Alphabet Soup Charity Funding Analysis and Recommendation

Overview

This report was prepared for the nonprofit foundation Alphabet Soup, which is seeking a tool to help it selection applicants to fund their business ventures. Alphabet Soup wants a tool that can help it select the applicants for funding with the best chance of success in their proposed ventures.

This project created a tool using a deep learning neural network to train on the data that Alphabet Soup provided about whether prior applicants had been successful in their ventures, and used a subset of the data provided in order to predict future success on untrained data. Thereafter, the model was modified in various ways, and rerun to determine whether the modifications improved performance. Two versions of the model are delivered with this report: the initial model and one that was best optimized given the dataset provided.

The models were designed and tested using Google Colab, so performance results as to run time likely would be different on a local device, but all of the runs finished within at most a few minutes in the Colab environment, so the should be readily accessible by Alphabet Soup.

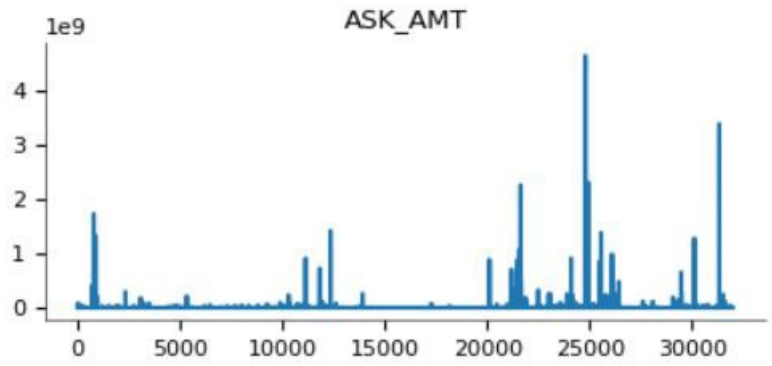
Results

➢ Raw Data

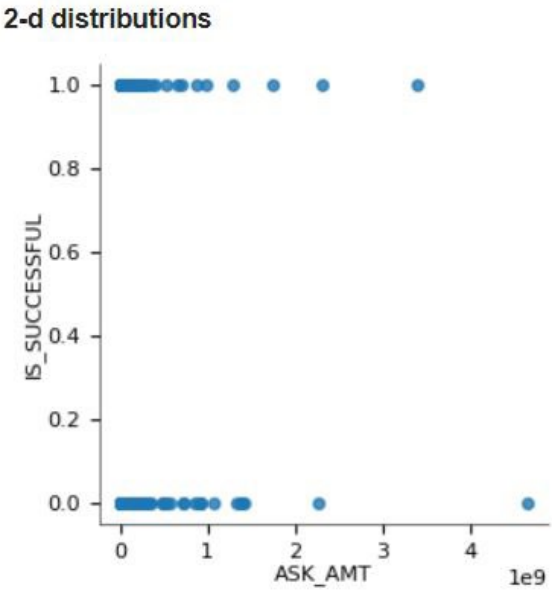
Alphabet Soup originally provided a csv file containing information on approximately 34,000 loan applications. The data included the following:

- 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 considerations for application
- ASK_AMT—Funding amount requested
- IS_SUCCESSFUL—Was the money used effectively

The amounts awarded for the prior funding are all under $50,000. The distribution of prior awards is illustrated as follows:



The size of the request does not appear to be related to whether a previous venture was successful, although the model was designed to include the size of the request in its features.



➢ Data Preprocessing

- Target variables. The "IS_SUCCESSFUL" column was used as the target for the model, as the model is designed to help Alphabet Soup decide whether a loan it is considering is likely to be successful. This column has binary values of 0 for not successful and 1 for successful.

- Feature variables. After the data were pre-processed and some columns were deleted or binned, all of the remaining columns were used as the features for the model.

- Variables to be removed. Initially, the EIN number and the company name of the applicant company were removed as not relevant to any decision-making by the model.

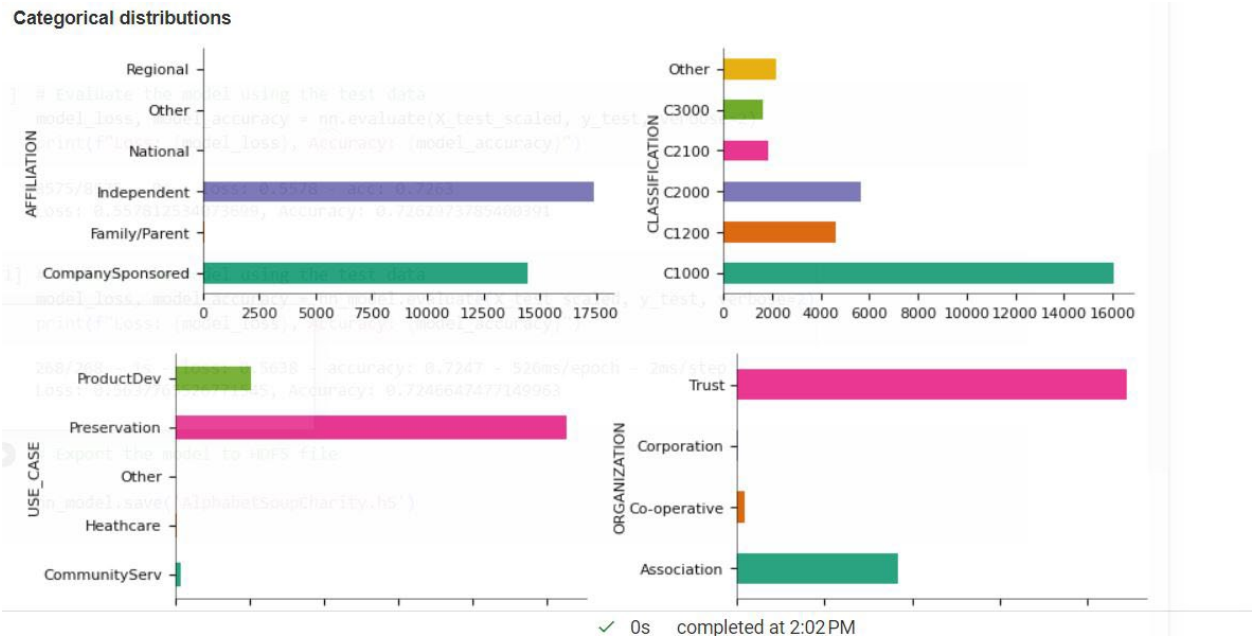
  During optimization, two other variables, "status" and "special_considerations" were also dropped. Each of these variables had binary yes/no values, with a tiny fraction of the total 34,299 applicant records being one value. For instance, the "yes" group for special considerations had 27 members, and the "0" group for status had 5 members out of the total

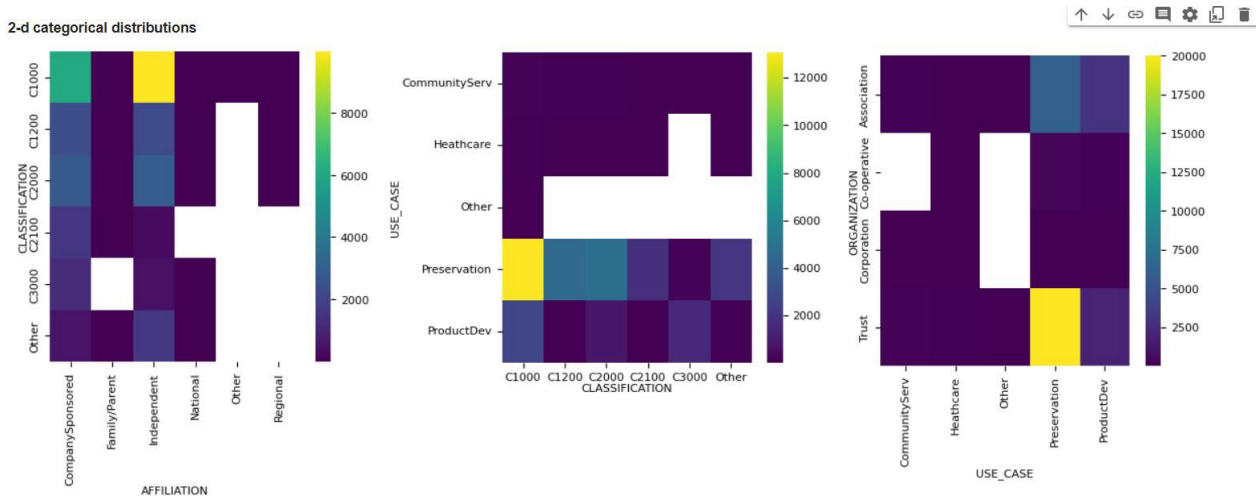
- Data transformation. All of the categorical variables were converted to numeric representations using Pandas "get_dummies" function.

  Initially, the "APPLICATION" and "CLASSIFICATION_TYPE" variables were binned to convert a number of very small groups to the group "other," as the larger groups contained several thousand to 17,000 members. Subsequently, during optimization, several other variables were binned so that very small groups were converted to join an existing "other" group, given the large size of the remaining groups in those variables. These included "USE_CASE", "ORGANIZATION," and "AFFILIATION". After little benefit was shown from creating these bins, later runs were completed with the original, scaled data. The bins for the "CLASSIFICATION_TYPE" also were modified to reduce the number of bins, and then returned to the original binning when little difference was detected.

  The relative group sizes, as seen in the charts that follow, make the decision to bin some of the groups seem reasonable, although it may not take into account all of the business reasons for having those categories. Examining the distribution of loan size requests and organization type, for example, appears to show distinct relationships.

  After all of the data had been binned, they were scaled using scikit learn's Standard Scalar function.





➢ Compiling, Training, and Evaluating the Model

- Model design How many neurons, layers, and activation functions did you select for your neural network model, and why?

  The initial model had two hidden layers and one output layer. Both used the "relu" activation function. That function is the most widely-used in general, and is particularly good where, as here, the data are not linear. The input dimensions was set to the number of features.

  The first layer had neurons equal to 3 times the number of features and an input dimension of the number of features. The second layer had neurons

equal to 2 times the number of features. This was based on common recommendations that the number of neurons be two or three times the input dimension, and some discussion among some experts that three hidden layers is the maximum which should be used.

The output layer had one neuron and an activation function of sigmoid, which creates an S curve between 1 and 0 that is good for binary classifications and probabilities. The same output layer was used in all subsequent optimization efforts.

The model was compiled with a loss function of "binary_crossentropy", the "adam" optimizer, and a metric of "accuracy." The loss function "binary_crossentropy" is designed for binary classification problems such as the one here, seeking a "successful/not successful" answer. The "adam" optimizer is a gradient descent optimizer that is supposed to be more efficient than other gradient descent optimizers.

The first model was run for 100 epochs, to avoid overtraining and overfitting. That number might seem small given the size of the dataset, but all subsequent increases in epochs showed little, if any, benefit. With 300 epochs, the model was swinging back and forth between better or worse performance in very small increments, indicating it might have surpassed the local minimum slope on the curve that it was attempting to reach, so that additional training would be unproductive.

Many additional configurations of the model were attempted in order to improve the accuracy of the predictions. These included:

- adding a third hidden layer, with activation function set at RELU, tanh, and sigmoid
- changing the activation function on the second layer to tanh or sigmoid
- increasing the number of neurons to 3 times the input dimensions for all layers and then only for the first two layers
- decreasing the number of neurons to the input dimension for all hidden layers or for all but the first
- increasing and decreasing the number of epochs, up to 300 and down to 50

In addition to changing the model itself, the data were re-examined and then modified to eliminate features or to change the binning of one feature, as discussed in the data preparation section, above.

- Performance

While the model came close to the target model performance of 75% accuracy, it never quite achieved that goal. It had pretty good performance that all ranged around 72%. One later iteration was 75.59% accuracy, and it never got below 72% or achieved 73% on the testing data, although it achieved almost 75% on the training data in numerous runs. Using the tanh activation function for the second layer appeared to produce slightly worse results, and using another sigmoid before the last output layer appeared to be slightly less good than using RELU but mu

The loss ranged between approximately 0.5.2 to 0.61, with the loss being 0.5743 for the accuracy of 75.59% with three hidden layers, all using RELU activation function, the first with 2 times the number of input items, and the others with the number of input items. While the accuracy was relatively good, but not great, for all of the runs, the loss result is really excellent for 34,000 items with approximately 40 features each.

- Steps taken to increase performance

Many additional configurations of the model were attempted in order to improve the accuracy of the predictions. These included:

  - adding a third hidden layer, with activation function set at RELU, tanh, and sigmoid
  - changing the activation function on the second layer to tanh or sigmoid
  - increasing the number of neurons to 3 times the input dimensions for all layers and then only for the first two layers
  - decreasing the number of neurons to the input dimension for all hidden layers or for all but the first
  - increasing and decreasing the number of epochs, up to 300 and down to 50

In addition to changing the model itself, the data were re-examined and then modified to eliminate features or to change the binning of one feature, as discussed in the data preparation section, above.

Although at least 25 different configurations of the model were examined, a Keras tuner was then created and tested to see if the top recommended configuration would produce better results than anything tried manually. It was allowed to choose from RELU, tanh, and sigmoid functions, with from one to forty neurons and one to five layers.

When the top three models were evaluated based on the recommendations of the keras tuner, the top model only produced a 73% accuracy. This suggests that even if the keras tuner were further modified, the accuracy would not improve greatly. The top recommendation produced a loss of 0.5523 and an accuracy of 0.7307. The model ran slightly faster than the manually-configured models which had many more neurons per layer. While those often ranged between 2 and 3 ms per step, with some models at 3 to 4 ms, the three keras-tuned models all ran at 2 ms per step.

The top recommended model was configured with three hidden layers, using the RELU activation function, with an initial 26 neurons in the first layer. Interestingly, the second and third models both used the tanh activation function, which in observed testing from the manual configurations, seemed to produce slightly worse results, but the manual configurations were never run with all tanh activation functions, and always started with one RELU.

Summary

Overall, the model was able to predict approximately three-fourths of the loans that were likely to be successful among approximately 34,000 loan applications, with an approximately 0.5 percent loss. The best-optimized models While this is quite beneficial for Alphabet Soup in deciding which loans to fund, and the loss numbers are great for a dataset of this size, the organization may want a more finely-tuned prediction to decide where to grant its limited funds.

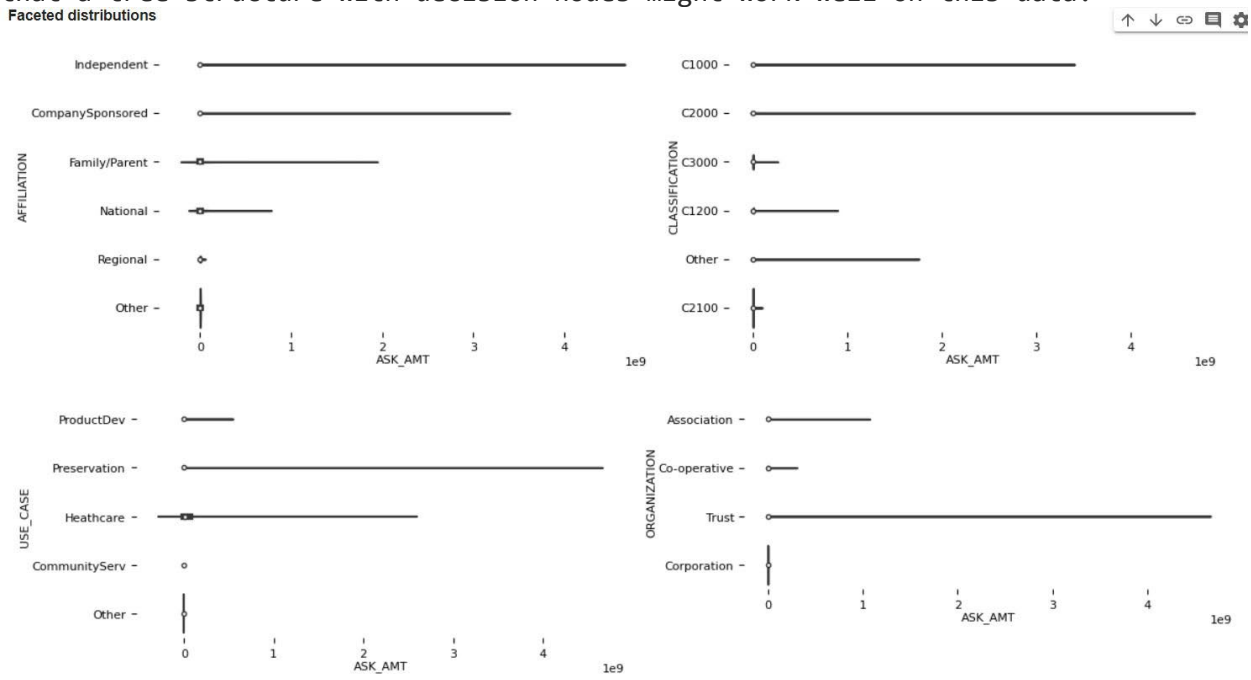
Another type of model that might work better on this dataset would be a random forest model. This type of model breaks down the data into random, smaller trees, conducts analysis on each smaller, simpler part, and then combines the trees into a whole. A random forest model works well on non-linear data, with multiple outliers, and with large datasets. That is the type of dataset that Alphabet Soup provided. Although outliers have been reduced by binning categories such as affiliation and organization, there are still quite small categories and then two large categories for several features. The data also has numerous categories rather than linear data such as amounts or temperatures, which would lend itself to node decision points.

A random forest model takes individual, weak components that don’t themselves classify well because trained on only a small subset, and combines them to make a much more powerful whole.

A random forest model therefore seems designed for precisely the type of situation this dataset presents. Each epoch of the training on this dataset increased accuracy very slowly and very slightly, with numerous different hyperparameters used to tweak the model for the most accurate outcomes. The trials resulted in improvements as different optimization efforts were

undertaken, but no huge improvement over multiple different configurations over multiple trials. A random forest model might work well, because it combines relatively weak predictions that, once combined, form a robust predictor.

Examining some charts of distributions produced by Google Colab suggests that a tree structure with decision nodes might work well on this data.



As a final step in the analysis, however, a random forest model was created and run multiple times, with the number of estimators from 128 to 526 and the number of random states from 78 to 240. The best results were produced with 128 estimators and 78 random states. This configuration produced a much worse result than any of the neural network models run in the original testing, at 70.95% accuracy.

In sum, it is not clear how Alphabet Soup came up with the 75% accuracy requirement, but the organization may need to adjust its expectations slightly, or expend huge amounts of money to get a more sophisticated model on a larger machine, which is probably not financially feasible for it.