**Predicting Customer Churn for KKbOX**

**Maroon - Team 4**

Bekzat Alish

Bryce Quesnel

Ishwarya Ravikumar

Karthik Narapareddy

Shuyu Sui

PySpark ML, Amazon ML and H2O Sparkling Water

Contents

[Problem Introduction 2](#_Toc500351075)

[Dataset Description 2](#_Toc500351076)

[Methodology 2](#_Toc500351077)

[Data storage – Amazon S3 3](#_Toc500351078)

[Storing and Manipulating the Data using Amazon S3 3](#_Toc500351079)

[Building our User Profile using EMR/Hive 3](#_Toc500351080)

[PySpark and Spark ML 4](#_Toc500351081)

[Introduction 4](#_Toc500351082)

[Modeling with Spark ML 5](#_Toc500351083)

[Evaluation of PySpark and Spark ML 7](#_Toc500351084)

[Amazon Machine Learning 7](#_Toc500351085)

[Introduction 7](#_Toc500351086)

[Workflow of Amazon ML 7](#_Toc500351087)

[Amazon ML Evaluation 10](#_Toc500351088)

[H2O & Sparkling Water 10](#_Toc500351089)

[Introduction 10](#_Toc500351090)

[Sparkling Water 11](#_Toc500351091)

[Modeling with Sparkling Water 12](#_Toc500351092)

[Process 12](#_Toc500351093)

[H2O Evaluation 13](#_Toc500351094)

[Comparison of Platforms 13](#_Toc500351095)

[Conclusion 14](#_Toc500351096)

# Problem Introduction

KKbOX is the leading music streaming provider in Asia and currently has a library of more than 30 million tracks. They are like the Swedish company, Spotify, both offering an unlimited membership supported via advertisements, as well as paid subscriptions that are ad-free. Kaggle currently has an open competition for predicting user churn from KKbOX’s paid users, and this is where our project idea developed from.

Churn is a problem many businesses face in the modern-day world of subscription services. As more and more companies move towards a subscription basis, particularly cloud services, we found this to be the perfect classification problem for us to both learn new skills, and evaluate different tools on.

For this task, we wanted to exploit the provided user history and music metadata to understand the user behaviors and predict whether a user will renew his/her subscription within 30 days after their current subscription expires. More than that though, the focus of our project was on evaluating 3 key machine learning platforms; Spark ML (PySpark), Amazon ML and H2O.

# Dataset Description

***transactions.csv:*** Transaction level data of users - Contains fields that include their payments, plan details, and cancellations etc.

***user\_logs.csv:*** Daily user logs describing listening behaviors of a user - Contains fields that describe how a user is interacting with the platform on a day to day basis like number of songs he/she listened only half way through, number of unique songs played on a given day etc.

***members.csv:*** user information - Contains user demographics and details like age, channel of registration, when did he/she register, when would the registration end, city etc.

While the actual problem dealt with churn, which is very generic to Data Sciences, we attempted to leverage and compare different big data machine learning platforms currently available in the market owing to sheer volume of the data involved. We tried to focus on qualitative and quantitative aspects of these platforms to understand what are the right circumstances to use them. We ended up choosing PySpark, Amazon ML and H2O as we wanted to compare GUI and non-GUI platforms.

# Methodology

Our approach was to first gather data from all provided files into a Hive database, then create a condensed dataset that could be run with our models on each platform. We would then evaluate each platform on the identical data set through the same set of tasks; some cleaning and munging, formatting, parameter tuning (where applicable) and finally modeling. We wanted to test both a logistic regression (simple classification) and random forest (more complex classification) in each platform, where available (Amazon ML has a very limited set of models, so we were not able to evaluate a random forest through that platform). Following these sets of tasks, we would be able to evaluate each of the platforms on 4 main categories: performance, flexibility, speed and learning curve

# Data storage – Amazon S3

# Storing and Manipulating the Data using Amazon S3

We chose to utilize Amazon Web Services for the main data storage and manipulation we needed to perform for two mains reasons. First, we wanted to better understand what was available in Amazon and how to link these different services together. Second, we were nervous about working with 30GB of data and wanted to make sure we could scale up in computing power to get the tasks done. Amazon S3 for storage and using Hive on an EMR instance were the systems we settled on for these reasons.

After downloading and unzipping the data from Kaggle’s website, we then uploaded it to AWS. This was somewhat of lengthy process and took roughly 1.5 hours, but that was expected and not much of an issue as we suffered no performance issues on our computers while uploading. Once uploading, our next step was sharing a single user’s S3 bucket with the rest of us so, we would not be duplicating data. This was where we think S3 starts to deteriorate a bit; the user permissions were very clunky and after spending much time on it, we realized there was no way for another user buckets to show up on your own S3 page; in order to access a different bucket, the link had to be sent directly to the other person via email/chat. Although not a major inconvenience, our team unanimously preferred other file sharing competitors like Google Drive for collaborating with files

## Building our User Profile using EMR/Hive

The main advantage of S3 is that it allows you to access its files easily via Hive when using an Amazon EMR instance. Through this feature, it was relatively simple to load our data after it was uploaded to S3. We initially tried a **M3.XLarge** instance but found it was not powerful enough for our computing, so we then moved up to use a **M4.4XLarge instance with 64 GB of RAM and 16 cores** to make sure we had enough power to query our data in a timely matter. This cost us roughly $2.40 per hour, so we were very purposeful in our use of this resource.

Knowing how large the data set was at 30GB, our goal with Hive was to maintain the necessary info from each of our tables, but get the data into a format in which it could be easily modeled on with minimal munging. As we were predicting churn on a user level, we decided to create a finalized table that represents each user’s ‘profile’. Our finalized attribute selection can be seen in our GitHub, titled ‘*dictionary.xlsx*’ for the data dictionary and ‘*final\_with\_churn SAMPLE.csv*’ for a sample of the table.

Using the Hue interface for Hive on our Amazon EMR instance, we first created each of the three tables as external tables stored in S3. From there, we wrote a Hive query to gather our final attribute section. Note that this query is accessible via our GitHub in txt documents ‘create\_table\_hive.txt’ and ‘overwrite\_table\_hive.txt’.

Running the hive query was one of the areas in which we would do further exploration moving forward. To create the user profile, it took about 40 minutes of run time for the query to complete. Before running the query, we created an additional Hive table to store this final user profile info, then overwrote that table with all the data from running the query itself. Future investigation could be done to understand how adding more nodes to the cluster vs using a EMR instance with more power could affect the timeliness of this as we only experimented with two. Regardless, we were able to complete our user profile in a single day using our Amazon EMR instance, which in the end cost us only $14.20.

Our opinion on S3 using Amazon EMR for Hive was mixed. While fairly intuitive to use individually, we thought S3 was lackluster in regard to collaboration. Additionally, Amazon EMR performed sufficiently for creating our user profile data with Hive; we thought the flexibility of Amazon’s variety of EMR instances was a major benefit when you’re not sure exactly how much computing power you’ll need.

The only downside we found of using EMR with Hive was unlike the virtual machines we used in class locally, tables in Hive were removed as soon as the EMR instance was terminated. Although you can store tables externally in S3 and import them quickly when running a new instance of Hive, it is not as convenient as a local machine in which Hive tables remain static and require no import. This is a minor complain; we feel comfortable recommending S3 and EMR for big data storage and aggregation tasks.

# PySpark and Spark ML

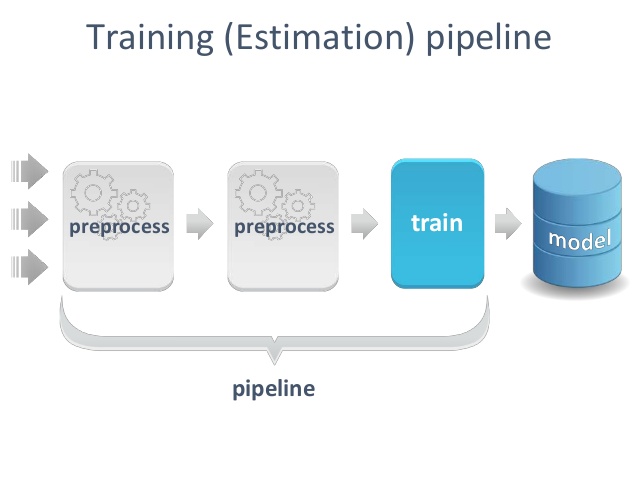
## Introduction

You might already know Apache Spark as a swift and powerful framework that is designed to efficiently run streaming, machine learning, graph processing, and SQL querying applications with a fast, iterative access to huge datasets. This section of the paper discusses the advantages of PySpark API and its interaction with the Spark ML programming platform, which allows to build machine learning algorithms using big data.

With PySpark API, data scientists who are familiar with Python can use their knowledge of this programming language to perform distributed data transformations on massive datasets, apply machine learning techniques, and obtain the results in Python-friendly format.

RDDs are sets of objects that represent data and are essentially the original building blocks of Spark. RDDs data storage model provides three main advantages: they are lazy, they are compile-time safe, and they are based on Scala API. However, building transformation chains is inefficient and difficult to read using RDDs. PySpark supplies two other data structures that are easier to work with – DataFrames and Datasets. Those data formats provide higher level abstraction on the data and allow to use query-type operations to manipulate the data.

PySpark allows data scientists to use Pandas – an open-source Python library for easy data structure manipulation and data analysis tools. Pandas by itself is limited to running on a local machine and can handle only relatively small datasets. Pandas users compromise with those disadvantages by down-sampling the original massive datasets, but training machine learning algorithms is always better with all the data available. Spark DataFrames that store data in a distributed storage system such as HDFS, providing dramatic increase in computational power through parallelizing data processing across multiple clusters. Spark DataFrames and Pandas integrate very well by allowing to use advantages of parallel computing and Pandas capabilities. Users of PySpark can take advantage of Pandas-based functions by aggregating, transforming, and computing data in PySpark, and then returning the datasets in Pandas format. Since PySpark supports all standard Python libraries, Python can unleash the power of Spark with only minimal modifications to the existing Spark code.



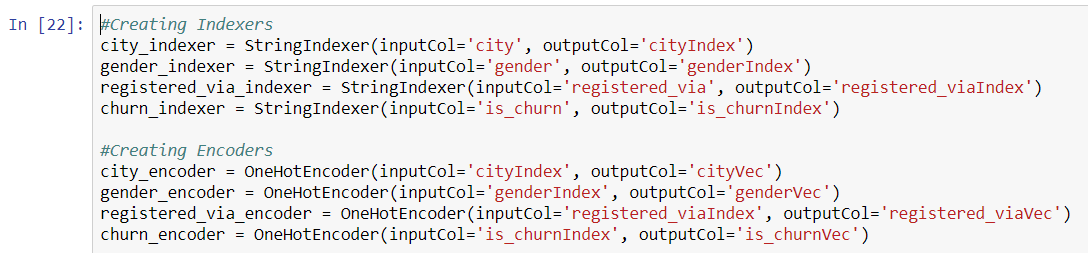
An important feature of Spark ML that allows creation of practical machine learning workflows is pipeline. A typical machine learning problem will involve many different lines of codes for feature transformation, selection, and processing. Pipelines provide a cleaner and more organized machine learning solution by allowing to wrap multiple steps of data extraction, transformation, feature selection, and model training into a single sequential workflow. The input DataFrames passes through each stage of the pipeline workflow in a sequential order until a machine learning model is trained in the final stage.

## Modeling with Spark ML

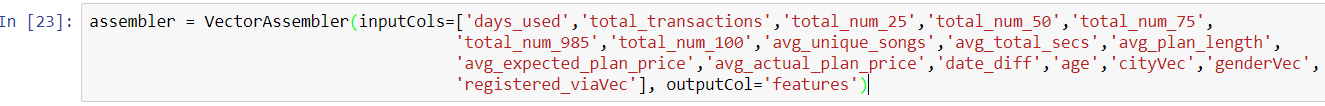
For example, let’s look at how we built our pipeline in Spark ML using PySpark on the DataFrames that we previously cleaned up.



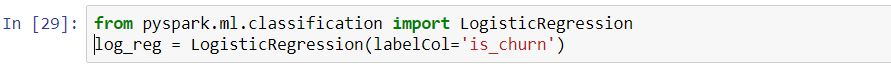
We load Spark ML libraries that will be used in building our ML pipeline.



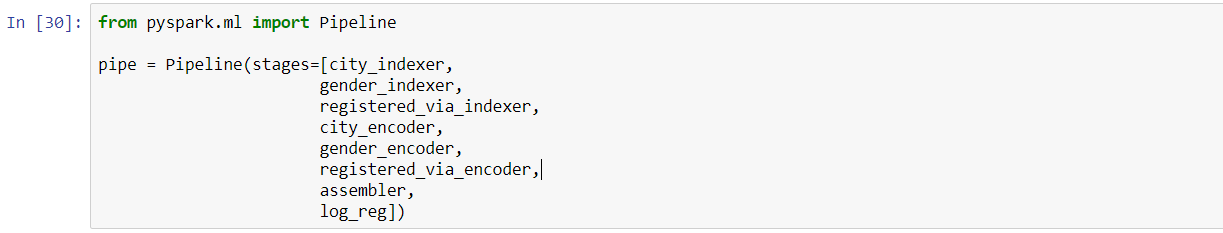
We build Indexers and Encoders for the categorical and binary variables in our dataset to match the format that is required by Spark ML algorithms.



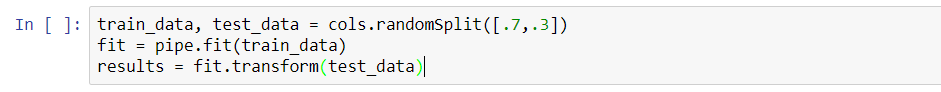
We build an Assembler that chooses the variables from the input dataset that we want to use in training our machine learning models. For example, the first model we build is logistic regression.



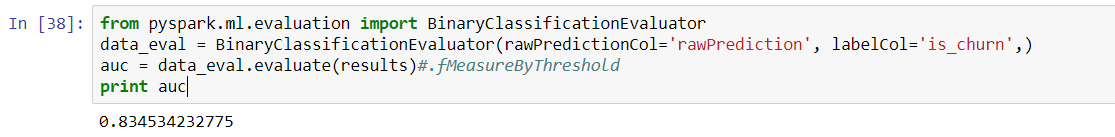
PySpark ML library for the logistic regression model is imported and the model instance is created. Note that the churn indexer and encoder were not used in this step; this is only required to be done on the label with tree models such as the random forest we used.



PySpark ML library for building the pipeline is imported and a sequential set of data transformation (indexers and encoders), feature selection (assembler), and model training (the last step) is included in the pipeline. Notice that in all previous steps we only built the pipeline but did not actually apply it on the dataset.



In this final stage, the dataset is randomly split into training and testing, and the model is fit and applied on the test set.



## Evaluation of PySpark and Spark ML

Our logistic regression model took roughly 2 minutes to run, while our random forest took between 5 and 6 minutes. A model can be evaluated using of the available metrics. We used AUC (Area Under Curve) as a performance metric for our model. The AUC of the model we built using a decision tree was 83.45%, while our random forest performed at 86.11%. One area we found lacking in PySpark 1.6 was binary evaluation metrics, however we did not count this against PySpark as future version (2.0) had a much more thorough set of options.

Overall, PySpark and Spark ML provide a powerful machine learning solution by combining the computational power of distributed operations and flexibility of libraries such as Pandas that are run locally. Although the learning curve is high, PySpark provided the largest range of algorithms. Please note that our entire notebook is available to view in our GitHub, titled *‘KKbOX\_LR\_RF.ipynb’*.

# Amazon Machine Learning (AML)

## Introduction

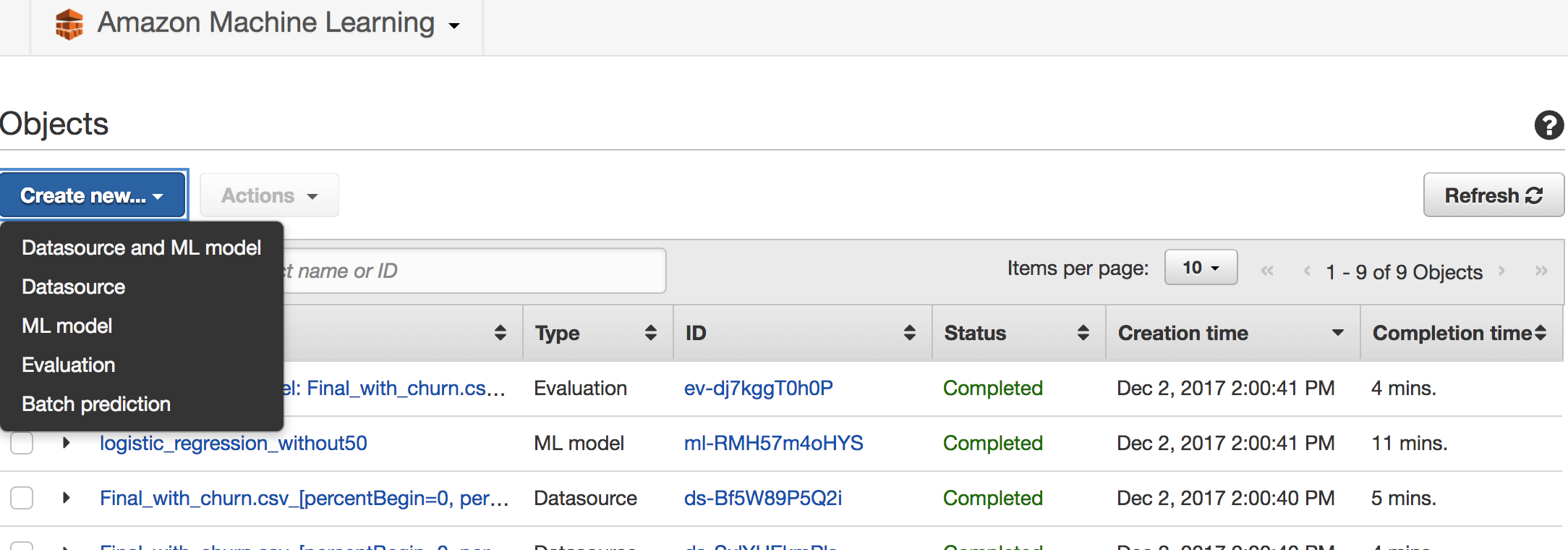
Amazon Machine Learning (Amazon ML) is provided on Amazon Web Service and we were initially very passionate about exploring the Machine Learning module. Amazon ML is fully integrated with the S3, EMR, EC2 and other services on AWS platform. In wild contrast to our expectation, we found that Amazon Machine learning provides very limited resource within the Amazon Machine Learning module. Logistic regression is the only model available for classification, while linear regression is the only model for numeric prediction.

AWS also has a separate deep learning module offered on AWS marketplace, where Amazon could charge additional fees for using the platform. AWS deep learning AMIs are compatible with the popular deep learning packages, but the relative cost is also higher than AWS machine learning module.

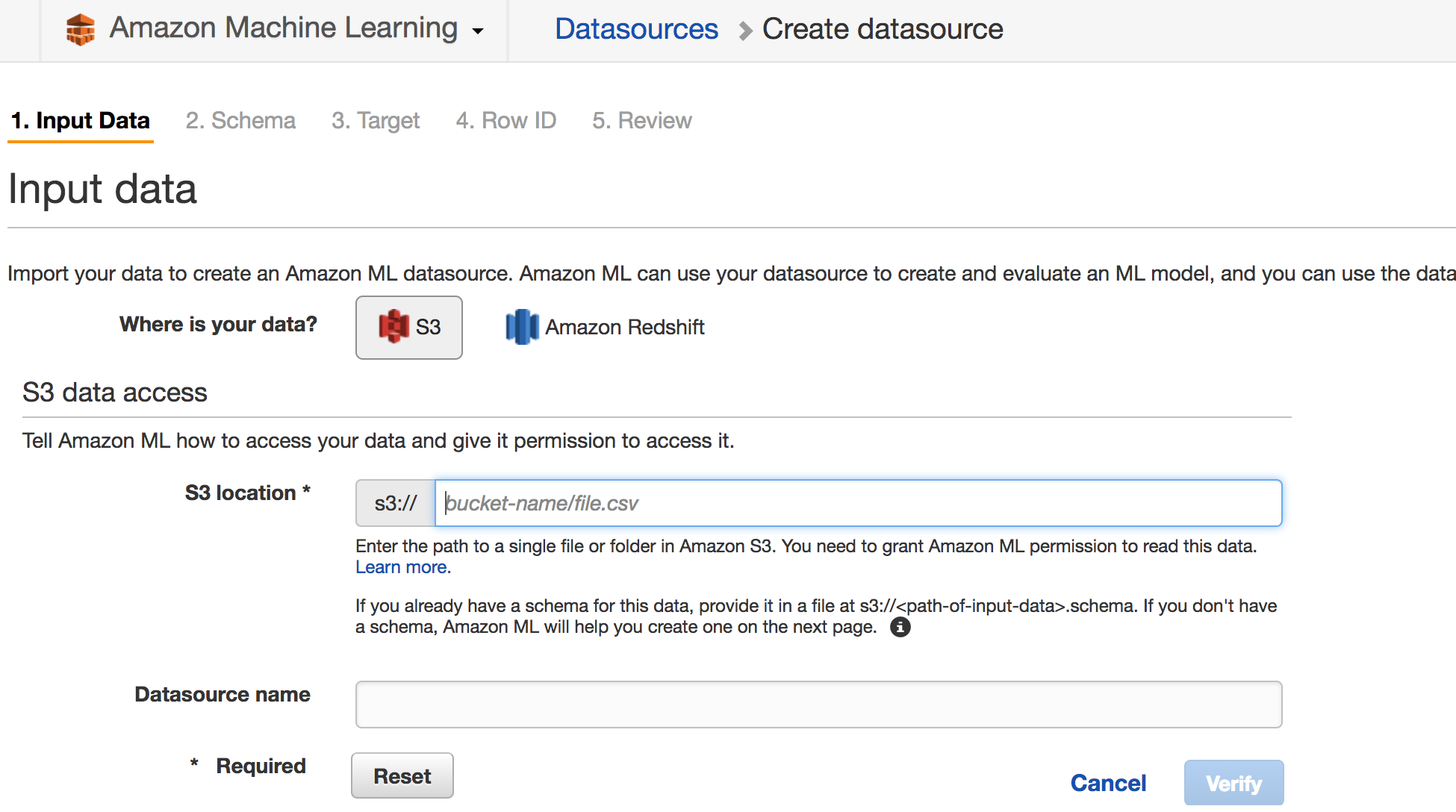
## 

## Workflow of Amazon ML

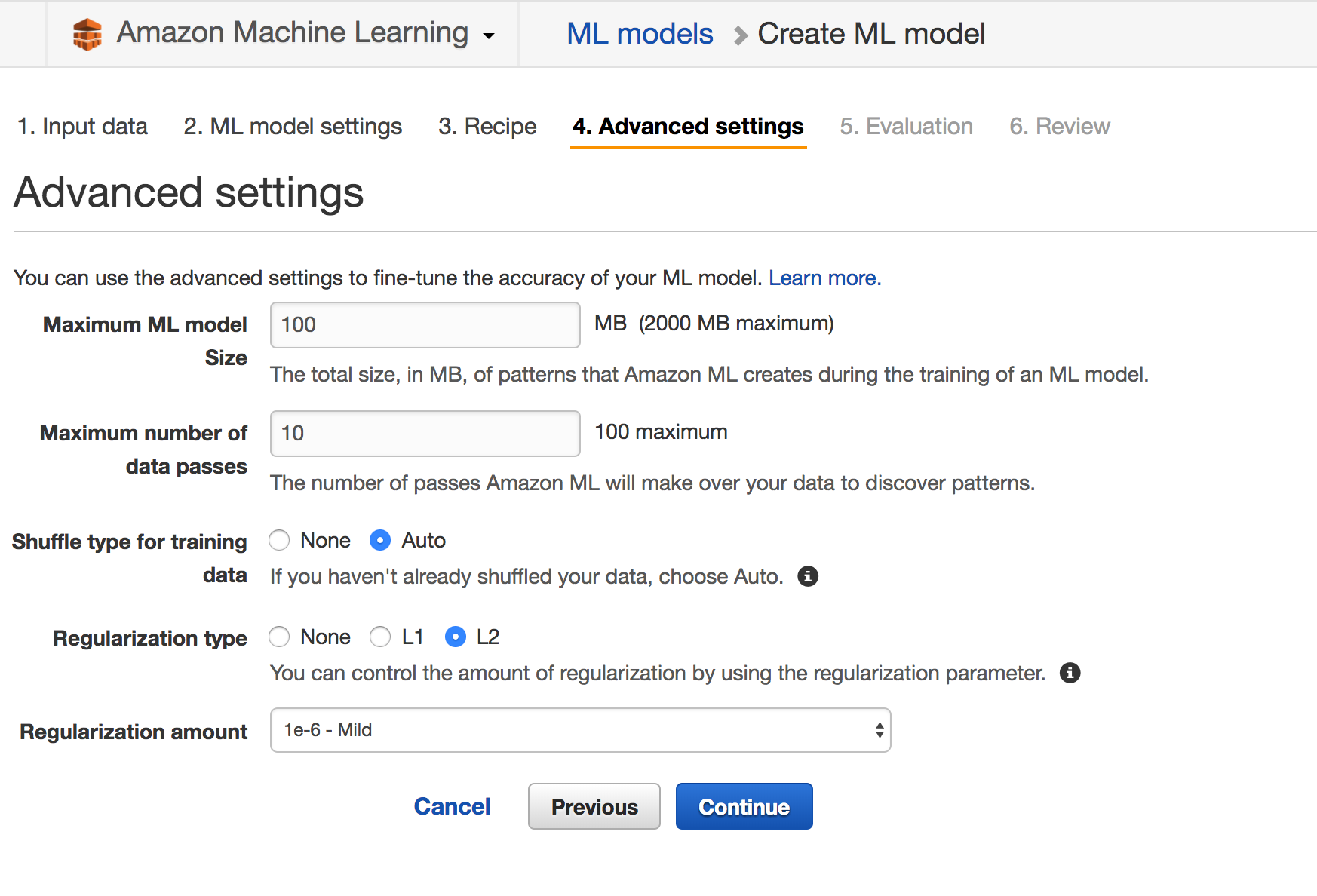
As a GUI based machine learning module, Amazon ML’s workflow is straightforward. The workflow consists of three parts, creating data sources, building machine learning models and finally model evaluation.

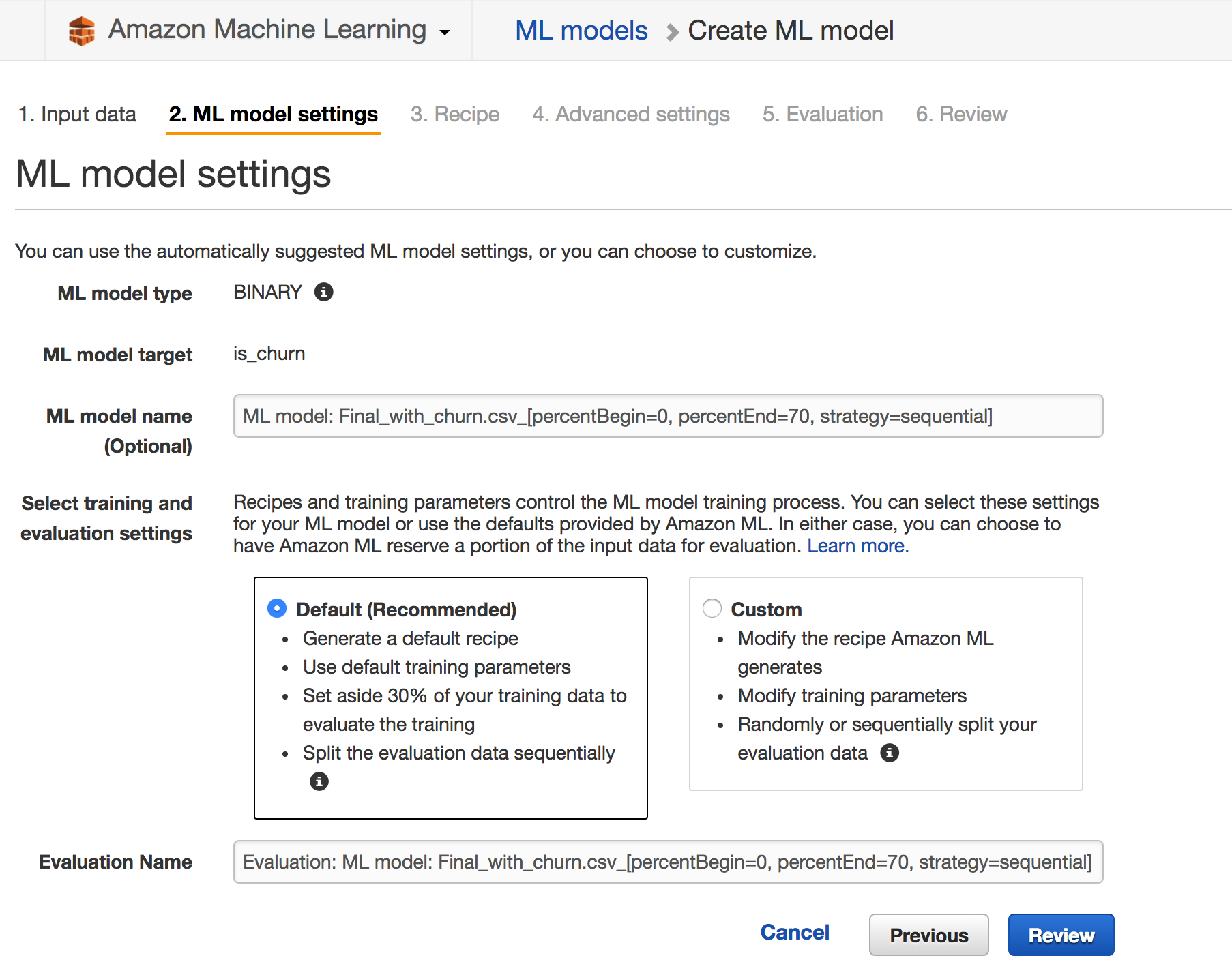


After entering Amazon ML from console, we could choose create new data sources on the main page. And we could choose either to import from S3 or Amazon Redshift as data sources. To import data from S3, we also need to change the bucket policy for the data we want to import. We used the code provided by Amazon.

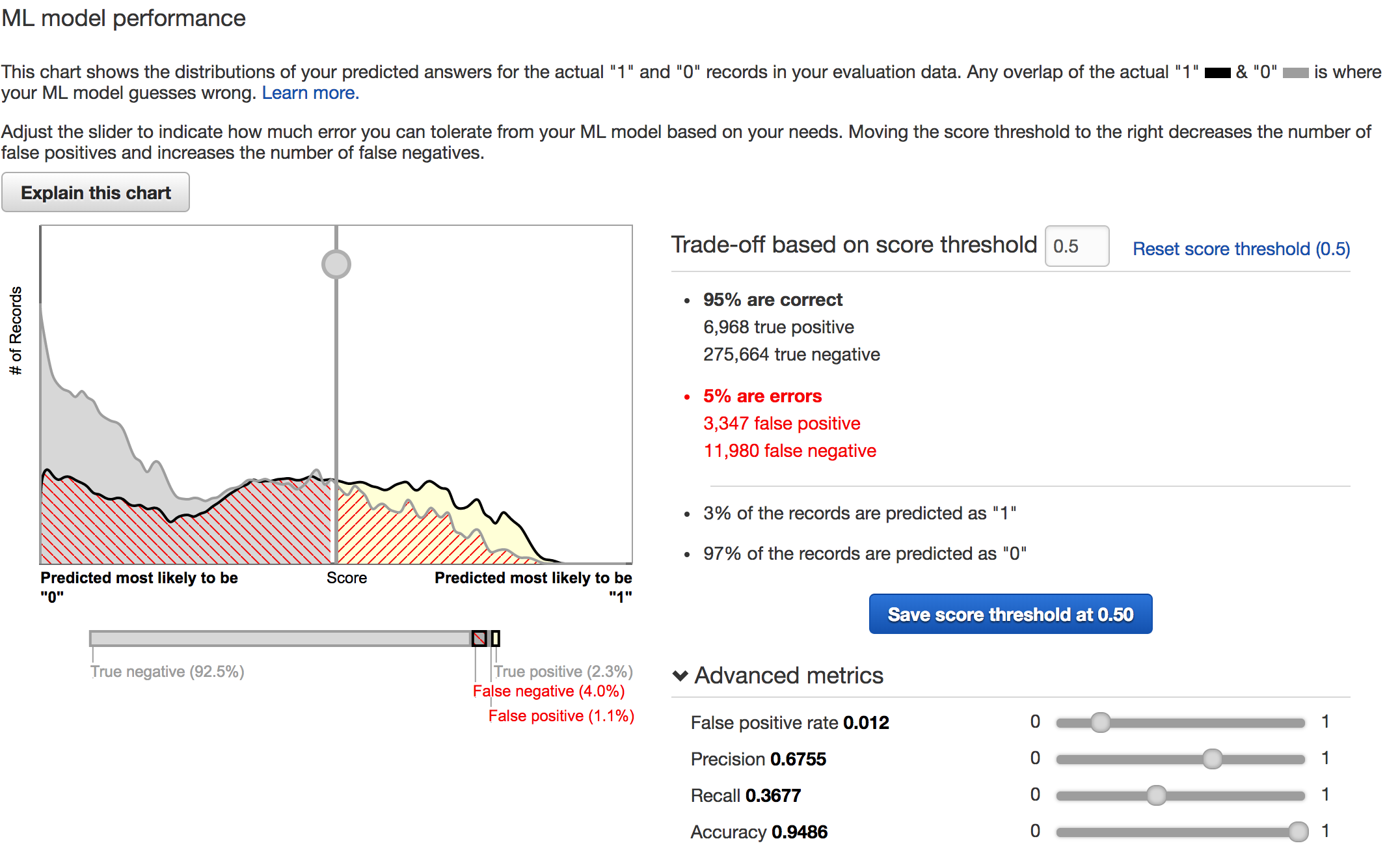


We could then create the binary classification for our churn prediction problem at the Amazon ML console by clicking the create new model button. Regularization and train test split could be customized through the console. Feature selection and grid search was not provided at the moment. If we choose to evaluate the model during model creation, evaluation would run automatically once the training completed.





Alternatively, we could also use a separate test set to evaluate the model by create a separate evaluate module session through the console. The evaluation would give confusion matrix output and accuracy result automatically. Ideally, we would want to see an interface for users to choose the optimal threshold for cost sensitive analysis.



## Amazon ML Evaluation

The model run on Amazon ML gave us the accuracy of 94.86%. This is because that the model implemented regularization during the model building process. However, the number of model is highly limited within the module, making it a less than ideal choice for machine learning on big dataset. For now, it also does not support cost sensitive analysis, and as it is GUI based, and the cost matrix based on threshold could not be exported to provide further support.

The Speed of AWS also tends to be slower than Spark and H2O. Amazon ML takes 11 minutes to fit the model and 4 minutes to perform an evaluation. Which is ten times the time we need from Spark or H2O.

The only good thing about Amazon ML is that it is relatively easy to learn with online step by step tutorial provided by the firm. Compared to the complicated pipeline concept on Spark, Amazon ML is relatively easy to new learners. However, it does not provide much customization for the time being.

# H2O & Sparkling Water

## Introduction

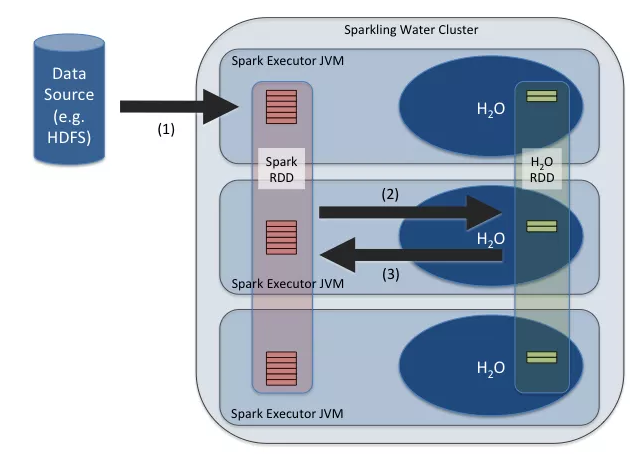
With Amazon ML being restrictive, we needed another framework to benchmark our Spark ML models. The options we considered were H2O and Mahout. Since H2O is well known for Deep Learning, we decided to utilize H2O algorithms for our modelling process.

H2O is open source framework which can work on top of distributed data clusters for machine learning model deployment. It provides advanced models for both supervised and unsupervised learning which can be easily deployed in production environment in a simple and efficient manner. The models, such as Deep Learning, Ensembles, Random Forests, run on H2O’s distributed Map/Reduce framework and multithreading is done by Java Fork/Join framework with parallel distributed data processing, storing it in a columnar format in a compressed way.

H2O is a versatile platform since it can be deployed on a local machine (H2O Flow), a large distributed data clusters on Spark (H2O Sparkling Water, which we used), or a deep learning over RNN through TensorFlow (H2O Deep Water). H2O’s key advantage is its feature of automation - it can automatically detect variable type and automatically encode variables to be passed to a model - this eliminates the necessity of a pipeline like you’d need in PySpark. In addition, it is highly customized to operate over multiple platforms such as Windows, MAC, Linux etc. and offering wide support for languages such as Java, Python and Scala. The H2O Rest API is used by H2O’s web interface (Flow UI), R binding (H2O-R), and Python binding (H2O-Python), Scala binding and Java binding

## Sparkling Water

Sparkling Water extends the functionality of H2O by deploying Spark on H2O to process big data. H2O’s algorithms can be run on distributed data clusters on Spark using Python or Scala. We used Sparkling Water to model the churn prediction on the DataFrames obtained from Hive.



## Modeling with Sparkling Water

Like Spark context, we create a H2O context, which will have functionalities required for modelling



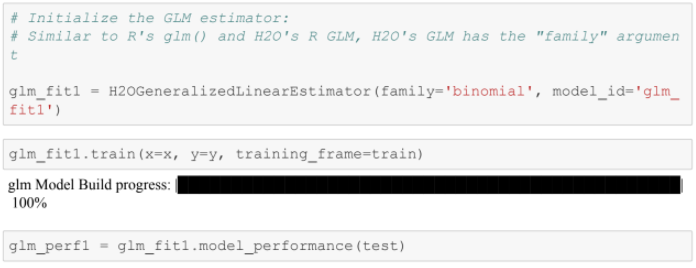
Once the data is read, we designate the churn variable as factor. This ensures that the model identifies this as a categorical variable and not a numerical variable and performs classification (logistic regression) instead of linear regression



The missing values are imputed, and the invalid data is removed to transform the data for modelling



The data is split into training and testing datasets in a ratio of 70:30. Then, a Generalized Linear Model is run on the training data and evaluated on the test data.



## Process

1. We initialize the H2O context similar to a Spark context. Then, we load the input file. Since H2O can automatically detect input data types, there is no necessity to specify the types of the data and the schema is automatically inferred. The output label is factorized explicitly since the model needs to know the output variable
2. We remove the invalid input data and impute the missing values. Then, the data is split into 70%-30% ratio for training and testing data. The input and output columns are chosen
3. Using Generalized Linear Model, we train the model on training data and the model is tested on the testing data

## H2O Evaluation

H2O performed very efficiently on Spark, producing a logistic regression model which ran for 18 seconds, producing an AUC of 86.5%. Similarly, for Decision Tree, it ran in approximately 7 minutes producing an AUC of 93.2%.

Performance of H2O was very efficient and captures the information in the model with very less number of steps required. The only limitation is that there are limited models available for the end user and most of them are advanced, making it little hard for a beginner. But H2O is expanding its offerings and model libraries at a brisk pace, so we can predict there would be more widespread adoption of H2O in the enterprise environment.

# Comparison of Platforms

We evaluated each platform on a 1-3 rating scale for four main categories, summarizing our experience. A perfect score would be 12, with 3 in each category. Each category is below:

* Performance: Based off the available features of the platform, how well does our prediction do?
* Flexibility: What type of options and features does the platform offer? How many models/parameters are available?
* Execution: How quickly does the platform run?
* Simplicity: How easy is the platform to learn and use?

|  |  |  |  |
| --- | --- | --- | --- |
|  | **PySpark ML** | **Amazon ML** | **H2O** |
| **Performance** | 3 | 1 | 3 |
| **Flexibility** | 3 | 1 | 2 |
| **Execution** | 2 | 1 | 3 |
| **Simplicity** | 1 | 3 | 2 |
| **Total Score** | **9** | **6** | **10** |

All platforms excelled in some areas, and all platforms has drawbacks.

1. PySpark we felt had the most overall options and flexibility when it came to modeling, but these additional options and flexibility resulted in a higher level of complexity than the others. What took a single line of code in H2O often took quite a few more lines of code than PySpark, and therefore we scored PySpark lower on simplicity. Additionally, evaluating PySpark models took significantly longer to run than either Amazon ML or H2O and there were very few options available for evaluation metrics.
2. Amazon ML was a disappointment to us. With only a single classification model (logistic regression) and no optionality to tune parameters, we were not able to evaluate this platform to the extent of either of the others. Although easy to familiarize and use, until they develop more algorithms and options, we do not recommend Amazon ML for machine learning. The ML process on the platform is easy to grasp and learn. Features within the ML model itself are also very disappointing, upon comparison with Spark and H2O
3. H2O was very satisfying to use. We had minimal issues installing and found it relatively easy to use and learn since we had no prior experience with it. One issue we found was its model selection is smaller than PySpark model, but it generally included the better performing models we would use. For example, there was no traditional decision tree, just a random forest classifier. Beyond that, we found it simpler to use than PySpark and were able to build our models much faster comparatively

## 

## Conclusion

Based off our experiences with each of these platforms, we would recommend H2O for anyone looking to pick up a new machine learning platform. PySpark may better fit situation where a large variety of models and more customization are needed, but we think H2O is a better overall platform to use. Additionally, H2O is relatively new, and we expect the available models and options to continue growing forward. Finally, based off our experience with Amazon ML we do not recommending using it; other GUI based machine learning approaches may be better, but Amazon ML is still lacking in many areas.

Amongst everything all platforms and models, the model we are most confident is H2O’s random forest. We did not choose our Amazon ML logistic regression because although the AUC was slightly higher, we felt there was no room for improvements through parameter tuning or feature engineering. We highly recommend using H2O for big data machine learning as we believe the platform will only improve with time.

References

* IBM. (December, 2016). *PySpark: High-performance data processing without learning Scala*. Retrieved from <https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=CDW12366USEN>
* H2O.AI. (September 24, 2014). *How Sparkling Water Brings H2O to Spark*. Retrieved from <https://blog.h2o.ai/2014/09/how-sparkling-water-brings-h2o-to-spark/>
* Pushkarev, S. (May 14, 2017). *Spark ML Pipeline serving*. Retrieved from <https://www.slideshare.net/StepanPushkarev/spark-ml-pipeline-serving>