Data Driven Online Revenue Prediction

Peter Fagan
University College Cork

ST4090 Written Report
School of Mathematical Sciences, University College Cork, Ireland

Supervised by: Dr Supratik Roy

April 2019

Data Driven Online Revenue Prediction

Peter Fagan

School of Mathematical Sciences, University College Cork, Ireland.
April 2019

Abstract
The leveraging of customer data to provide actionable commercial insights is a well-established practice in modern data analysis. Thanks to technological advances over the past few years the implementation of machine learning algorithms on large sets of data has become more feasible. This combined with the accessibility of large and scalable computing resources has enabled businesses to analyse large datasets which they previously might not have been capable of.
In light of these points this report will analyse customer data generated by an online merchandise store (Google Merchandise Store), with the goal of predicting how much a particular customer will spend within some future time interval. This report will outline the data handling, data analysis and model selection procedures employed to generate such predictions. Furthermore, a discussion on the technologies used to perform such analysis is included as part of this report.

1. Introduction
In 1997 the Google search engine was founded. Since then the use of internet technologies has experienced remarkable growth. Online ecommerce is now estimated to be worth $29 trillion and this figure continues to grow. With the growth of ecommerce there has been an influx of web traffic data. In this report I outline an approach for using this data to predict customer behaviour namely spending.
Google Analytics [1] is a web-based analytics service provided by the tech giant Google. It is currently the most widely used web analytics software. This service tracks and reports on website traffic. The data for this report is generated by this service running on the Google Merchandise Store. This data is made publicly available through a competition hosted on the data science platform Kaggle [2]. The competition outline is as follows. The goal is to predict the revenue generated per user visiting the store. However, we are not asked to predict simply the revenue but we are asked to predict the logarithm of the sum of all transactions made by the user plus one.

$$y_{user} = \sum transaction_{user}$$

$$\text{target} = \ln\left(y_{user} + 1\right)$$

In order to do so we are given a training set of web traffic data for the period spanning August 1st 2016 to April 30th 2018. Our test set is in the same format as the training set except it is over a future interval from May 1st 2018 to October 15th 2018. The task is to predict the target variable over the interval December 1st 2018 to January 31st 2019,
which takes place 46 days in advance of the test period.
The motivation for our analysis is that we can use the resulting model to predict future revenue, identify attributes of high spending users and hence generate a strategy such as targeted advertising to increase future revenue. The end goal is to increase future revenue through enhancement of the current marketing strategy. A discussion of which will be included later in the report.
All analysis for this report is performed using either R or Python and all code is made available at https://github.com/peterfaganucc/Google-Analytics-Customer-Revenue-Prediction .

2. The Dataset
The raw data consists of two csv files with model training and testing data respectively. The stored files require approximately 8gb and 4gb of storage respectively. Within each file are the same set of 14 variables which include customer id, date, transaction, geographical, device and traffic data. A number of the columns in the original csv are composed of data in JSON format [3]. There are 1708337 rows in the training set and 401589 in the test set. Each row represents the data for a single customer visit on the online store.
While this dataset is clearly structured it was still necessary to do some work in order to transform the data into a more usable format by parsing the JSON columns. An issue I encountered in tackling this

problem was time it took to run on my local machine. In both Python and R the process would stall or take too many hours. Furthermore, importing the csv files in the original form would tend to crash on my local machine. To remedy this issue I rented an Ubuntu virtual machine from the Google Cloud Platform. I increased the available RAM to increase performance and installed Python and the Jupyter Notebook ide onto this machine. Using Python on this virtual machine I was able to parse the JSON columns effectively. I dropped columns which contained miscellaneous data and dumped the parsed csv data to my storage bucket on the GCP. This new csv contained a greater number of columns compared to the original csv since each JSON column contained multiple data fields. With the dataset parsed further analysis was completed using R.

3. Exploratory Data Analysis

The first explored aspect of the data was to investigate the number and distribution of missing values in the data. Most Na values in the data were denoted by miscellaneous characters such as, "missing" or "not-set". Therefore, I identified the set of characters denoting Na values and I converted each instance of these characters to Na values. Having done this I plotted the occurrence of Na values as a horizontal bar chart.

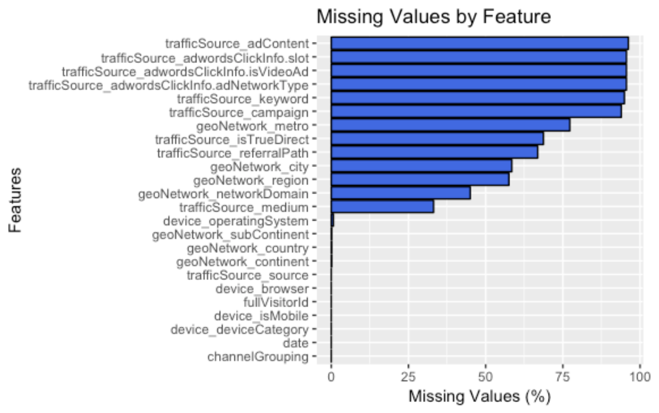

Fig 1: Horizontal bar chart of non-numeric features indicating the percentage of Na values per feature.

Further investigation of the usefulness of variables with a large proportion of Na values is reserved till later in the report. Features with more than 70% Na values will likely be dropped from the list of features used unless they prove valuable in our predictive analysis.

Next I analysed the distribution of our target variable for users who generated revenue. The distribution was found to be approximately normal on inspection.

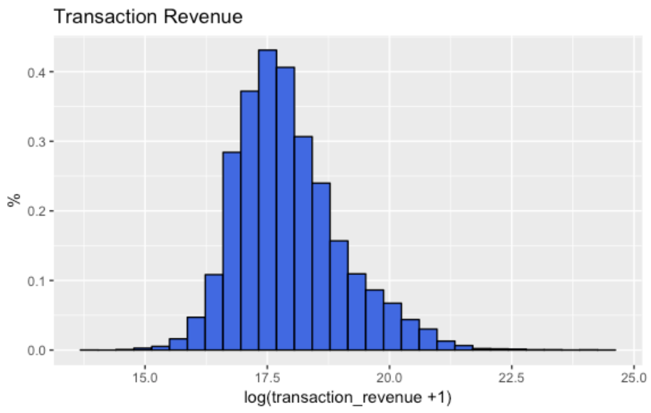

Fig 2: Distribution of revenue for customers who generated revenue.

Investigating the proportion of customers who visited the site and actually spent money we can see that most of the stores profits are generated by a small proportion of its visitors. This indicates that our dataset is unbalanced as the target variable is more often than not simply zero.

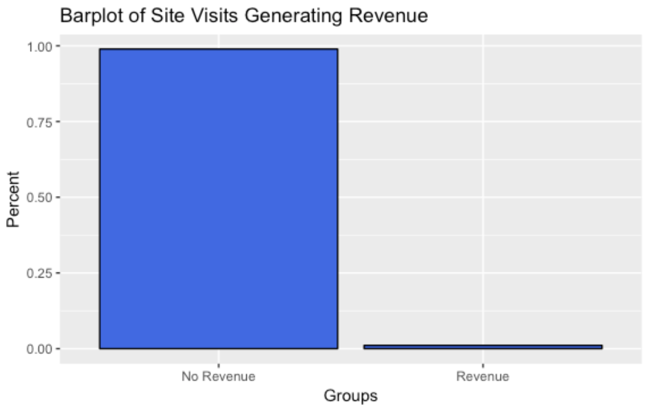

Fig 3: Bar chart indicating proportion of website visitors who generated revenue.

Next I plotted the total daily transaction revenue over time to visually assess if there is any trends or seasonal effects present.

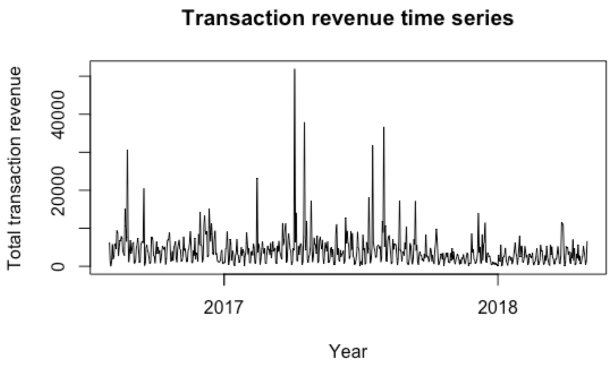

Fig 4: Time series plot of the transaction revenue.

As we can see there is no obvious seasonal trend present in the transaction revenue. Neither is there much trend by simple inspection of this plot.

Next we investigate the distribution of transaction revenue amongst various levels of our categorical variables. This will give us an overall idea of the differences between categorical levels, as well as helping identify attributes that relate to higher revenue generation. From (Fig 5) we see that Chrome OS and Macintosh OS users generate more revenue on average. People living in the Americas also generate more revenue on average, while those customers who were referred to the website on average spent more. Desktop users were also found to be generate more revenue on average when compared with mobile users. While these plots indicate which levels within a category yield the highest mean revenue they don't give us an indication of the differences between levels across all website visitors (not just those that spent money). For this we can generate boxplots which indicate the mean transaction revenue for particular levels. In doing this we discover that the mean is almost always zero with some levels having a greater number of outliers.

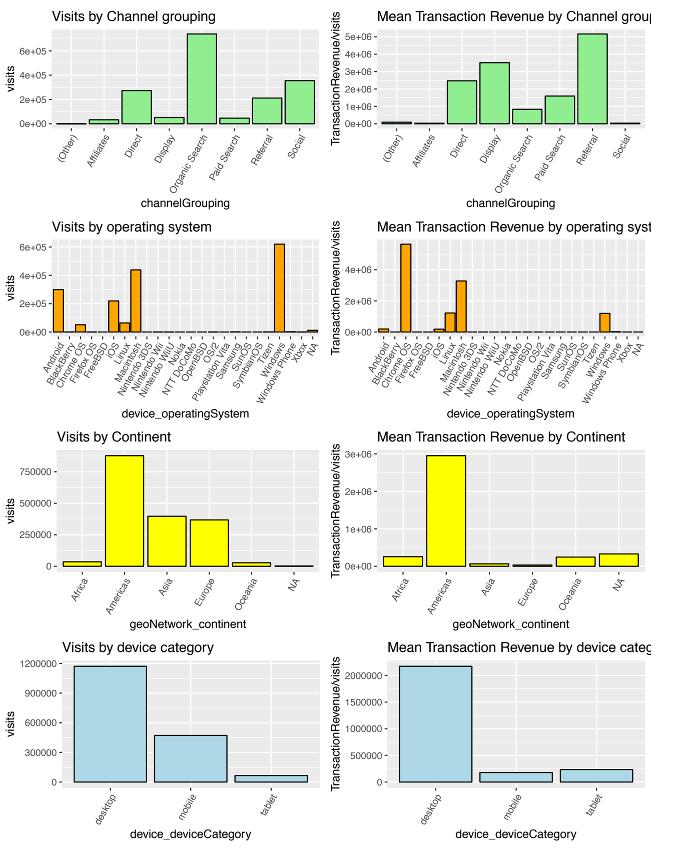

Fig 5: Bar chart of Visits and Mean transaction revenue for various categorical variables.

This isn't promising for predictive performance as we would ideally like to see disparity between the levels of our categorical features. However, it is worth mentioning at this point that this is likely due to the uneven distribution in customer spending discovered earlier where it was evident that most website visits don't generate revenue.

We also investigate categorical variables with large number of levels. There are a number of categories with more than 32 levels. This is an issue since it slows the performance of models and also can lead to less accurate models due to increasing the size of our hypothesis set (the curse of dimensionality). By following Occam's Razor [7] a valuable rule of thumb is to keep the number of features retained to be the minimal meaningful amount. It is quite clear from investigations that many of the levels in these large categorical variables are redundant and therefore it is necessary to reduce the number of levels. The list of features with greater than 32 levels as follows:

device_browser
geoNetwork_city
geoNetwork_country
geoNetwork_metro
geoNetwork_networkDomain
geoNetwork_region
totals_sessionQualityDim
trafficSource_adContent
trafficSource_campaign
trafficSource_keyword
trafficSource_referralPath
trafficSource_source

The resolution of these variables into more suitable variables is reserved for the feature engineering section. This concludes the initial exploration of the dataset.

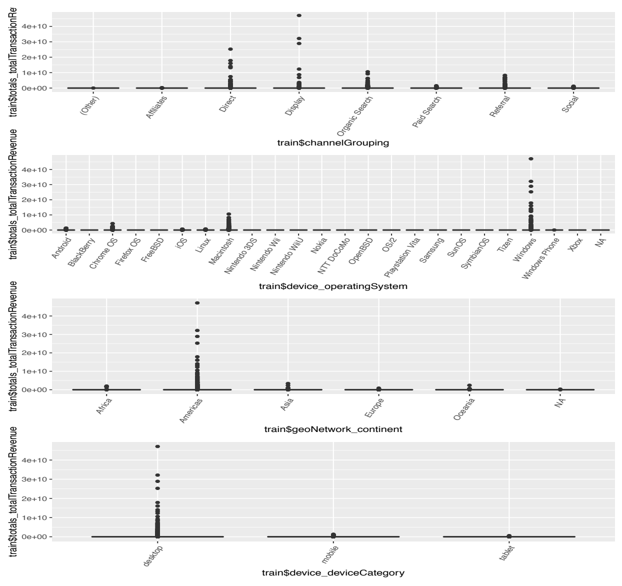

Fig 6: Boxplot of Visits and Mean transaction revenue for various categorical variable

4. Dataset Construction

Further work is needed to transform this dataset into a suitable format for prediction. The training set is constructed by analogy to the prediction problem. Therefore, the training set is split into non-overlapping intervals of 168 days in analogy to the test set. The training set is then filtered per user and summary statistics for each customer's web traffic data is calculated. Furthermore, for each of these sets of 168 days the target variable and return status of the customer is calculated from the future prediction period which takes place 46 days in advance and lasts for 62 days (needs to lie within the training set to be calculated). The target variable is combined with the target variable to form the training set. Once the training set is constructed in this way the test set is left as it is given to us originally and we predict the target variable on that test set. Note this is the format of the problem as outlined on Kaggle. For this report since we will discuss model performance we construct our test set from the data on which we can calculate the target variable. Therefore we construct the test set from the data on which we calculated the target variable. In this report a 70/30 split was chosen for the training and test sets.

5. Feature Engineering

Given the insights we have gained from the exploratory data analysis performed we can begin to apply this knowledge in engineering our features. One of the first issues we need to deal with is the categorical variables with too large a number of levels. Having too large a feature space is problematic since it increases the magnitude of our hypothesis space hence increasing computation cost and potentially decreasing model accuracy. To deal with having too many levels in the categorical variables I considered label encoding, one-hot encoding [8] and target encoding the categorical variables. These methods of encoding take categorical variables and map them to numerical values. I deemed one-hot encoding to be unsuitable given it would drastically increase the feature space, while label encoding suffered from similar issues. As a result I chose to target encode categorical variables with large number if levels. Therefore for each distinct level in the categorical variable being encoded, I calculated the mean of the target variable. Then I mapped each category to its corresponding mean.

For categorical variables with a relatively small number of missing values I simply regarded these Na values as a category in their own right. For each customer statistics were calculated summarising the original data. The time interval over which the customer browsed the website was constructed as a feature. The times that the first and last sessions occurred from the start and end of the investigation period were made into a feature. Statistics such as the mean/median/mode of the number of pageviews were each constructed as features. Further features were created a full list of which is made available in the code found in the appendix.

6. Feature Analysis (investigation)

In this section I investigate the predictive power of feature variables and consider removing features which may be unnecessary and in some cases leading to reductions in the accuracy of our models. In the first step of analysis I examined the correlation between features [9]. To do so I computed the correlation matrix. I then identified the list of feature pairs with correlations greater than 80%. Some of the variables in these pairs may need to be dropped.

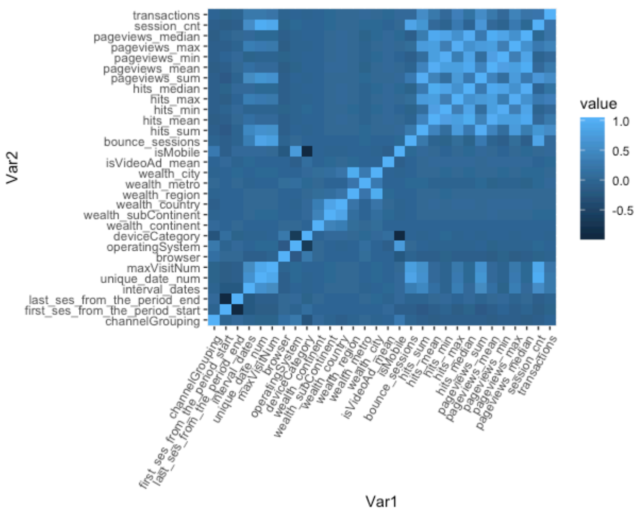

*Fig 7: Visual representation of the correlations between features. Lighter shades of colour denote high correlation between features. *Note: diagonal is perfectly correlated since each variable will be perfectly correlated with itself as expected.*

To get an even better understanding of which variables prove important in predicting the target variable I fit a gradient boosted tree model [11] in order to determine the importance of each feature. Using this set we can gain some commercial insights into what is an important feature of customers who generate revenue. In this instance we see that the channel grouping of the customer is quite important as is the time the first session starts at and the last session ends at. These features will be discussed further in the discussions section.

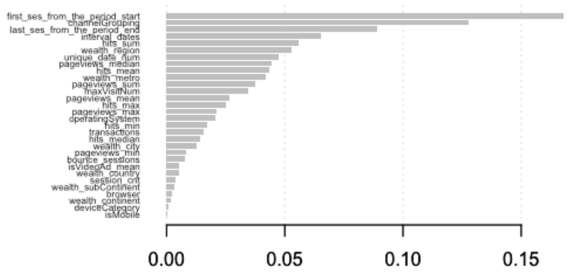

Fig 9: Plot of variable importance from the fitted gradient boosting tree model.

7. Model Fitting

To predict the customer revenue generated per user I split the problem into a regression and a classification problem. The regression problem is to simply predict the expected revenue generated by a customer over the prediction period while the classification problem is to predict whether or not a customer will return to visit the store over the prediction period. The final prediction is generated by combining the predicted revenue with the probability of the customer returning to visit the store over the prediction period. These results are combined through simply multiplying one by the other.

I fitted three different types of models to the data: a gradient boosted tree model, a neural network [13] and a penalised regression model (Elastic Net) [12]. Each model was fitted using 10-fold cross validation [14].

Cross Validation

Cross validation is a technique which aims to assess out of sample performance of a model. This is achieved through training on a majority subset of the known data and testing the model trained in the majority subset on a the subset excluded from the training process. This process is repeated for new subsets. The schema outlined in the following steps is for a form of cross validation known as leave-one-out cross validation:

(1) Partition the dataset into k randomly sampled subsets without replacement of equal size (equal size where applicable).

(2) Train a model on k-1 subsets or folds of this dataset through omitting the 1st subset from training.

(3) Evaluate the trained model performance on the 1st subset i.e. the subset left out of the training procedure.

(4) Repeat this procedure each time omitting the next subset until all subsets have been omitted from the training process at least once.

The use of cross validation helps to reduce model overfitting and give an insight into the generalisation performance of a model [14].

Next I will now outline the hyperparameter tuning methods I used for each model and an analysis of each model's performance. Note that for each of the models below the metric being minimised by the objective functions is the root mean squared error of predictions. Furthermore for prototyping code models are run on a lower percentage of the overall dataset for performance reasons. A discussion of complexity of model training is reserved for the discussion section.

Regression

XGBoost

The xgboost algorithm is an effective gradient tree boosting algorithm. It is regarded as a state of the art ensemble learning model and that is why I choose it as the first model I would attempt to fit to the data. Unlike other ensemble algorithms such as random forest the xgboost algorithm deals with minimising the objective function differently. To each (base) tree/learner the algorithm fits a successive tree to the residuals. In this way the training of the boosted tree algorithm differs from random forest as it is an additive model. This is the basic concept of tree boosting. In the xgboost algorithm multiple subsequent trees are considered at each step and the option that minimises the loss function is chosen (greedily). For this reason the xgboost algorithm is an ensemble additive model. The original paper outlines in detail the algorithm and its implementation [15].

The parameters that need to be tuned when fitting the xgboost model are as follows: nrounds, max_depth, eta, gamma, min_child_weight, colsample_bytree and subsample. I decided I would use a grid search to fit parameters. The upside to this method is the ease of implementation and interpretability however it comes at a computational cost. I tuned the parameters sequentially starting with fixing all parameters except for the learning rate/shrinkage (eta) and the number of rounds. Using 10 fold cross validation I optimised these parameters.

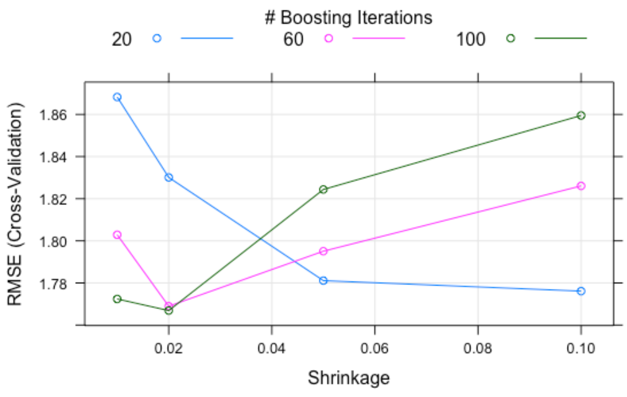

Fig 10: Line plot of cross validated rmse for varying shrinkage and rounds.

Using the same methods I optimised over the rest of my parameter set sequentially. The resulting model had the following parameter set and returned the following training and test scores:

```
Nrounds: 100
Max_depth: 8
 Eta: 0.02
 Gamma: 0
Colsample_bytree: 0.8
Min_child_weight 1
Subsample 1
RMSE = 0.914537
```

Neural Network

The second model I chose to fit was a neural network. The neural network I fit was a sequential neural network [16]. I tuned the size of the layer as well as the weight decay parameter using grid search.

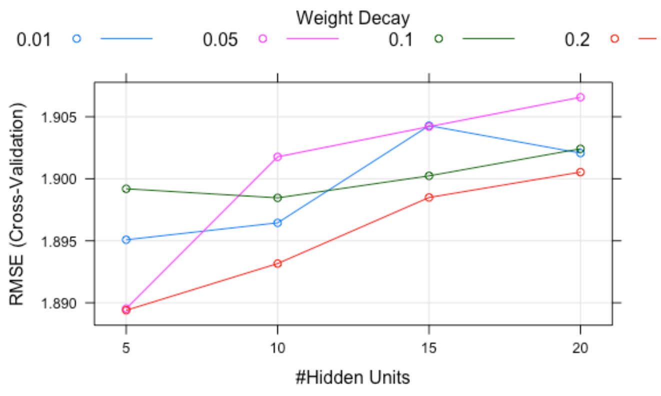

Fig 11: Line plot of rmse for varying number of hidden layers and weight decay.

The final optimised model was fit using the below parameters and resulted in the following training and test scores:

```
Size: 5
Decay: 0.2
RMSE = 0.3358464
```

Generalised Linear Model

The final model I decided to fit was a penalised regression model. The regularisation term in this case is a combination of the L1 and L2 penalty terms from ridge and lasso regression respectively. I optimised the model over the constant the elastic net [18] constants lambda and alpha resulting in the following train and test scores:

```
Alpha = 0.4
Lambda = 0.1
RMSE = 0.5669421
```

Note: the constant lambda and alpha make up the coefficients of the regularisation term where betas are the estimated coefficients:

$$\lambda\left[\frac{1-\alpha}{2}\sum_{j=1}^{m}\beta_j + \alpha\sum_{j=1}^{m}|\beta_j|\right]$$

Classification

Logistic Regression Elastic Net

I cross validated a Logistic regression model with Elastic Net penalty to predict the binary variable return denoting whether a customer returned to visit the store over the prediction period. The following is the confusion matrix of the results:

```
        ret_test_sample
pred_glmcl      X0      X1
        X0 1280148    4873
        X1   25760     832

Accuracy = 97.66%
```

From this classification model I extracted the class probabilities and multiplied these predictions with that of my final regression model.

8. Results

Having fitted these models a decision is required to choose the most suitable model for future predictions. Therefore, we need to determine the appropriate statistical test to validate our choice of model. At first it would appear that we could employ a simple t-test on the cross validated samples however the assumption of independent samples is violated. The reason for this is that we reuse training samples (k-1) times to train each model on k-fold cross validation making our cross validated samples dependent. However we can modify our training schema in order to perform these tests by simply constructing 30 completely independent samples and running models on these samples. Since our dataset is quite large this is feasible therefore proceeding in this manner I recorded the RMSE for each model run on each independent set of data.

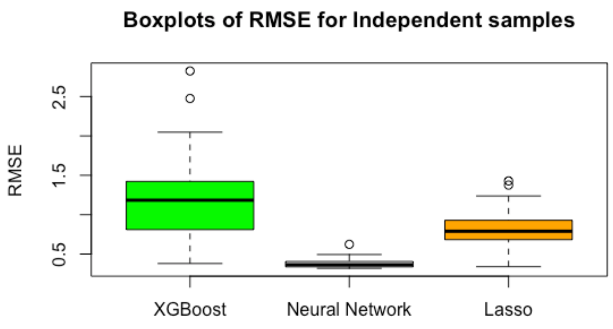

Fig 12: Boxplots of the rmse for models trained on independent samples of the data. Each train and test set pair is independent of each other pair. 30 samples were taken in total.

The distribution of our RMSE performance metric for each model is unknown. However, by the central limit theorem [17], we know that the mean of the sample of performance metrics is approximately normal (since n>=30). This allows us to employ a t-test to assess the differences in performance of our models. Therefore, I will use a one-way ANOVA to aid in the model selection process. The output of the ANOVA analysis indicates there is no significant differences in the mean performance of the three models:

```
Df Sum Sq Mean Sq F value Pr(>F)
Model         1  0.058 0.05784   0.254  0.616
Residuals   88 20.076 0.22814
```

Since the p-value 0.616>0.05 is greater than our significance level we fail to reject the null hypothesis, hence there is no statistically significant differences between the model performances of all three models. Despite this when result I also performed t-tests between individual models namely the neural network model and the penalised regression model as well as the neural network and the boosted tree model separately resulting in the following p-values (2.846e-10, 3.502e-09) which lead me to conclude that there is a statistically significant difference in the performance of the neural network and the other two models when examined individually. This is further verified by the non-parametric Wilcoxon ranked-signed test which also rejects the null hypotheses when comparing the neural network to the other model separately. This leads me to choose the neural network as the best model for minimising the RMSE. I chose the neural network since there is statistical significance under the t-test for the differences in model performance between models individually with the neural network minimising the RMSE more than the other models.

Taking the predictions of the cross validated neural network model and multipliying it with the customer return probabilities from the classification model the RMSE of predictions on the test set reduces to 0.2872951 from 0.3358464.

Running the same analysis on the entire set of data for the Kaggle competition submission resulted in a score of 0.88839 which was within the top 15% of the Kaggle leaderboard.

9. Discussion

We can clearly see that there is an application to using web traffic data in order to predict customer revenue. It is likely worthwhile attempting to collect supplementary customer data to improve predictions and gain further insights. That being said we can identify some key features of web traffic data that appear more important in predicting customer spending such as: the time between visits to the online store, the number of times a customer visited the store and the channel grouping classification of the customer i.e. the traffic channel which they visited the store from. As an example users who clicked on links from social platforms to get to the the store would have the channel classification social. From a Marketing perspective these insights are useful as we can identify the characteristics of customers who tend to spend money on the site.

In this report we have also developed machine learning models that can predict future customer revenue based on web traffic data. These models can be used in financial forecasting and reporting.

Throughout this project many of the encountered stumbling blocks were in the technologies used. Due to the size of the dataset being analysed the preprocessing and model training procedures required large computational resources. The majority of time spent on this report was spent in understanding and implementing technologies to tackle these problems. Firstly, in order to parse and preprocess the data I setup a ubuntu virtual machine on the google cloud platform using an predefined ubuntu image. On this machine I installed r software along with Hadoop and spark by following the guide outlined in [4]. I did get my code running on this virtual machine however despite this effort I was also able to improve the speed of my code running locally and hence this step became redundant. I suspect there were performance enhancement through running query commands on the google cloud cluster and given more time I would measure the differences in performance speeds. Through solving this initial challenge I learnt how to connect to a server via ssh tunnelling, use basic bash commands and write bash scripts as well as learning how to setup a virtual machine on the google cloud platform. Model training also became a performance issue. In order to improve training performance I employed parallel processing locally for the algorithms that were parallelisable. This was completed seamlessly within R using common parallel processing libraries. This allowed me to run models on large subsets of the data with reasonable speed. For training on the full and generating predictions for the Kaggle competition model training to a long time to complete. I attempted to improve the speed of training models through using the google cloud platform utility cloud ml. This utility enables users to perform model training on a google cloud platform through submitting a docker container with the training procedure and data. I didn't fully complete implementation of this method however I learnt quite a bit about the use of containers and how to construct containers. Given more time I would fully implement this method for model training. I plan to use this feature of the google cloud platform in future projects. Since I wasn't able to fully implement this infrastructure I trained models locally which generally ran overnight.

10. Conclusions

In this report I have pre-processed a dataset and performed exploratory data analysis on the data. I constructed a training and testing set to tackle the given prediction problem. I investigated correlations amongst features as well as feature importance analysis to gain extra insight on the usefulness of particular categories in predicting revenue. I fit various models to the data and tuned model parameters through combining grid search and cross validation. I evaluated model performance through employing an inferential statistical test. This resulted in a model that can be used to predict revenue generated by individual customers visiting the online store. I discovered the underlying technological challenges in analysing large sets of data as well as potential solutions to these challenges. In conclusion I have analysed a set of web traffic data and designed a method of predicting customer revenue based on this data through fitting machine learning models to the data.

11. Outlook

Given more time I would spend more time exploring the effects of feature engineering has on model performance. I feel the key to performing well in this challenge was clever feature engineering. I would also collect statistics on the length of time that models took to train as well as including an in-depth discussion on model complexity and calculation/outline of this complexity for each model. In future analysis of customer spending I would consider how to pair web traffic data with other customer data. I feel that clustering/profiling customers by web traffic and external data would be quite valuable and is something I would investigate in future work. Overall I feel that analysing customer data in the way I have outlined is of value. I hope to employ and improve upon these methods in future analysis and potentially couple the analyisis of web traffic data with other forms of data to provide more meaningful insight.

12. Acknowledgements

I would like to acknowledge my parents for supporting and encouraging my education. I would like to thank my project supervisor Dr Supratik Roy for supervising this work.

13. References

[1] https://analytics.google.com/analytics/web/

[2] https://www.kaggle.com/c/ga-customer-revenue-prediction/overview

[3] Jackson, Wallace. (2016). An Introduction to JSON: Concepts and Terminology.

[4] https://cloud.google.com/solutions/running-rstudio-server-on-a-cloud-dataproc-cluster

[5] Shoro, Abdul & Soomro, Tariq. (2015). Big Data Analysis: Apache Spark Perspective. Global Journal of Computer Science and Technology.

[6] Zhang Z. (2016). Missing data imputation: focusing on single imputation. *Ann Transl Med.* 2016;4(1):9.

[7] Salkind, N. J. (2010).*Encyclopedia of research design*Thousand Oaks, CA: SAGE Publications, Inc.

[8] Potdar, Kedar & Pardawala, Taher & Pai, Chinmay. (2017). A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. International Journal of Computer Applications. 175. 7-9.

[9] Andrew. Hall, Mark. (2000). Correlation-Based Feature Selection for Machine Learning. Department of Computer Science.

[10] Hocking, R. R. (1976) "The Analysis and Selection of Variables in Linear Regression," *Biometrics, 32.*

[11] Chen, Tianqi & Guestrin, Carlos. (2016). XGBoost: A Scalable Tree Boosting System.

[12] Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological), 58*(1), 267-288.

[13] Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York.

[14] Stone M. (1974). Cross-validatory choice and assessment of statistical predictions. J. Royal Stat. Soc., 36(2), 111–147.

[15] Tianqi Chen and Carlos Guestrin. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). ACM, New York, NY, USA.

[16] Christopher M. Bishop. (2006). Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg.

[17] Heyde, C. (2014). Central Limit Theorem . In Wiley StatsRef: Statistics Reference Online (eds N. Balakrishnan, T. Colton, B. Everitt, W. Piegorsch, F. Ruggeri and J. L. Teugels).

[18] Giglio, C, Brown, SD.(2018). Using elastic net regression to perform spectrally relevant variable selection. Journal of Chemometrics.

14. Appendix

Preprocessing & Feature Engineering.R

```r
library(plyr)
library(lubridate)
library(tidyverse)
library(gridExtra)


#1. Read in the Data
ctypes <- cols(trafficSource_adwordsClickInfo.isVideoAd = col_logical(),
               trafficSource_isTrueDirect = col_logical(),
               device_isMobile = col_logical(),
               fullVisitorId = col_character(),
               channelGrouping = col_factor(),
               date = col_datetime())

train <- read_csv("/Users/peterfagan/Desktop/train.csv",na = c("not available in demo dataset", "(not
provided)","(not set)", "<NA>", "unknown.unknown",  "(none)","","NA"),col_types = ctypes)
test <- read_csv("/Users/peterfagan/Desktop/test.csv",na = c("not available in demo dataset", "(not
provided)","(not set)", "<NA>", "unknown.unknown",  "(none)","","NA"),col_types = ctypes)


good_cols =
c("channelGrouping","date","fullVisitorId","visitNumber","visitStartTime","device_browser","device_deviceCate
gory",

"device_isMobile","device_operatingSystem","geoNetwork_city","geoNetwork_continent","geoNetwork_country","geo
Network_metro","geoNetwork_networkDomain","geoNetwork_region","geoNetwork_subContinent","totals_bounces",

"totals_hits","totals_newVisits","totals_pageviews","totals_sessionQualityDim","totals_timeOnSite","totals_to
talTransactionRevenue","totals_transactionRevenue",

"totals_transactions","trafficSource_adContent","trafficSource_adwordsClickInfo.adNetworkType","trafficSource
_adwordsClickInfo.isVideoAd",
              "trafficSource_adwordsClickInfo.page",
"trafficSource_adwordsClickInfo.slot","trafficSource_campaign","trafficSource_isTrueDirect","trafficSource_ke
yword","trafficSource_medium","trafficSource_referralPath","trafficSource_source")

train <- train %>% select(c(good_cols))
test <- test %>% select(c(good_cols))
train <- as.data.frame(train)
test <- as.data.frame(test)


#Exploratory Data Analysis

#Plot of NA values of non-numeric variables
notnum <- function(x){
  if(is.numeric(x)){
    return(FALSE)
  }
  else{
    return(TRUE)
  }
}

sub <- Filter(notnum,train)
nrows = dim(sub)[1]
ncols = dim(sub)[2]
na_count = numeric(ncols)
for(i in 1:ncols){
  na_count[i] = sum(is.na(sub[,i]))
}
df_navals = as.data.frame(names(sub))
df_navals$na_count = na_count
df_navals$na_percentage = ((df_navals$na_count/nrows)*100)
colnames(df_navals)<-c("Variables","Na_count","Na_percent")
ggplot(data=df_navals,aes(x=reorder(Variables,Na_percent),y=Na_percent))+geom_bar(stat='identity',colour='bla
ck',fill='royalblue')+coord_flip()+ggtitle("Missing Values by Feature")+xlab("Features")+ylab("Missing Values
(%)")


#(***Optional***) Drop non-numeric variables with more than 95% NA values
drop_list = as.character(df_navals[df_navals$na_percentage>0.95,][,1])
train = train[,!(names(train) %in% drop_list)]



# Target variable visualisations

#Target variable behaviour over time
train$totals_totalTransactionRevenue <-
ifelse(is.na(train$totals_totalTransactionRevenue),0,train$totals_totalTransactionRevenue)
test$totals_totalTransactionRevenue <-
ifelse(is.na(test$totals_totalTransactionRevenue),0,test$totals_totalTransactionRevenue)

target_agg <- aggregate((train$totals_totalTransactionRevenue*1e-06), by = list(train$date),sum)
plot(target_agg,t='l',main="Transaction revenue time series",xlab= "Year",ylab="Total transaction revenue")

#Target variable distribution
target = train$totals_totalTransactionRevenue
target_spent = log(target[target>0] +1)
target_spent = as.data.frame(target_spent)
colnames(target_spent)<-c("log_target")
ggplot(target_spent,aes(log_target,stat(density))) +
geom_histogram(colour='black',fill='royalblue')+ggtitle("Transaction Revenue")+xlab("log(transaction_revenue
+1)")+ylab("%")

#Distribution of expenditure
```

```r
no_spend_percent = (length(target)-(dim(target_spent)[1]))/length(target)
spend_percent = (dim(target_spent)[1])/length(target)
df_spend = as.data.frame(rbind(spend_percent,no_spend_percent))
df_spend$V2 = c("Revenue","No Revenue")
colnames(df_spend) <- c("Percent","Groups")
ggplot(df_spend,aes(x=Groups,y=Percent))+geom_bar(stat='identity',colour='black',fill='royalblue')+ggtitle('B
arplot of Site Visits Generating Revenue')

#Checking for features that remain constant or show no variability
unique_vals = apply(train,2,n_distinct)
unique_vals[unique_vals==1]


#Checking for categorical features with a large number of levels
cat_levels <- sapply(train,n_distinct)
for(i in 1:length(cat_levels)){
  if(cat_levels[i] > 32){
    print(names(cat_levels[i]))
  }
}

#Visualisations of categorical variables levels with repect to target variable
train <- as_tibble(train)

p1 <- train %>%
  group_by(channelGrouping) %>%
  summarise(TransactionRevenue = sum(totals_totalTransactionRevenue),visits = n())
P11 <- ggplot(p1, aes(x=channelGrouping,y=visits)) +
geom_bar(stat='identity',colour='black',fill='lightgreen')+ggtitle("Visits by Channel grouping") +
theme(axis.text.x = element_text(angle = 60, hjust = 1))
P12 <- ggplot(p1, aes(x=channelGrouping,y=TransactionRevenue/visits)) +
geom_bar(stat='identity',colour='black',fill='lightgreen')+ggtitle("Mean Transaction Revenue by Channel
grouping") + theme(axis.text.x = element_text(angle = 60, hjust = 1))



p2 <- train %>%
  group_by(device_operatingSystem) %>%
  summarise(TransactionRevenue = sum(totals_totalTransactionRevenue),visits = n())
P21 <- ggplot(p2, aes(x=device_operatingSystem,y=visits)) +
geom_bar(stat='identity',colour='black',fill='orange')+ggtitle("Visits by operating system") +
theme(axis.text.x = element_text(angle = 60, hjust = 1))
P22 <- ggplot(p2, aes(x=device_operatingSystem,y=TransactionRevenue/visits)) +
geom_bar(stat='identity',colour='black',fill='orange')+ggtitle("Mean Transaction Revenue by operating
system") + theme(axis.text.x = element_text(angle = 60, hjust = 1))


p3 <- train %>%
  group_by(geoNetwork_continent) %>%
  summarise(TransactionRevenue = sum(totals_totalTransactionRevenue),visits = n())
```

```r
P31 <- ggplot(p3, aes(x=geoNetwork_continent,y=visits)) +
geom_bar(stat='identity',colour='black',fill='yellow')+ggtitle("Visits by Continent") + theme(axis.text.x =
element_text(angle = 60, hjust = 1))
P32 <- ggplot(p3, aes(x=geoNetwork_continent,y=TransactionRevenue/visits)) +
geom_bar(stat='identity',colour='black',fill='yellow')+ggtitle("Mean Transaction Revenue by Continent") +
theme(axis.text.x = element_text(angle = 60, hjust = 1))

p4 <- train %>%
  group_by(device_deviceCategory) %>%
  summarise(TransactionRevenue = sum(totals_totalTransactionRevenue),visits = n())
P41 <- ggplot(p4, aes(x=device_deviceCategory,y=visits)) +
geom_bar(stat='identity',colour='black',fill='lightblue')+ggtitle("Visits by device category") +
theme(axis.text.x = element_text(angle = 60, hjust = 1))
P42 <- ggplot(p4, aes(x=device_deviceCategory,y=TransactionRevenue/visits)) +
geom_bar(stat='identity',colour='black',fill='lightblue')+ggtitle("Mean Transaction Revenue by device
category") + theme(axis.text.x = element_text(angle = 60, hjust = 1))


grid.arrange(P11,P12,P21,P22,P31,P32,P41,P42, ncol = 2, nrow = 4)


g1 <- ggplot(train, aes(x=train$channelGrouping,y=train$totals_totalTransactionRevenue)) + geom_boxplot() +
theme(axis.text.x = element_text(angle = 60, hjust = 1))
g2 <- ggplot(train, aes(x=train$device_operatingSystem,y=train$totals_totalTransactionRevenue)) +
geom_boxplot()+ theme(axis.text.x = element_text(angle = 60, hjust = 1))
g3 <- ggplot(train, aes(x=train$geoNetwork_continent,y=train$totals_totalTransactionRevenue)) +
geom_boxplot()+ theme(axis.text.x = element_text(angle = 60, hjust = 1))
g4 <- ggplot(train, aes(x=train$device_deviceCategory,y=train$totals_totalTransactionRevenue)) +
geom_boxplot()+ theme(axis.text.x = element_text(angle = 60, hjust = 1))

grid.arrange(g1,g2,g3,g4, ncol = 1, nrow = 4)

train <- as.data.frame(train)

Feature_engineering <- function(data)
{
  #Preprocessing features some more...
  for(i in 4:dim(data)[2]){
    if(is.character(data[,i])){
      data[,i] = as.factor(data[,i])
    }
  }
  data$visitStartTime = as_datetime(data$visitStartTime)
  data$device_deviceCategory = as.factor(data$device_deviceCategory)
  data$device_isMobile <- as.numeric(as.factor(data$device_isMobile))
  data$trafficSource_isTrueDirect =
as.numeric(ifelse(is.na(data$trafficSource_isTrueDirect),FALSE,data$trafficSource_isTrueDirect))
  data$trafficSource_adwordsClickInfo.isVideoAd <-
(ifelse(is.na(data$trafficSource_adwordsClickInfo.isVideoAd),0,1))
```

```r
  # Feature Engineering
  #Reducing the dimension of categorical features through grouping levels with few occurences into the one
level
  data$device_browser = fct_lump(data$device_browser,n=10)
  data$device_operatingSystem = fct_lump(data$device_operatingSystem,n=10)

  #Make NA values a factor level
  for(i in 1:dim(data)[2]){
    if(is.factor(data[,i])){
      data[,i] = fct_explicit_na(data[,i])
    }
  }

  #Make dates into hour,day,week,month as separate variables
  data$month_day <- as.factor(as.integer(mday(data$date)))
  data$month <- as.factor(as.character(months(data$date)))
  data$hour <- as.factor(as.POSIXlt(data$visitStartTime,origin='1970-01-01')$hour)
  data$weekday <- as.factor(wday(as.POSIXlt(data$visitStartTime, origin='1970-01-01')))



  #Target encoding geographical features
  wealth_continent <- aggregate(totals_totalTransactionRevenue ~ geoNetwork_continent, data, mean)
  wealth_subContinent <- aggregate(totals_totalTransactionRevenue ~ geoNetwork_subContinent, data, mean)
  wealth_country <- aggregate(totals_totalTransactionRevenue ~ geoNetwork_country, data, mean)
  wealth_city <- aggregate(totals_totalTransactionRevenue ~ geoNetwork_city, data, mean)
  wealth_metro <- aggregate(totals_totalTransactionRevenue ~ geoNetwork_metro, data, mean)
  wealth_region <- aggregate(totals_totalTransactionRevenue ~ geoNetwork_region, data, mean)

  wealth_continent[,2] <-
scale(wealth_continent[,2],center=min(wealth_continent[,2]),scale=max(wealth_continent[,2])-
min(wealth_continent[,2]))
  wealth_subContinent[,2] <-
scale(wealth_subContinent[,2],center=min(wealth_subContinent[,2]),scale=max(wealth_subContinent[,2])-
min(wealth_subContinent[,2]))
  wealth_country[,2] <-
scale(wealth_country[,2],center=min(wealth_country[,2]),scale=max(wealth_country[,2])-
min(wealth_country[,2]))
  wealth_city[,2] <- scale(wealth_city[,2],center=min(wealth_city[,2]),scale=max(wealth_city[,2])-
min(wealth_city[,2]))
  wealth_metro[,2] <- scale(wealth_metro[,2],center=min(wealth_metro[,2]),scale=max(wealth_metro[,2])-
min(wealth_metro[,2]))
  wealth_region[,2] <- scale(wealth_region[,2],center=min(wealth_region[,2]),scale=max(wealth_region[,2])-
min(wealth_region[,2]))



  colnames(wealth_continent) <- c("geoNetwork_continent","wealth_continent")
  colnames(wealth_subContinent) <- c("geoNetwork_subContinent","wealth_subContinent")
  colnames(wealth_country) <- c("geoNetwork_country","wealth_country")
  colnames(wealth_city) <- c("geoNetwork_city","wealth_city")
  colnames(wealth_metro) <- c("geoNetwork_metro","wealth_metro")
  colnames(wealth_region) <- c("geoNetwork_region","wealth_region")


  data <- join(data,wealth_continent,by="geoNetwork_continent",type='left')
  data <- join(data,wealth_subContinent,by="geoNetwork_subContinent",type='left')
  data <- join(data,wealth_country,by="geoNetwork_country",type='left')
  data <- join(data,wealth_city,by="geoNetwork_city",type='left')
  data <- join(data,wealth_metro,by="geoNetwork_metro",type='left')
  data <- join(data,wealth_region,by="geoNetwork_region",type='left')

  data$wealth_continent <- as.numeric(data$wealth_continent)
  data$wealth_subContinent <- as.numeric(data$wealth_subContinent)
  data$wealth_country <- as.numeric(data$wealth_country)
  data$wealth_city <- as.numeric(data$wealth_city)
  data$wealth_metro <- as.numeric(data$wealth_metro)
  data$wealth_region <- as.numeric(data$wealth_region)

  #Dropping columns that are no longer needed
  data$geoNetwork_continent <- NULL
  data$geoNetwork_subContinent <- NULL
  data$geoNetwork_country <-NULL
  data$geoNetwork_city <- NULL
  data$geoNetwork_networkDomain <- NULL
  data$geoNetwork_metro <-NULL
  data$geoNetwork_region <- NULL
  data$trafficSource_adContent  <- NULL
  data$trafficSource_adwordsClickInfo.adNetworkType<- NULL
  data$trafficSource_adwordsClickInfo.page    <- NULL
  data$trafficSource_adwordsClickInfo.slot  <- NULL
  data$trafficSource_campaign    <- NULL
  data$trafficSource_keyword      <- NULL
  data$trafficSource_medium    <- NULL
  data$trafficSource_referralPath   <- NULL
  data$trafficSource_source   <- NULL

  #Making features numeric
  for(i in 1:dim(data)[2]){
    if(is.factor(data[,i])){
      data[,i] <- as.numeric(data[,i])
    }
  }
  return(data)
}

train <- Feature_engineering(train)
test <- Feature_engineering(test)
#Formatting original training set to reflect the prediction problem.
#Data for 168 days is combined with target variable calulated over a subsequent
```

```r
#period of 62 days taking place 46 days in advance of initial period.
train$date <- ymd(train$date)
test$date <- ymd(test$date)
(max(test$date) - min(test$date))
(max(train$date) - min(train$date))

Generate_training_set <- function(data,t)
{

  #Split into windows corresponding to the traffic data and the period used to calculate
  #the target variable.
  traffic_dat_window <- data[data$date >= min(data$date)+(168*t) & data$date <= min(data$date)+(168*(t+1)),]
  target_calc_window <- data[data$date >= min(data$date)+(168*(t+1))+46 & data$date <=
min(data$date)+(168*(t+1))+46+62,]


  #Returning and non-returning customers ids
  target_fvid = unique(target_calc_window$fullVisitorId)
  ret_fvid = traffic_dat_window[(traffic_dat_window$fullVisitorId %in% target_fvid),]$fullVisitorId
  nret_fvid = traffic_dat_window[!(traffic_dat_window$fullVisitorId %in% target_fvid),]$fullVisitorId

  #target window filtered by returned customers
  target_calc_window_ret = target_calc_window[target_calc_window$fullVisitorId %in% ret_fvid, ]

  #Create a dataframe for training
  dtrain = aggregate(totals_totalTransactionRevenue ~ fullVisitorId, target_calc_window_ret,
function(x){log(1+sum(x))})
  dtrain$return = 1
  colnames(dtrain) = c("fullVisitorId","target","return")
  dtrain_nret = data.frame(fullVisitorId = nret_fvid, target = 0, return = 0)
  dtrain = rbind(dtrain, dtrain_nret)
  dtrain_maxdate = max(traffic_dat_window$date)
  dtrain_mindate = min(traffic_dat_window$date)



  getmode <- function(v) {
    uniqv <- unique(v)
    uniqv[which.max(tabulate(match(v, uniqv)))]
  }

  features <- traffic_dat_window %>%
    group_by(fullVisitorId) %>%
    summarise(
      channelGrouping = getmode(ifelse(is.na(channelGrouping),-9999,channelGrouping)),
      first_ses_from_the_period_start = min(date) - dtrain_mindate,
      last_ses_from_the_period_end = dtrain_maxdate - max(date),
      interval_dates = max(date) - min(date),
      unique_date_num = length(unique(date)),
      maxVisitNum = max(visitNumber,na.rm=TRUE),
      browser = getmode(ifelse(is.na(device_browser),-9999,device_browser)),
      operatingSystem =  getmode(ifelse(is.na(device_operatingSystem),-9999,device_operatingSystem)),
      deviceCategory =  getmode(ifelse(is.na(device_deviceCategory),-9999,device_deviceCategory)),
      wealth_continent = mean(wealth_continent),
      wealth_subContinent = mean(wealth_subContinent),
      wealth_country = mean(wealth_country),
      wealth_region = mean(wealth_region),
      wealth_metro = mean(wealth_metro) ,
      wealth_city = mean(wealth_city),
      isVideoAd_mean = mean(trafficSource_adwordsClickInfo.isVideoAd),
      isMobile = mean(ifelse(device_isMobile == TRUE, 1 , 0)),
      isTrueDirect = mean(ifelse(is.na(trafficSource_isTrueDirect) == TRUE, 1, 0)),
      bounce_sessions = sum(ifelse(is.na(totals_bounces),0,totals_bounces)),
      hits_sum = sum(totals_hits),
      hits_mean = mean(totals_hits),
      hits_min = min(totals_hits),
      hits_max = max(totals_hits),
      hits_median = median(totals_hits),
      pageviews_sum = sum(totals_pageviews, na.rm = TRUE),
      pageviews_mean = mean(totals_pageviews, na.rm = TRUE),
      pageviews_min = min(totals_pageviews, na.rm = TRUE),
      pageviews_max = max(totals_pageviews, na.rm = TRUE),
      pageviews_median = median(totals_pageviews, na.rm = TRUE),
      session_cnt = NROW(visitStartTime),
      transactions  = sum(totals_transactions,na.rm = TRUE)
    )

  features <- as.data.frame(features)
  dtrain = join(dtrain, features, by = "fullVisitorId")
  glimpse(dtrain)

  #Setting found NA values to zero (Only 10 occurences)
  dtrain$pageviews_mean = ifelse(is.na(dtrain$pageviews_mean),0,dtrain$pageviews_mean)
  dtrain$pageviews_median = ifelse(is.na(dtrain$pageviews_median),0,dtrain$pageviews_median)
  dtrain$pageviews_max <- ifelse(is.infinite(dtrain$pageviews_max),0,dtrain$pageviews_max)
  dtrain$pageviews_min <- ifelse(is.infinite(dtrain$pageviews_min),0,dtrain$pageviews_min)

  return(dtrain)
}

tr0 <- Generate_training_set(train,0)
tr1 <- Generate_training_set(train,1)
tr2 <- Generate_training_set(train,2)
training_set <- rbind(tr0,tr1,tr2)




Generate_testing_set <- function(data)
```

```r
{
  dtrain_maxdate = max(data$date)
  dtrain_mindate = min(data$date)



  getmode <- function(v) {
    uniqv <- unique(v)
    uniqv[which.max(tabulate(match(v, uniqv)))]
  }

  features <- data %>%
    group_by(fullVisitorId) %>%
    summarise(
      channelGrouping = getmode(ifelse(is.na(channelGrouping),-9999,channelGrouping)),
      first_ses_from_the_period_start = min(date) - dtrain_mindate,
      last_ses_from_the_period_end = dtrain_maxdate - max(date),
      interval_dates = max(date) - min(date),
      unique_date_num = length(unique(date)),
      maxVisitNum = max(visitNumber,na.rm=TRUE),
      browser = getmode(ifelse(is.na(device_browser),-9999,device_browser)),
      operatingSystem =  getmode(ifelse(is.na(device_operatingSystem),-9999,device_operatingSystem)),
      deviceCategory =  getmode(ifelse(is.na(device_deviceCategory),-9999,device_deviceCategory)),
      wealth_continent = mean(wealth_continent),
      wealth_subContinent = mean(wealth_subContinent),
      wealth_country = mean(wealth_country),
      wealth_region = mean(wealth_region),
      wealth_metro = mean(wealth_metro) ,
      wealth_city = mean(wealth_city),
      isVideoAd_mean = mean(trafficSource_adwordsClickInfo.isVideoAd),
      isMobile = mean(ifelse(device_isMobile == TRUE, 1 , 0)),
      isTrueDirect = mean(ifelse(is.na(trafficSource_isTrueDirect) == TRUE, 1, 0)),
      bounce_sessions = sum(ifelse(is.na(totals_bounces),0,totals_bounces)),
      hits_sum = sum(totals_hits),
      hits_mean = mean(totals_hits),
      hits_min = min(totals_hits),
      hits_max = max(totals_hits),
      hits_median = median(totals_hits),
      pageviews_sum = sum(totals_pageviews, na.rm = TRUE),
      pageviews_mean = mean(totals_pageviews, na.rm = TRUE),
      pageviews_min = min(totals_pageviews, na.rm = TRUE),
      pageviews_max = max(totals_pageviews, na.rm = TRUE),
      pageviews_median = median(totals_pageviews, na.rm = TRUE),
      session_cnt = NROW(visitStartTime),
      transactions  = sum(totals_transactions,na.rm = TRUE)
    )

  features <- as.data.frame(features)
  #Setting found NA values to zero (Only 10 occurences)
  features$pageviews_mean = ifelse(is.na(features$pageviews_mean),0,features$pageviews_mean)
  features$pageviews_median = ifelse(is.na(features$pageviews_median),0,features$pageviews_median)
  features$pageviews_max <- ifelse(is.infinite(features$pageviews_max),0,features$pageviews_max)
  features$pageviews_min <- ifelse(is.infinite(features$pageviews_min),0,features$pageviews_min)


  return(features)
}

testing_set <- Generate_testing_set(test)


write_csv(training_set,path="/Users/peterfagan/Desktop/training_set.csv")
write_csv(testing_set,path="/Users/peterfagan/Desktop/testing_set.csv")
```

Model Fitting.R

```r
library(plyr)
library(reshape2)
library(lubridate)
library(tidyverse)
library(gridExtra)
library(xgboost)
library(Matrix)
library(caret)
library(glmnet)
library(randomForest)

library(parallel)
library(doParallel)
library(cloudml)

train <- read_csv("/Users/peterfagan/Desktop/training_set.csv")
test <- read_csv("/Users/peterfagan/Desktop/testing_set.csv")

train <- as.data.frame(train)
test <- as.data.frame(test)
train$fullVisitorId <- NULL


### Splitting up datasets for analyisis ###

#Note in report we need to be able to evaluate performance.
#Therefore we require target variable values for test set.

#Splitting into training and test sets (smaller samples for quick code)
N_sample = round(nrow(train)*0.01)
itrain_sample = sample(1:N_sample,size=round(N_sample*0.7),replace=FALSE)
x_train_sample <- train[itrain_sample,c(-1,-2)]
```

```r
x_test_sample <- train[-itrain_sample,c(-1,-2)]
target_train_sample <- train[itrain_sample,1]
target_test_sample <- train[-itrain_sample,1]
ret_train_sample <- as.factor(make.names(train[itrain_sample,2]))
ret_test_sample <- as.factor(make.names(train[-itrain_sample,2]))


#Full samples
N = nrow(train)
itrain = sample(1:N,size=round(N*0.7),replace=FALSE)
x_train <- train[itrain,c(-1,-2)]
x_test <- train[-itrain,c(-1,-2)]
target_train <- train[itrain,1]
target_test <- train[-itrain,1]
ret_train <- as.factor(make.names(train[itrain,2]))
ret_test <- as.factor(make.names(train[-itrain,2]))


x = data.matrix(x_train_sample)
y = target_train_sample
y_ret = ret_train_sample
x_t = data.matrix(x_test_sample)

### Feature Analysis ###

#Variable Importance (Interpretability)
param <- list(max.depth = 5, eta = 0.1,  objective="reg:linear",subsample=0.9)
xgb_mod <- xgboost(param, data = x, label =y,nround = 10)
pred_xgb <- predict(xgb_mod, x_t)
xgb_imp <- xgb.importance(feature_names = colnames(x),model = xgb_mod)
xgb.plot.importance(xgb_imp)
xgb.sel <- xgb_imp$Feature

#Variable Correlations
cor_mat <- round(cor(x[,-18]),2)
high_cor <- findCorrelation(cor_mat,cutoff=0.8)
colnames(x)[high_cor]
melted_cor_mat <- melt(cor_mat)
ggplot(data = melted_cor_mat, aes(x=Var1, y=Var2, fill=value)) + geom_tile() + theme(axis.text.x =
element_text(angle = 60, hjust = 1))



### HyperParameter Tuning (Regression) ###

#XGBoost (Regression)
#Grid search to tune parameters.
#For the case of brevity of run time it has been minimised here.

xgb_trcontrol = trainControl(
  method = "cv",
  number = 10,
  verboseIter = FALSE
)

xgbGrid <- expand.grid(nrounds = c(20,60,100),
                       max_depth = 8,
                       eta = c(0.01,0.02,0.05,0.1),
                       gamma=0,
                       min_child_weight = 1,
                       colsample_bytree = 0.8,
                       subsample = 1
)

cluster <- makeCluster(detectCores() - 1) #leave 1 core for Operating System
registerDoParallel(cluster)

set.seed(150)
xgb_model = caret::train(
  x, y,
  trControl = xgb_trcontrol,
  tuneGrid = xgbGrid,
  method = "xgbTree",
  metric="RMSE",
  allowParallel = TRUE
)

stopCluster(cluster)

xgb_pred <- predict(xgb_model,x_t)
(RMSE_xgb <- sqrt(mean((xgb_pred-target_test_sample)^2)))

plot(xgb_model)
xgb_model$bestTune

#Neural Network (Regression)
#Grid search to tune parameters.
#For the case of brevity of run time it has been mimimised here.

nnet_trcontrol <- trainControl(
  method = 'cv',
  number = 10,
  verboseIter = FALSE
)

nnetGrid <- expand.grid(
  size = c(5,10,15,20),
  decay = c(0.01,0.05,0.1,0.2)
)
```

```r
cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
registerDoParallel(cluster)

set.seed(150)
nnet_model = caret::train(
  x, y,
  trControl = nnet_trcontrol,
  tuneGrid = nnetGrid,
  method = "nnet",
  metric="RMSE",
  preProcess=c("scale","center"),
  allowParallel = TRUE
)


stopCluster(cluster)

nnet_pred <- predict(nnet_model,x_t)
(RMSE_nnet <- sqrt(mean((nnet_pred-target_test_sample)^2)))

plot(nnet_model)
nnet_model$bestTune



#Lasso Regression

glmnet_trcontrol <- trainControl(
  method = 'cv',
  number = 10,
  verboseIter = FALSE
)

glmnetGrid <- expand.grid(
  alpha =  seq(0,1,0.1),
  lambda = seq(0.001,0.1,by = 0.001)
  )

registerDoSEQ()
set.seed(150)
glmnet_model <- train(
  x, y,
  method = "glmnet",
  trControl = glmnet_trcontrol ,
  preProcess=c("scale","center"),
  tuneGrid = glmnetGrid)

pred_glmnet <- predict(glmnet_model, newx = x_t, s = "lambda.min")
(RMSE_glm <- sqrt(mean((target_test_sample-pred_glmnet)^2)))

glmnet_model$bestTune
```

```r
### Model Comparison ###
results <- resamples(list(xgboost=xgb_model, NeuralNetwork=nnet_model,Lasso =glmnet_model ))
results$values
summary(results)
bwplot(results)

rmse_xgb = rmse_nnet = rmse_glmnet = numeric(10)
folds = cut(1:30000,30,labels = FALSE)
random_folds = sample(folds,10000)

for(i in 1:30){
  itrain = which(random_folds==i)

  x_train <- data.matrix(train[itrain,c(-1,-2)])
  x_test <- data.matrix(train[-itrain,c(-1,-2)])
  y_train <- train[itrain,1]
  y_test <- train[-itrain,1]


  cluster <- makeCluster(detectCores() - 1) #leave 1 core for Operating System
  registerDoParallel(cluster)


  xgb_model = caret::train(
    x_train, y_train,
    method = "xgbTree",
    metric="RMSE",
    allowParallel = TRUE
  )

  stopCluster(cluster)

  xgb_pred <- predict(xgb_model,x_test)
  rmse_xgb[i] <- sqrt(mean((xgb_pred-y_test)^2))


  cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
  registerDoParallel(cluster)

  nnet_model = caret::train(
    x_train, y_train,
    method = "nnet",
    metric="RMSE",
    preProcess=c("scale","center")
  )

  stopCluster(cluster)
  nnet_pred <- predict(nnet_model,x_test)
  rmse_nnet[i] <- sqrt(mean((nnet_pred - y_test)^2))


  registerDoSEQ()
  glmnet_model <- train(
    x_train, y_train,
    method = "glmnet"
    )

  pred_glmnet <- predict(glmnet_model, newx = x_test)
  rmse_glmnet[i] <- sqrt(mean((y_test-pred_glmnet)^2))


}

boxplot(rmse_xgb,rmse_nnet,rmse_glmnet,
        col = c("green","white","orange"),
        names=c("XGBoost","Neural Network","Lasso"),
        main="Boxplots of RMSE for Independent samples",
        ylab = "RMSE")

#Normality Assumption
shapiro.test(rmse_xgb) #Fails normaility test
shapiro.test(rmse_nnet) #Fails normality test
shapiro.test(rmse_glmnet)

#Assumption of homogeneity of variances
bartlett.test(dat$RMSE,dat$Model) #passes implies unequal variances

rmses <- c(rmse_xgb,rmse_nnet,rmse_glmnet)
groups <- as.factor(c(replicate(10,"rmse_xgb"), replicate(10,"rmse_nnet"), replicate(10,"rmse_glmnet")))
dat <- as.data.frame(cbind(groups,rmses))
colnames(dat)=c("Model","RMSE")
anova <- aov(RMSE~Model,data=dat)
summary(anova) #Not significant at 0.05 level

t.test(rmse_nnet,rmse_glmnet,paired = TRUE)
t.test(rmse_nnet,rmse_xgb,paired = TRUE)
t.test(rmse_xgb, rmse_glmnet,pared=TRUE)




### Hyperparameter tuning (classification) ###

#Next we aim to improve our above predictions through additionally predicting the
#likelihood of a customer returning. This will be treated as a binary classification,
#problem from which I will extract class probabilities
```

```r
#Neural Network (Classification)
nnetcl_trcontrol <- trainControl(
  method = 'cv',
  number = 10,
  verboseIter = FALSE,
  classProbs=TRUE,
  summaryFunction=twoClassSummary
)

nnetclGrid <- expand.grid(
  size = 10,
  decay = 0.01
)

cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
registerDoParallel(cluster)

nnetcl_model <- caret::train(
  x, y_ret,
  trControl = nnetcl_trcontrol,
  method = "nnet",
  metric="ROC",
  preProcess=c("scale","center"),
  allowParallel = TRUE
)


stopCluster(cluster)

#Determining the classification accuracy
nnetcl_pred <- predict(nnetcl_model,x_t,type="prob")
tb = table(max.col(nnetcl_pred),ret_test_sample)
(acc_nnetcl = sum(diag(tb))/sum(tb))

#Lasso for Classification
glmcl_mod <- cv.glmnet(x, y_ret, alpha = 1, family="binomial",type.measure = "auc", nfolds = 10)
pred_glmcl <- predict(glmcl_mod, newx = x_t, s = "lambda.min",type="class")
(tb = table(pred_glmcl,ret_test_sample))
(acc_glmcl = sum(diag(tb))/sum(tb))

pred_glmcl <- predict(glmcl_mod, newx = x_t, s = "lambda.min",type="response")



### Final Model & Prediction ###
nnet_trcontrol <- trainControl(
  method = 'cv',
  number = 10,
  verboseIter = FALSE
)
```

```r
nnetGrid <- expand.grid(
  size = c(5,10,15,20),
  decay = c(0.01,0.05,0.1,0.2)
)

cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
registerDoParallel(cluster)







set.seed(150)
nnet_model = caret::train(
  x, y,
  trControl = nnet_trcontrol,
  tuneGrid = nnetGrid,
  method = "nnet",
  metric="RMSE",
  preProcess=c("scale","center"),
  allowParallel = TRUE
)


stopCluster(cluster)

nnet_pred <- predict(nnet_model,x_t)


final_prediction <- pred_glmcl*nnet_pred
(final_rmse <- sqrt(mean((target_test_sample - final_prediction)^2)))





### Kaggle Competition submission (Run as google cloudml job) ###
x = data.matrix(x_train)
y = target_train
y_ret = ret_train
x_t = data.matrix(x_test)

nnet_trcontrol <- trainControl(
  method = 'cv',
  number = 10,
  verboseIter = FALSE
)
```

```r
nnetGrid <- expand.grid(
  size = c(5,10,15,20,25,30),
  decay = c(0.01,0.05,0.1,0.2)
)

cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
registerDoParallel(cluster)

set.seed(150)
nnet_model = caret::train(
  x, y,
  trControl = nnet_trcontrol,
  tuneGrid = nnetGrid,
  method = "nnet",
  metric="RMSE",
  preProcess=c("scale","center"),
  allowParallel = TRUE
)


stopCluster(cluster)

nnet_pred <- predict(nnet_model,x_t)

glmcl_mod <- cv.glmnet(x, y_ret, alpha = 1, family="binomial",type.measure = "auc", nfolds = 10)
pred_glmcl <- predict(glm_mod, newx = x_t, s = "lambda.min",type="class")


final_prediction = nnet_pred*pred_glmcl
```