UNIVERSITY OF MINNESOTA
Driven to Discover℠

Google Analytics

Predicting purchases for the Google Store

MSBA 6420 Predictive Analytics

Final Project

Professor Yicheng Song.

Danny Moncada
54451738

Eduardo Chavez
8003965

INDEX

Business Case

The 80/20 rule has proven true for many businesses, this is, only a small percentage of customers produce most of the revenue. As such, marketing teams are challenged to make appropriate investments in promotional strategies. Online retailers is one of the many channels a store uses to promote its merchandise, for Google, its online store is the main site for shopping. A deep analysis of online retailers requires understanding of several variables related to the clickstream generated by visitors. Google, with more than 100,000 products to offer, is not the exception.

The subject of this analysis, the Google Store, sells a variety of products from clothes to hardware. There's no specific target audience since there's a product for all the different demographics and possible segmentations. Google Store has multiple navigation paths to complete a sale. The store is accessible from multiple countries and Google Search service is the most common *entrance* to their front end.

Currently, we see a large amount of visits but only 1.7% can be identified as purchases. Therefore, the key challenge here is to design an automated modeling process that identifies behaviors that leads to full purchases. Ideally, we would uncover patterns hidden in the data sets that could help marketing specialists to design a strategy. Furthermore, a machine learning model that predicts a sale, leads to the enhancement of those marketing techniques.

Solution Overview

This document will provide an understanding of the modeling techniques and machine learning algorithms that were used against the Google Store data set. The document will provide three main business solutions:

- Data requirements. A list of attributes that represents the largest amount of data points and a list of potential attributes that Google may need to start gathering or merging in case the data already exists in their transactional logs.
- Model Selection. A machine learning algorithm that provides prediction capabilities for individual use cases, as well as the interpretation of the model.
- Model Evaluation. Evaluation metrics that proves the efficiency of our machine learning pipeline along with recommendations to improve it.

The document goes through the different phases of the Machine Learning modeling:

1. Data Cleaning
2. Data Preparation
3. Hyperparameters
4. Model Selection

5. Model Testing
6. Model Evaluation

Technical Specifications

Windows VM

- MSBA - PCL3 VM
- CPU Intel Xeon CPU E5-2695 v3 @ 2.30GHz
- RAM 96GB

Python Libraries

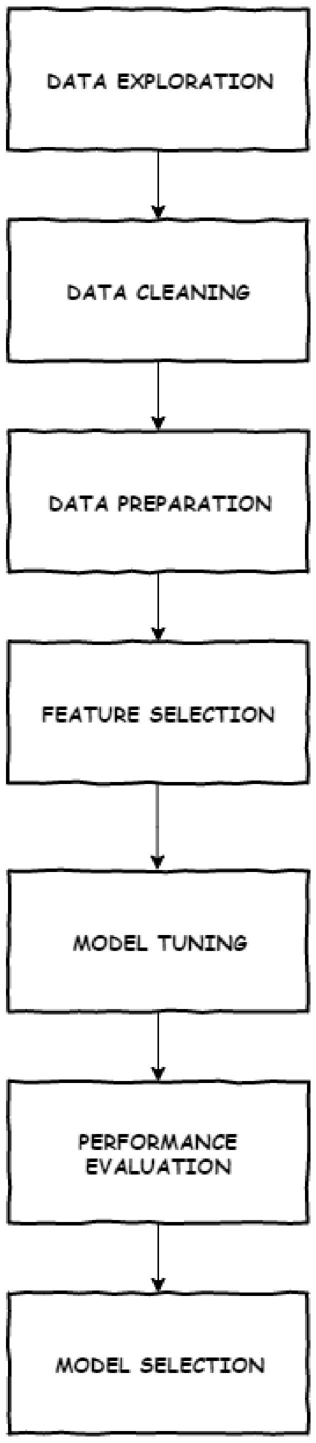
- Python 3.7
- Keras 2.3.0
- Tensorflow 2.0.0
- Pandas 0.24.2

Databricks Community Edition

- 6GB Memory
- Cluster Architecture
- 0.88 Cores
- DBR6.1
- Scala 2,11
- 8GB GPU

.

Methodology



The first step involves exploring the distribution of the dataset, performing normality tests and assessing general descriptive statistics of the dimensions. Is in this phase that we will uncover any missing data or any issue with the dataset.

The complexity of the dimensions will be exposed in this phase. The data exploration leads us to cleaning the data set, missing values, null values and analyzing categorical values, their cardinality and nature, as well as other dimension data type mismatches.

During the preparation we will follow up on the cleaning effort. Encoding categorical values, ensembling vectors, setting up the training and testing datasets as well as the identifying of the label, our target variable.

The feature selection is key to disregard the noise and complexity of the model. By assessing the dimensions that holds as much variety as possible we will be able to reduce the number of attributes and mitigate the data sparsity phenomenon better known as the *Curse of Dimensionality*.

After finding the best dimensions, we will start using techniques to test a reasonable amount of hyperparameters for every algorithm we want to test. The output of this phase is a group of settings containing the best parameters for our algorithm.

We will then compare with other algorithms using the best parameters and apply it against our test data set and evaluate the performance. We will perform a series of statistical tests to assess its accuracy.

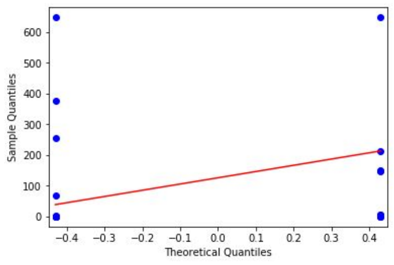
After analyzing the evaluation metrics we will then interpret the results and proceed to select the best model. We expect to obtain a generalized algorithm that performs optimally.

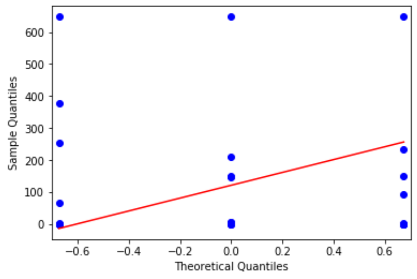
Data Exploration

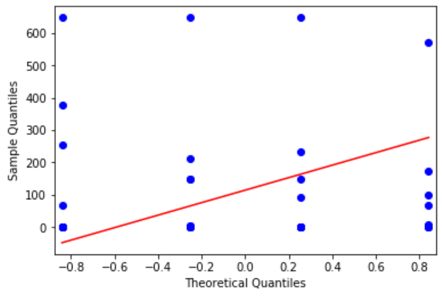
- Data contains 1.7M observations.
- There are 3 fields with nested JSON attributes
    - Focusing on Totals and Hits attributes. These two columns contain interesting information. Hits has page views, clicks and others and Totals contains Revenue. This is the metric we wish to predict.
- Flattening this data can benefit us in reducing rows and complexity. An initial flattening analysis is showing that we can reduce the number of rows to 903653.
- Date dimension is in Integer format. We need to convert this to a string datatype.
- 1.27% (11515 rows) only contains revenue data.
- We have several categorical columns, most of which will present null values:
    - 'channelGrouping'
    - 'socialEngagementType'
    - 'geoNetwork.city'
    - 'geoNetwork.country'
    - 'trafficSource.adwordsClickInfo.adNetworkType'
    - 'trafficSource.source'
- Numerical attributes:
    - 'visitNumber'
    - 'Totals.hits'
    - 'Totals.pageviews'
    - 'Totals.visits'
    - 'totals.transactionRevenue'
- Other than null values, we did not find significant missing values.
- Testing normality:

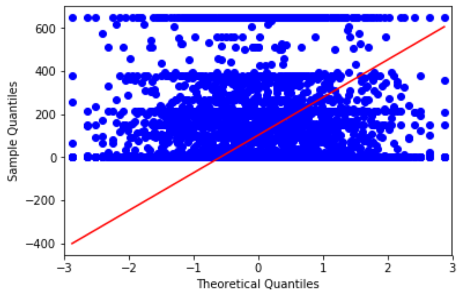
  QQPlot

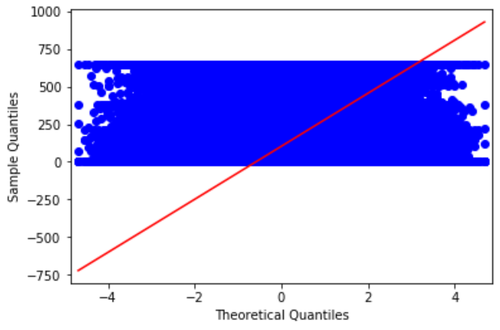
    - The QQPlot shows the complexity of the Data, It lacks of normality. With a quite low P-Value is fair to reject the hypothesis. “Normal Q-Q Plot” provides a graphical way to determine the level of normality. The red line indicates the values your sample should adhere to if the distribution was normal. The dots are actual data.
    - If the dots fall exactly on the red line, then the data is normal. If they deviate from the black line, the data is non-normal. After testing all the attributes we find the dots distributed across the canvas of the plot. The following charts shows the distribution of the data for a sample of attributes across the Theoretical Quantiles:


Sample Quantiles
Theoretical Quantiles


Sample Quantiles
Theoretical Quantiles


Sample Quantiles
Theoretical Quantiles


Sample Quantiles
Theoretical Quantiles


Sample Quantiles
Theoretical Quantiles

Shapiro Test

Statistics=0.642, p=0.000

Sample does not look Gaussian (reject H0)

Shapiro-Wilk (S-W) test is designed to test normality by comparing the data to a normal distribution with the same mean and standard deviation of the sample. If the test is NOT significant, then the data are normal, any value above .05 indicates normality. If the test is significant (less than .05), then the data are non-normal. In this case is evident that the p-value is significantly lower than .05.

Data Cleaning

DECISION #1

1. We decided to scale down revenue values dividing by 1000000.
2. All the missing revenue values were filled up with 0.
3. For our Classification problem, we derived a column called Purchase containing 1 or 0 depending on the revenue amount.
4. The rest of the missing categorical values were filled up with ‘na’ in order to avoid issues with our algorithms.

Data Preparation

The flattening of the file includes deconstructing the JSON portion inside. The following is an example highlighting the important metrics we need extract:

```
{

"visits":"1",

  "hits":"17",

  "pageviews":"13",

  "timeOnSite":"611",

  "transactions":"1",

  "transactionRevenue":"24980000",

  "totalTransactionRevenue":"26980000",

  "sessionQualityDim":"20"

}
```

For this purpose we implemented a flattening code that iterates over every attribute inside the JSON Code. Revenue will be scaled logarithmically. The algorithm to predict is the following:

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

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

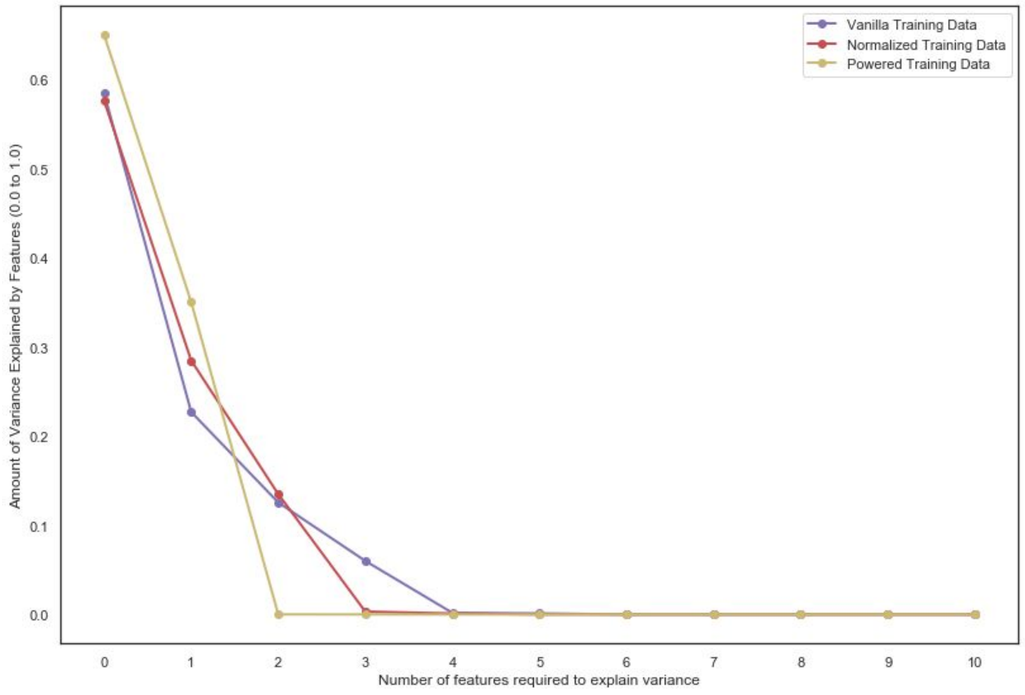
DECISION #2

To comply with the equation above, part of the flattening is aggregating the user data at the day level, which explains why the number of records went down. At this point our data is ready to start evaluating our features.

Feature Selection

PCA

Principal Component Analysis is used to explain the variance-covariance structure of a set of variables through linear combinations. In this case, due to the nature of the business case, we are trying to use it as a dimensionality-reduction technique.



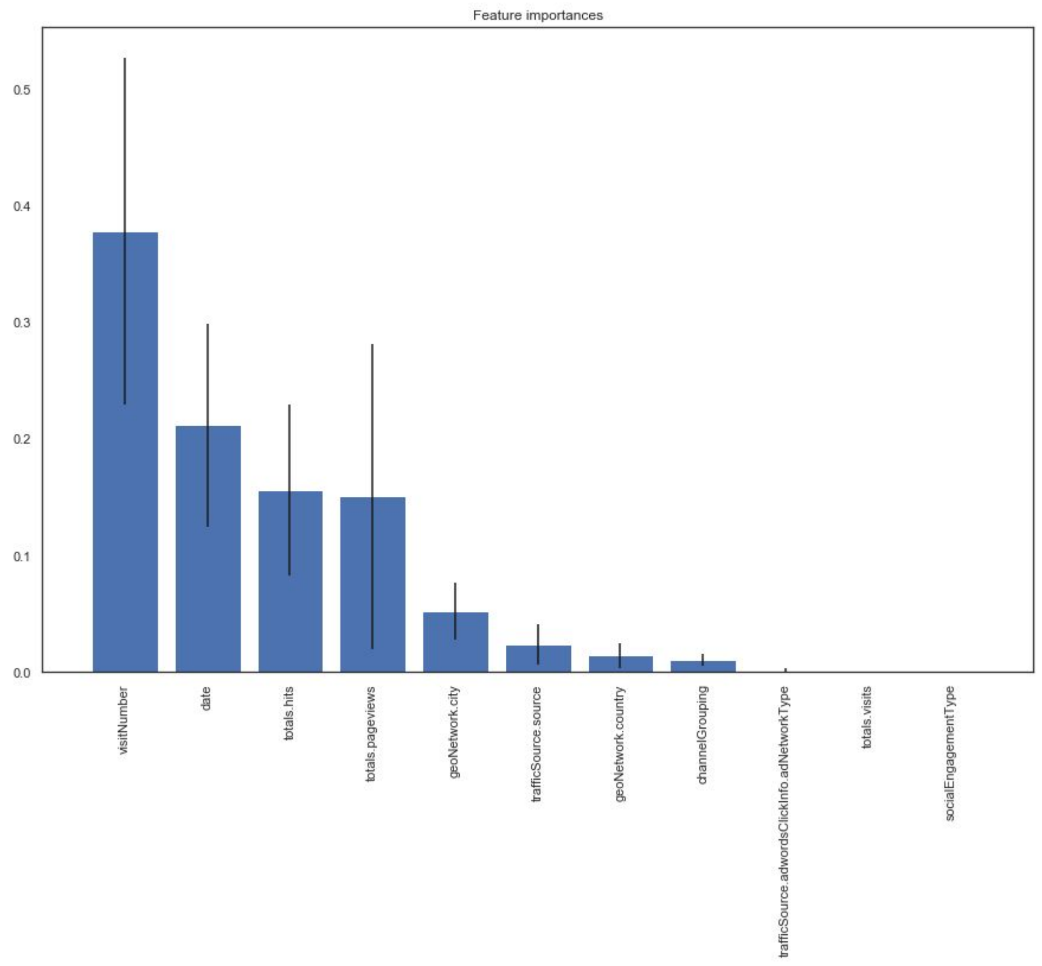
The result above is leading us to believe we will be reducing our amount of attributes on this pipeline from more than 10 to around 4 attributes. The "vanilla" or untouched training data gave us the best sense of how much variance was being captured with more features.

The "elbow" for this line is right on 4 features. The rest of the lines indicate around 5 features as the top limit before it starts to deprecate. Within these attributes is fair to say we have the most variance on this data. Remember that Python numeration starts at 0, in other words after five features we're not getting much information gain.

To reinforce this analysis we will conduct a Random Forest analysis on this data and obtain the feature importance histogram.

RANDOM FOREST

We will be using Random Forest Feature Importance for finding the most important attributes. The reason is because the tree-based strategies used by random forests naturally ranks by how well they improve the purity of the node. This mean decrease in impurity over all trees (called gini impurity).



Nodes with the greatest decrease in impurity happen at the start of the trees, while notes with the least decrease in impurity occur at the end of trees. Thus, by pruning trees below a particular node, we can create a subset of the most important features. On our plot above we identified the following attributes as being the most meaningful:

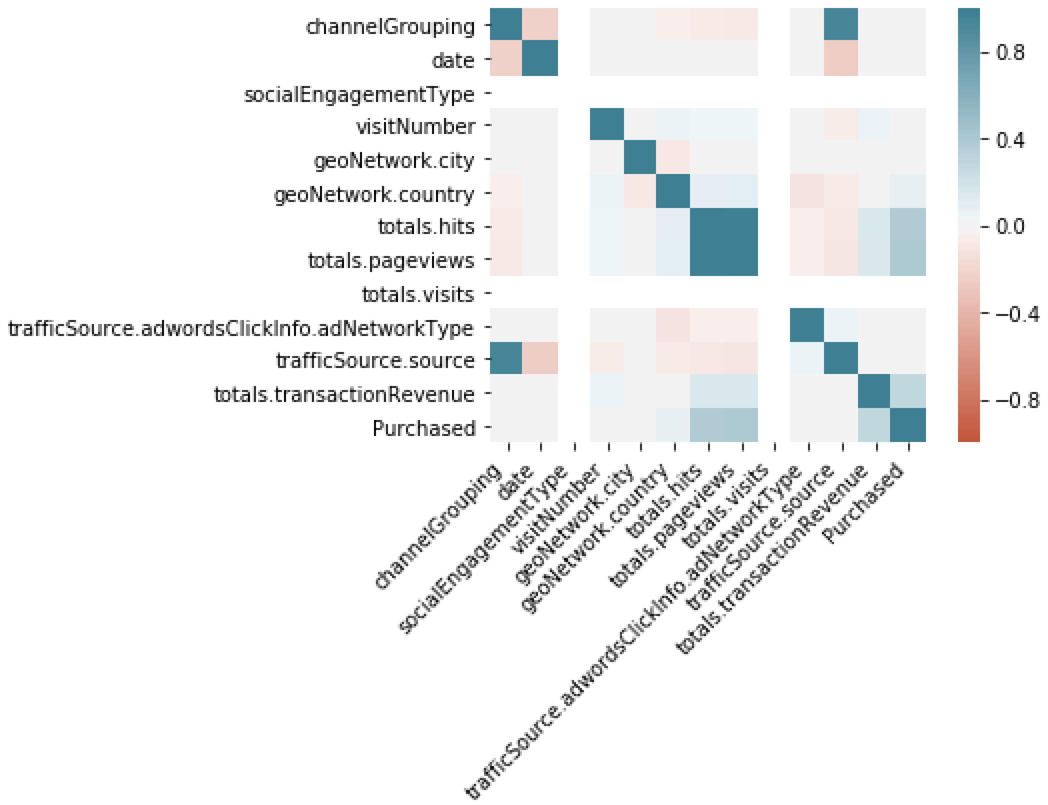
- Visit Number
- Date
- Total Hits
- Page Views
- City
- Source

CORRELATION

This heatmap plot measure evaluates subsets of features on the basis of the following hypothesis: "Good feature subsets contain features highly correlated with the classification, yet uncorrelated to each other. Our plot indicates **page views, hits, country** and **visit number,** as being the most important, a quite consistent result with our previous two tests.

DECISION #3

For now, we feel confident on continuing with these 5 features.



Modeling

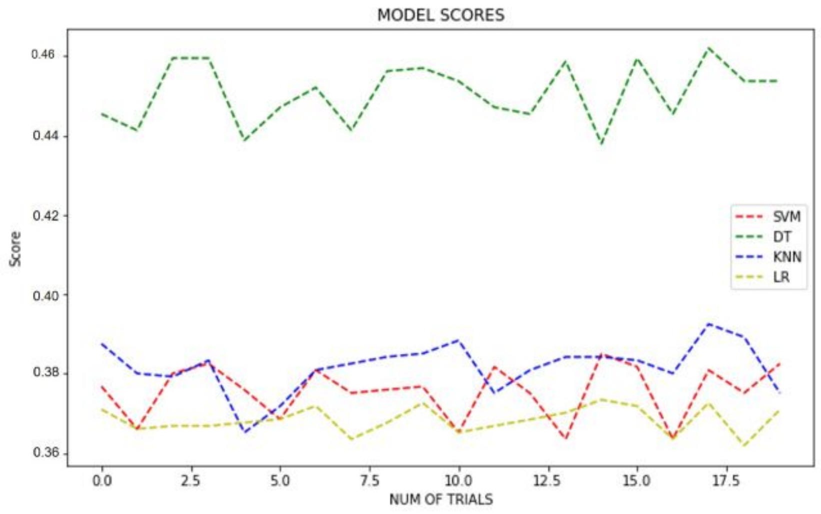
After we have identified our best features, we are ready to move on to the algorithm selection for the regression. Remember, the objective is to obtain a prediction on Revenue for each user. In this phase we need to do some data preparation before we apply the models. The following are the steps we took before moving on:

1. Splitting Train and Test data.
2. We will re-initialize our training and testing data so we can start with a fresh set of data
3. We now subset our data(frame) even further by using the seven features we selected above.

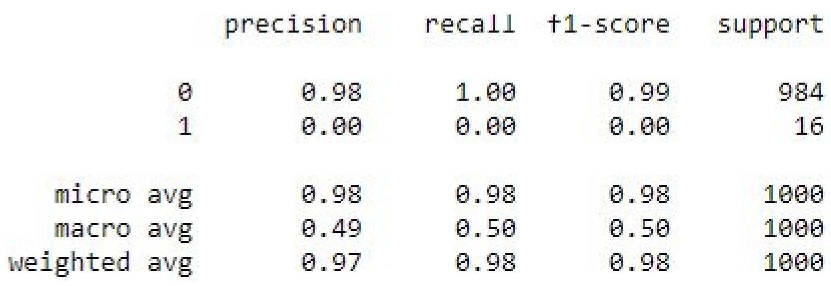
DECISION #4

After evaluating traditional algorithms for Classification, we decided to scrap them all once and for all. We attempted running Support Vector Machines, Decision Trees, K-Nearest Neighbor and Logistic Regressions. The results are presenting an extremely poor performance and the effort that takes to run them makes it quite difficult to pursue any of them further. Classification resulted to be computationally expensive in such a way that we have decided to stick with Gradient Boost algorithms and restricting our analysis on Regression rather than continuing pursuing Classification. For now, technology is limiting us to proceed.

SVM: 28hrs
Decision Trees: 36hrs
K-NN: 21hrs
LR:17hrs



As a final note, we attempted to run a Random Forest Classifier for more than 4 days but the result was futile, the execution halted and the Windows VM Machine became constantly unusable. Restricted by storage and ram memory we tried cloud computing using Data Bricks and execution took around 51 minutes, but once again the result was disappointing as no purchase prediction was obtained from the exercise.



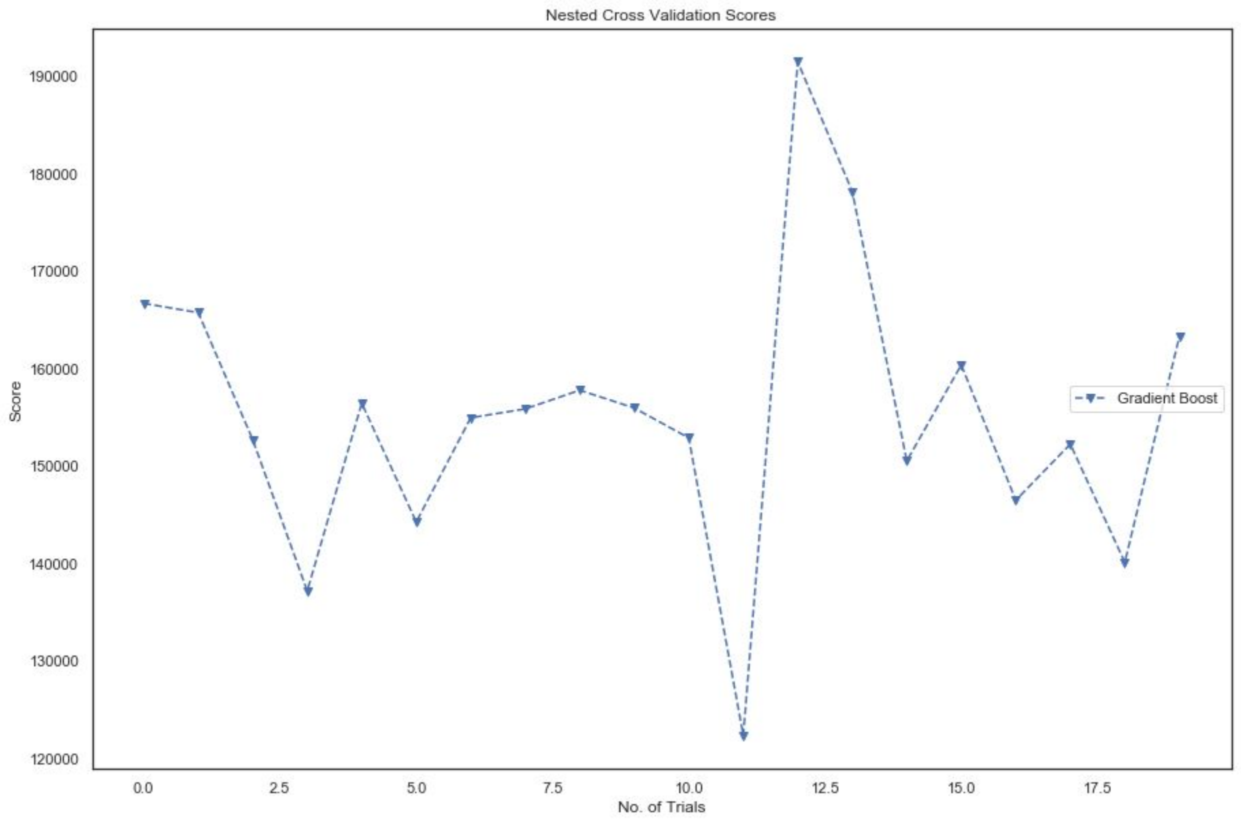
```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       984
           1       0.00      0.00      0.00        16

   micro avg       0.98      0.98      0.98      1000
   macro avg       0.49      0.50      0.50      1000
weighted avg       0.97      0.98      0.98      1000
```

Enter Boosting Algorithms for Regression

We started with one algorithm, GradientBoostedRegressor, which is one of the better "out-of-the-box" algorithms available in sklearn. We immediately encounter a difficult scenario, after 20 trials values oscillated from 120000 to 190000. Not a good score and more important too inconsistent.



The output of the nested cross validation for GradientBoost shows us that we have a long way to go to identify a good model as the best that we were able to do after our first attempt with our first data mining algorithm was a MSE of 122344, with an RMSE of ~349. This first attempt displays a poor score for one of the best algorithms out there.

DECISION #5

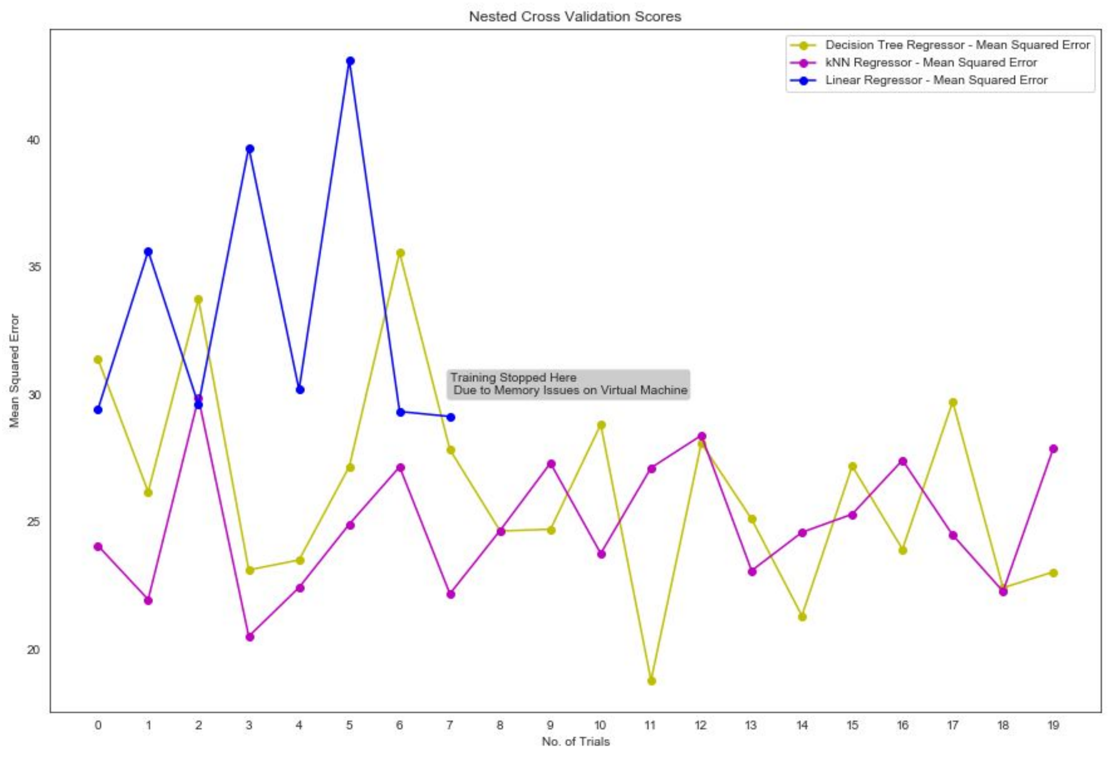
After reviewing the results of the gradient boosted algorithm, we came to the realization that this data mining happened without normalization of the training data.

Normalization is an important part of the model selection process; it is the process of scaling individual samples to have unit norm. This process can be useful if you plan to use a quadratic form such as the dot-product or any other kernel to quantify the similarity of any pair of samples.

This could possibly be one of the reasons that the model did not perform well. Before we try any other algorithms, we will normalize the data first.

Decision Tree, KNN and Linear Regressions

Having limited technical resources we decided to test these three algorithms initially. This would give us a sense of expected performance by using a technique called as **Nested Cross Validation**, on which a range of hyperparameters are tested and compared between the different runs, so that finally the **best settings** can be **picked**. After the initial loop, another technique called **GridSearch** fires up and we move on obtaining the scores for each algorithm. The process described produces the comparison below:

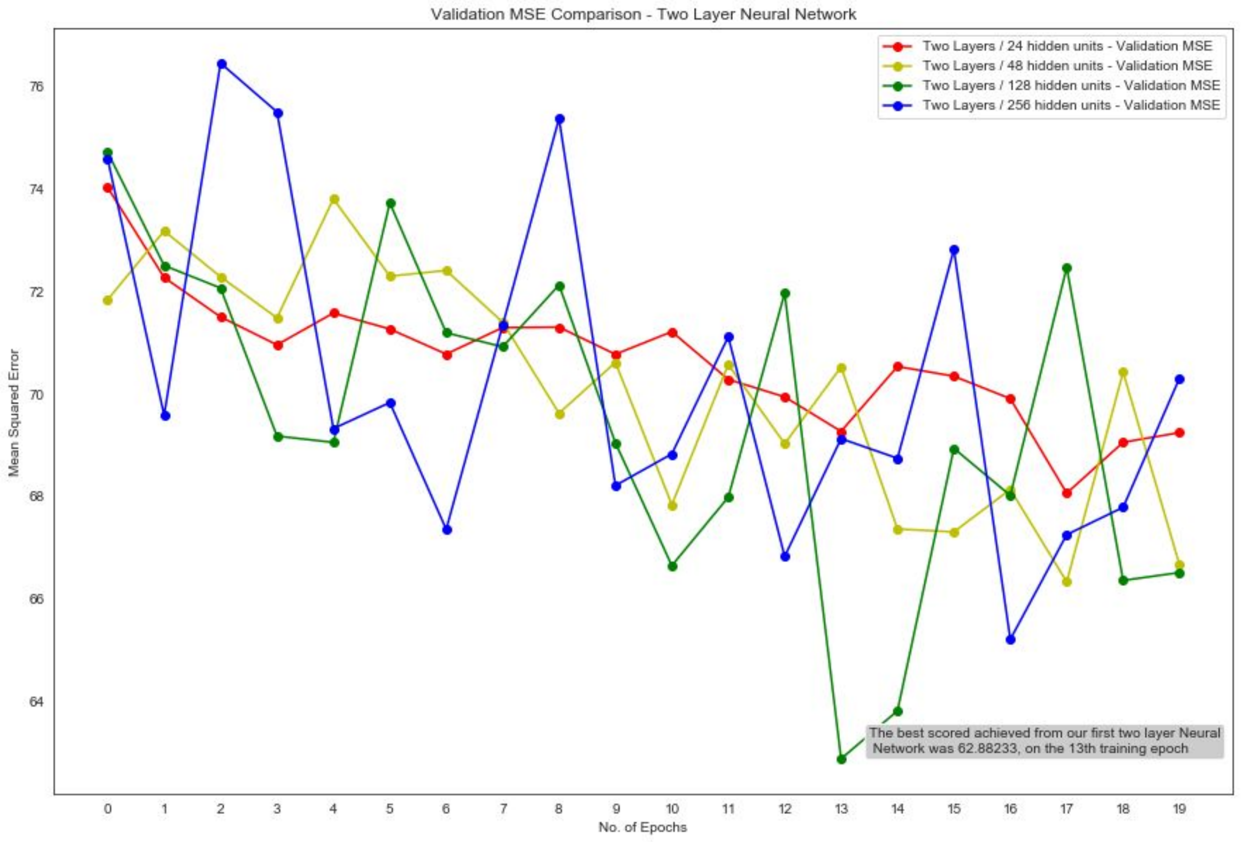


The execution above shows an interrupted Linear Regression due to Memory issues. However the 8 runs it completed are sufficient to show us the trend to outperform the kNN and Decision Tree. We used the Mean Squared Error as the objective function & error measurement. An objective function is also known as a loss function (or optimization score function) and is one of the two parameters required to compile a model.

The error measurement is also known as a metric function; it is used to assess how well the models are performing. Metric functions are supplied in the metrics parameter when a model is compiled. We noticed that each model performed about the same, and the best MSE score was ~18 (which is an RMSE of 4.24) by our DecisionTreeRegressor.

At this point in the regression analysis, it was clear that this was going to be more of an exploratory analytics type effort than earlier conceived. We had attempted some of "out-of-the-box" algorithms and, up until this point, had been unsuccessful in building a robust model. While the results present a low Mean Square Error, is now when our strategy of running a different set of algorithms proceeds. Now we move on into Deep Learning. We will run several neural networks with two layers and visualize the performance.

Deep Neural Networks



The steps taken to run the algorithms above were:

1. Build a two-layer neural network (using a function).
2. Initialize a list of hidden units to test with the two layer model.
3. Run some training epochs to see if the neural network could perform better than our earlier attempts.

The result shows a fluctuation from 64 to 76 as the Mean Squared Error with clear dominance of the Neural Network configured with 256 hidden units. Our Neural Network model didn't perform as well as we had opened, and the best MSE we were able to obtain was 62 (which is an RMSE of 7.87). This did not outperform our earlier attempts.

Is time to move to our third phase, Boosting Regressors. Enter Light GBM, an algorithm developed by Microsoft in 2017 and is a Gradient Boost algorithm that uses tree based algorithms. LightGBM is a gradient boosting framework using tree based learning algorithms. It has the following "features":

1. Faster training speed and higher efficiency
2. Lower memory usage
3. Better accuracy
4. Parallel and GPU learning
5. Handling large-scale data

As mentioned previously in the analysis, we realized that this had become an exploratory type analysis. Once we had discovered this new algorithm, we also reviewed the specifics of the competition one more time. This is where we made the realization that our "target" up until this point had been wrong. The natural log of revenue was what were looking for, and not the revenue target column we had created on our own previously.

1. Build an LGB Model
2. Revisit our training data and create a new instance of training data
3. Create a new target variable using the appropriate function - using a np.log1p function
4. Use a pandas function factorize to transform our categorical features into integers
5. Convert all numeric fields to integer
6. Run the model

DECISION #6

Light GBM

The algorithm run taking up all the resources on our VM. After a fast execution of 1 min, we obtain very promising results.

Results:

```
Training until validation scores don't improve for 200 rounds
[500]     training's rmse: 1.65597     valid_1's rmse: 1.66146
[1000]    training's rmse: 1.61644     valid_1's rmse: 1.63829
[1500]    training's rmse: 1.59667     valid_1's rmse: 1.63148
[2000]    training's rmse: 1.58229     valid_1's rmse: 1.62939
[2500]    training's rmse: 1.57079     valid_1's rmse: 1.62905
[3000]    training's rmse: 1.56038     valid_1's rmse: 1.62844
Early stopping, best iteration is:
[3276]    training's rmse: 1.5551      valid_1's rmse: 1.62817
LGBM: RMSE val: 1.62817 - RMSE train: 1.5551
Wall time: 1min 21s

LightGBM feature importance:
              feature  split       gain
3     totals.pageviews  22937  39.359488
2          totals.hits  24231  20.268550
5    geoNetwork.country   8059  11.490184
0          visitNumber  17199   9.355165
6  trafficSource.source  11723   7.663288
1                 date  25161   6.763533
4      geoNetwork.city  18454   5.099792
```

For this execution we went through the same search effort of the best parameters. We're ready to try our newly created model! We run it twice, to also capture how quickly it is able to perform training and validation concurrently. One of the benefits of LGBM is speed. Our best RMSE by almost 3 - the LGB algorithm is working much better than we had expected. To find out more about how the algorithm ranked the feature importance, for the seven features we selected we then extract that information using the methods inside the libraries.

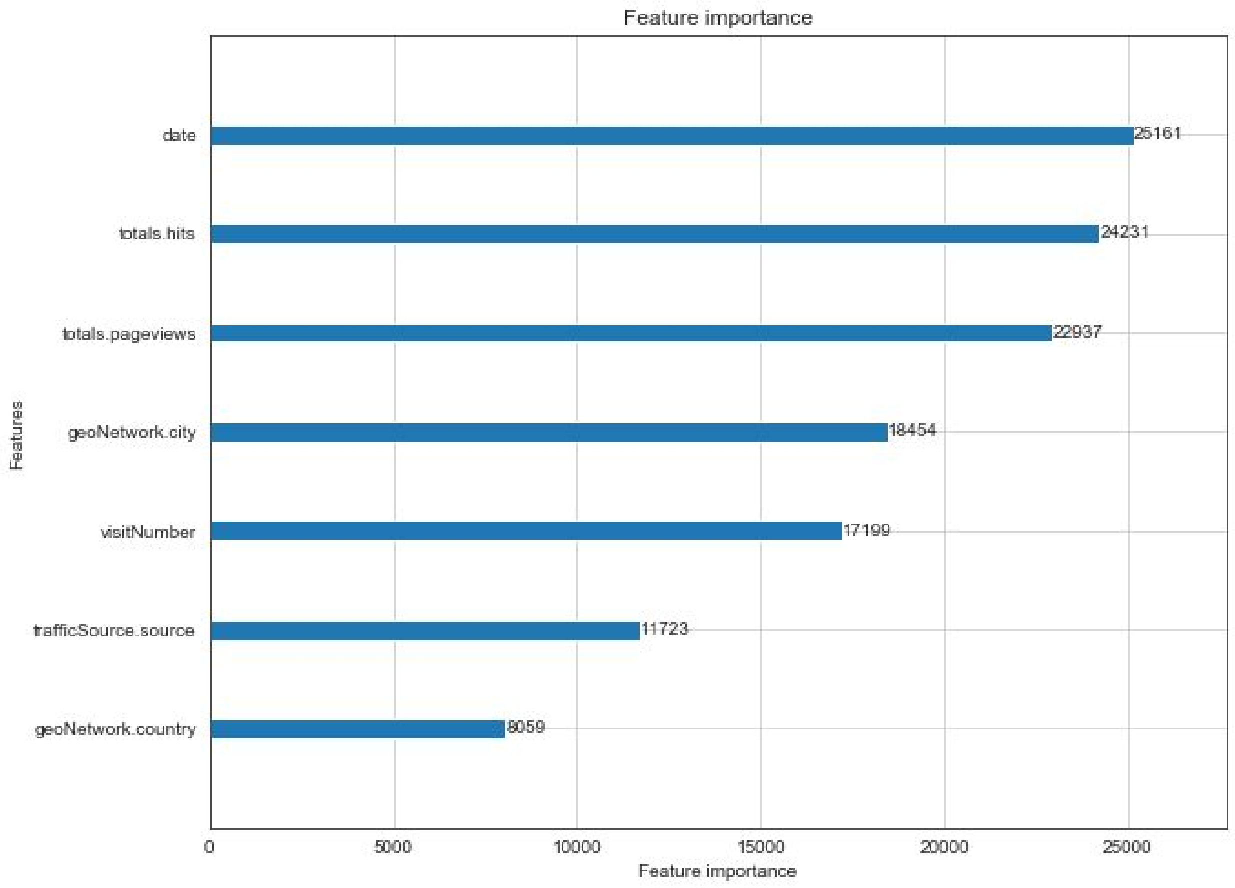
LightGBM feature importance:

feature split   gain

- 3   totals.pageviews 22937 39.359488
- 2   totals.hits 24231 20.268550
- 5   geoNetwork.country   8059 11.490184
- 0   visitNumber 17199   9.355165
- 6 trafficSource.source 11723   7.663288
- 1   date 25161   6.763533
- 4   geoNetwork.city 18454   5.099792

This is by far the best performance that we have seen out of the various models that we tried to use. After such an exhaustive search, we're finally confident enough of a model to use it on our testing data. The graphical output of the feature importance is below:

Feature Importance



This is the end of our technical research. We have found an algorithm that runs fast, that provides a good score and that works with our limited hardware resources. We have produced an algorithm that will be able to predict the revenue obtained from a customer by identifying the characteristics and fitting them to a pattern with a high percentage of confidence. Attached to this document, we have made our appendix available with the subjects below.

Appendix log

1. Exploratory and Classifier Code
2. Classifier Code part 2 performed in Databricks
3. Exploratory and Regression Code
4. LGBM Revenue Regression
5. Graphical Decision Trees:

Conclusions

We learned how technical limitations can spark creative solutions to deal with large datasets. Partitioning and Sampling can give us, with less certainty, a glimpse into the shape and form of the patterns of the whole dataset. Random Sampling plays an important role when limitations are obvious. We still think that achieving such good RSME using LightGBM is good enough regressor due to the complexity of the data.

From the data preparation stage, we found out how important aggregating it to the right level is. Is a valid technique to shrink down the number of observations thus reducing complexity. Is quite important to consider separate pipelines for all the modeling techniques we want to use, including using different machines,

LightGBM and XGBoost resulted to be the best techniques we studied before performing this exercise, however from analyzing each of the characteristics, the time we had left and the hardware limitations we experienced, we picked the right one to get the best result. On the attempt, we learned why histogram based split is much more faster than traditional sklearn GBM methods, as it relies on pre-sort dataset , adding time complexity to the execution, while LightGBM reduces cost of calculating the gain for each split.

Further steps include testing CatBoostas is one of the newest algorithms and it promises similar if not better performance than LightGBM. Nevertheless, after several hours of training and testing data, we feel that the resulted model can help prepare a marketing teams to predict revenue and find user behavior that leads to the final sale.

Cited Works

[1] (2019, February 21). Google Analytics Customer Revenue Prediction. *Kaggle*. Retrieved from https://www.kaggle.com/c/ga-customer-revenue-prediction

[2] Peller, J. (2018, September 25). Quick start: read csv and flatten json fields. *Kaggle*. Retrieved from https://www.kaggle.com/julian3833/1-quick-start-read-csv-and-flatten-json-fields

[3] (2019, November 28). LightGBM, Light Gradient Boosting Machine. *GitHub*. Retreived from https://github.com/microsoft/LightGBM

[4] FabienDaniel. (2018, September 25). LGBM (RF) starter. *Kaggle*. Retrieved from https://www.kaggle.com/fabiendaniel/lgbm-starter

[5] Swalin, Alvira. (2018, March 13). CatBoost vs. Light GBM vs. XGBoost. *Towards Data Science*. Retrieved from https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

[6] (2019, November 28). RobustScaler. *sklearn*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html#sklearn.preprocessing.RobustScaler