Predicting Customer Churn Using Apache Spark Machine Learning

YATIN SHARMA, DIKSHA YADAV[1,*]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: yatins@indiana.edu, yadavd@iu.edu

project-001, May 2, 2017

This report presents a reference implementation to the Customer Churn Prediction Project that is built using Apache Spark Machine Learning. In this report, we discuss the models used for predicting customer churn along with their accuracies. We will also discuss the software deployment on Chameleon and Jetstream cloud computing environments using Ansible and Cloudmesh Client scripts.

Keywords: Prediction, Bigdata, Apache Spark, ML, Hadoop, ,Ansible, Cloudmesh, Analytics

https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IR-P016/report/report.pdf

CONTENTS

1. INTRODUCTION

One of the most important business metrics is churn rate- describing the rate at which your customers stop doing business with you. Our aim is to build a predictive model which uses data science to predict in advance which customers are at risk of leaving. Such model will allow the company to be proactive and focus on such customers. We used Apache Spark[**?**] Machine Learning library(ML)[**?**] for fitting a predictive model on our dataset. Detailed analysis and modeling was carried out in Python Programming language and different machine learning algorithms were tested on the same dataset. Application deployment was initially done on localhost, followed by deployment on two different clouds. Finally Performance metrics on each of the infrastructure was compared and has been included the report.

2. ARCHITECTURE

Our application was prototyped on smaller dataset locally and was then modified to run on cluster. Fig **??** below describes the runtime architecture of a distributes spark application.

In distributed mode, Spark uses a master/slave architecture with one central coordinator and many distributed workers(2 in our case). The central coordinator is called the driver. The driver communicates with the distributed worker called executors. The driver runs in its own Java Process and each executor is a separate Java process. A driver and its executers are together termed a Spark Application. A Spark application is launched on a set of machines using an external service called cluster manager.

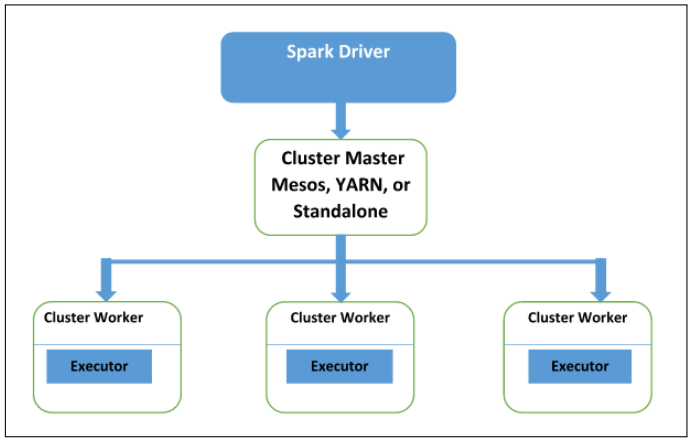

Fig. 1. Runtime architecture of spark in distributed mode[**?**]

We used Hadoop YARN[**?**] as our cluster manager in cloud deployments.

2.1. The Driver

The driver is the process where main() method of our program runs. It is the process running the user code that creates a SparkContext, creates RDDs(Resilient Distributed Dataset), and performs transformations and actions.

2.2. Executors

Spark executors are worker processes responsible for running individual tasks in a given spark job.Executors have two roles. First, they run the tasks that make up the application and return the result to the driver. Second, they provide in-memory storage for the RDDs.

2.3. Hadoop YARN

YARN is cluster manager that provides a data processing framework to run on a shared resource pool, typically on the same node as the Hadoop filesystem(HDFS)[**?**]. In our project, an HDFS Spark cluster was set up on cloud using Cloudmesh Client.[**?**] The dataset was stored on the same HDFS system which was then accessed by our spark appication for quick data processing. Using YARN in spark is also straightforward with the help of spark-submit as follows:

```
$ ./bin/spark-submit --master yarn yourapp
```

2.4. Launching a program

Spark provides a single script to submit our program to it called spark-submit. Through various options, spark-submit can connect to different cluster managers and control how many resources the application gets. For example following command can be used to launch a spark application.

```
$ ./bin/spark-submit --class path.to.your.Class
--master yarn --deploy-mode cluster [options]
<app jar> [app options]
```

3. TECHNOLOGIES

Table **??** lists the set of technologies that were used in building this project.

Technology	*Usage*
Python[**?**]	Development
Hadoop[**?**]	Big Data Technology
Spark[**?**]	Big Data Technology
Apache Spark ML[**?**]	Machine Learning Library
Cloudmesh Client[**?**]	Cloud Resource Manager
GitHub[**?**]	Project Repository
Ansible[**?**]	Application Deployment & Configuration Management
Chameleon[**?**], **JetStream**[**?**]	Cloud deployment & Benchmarking

Table 1. Technologies used

- **Python**: Python is a high level programming language having various popular libraries for data analysis and machine learning algorithms, making it our preferred choice fo the project.
- **Hadoop**: Apache Hadoop is a framework that allows processing and storing huge amounts of data across a distributed resource pool. Therefore, in order to improve overall performance and scalability Hadoop was installed at the foundation of each virtual machine in the cloud.
- **Spark**: Apache Spark is a cluster computing platform designed to be fast and general purpose. It extends the popular MapReduce model to efficiently support more types of computations including interactive queries and stream processing. We will primarily use Spark's machine learning library to build a predictive model on our dataset.
- **ML**: ML is Spark's primary machine learning library. It is DataFrame[**?**] based API that makes practical machine learning scalable. The main concepts in Spark ML are:
  1. DataFrame A DataFrame is a Dataset organized into named columns.[**?**]
  2. Transformer A Transformer is an abstraction that includes feature transformers and learned models. It converts one dataframe into another.[**?**]
  3. Estimator An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer. For example, training/tuning on a DataFrame and producing a model.
  4. Pipeline A Pipeline is specified as a sequence of stages, and each stage is either a Transformer or an Estimator. It represents a workflow, which is run in a specific order.[**?**]
- **Cloudmesh Client**: Cloudmesh Client provides an application programming interface(API), which allows us to manage a set of cloud resources. It standardize access to various clouds and clusters. In our project, Cloudmesh Client is used to set up Hadoop virtual cluster with Spark on two cloud environment. Following tasks are performed using Cloudmesh Client:
  1. Uploading Public Key- using Cloudmesh's key add and upload commands.
  2. Uploading Security Rules- using Cloudmesh's secgroup commands.
  3. Creating Hadooop/Spark Cluster- using Cloudmesh's deploy command.
- **GitHub**: GitHub is 'a web-based Git or version control repository and Internet hosting service' [**?**]. We used Github repositories to store our files related to documentation, ansible scripts and python code.
- **Ansible**: Ansible is an open source automation platform that automates Application Deployment, and many other IT needs. In our project, Ansible is used to manage the configuration and automate deployment of software stacks on to the clouds. Ansible scripts are run via localhost by providing it with the IP addresses of the cloud VM's set up using Cloudmesh Client.

4. CLOUD INFRASTRUCTURE

Two clouds were selected for deployment: Chameleon Cloud and Jetstream. For successful deployment, following applications/utilities were used.

4.1. Chameleon Cloud

Chameleon is a collaborative cloud service funded by National Science Foundation primarily meant for research community. In our project three virtual machines were created on this cloud. One acting as master node and the other two as executor nodes for our spark application.

4.2. Jetstream Cloud

Jetstream is a cloud service led by Indiana University's Pervasive Technology Institute (PTI) in collaboration with other universities [**?**] across the United States. In our project three virtual machines were created on this cloud. One acting as master node and the other two as executor nodes for our spark application.

4.3. Cloud Hardware Comparison

Table 2. below shows a comparison of key computing resources on Chameleon and Jetstream Cloud environment.

Clouds	*Chameleon*	*Jetstream*
CPU	Intel Xeon X5550	Dual Intel E-2680v3 "Haswell"
Cores	1008	7680
RAM	5376 GB	40 Tbr
Speed	2.3 GHz	2.5 GHz
Storage	1.5 PB	2TB

Table 2. Cloud Hardware Specifications.

5. GETTING STARTED

The following section describes the process involved in building the spark application to predict the customer churn.

5.1. Loading the data

The 'Customer Data' used in the project contained 21 columns and 7000 rows. The data was stored in Hadoop distributed file system (HDFS) configured on the cloud cluster. Each row in the dataset represents an observed customer and each column represents attribute of that customer. Dataset was loaded into Spark DataFrame [**?**] along with the specified schema. After the dataframe was instantiated, it was queried using SQL queries with the help Pyspark DataFrame API.

5.2. Feature Engineering

Every single information we use to represent a customer is called a "feature" and the activity of finding useful features is called "feature engineering." In the Customer data set the data is labeled with two classes – Yes (Churned) and 0 (Not Churned). The features for each customer consist of:

Labels–> Churn: Yes or No

Features–> customerID,gender,SeniorCitizen,Partner, Dependents, PhoneService,MultipleLines,InternetService, OnlineSecurity, OnlineBackup,DeviceProtection,TechSupport, StreamingTV, StreamingMovies,Contract,PaperlessBilling, PaymentMethod, MonthlyCharges,TotalCharges

5.3. Define Feature Array

In order for the features to be used by a machine learning algorithm (Spark ML), It is useful for combining raw features and features into a single feature vector, which are vectors of numbers representing the value for each feature. Next, VectorAssembler[**?**] is used to transform and return a new dataframe with all of the feature columns in a vector column. Finally we use this dataframe to train our model.

5.4. Model Training

Fig **??** below shows the workflow used in training our machine learning algorithm.

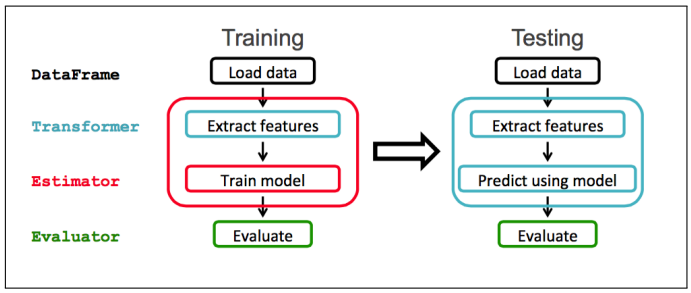

Fig. 2. Spark ML Workflow [**?**]

We used the following Machine Learning Algorithm and compared them using metrics like accuracy, AUROC and confusion matrix.

1. Random Forest
2. Decision Tree
3. Support Vector Machine

For each of the algorithms, the data was split into training and testing subsets with a ratio of 70 percent for training and 30 percent for testing.The predictive performance of each of the algorithm was evaluated by comparing predictions on the testing data set with true values (known as ground truth) using a variety of metrics such as Accuracy, AUROC and Confusion Matrix.

6. MODEL EVALUATION

We used Spark ML's inbuilt classification evaluators to evaluate the predictions of our algorithms. Table **??** below shows the accuracy for each of the algorithm used.

Classification Algorithm	*Accuracy (%)*
Decision Tree	83.75
Support Vector Machine	86.82
Random Forest	84.02

Table 3. Predictive Accuracy

7. DEPLOYMENT

The project deployment scripts were designed to be simple and reproducible. The deployment consists of Ansible playbooks and Cloudmesh Client scripts. It is run on user's local machine(Ubuntu 16.04). Deployment scripts will install all the necessary software along with project codes onto the cluster nodes, run spark job on the cloud and finally fetch result on the users local machine. The following section describes how to use the deployment scripts to install and run the project in the cloud.

7.1. Requirements

In order to run the project deployment scripts, one must have Ansible and Cloudmesh Client configured and installed on their local machine.

1. Cloudmesh Client: In our project Cloudmesh Client is used to set up Hadoop virtual cluster with Spark on two cloud environment. In order to run a virtual machine in the cloud, /.cloudmesh/cloudmesh.yaml configuration file should be configured to fill in detail about our cloud accounts such as username, password, project name, etc. Following tasks are performed in our project using Cloudmesh Client:

   (a) Uploading Public Key- using Cloudmesh's key add and upload commands.

   ```
   $ cm key add --ssh
   $ cm key upload
   ```

   (b) Uploading Security Rules- using Cloudmesh's secgroup commands.

   ```
   $ cm secgroup upload
   ```

   (c) Creating Hadooop/Spark Cluster- using Cloudmesh's deploy command.

   ```
   $ cm cluster define --count 3
    --image CC-Ubuntu14.04
   $ cm hadoop define spark
   $ cm hadoop sync
   $ cm hadoop deploy
   ```

2. Ansible Scripts: In our project, the entire cloud interaction including spark application installation, uploading data, setting up environment, running the spark application, and fetching the result is performed via Ansible playbooks. Ansible script was used in the following way:

   (a) `Local Configuration` : In the host file the IP address of remote namenode of the cluster is specified as follows:

   ```
   [predict]
   129.114.33.144 ansible_ssh_user=cc
   ```

   (b) `predict_setup.yaml` : sets up environment, copies files/code and installs dependencies on the namenode of the cluster.

   (c) `predict_execute.yaml` : This moves the downloaded data to HDFS through the following command:

   ```
   $ hdfs dfs  -put <source> <hdfs-folderpath>
   ```

   This HDFS file will serve as an input to our spark application. Finally the spark application is submitted to the namenode, which runs it on distributed framework and fetches result from remote to local after job completion.

8. BENCHMARKING

After the Hadoop Spark cluster is set up on the cloud, a benchmarking process is run to asses the performance of our deployment and Spark job on the two clouds. Figure **??** below compares the time set up environment, install dependencies, run application and fetch result from both the clouds. For measuring the time we used shell script to calculate the time elapsed between begin and completion of Ansible playbooks.

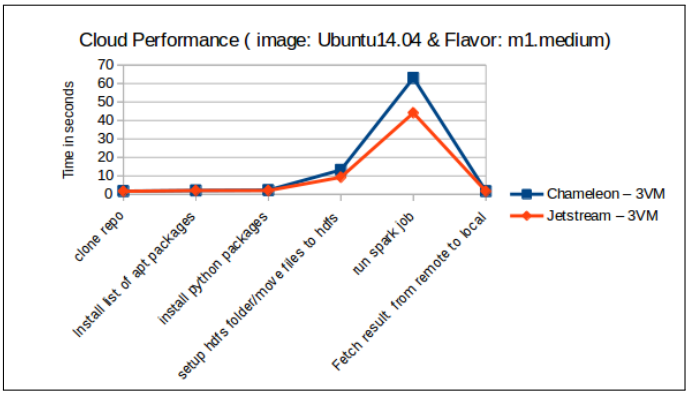

Fig. 3. Time Comparison in seconds

First time installation for the dependencies and apt-packages takes considerable amount of time(10-15 minutes) and hence are not included in the above comparison graphs. Specifically, Spark application performs 23 percent faster on 3 node Jetstream cluster when compared to similar cluster on chameleon.

9. CONCLUSION

We have built and tested a fully automated program to configure, deploy and run Apache Spark's Machine Learning algorithm for customer churn prediction on two cloud environments(Chameleon and Jetstream). The use of Ansible Galaxy to automate deployment and Cloudmesh Client to setup Virtual Cluster on cloud helped in reproducibility and scalability.

10. ACKNOWLEDGMENT

The authors would like to thank Gregor von Laszewski and course TA's for their technical support and guidance.

11. WORK BREAKDOWN

The work in this project was equally distributed between the following authors.

Diksha Yadav: Data analysis and building Spark pipeline for machine learning algorithm in stand alone mode, benchmarking and troubleshooting.

Yatin Sharma: Code deployment, building ansible scripts, benchmarking and troubleshooting.