

Abstract

In this paper we discuss my final project for CSIT 565. For my final project, I intended to create an end-to-end example of data entering a data lake and then that data being fed into a machine learning model automatically and have it make predictions based on that input. I was not able to achieve this as the end result. However, we do explore the different services required to achieve this. And we walk through an example of how to create a machine learning model on Amazon Redshift.

Introduction

With the advent of cell phones and social media nearly all our actions are tracked and monitored. And with the growing field of “internet of things” technology, even more of our day-to-day actions will be tracked. All these devices produce data, and all this data can be used to better tailor our digital experiences. An issue that occurs when storing all this data is the ability to curate it and fit it into an existing structure that is a data warehouse. The solution to that is to use data lakes. Data lakes allow for the data to be stored in raw format and then translated to whatever schema is needed on read. This ability to pull only the needed data out, is particularly useful when applying it to machine learning. Add in automatically updating and additional fitting of learning models, we can create an application that is taking in data from various sources on the fly and updating a learned model as it goes.

The goal of this project is to explore this pipelining and machine learning capabilities on the AWS platform. Specifically, using Amazon Redshift, Amazon S3, Amazon SageMaker, AWS Glue, and AWS Data Lake Formation. The stretch goal is to make an end-to-end example of data lake to machine learning model prediction application.

Amazon Redshift Overview

Amazon Redshift is Amazon’s cloud-based data warehousing platform. Like many data warehousing platforms, it consists of clusters which are used to service the data queries of the user. Each cluster is made up of compute nodes. Clusters which have more than one compute node, have a leader node which is defined. The leader node coordinates the compute nodes and acts as the main interface to the client. The data itself is stored in a separate storage tier known as Redshift Managed Storage. This separation allows for scaling of storage space without interrupting the service. Data is stored in a columnar format which drastically reduces the on-disk I/O and reduces the amount of data you need to load from the disk. Because each column holds the same type of data, compression can be used to further decrease disk I/O. The standard block size used by Redshift is 1 MB.

It is important to note that the Redshift databases are structured and optimized to analyze relational data. The use we will be discussing in this paper will be an interface between a data lake stored on Amazon S3 and running a ML operation on Redshift.

Amazon S3 Overview

Amazon Simple Storage Service (Amazon S3) is a cloud object storage service. This provides the base storage location for building a data lake. It integrates into AWS Glue, Redshift and SageMaker and uses common policies for access.

AWS Data Lake Formation

A data lake is a modern data management strategy and they have become popular due to their ability to store large amounts of data at a low cost. They allow data to be stored in its original raw form without minimal preprocessing or modifications. This means both unstructured and structured data can be stored with no predefined schema. As compared to a data warehouse which requires all its data to be highly and its schema defined on implementation. However, with a data lake, as is with any data collection, it is important to have good data-quality and good data-governance otherwise it will become unmanageable.

AWS Data Lake Formation has the ability to clean, deduplicate and reformat data as it is input to the data lake. It also can encrypt and apply access controls to the data.

Upon data being added to a data lake, it needs to be cataloged and labeled using metadata. AWS Glue uses “Crawlers” to catalog the data and that allows users to discover what datasets are available and speeds any queries run directly on the data.

AWS Glue

AWS Glue acts as the go between for the Redshift and the data lake that is stored in Amazon S3. It has the ability to trigger and automatically transfer data between services. Upon the transfer of data between services it can apply filters and reformatting rules to the data to allow to fit into a given database table. This works in reverse as well, if data is no longer useful in a Redshift database it can be moved and stored into the data lake. This saves space will allow for overhead in the redshift database.

Amazon Redshift ML with Amazon SageMaker

Amazon Redshift has built in the ability to create SQL objects that are machine learning (ML) models. Once the appropriate policies have been set in place on the database and the other services, it is possible to pull in the data and load it to a table in the redshift database and then train a ML model on that table. The ML tools within Redshift are directly tied to SageMaker.

The example I will discuss in this paper is based on the churn tutorial in the redshift developers guide. This example does not use the other aspects of the pipeline discussed above but is meant to show the ML capabilities of Redshift itself.

The first step is to build a table which will be used to represent the incoming data and then copy the incoming data to it. This will turn any non-structured data into a structured dataset.

```
/*Create dist table*/
DROP TABLE IF EXISTS customer_activity;

CREATE TABLE customer_activity (
  state varchar(2),
  account_length int,
  area_code int,
  phone varchar(8),
  intl_plan varchar(3),
  vMail_plan varchar(3),
  vMail_message int,
  day_mins float,
  day_calls int,
  day_charge float,
  total_charge float,
  eve_mins float,
  eve_calls int,
  eve_charge float,
  night_mins float,
  night_calls int,
  night_charge float,
  intl_mins float,
  intl_calls int,
  intl_charge float,
  cust_serv_calls int,
  churn varchar(6),
  record_date date
);
```

```
/*Copy CSV file into table dist table*/
COPY customer_activity
FROM 's3://redshift-downloads/redshift-ml/customer_activity/'
REGION 'us-east-1' IAM_ROLE default
FORMAT AS CSV IGNOREHEADER 1;
```

Next, we must create the model to be trained. This involves selecting the data from the base table and using it to define the feature set of the model. Notice how some of the features here are derived from the original data and not saved directly into the table. This model will create features 'average_daily_spend' from 'total_charge / account_length' and 'average_daily_cases' from 'cust_serv_calls / account_length'. The target of this model will be the "churn" value in the table, which is a 1 if churned and 0 is not churned. We will use all the data prior to 1-1-2020 to train. It will use the defined 'ml_fn_customer_churn_auto' function which is defined in Amazon SageMaker to build the model. Once the model is created, we will have a SQL object which can be referenced to apply and make predictions. The SQL model will be stored in the S3 bucket we created for this, called 'ml-output-buc'. We will be able to reference the model using the name we created, 'customer_churn_auto_model'.

```
CREATE MODEL customer_churn_auto_model
FROM
(
  SELECT state,
         account_length,
         area_code,
         total_charge/account_length AS average_daily_spend,
         cust_serv_calls/account_length AS average_daily_cases,
         churn
  FROM customer_activity
  WHERE record_date < '2020-01-01'
)
TARGET churn FUNCTION ml_fn_customer_churn_auto
IAM_ROLE default SETTINGS (
  S3_BUCKET 'ml-output-buc'
);
```

The model building process can be a lengthy process. This model took a little over an hour to be created, and cost about \$10 at the current rate AWS charges for the use of SageMaker services. We can check on the model by simply querying the model status via the 'SHOW MODEL customer_churn_auto_model;' query. This will return a table which give you details about the model. An example output is shown below.

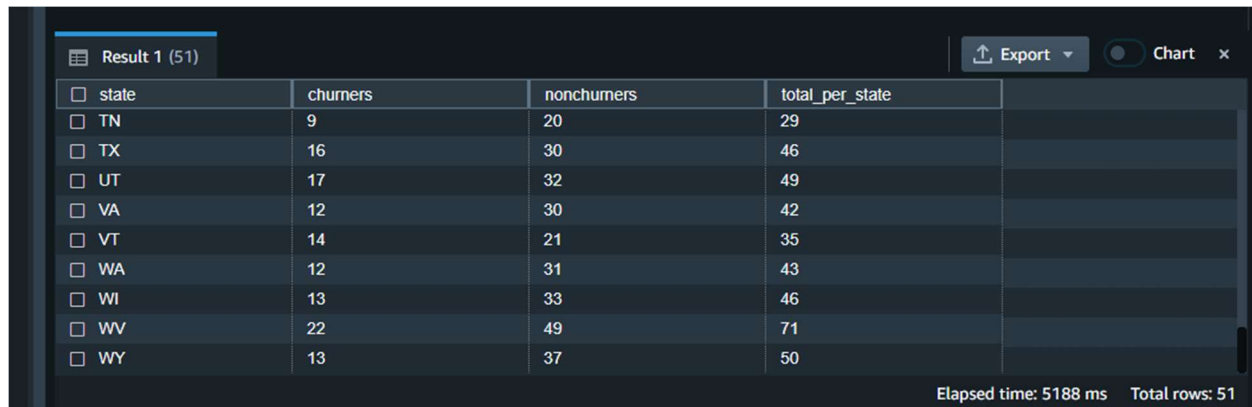
```
Key      Value
Model Name      customer_churn_auto_model
Schema Name     public
Owner           awsuser
Creation Time    Wed, 07.12.2022 04:06:38
Model State     READY
Validation:binary_f_beta0.368351
Estimated Cost  10.436314

TRAINING DATA:
Query: SELECT STATE, ACCOUNT_LENGTH, AREA_CODE, TOTAL_CHARGE/ACCOUNT_LENGTH AS AVERAGE_DAILY_SPEND, CUST_SERV_CALLS/ACCOUNT_LENGTH AS AVERAGE_DAILY_CASES,
CHURN
      FROM CUSTOMER_ACTIVITY
      WHERE RECORD_DATE < '2020-01-01'
Target Column   CHURN

PARAMETERS:
Model Type      mlp
Problem Type    BinaryClassification
Objective       F1
AutoML Job Name redshiftml-20221207040638573674
Function Name   ml_fn_customer_churn_auto
               ml_fn_customer_churn_auto_prob
Function Parameters state account_length area_code average_daily_spend average_daily_cases
Function Parameter Types varcharint4 int4 float8 int4
IAM Role        default-aws-iam-role
S3 Bucket       ml-output-buc
Max Executions  5400
```

Now that a model is created, we can run predictions against it. For this we will use the data after 1-1-2020 to check the see how the model does with its predictions.

```
WITH predicted AS (
  SELECT
    state,
    ml_fn_customer_churn_auto(
      state,
      account_length,
      area_code,
      total_charge / account_length,
      cust_serv_calls / account_length
    ) :: varchar(6) AS active
  FROM
    customer_activity
  WHERE
    record_date > '2020-01-01'
)
SELECT
  state,
  SUM(
    CASE
      WHEN active = 'True.' THEN 1
      ELSE 0
    END
  ) AS churners,
  SUM(
    CASE
      WHEN active = 'False.' THEN 1
      ELSE 0
    END
  ) AS nonchurners,
  COUNT(*) AS total_per_state
FROM
  predicted
GROUP BY
  state
ORDER BY
  state;
```



The screenshot shows a Redshift query result titled "Result 1 (51)". It displays a table with four columns: "state", "churners", "nonchurners", and "total_per_state". The table lists data for ten states: TN, TX, UT, VA, VT, WA, WI, WV, and WY. Each row includes a checkbox to the left of the state name. The "total_per_state" column shows the sum of churners and nonchurners for each state. At the bottom right, it indicates "Elapsed time: 5188 ms" and "Total rows: 51".

<input type="checkbox"/> state	churners	nonchurners	total_per_state
<input type="checkbox"/> TN	9	20	29
<input type="checkbox"/> TX	16	30	46
<input type="checkbox"/> UT	17	32	49
<input type="checkbox"/> VA	12	30	42
<input type="checkbox"/> VT	14	21	35
<input type="checkbox"/> WA	12	31	43
<input type="checkbox"/> WI	13	33	46
<input type="checkbox"/> WV	22	49	71
<input type="checkbox"/> WY	13	37	50

This prediction is organized by state and estimates the number of churns and non-churns that will occur. Due to the time I did not actually check the accuracy of the model, which in hindsight I should've taken the time to check.

Redshift Machine Learning integration offers a number of different ML options including Remote inference, K-mean clustering, Multi-class classification, and Various regression models. The model creation has other options which were not explored here, such as setting the objective (which effects the error metrics in training) and configuring hyperparameters (which effects model specific factors).

Lessons Learned

During this project I hit several roadblocks. The main of which was setting up the common role among the AWS services. It was not discussed in this paper as it was not a central theme to the project itself but ended up taking a large amount of time to address and get correct. Another issue I ran into was simply the amount of time as compared to the scope of the project. Although I had a concrete goal, learning to configure of each of the services is time consuming and I was not able to complete everything.

Summary

It is easy to see how these services can be combined in an application that will take data from several sources into a data lake and then transfer the data using AWS Glue to Redshift. Then Redshift can apply a machine learning model to make additional predictions about the data or further refine the model. Machine learning and big data have transformed how the world is shown to us; from the movies we are given as suggestions, to ads we're given, to the news updates we are shown. The integration of these model and the ability to train them have become seamless to the point that the users do not need to even understand what is really driving their creation. As this trend continues and the interface of machine learning to big data simplifies further, I expect even more of our everyday lives to be customized to individual preferences, even if those preferences are not ones, we knew we had.

References

- Redshift Developers Guide

<https://docs.aws.amazon.com/redshift/latest/dg/welcome.html>

- Amazon Solutions

<https://aws.amazon.com/solutions/implementations/data-lake-solution/>

- AWS Dojo

<https://aws-dojo.com/>