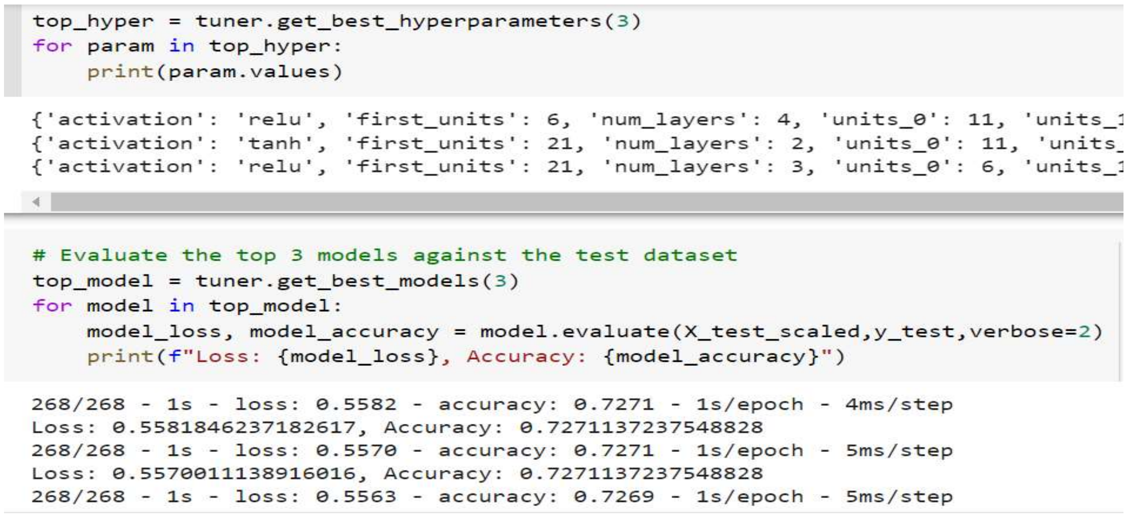


- **The Attempts of Three Optimization**
  - The First Attempt to Optimize the Model
    - Drop the non-beneficial ID columns, 'EIN','NAME',and some other columns,'SPECIAL_CONSIDERATIONS' and 'STATUS'
    - Reduce the number of epochs to 50
    - layer1=80: activation function="relu"
    - layer2=30: activation function=tanh
    - the performance of the model shows the accuracy is 72.54%, and the loss value of 56 indicates that the model can be further optimized.
  - 

```
# 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 - 2s - loss: 0.5622 - accuracy: 0.7255 - 2s/epoch - 7ms/step
Loss: 0.5622472763061523, Accuracy: 0.7254810333251953
```

  - The Second Attempt to Optimize the Model: Use KerasTuner to create and compile a new Sequential deep learning model with hyperparameter options.
    - Drop the non-beneficial ID columns, 'EIN','NAME',and some other columns,'SPECIAL_CONSIDERATIONS' and 'STATUS'.
    - Allow KerasTuner to select between ReLU and tanh activation functions for each hidden layer: activation = hp.Choice('activation',['relu','tanh'])
    - Allow KerasTuner to decide from 1 to 30 neurons in the first dense layer. Note: To limit the tuner runtime, increase your step argument to at least 5.
    - Allow KerasTuner to decide from 1 to 5 hidden layers and 1 to 30 neurons in each Dense layer.
    - Run the kerasturner search for best hyperparameters
    - Import the KerasTuner library and create a Hyperband tuner instance. Use the following parameters in the tuner:
      - ✓ The objective is "val_accuracy"
      - ✓ max_epochs equal to 20
      - ✓ hyperband_iterations equal to two.
      - ✓ Run the KerasTuner search for best hyperparameters over 20 epochs.
    - Retrieve the top three model hyperparameters from the tuner search and print the values. Retrieve the top three models from the tuner search and compare their predictive accuracy against the test dataset.
    - Results:
      The top 1 model: the activation function is relu, numer of hidden layers is 4 and the accuracy is 72.72%. The loss value of 55.8 indicates this model needs to be further optimized.



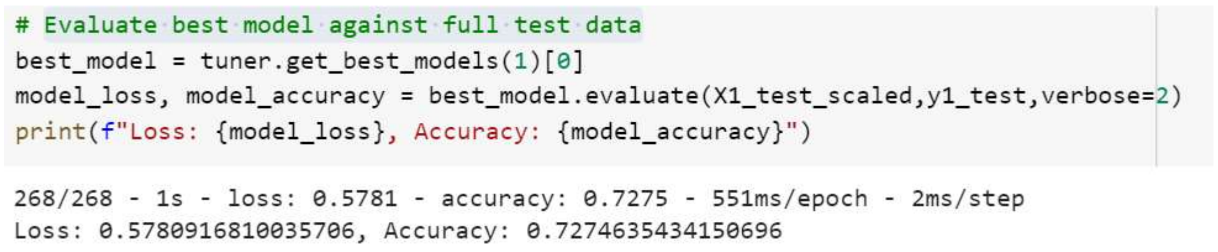
```
top_hyper = tuner.get_best_hyperparameters(3)
for param in top_hyper:
    print(param.values)
```

```
{'activation': 'relu', 'first_units': 6, 'num_layers': 4, 'units_0': 11, 'units_1
{'activation': 'tanh', 'first_units': 21, 'num_layers': 2, 'units_0': 11, 'units_
{'activation': 'relu', 'first_units': 21, 'num_layers': 3, 'units_0': 6, 'units_1
```

```
# Evaluate the top 3 models against the test dataset
top_model = tuner.get_best_models(3)
for model in top_model:
    model_loss, model_accuracy = model.evaluate(X_test_scaled,y_test,verbose=2)
    print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 1s - loss: 0.5582 - accuracy: 0.7271 - 1s/epoch - 4ms/step
Loss: 0.5581846237182617, Accuracy: 0.7271137237548828
268/268 - 1s - loss: 0.5570 - accuracy: 0.7271 - 1s/epoch - 5ms/step
Loss: 0.5570011138916016, Accuracy: 0.7271137237548828
268/268 - 1s - loss: 0.5563 - accuracy: 0.7269 - 1s/epoch - 5ms/step
```

- The Third Attempt to Optimize the Model: Use KerasTuner to create and compile a new Sequential deep learning model with hyperparameter options.
    - ❖ Drop the non-beneficial ID columns, 'EIN','NAME',and some other columns,'SPECIAL_CONSIDERATIONS' and 'STATUS'.
    - ❖ Allow KerasTuner to select among ReLU, tanh and sigmoid activation functions for each hidden layer: activation = hp.Choice('activation',['relu','tanh','sigmoid'])
    - ❖ Allow KerasTuner to decide from 1 to 10 neurons in the first dense layer. Note: To limit the tuner runtime, increase your step argument to at least 2.
    - ❖ Allow KerasTuner to decide from 1 to 6 hidden layers and 1 to 10 neurons in each Dense layer.
    - ❖ Run the kersturner search for best hyperparameters
    - ❖ Import the KerasTuner library and create a Hyperband tuner instance. Use the following parameters in the tuner:
        - ✓ The objective is "val_accuracy"
        - ✓ max_epochs equal to 20
        - ✓ hyperband_iterations equal to two.
        - ✓ Run the KerasTuner search for best hyperparameters over 20 epochs.
    - ❖ Get best model hyperparameters and Evaluate best model against full test data
    - ❖ Results:
        - ✓ The best model hyperparameter: the activation function is sigmoid, number of hidden layers is 2.
        - ✓ The best model: the accuracy is 72.75%. The loss value of 57.8 indicates this model needs to be further optimized.



```
# Evaluate best model against full test data
best_model = tuner.get_best_models(1)[0]
model_loss, model_accuracy = best_model.evaluate(X1_test_scaled,y1_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 1s - loss: 0.5781 - accuracy: 0.7275 - 551ms/epoch - 2ms/step
Loss: 0.5780916810035706, Accuracy: 0.7274635434150696
```

Summary

Like the first and second model, the automatically two optimized neural network models did not achieve the 75% of accuracy. In the three attempts of optimization, I tried to reduce some input features and adjust the echo number, activation functions and layers. The model can be further improved by doing cross-validation, feature engineering, trying out more advanced machine learning algorithms, or changing the arguments in the deep learning network we built above. Performance may be improved by using other machine learning models to solve this classification problems. For example, Random Forest Classifier and Logistic Regression may be able to perform well in predicting the outcome of the target variable: whether applicants will be successful if funded by Alphabet soup.