Capstone Project

Andreas Erga
December 15, 2018

Definition

Project Overview

Today it's become a rule of thumb that a typical marketing department uses 30-50% of its' marketing budget on activities that bear little to no results.
By analyzing and accurately segment customers according to propensity to spend a business a marketing department can optimize and allocated its marketing budget according to activities that generate income while at the same time free users of unwanted ads and promotions.

There is currently several papers published concerning customer values and maximizing customer revenue from traffic source (1, 2). However in my research I had problems finding papers that restricts its dataset to only onlinedata the entity or company owns/ have generated. As acquiring data from 2nd or 3rd parties (figuring where, how and which data to trust can lead to project in itself) usually s(et the bar quite high for initializing a analysis/ predictive task.

Problem Statement

How far along in predicting revenue for a given user can a company expect to get using only its own data.

The 80/20 rule has proven true for many businesses–only a small percentage of customers produce most of the revenue. As such, marketing teams are challenged to make appropriate investments in promotional strategies.

The challenge is to analyze a Google Merchandise Store (also known as GStore, where Google swag is sold) customer dataset to predict revenue per customer. If successful we can provide data that is actionable results and a better use of marketing budgets.

This task is from the Google Analytics Customer Revenue Prediction competition hosted by Kaggle (3).

The task is twofold:

- ➔ How does XGboost compare to a simple BaggingRegressor
- ➔ Given current hardware limitations on how does the end model rank against the public leaderboard on the competition page.

Metrics

For each fullVisitorId in the test set, the task is to predict the natural log of their total revenue. December 1st, 2018 to January 31st, 2019.

FIG 1. Formal definition of evaluation metric (3)

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

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

Analysis

Data Exploration

The dataset consists of two files, a training and test set. Each row in the dataset is one visit to the store.

- fullVisitorId- A unique identifier for each user of the Google Merchandise Store.
- 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.
- socialEngagementType - Engagement type, either "Socially Engaged" or "Not Socially Engaged".
- totals - This section contains aggregate values across the session.
- trafficSource - This section contains information about the Traffic Source from which the session originated.
- 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, you should use a combination of fullVisitorId and visitId.
- 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).
- 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 - This set of columns mostly includes high-level aggregate data

Fig 2. Data from a single visitorid after initial processing:

```
channelGrouping                                                    Referral
date                                                               20171016
fullVisitorId                                          8934116514970143966
visitId                                                          1508176307
visitNumber                                                             6
visitStartTime                                                   1508176307
device.browser                                                       Chrome
device.deviceCategory                                               desktop
device.isMobile                                                           0
device.operatingSystem                                             Chrome OS
geoNetwork.city                                                    Cupertino
geoNetwork.continent                                                Americas
geoNetwork.country                                             United States
geoNetwork.metro                          San Francisco-Oakland-San Jose CA
geoNetwork.networkDomain                                           (not set)
geoNetwork.region                                                 California
geoNetwork.subContinent                                      Northern America
totals.bounces                                                          NaN
totals.hits                                                               2
totals.newVisits                                                        NaN
totals.pageviews                                                          2
totals.transactionRevenue                                               NaN
trafficSource.adContent                                                 NaN
trafficSource.campaign                                           (not set)
trafficSource.isTrueDirect                                              NaN
trafficSource.keyword                                                   NaN
trafficSource.medium                                               referral
trafficSource.referralPath    /a/google.com/transportation/mtv-services/bike...
trafficSource.source                                           sites.google.com
customDimensions                    [{'index': '4', 'value': 'North America'}]
Name: 1, dtype: object
```

Fields named 'device', 'geoNetwork', 'totals', 'trafficSource' contained aggregate json data and where converted to columns.

From figure 2 there it is apparent that a number of features contain nan values. These have to be correctly encoded depending on decisions made for handling categorical data. In this instance - where several features were dropped because of restrictions discussed further along in the report - most nan values where converted to zero values and some were dropped altogether.

Columns with constant values across each visitorId/row provide no information and can safely be dropped.

Exploratory Visualization

Fig 2. Initial overview of training and test data.

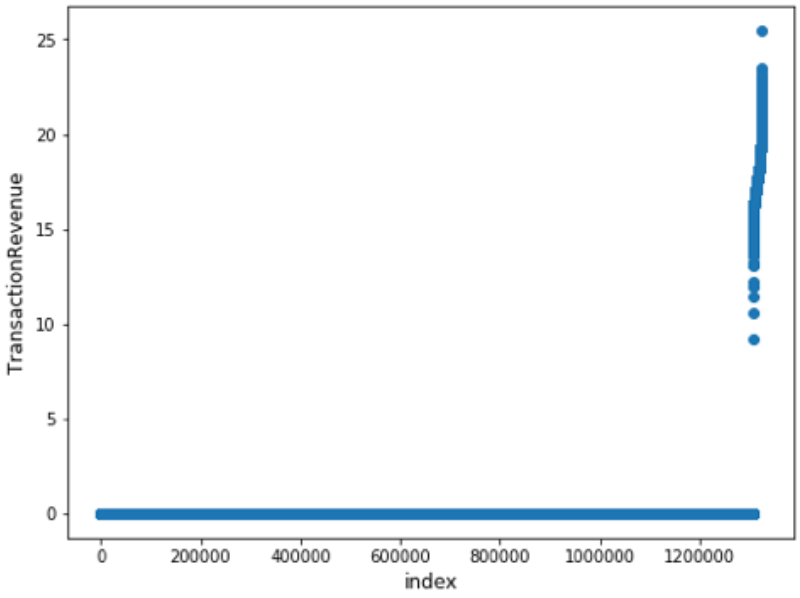
```
Number of rows in train set:  1708337
Number of visitors in train set:  1323730
Number of purchases made in train set:  18514
Number of customers in train set:  16141
Total amount spent:  2316344980000.0
Mean amount spent: 1749862.1168969502
Standard deviation: 115399633.82418895
PCT customers:  1.219357421830736 %

Number of rows in test set:  401589
Number of visitors in test set:  296530
Number of purchases made in test set:  4594
Total amount spent:  558347940000.0
Mean amount spent: 1882939.1292617947
Standard deviation: 75361967.48034734
PCT customers:  1.401544531750582 %

Number of common visitors in train and test set :  2759
```

The training data contains 1,700 000 mill rows and 1 300 000+ unique visitor ids. Only about 1,2% of the user decided to purchase from the google store within the dataset´s timeframe. While the test set contains 401589 rows, 296530 visitors and 4594 purchases were made which is about 1,4%.

Fig 3. Grouped and sorted unique purchasers:



The scatter diagram gives us a better visual understanding of the problem. It's becoming apparent that identifying and estimating the log revenue of each visitorId is like looking for a needle in the haystack.

Fig 5. Number of purchasers by subcontinent.

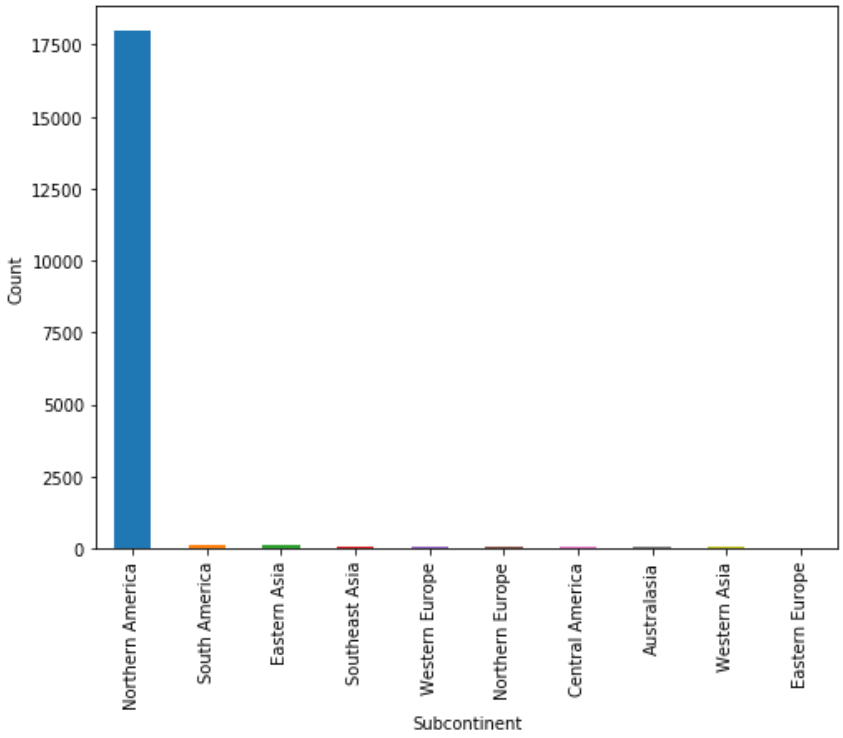
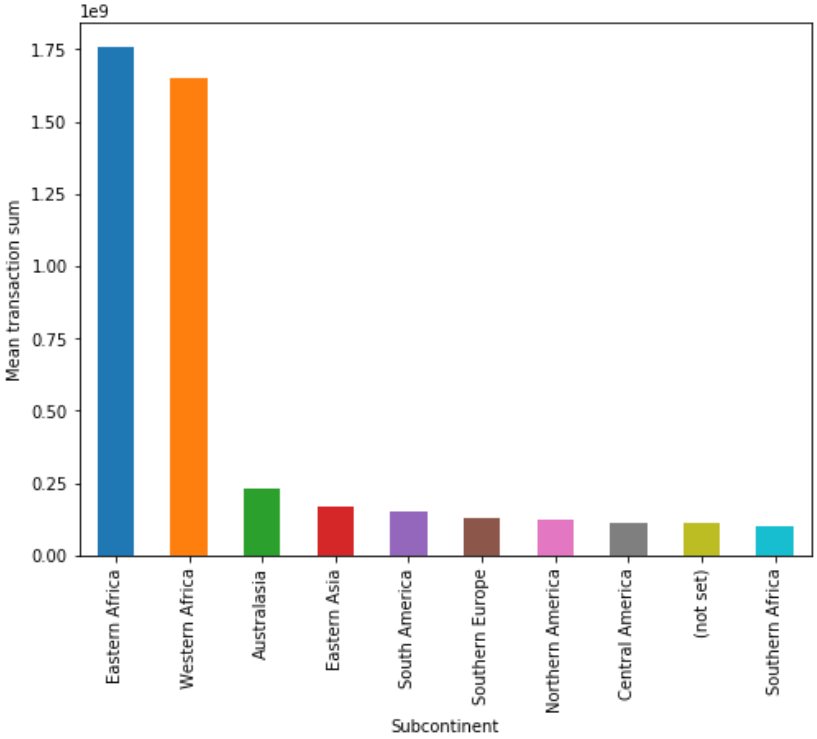


Fig 6. Mean purchase amount by subcontinent.



From figure 5 & 6 it's obvious that visitors from north america with 17600+ purchases is responsible for the lion's share of purchases made in the store. But by looking at the mean transaction revenue north america does not even break the top ten. On the other hand, when one instance is responsible for 95+ percentage we can concluded that the majority of the other instances do not have enough transactions and therefore its' mean transaction sum is likely to be highly volatile.

Fig 6. Count of purchases made by date.

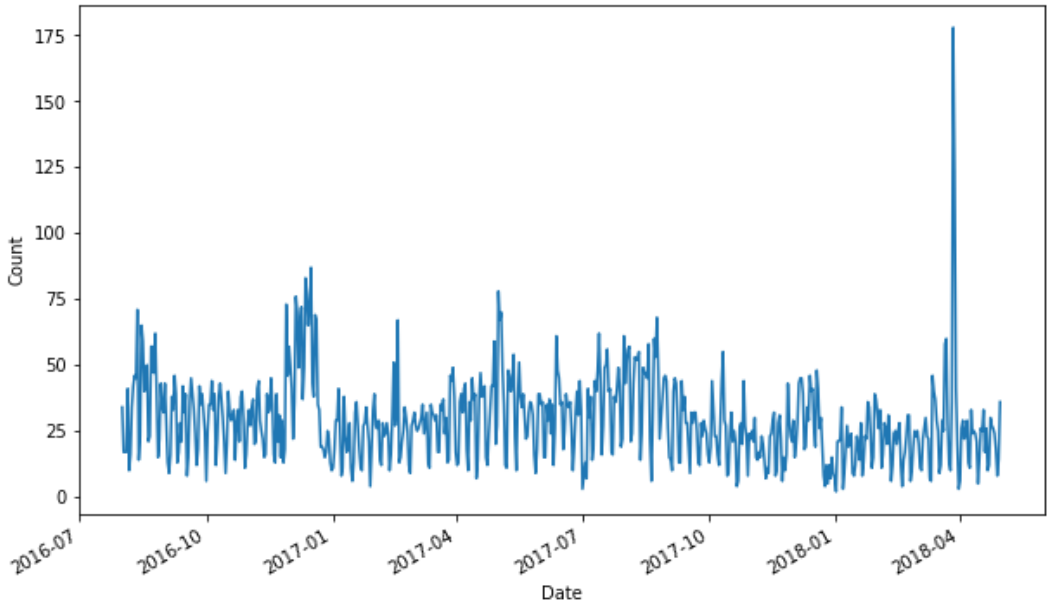


Fig 7. Revenue by date.

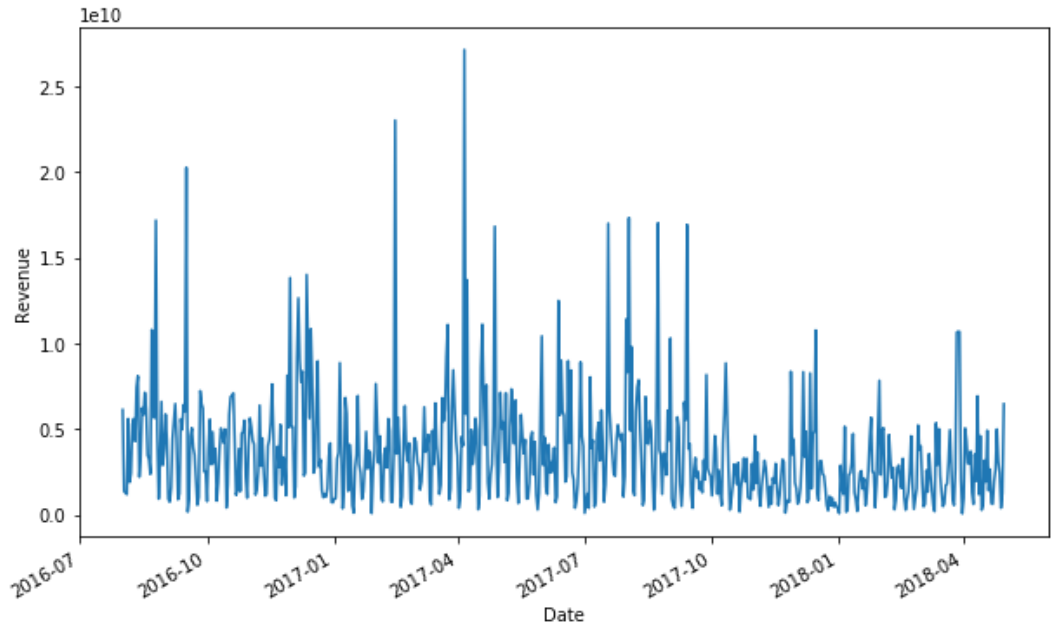
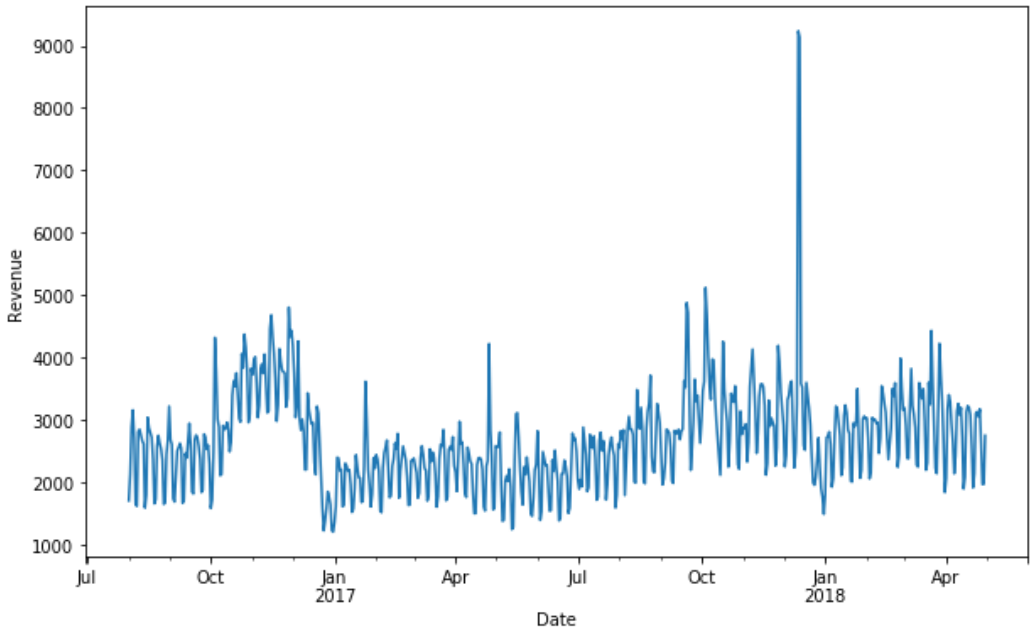
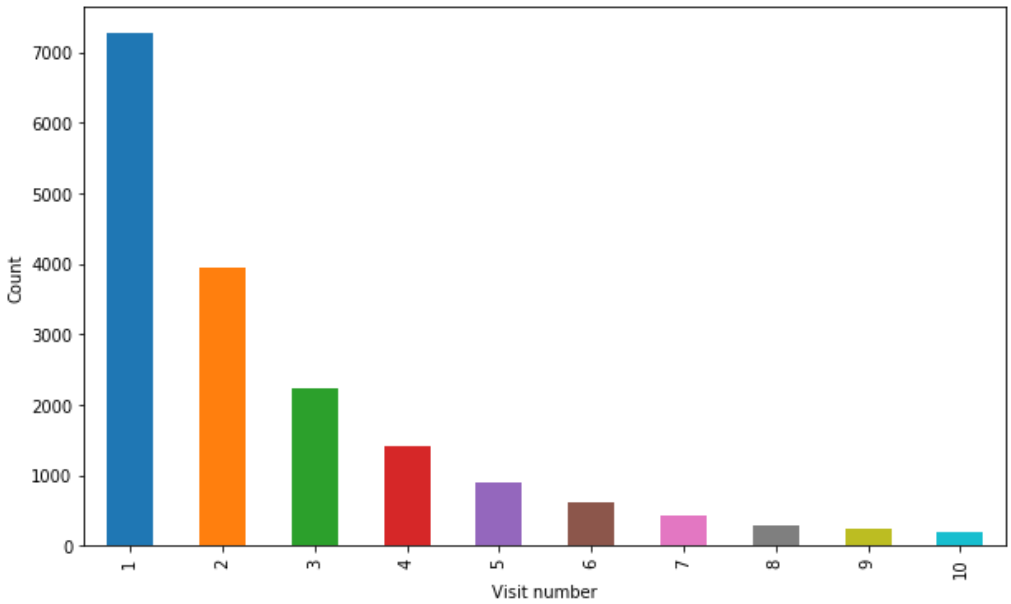


Fig 8. Visits by date.



While the total number of transactions spike in april (fig 6) the total revenue graph (fig 7) does not show a similar behavior. It's unproven but we can speculate that this might be gifts bought by Google itself and distributed among its employees, and therefore total revenue is not affected by it. There seems to be some correlation between number of sales and total revenue during the holiday season at the end of 2017 and somewhat during the spring of 2017.

Fig 9. Transactions by number of visits.



The majority of purchases are made during a users first visit giving the model less time to find a pattern and therefore predict a value.

Algorithms and Techniques

The model used in this project is XGBoost. It is currently one of the most widely used models in competitions. As the name implies it does so by boosting multiple weaker (simpler) models/learners into a strong learner. The goal is to teach the model to predict y-hat by minimizing the mean square error. It does so by building the trees (weak learners) sequentially, where each tree aims to reduce the error of the previous tree.

In contrast to bagging techniques like Random Forest, in which trees are grown to their maximum extent, boosting makes use of trees with fewer splits. **(4)**

"Boosting consists of three steps:

- *An initial model F0 is defined to predict the target variable y. This model will be associated with a residual (y – F0)*
- *A new model h1 is fit to the residuals from the previous step*
- *Now, F0 and h1 are combined to give F1, the boosted version of F0. The mean squared error from F1 will be lower than that from F0:*

To improve the performance of F1, we could model after the residuals of F1 and create a new model F2:

This can be done for 'm' iterations, until residuals have been minimized as much as possible." **(4)**

There are a number of parameters to tune but the following is in my view the most important to optimize and therefore used:

- ➔ Eval metric:
  - ◆ How the model shall be evaluated.
- ➔ Objective:

- ◆ The task or objective the model shall learn.

➔ Max depth :
- ◆ The number of nodes allowed from the root to the farthest leaf.

➔ Min child weight:
- ◆ The minimum weight needed to to create a new node in the tree.

➔ Subsample
- ◆ The percentage of observations to include at each step.

➔ Colsample bytree:
- ◆ The percentage of features to include at each step.

➔ ETA
- ◆ Learning rate.

During training the data set is loaded into the model and batches are selected according to the parameters are in the model. The data is then processed according to the technique described above.

Benchmark

Initially a simple linear regressions model was selected as a benchmark model. After a couple of runs it became apparent that its' predictions resulted in a log mean square error that was unsuitable. Therefore a Bagging Regressor with no optimization steps was selected as a benchmark.
Benchmark results:

➔ RMSE: 1.7452
➔ Prediction time (Wall time) : 2.08 s
➔ Ranking on public leaderboard on Kaggle: 817

A bagging model have a few different characteristics compared to a boosting model. In short: Where boosting builds its weak learners sequentially in an effort to minimize previous learners error. Bagging builds its trees in parallel and summarizes of its results. For this reason a bagging regressor was selected as a benchmark.

Methodology

Data preprocessing

Data is easily obtained from kaggle.com, since this is part of a competition hosted by kaggle.com. The data already came split into a training and test set.

- ➔ Given the size of the data and restrictions my current hardware setup made on my memory and processing power the data was loaded in on batches and finally converted in to a dataframe.
- ➔ JSON data parsed and categorised.
- ➔ Initial data types checked and if not suitable changed to reduce pressure on memory.
- ➔ Columns with constant values removed.
- ➔ For easier processing the training and test set is combined.
- ➔ Target variable checked for nan or no values and converted to Zero values.
- ➔ Date fields types changed to date data types.
- ➔ Exploratory analysis.
- ➔ Feature selection.
    - ◆ Because of the aforementioned restrictions on current hardware and this projects aim is not to win the competition but compare two models, feature selection is quite restrictive and valuable information is probably lost during this step.
- ➔ Split and categorize text data into columns. One hot encoding
- ➔ Replace the original transaction amount for each observation with the natural log of said observation.
- ➔ Split the total data back to training and test data.
    - ◆ Thankfully google provided the data split according to dates so it was no problem splitting the data back again.
- ➔ Load data into models

Note about feature selections:
Before diving into feature selections I must stress that these restrictive hypothesis regarding features come partially as a result of the current setup and aim for project.

Not to win the competition, but to compare XGBoost with a standard bagging regressor and then see how it performs in the competition under these restrictions.

This particular dataset contains lots of categorical data , operating systems, location etc. For this information to be useful for the model such data have to be transformed into readable data for the model. A common technique is called one hot encoding. For each unique entry into the original columns/feature another is added and named from the entry and flagged with an 1 all any rows that do not have the same entry gets a zero. Even though it adds a lot of data, for categorical data it is a preferred technique over Label encoding commonly used for ordinal data (6).

During the feature selection a number of subjective decisions where made. The data came with locations as general as continent down to city and regions. In my opinion the culture and income/purchasing power do not vary to much from city to city or even country to country as it does between sub continent. For that reason all geolocation data was removed except for subcontinent.

In most parts of the world there exists close to a ISP monopoly or duopoly . As a user there aren't many options when selecting an internet provider. As a result from this, a correlation between network info and purchasing behavior is not probable. This info should be picked up by device Category, is mobile, operating system
Info and to a lesser extent subcontinent. Therefore geo network domain and most network info is dropped.

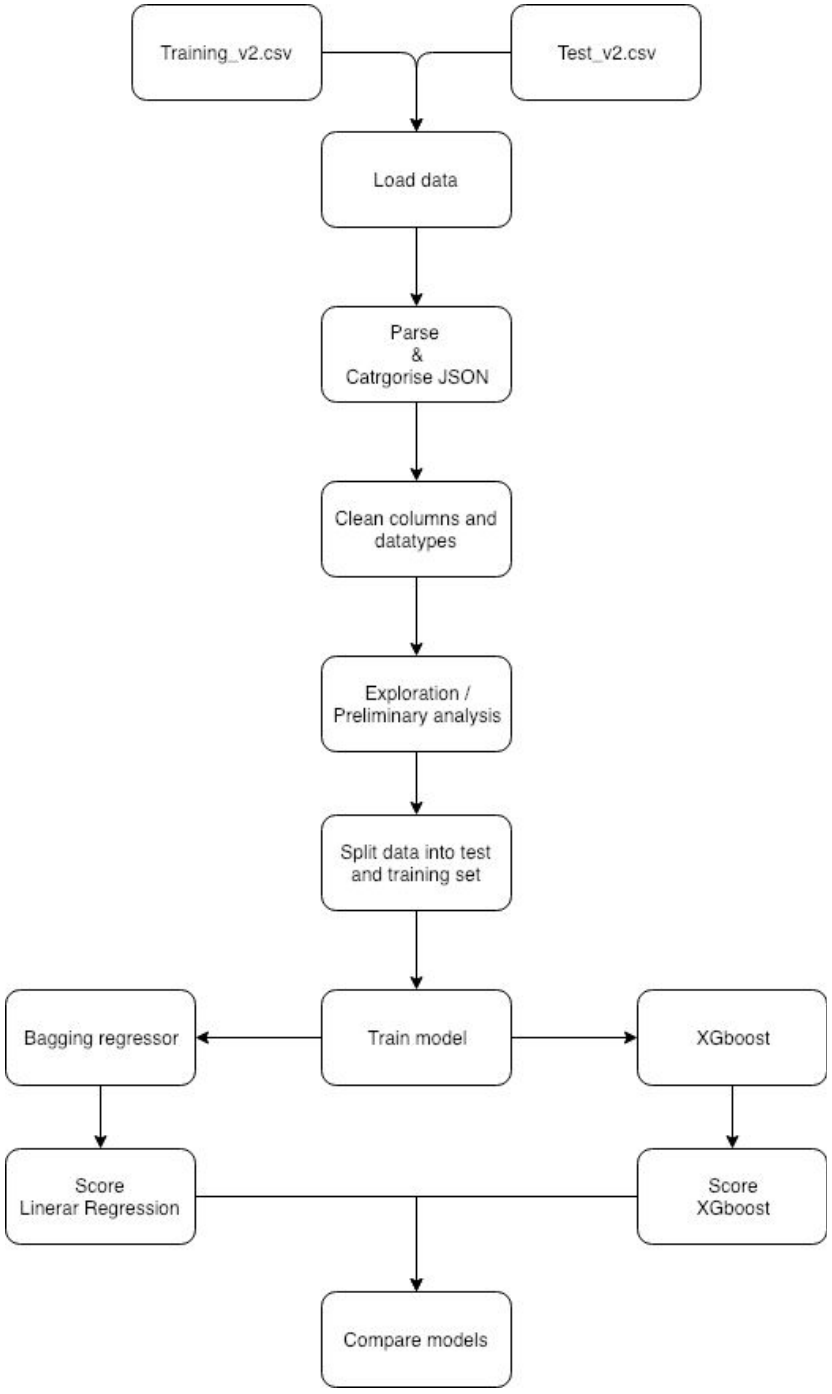
The same logic is true for elimination of keyword, ad content etc. Here Channel grouping, flag if traffic is direct and medium provide the necessary info.

Implementation

- ➔ After preprocessing and selection outlined above. Subsample training set.
- ➔ Usually, split the data into train and test set, but in this instance Google have done it for us.
- ➔ Subsample training set
- ➔ Initial dummy benchmark done by using a standard implementation of bagging regressor.
- ➔ Cross validation
    - ◆ Number of folds to use: 5
        - Define how many times the set shall be shuffled and trained on.
    - ◆ Seed : 42
        - a random number for repeatability.
    - ◆ Early stopping rounds: 5
        - if adding another boost round /tree show no signs of improving the model after 5 rounds. stop
- ➔ Define search params for XGboost:
    - ◆ Evaluation metric: Root mean square error (rmse).
    - ◆ Objective: linear regression.
    - ◆ Max_depth: 6-8.
    - ◆ Min child weight: 6-8.
    - ◆ Subsample : (0.7 - 1)
    - ◆ Colsample bytree: (0.7 - 1)
    - ◆ Learning rate : (0.1, 0.05)
- ➔ Iteratively search through relevant parameters pairwise using the subsampled training set.
- ➔ Train with final parameters on full training set.
- ➔ Predict on test data.
- ➔ Report findings.

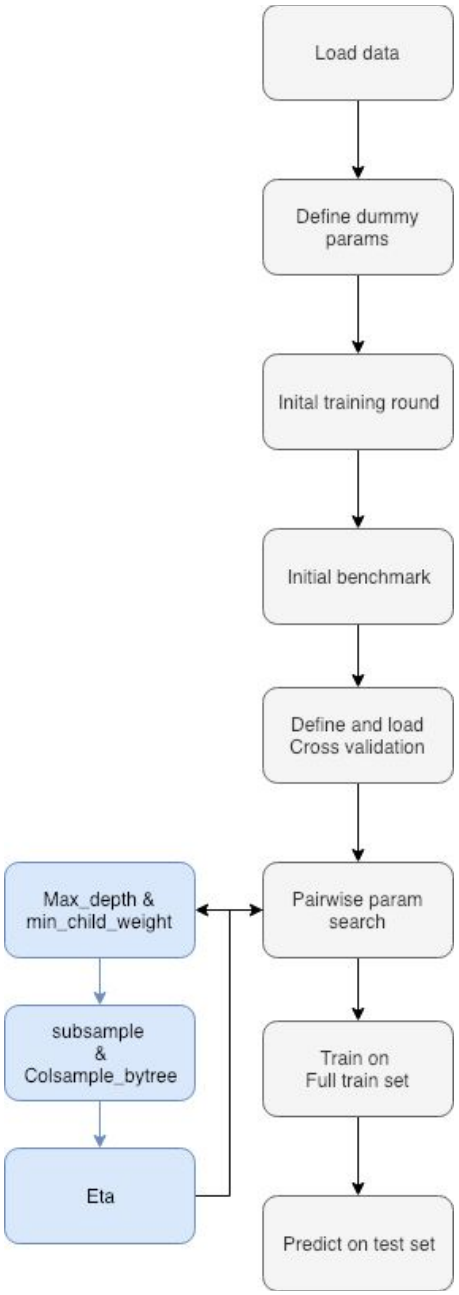
Thankfully XGBoost is well documented (5) and there exists an abundance of information online (4-9). But restrictions on memory and hardware pressed decisions concerning feature selections. Unfortunately - to keep training time within reason - even restrictive feature selection proved not to be enough when loading the training set into the model. For this reason parameter search was performed on a subsampled training set. This can be done in several ways but in this instance a random subsample of 2 times the test set was done.

FIG 9: Overview of the process end to end



Refinement

Fig 10. The initial model with default parameters achieved a rmse per visitorid of 1.63221 on the training set.



Given the amount of data and current restrictions, the refinement process consist of finding the parameters that should obtain best results while simultaneously keeping the time for training within reason.
Cross validation , a technique commonly used to refining a model was employed to refine our model. A common problem with supervised machine learning is training and testing on the same data set. This leads to overfitting. One approach to navigate this issue is to split the data into a training and a test set. This mediate the problem of overfitting somewhat. But unfortunately it is still all too common to tune the parameters to the fit the test set and then overfitting occurs. The most robust way to overcome the problem of overfitting is to employ cross validation. The technique folds the selected data n amounts of time while holding out x% for test for each fold. A simple way of thinking about it is to visualize a deck of cards shuffled n times and x cards held out for test for each shuffle. XGBoost comes with Cross validation included.

The first step in refining the model is to define the number of boost rounds or trees to be built and early stopping. This to give the model enough trees to find a suitable end model while stopping if no improvements are seen after x amount of trees after last best model. For parameter search 5 rounds was selected as an early stopping criteria. While the final training step utilized 10.

Parameter search:

Search 1:

Max_depth:
Building av deeper tree should in theory give the model better learning, but a tree with too many nodes a prone to overfitting and don't generalize well on unseen data.
Search range: (4 - 10, with 2 steps).

Min_child weight:
If each sample has a weight of 1 as in this instance where linear regression is used as objective, defines how many samples is required to make a new node or partition. The fewer samples are needed to create new nodes the more complex the tree can be built but once again more prone to overfitting.
Search range :(6 - 9)

Result: Max_depth: 6, Min_child_weight: 6, RMSE: 1.5689

Search 2:

Subsample:
How many samples or rows to used at each step. Smaller number leads to a more conservative model while higher makes it more prone to overfitting
Search range:(0.7 - 1)

Colsample_bytree :
Fraction of features, how many features to use.
Search range:(0.7 - 1)

Result:, Subsample: 1, Colsample_bytree: .9, RMSE: 1.5688

Search 3:

Eta:
Also known as learning rate. As xgboost consists of weak learners that aim to reduce to previous learners errors by reducing the mean square error. Eta defines how big the nexts trees step will be. A higher eta will reduce training time but also exposing the model to overfitting. Another way of thinking of its is how certain should the model be when correcting for previous errors.

Search range (0.1, 0.05)

Result: Max_depth: .03, RMSE: 1.5685

Results

Model Evaluation and Validation

Validation of the model was two stepped. To verify that that the subsamp of the training set for parameter selection was doable, a final training round with the final parameters was performed on the full training set. This resulted in a rmse of:

To verify the robustness of the final model test loaded into the model, prediction performed and rmse reported. Negative predicted values is an issue for XGBoost in setting such as this. A simple way of handling this is to just set the negative values to zero in the final step.

XGBoost achieved a mean root square error of 1.65 compared to the bagging regressor which achieved a score of 1.745.

Bagging Regressor:

- ➔ RMSE: 1.7452
- ➔ Prediction time (Wall time) : 2.08 s
- ➔ Ranking on public leaderboard on Kaggle: 817

XGBoost:

- ➔ RMSE: 1.6466
- ➔ Prediction time (Wall time) : 5.36 ms
- ➔ Ranking on public leaderboard on Kaggle: 719

The XGBoost model placed approximately 100 better than the bagging regressor on kaggle.com, while while using only a fraction of the bagging regressors prediction time. Furthermore as mentioned, XGBoosts' parameter search was heavily subsampled training set. Unfortunately the top 500 rankings in competition have an rmse of approximately 0. A probable sign of overfitting in my opinion. Therefore a final

conclusion on both models performances should not be drawn before the competition closes.

Justification

Compared to Kaggle:
As stated above even though the parameter search was performed on a subset of the data. Final training was done on the full set. Unexpectedly some the model shows some signs of overfitting when test data is loaded into the model, some discrepancies are to be expected. With an rmse of 1.55 and 1,64 the model missed by approximately 5% more on the test data. Well within expectation. The final test will be how well it performs when the competition is finalized.

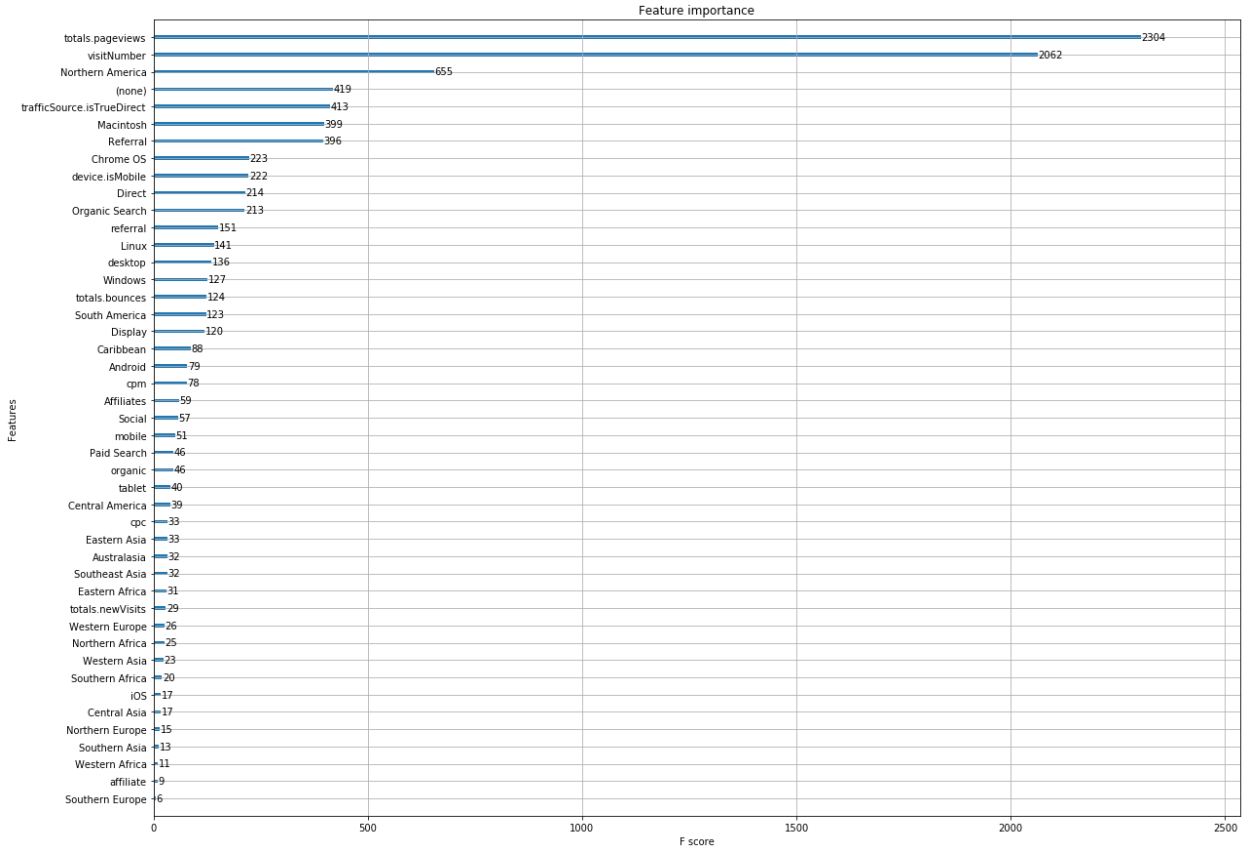
Compared to Bagging regressor:
Previous arguments regarding the search parameters also holds true for this comparison. The model beat the benchmark rmse by approximately 12%.

Conclusion:
The model seems to predict the visitor value within reason. But ultimately it is important to see how these model performs in real life to unseen data. Even though test data is technically unseen. Tuning parameters is an iteratively task, where comparisons about how well a model fits to the test are done. This can and often leads to tuning the model to fitting the test data and not necessarily unseen data.

Conclusion

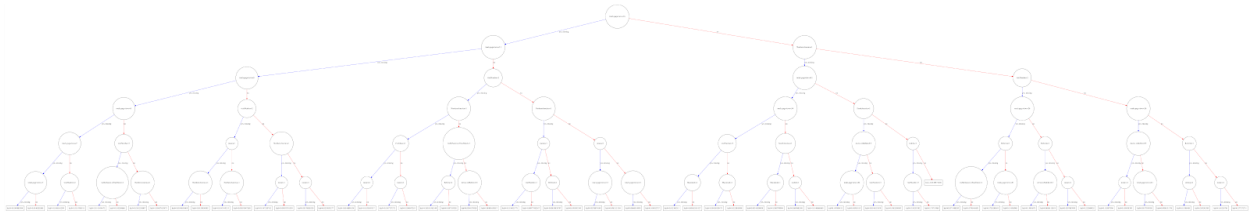
Free-Form Visualization

Fig 11. Feature importance



Feature importance numbers are relative. Designed to show relative information gain compared to other features. From the above chart, total pageviews is the most important feature when prediction a visitors value. Number of visits closely follows and after a significant drop off in importance, a flag for visit from North america follows..

Fig 12. Tree plot



Unfortunately the tree plot is too large to be properly viewed in this paper (separately included). The plot shows how the feature importance chart is used in practice,meaning: The top three

important features are used several times as breakpoints. Example: A Total pageviews is the initial split and single visitorId might se the same split up to 5 later in the tree (depending on route), naturally with different splitting criteria.

Reflection

Data and memory resources was a challenge that persisted throughout the entire project. The challenge was to create a robust enough model to compare it to bagging regressor and also achieve a reasonable rmse score and ranking on kaggle. This while downsampling the training data.

At first the challenge was to sucessfully load in the data provided by google (33 gb). After some back and forth and unsuccessfully using the kaggle notebook ui. The solution was provided by a kernel on kaggle. (10)

In my opinion tuning the parameters are hafl science and half “feel”/experience. Even though I’ve tuned plenty of models before and the parameters for the most part are similar. Research into how to best tune XGboost took some time. After some back and forth I drew inspiration from Cambridge Sparks’ stepwise approach as the parameters chosen aligned with my intuitive feel and hardware resources I had available put restrictions on how many I could afford timewise(7).

Even though feature selection and a deep discussion around techniques for how to handle it, is not part of this project. It is impossible to ignore when it comes to dealing with supervised learning algorithms. Much of the magic in this and and other learning models comes from how one handles feature selections. Picking correct features and while dropping features that are highly intercorrelated.
The kaggle competition is unfortunately closed for new entries but on the public dataset the model now ranks approximately among the top 700, but the top 500 have a rmse of 0.02 or under, an unrealistic rmse and sign of overfitting in my opinion, but we’ll see in about two months time when the official results are in.

Even though XGboost performed better than the baseline bagging regressor. I expected the performance gap to be wider. This might be a result for the extreme subsampling of the training data.

Improvement

If the objective was to win the competition, the first and most obvious way to improve the model is to include the full dataset while training the model.
For parameter search only the most important was included and the search should possibly span over a wider range. I debated back and forth about including gamma or not. It specifies if how much improvement in erros loss must be seen to justify a split.

As noted there are two factors, over- and underfitting competing when optimizing such a model as this. My main concern was avoiding overfitting while keeping (in my optionion) a reasonable root mean square error. But such a statement leave this model open for attacks concerning its' learning rate. A final eta of 0.03 is probably on the high side, but when training time is also a factor so such tradeoffs have to be made.

As a final note, I considered using LightGBM instead as it seems to have overtaken XGBoosts place as the most popular model used on Kaggle. For further improvement this is an obvious place to start.

Links, resources and quotes:

1. https://arxiv.org/pdf/1702.02215v1.pdf
2. https://arxiv.org/pdf/1406.0728v2.pdf
3. Competition page: https://www.kaggle.com/c/ga-customer-revenue-prediction
4. https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/
5. https://xgboost.readthedocs.io/en/latest/
6. https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc273365621
7. https://blog.cambridgespark.com/hyperparameter-tuning-in-xgboost-4ff9100a3b2f
8. https://stackoverflow.com/questions/34469038/understanding-python-xgboost-cv
9. https://www.slideshare.net/ShangxuanZhang/kaggle-winning-solution-xgboost-algorithm-let-us-learn-from-its-author
10. https://www.kaggle.com/sudalairajkumar