8/29/2023

Alphabet Soup Neural Network Model Analysis

Overview

This project was tasked with developing and implementing a Neural Network model, which would aid Alphabet Soup in selecting funding applicants based on potential success. The model was based on a historical dataset comprised of over 34,000 entries containing metadata information from previous funding recipients. The goal was to achieve 75% accuracy in identifying successful applicants. Ultimately, we were not able to achieve accuracy over 73% after multiple modeling attempts.

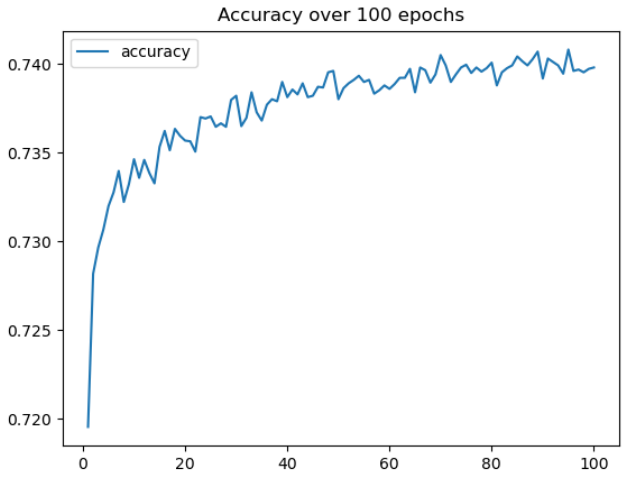
Results

This project was built in a Jupyter Notebook using Python and TensorFlow/Keras for modeling. Python was used for data cleaning and organization prior to modeling and SciKit Learn was used to prepare, scale and split the data for the model.

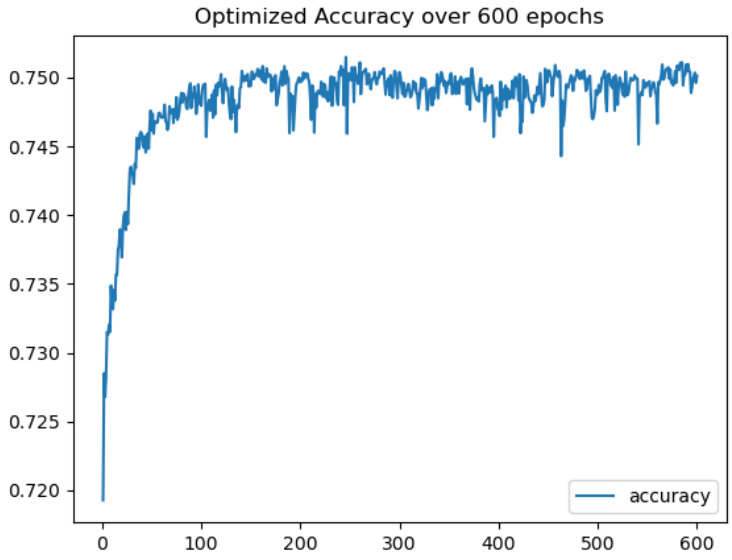
- **Data Processing**
  - Target: We used the "IS_SUCCESSFUL" field from the dataset as our target. This variable was split off from our dataset and used to train and test our model.
  - Features: We used Application Type, Affiliation, Classification, Use Case, Organization, Income Classification, Special Considerations and Ask Amount for features. We believe that any of these could play into the success or failure of a project, so we maintained them in the testing and training process.
  - Omitted: We did not include the EIN, Name or Status in our modeling. EIN and Name would have hindered training since they are unique to each column could not be used to predict future decisions. Status was dropped in optimization attempts since it did not have a positive impact on accuracy score.

- **Compiling, Training and Evaluating the Model**
  - We used a Tensor Flow Keras deep neural network for our modeling.
  - Our initial attempt used 2 hidden layers, one with 80 neurons and one with 30. Both layers used the most recommended "Relu" activation functions.
  - We used a "Sigmoid" activation on the output layer, since this is a classification model.
  - We used standard loss and optimizations for our compiler and tested on accuracy.
  - We ran our initial model for 100 epochs.

- Our initial model tested at **72.94% Accuracy** when weighed against the testing data.
- Training accuracy below:



- We attempted several optimizations including increasing layers, neurons per layer, and epochs. We also tried to divide the Ask Amounts into bins since they are so widely dispersed. While our optimizations were able to reach 75% accuracy in training, they still scored below that number in testing, suggesting that the extra computation and complexity that we were applying was overfitting our model, causing it to perform better on training data but unreliably on testing data and future datasets.
- Training accuracy below:



Summary

We were not able to reach the target of 75% accuracy with our models. Since our accuracy was just short of the target, the model my provide some value as is or there are several other options that could be tested.

- **Continue to refine model**: More work could be done to resize bins, omit columns and attempting varying levels of complexity. Since our initial models achieved close to the target accuracy, it may be possible to reach that with several more attempts. The risk of this is overfitting our model, creating something that is not reliable in the future.
- **Assess Features:** Upon initial inspection, it was difficult to tell how important some of the features such as Classification or Use Case were to outcome or other decision making. Attempts to omit certain features in this model did not yield results but separate logistic regression model weighing each feature's impact on success may help determine what should be included in a future model.
- **Employ other Algorithms:** We could attempt to use Logistic Regression or other Classification Algorithms to achieve better results. The Logistic Regression model is simpler to set up and very efficient so it would not be difficult to test it as an alternative.