Google Analytics Customer Revenue Prediction

CHANG Yingshan (20413368, zdxdsw) | CHENG Ka Leong (20306618, felixcheng97)
LI Haotian (20328915, liht1996) | WONG Yuen Ting (20348434, inezmeow)

Background & Objective

In the business world, especially retail area, marketers see sales volume as the most valuable assets. If the company can understand and anticipate the selling information, the marketing teams can make appropriate investments in promotional strategies. Google Merchandise Store is an online google store which sells Google swag. Currently, it is using Google Analytics to help improve the store performance, but it is not yet satisfactory. Google announces a Kaggle competition and hopes that the model created can be more actionable and can make a better use of current resources.

Our objective is to use the past customer purchasing data to forecast the sum of the logarithm of money spent by a GStore customer in the future.

Cloud Computing Platforms & Tools

1. **Intel DevCloud - Data Preprocessing Platform**
   Considering the number of the data records and the size of the dataset, if we do all the works including data processing and model training on our local computer, it would take days to finish all the work. Intel DevCloud is a strong cloud computing platform, we can make use of it to flatten our training and testing data.
2. **Amazon S3 - Storage Platform**
   Amazon S3 is useful for storing large amounts of unstructured object data. We create an S3 bucket to store the flattened training and testing files. In Databricks, we can access AWS S3 buckets by mounting buckets using DBFS, with the provided ACCESS_KEY, SECRET_LEY, and AWS_BUCKET_NAME.
3. **Databricks (Spark) - Visualization and Model Training Tool**
   To have better data visualization and more efficient model training, we use Databricks (Spark) as our visualization and model training tool, as it supports a lot of useful functions, like Map-Reduce, SQL context, and etc.

Data & Data Pre-Processing

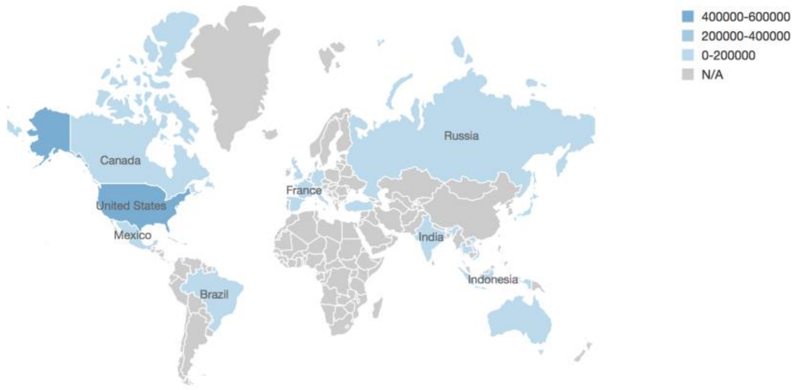
1. **Dealing with Null Value**
   There are a lot of null values in the data set. For example, as a data record is collected into the dataset once a customer visits the GStore webpage, regardless of making an actual purchase or not, over 99% of the data has a null value in the "total_revenue" attribute.
2. **Complex Data Structure**
   Given training data and testing data have complex data format including complex dictionaries and array of dictionaries, Spark does not support using StructType to create a schema for data reading, it is essential for us to pre-process data using Pandas and Numpy before feature engineering.
3. **Cloud Computing**
   The training dataset contains over 1.6 million data instances and the size of it is over 20G. So, main tasks are performed on Intel DevCloud is to flatten all dictionaries. Afterward, the flatten data is uploaded to Amazon S3 for future use in Databricks.

Visualization

After pre-processing the complex raw dataset, visualization of the dataset helps us to better understand data set as well as the relationship among multiple columns, which is also conducive to selecting the most important features. Thankfully, Spark DataFrame API is already a multi-functional platform that also provides data visualization tools. With the help of it, we can visualize the data using bar charts, scatter plots, pie charts, geographical plots, and etc.

1. GeoNetwork

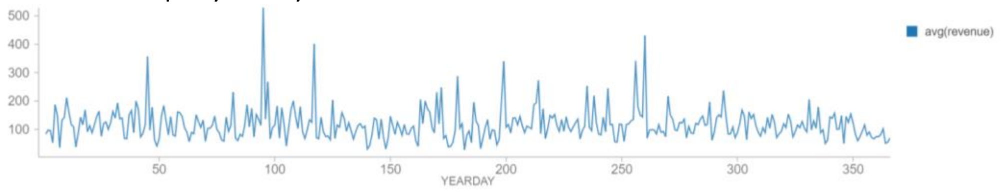
There is an interesting tool on Spark that can display data grouped by country through a map. After a little bit preprocessing, geoNetwork_country column is converted from String to the standard country code for each country. The following map shows the number of data records generated from each country. Note that the US is the greatest contributor in this dataset.



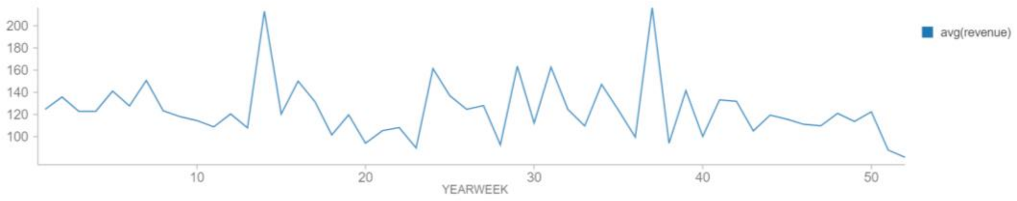
2. Date

In the raw dataset, information about date embedded in the column "visitId", which is read into Spark Dataframe as String and further cast to Date object. Next, features such as "week", "month", "year", "yearday", "yearweek" can be extracted from Date objects.

Sum revenue per yearday is summarized below:



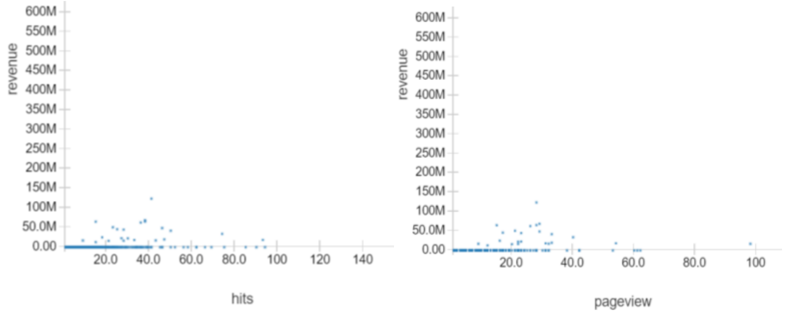
Sum revenue per yearweek is summarized below:



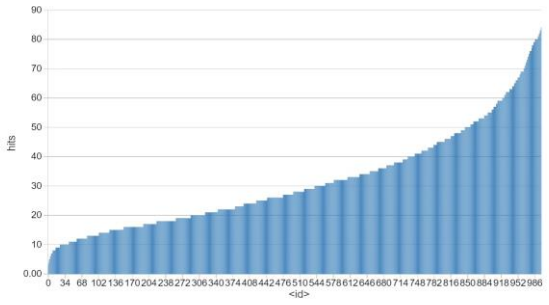
It is noticeable that the occurrence of peaks in the two figures is consistent.

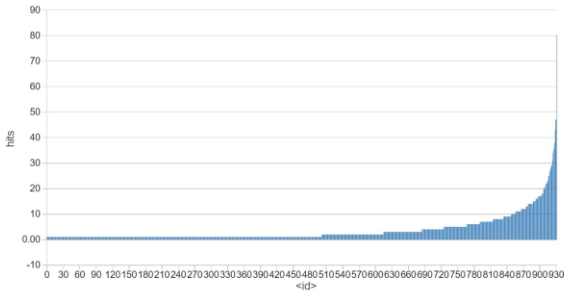
3. Numeric data

For numeric data, we would like to know whether or not a numeric feature has a positive or negative relationship with total revenue (our prediction label). At the very beginning, the first idea we came up with was using scatter plots. For example, we displayed two scatter plots to see if the corresponding revenue is high when total_hits and total_pageveiws are high.



However, it turned out that scatter plot is not suitable for visualizing the underlying relationship since most of the points just clustered near x-axis. Then we developed another way to look into these two features.

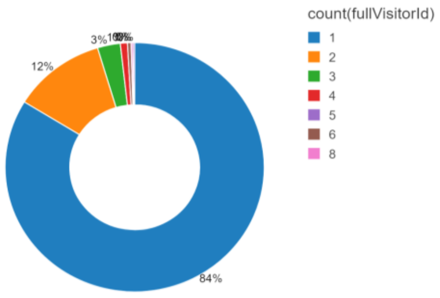


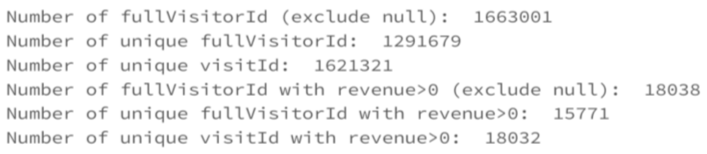


The left figure shows the pageviews (sorted) after filtering out zero revenues, while the right one shows the pageviews (sorted) whose corresponding revenues are zeros. As can be seen, more than 70% of the data records with positive revenues has pageviews larger than 20. On the contrary, almost 85% of the data records with zero revenue has pageviews less than 10. To a certain extent, this indicates that revenue is likely to be greater when pageview is larger.
By the same token, analysis on totals_hits and totals_transactions indicate that larger hits and larger totals_transactions are likely to be associated with greater revenue.

4. **FullVisitorId**

FullVisitorId might be a useful feature at first glance of the data. Different visitors have different purchasing behaviors and if we were able to train a model that understands each visitor's purchasing behavior, it should have been able to make accurate predictions from user-based features.
But this idea was abandoned when we looked deeper into the data:





```
Number of fullVisitorId (exclude null):  1663001
Number of unique fullVisitorId:  1291679
Number of unique visitId:  1621321
Number of fullVisitorId with revenue>0 (exclude null):  18038
Number of unique fullVisitorId with revenue>0:  15771
Number of unique visitId with revenue>0:  18032
```

Simple exploration revealed that 84% of fullVisitorId only corresponds to a single visit. 12% and 3% of fullVisitorId correspond to two and three visits. The rest with more than three visits makes up only 1%. It may also be possible that users can access the online store with different fullVisitorIds. Since we didn't know how fullVisitorId is associated with each real customer, this feature does not carry much importance.

Feature Engineering

1. **Date**

The date feature given in the raw data set is in the format "yyyyMMdd". This column is converted to "DateType" object by firstly casting it to string while loading the data and secondly cast the string to DateType via to_date method. Furthermore, two more features: "yearday" and "yearweek" are extracted from the DateType object by dayofyear and weekofyear methods.

2. **Category grouping: fragmented -> others**

After data exploration and analysis, four categorical features are selected, namely "channelGrouping", "geoNetwork_country", "device_operatingSystem" and "device_browser". In the raw data, there exists a problem that each categorical column has so many categories. For instance, in "geoNetwork_country" column, hundreds of countries have appeared, among which the number of occurrences of most of the countries is negligible (less than 1% of total data)

compared with the total training data size (~ 1.6M). A map function is used to combine the categories in the "long tail" to one category since they are no longer informative. To be more specific, countries whose number of occurrences is less than 5000 are labeled as "other1", while countries whose number of occurrences is between 5000 and 10000 are labeled as "other2". The remaining three categorical columns are processed by the same token.
By doing so, the dimensionality of categorical features is reduced significantly, which saves lots of computing work and memory space when it comes to "one-hot" representation in the later steps.

3. **Feature Crosses**
   Several features are generated by combining two columns of the raw dataset to a single column in order to make the combined feature carry more information.
   Since the raw dataset is generated from browse history, more than 99.9% of the records have no revenue. Therefore, some numeric features like "hits", "pageview" of a single row are meaningless if they are directly put to the machine learning model. Thus, it is necessary to combine these numeric features with another categorical feature.
   The basic idea is, firstly, select a categorical column as "key" and a numeric column as "value". Then group by key and calculate the sum/average/max value. Finally, the new feature column is merged back to the dataset. For instance, we grouped the data by "geoNetwork_country" and calculated the summation of pageviews, which is output to a new column called "sum_pageviews_per_country". Similarly, "fullVisitorId", "yearday", "yearweek" are also selected as "key", while "hits", "pageviews" and "totals_timeOnSite" are selected as "value". Five more features are generated from the key - value space.

Machine Learning Pipeline

1. **Encode categorical features**
   StringIndexer (*pyspark.ml.feature.StringIndexer*) is applied to categorical feature columns to replace each string by an integer as "index".
   Then OneHotEncoderEstimator (*pyspark.ml.feature.OneHotEncoderEstimator*) is used to convert each index to a one-hot vector, where only one entry has "1" and all other entries have "0".
2. **Add continuous variable**
   The column names of continuous (numeric) features are combined with those of categorical features into a single list as "assemblerInputs".
3. **Assemble the steps**
   All inputs are passed to VectorAssembler (*pyspark.ml.feature.VectorAssembler*). In this stage, vectorAssembler will concatenate all feature columns together into a single column vector.
4. **Push all stages to Pipeline**
   All steps above are executed in the pipeline.
5. **Dense vector**
   To make the computation faster, training inputs are converted to a new DataFrame using DenseVector (*pyspark.ml.linalg.DenseVector*) and sqlContext. The new dataframe has only two columns: "features" and "revenue" (label).
6. **Create train/test set**
   The input dataframe is split to 80% and 20%, we use them as train/test set by *randomSplit*.

Model Training & Evaluation

Machine Learning Models

1. **LinearRegression**
   First of all, we used a linear regression model to test if the data has a linear relationship. However, the result is not as good as other two tree models, which illustrates that it is a non-linear relationship.

2. **GBTregressor**
   GBTs are ensembles of decision trees that are iteratively trained to minimize a loss function. It supports both continuous and categorical features.
   Hyperparameters for the best validation are: maxIter = 1, maxDepth=6, maxBins = 32.
3. **RandomForestRegressor**
   Random forests combine multiple decision trees in order to reduce the risk of overfitting. It supports both continuous and categorical features.
   Hyperparameters for the best validation are: maxDepth = 6, maxBins = 32, numTrees = 5.

Predictions and Evaluation

After training, predictions are generated on the testing set. RegressionEvaluator (*pyspark.ml.evaluation.RegressionEvaluator*) is used to calculate RMSE of predictions. Our smallest RMSE on validation data reached 0.07.

Result

After applying data preprocessing, data visualization, feature engineering and, model training on various platforms, we successfully build three machine learning models as mentioned above. It turns out that Random Forest regressor performs the best.

Model	GBTregressor	RandomForestRegressor	LinearRegression
Validation RMSE	0.07452828735717	0.07142599074841	0.24959473159865

Discussion and Finding

1. **Databricks and Spark**
   The memory of a Databricks cluster is only 6G which is not enough for a large dataset as what we used. As the result, we cannot even use a dataframe with more than 20 features for training and our training result could be better if a cluster with better performance is provided Spark does not support use StructType and ArrayType when constructing reading schema, which makes it difficult to deal with complex data format including dictionary. In Spark, several continuous join executions will lead to duplicate columns with the same name. It is quite common to meet this problem, so it is better to rename one column before joining. Due to lazy execution, as the number of features grows, the execution time of every line of code will grow fast. It is necessary to cache the result every several steps to decrease the total running time.
2. **Spark DataFrame and Pandas DataFrame**
   Comparing two kinds of dataframe, we find that the function provided by Spark is much fewer than Pandas. However, Spark supports the use of SQL, which could greatly compensate for those missing functions. For example, when using Pandas, we could directly do the division between two columns but Spark does not support this action. Instead, we need to change the dataframe into a table in SQLcontext and write a command like “select a / b as c from table”.

Data Source and Source Code

Raw Dataset: https://www.kaggle.com/c/ga-customer-revenue-prediction
Flatten dataset on Amazon S3: https://s3.amazonaws.com/4651project/data/train_v2_flatten.csv
Flatten Source Code: http://hlibg.student.ust.hk/comp4651/flatten.py
Visualization Source Code: http://hlibg.student.ust.hk/comp4651/Visualization.html
Gradient-Boosted Trees Source Code & Result: http://hlibg.student.ust.hk/comp4651/GBT.html
Random Forest Source Code & Result: http://hlibg.student.ust.hk/comp4651/RF.html
Linear Regression Source Code & Result: http://hlibg.student.ust.hk/comp4651/LR.html