

## Charity Funding Predictor

### Model Analysis

#### Purpose

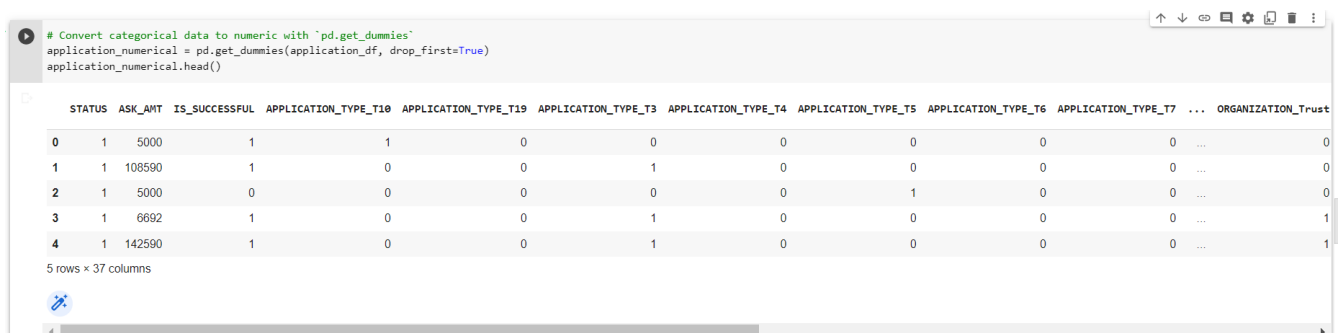
The binary classifier has been created in order to analyse the dataset of over 34,000 organisations and predict whether or not applicants will be successful to receive funding from the non-profit foundation - Alphabet Soup. The CSV file contains several columns with organisations metadata (e.g. name, classification, use\_case, income\_amt). The designed neural network model based on the input data evaluates which organisations classify for donation and which are considered too risky.

#### Results

The dataset used for the project was a charity.csv file with details for over 34,000 organisations. Using libraries like Pandas, Scikit-Learn and TensorFlow the dataset has been pre-processed, compiled, trained, evaluated and then optimised to achieve the highest possible accuracy.

#### Data Pre-processing

Before feeding the data into the model, following transformations have been performed to ensure the quality of the data. The identification columns (EIN, NAME) not contributing to the analysis have been discarded. The columns containing more than 10 unique values have been inspected and the rare categorical variables have been binned together into a new value (Other). As seen in the Figure 1. the categorical variables have been converted into the numerical ones using `pd.get_dummies()`.



```
# Convert categorical data to numeric with 'pd.get_dummies'
application_numerical = pd.get_dummies(application_df, drop_first=True)
application_numerical.head()
```

	STATUS	ASK_AMT	IS_SUCCESSFUL	APPLICATION_TYPE_T10	APPLICATION_TYPE_T19	APPLICATION_TYPE_T3	APPLICATION_TYPE_T4	APPLICATION_TYPE_T5	APPLICATION_TYPE_T6	APPLICATION_TYPE_T7	...	ORGANIZATION_Trust
0	1	5000	1	1	0	0	0	0	0	0	...	0
1	1	108590	1	0	0	1	0	0	0	0	...	0
2	1	5000	0	0	0	0	0	1	0	0	...	0
3	1	6692	1	0	0	1	0	0	0	0	...	1
4	1	142590	1	0	0	1	0	0	0	0	...	1

5 rows × 37 columns

Figure 1. Cleaned, numerical dataset

The cleaned dataset has been split into target variable and features. The binary values of the 'IS\_SUCCESSFUL' column became the y (target) variable and the remaining values were considered features.

Following that the pre-processed data has been split into training and testing data and scaled using `StandardScaler()`.

## Compiling, Training, and Evaluating the Model

Using the TensorFlow library the deep learning model has been designed to create a binary classification model to predict if an Alphabet Soup-funded organization will be successful based on the features in the dataset. As seen in the figure 2, initially, 2 hidden layers have been created with 80 and 30 neurons respectively. The input variables for the input layer have been specified by the number of the features in the dataset. A good rule of thumb when building the initial model is to use 2–3 times as many neurons as there are input features, hence it has been specified as 80. The relu activation function has been chosen for the hidden layers and sigmoid for the output layer. As the ReLU function is always a good starting point and ideal for classification, it has been chosen for the hidden layers in this model as well. The sigmoid function is ideal for a binary classification dataset and works best with values normalized to a probability between 0 and 1, that's why it was chosen for the output layer.

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len(X_train_scaled[0])
hidden_nodes_layer1 = 80
hidden_nodes_layer2 = 30
output_nodes_outputlayer = 1

# Create the Keras Sequential model
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(
    tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu")
)

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))

# Output layer
nn.add(tf.keras.layers.Dense(units=output_nodes_outputlayer, activation="sigmoid"))

# Check the structure of the model
nn.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 80)	2960
dense_1 (Dense)	(None, 30)	2430
dense_2 (Dense)	(None, 1)	31

=====  
Total params: 5,421  
Trainable params: 5,421  
Non-trainable params: 0

Figure 2. Design of the deep neural network

As seen below the model achieved around 72,83% accuracy, meaning the target performance hasn't been met. In order to achieve satisfactory result, further optimisation has been performed.

```
# 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: 0.5545 - accuracy: 0.7283 - 423ms/epoch - 2ms/step  
Loss: 0.5545328855514526, Accuracy: 0.7282798886299133

Figure 3. Model loss and accuracy

## Optimisation

The model has been optimised to achieve the highest possible accuracy. To remove unnecessary noise from the dataset further columns have been discarded. Only organisations with the active status have been selected and the status column has been deleted.

As seen in the figure 5 the number of values in the bin for the APPLICATION\_TYPE has been increased and all of the application type's smaller than 1,000 have been grouped together as opposed to the initial cut-off below 500.

```
[7] # Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df = application_df.drop(['EIN', 'NAME', 'SPECIAL_CONSIDERATIONS'], axis=1)

#selecting organisations with active status
application_df = application_df.loc[application_df['STATUS'] == 1]
application_df = application_df.drop(columns='STATUS')
application_df.head()
```

	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZATION	INCOME_AMT	ASK_AMT	IS_SUCCESSFUL
0	T10	Independent	C1000	ProductDev	Association	0	5000	1
1	T3	Independent	C2000	Preservation	Co-operative	1-9999	108590	1
2	T5	CompanySponsored	C3000	ProductDev	Association	0	5000	0
3	T3	CompanySponsored	C2000	Preservation	Trust	10000-24999	6692	1
4	T3	Independent	C1000	Healthcare	Trust	100000-499999	142590	1

Figure 4. Data cleaning

```
# Choose a cutoff value and create a list of application types to be replaced
# use the variable name `application_types_to_replace`
application_type = application_df['APPLICATION_TYPE'].value_counts()

application_types_to_replace = application_type[application_type<1000].index

# 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	27032
Other	2266
T4	1542
T6	1216
T5	1173
T19	1065

Name: APPLICATION\_TYPE, dtype: int64

Figure 5. Binning

Also the design of the deep neural network has been changed. Additional hidden layer has been added to the model to increase the accuracy. However, as this increases the training iterations and memory resources, it was decided to still keep the number of the hidden layers below 4. The activation function has been changed to the LeakyReLU as it is a good alternative to the ReLU function because of its ability to characterize negative input values.

```

# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len(X_train_scaled[0])
hidden_nodes_layer1 = 110
hidden_nodes_layer2 = 80
hidden_nodes_layer3 = 30
output_nodes_outputlayer = 1

from keras.layers import LeakyReLU
# Create the Keras Sequential model
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(
    tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="LeakyReLU")
)

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="LeakyReLU"))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation="LeakyReLU"))

# Output layer
nn.add(tf.keras.layers.Dense(units=output_nodes_outputlayer, activation="sigmoid"))

# Check the structure of the model
nn.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 110)	3520
dense_1 (Dense)	(None, 80)	8880
dense_2 (Dense)	(None, 30)	2430
dense_3 (Dense)	(None, 1)	31

=====  
 Total params: 14,861  
 Trainable params: 14,861  
 Non-trainable params: 0

Figure 6. Optimised design of deep neural networks

Despite the optimisation the model still didn't achieve the desired performance. The accuracy slightly improved, but still remained below the target of 75%. The model accomplished accuracy of 73,57%.

```

# 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: 0.5508 - accuracy: 0.7357 - 436ms/epoch - 2ms/step  
 Loss: 0.5507572889328003, Accuracy: 0.7357126474380493

Figure 3. Optimised model loss and accuracy

## Summary

The initial model achieved a moderate accuracy of 72,93% and model loss of 55,45%. The optimisation of the model improved the results up to accuracy of 73,57% and model loss decreased to 55,08%, however reaching the target of 75% was unsuccessful.

A better solution for this classification problem could be building a Random Forest model. The Random Forest is a much easier and flexible machine learning algorithm. It produces a prediction even without the data pre-processing and will work with missing and categorical data. Training the Random Forest model requires only setting the number of trees, while using Neural Networks requires defining the model's architecture and a lot of manual tuning to perfect the model, each of the Neural Networks hyper-parameters is crucial and can easily affect the result. Neural Networks works great with images, audio or text data, whereas Random Forest seems like a better and simpler choice for the tabular data.