

COURSE

**CSYE7374 54713 Parallel Machine Learning & AI SEC 02 Summer 2020**

Final Project Report

**Detecting Unknown Mining Pools and their Behaviour within Bitcoin using Parallelized Machine Learning**

INSTRUCTOR

Dr. Handan Liu

PRESENTERS

Srinjoy Chakravarty

(NUID 001086186)

Srishti Ashok Mishra

(NUID 001305178)

Contents

[Contents 2](#_Toc47907631)

[Introduction 3](#_Toc47907632)

[Background 3](#_Toc47907633)

[Objective 3](#_Toc47907634)

[Motivation 3](#_Toc47907635)

[Problem Analysis and Algorithms 4](#_Toc47907636)

[Description of Dataset 4](#_Toc47907637)

[Data Extraction and Transformation (ETL) 4](#_Toc47907638)

[Concepts and Definitions 6](#_Toc47907639)

[Benchmarking Disk Read/Write Speed with Pandas Binary Formats 6](#_Toc47907640)

[Network Analysis: Bitcoin blockchain data 13](#_Toc47907643)

[Serial Model Comparison – ANN vs RF vs NB 17](#_Toc47907646)

[Artificial Neural Network 17](#_Toc47907647)

[Naïve Bayes 19](#_Toc47907650)

[Model Training Parallelization 21](#_Toc47907652)

[Conclusion: 24](#_Toc47907656)

# Introduction

#### What is Blockchain?

A blockchain is a time-stamped series of an immutable record of data that is managed by a cluster of computers around the world and it not owned by any single entity. Each of these blocks of data are secured and bound to each other using cryptographic principles hashing. In simple terms, blockchain is a chain of blocks; the words “block” and “chain” are the digital information and public database, respectively. The core components of blockchain architecture are:

**Node** — computing ***peer*** within the ***network***

**Transaction** — immutable ledger ***record*** indicating a ***transfer*** of ***value***

**Block** — *data-structure* used to batch transactions and ***relay*** them ***peer-to-peer***

**Chain** — a ***sequence*** of blocks in ***chronological*** order

**Cryptocurrency** — algorithmic value token generated outside central bank or governmental purview

**Miners** — powerful nodes which create new monetary supply

**Consensus** — decentralized code-driven governance to validate all network activity

Machine learning algorithms when applied on big data, allow us to analyze the actual extent of ‘cybersecurity’ behind the ‘proof-of-work’ generated by various entities, powering the distributed ledger technologies, that claim to be pseudonymous and decentralized in nature

# Background

By analyzing the different characteristics of bitcoin transactions and identifying the behavior of publicly-known Bitcoin pools in choosing which transactions to mine, we build a Machine Learning Classifier that is able to predict whether a transaction is mined by a mining pool or not

# Objective

By classifying address signatures, we can identify mining entities and their potentially malicious behavior on the network to help the community make more responsible mining decisions.

# Motivation

Imagine if Nakamoto had moved one of his coins (he hasn’t moved his coins in years) and our classifier flagged it to have mined by an Unknown Mining Pool. Our deep-dive analysis would certain be useful.

# Problem Analysis and Algorithms

We use a combination of different Classification Algorithms and an Artificial Neural Network (ANN) to train our prediction models.

1. Serial Model 1 = Random Forest (SKLearn)

* We check the accuracy of our classifier by plotting a Precision-Recall Curve and calculating the area under it
* We use a Confusion Matrix to list out false positives and false negatives
* These transactions are then further investigated with network analysis

2. Serial Model 2 = Gaussian Naive Bayes (SKLearn)

3. Serial AI Model = Artificial Neural Net (TensorFlow)

4. Parallelized ML Model = Multi-Node Stacked Classifier over Distributed Memory

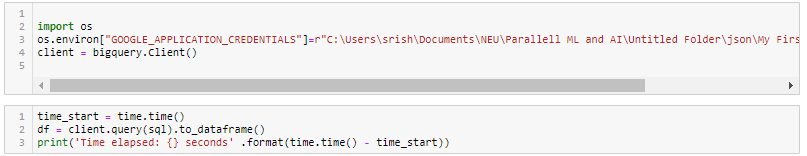
*(using Open-source MPI)*

# Description of Dataset

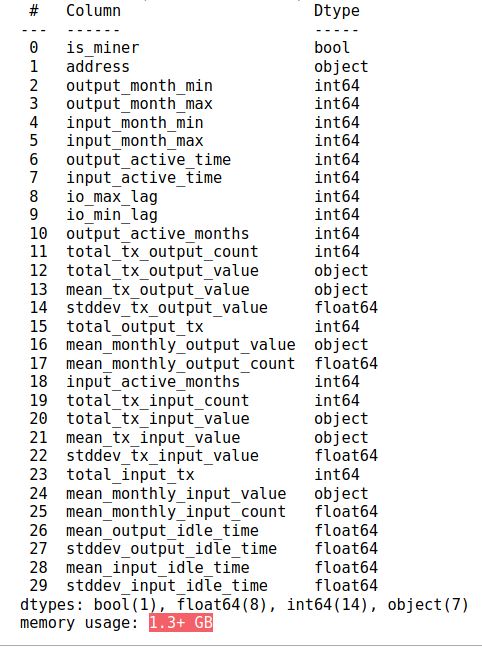
* We use Google BigQuery to dynamically query the live incoming blocks of the Bitcoin Blockchain using their ETL Dataset API. The API returns two tables: **Transactions** and **Blocks** which contain a host of features including *block\_id, is\_miner, address, transaction\_id, timestamp, input, output,* (updated every 10 minutes i.e. Average Bitcoin Block Time)
  + **Link:** https://tinyurl.com/yyalnexb

# Data Extraction and Transformation (ETL)

* The data is extracted by connecting to Google BigQuery by linking a service account key (environment variable) to a JSON keystore file which is obtained from the GCP console
* Google BigQuery allows the use of API client python libraries including the functions you need to connect your Jupyter Notebook to the BigQuery : sudo pip3 install google-cloud-bigquery
* We set the environment variable called “GOOGLE\_APPLICATION\_CREDENTIALS” to point our Notebook to BigQuery’s service account key
* There are two main tables we utilized as our primary schema: a ***“blocks”*** table and a ***“transactions”*** table, which were uniquely combined with SQL queries.
* SQL queries extract, transform, load bitcoin mining pool address signatures and provide statistics of their behaviour over time ¹
* Connecting to client :



* Extraction joins were from **bigquery-public-data.crypto\_bitcoin.transactions** and **bigquery-public-data.bitcoin\_blockchain.blocks**
* Every time our code is run, the amount of extracted data is about 1.3 GB+ with 30 columns *(out of which we use 26 as features x1, x2, x3….xn)*



* To preprocess and clean the data, we drop the columns with null values, and then split the data into training and testing sets to train the model. We choose a training to testing ratio of 80:20 to optimize for accuracy without the dangers of overfitting.

# Concepts and Definitions

* **Pool:** A pool is a platform with specialized software where miners combine the computational power of their equipment for more efficient mining of a certain cryptocurrency.
* **Mining Pool:** A pool for mining is an association of miners, who are seeking for best reward. The computing power of each participant’ device makes up the total hashrate of the pool. Such cooperation gives much more chances to find a block and get a reward, which is distributed among the participants according to the system accepted by the operator of the particular pool for mining.

*In simpler terms - A mining pool can be compared to a lottery pool. Your chances of winning the popular lottery are very low, but when you team up with a lot of other people and agree to share the money you win if you succeed, your chances are multiplied by the number of participants.*

* **Pooled Mining:** Pooled mining, "pools" all of the resources of the clients in that pool to generate the solution to a given block. When the pool solves a block, the 6.25 Bitcoin block reward (approximately $73,588 block reward) is split and distributed amongst the pools participants.
* **Solo Mining:** Solo mining is when a miner performs the mining operations alone without joining a pool. All mined blocks are generated to the miner's credit.
* **UTXO** stands for the unspent output from bitcoin transactions. Each bitcoin transaction begins with coins used to balance the ledger. UTXOs are processed continuously and are responsible for beginning and ending each transaction. Confirmation of transaction results in the removal of spent coins from the UTXO database.

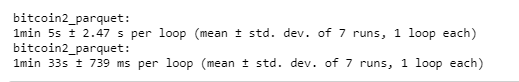
**Source:** <https://tinyurl.com/yxaq7vx6>

* Mempool is the node's holding area for all the pending transactions. It is the node's collection of all the unconfirmed transactions it has already seen enabling it to decide whether or not to relay a new transaction..

**Source:** https://tinyurl.com/yxg2skom

# Benchmarking Disk Read/Write Speed with Pandas Binary Formats

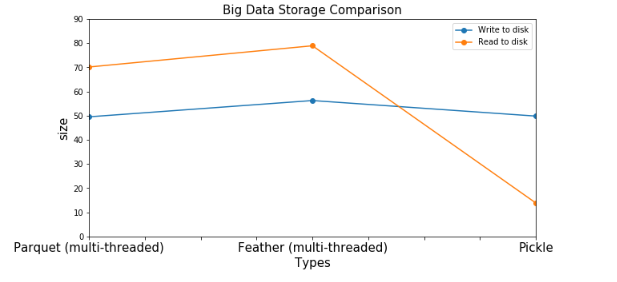
* When the number of observations in your dataset is high, the process of saving and loading data back into the memory becomes slower, and each kernel’s restart steals time and forces you to wait until the data reloads
* There are plenty of panda-supported binary formats to store the data on disk.
* Methods used are: Parquet, Feature, HDF5, Pickle
* Parquet : Parquet is a standard Storage format for analytics that is supported by AWS, BigQuery, Impala, Hive and Spark and is designed for long-term storage with layers of encoding and compression



* Feather: Feather is a binary data format for quick loading and transformation which enables faster I/O speeds and less memory and with which data frames can be exchanged between Python and R

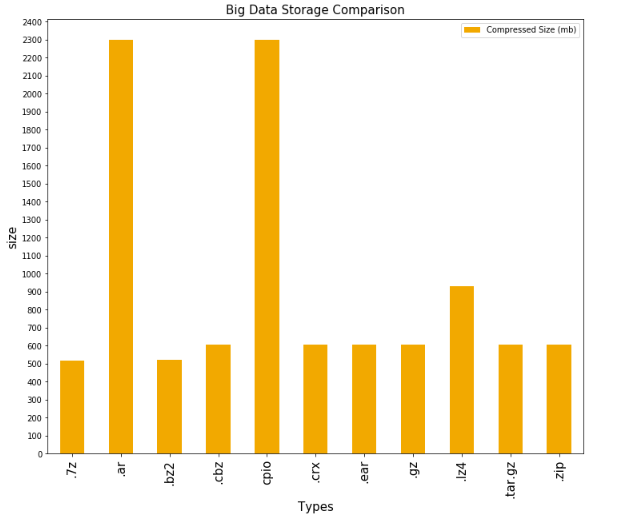


* Pickle : Pickles are primarily used in serialization and deserialization of Python object structures from byte streams to act as file or database storage, maintain program state across sessions, or transport data over the network

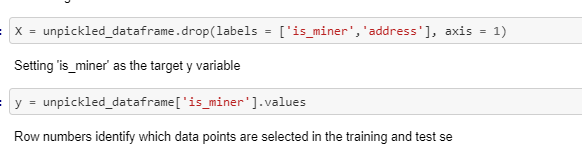


**Dataset Compression (Post - ETL)**

* We tried 27 different data compression techniques to minimize the size of our dataset (1.3+ GB ) and reduce network latency when passing data between local workstations, servers and high-performance cluster nodes. The file format with the Highest Compression Ratio was .7zip with Space Saving Score of 78.35%
* Space Saving = (Compressed Size) / (Uncompressed Size)
* 0.7835 = 1 - 519.3 / 2,982.4



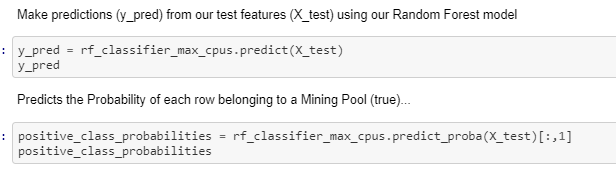
* Processing : Cleaning data to drop columns that contain features with null values
* Removing columns with non-numerical features from x variables



**Random Forest Hyper-parameters**

* Training the random forest classification model on the training set
* Running model jobs in parallel using all the processors available with n\_jobs
* Entropy chosen as criterion to measure quality of splits by information gain
* 200 trees used in random forest for sufficient accuracy

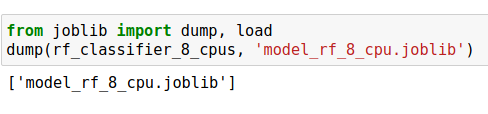
**Prediction and Probability**



**Optimizing Model Persistence (Pickle vs. JobLib)**

The idea behind any Data Science is to be able to build a model and throw away all the data.

We compared data write speeds of both Pickle vs JobLib (the palatalization library) behind sci-kit learning (written in Cython) and found that joblib was usually significantly faster on large numpy arrays because it has a special handling for the array buffers of the numpy data structure.



**Optimizing Random Forest Model Training with Multiple Cores (Memory vs Duration)**

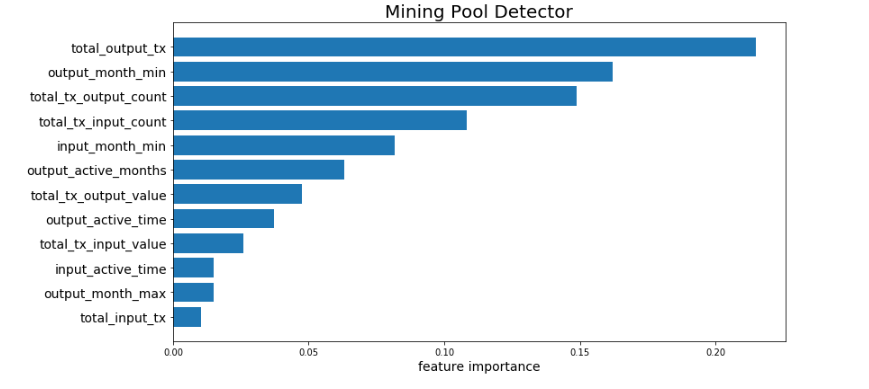
### 

### 

* To check how good our model has been trained, we focused mainly on the area below the Prevision-Recall Curve

**Best Mining Pool Indicator**

* Which features *(in the bitcoin transaction)* **best indicate** whether it will be mined by a publicly known mining pool (provide the most signal)?
* To understand which features, provide the most signals about the mining pool, we plot the top 12 features that are important according to the analysis



We found that the total amount of transaction outputs **(UTXO)** of a Bitcoin transaction, proved to be the heaviest contributor to whether a transaction would be picked up by a known Mining Pool from the **mempool**.

**Precision-Recall Trade-off**

***Objective: Predict the Probability of each row belonging to a Mining Pool***

* An ideal system with high precision and high recall will return many results, with all results labeled correctly
* A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels
* A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels

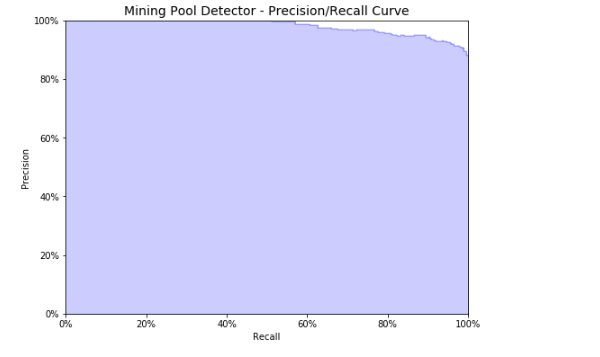
***matplotlib.pyplot.step* parameters:**

* The color = #f2a900 parameter ensures the plotline is bitcoin orange
* The alpha = 1 parameter makes the plotline 100% opaque vs. the area under the curve alpha = 0.4 i.e. 40% opaque
* The where = post parameter ensures y-values are continued constantly to the right from every x-position

*An ideal system with high precision and high recall will return many results, with all results labeled correctly*

**Source:** [**https://tinyurl.com/yxbjk26m**](https://tinyurl.com/yxbjk26m)

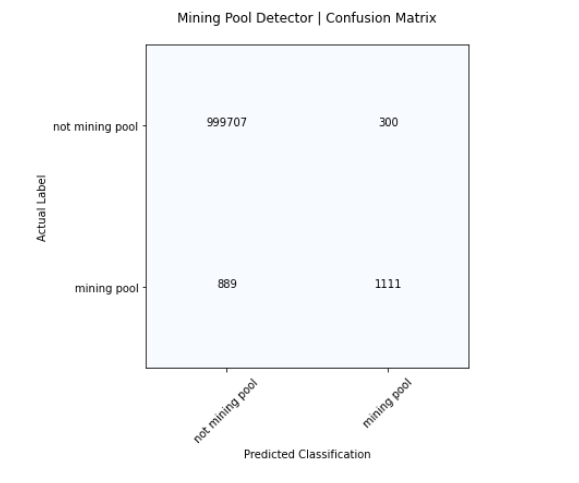
**Mining Pool Detector – Precision/Recall Curve**

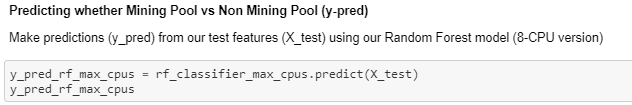


Our curve when applied on a heavily trained model should great Precision even at the highest end of the Recall spectrum.

**Confusion Matrix Computation**

A confusion matrix is a 2x2 table used to **visualize the learning performance** of a supervised classification algorithm (Random Forest) on a set of test data for which the true values are known





**Result:** The confusion matrix from our predictive model allows us to use **‘False Positives’** to identify transactions that have been mined by a **Shadow/ Dark** mining pool

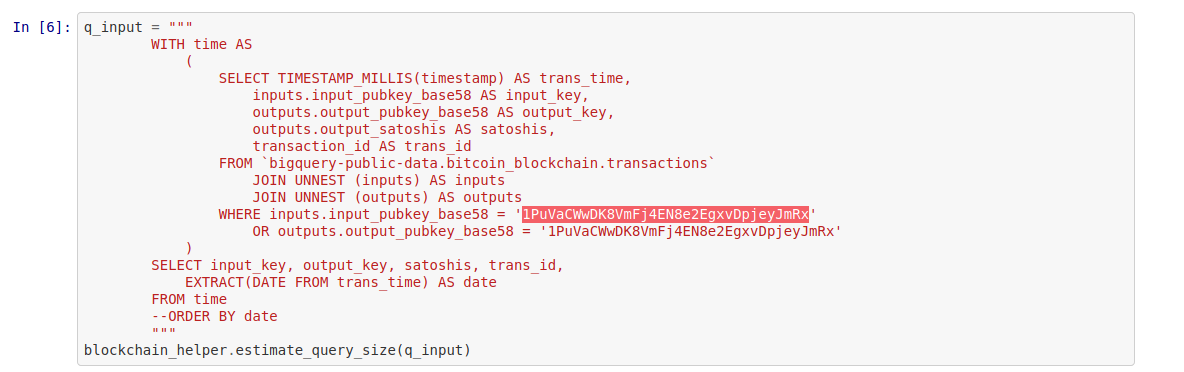
**Reasoning:**data points (transactions) where our model predicts **mined\_by\_mining\_pool == true**, but in actuality have been labeled by the network as **mined\_by\_mining\_pool == false**

*This poses the question about why this is? Why does the public not have information about these mining pools? Are they covert on purpose? Do they have nefarious network activity?*

**Objective:** identify **node** **addresses** flagged to be ‘unknown’ by our prediction classifiers (false-positive) and do further data analytics (node centrality metrics etc.) to unravel **malicious** network activity

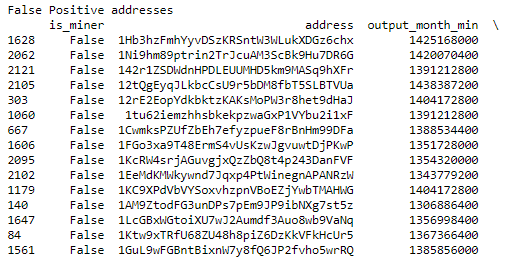
We decided to investigate more by taking a sample **False Positive** *node address* for our **network analysis** section below. The node address we decided to pick (from the list of unnknown mining pools that our ML classifier listed out) was: **1PuVaCWwDK8VmFj4EN8e2EgxvDpjeyJmRx**

So we went ahead and customized yet another SQL statement for Google BigQuery:



**Unknown Mining Pool Detection**

Our machine learning classifier generates the following list of Unknown Bitcoin Mining pool node addresses.



# Network Analysis: Bitcoin blockchain data

* We decided to adopt some great work\* on Bitcoin activity analysis to investigate our chosen unknown node address: **1PuVaCWwDK8VmFj4EN8e2EgxvDpjeyJmRx**

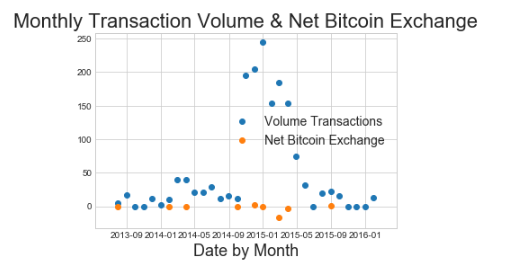
\* https://tinyurl.com/y3ms538b

* **Data cleaning and processing:** We first applied some data transformations, converting the satoshis used in the wallet of our chosen Node Address to bitcoin, and converting the sending out of bitcoins into negative values
* **Exploratory Data Analysis** includes, unique addresses that were included in the transaction history of our chosen Node wallet, the top addresses that our **Unknown Node wallet** interacted with, the transaction activity of our chosen **Unknown Mining Node** across time and plotting the Date vs Transaction graph to see the volume of transactions per day

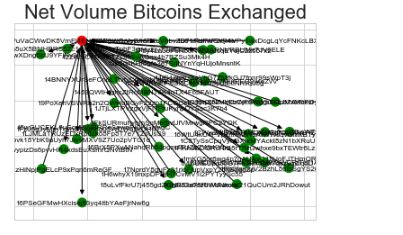
# 

# 

* Furthermore, we looked at the difference between input transactions and output transactions (which indicates cleaning of stained bitcoin by breaking them into chunks and harder to trace) and the total amount of bitcoins sent each month



We then used a Network X Graph to plot the top 20 wallets that our Unknown (Dark/ Shadow) Mining Node Address sent and received bitcoins from.



We noticed that amongst the top nodes our **Unknown Node Wallet** was sending Bitcoin to,

Address **1NxaBCFQwejSZbQfWcYNwgqML5wWoE3rK4** seemed to come out on top. We therefore decided to use Graph Theory to investigate this suspicious address further.

**Node Centrality Analysis**

* We decided to compute the degree centrality for the node address **1NxaBCFQwejSZbQfWcYNwgqML5wWoE3rK4**
* Also, see what fraction of nodes it is connected to (amongst the top interaction nodes with our original node address)

**Compute the degree centrality for nodes**

We found that the address **1BT4DYrt3ZoSz6WeGEZzrj4tdidUcpCfQ6** accounted for almost **76%** funds sent to **1NxaBCFQwejSZbQfWcYNwgqML5wWoE3rK4** (our unknown mining node’s most frequently send address)



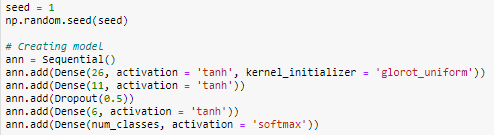
# Serial Model Comparison – ANN vs RF vs NB

# **Artificial Neural Network**

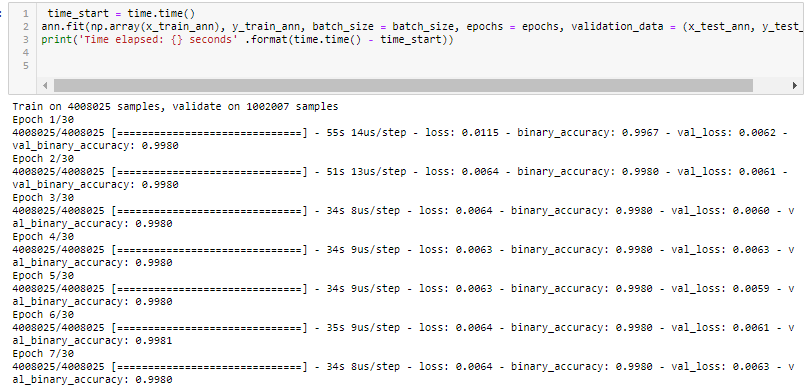
* We trained the pickled data from our original Google Big Query on different models including an Artificial Neural Network, Random Forest Classifier, Naïve Bayes,
* Scaling : Data preparation involved using techniques such as the normalization and standardization to rescale input and output variables prior to training a neural network model
* Feature Scaling is a method used to normalize the range of independent variables or features of data.
* **Hyperparameters** used are B*atch\_size, learn\_rate* and *epoch*

*We encoded out dependent and independent variable and converting it to a binary class matrix*

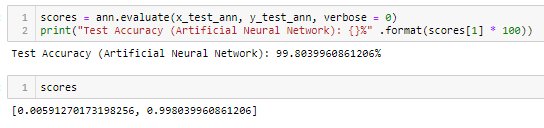
* y\_train\_ann.shape is (4008025, 2)
* y\_test\_ann.shape is (1002007, 2)
* We use a sequential model, The Sequential model allows us to build deep neural networks by stacking layers one on top of another
* Added dense layer, with **tanh** as our activation function f or the neurons!



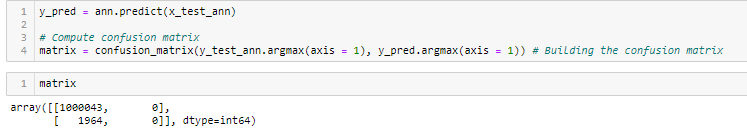
**Model fitting on 4008025 samples :**



After fitting the model, our ANN runs with an accuracy of **Test Accuracy (Artificial Neural Network): 99.8039960861206%**



**Confusion Matrix for ANN:**

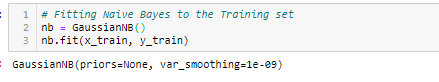


# **Note:** We also ran our Neural network using RC OpenOnDemand solution to get some performance gains on running our Jupyter Notebook by using a 1 GPU 2 CPU setup

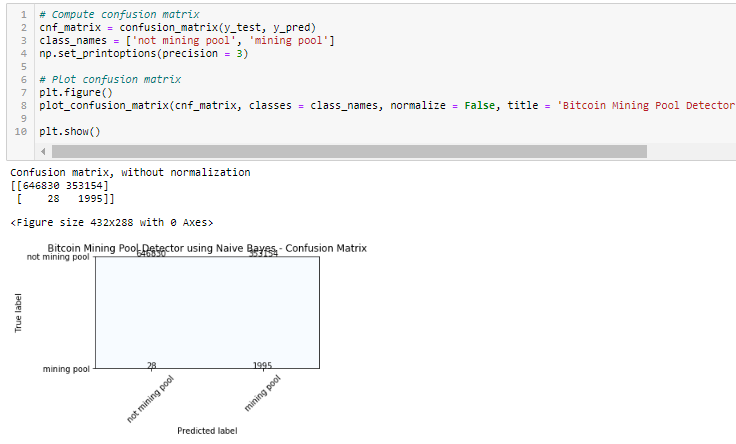
# 

# **Naïve Bayes**

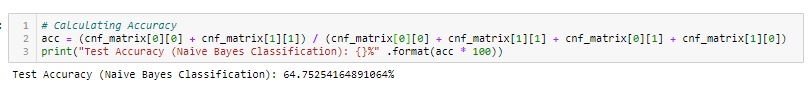
Fitting Naïve Bayes Algorithm to pickled Google Big Query Bitcoin dataset



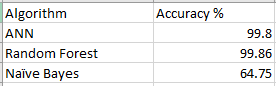
**Confusion Matrix**



**Accuracy of Naïve Bayes algorithm is : 64.75%**



* Post analysis, when the accuracy is compared, Random Forest Classifier performs best with our data, while ANN is very close to the accuracy of Random Forest Classifier



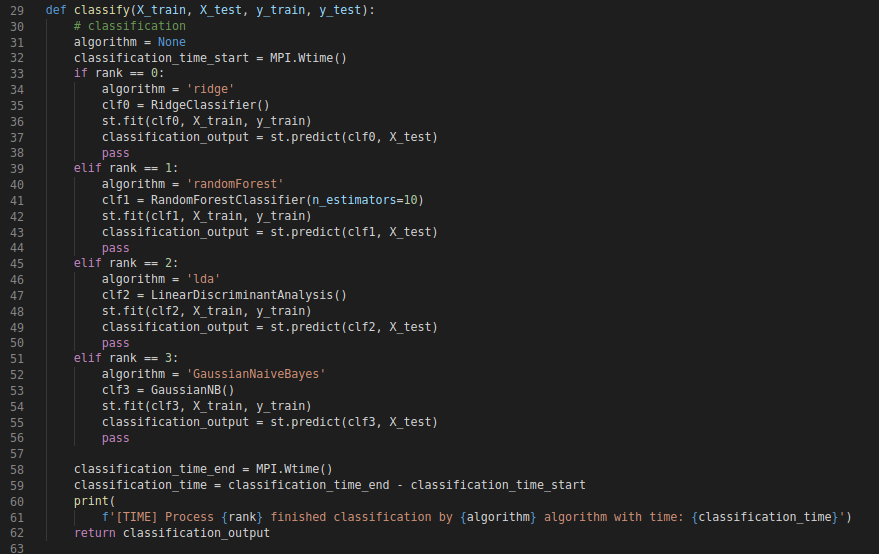
# 

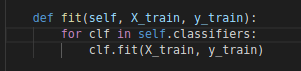
### **Model Training Parallelization**

We used a Stacking Algorithm **based on 4 classifiers:**

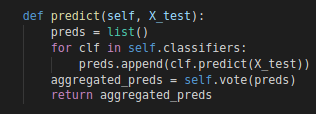
* RidgeClassifier
* RandomForestClassifier
* LinearDiscriminantAnalysis
* GaussianNB

**How Stacked Classifier Works**

* The program breaks down 4 MPI processes and trains 4 ML modes on the same dataset in parallel using **classify()**
* Each MPI process then provides a probabilistic prediction with **fit()**

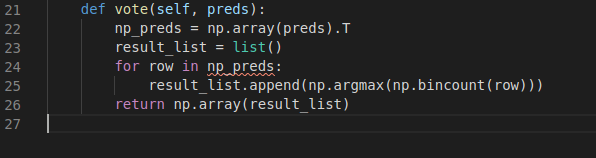


* Each MPI process then provides a probabilistic prediction with **predict()**



**Vote Function¹**

* The Master MPI Process then receives the 4 fitted models and then for each row takes a vote on which model to use for its prediction (binary 0 or 1)



**Improvements for Vote Function**

* **np.bincount** gives you the ratio of which model voted for which outcome [0 | 4], [1 | 3], [2 | 2], [3 | 1], [0 |4]
* Unfortunately, since its an even number of models **np.argmax** defaults to the first occurrence i.e. mining\_pool == False

**Solution** can take 2 approaches**:**

1. Numpy argmax - random tie breaking

2. Odd number of Classifiers (5 MPI Processes)

### We then ran the model in 2 variations. One in **Serial** and one in **Parallel** using OpenMPI

**Parallel Run Setup**

* Hardware Setup on Research Computing’s Discovery Cluster

*(4-node CPU cluster each with 180 GB RAM)*

*> $ srun -p short --cpus-per-task 4 --ntasks 4 --nodes 4 --pty --export=ALL --mem=180Gb --time=08:00:00 /bin/bash*

* We created a virtual anaconda environment to install our custom software

*> $ conda create -n <yourenvironmentname> python=3.7 anaconda*

* We used **MPI** for Python to interface the distributed memory modules of 4 separate HPC CPU nodes

*> $ conda install -c anaconda mpi4py*

We used OpenMP version 3.1.2

*> $ module load openmpi/3.1.2*

*Parallel Code Run*

# 

**Performance Gains in Model Training gained with Distributed Memory Parallelization**

# 

# Conclusion:

### We were able to optimize using parallelization at every step in the Data Science pipeline.

### We were able to use parallelization when saving our queried data from BigQuery using pyarrow's underlying parallelization (use\_threads = True) for Parquet and Feather.

### We used the underlying Cython parallelization provided by sklearn for all our ensemble methods using (n\_threads).

### Additionally, we chose the highest compression ratio when writing our raw dataset to file storage. Moreover, we chose to choose the best read/write speeds when compressing our dataset when sending between servers and HPC nodes.

### We even chose the most speediest model persistent format (job\_lib) to use for future predictive work.

### Most satisfyingly, we were able to utilize MPI to parallelize model training on a single dataset to optimize and vote on the best value for each row of data.