

Machine Learning Engineer Nanodegree

Capstone Proposal

Rhys Shea

September 2018

Domain Background

This project is a recent Kaggle competition from RStudio, Google Cloud and Kaggle to demonstrate the impact that thorough data analysis can have. For many businesses the 80/20 rule has proven true, where only a small percentage of customers produce most of the revenue. This challenges marketing teams to invest in the most appropriate promotional strategies and target the right customers to increase revenue. Statista.com shows that Alphabet's expenditure on marketing in 2017 was almost \$13 billion^[1] and a Wall Street Journal article shows that marketing budgets comprise 11% of total company budgets on average^[2]. This is a phenomenal amount of money that could be going to waste according to the 80/20 statistic.

I personally find this problem of particular interest as it can be replicated across the retail industry. Though there is no such thing as one solution fits all in machine learning, I believe the lessons learned will be of great value and can lead to further investigation, which in turn could be beneficial to many small companies who would welcome insights for saving on their marketing budget.

Problem Statement

In this competition, the challenge is to analyse a Google Merchandise Store (also known as GStore) customer dataset to predict revenue per customer. Hopefully, the outcome will be more actionable operational changes and a better use of marketing budgets for those companies who choose to use data analysis on top of Google Analytics data.

The prediction to be made is the natural log of the sum of all transactions per user. For every user in the test set, the target is:

$$y_{user} = \sum_{i=1}^N transaction_{user_i}$$

$$target_{user} = \ln(y_{user} + 1)$$

This is a regression problem, not a classification problem. Therefore the output should be a continuous natural log of the total transactions per user.

Alongside other features of potential importance, there exists a total revenue for each user, which will be used as a target. The output of the model will be the predicted total revenue for users where only the other features are known. The model will aim to find a pattern between the known features and the target.

Datasets and Inputs

The data source is from the 'Google Analytics Customer Revenue Prediction' competition, the API of which is `kaggle competitions download -c ga-customer-revenue-prediction`. The data is also accessible through the Google Cloud Platform BigQuery as the `ga_train_set` dataset, and the `ga_test_set` dataset, under the `kaggle-public-datasets` project

There are two datasets complimenting this project; *train.csv* and *test.csv*. These contain the data necessary to make predictions for each visitor to the store. The predicted data is to be submitted as *sample_submission.csv*.

The data is formatted according to the columns listed under Data Fields below. Each row in the dataset is one visit to the store. Not all rows in *test.csv* correspond to a row in the submission file, but all unique visitor IDs will correspond to a row in the submission. As shown below in the list, the data consists of both numeric and categorical data.

There are multiple columns that contain JSON blobs of varying depth. In one of those JSON columns, totals, the sub-column *transactionRevenue* contains the revenue information to be predicted. This sub-column exists only for the training data.

Train.csv

- 903,653 rows
- 12 columns initially, which expands to 55 columns once the JSON blobs have been flattened
- Time period: 1 Aug, 2016 to 31 July, 2017

Test.csv

- 804,684 rows
- 53 columns after data flattening
- Time period: 2 Aug, 2017 to 30 Apr, 2018

Calling the difference method between the columns of each loaded dataframe shows the missing 2 columns are *totals.transactionRevenue* and *trafficSource.campaignCode* from *test.csv*. This makes sense, as the *transactionRevenue* is the target for the test data. There is no overlap in the training and testing dataset timeline. I intend to use the full dataset provided as there is only 2 years of data in total between both datasets.

Data Fields

- *fullVisitorId* - A unique identifier for each user of the Google Merchandise Store (numeric, *must be loaded as strings in order for all ID's to be properly unique*)
- *channelGrouping* - The channel via which the user came to the Store.
- *date* - The date on which the user visited the Store
- *device* - The specifications for the device used to access the Store.
- *geoNetwork* - This section contains information about the geography of the user (categorical)
- *sessionId* - A unique identifier for this visit to the store (numeric)
- *socialEngagementType* - Engagement type, either "Socially Engaged" or "Not Socially Engaged" (categorical)
- *totals* - This section contains aggregate values across the session (numeric)
- *trafficSource* - This section contains information about the Traffic Source from which the session originated (categorical)
- *visitId* - An identifier for this session. This is part of the value usually stored as the `_utmb` cookie. This is only unique to the user. For a completely unique ID, a combination of *fullVisitorId* and *visitId* should be used (numeric)
- *visitNumber* - The session number for this user. If this is the first session, then this is set to 1.
- *visitStartTime* - The timestamp (expressed as POSIX time).

Removed Data Fields

Some fields were censored to remove target leakage. The major censored fields are listed below.

- *hits* - This row and nested fields are populated for any and all types of hits. Provides a record of all page visits.
- *customDimensions* - This section contains any user-level or session-level custom dimensions that are set for a session. This is a repeated field and has an entry for each dimension that is set.
- *totals* - Multiple sub-columns were removed from the totals field.

Solution Statement

There are many features in this dataset, which I envisage will make for a very complex model to predict the desired target. Simplification of the data through data exploration will be useful to determine which features appear to be important when predicting the target. I plan to implement a Deep Learning regression model in order to solve this problem. In recent years, deep artificial neural networks have become powerful tools in solving complex problems. Though the methodology has been known for many years, it is only with the advancements in processing power that these methods have been put to practical use^{[3][4]}. My reason for choosing this method is two-fold; firstly the power of deep learning to solve complex models and secondly for my own learning. I would like to expand my experience using these techniques as well as become more familiar with TensorFlow. One of the sources I will be using to this end is the TensorFlow extracurricular section of the nanodegree.

Benchmark Model

This kernel^[5] will serve as a useful baseline in order to explore the data and begin to make adequate predictions. The current best submission on the leader board shows a RMSE of 1.44, which I do not anticipate to achieve at this stage. However the benchmark achieves a RMSE of 1.70 using a gradient booster method, I hope to be able to achieve an improved score on this using deep learning methods. Additionally this kernel will serve as a baseline model for designing the neural network, with the below structure^[6].

```
-----  
Layer (type)                Output Shape                Param #  
-----  
dense_1 (Dense)              (None, 256)                 9984  
-----  
dense_2 (Dense)              (None, 128)                 32896  
-----  
dense_3 (Dense)              (None, 16)                  2064  
-----  
dense_4 (Dense)              (None, 1)                   17  
-----  
Total params: 44,961  
Trainable params: 44,961  
Non-trainable params: 0  
-----
```

Evaluation Metrics

Submitting to Kaggle will provide feedback on 30% of the data for the *test.csv* data, which will be of use to see where my model stacks up in the broader competition. However, it would be useful to know how the model is performing before submitting to the competition; therefore the data will be split further into *validation* data in order to test the accuracy of the model's prediction against data where the actual target is available for comparison with the prediction. I am considering an 80/20 split of the training data to be split into training (80%) and validation (20%), but I have not yet concluded whether this will be done at random or linearly in time. It is common in forecasting models to use the root-mean square as a means to evaluate performance. A root-mean square error metric will be used here to compute deviations between the predictions and targets. This will be done for all users, N .

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (\hat{y}_i - y_i)^2}{N}}$$

The baseline also employs the use of RMSE to test the performance of the model and so can be compared to my own model.

Project Design

Data exploration

The dataset is split into many columns, several of which consist of JSON blobs of varying depth. The first step will be to flatten the dataset to a single layer. A method for this has already been supplied by this kernel^[7] and is shown below.

```
def load_df(csv_path='../input/train.csv', nrows=None):
    JSON_COLUMNS = ['device', 'geoNetwork', 'totals', 'trafficSource']

    df = pd.read_csv(csv_path,
                     converters={column: json.loads for column in JSON_COLUMNS},
                     dtype={'fullVisitorId': 'str'}, # Important!!
                     nrows=nrows)

    for column in JSON_COLUMNS:
        column_as_df = json_normalize(df[column])
        column_as_df.columns = [f"{column}.{subcolumn}" for subcolumn in column_as_df.columns]
        df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)
    print(f"Loaded {os.path.basename(csv_path)}. Shape: {df.shape}")
    return df
```

Data cleaning


A brief overview of the dataset reveals that there are null values throughout. If there are entire columns that contain only null values they will not be of any use to the model. Removing these columns will save on space and time during the running of the model. This will be decided after the data exploration step. The code below will give insight into these columns that have only one unique value.

```
Const_cols = [c for c in train_df.columns if train_df[c].nunique(dropna=False)==1]
```

Normalisation and One-hot encoding

There are many different types of data in the dataset provided, such as numerical data and strings. It is usually good practice to normalise numerical data of varying scales to values between 0 and 1 to aid prediction. It is also prudent to one-hot encode the data in order for stringified data to become useful. Data as strings are categorical, which many machine-learning algorithms cannot directly handle. Therefore it is necessary to convert categorical data into numbers. A simple example of this is shown below.

Color			
Red			
Red			
Yellow			
Green			
Yellow			



Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

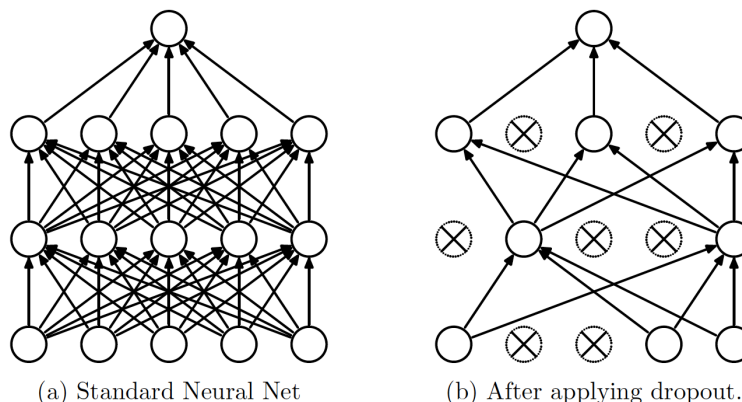
Feature importance

In this stage I will be investigating the importance of different features using inbuilt methods of the chosen predictive model. This is a very powerful tool to see what is having the biggest impact on the predictive capability of the model. By pursuing the most important features I should be able to quickly come up with an adequate model that can be further refined.

Model

The initial model will look similar to the benchmark model shown above, with a hidden layer consisting of 4 fully connected layers. Inspiration will also be taken from this introductory model for predicting stock prices in TensorFlow^[8]. The final layer will be a continuous single layer in order to output a continuous natural log of total revenue.

The model will be run over 500 epochs initially, which may change after further testing and knowledge. This value was chosen to balance the time spent running the model against having a representative view of the model's performance. I will use early stopping to prevent overfitting as well as a dropout rate. Overfitting to the training data can be disastrous to a model, resulting in very good results in training but very poor results thereafter. Using a dropout rate is an effective way of preventing this by randomly blocking some nodes during each run so that the model does not become overly dependent on a single pathway in the network^[9]. The diagram below illustrates this. Though this may require more epochs to converge to the optimal solution, it will decrease the runtime per epoch. Early stopping will also be useful in decreasing the runtime of the model as the model will be setup to stop running if improvement is not seen after so many consecutive epochs. This is a basic description of my expected model to begin with. After further experimentation and investigation into other methods this model should be elaborated on.



References

- [1] <https://www.statista.com/statistics/507853/alphabet-marketing-spending/>
- [2] <https://deloitte.wsj.com/cmo/2017/01/24/who-has-the-biggest-marketing-budgets/>
- [3] Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015) 85–117
- [4] <http://www.deeplearningbook.org/>
- [5] <https://www.kaggle.com/sudalairajkumar/simple-exploration-baseline-ga-customer-revenue>
- [6] <https://www.kaggle.com/dimitreoliveira/deep-learning-keras-ga-revenue-prediction>
- [7] <https://www.kaggle.com/julian3833/1-quick-start-read-csv-and-flatten-json-fields/notebook>
- [8] <https://medium.com/mlreview/a-simple-deep-learning-model-for-stock-price-prediction-using-tensorflow-30505541d877>
- [9] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15 (2014) 1929-1958