Machine Learning Solutions to Predict Customer Churn

Section 2 - Team 10
Disha Dubey, Ramnath Kamakoti, Bingjie Lu, Bhanu Matta, Andrew Tsai

Project Objective

For any subscription business, accurately predicting churn is critical to long-term success. How a business manages their churn is critical to their success. KKBox is a music streaming service with a data set public ally available at Kaggle. We built various churn prediction models to learn which of their paid customers are likely to churn, and in turn be able to proactively take corrective actions. We used the Spark ML statistics package to build the algorithms, and Hive to ingest the data. We wish to demonstrate the feasibility of machine learning over a large dataset.

Problem Context

KKBox is a popular music streaming service in Asia. With over 30 million tracks, they boast the world's largest Asian Pop music library. They offer a generous, unlimited version of their service to millions of people, supported by advertising and paid subscriptions. This remuneration model is dependent on accurately predicting churn of their paid users. Their remuneration from income is based on the number of active subscribers. KKBox has members subscribe to their service. When these subscriptions are about to expire, the user can choose to renew or cancel the service. They also have the option to auto-renew, but can still cancel their membership any time. We are tasked with building a model to determine whether a user will churn after their subscription expires ("WSDM – KKBox's Churn Prediction Challenge").

Big Data

IBM defines Big Data with the Four Vs: Volume, variety, velocity and veracity are the defining properties of Big Data ("The Four V's of Big Data").

- Volume: the size of the data
- Variety: a measurement of the heterogeneity of the data
- Velocity: how much speed is necessary to process the data
- Veracity: refers to the uncertainty of the data

We did not have any problems with the variety of the data. The data that we obtained was all structured. The data was static and not streaming, therefore velocity was not an issue. Also, the data was from the client themselves and we were assured of some degree of certainty, though data cleaning was still an important step. This problem became a big data problem primarily due to the size of the data sets involved. The three datasets together are over 30 GB, more than could be loaded in this team's RAM. Thus, we classified this as a big data problem.

Methodology

We had a variety of possible solutions to choose from for this analysis. While we explored three overarching philosophies, we moved further in AWS and EMR for our final solution. This was motivated by several factors: ease of use, ability to scale up, and flexibility.

We began by examining the data with local hardware. For obvious reasons, our own machines had the greatest flexibility, and ease of use. We could load our machines with whichever tools we were comfortable, as well as with hardware of our selection. However, this was flawed by the lack of scaling. In fact, amongst our team, none of us had a laptop with greater than 16 GB of RAM, and our data set was more than 30 GB. Therefore, we could not use our local hardware for anything more than sampling and exploration. To examine the entire data set would have been very difficult.

Next, we examined the Minnesota Supercomputing Institute resources, namely Mesabi and Itasca. This solution had the potential for extremely large jobs, allowing us to scale upwards quite high. Looking at Mesabi's specifications, we saw that our total potential memory for a job was more than 60 TB ("Mesabi"). Also, the machines were relatively easy to use, as they simply entailed logging in to the machine, setting up your scratch space, and submitting a job to the queue. The processors are quite fast, and with the DoMC library in R, we saw impressive speed increases. Modules and packages are loaded with shell scripts, and an environment per job needed to be set up through the submission script. An example submission script is in our Github. However, when submitting a large job, we waited in the queue for quite a long time. In one instance, a job remained in the queue for multiple days before erroring out ("Minnesota Supercomputing Institute").

Finally, we looked at AWS. Because all the machines are virtual, spinning up a machine such as an EC2 instance, or an EMR cluster was very easy. However, set up of the machines needed to be done ad hoc. Because of the difficulties in usage, we finally settled on a Qubole wrapper to continue usage of the EMR clusters. In addition, the entire suite of AWS solutions was extensive and easy to explore. We continued, more fully exploring the AWS suite.

Dataset

Customers subscribe to KKBox's services with a monthly subscription. When the subscription is about to expire, the user can choose to renew or cancel the service. They also have the option to auto-renew, but can still cancel their membership any time. We hope to predict whether a user makes a new service subscription transaction within 30 days after their current membership expiration date. If the user does not he is said to have churned. Since the majority of KKBox's subscription length is 30 days, a lot of users re-subscribe every month ("WSDM – KKBox's Churn Prediction Challenge").

The `is_cancel` field indicates whether a user actively cancels a subscription. The problem is further complicated by the fact that a cancellation does not imply the user has churned. A user may cancel service subscription due to change of service plans or other reasons. The criteria of "churn" is no new valid service subscription within 30 days after the current membership expires.

The train and the test data are selected from users whose membership expire within a certain month. The train data consists of users whose subscription expires within the month of February 2017, and the

test data is with users whose subscription expires within the month of March 2017. This means we are looking at user churn or renewal roughly in the month of March 2017 for train set, and the user churn or renewal roughly in the month of April 2017.

Dataset

members.csv (500 MB)

- msno: user id
- city
- bd: age
- gender
- registered_via
- registration_init_time
- expiration_date

Transactions.csv (1.7 GB)

transactions of users up until 2/28/2017.

- msno: user id
- payment_method_id: payment method
- payment_plan_days: length of membership plan in days
- plan_list_price: in New Taiwan Dollar (NTD)
- actual_amount_paid: in New Taiwan Dollar (NTD)
- is_auto_renew
- transaction_date: format `%Y%m%d`
- membership_expire_date: format `%Y%m%d`
- is_cancel: whether the user canceled the membership in this transaction.

user_logs.csv (30 GB)

daily user logs describing listening behaviors of a user. Data collected until 2/28/2017.

- msno: user id
- date: format `%Y%m%d`
- num_25: # of songs played less than 25% of the song length
- num_50: # of songs played between 25% to 50% of the song length
- num_75: # of songs played between 50% to 75% of the song length
- num_985: # of songs played between 75% to 98.5% of the song length
- num_100: # of songs played over 98.5% of the song length
- num_unq: # of unique songs played
- total_secs: total seconds played

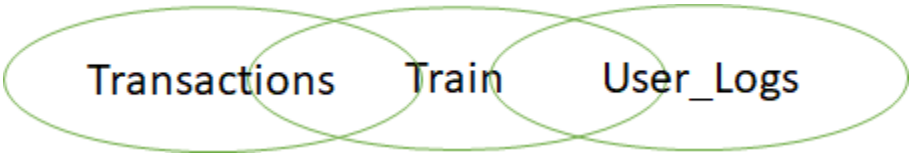
train.csv (88 MB)

- msno: user id
- is_churn: This is the target variable. Churn is defined as whether the user did not continue the subscription within 30 days of expiration. `is_churn = 1` implies customer churn.

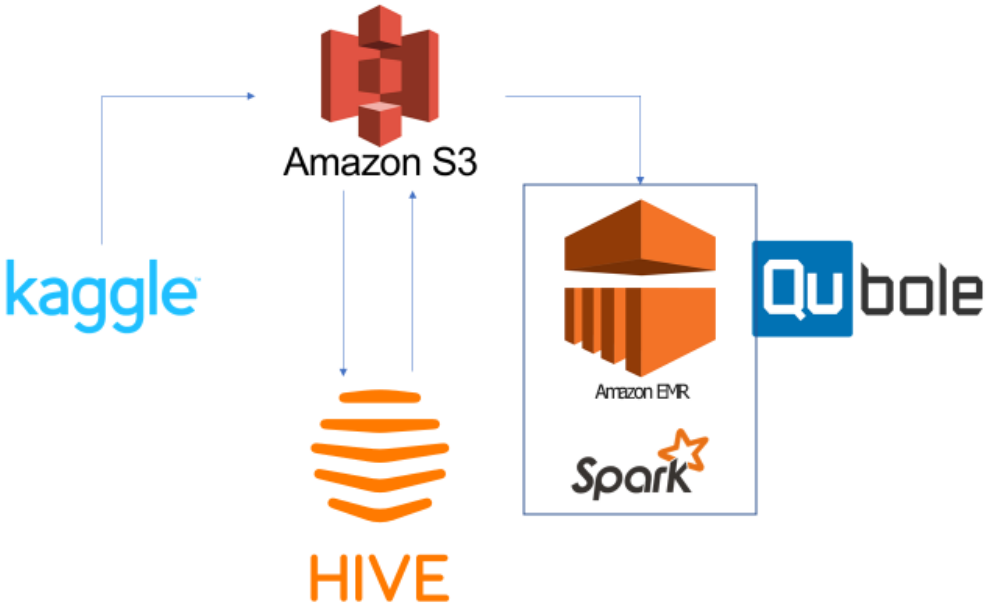
Data Preparation

We began by loading the dataset into Amazon S3. The data cleaning and data preparation was done in Hive. The cleaned and prepared data was then loaded back to S3.

1. Creation of tables for the three datasets in hive and loading the datasets into hive.
2. Removal of null values and outliers from the three data sets.
3. Quality control done to ensure data integrity.
4. aggregating user_logs table at the msno level and fetching the latest records.
5. Treating user log tables for missing values.
6. Joining the three tables together by msno to create the master data table.



Tools Used



Kaggle

The source of our data. We used an open source command line interface to put data directly into S3 from Kaggle.

S3

With the rising demand for voluminous storage in the industry, S3 has become an important player in securely collecting, storing and analyzing data at a massive scale. S3 claims 99.999% durability, stores data for millions of applications while providing comprehensive security and compliance capabilities. S3 further provides query in place functionality which means that robust analytics can be directly run on the data. It is one of the most widely used and the largest ecosystems for storing big data.

- Durability, Availability and Scalability - Data is distributed across a minimum of 3 facilities that are at least 10 km apart. This means that the data you store is always accessible and can be replicated from another AWS region.
- Security and Compliance - S3 supports more security and compliance standards than any other offerings in the sector. Security Standards include but are not limited to PCI-DSS, HIPAA/HITECH.
- Query in Place - S3 is the only cloud platform that facilitates analytics without extracting or moving the data through databases. Developers with SQL knowledge can access vast amounts of data on demand through an array of big data tools available on AWS like Athena and Redshift.

We needed to ingest 32 gigabytes of data which encompassed 5 million rows. While our local machines were able to store the data, our local data analytics tools such as R or Python were unable to ingest the

data due to limited memory capabilities. We thus turned to S3 for storage, to facilitate our analytics pipeline.
In addition, because AWS's local scratch storage EBS is destroyed after an instance is terminated, none of the scratch space could be used for any kind of storage. However, S3 could be used very similarly to HDFS, making usage of S3 as a general-purpose storage solution simple ("What is Cloud Object Storage? – AWS").

Hive

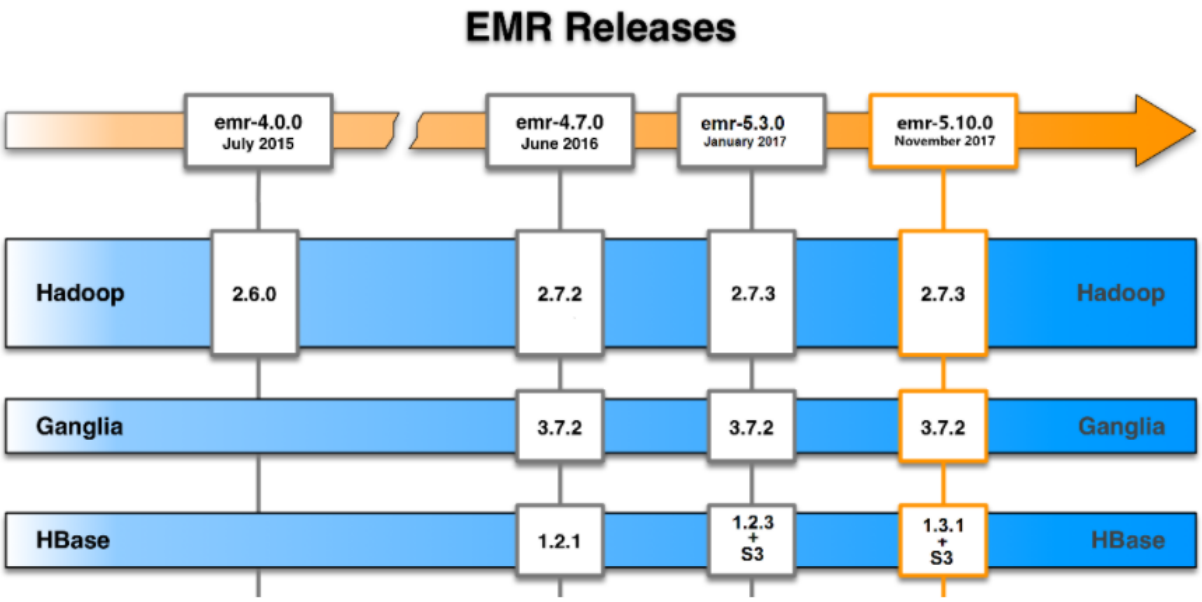
We needed to ingest a large stream of data, and utilize the EMR cluster. We decided to use Apache Hive, a data warehousing solution. Hive is structured on top of Apache Hadoop to read, write, and manage large datasets. Hive operates in distributed storage, and utilizes a SQL like interface.

- Allows data warehousing tasks such as extract/transform/load (ETL), reporting, and data analysis on distributed file systems by enabling access to SQL.
- Acts as a mechanism to impose structure on a variety of data formats
- Allows access to files stored on HDFS or HBase.
- Query execution via Apache Tez, Apache Spark, or MapReduce
- Procedural language with HPL-SQL
- Allows user defined functions (UDFs), user defined aggregates (UDAFs), and user defined table functions (UDTFs) to be created to extend functionality.
- Supports Multiple Formats via built in connectors for comma and tab-separated values (CSV/TSV) text files, Parque, ORC formats and allows extension with connectors for other formats.
- Hive is designed to maximize scalability, performance, extensibility, fault-tolerance, and loose-coupling with its input formats.

("Apache Hive").
Hive was used in this project for data cleaning. On the lookout for a data tool that allows access to a distributed file system, in this case, S3. We chose Hive to do the data preparation. Scripts were written to remove outliers, do quality control, aggregate data to the required granularity, and join the different data sets into one single dataset for usage in Pyspark.

EC2/EMR

Elastic Cloud Compute, or EC2, is Amazon's central cloud computing platform. It provided the backbone of our project.



Amazon EMR Release Velocity

With Elastic MapReduce, we can provide many compute instances to process data at any scale. We can increase or decrease the instances we use manually, and pay for what we use. For all our processing including data ingestion and munging, we used EC2/EMR instances. Hive was straightforward as we were able to use data from S3 in a straightforward manner. However, Spark was not quite as straightforward. While a self-managed EMR was easy to set up, and saving shell scripts in S3 eased some troubles, we found the ad hoc set up to be tiresome. In fact, it is our conclusion that self-managed EMR would be difficult to scale, and a wrapper is necessary as a quality of life issue ("Amazon EMR – Amazon Web Services").

Qubole

Qubole is a one stop solution for integrating cloud architecture to improve developer experience, collaboration, and ease of use. It helps in switching to an autonomous platform where all the solutions can function in the cloud, which lowers the costs and increases the flexibility. This approach also entails the movement from traditional data warehouses in physical spaces to data lakes in the cloud. Combining APIs, accessing machine learning algorithms with Spark, being able to code in Scala, Java, or Python and sharing notebooks among developers to ensure parallel development are some of the many unique features that this wrapper possesses.

Because of the above-mentioned difficulties with EMR, our team chose Qubole as a wrapper. This solution solved many of the difficulties, including cluster set up, notebook interaction, and other EMR automation. However, our team again had difficulty with collaboration on Qubole. We found the documentation of many features to be lacking. However, we were able to put together Pyspark regressions on the entire Kaggle data set. Also, Qubole integration with Github was straightforward. We note, however, that Qubole does not integrate well with the UMN enterprise version of Github.

Spark ML

The key differentiator between our local machines and EMR is the cluster computing. With the distributed architecture, loading and processing large amounts of data are possible. To leverage the cluster, the Spark toolset provided a seamless way to use the distributed disk space, memory, and processing power of the cluster.

Spark specializes in iterative computation which makes Spark ML faster than other machine learning tools. We are mindful of the quality of the algorithms in terms of computation expense across which we make these comparisons, because EMR clusters are charged by the minute ("Apache Spark").

For the KKBox dataset, we ran a logistic regression, and compared the experience of coding this solution with other architectures. We note here that our local machines were loaded with the Cloudera virtual machine, which sped up our development as we did not need to wait for the EMR clusters to develop the Pyspark code.

In Pyspark, we can create pipelines for machine learning problems. This includes setting up variables, feature extraction, training the model and predicting. We used Pyspark for a classification problem and realized it is as flexible as using Python in any other development environment. This helped us realize that Pyspark offers good solutions for advanced algorithms such as collaborative filtering, topic models, or Latent Dirichlet Allocation.

Conclusion

Ultimately, we were able to develop a working model to predict KKBox's user churn with a logistic regression model with an AUC of 0.78. Our next steps involve feature selection, model tuning, and other techniques to further refine our accuracy.

These technologies had quite a steep learning curve. Many of our difficulties revolved around the set up of the development environment. IAM issues, security policy misconfigurations, and other configuration difficulties plagued our initial stages. Qubole we did not find to be well documented, and configuration of the machines, and loading packages, we found to be time consuming. However, after initial set up, we were able to develop code quickly and easily.

We also note that AWS's services are interconnected well. S3 and EC2 worked together very well and intuitively. Complicated system architecture can be implemented easily and quickly. In addition, we also note that Amazon's Machine Language suite were significantly easier to use than our S3/EMR/Qubole solution. However, this ease of use came at a steep price: at $0.10 / 1,000 predictions, to test Amazon Machine Learning suite against our training data would have cost us our entire free education allotment!

References

1. Apache Hive. (n.d.). Retrieved December 07, 2017, from https://cwiki.apache.org/confluence/display/Hive/Home
2. Apache Spark. (n.d.). Retrieved December 07, 2017, from https://spark.apache.org/mllib/
3. Amazon EMR – Amazon Web Services. (n.d.). Retrieved December 07, 2017, from https://aws.amazon.com/emr/
4. The Four V's of Big Data. (n.d.). Retrieved December 07, 2017, from http://www.ibmbigdatahub.com/infographic/four-vs-big-data
5. Minnesota Supercomputing Institute. (1970, December 01). Retrieved December 07, 2017, from https://www.msi.umn.edu/
6. Jeykottalam Follow. (2014, November 21). Machine Learning Pipelines. Retrieved December 07, 2017, from http://www.slideshare.net/jeykottalam/pipelines-ampcamp
7. Mesabi. (n.d.). Retrieved December 07, 2017, from https://www.msi.umn.edu/content/mesabi
8. What is Cloud Object Storage? – AWS. (n.d.). Retrieved December 07, 2017, from https://aws.amazon.com/what-is-cloud-object-storage/
9. WSDM – KKBox's Churn Prediction Challenge | Kaggle. (n.d.). Retrieved December 07, 2017, from https://www.kaggle.com/c/kkbox-churn-prediction-challenge

Appendix

Github: https://github.com/andrewtsaimn/kaggle-kkbox