CAPSTONE PROJECT FINAL REPORT

Submitted to

University of Washington, Seattle
Amplero

Advisor

Megan Hazen

Sponsor

Amplero
Dr.Luca Cazzanti

Group Members

Khyati Parekh
Niharika Sharma
Rajiv Veeraraghavan

14th March, 2018

Table of contents

Introduction and Background

Our aim was to give Amplero a fair idea about whether a user would churn out of a network or not. When a company interacts with a user, it is important that they do it in a timely manner as it can have positive effects on user retention, engagement, and revenue. These attributes are difficult to represent with standard model performance metrics; specialized metrics and key performance indicators may need to be developed. This is among a few of the challenges that we had faced in developing models to analyze the data. Deliverables also consist of model complexity and data requirements assessment.

Amplero is an Artificial Intelligence Marketing (AIM) company that enables business-to-consumer (B2C) marketers at global brands to optimize customer lifetime value at a scale that is not humanly possible. Unlike traditional rules-based marketing automation systems, Amplero's Artificial Intelligence Marketing Platform leverages machine learning and multi-armed bandit experimentation to dynamically test thousands of permutations to adaptively optimize every customer interaction and maximize customer lifetime value and loyalty. With Amplero, marketers in competitive, customer-obsessed industries like telecom, banking, gaming and consumer tech are currently seeing measurable lift across key performance indicators—including 1-3% incremental growth in customer topline revenue and 3-5x lift in retention rates [2].

In the recent economy, subscription marketers are challenged to address the ever-changing behavior of customers. Using multivariate time series, our goal is to predict the behavior of customers. We have used multivariate time series because we are assuming that it describes the erratic behavior of the users along several dimensions and carry much more predictive power than univariate models do. Some ideas and techniques we had for implementing the above are simple regression models, complex classification models and neural nets. We finally settled on using voting classifier and neural nets as our models.

Data

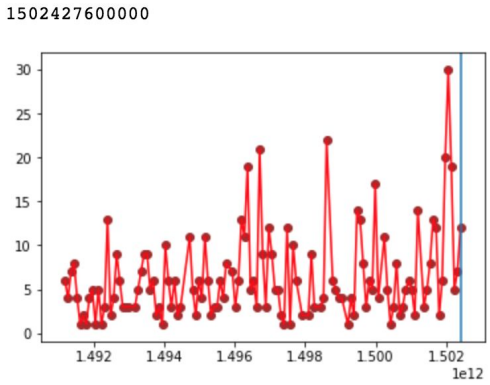
The dataset consists of usage statistics and social attributes time series for approximately 12M prepaid mobile phone users. A random sample of 1M users were selected. Subsequently, the users belonging to the ego networks of these 1M users were added. Finally, the users belonging to the ego networks of these additional users were also added, for a total of 12 M users.

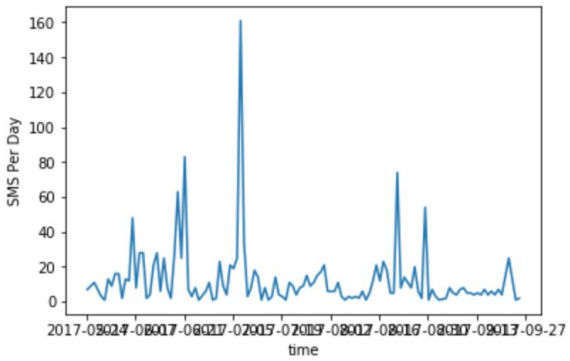
We used the following time series for our analysis:

1) **Voice Call Time Series** - Number of calls per day;
2) **SMS Time Series** – Number of SMS per day;
3) **Data Time Series** – Data used in KB per day;
4) **Recharge Time Series** - Amount and time of a prepaid account recharge; and
5) **Carrier Reported Subscription State Delta Time Series** – Tells whether an entity ID active or inactive.

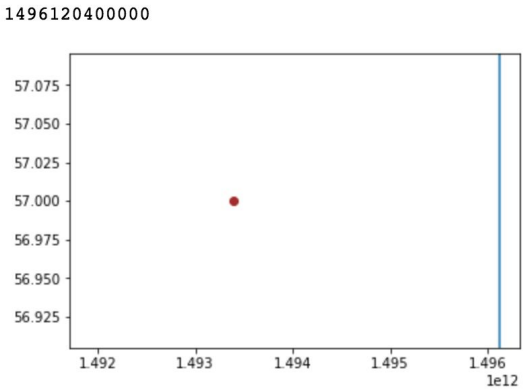
The following graphs represent the activity of a User with given entity ID. The x axis represent the timestamps of the user and the y axis represent the amount of voice calls, SMSes, and recharge the user has done.

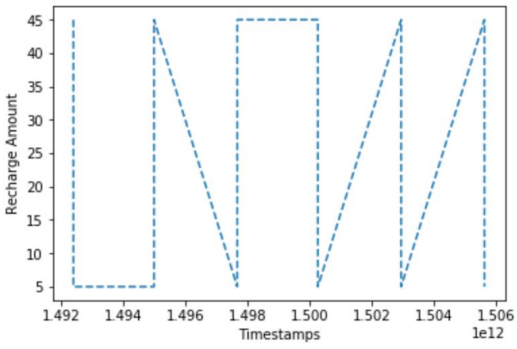
Voice Call



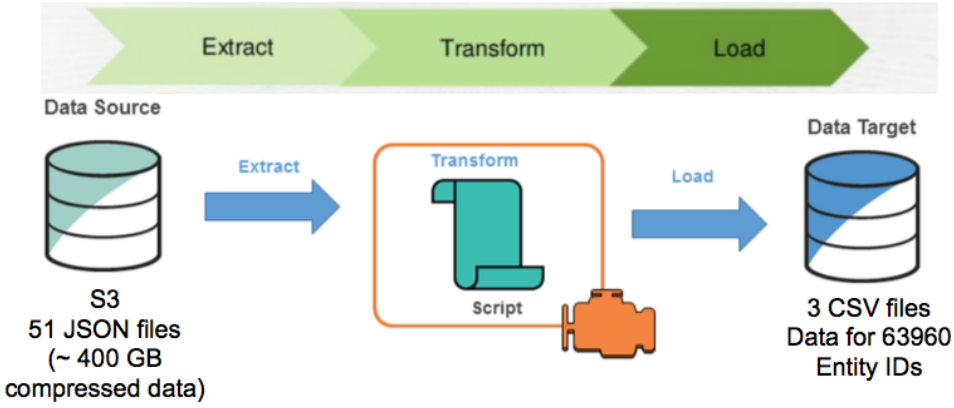


Recharge Time Series





Method Pipelines



Extract Phase of Data

During extraction, the time-series data was identified and extracted from many different compressed json file, using the Boto client from the Amplero's S3 bucket. Initially, it was not possible to identify the specific subset of interest, therefore more data from the time-series was extracted than necessary. The identification of the relevant data was done at a later point in time by exploring the different time-series. The size of the extracted data varied from hundreds of kilobytes up to gigabytes, depending on type of time-series. The same is true for the time delta between two time-series extractions: the time span varied between days/hours and minutes to near real-time.

Transform Phase of Data

During transformation stage, a series of rules or functions were applied to the extracted data in order to prepare it for loading into the end target. Some data did not require any transformation at all - for example the json structure of the time-series data, but the nest json format was removed to increase readability and understanding of the time-series.
Following steps were taken into consideration and executed to transform the data -

- Cleaning of data;
- De-selecting null columns;
- Selecting only certain time-series to load;
- Deriving a new calculated fields;
- Applying data validation;
- Splitting the json into multiple jsons;
- Creating new dataframe/ data structure for the time-series data;
- Creating multiple boto clients to access different S3 buckets;
- Maintaining data security and integrity

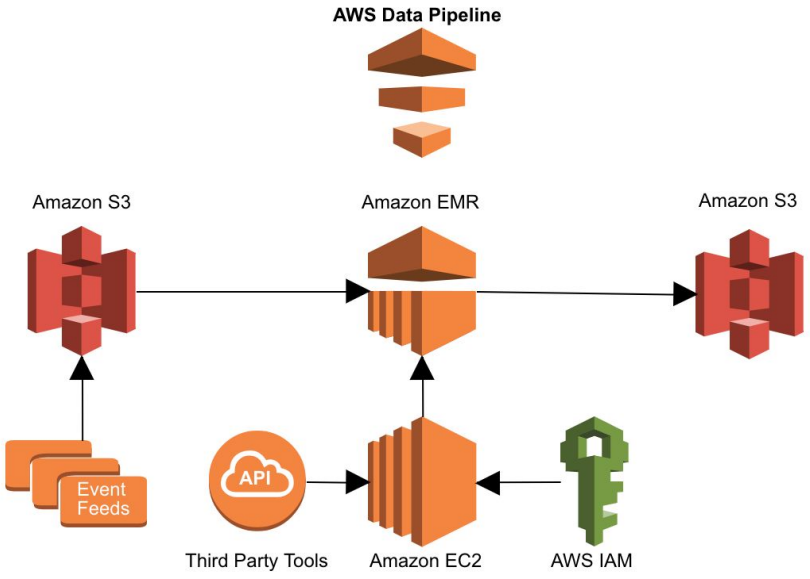
The challenges were -

- Large Compressed files
- Read line by line (parallel processing also time consuming),
- Cannot store Time Series data to a data frame because of memory issues
- Not enough/ satisfactory GZip library documentation

Load Phase of Data

After data was extracted and transformed, it was physically loaded/ transported to CSV files for further processing. The requirement for this step is majorly for the scalability of the models. This step is for data manipulation operations, with a particular emphasis on model implementations, using ETL and the parallel query infrastructure.

We extensively used AWS for the ETL process. The following figure demonstrates the AWS Data Pipeline for our project. The event feeds is the continuous streaming of JSON files to the Amplero's S3 bucket. We used third party tools like GZip libraries, launched julia --toree --torch --ruby --ds-packages --ml-packages --python-packages --jupyterhub on Amazon EC2 instances using AWS EMR. To maintain data security and integrity we used AWS IAM users to launch any service.



Implementation Framework and Details

Voting Classifier

Feature Extraction and Selection

Features considered for each time series / Entity ID

- Mean,
- Variance,
- Min,
- Max,
- Number of Inactive days
    - The time between the last day of active to going inactive for churners
    - The time between the last day of active to 27th sept 2017 for non-churners. We selected 27th sept 2017 because the period spanned by the time series is at least 120 days, ending Sept. 27, 2017.

Using the following Time Series -

- Voice Call Time Series - Number of calls per day
- SMS Time Series – Number of SMS per day
- Data Time Series – Data used in KB per day
- Recharge Time Series - Amount and time of a pre-paid account recharge
- Carrier Reported Subscription State Delta Time Series – Tells whether an entity ID active or inactive. Needed for churn

Model

Rejected users based on the following flow and criteria:

1. Removing NA / Null values
2. Removing users who have activity after they have churned (from churned users only)
3. Data Balancing: Data is biased - 96% non churn and 4% churn / Removed the class bias by using 50-50 split

To select final models for the voting classifier, we -

- Experimented with the feature vector (with & without inactive days)
- Selected different K size for KNN
- Experimented with RBF, linear and poly kernel for SVC

The Voting Classifier consists of three different models -

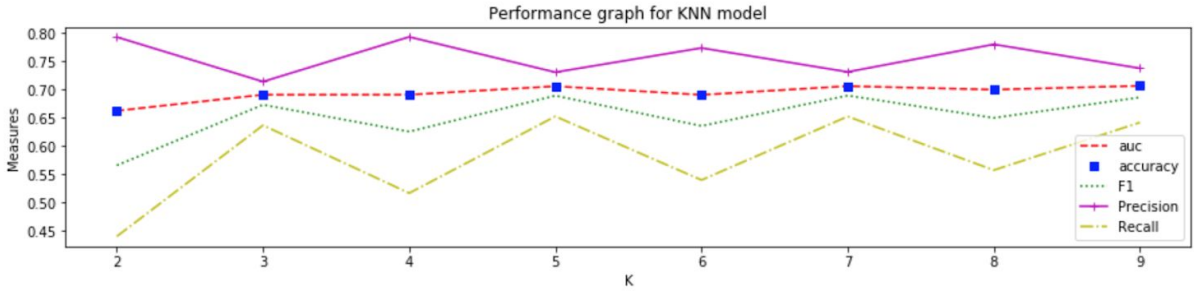
- Logistic Regression,
- SVC [kernel="rbf],
- KNN [n=5]

Results

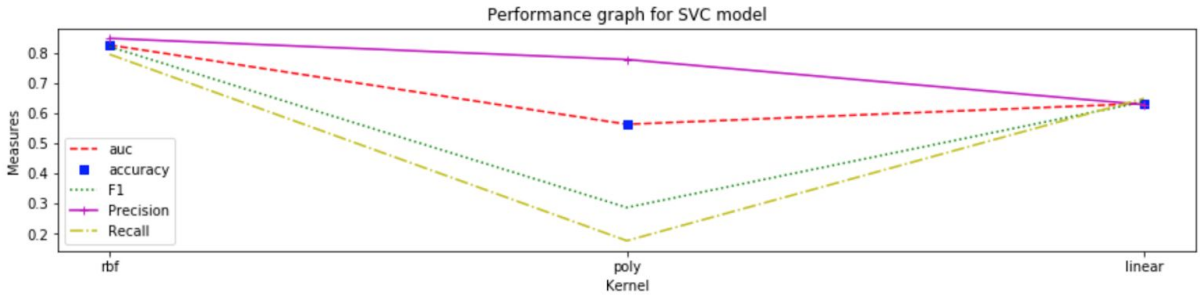
The best result is using Voting Classifier with Best Models: RBF SVC, Logistic Regression, and KNN with soft voting [3, 2, 1] having ~0.80 as the final score. For testing the performance used the following performance metrics -

- AUC,
- Precision,
- Recall,
- Accuracy, and
- F1 score.

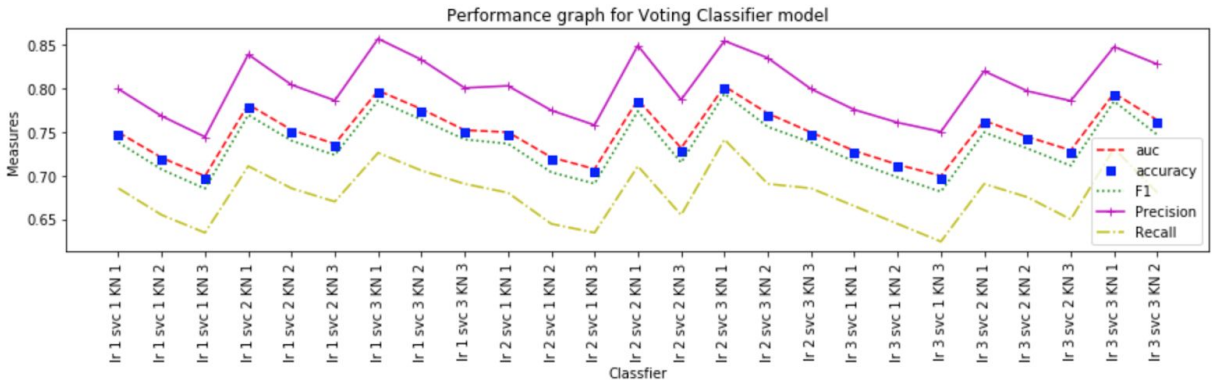
Performance of KNN model varying K from 2 to 9 -



Performance of SVC model varying the kernel -



Performance of Voting Classifier -



Note - Meaning x axis label in the above figure - eg. - lr 1 svc 1 KN 1 means that Voting Classifier consist of Models: Logistic Regression, RBF SVC, and KNN with soft voting [1, 1, 1]

Neural Network

Feature Extraction and Selection

Here, we consider multiple time series and extract features from them to feed to the neural network model (feedforward network - multilayer perceptron). In each time series, we break them into ‘n’ number of chunks which corresponding to ‘n’ features. In each chunk, we consider the average of all the time series values to determine the corresponding feature value for that chunk. Therefore, we would have generated ‘n’ features from each time series. In our model, we consider multiple time series and hence we concatenate the features generated from each time series and thereby obtain the final feature vector.

In the feature extraction process, we ignore 14 days worth of data just before a user churns. This is done so that we don’t use the time series information just before a user churn to predict churn (We would like to predict churn at least 2 weeks in advance).

The number of chunks (n) used can have a huge impact on the model performance. If we use a large value for ‘n’ - it can lead very sparse feature sets and very small value for ‘n’ can lead to compact features. This especially has a compounding effect because we ignore 14 days of data for churners and don’t do that same for non-churners. Hence, the features generated from time series corresponding to churners is bound to be more sparse. We tried different values for ‘n’ to trade off between model performance and the data sparsity problem.

Model

Once we have generated the features from the time series, we undersampled from the majority class to deal with the sampling bias problem.

We then employed a feedforward neural network with multiple hidden layers and multiple hidden nodes (hyperparameters) to predict churn. We tuned the hyperparameters to optimize for performance.

Results

A neural network model with 1 hidden layer and 1000 hidden nodes gave the best performance. Please find below the model performance metrics.
Neural network model including 14 days worth of data before churn:

- Accuracy: 0.7489
- AUC: 0.75
- Precision: 0.7321
- Recall: 0.7446
- F1 score: 0.7575

Neural network model excluding 14 days worth of data before churn:

- Accuracy: 0.658
- AUC: 0.658
- Precision: 0.663
- Recall: 0.644
- F1 score: 0.653

Tools and Libraries

Table 1: Tools and Services

Services/ Tools	**Cost per month**
AWS EC2 Instance m3.2xlarge (On-demand usage)	$54.37
S3 Storage space	$1.4
Amazon EMR	$55.69
Key Management Service	(free tier)
Data Transfer	(free tier)
Python / Jupyter Notebook	Open Source
Scikit learn and other python library	BSD license

* The cost mentioned is what we incurred in the AWS bills

Python Libraries used for this project -

- Matplotlib - for data visualization
- Numpy - for data processing
- Pandas - for data processing
- Scikit learn - for modelling
- DataTime - for date formatting
- GZip - for handling compressed data
- pprint - provides a capability to pretty-print
- io - for working with streams
- boto3 - for AWS S3 connections
- json - for handling data in json format
- StringIO - for Reading and writing strings as files
- statistics - for exploratory analytics
- statsmodels - for experimenting with arima model

Conclusions

Number of inactive days seems to be a very predictor of churn. If we remove the number of inactive of days as a feature in the model, the performance of the model drops.

Voting Classifier and Neural network models can enable mobile networks to identify users that may churn in the future. We think this information is crucial and can be used to provide tailor made promotions to users so that they don't churn from the network.

Future Work/ Planned Enhancements

We would like our models to advanced techniques such as convolution neural networks and LSTMs as they tend to provide improved performance in such uses cases. Convolutional neural networks in particular will be better as they capture local structure within features which other neural network algorithms such as multilayer perceptron fail to do.

We also would like to consider a much larger dataset to avoid issues with overfitting in a real time environment. Currently, we use a smaller dataset when compared to data that will be used in production.

References

1. Amplero
2. meganursula/DATA590A
3. http://shodhganga.inflibnet.ac.in/bitstream/10603/75388/19/19_appendix.pdf
4. http://www.ijmer.com/papers/Vol4_Issue7/Version-2/IJMER-47020105.pdf
5. https://en.wikipedia.org/wiki/Autoregressive_model
6. https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/
7. https://github.com/jaungiers/LSTM-Neural-Network-for-Time-Series-Prediction
8. https://github.com/hmmlearn/hmmlearn
9. BIOST 558 A Sp 17: Statistical Machine Learning For Data Scientists
10. https://en.wikipedia.org/wiki/Extract,_transform,_load
11. https://docs.oracle.com/cd/B19306_01/server.102/b14223/ettover.htm
12. https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/
13. https://www.cs.cornell.edu/courses/cs578/2003fa/performance_measures.pdf