FIN-525

Financial Big Data

Predicting Cryptocurrency "Flash Crashes" on Minute Time Resolution Data

Students:
Linas Syvis
Alexander Rusnak

Professor:
Damien Challet

Lausanne, January 14, 2021

1 Introduction

In the currently available academic literature on high-frequency trading it is considered that in most cases high-frequency traders improve market liquidity [1], [2] and efficiency [3]. It is also known that in some cases the high-frequency traders cause or magnify huge losses for the market. Such cases are known as "flash crashes". There is no rigorous definition of what can be called a flash crash, however, literature agrees that it is an event in financial markets wherein the selling or purchasing of financial assets rapidly amplifies price movement. The result appears to be a rapid sell-off or buy-out of securities that can happen over a few minutes, resulting in dramatic declines or increases. High-frequency trading companies are said to be highly responsible for flash crashes recently [4]. Indeed, they consume liquidity during flash crashes, which is instead provided by "traditional" slow traders, and generate a transitory price impact which is unrelated to the permanent impact. In order to be able to foresee and prevent such events it is largely complicated. There may be too many variables in the financial markets, for machine learning in its current state of development to deal with [5].

However, the data that needs to be considered when it comes to trading cryptocurrencies is of a smaller magnitude. Data in cryptocurrencies might be more speculative and co-dependent, which may result in far more predictable patterns than in the financial markets [5].

The cryptocurrency markets operate without breaks and cryptocurrency trading is often done via automated trading programs. High-frequency trading, on the other hand, is not possible on any of the larger cryptocurrency exchanges in order not to break the exchange itself. However, cryptocurrency markets are still prone to the inaccurate feedback loops that define automated market behavior. In other words, trading bots detect increased volume which activates new trading programs which detect increased volume and so on and so forth.

In this paper, we analyze such market behaviour in the cryptocurrency markets. More specifically, we define more rigorously what a flash crash is, analyze them, and develop a model detecting them for the largest cryptocurrencies.

2 Data analysis

In this paper we use data of the largest cryptocurrencies by their Market Cap. according to [6]. The data was downloaded from Kaggle website [7]. It contains the historical trading data (OHLC) of the cryptocurrency/USD pairs at 1 minute resolution reaching back until the year 2013. It was originally collected from the Bitfinex exchange using their API. There are no timestamps for time periods in which the exchange was down. Also, if there were time periods without any activity or trades there will be no timestamp as well.

2.1 Exploratory data analysis

We first begin our analysis by taking a global view of the structure of the data, which is visualized in Figure 10. The data originally consisted of 6 columns: *time*, *open*, *close*, *high*, *low*, *volume*. Since timestamps were encoded in a 13-digit number, in order to better understand them, we transformed them into a date and time format. There were no missing values, which, when modelling, need to be replaced. However, it seems that some missing data might be filled in by the exchange itself (row 2) and may not be too reliable. Also, some cryptocurrencies, like XRP or LTC, had large time gaps of no data in 2013 and 2014. After a more in depth analysis, we observed that such problems occur mostly with older data.

	time	open	close	high	low	volume	ticker
2013-04-01 00:07:00	1364774820000	93.25	93.30	93.30	93.25	93.300000	BTCUSD
2013-04-01 00:08:00	1364774880000	100.00	100.00	100.00	100.00	93.300000	BTCUSD
2013-04-01 00:09:00	1364774940000	93.30	93.30	93.30	93.30	33.676862	BTCUSD
2013-04-01 00:11:00	1364775060000	93.35	93.47	93.47	93.35	20.000000	BTCUSD
2013-04-01 00:12:00	1364775120000	93.47	93.47	93.47	93.47	2.021627	BTCUSD

Figure 1: Data structure.

We further look into the data more in detail and pick a few of the largest cryptocurrencies we used in the analysis. In Figure 2 we can see the exchange rate of BTC/USD evolution over time as the blue line. We notice that the coin has grown exponentially: since 2017 it *exploded* and is currently around an all time high.

This *explosion* is a reason we cannot have other cryptocurrencies in the same graph as BTC - their lines are barely visible. ETH - the second largest cryptocurrency looks much smaller when judging just according to price. However, its' circulating supply (which is around 6x larger) makes the coin's market capitalization much closer to the one from Bitcoin. We show the evolution of market capitalization of the same coins in Figure 12 in the Appendix. If we visualize ETH/USD and LTC/USD in another graph, they show a similar pattern as BTC/USD (especially ETH/USD). There might be many reasons for such similarity: high correlation between the cryptocurrencies; the effect of strengthening/weakening USD, performance of stocks in financial markets, etc.

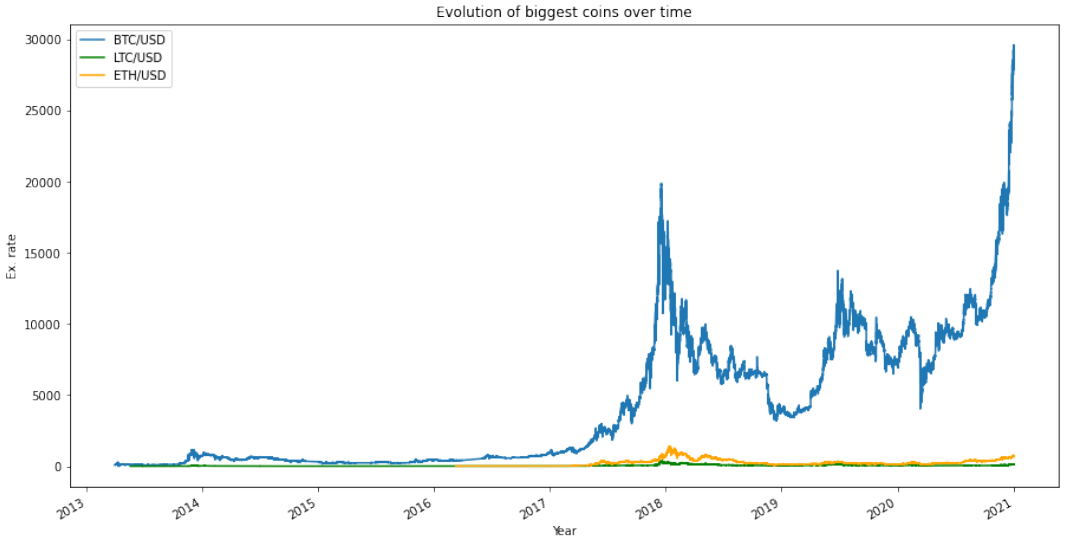

Figure 2: BTC/USD evolution over time.

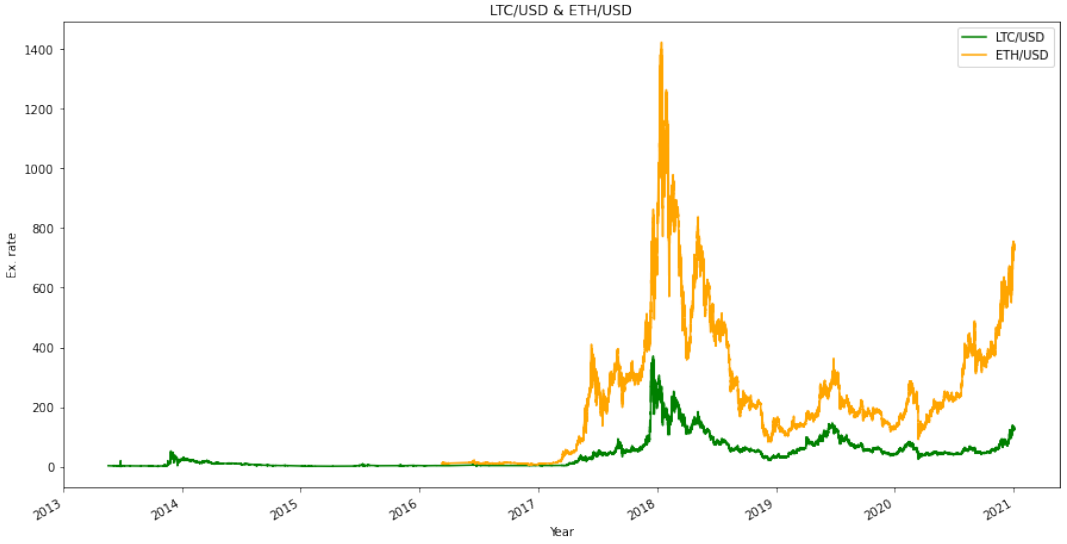

Figure 3: ETH/USD and LTC/USD evolution over time.

Next, we take a look at one of the most recent crashes of Bitcoin, which happened in August, 2020. When looking at the Figure 4, we observe a sharp decline of around 12 percent in a span of 15 minutes. Such drop was then followed by a small recovery. However, the effect of the decrease stayed for the longer term. The reason why this Bitcoin flash crash happened was not immediately clear. A most common explanation found was that it was caused by so-called *whales* who control large amounts of Bitcoin and other cryptocurrencies moving the market. The market is more easily pushed around by whales when trading volumes are lower, such as early on that Sunday morning. Indeed, when looking at the data, the volumes dramatically increased during the flash crash. The Figure 11 in Appendix displays

this phenomenon.

One might be able to exploit such rate changes and generate profit, by having a *flash crash* detection system. To take advantage of a sharp movement, it is essential to study the underlying reasons, the environment just before and after the crash and the aftermath.

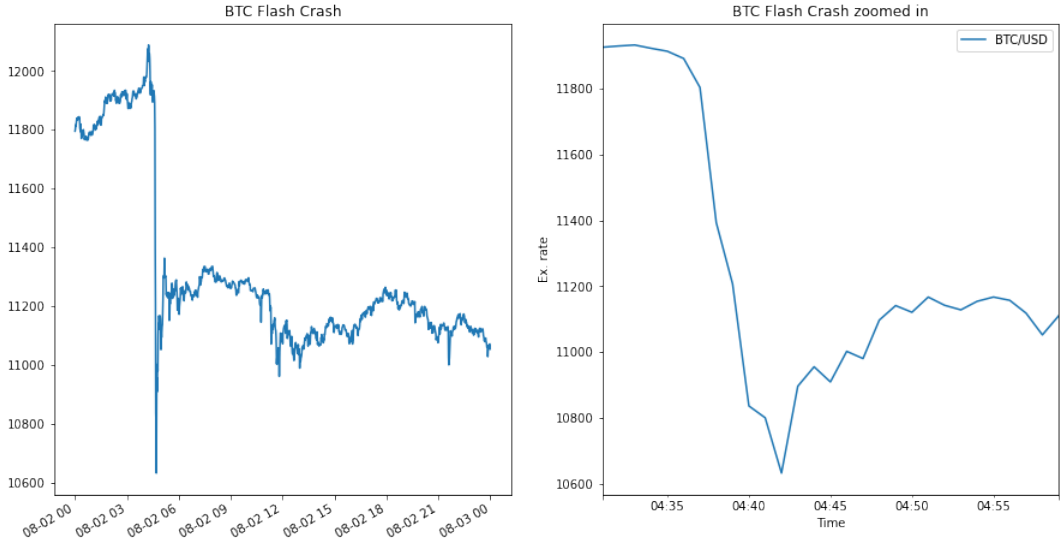

Figure 4: BTC/USD flash crash.

2.2 Definition of a flash crash

As mentioned in the introduction, in the scientific literature, there is no specific definition of a flash crash. Intuitively, it is some abnormal decline that happens in a very short period of time. Now, what one may call 'abnormal' is abstract and 'very short period of time' depends on the situation. So, there are at least two parameters that one has to specify: period of time and magnitude, when defining what a flash crash is. In this project, we decided to call 'flash crashes' movements of 2% or more in the period of 15 minutes. One may say that a change of 2% is relatively small for cryptocurrencies, however, that happening in a 15 minutes timespan was not that common. For example, Bitcoin had around 1200 such crashes since 2013, out of over 3 million timestamps. This makes for a good amount of data points when modelling. It was important not to restrict the flash crash definition too much, because then we would only have a few data points to learn from. It was also important not to have too loose of a definition for a flash crash in order not to label too many points as flash crashes. In the end, it should still be a relatively rare phenomenon.

3 Modelling

3.1 Volume-synchronized probability of informed trading

As an additional metric to describe our data and provide more structured information to our classifier, we implemented the VPIN flow toxicity metric, which stands for volume-synchronized probability of informed trading. Such metric was introduced by Easley et al. [8].

Probability of informed trading has a well documented history of being used to estimate flow toxicity in high frequency data, and thus is useful for estimating flash crashes as it can provide an approximation of how informed trading events are. It has been demonstrated that VPIN has a correlation with high short-term toxicity induced volatility, and can predict large price moves from this volatility. If the trades are extremely toxic (uniformed) then it is likely they are entering feedback loops or stochastic processes that could lead to flash crashes. Traditional PIN approximation rates model daily buys and sells of stocks, but are based on a time-variant approach of the arrival rate of particular traders. Since the flow of trades does not always distribute equally between time frames, it is important to equalize this distribution when determining the probability of informed trading. Thus, the VPIN metric recalibrates the data to have equal volume of trades in each time interval, which overcomes variance in volume flow in highly active or evolving markets.

In the figure below, we compare the VPIN metric across time for 8 coins in our dataset. It demonstrates clearly that the bulk of toxic flow occurs in a more recent time span for smaller / less stable coins, which have demonstrably higher volatility than larger and more established coins like bitcoin.

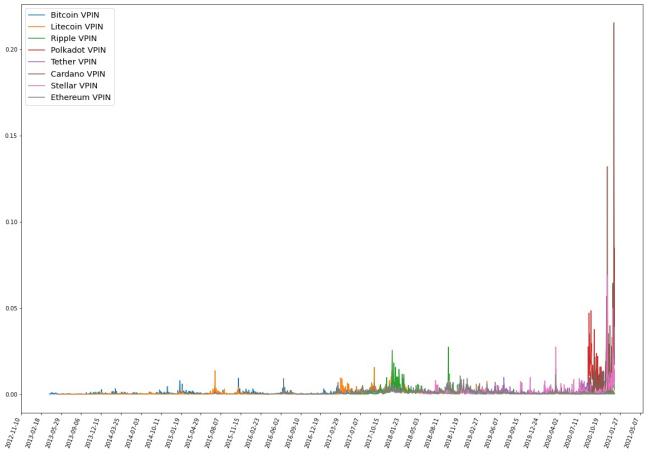

Figure 5: VPIN Metric

Coin Name	Mean Volume	Coef. of Variation	Mean VPIN
Bitcoin	17.97	0.946444	0.0007116
Ethereum	142.64	0.744645	0.0012860
Tether	13521.55	0.009645	0.0035474
Ripple	33354.14	0.786306	0.0016066
Cardano	7122.49	0.227688	0.0241297
Litecoin	168.14	0.857291	0.0011752
Polkadot	416.19	0.187772	0.0190554
Stellar	9526.83	0.567248	0.0047683
Tops Coins	8033.75	0.540879	0.0070351
All USD Paired Coins	17006.08	2.090281	N/A

We also found a relationship between the crashes themselves and the VPIN metric, which is a result we were looking for. The correlation between the VPIN metric and the flash crash binary variable was 0.153934. Future analysis could be done to correlate the directionality and percent change of the crashes to the VPIN metric.

3.2 Gradient Boosted Tree Classifier

In order to classify which subsequent frames might contain a crash event, we decided to use a gradient boosted tree classifier fed with the preprocessed data of the 8 top coins on which we focused our analysis. We staged our problem as a binary classification problem, with 0 representing no crash in the following frame, and 1 representing a crash in the following frame. Gradient boosted trees are based on a standard random forest algorithm, which itself is based on an ensemble of decision trees. The decision trees create delineating partitions between different values of the data that it uses to sort them into classes. Since random forests use many trees, they can learn more complex combinations of data that point to class differentiation. The gradient boosted aspect refers to the way in which the trees are initialized, the model incorporates the gradient with respect to the error from the last created tree, and uses it to generate progressively more accurate trees as it trains. This approach has been demonstrated to be substantially more computationally efficient and accurate than a standard random forest. In particular, we use the XGBoost library for our implementation. At each time step, we feed the model the open, close, high, and low values of that step. We also give the price change, raw volume, adjusted volume, and the VPIN of the preceding day. As previously mentioned, the target of the model is a binary class label representing either a crash event in the following time step, or the lack of a crash event.

3.3 Acceleration

Training our model on such large amounts of data is a computationally expensive task, and one that consumes a lot of time on our relatively small devices. To that end, we accelerated training the XGBoost algorithm by distributing computations

to the GPU. To test the speed up capabilities, we trained using a subset of our data (just the training set of LiteCoin, or 1,455,199 rows) on both the CPU and GPU of one of our local machines. In total, this smaller dataset took 166.89 seconds to train on the CPU and only 143.74 seconds when distributed to the GPU, a 13.86 percent reduction in compute time. When scaled up to our larger top coins dataset, this is a substantial saving in time of approximately 2 minutes.

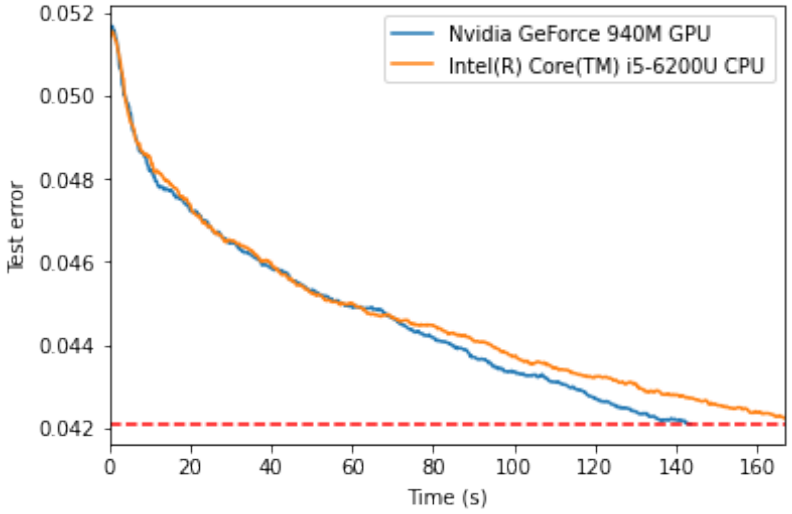

Figure 6: Hardware Comparison on LiteCoin Data

In addition to using the GPU for our model, we also used a Dmatrix for our train and test sets instead of a standard pandas dataframe or numpy matrix. A DMatrix is an optimized data structure that is part of the XGBoost ecosystem, and generally increases memory efficiency and speed. We utilized dask as well wherever appropriate in our code to speed up computations on the CPU by optimizing their execution. We also saved with the parquet file type in any function where the repetitive access of files was needed, and thus could be sped up by quicker read / write speeds.

3.4 Dask implementation

As these days the datasets and computations scale faster than CPUs and RAM, we need to find ways to scale our computations too. For this reason, in addition to our two limited laptops with i5 and i7 processors and 8GB of RAM, we used Dask. Dask is a dynamic task scheduling open-source library, which splits up large computations and routes parts of them efficiently onto distributed hardware.

To avoid excess memory use, Dask is good at figuring out how to evaluate computations in a low-memory environment by pulling in chunks of data from disk, doing the necessary processing, and throwing away temporary values rapidly. Dask enabled us to efficiently parallel computations on our laptops by using and abusing their multi-core CPUs. As a result it saved significant amount of time when labeling the data into "flash crash" / "not flash crash" and modelling multiple currencies.

3.5 Data Pipeline

We used multiple base datasets of varying sizes, but generally used the same preprocessing pipeline. After taking the raw cryptocurrency data and labeling it with crashes and calculating the VPIN, we changed the datetime to a singular float number representing the 24 hour clock time of the event. Our input to our model was then: float time, high, low, open, close, raw volume, change in price over frame, and vpin. We then used a standard scaler on all the features, and split them into train and test datasets for the model. The main datasets we ended up using were: the top coins dataset consisting of 9,386,046 rows, the LTC coin set consisting of aforementioned 1,455,199 rows, and a final USD paired dataset of all the coins we could process in time / handle in the memory of our computer when training the model, of 12,317,928 rows. We also used a multi coin (IOT, EOS, ETH) JPY paired dataset of 436,924 rows and a multi coin (IOT, EOS, ETP) ETH paired dataset of 792,569 rows for cross pair compatibility testing. Our largest computation hurdle in terms of time was processing the labels for the data and calculating the VPIN, while our biggest challenge in terms of memory allocation was handling the largest datasets while training the model.

4 Results

We found that we were accurately able to train our XGBoost model on our dataset to predict which frames preceded flash crashes. Our main observations in regards to model performance were: quantity of data vs specificity of data, feature importance, and applicability across currency pairings.

4.1 Quantity vs Specificity

We observed through our experiments with different models that not all coins have the same patterns, but that in general having a larger dataset for training is particularly useful even for more specific, particular coins. We observed that while training a model on one coin with a sufficiently large dataset performed slightly better on that particular coin then a more general model, it was horrible in terms of generalization as it overfit to the specific coin. That is, as the singular coin model became better at targeting that particular coin, it became much worse at targeting other coins which were out of sample. This does not bode well for the robustness or generalizabilty of the single coin models across the cryptocurrency trading space or across time. Furthermore, when trained and tested on the entire dataset, the top coins model achieved superior loss values than the single coin model did when tested on the single coin test set. This could be due to class imbalances in the larger dataset, but we find this unlikely considering our test coin (LiteCoin) did not have substantial volatility relative to other coins in our set.

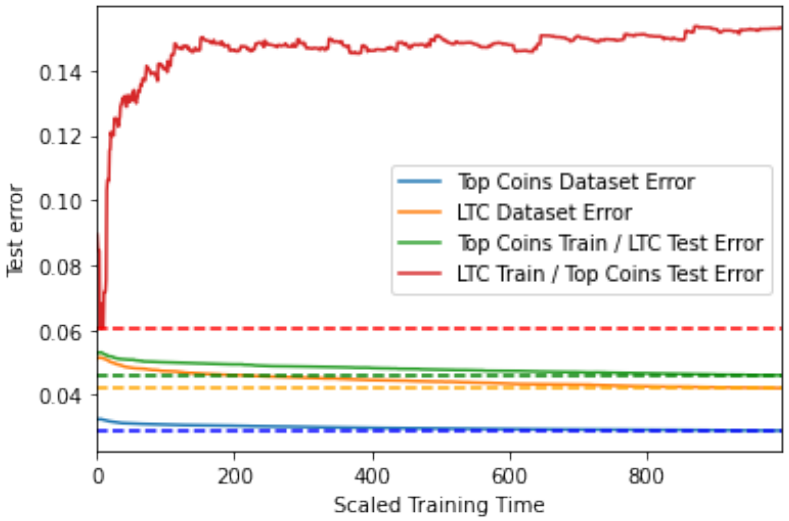

Figure 7: Comparative Loss on Different Train and Test sets

4.2 Feature Importance

We observed that the feature importance of different models varied based on training runs due to the stochastic nature of the model, but primarily based on the dataset.

When using the smaller LTC set, the primary feature of importance was actually the float time, but when looking at the larger datasets, the VPIN metric was the most important predictor. This is indicative of the importance of the VPIN metric to predicting highly volatile markets and flash crashes.

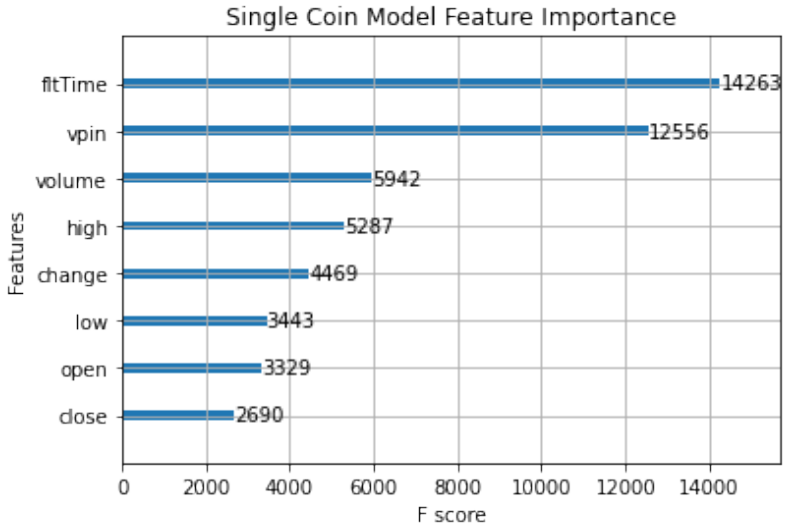

Figure 8: LTC Dataset Feature Importance

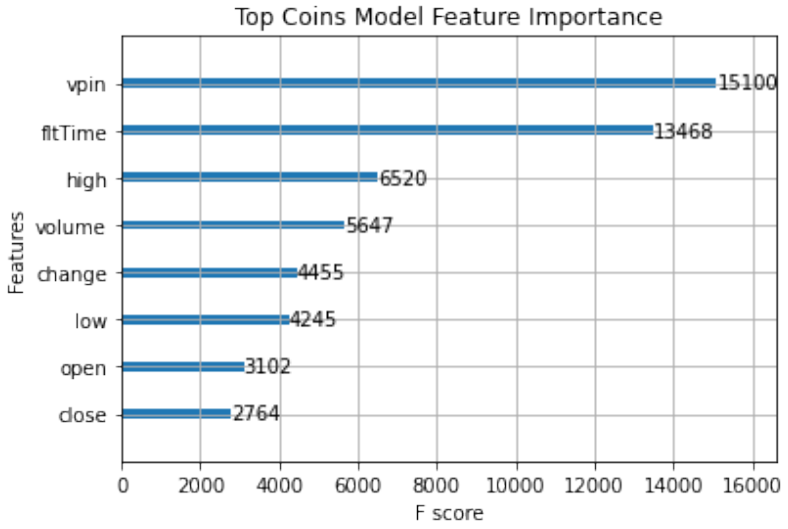

Figure 9: Top Coins Dataset Feature Importance

4.3 Applicability Across Currency Pairings

In addition to examining which models performed best on more broad and more specific datasets, we also explored the efficacy of models trained on Crypto-USD paired data to other pairings. In particular we looked at Crypto-JPY pairings, and

Crypto-Crypto pairings (with ETH as the common base pairing for the Crypto-Crypto set). We also used our Max Coins USD dataset for training in this analysis, as we learned in our earlier analysis that more training data led to better out of sample results/ more generalizable results. We observed that there was divergence between the JPY pairings and USD pairings, and between the ETH pairings and USD Pairings (i.e. as the model increased its performance on the USD pairing training set, it's performance on the other pairings decreased). However, the performances were not equal. With the JPY pairing, the error remained relatively low in relation to the USD pairing error. In contrast, the ETH pairings had the worst error rate we observed throughout all of our testing, indicating there is a substantial divergence in the movements of the market for the crypto-crypto pairings. This makes logical sense as the JPY and USD share many characteristics because they are both highly traded fiat currencies from large westernized economies, but we did not expect the difference in performance to be so stark. Further examination of this disparate performance could be a path for future research.

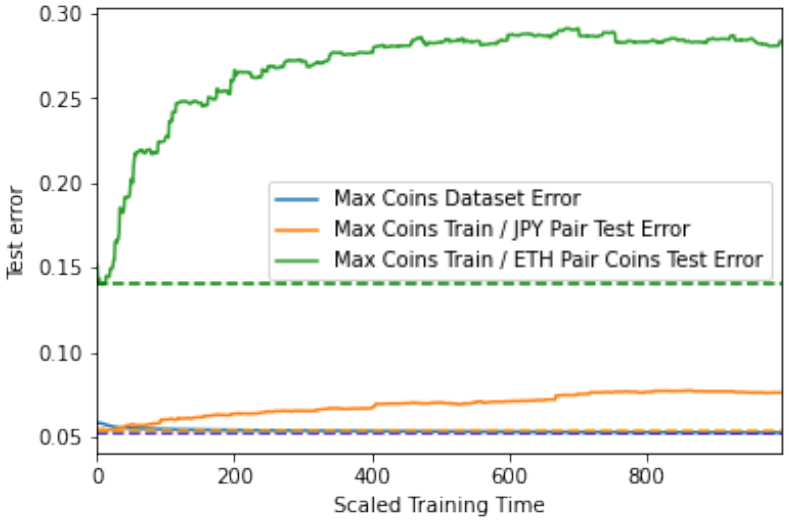

Figure 10: Max Coins Model performance on Max Coins, JPY Pairing, and ETH Pairing Test Sets

5 Conclusion

While high frequency traders and algorithmic traders add substantial liquidity to markets, they also can be responsible for high volatility and large scale price moves known as 'flash crashes.' Though much research has been done on this topic in regards to equities, traditional currencies, and other securities, the cryptocurrency space remains less examined. It is particularly interesting to analyze because of the unique restrictions on trading in regards to coin size and mining time that define the moves of the market. We analyzed copious amounts of cryptocurrency data using the volume-synchronized probability of informed trading metric to determine flow toxicity, and found that it does have a correlated relationship to and predictive effect for flash crashes. We then utilized an XGBoost gradient boosted tree model to further test the efficacy of the VPIN metric, which we confirmed on larger datasets as the most important variable for prediction. We also tested the predictive power of our tree model across more specific coin datasets and other currency pairings, finding that while more data does lead to a less fragile model, there are substantial differences between currency pairing types, particularly cryptocurrency-cryptocurrency pairs. In summation, we effectively created a predictive classifier, using large amounts of data, for flash crashes in the cryptocurrency space and confirmed the importance of the VPIN metric for predicting these crashes.

6 Appendix

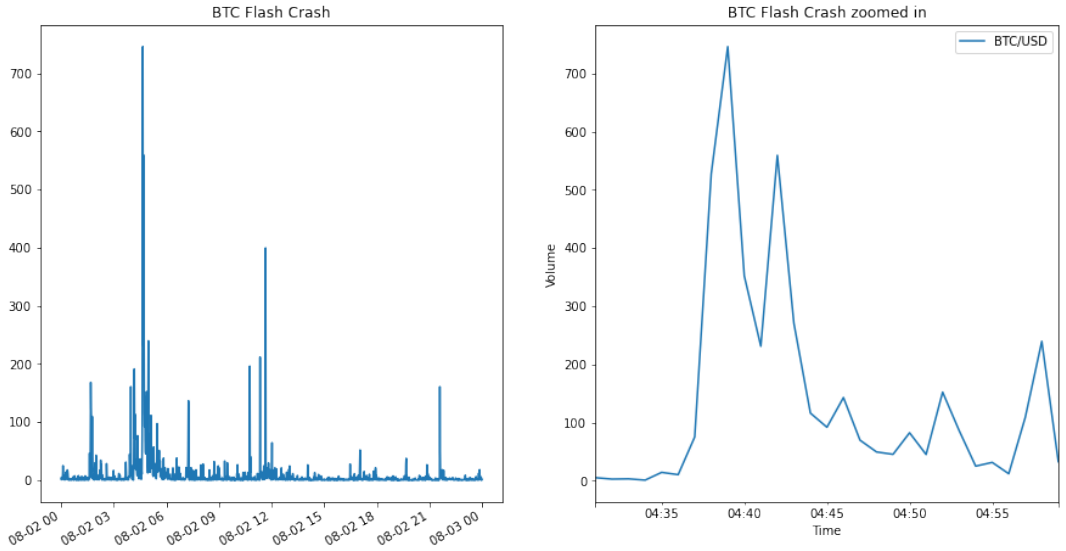

Figure 11: BTC/USD flash crash Volume.

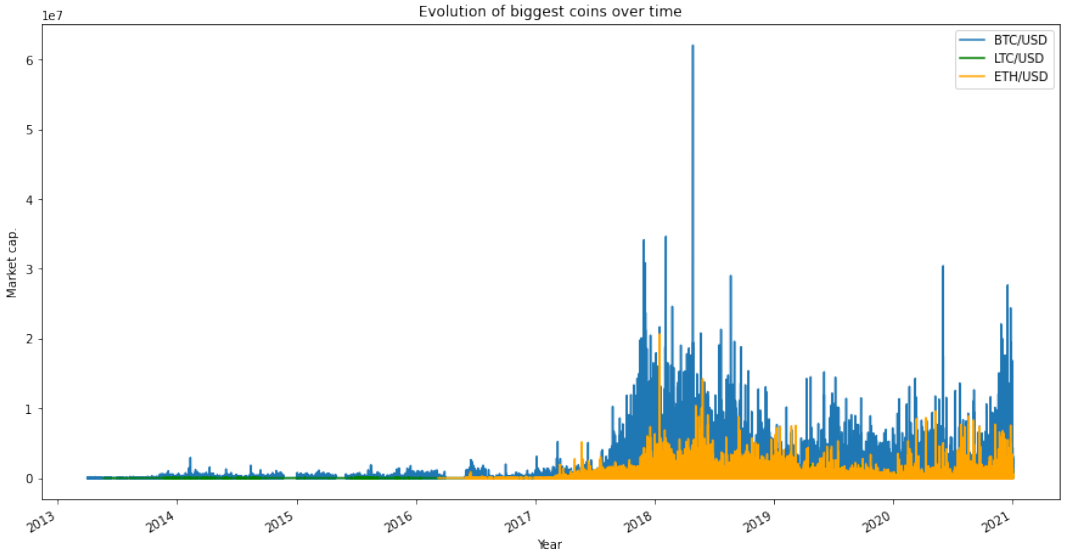

Figure 12: Cryptocurrencies market cap over time.

Project code

https://github.com/amrusnak/FinBigDataProject

```
# Important Info for this Notebook:

Since our datasets are much too large to be included in a github reposito

### Please run each cell one by one, with respect to the comments, so as

import pandas
from math import *
import math
import numpy as np
import dask
import matplotlib.pyplot as plt
from datetime import timedelta
import matplotlib.dates as mdates
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from xgboost import XGBClassifier
import time

import xgboost as xgb
from sklearn.metrics import accuracy_score
dask.config.set(scheduler="processes")


# Labeling crashes in the given dataset
# @dask.delayed
def label_data(flashcrash,
    timespan = 10, #minutes
    change = 0.1): #0.1 - 10% change

    crash = []

    for i in np.arange(sum(flashcrash.index <= flashcrash.index.max() - d

        if any(abs(flashcrash[(flashcrash.index < flashcrash.index[i] + d
            crash.append(flashcrash.index[i])

    return crash
```

```
coin = pd.read_parquet('testData/adausd.gzip').drop('Unnamed: 0', axis=1)
coin.index = pd.to_datetime(coin.time.values, unit='ms')
coin

# Test labeling function
t0=time.time()
flashcrash = np.log(coin[(coin.index.date >= dt.date(2008,3,1)) & (coin.i
#crashD = label_data(flashcrash, timespan = 15, change = 0.02)
#crash = dask.compute(crashD)
crash = label_data(flashcrash, timespan = 15, change = 0.02)
t1=time.time()
print('Runtime: %.2f s' %(t1-t0))
pd.DataFrame(crash).to_csv("testData/crashSetADA.csv")
crash

# This function we used to run multiple files, given we only included one
# it isn't demonstrated
def findCrashesMulti(fileList, coinName):
    for i in range(len(fileList)):
        print(fileList[i])
        coin = pd.read_parquet('/media/alexander/AMR_CHAMPER/FInBigData/p
        coin.index = pd.to_datetime(coin.time.values, unit='ms')
        flashcrash = np.log(coin[(coin.index.date >= dt.date(2008,3,1)) &
        crash = label_data(flashcrash, timespan = 15, change = 0.02)
        pd.DataFrame(crash).to_csv("/media/alexander/AMR_CHAMPER/FInBigDa

# some necessary functions for the VPIN function
def std(list):
    element = 0
    for item in list:
        element = element + float((item**2)/( len(list)))
    return math.sqrt(element)

def phi(x):
    #'Cumulative distribution function for the standard normal distributi
    return (1.0 + erf(x / sqrt(2.0))) / 2.0

def calx(v_i,delta_p_i,sigma):
    x = v_i * phi(delta_p_i/sigma)
    return x

def distance(a, b):
    if (a == b):
```

```
        return 0
    elif (a < 0) and (b < 0) or (a > 0) and (b > 0):
        if (a < b):
            return (abs(abs(a) - abs(b)))
        else:
            return -(abs(abs(a) - abs(b)))
    else:
        return math.copysign((abs(a) + abs(b)),b)

# Preparing the coin data for VPIN calculation, mostly through seperating
# and adding a close - open change column

def prepCoinVPIN(coinData):
    coinData.index = pd.to_datetime(coinData.time.values, unit='ms')
    coinData['datetime'] = coinData.index
    coinData.time = coinData.index.strftime('%H:%M:%S')
    coinData['fltTime'] = coinData.index.strftime('%H%M').astype(float)
    coinData['date'] = coinData.index.strftime('%Y-%m-%d')
    coinData['change'] = coinData.close - coinData.open
    coinData = coinData.reset_index().drop('index', axis=1)
    return coinData

# Testing prepCoinVPIN function

coin = pd.read_parquet('testData/adausd.gzip').drop('Unnamed: 0', axis=1)
coin = prepCoinVPIN(coin)
coin

# Splitting the coin into different lists for VPIN analysis

def splitCoinVPIN(preppedCoin):
    high = preppedCoin["high"].tolist()
    low = preppedCoin["low"].tolist()
    vol = preppedCoin["volume"].tolist()
    chg = preppedCoin["change"].tolist()
    date = preppedCoin["date"].tolist()
    time = preppedCoin["time"].tolist()
    sigma = std(chg)
    totalvol = sum(vol)
    return high, low, vol, chg, date, time, sigma, totalvol

# Testing splitCoinVPIN fucntion
high, low, vol, chg, date, tme, sigma, totalvol = splitCoinVPIN(coin)
chg
```

```
# prep and calculate the VPIN metric for a coin.

#@dask.delayed
def calcVPIN(coin):
    df = prepCoinVPIN(coin)
    high, low, vol, chg, date, time, sigma, totalvol = splitCoinVPIN(df)
    dateset = tuple(set(date))
    vpin=[]

    for eachdate in dateset:
        dateslice = df[np.logical_not(df['date'] != eachdate)]
        datevol = dateslice['volume'].sum()
        vbs = datevol/50

        delta_p_i = []
        v_i = []
        x=[]

        highslice = dateslice["high"].tolist()
        lowslice = dateslice["low"].tolist()
        volslice = dateslice["volume"].tolist()

        highbas=[]
        lowbas=[]
        volbas=[]

        for i in range(len(dateslice.index)):
            highbas.append(highslice[i])
            lowbas.append(lowslice[i])
            volbas.append(volslice[i])
            if sum(volbas) > vbs:
                v_i.append(sum(volbas[0:-1]))
                delta_p_i.append(abs(max(highbas)-min(lowbas)))
                highbas = highbas[-1:]
                lowbas = lowbas[-1:]
                volbas = volbas[-1:]
        for i in range(len(v_i)):
            x.append(calx(v_i[i],delta_p_i[i],sigma))
        v_tau_b=sum(x)
        dateitem= abs(2*v_tau_b - vbs)
        vpin.append(dateitem/totalvol)
        df.loc[df[df['date']==eachdate].index, 'vpin'] = dateitem/totalvo
    return df
```

```
# Test/ Use calcVPIN function
coin = pd.read_parquet('testData/adausd.gzip').drop('Unnamed: 0', axis=1)
# coinDFD = calcVPIN(coin)
# coinDF = dask.compute(calcVPIN(coin))
coinDF = calcVPIN(coin)
coinDF

# attach seperate crash labels to VPIN dataframe
def setCrashes(coin, crashes):
    coin['isCrash'] = 0
    coin.loc[coin[coin['datetime'].isin(crashes)].index, 'isCrash'] = 1
    return coin

# Test / use setCrashes Function

coinDF['datetime']=pd.to_datetime(coinDF['datetime'])
crashes = pd.read_csv('testData/crashSetADA.csv')
crashDF = setCrashes(coinDF, crashes['0'])
crashDF.to_parquet('testData/adausdCrash.gzip')
crashDF

# Calc VPIN, attach crash labels, and save for list of coins
# As we are only demonstrating one coin, we don't use this function
def vpinCrashMulti(fileList, coinName):
    for i in range(len(fileList)):
        print(fileList[i])
        try:
            coin = pd.read_parquet('/media/alexander/AMR_CHAMPER/FInBigDa
            coinDF = calcVPIN(coin)
            coinDF['datetime']=pd.to_datetime(coinDF['datetime'])
            crashes = pd.read_csv("/media/alexander/AMR_CHAMPER/FInBigDat
            crashDF = setCrashes(coinDF, crashes['0'])
            crashDF.to_csv('/media/alexander/AMR_CHAMPER/FInBigData/Crash
        except:
            print('no crashes!')

# Load in list of USD paired coins minus top coins which we had already r
universeUsd = pd.read_csv('universeUsd.csv')
topCoins = ['btc', 'eth', 'ada', 'ust', 'usdt', 'xlm', 'xrp', 'ltc', 'dot
universeUsd = universeUsd[~universeUsd.coinTag.isin(topCoins)].reset_inde
universeUsd

# using vpinCrashMulti for ethereum pairings
```

```
# demo, won't run as files aren't present

filesEth = ['ioteth.gzip', 'etpeth.gzip', 'eoseth.gzip']
coinsEth = ['ioteth', 'etpeth', 'eoseth']

vpinCrashMulti(filesEth, coinsEth)

# using vpinCrashMulti for USD pairing universe
# demo, won't run as files aren't present

vpinCrashMulti(universeUsd.parquetFiles, universeUsd.coinTag)

# Loading in and processing VPIN data for top coins

# None of these files are present for github memory reasons,
# so these won't run if you test them

coinDF = pd.read_csv('/media/alexander/AMR_CHAMPER/FInBigData/vpin/btcusdV
coinDF['date'] = pd.to_datetime(coinDF['date'])

coinDF2 = pd.read_csv('/media/alexander/AMR_CHAMPER/FInBigData/vpin/ltcusd
coinDF2['date'] = pd.to_datetime(coinDF2['date'])


coinDF3 = pd.read_csv('/media/alexander/AMR_CHAMPER/FInBigData/vpin/xrpusd
coinDF3['date'] = pd.to_datetime(coinDF3['date'])
coinDF4 = pd.read_csv('/media/alexander/AMR_CHAMPER/FInBigData/vpin/dotusd
coinDF4['date'] = pd.to_datetime(coinDF4['date'])


coinDF5 = pd.read_csv('/media/alexander/AMR_CHAMPER/FInBigData/vpin/ustusd
coinDF5['date'] = pd.to_datetime(coinDF5['date'])
coinDF6 = pd.read_csv('/media/alexander/AMR_CHAMPER/FInBigData/vpin/adausd
coinDF6['date'] = pd.to_datetime(coinDF6['date'])

coinDF7 = pd.read_csv('/media/alexander/AMR_CHAMPER/FInBigData/vpin/xlmusd
coinDF7['date'] = pd.to_datetime(coinDF7['date'])
coinDF8 = pd.read_csv('/media/alexander/AMR_CHAMPER/FInBigData/vpin/ethusd
coinDF8['date'] = pd.to_datetime(coinDF8['date'])

# Calc VPIN means for top coins
print(coinDF['vpin'].mean())
print(coinDF2['vpin'].mean())
```

```
print(coinDF3['vpin'].mean())
print(coinDF4['vpin'].mean())
print(coinDF5['vpin'].mean())
print(coinDF6['vpin'].mean())
print(coinDF7['vpin'].mean())
print(coinDF8['vpin'].mean())

# Mean of VPIN between top coins
sVPIN = (coinDF['vpin'].mean() + coinDF2['vpin'].mean() + coinDF3['vpin']

mVPIN = sVPIN/8
mVPIN

# Plot VPIN through time for top coins
# No datasets present for this plot

fig, ax = plt.subplots(figsize=(20, 15))


plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.DayLocator())

ax.plot(coinDF.date, coinDF.vpin, label='Bitcoin VPIN')
ax.plot(coinDF2.date, coinDF2.vpin, label='Litecoin VPIN')
ax.plot(coinDF3.date, coinDF3.vpin, label='Ripple VPIN')
ax.plot(coinDF4.date, coinDF4.vpin,  label='Polkadot VPIN')
ax.plot(coinDF5.date, coinDF5.vpin, label='Tether VPIN')
ax.plot(coinDF6.date, coinDF6.vpin, label='Cardano VPIN')
ax.plot(coinDF7.date, coinDF7.vpin, label='Stellar VPIN')
ax.plot(coinDF8.date, coinDF8.vpin, label='Ethereum VPIN')

stepsize=100
start, end = ax.get_xlim()
ax.xaxis.set_ticks(np.arange(start, end, stepsize))
plt.gcf().
autofmt_xdate()
plt.yticks(fontsize="xx-large")
plt.xticks(rotation=70, fontsize="xx-large")
ax.legend(fontsize="xx-large")
plt.savefig("vpin.jpg")
plt.show()
plt.close()

# List of all fully processed files with VPIN and Crashes
```

```
# allCoinsCrash.gzip is the top coins parquet file
# allCoinsCrash.gzip is the max possible coins parquet file

# this directory isn't present in the repo so don't re run
os.listdir('/media/alexander/AMR_CHAMPER/FInBigData/Crashes/')


# These datasets aren't present so don't re-run
# Creating original top coins combined dataset, appended seperately to ch
allCoins = pd.read_csv('/media/alexander/AMR_CHAMPER/FInBigData/Crashes/b
allCoins

allCoins = allCoins.append(pd.read_csv('/media/alexander/AMR_CHAMPER/FInB
allCoins

allCoins = allCoins.append(pd.read_csv('/media/alexander/AMR_CHAMPER/FInB
allCoins = allCoins.append(pd.read_csv('/media/alexander/AMR_CHAMPER/FInB

allCoins = allCoins.append(pd.read_csv('/media/alexander/AMR_CHAMPER/FInB
allCoins = allCoins.append(pd.read_csv('/media/alexander/AMR_CHAMPER/FInB


allCoins = allCoins.append(pd.read_csv('/media/alexander/AMR_CHAMPER/FInB
allCoins = allCoins.append(pd.read_csv('/media/alexander/AMR_CHAMPER/FInB

allCoins = allCoins.drop({'Unnamed: 0', 'Unnamed: 0.1'}, axis=1)

# Save top coins
allCoins.to_parquet('/media/alexander/AMR_CHAMPER/FInBigData/Crashes/allC

# Resuming runable section relying on present data
# Reading in LTC data for model training
adaCrash = pd.read_csv('testData/adausdCrash.csv')
adaCrash

# this dataset isn't present in github repo
# Loading in max coins data for training
allCoins = pd.read_parquet('/media/alexander/AMR_CHAMPER/FInBigData/Crash

# this dataset isn't present in github repo
# loading in ETH pairing data for testing
ethCrash = pd.read_parquet('/media/alexander/AMR_CHAMPER/FInBigData/Crash

# Scales data and then seperates into train and test sets
```

```
def seperateTarget(coinData):
    coinY = coinData['isCrash']

    coinX = coinData[{'open', 'close', 'high', 'low', 'volume', 'change',
    scaler = StandardScaler()
    scaler.fit(coinX)
    scaler.transform(coinX)
    X_train, X_test, y_train, y_test = train_test_split(coinX, coinY, tes
    return X_train, X_test, y_train, y_test


#seperating into train and test sets (notice ethCrash we are overwriting
# was only used for testing)

# X_train, X_test_l, y_train, y_test_l = seperateTarget(ethCrash)
# X_train, X_test, y_train, y_test = seperateTarget(allCoins)
X_train, X_test, y_train, y_test = seperateTarget(adaCrash)

X_train

# Convert train and test to Dmatrix for speed increase and set params
dtrain = xgb.DMatrix(X_train, y_train)
dtest = xgb.DMatrix(X_test, y_test)
param = {}
param['objective'] = 'binary:logitraw'
param['eval_metric'] = 'error'
param['tree_method'] = 'gpu_hist'
param['silent'] = 1

# Compare GPU and CPU training speed
num_round = 1000
print("Training with GPU ...")
tmp = time.time()
gpu_res = {}
xgb.train(param, dtrain, num_round, evals=[(dtest, "test")],
          evals_result=gpu_res)
gpu_time = time.time() - tmp
print("GPU Training Time: %s seconds" % (str(gpu_time)))

print("Training with CPU ...")
# set tree method back to CPU
param['tree_method'] = 'hist'
tmp = time.time()
cpu_res = {}
```

```
xgb.train(param, dtrain, num_round, evals=[(dtest, "test")],
          evals_result=cpu_res)
cpu_time = time.time() - tmp
print("CPU Training Time: %s seconds" % (str(cpu_time)))

# Plot runtime difference between CPU and GPU

# Note on this plot: the speed up with GPU comes from running large datas
# data between memory and the GPU is actually slower than between mem and
# As a result, on smaller datasets the GPU is slower, while on larger it
# quick

min_error = min(min(gpu_res['test'][param['eval_metric']]), min(cpu_res['
gpu_iteration_time = [x / (num_round * 1.0) * gpu_time for x in range(0,
cpu_iteration_time = [x / (num_round * 1.0) * cpu_time for x in range(0,
plt.plot(gpu_iteration_time, gpu_res['test'][param['eval_metric']], label
plt.plot(cpu_iteration_time, cpu_res['test'][param['eval_metric']], label
plt.legend()
plt.xlabel('Time (s)')
plt.ylabel('Test error')
plt.axhline(y=min_error, color='r', linestyle='dashed')
plt.margins(x=0)
#plt.ylim((0.23, 0.35))
plt.show()

# Actually used for analysis training cell running on GPU
num_round = 1000
tmp = time.time()
total_res = {}
modelFull = xgb.train(param, dtrain, num_round, evals=[(dtest, "test")],
          evals_result=total_res)
total_time = time.time() - tmp
print("Total Training Time: %s seconds" % (str(total_time)))

# Save the error tape after training for use in plotting
pd.DataFrame(total_res).to_csv('errorTapes/demoError.csv')


# Save model for future usage
pickle.dump(modelFull, open("models/demoModel.pickle.dat", "wb"))

# Loading models
loaded_model = pickle.load(open("models/demoModel.pickle.dat", "rb"))
loaded_model_full = pickle.load(open("models/fullModel.pickle.dat", "rb")
```

```
#predictions for loaded model
loaded_model.predict(dtest)


# Loading error tapes for plotting

#ltcError = pd.read_csv('ltcErrorTape.csv')
fullError = pd.read_csv('errorTapes/allCoinsPlusErrorTape.csv')
fullErrorjpy = pd.read_csv('errorTapes/allCoinsPlusJpyErrorTape.csv')
fullErroreth = pd.read_csv('errorTapes/allCoinsPlusEthErrorTape.csv')

# Manual destring

#ltcError = ltcError['test'][0].strip('][').split(', ')
#ltcError = [float(ele) for ele in ltcError]

fullError = fullError['test'][0].strip('][').split(', ')
fullError = [float(ele) for ele in fullError]

fullErrorjpy = fullErrorjpy['test'][0].strip('][').split(', ')
fullErrorjpy = [float(ele) for ele in fullErrorjpy]

fullErroreth = fullErroreth['test'][0].strip('][').split(', ')
fullErroreth = [float(ele) for ele in fullErroreth]


# List form error tape
fullError

# Plot various training errors for comparison
num_round=1000
total_time = 1000
min_error = min(fullError)
#min_error_ltc = min(ltcError)
min_error_fl = min(fullErrorjpy)
min_error_lf = min(fullErroreth)
gpu_iteration_time = [x / (num_round * 1.0) * total_time for x in range(0

plt.plot(gpu_iteration_time, fullError, label='Max Coins Dataset Error')
plt.plot(gpu_iteration_time, fullErrorjpy, label='Max Coins Train / JPY F
plt.plot(gpu_iteration_time, fullErroreth, label='Max Coins Train / ETH F
plt.legend()
plt.xlabel('Scaled Training Time')
```

```
plt.ylabel('Test error')
plt.axhline(y=min_error, color='b', linestyle='dashed')
#plt.axhline(y=min_error_ltc, color='orange', linestyle='dashed')
plt.axhline(y=min_error_fl, color='orange', linestyle='dashed')
plt.axhline(y=min_error_lf, color='green', linestyle='dashed')
plt.margins(x=0)
#plt.ylim((0.23, 0.35))
plt.show()

# Feature importance plot
plt.figure(figsize=(16, 12))
xgb.plot_importance(loaded_model_full)
plt.title('Top Coins Model Feature Importance')
plt.show()

# Just combining ETH pairing frames into a singular dataset

# This dataset isn't present
eth = pd.read_csv('/media/alexander/AMR_CHAMPER/FInBigData/Crashes/ioteth
eth

# datasets not present
eth = eth.append(pd.read_csv('/media/alexander/AMR_CHAMPER/FInBigData/Cra
eth

# no dataset
eth.to_parquet('/media/alexander/AMR_CHAMPER/FInBigData/Crashes/ethCrash.

# Load allCoins for testing correlation between columns
# dataset not present
allCoins = pd.read_parquet('/media/alexander/AMR_CHAMPER/FInBigData/Crash

# Intra column Correlation
# dataset not present
allCoins.corr()

ada = pd.read_parquet('testData/adausdCrash.gzip').reset_index().drop('in
ada.corr()
```

References

[1] J. Hendershott and Menkveld, “Does algorithmic trading improve liquidity?” 2011. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1111/j.1540-6261.2010.01624.x

[2] C. M. Jones, “What do we know about high-frequency trading?” [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2236201

[3] B. C. E. H. Chaboud, A. P. and C. Vega. (2014) Rise of the machines: Algorithmic trading in the foreign exchange market. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/jofi.12186

[4] W. Kenton. (2019) Flash crash. [Online]. Available: https://www.investopedia.com/terms/f/flash-crash.asp

[5] P. Tan. (2019) Ai struggles to beat financial markets, but could it beat cryptocurrency markets? [Online]. Available: https://medium.com/swlh/ai-struggles-to-beat-financial-markets-but-could-it-beat-cryptocurrency-markets-7ddd7cb6f6a1

[6] Coinmarketcap. Cryptocurrency prices by market cap. [Online]. Available: https://coinmarketcap.com/

[7] Kaggle. 400+ crypto currency pairs at 1-minute resolution. [Online]. Available: https://www.kaggle.com/tencars/392-crypto-currency-pairs-at-minute-resolution

[8] M. O. David Easley, Marcos M. Lopez de Prado, “Flow toxicity and liquidity in a high-frequencyworld,” 2012.

[9] Dask, “Why Dask?” 2020. [Online]. Available: https://docs.dask.org/en/latest/why.html

[10] Jinzhi Jiang , “Volume-Synchronized Probability of Informed Trading” 2015.

[11] Forbes, “A Massive Bitcoin Flash Crash Just Created $1 Billion Of Crypto Chaos” 2020.

[12] Patrick Tan, "AI Struggles to Beat Financial Markets, But Could It Beat Cryptocurrency Markets?" (2020), [Online]. Available: https://medium.com/swlh/ai-struggles-to-beat-financial-markets-but-could-it-beat-cryptocurrency-markets-7ddd7cb6f6a1

[13] Torben Andersen, Oleg Bondarenko, "The Trouble with VPIN" (2012), [Online]. Available: https://insight.kellogg.northwestern.edu/article/the$_t rouble_w ith_v pin$

[14] Bellia, Mario; Christensen, Kim; Kolokolov, Aleksey; Pelizzon, Loriana; Renò, Roberto, "High-frequency trading during flash crashes: Walk of fame or hall of shame?", [Online]. Available: https://www.econstor.eu/bitstream/10419/215430/1/1693370115.pdf

[15] Avanidhar Subrahmanyam, "Algorithmic trading, the Flash Crash, and coordinated circuit breakers", https://www.sciencedirect.com/science/article/pii/S2214845013000082

[16] Fry, John and Serbera, Jean-Philippe, "Modelling and mitigation of Flash Crashes", [Online]. Available: https://mpra.ub.uni-muenchen.de/82457/1/MPRA$_paper_8$2457.*pdf*

[17] CFA Institute, "Flash crashes", [Online]. Available: https://www.cfainstitute.org/en/advocacy/is crashes

[18] Ian Poirier, "High-Frequency Trading and the Flash Crash: Structural Weaknesses in the Securities Markets and Proposed Regulatory Responses", https://repository.uchastings.e

[19] Anton Golub, John Keane, and Ser-Huang Poon, "High Frequency Trading and Mini Flash Crashes", https://arxiv.org/pdf/1211.6667.pdf