COMP 4651 Cloud Computing and Big Data Systems - Project Report

Bitcoin price prediction for 1 hour based on 90 days of hourly data

Group Members: Lalwani, Monesh Govind (20598730), Singh, Jaspreet Dhaliwal (20600856),Kotecha, Kinshuk (20557138), Khan, Umer Shehzad (20598687)
Supervisor: WANG, Wei

Abstract

This project aims to predict the Bitcoin (BTC) price prediction for one hour with data of 90 days of hourly data provided from Coinbase. We apply simple machine learning techniques on the data we scrapped and stored on the AWS cloud using APIs and cloud connections to predict the value of BTC based on its historical data. The results show that there is a 73% accuracy in predicting 1 hour of BTC value by performing Support Vector Regression (SVR) technique and is displayed on a web application using EC2.

Introduction

Understanding the stock market and the cryptocurrency market has become more and more in demand. To know more about the markets and specifically the crypto market, we can infer information from countless sorts of data. Cryptocurrency can be monitored using several indicators such as Bollinger Bands (BB), Relative Strength Index (RSI), Moving Average Convergence-Divergence Indicator (MACD). In our project, we take a simpler approach by analyzing the historical data of the prices of BTC to predict the future value and focus more on the cloud infrastructure to maintain smooth workflow and accurate prediction. With the usage of APIs and AWS cloud connection, large amount of data can be scrapped and used as the dataset and maintain backend communication for building a pipeline respectively.

In most cases of cryptocurrency analysis, the data is fed in once to train the model and that same historical data of a period is used for prediction. This will affect the accuracy of prediction because the most recent trend of the crypto is not taken into consideration. To alleviate this problem, this project aims to:

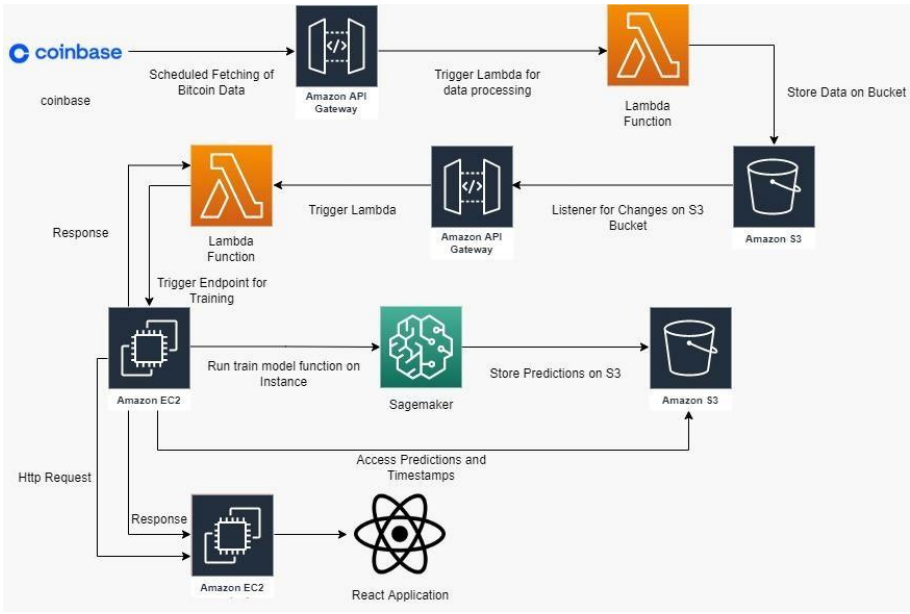
1) Scrap data hourly for new data to keep the model training up to date
2) Build a bitcoin prediction model to predict for the next hour
3) Build a frontend display for the prediction
4) Connect the backend to the frontend for full communication

Division of Work

There are four members in this project. Singh Jaspreet oversees the data scraping, uploading, and updating the S3 bucket with Lambda and EventBridge. Kotecha Kinshuk handles the lambda for fetching data from S3 and sending it to EC2 and build the frontend for prediction display. Khan Umer is responsible for fetching constant data from backend and building the prediction. Lalwani Monesh’s job is to build and maintain the backend communication with EC2, S3 Bucket, and SageMaker.

Project Infrastructure

To efficiently understand our project infrastructure, the infrastructure workflow is shown in the figure below.



Data Management

Scraping Historical Data on Local Workspace

To train the Bitcoin prediction model, a continuous stream of data is required at regular intervals. The current implementation is to provide new data hourly at 5-minute intervals. To scrape data, there are a lot of pre-existing free APIs. However, all of them have imposed limitations to restrict free usages. Currently, REST API from CoinAPI.com is used to provide data due to the data being formatted at regular intervals. Attached below is the JSON schema for one record. The data appropriately divides the time into regular five-minute intervals.


```
{"time_period_start": "2021-09-06T00:00:00.0000000Z",
  "time_period_end": "2021-09-06T00:05:00.0000000Z",
    "time_open": "2021-09-06T00:01:00.0000000Z",
   "time_close": "2021-09-06T00:04:00.0000000Z",
         "rate_open": 51796.17511350881,
         "rate_high": 51796.17511350881,
          "rate_low": 51725.54974283357,
        "rate_close": 51792.22516195528},
```


Extracting 90 days data

In this stage, the 90 days data is fetched and stored using 3 API calls from CoinAPI instead of 1 due to API restriction limits. As a result, monthly data at 5-minute intervals is collected and stored in “data1.json”, “data2.json” and “data3.json” respectively. The three json files are aggregated in a “finaldata.json” file that prevents duplicates and is ordered according to entries. The API calls are stored in “coinapi.py” while the aggregation is done by “jsoncombiner.py” in the “scraping” folder.

Update S3 Bucket with Lambda Integration

Storing Historical Data in S3 Bucket

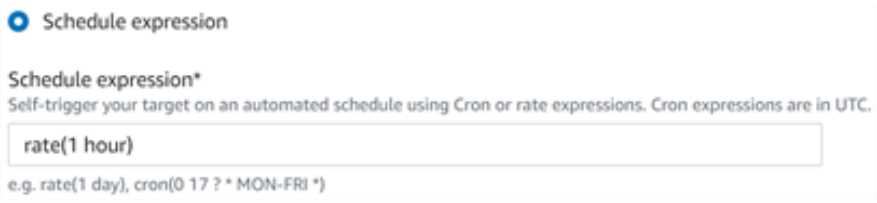
After scraping, a bucket called “comp4651-project-rawdata” is created which will store “finaldata. json”. The file is manually uploaded to s3 at this stage and will be updated hourly using a lambda function. A Lambda trigger is set up to send data for training when updates are received. The main purpose of the s3 is to act as a raw delta lake that stores all the data.

Name	Type	Last modified	Size	Storage class
finaldata.json	json	December 5, 2021, 00:45:04 (UTC+08:00)	2.0 MB	Standard

Lambda Integration

Lambda Trigger:

A lambda function is created that should execute hourly. The hourly execution is done using EventBridge, in which a new rule to schedule execution hourly is added. The code is located in the “get_data_hourly.py” file located in the “Lambda_Functions” folder.



Data Integrity and Fault Tolerance:

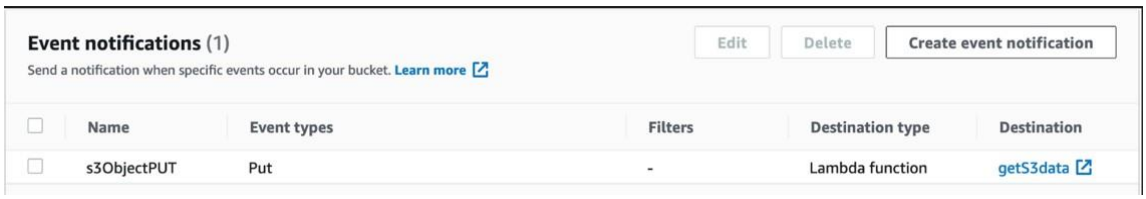
To ensure fault tolerance, the API will collect two hours’ worth of Bitcoin prices upon each lambda trigger, this is to ensure that if the HTTP request fails in one execution of Lambda, the next iteration of Lambda will fetch the data. The algorithm also prevents duplicates and makes sure each entry is present at regular 5-minute intervals. To verify the claim, a Python script called “check.py” in the “scraping “folder is run.

Posting Updated Files with Lambda

Another Lambda function is set up to send data via HTTP Post requests once there is an update in files. To do that, an event trigger is setup in the s3 bucket policy that triggers the call of the lambda function.

S3 Event Notifications

A notification is setup in the s3 policy to execute the lambda function “gets3data” once there is a notification of any file upload (update in our case)



Posting Updated Contents to URL endpoint

In the lambda function, the contents of the file are loaded using get_bucket() method and a post request is sent to a URL which trains the model ("http://54.210.41.176:8080/coinRec")

Bitcoin Prediction Model on SageMaker

Data pre-processing

The data fetched by the prediction model is in the form of a JSON file which needs to be pre-processed. The data has a "prices" column with each row containing a list of two values: the UNIX timestamp and the corresponding BTC price. To process the data, we create two Numpy arrays, one containing all the UNIX timestamps, and the other containing the values.

Model Training and Prediction

With the clean data, we add a prediction column which contains the prediction values of each of the prices. The predicted data is formed by shifting the price column up by 12 intervals which represents each price mapping to 12 values under it (one hour interval in total) as its predictedprice. Due to this, the last 12 values of the prediction column become NaN and will not be pushed into the model for training. After that, the sklearn library is used to import train_test_split package to create training and testing data with 80:20 split.

We trained the model using the sklearn.svm library by importing Support Vector Regression (SVR) package. SVR is used because it can perform regression to find similarity between the historical data sequentially. Radial Basis Function (RBF) kernel was used to create and train the SVR which is also similar to how K-NN Algorithm works. By configuring the parameters, we can get the confidence of the SVR RBF which acts as the accuracy also. Our model had an accuracy of 72.787% which is acceptable for our case. The final values for our prediction are stored in the interval_prediction (predicted prices) variable along with just_time_stamp_prediction_arr (UNIX timestamps).

Backend Communication

Our main framework for our backend communication lies with a flask application that is hosted on an Elastic Compute Cloud (EC2), the instance is of type m4.xlarge. The application is responsible for handling the communication within their entire project and server different endpoints for getting data.

Setting up of Services

Firstly, the EC2 hosts the application on an open port for easy access between different services. It is continuously run on the service through a terminal multiplexer package which essentially acts as GNU screen. This ensures that the process from the multiplexer always runs to keep the flask application open. The instance also makes use of the boto3 package for communicating with different services such as placing and updating objects on an S3 bucket.

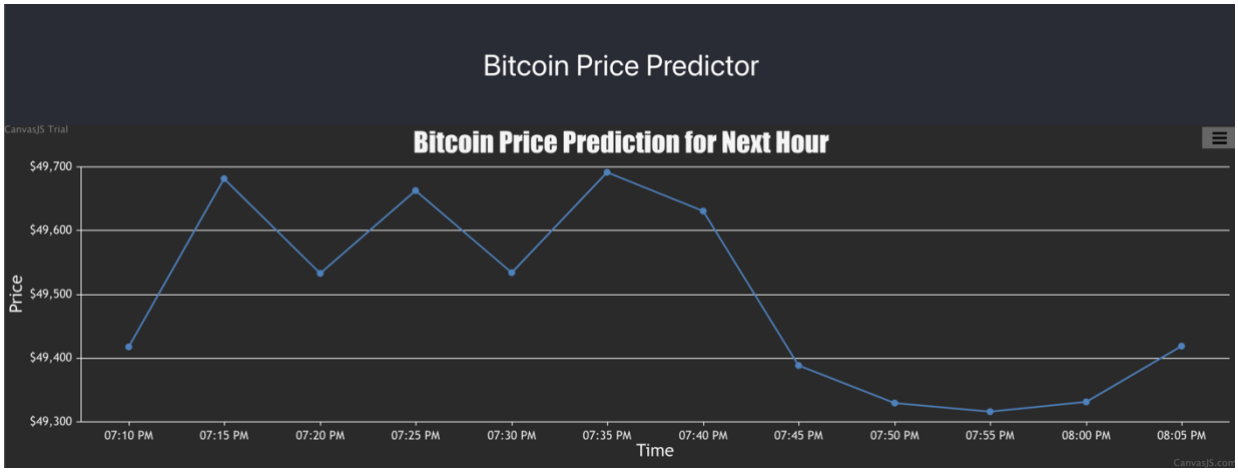
Main Usages of the Backend

Within the application, there are 2 main endpoints that is callable from the different services. The first one relates to the receiving of newly uploaded data. This type of request stems from a post request containing the latest 90-day worth of information that is recently updated on our bucket and is sent for training. The endpoint then sends the data through the training function on the instance that was pre-trained on SageMaker. The train function returns the correct new timestamps for the following hour alongside the predicted value and is stored on another bucket. This bucket stores the files for the newly predicted value for easy access for later usages.

The second endpoint serves as the communication endpoint between the frontend application and the flask app. When the react application sends a post/get request to the flask app, it triggers the endpoint for sending data, then the application will fetch the latest 12 5-minute data points on the bucket utilizing boto3 again alongside its timestamps and send it back to the frontend for processing.

Frontend Display

To display our results, we created a Single Page Application using React. This application was deployed using EC2. The application communicates with our EC2 instance and gets a list of values that our model predicted. These values are then used to plot a line graph to better visualize the results. The graph indicates the predicted price of Bitcoin for the next hour.



Appendix

GitHub Repository: https://github.com/hkust-comp4651-21f/project-bitcoinpriceprediction