Smarter Fees: Predicting the Bitcoin Transaction Fee Market

Olaoluwa Osuntokun
osuntokun@cs.ucsb.edu

Sharath Rao
sharathrao@cs.ucsb.edu

Metehan Ozten
mozten@cs.ucsb.edu

Department of Computer Science, UC Santa Barbara

***Abstract*—Transaction fees are integral to the long term security, and stability of Bitcoin. As the block subsidy asymptotes to zero, fees must make up for the loss in revenue to maintain an adequate level of security. As the fee market matures and becomes more dynamic, users will need to adapt or be unknowingly priced out. By predicting the economic climate of the future transaction fee market, we can provide estimates for users, giving them up to date market information. Existing techniques for fee estimation in the Bitcoin network lack critical network, and transaction context, causing estimates to be lagging and overly general. We present the first data-driven Machine Learning based technique for transaction fee prediction. We first develop an effective offline ensemble regressor, then adapt the technique to an online context. Our method is able to adaptively adjust to the current fee market via a novel online, rolling window based ensemble regression technique.**

***Keywords*—Bitcoin, Machine Learning, Decision Trees, XGBoost, Gradient Boosted Regreesion Trees, Random Forests.**

I. Introduction

Bitcoin represents the first true, decentralized, electronic-cash system [19]. All transaction data is recorded in a logically central, public authenticated data structure [20]. This data structure is the blockchain: an authenticated, append-only log. Bitcoin miners are custodians of the blockchain. They provide a total ordering [21] of all transactions in the blockchain to the network as a service, thereby solving the double spend problem in a global, distributed environment. In order to achieve a global rate limit of the rate at which new entries can be appended to the log, Bitcoin leverages a modified version Hashcash [14], adding inflation control. The total number of Bitcoin that we ever exist is 21 Million. An individual Bitcoin is divisible down to 100,000,000 pieces, with the ultimate constituent called a satoshi. Therefore in total the upper limit of tokens is over 2 quadrillion.

Completing enormous the Proof of Work (PoW) required in today's network requires vast amount of computing power, draining a colossal amount of electricity. As of 2015, collectively, miners within the Bitcoin network have computed over 2^{80} total SHA256 computations [11]. The task of assembling transactions, and time stamping them into the chain via PoW demands a prodigious amount of electricity, therefore miners must be incentivized to participate in the network, or else, Bitcoins security falls apart. Bitcoins security approximates the byzantines general problem [**?**], degrading into a security model of economic opportunity cost. Currently, miners who successfully solve the proof of work for a block are rewarded 25 Bitcoin in subsidy. However, this subsidy is ephemeral, and will eventually asymptote to zero. Therefore, transaction fees paid by the users of Bitcoin will need to account for the loss in subsidy. It has been shown that [5] without either a fixed Block size, or a non-zero transaction fee, Bitcoins economic security model degrades, and the currency loses all security [15].

Until now, the nature of the Bitcoin fee market has remained relatively unanalyzed. Proponents of the catalyzation of this fee market have only proclaimed its necessity, but have not analyzed the raw data available publicly. Existing techniques for transactions fee estimation [2][3] do not take into account valuable context on both the network and transaction level. This results in estimations which are overly general, and lagging. Occasionally, the Bitcoin network is flooded with bursts of traffic as agents perform spam attacks flooding the network with transactions. During such network anomalies, traditional estimation methods are unable to react, leading to in practice, useful predictions. In contrast, our method based on a sliding window of ensemble trees ensemble is dynamic, adaptive, and able to quickly adjust to changing network conditions.

II. Background

In this section we provide background for our work. First, we explain how Bitcoin transactions work. Second, we briefly introduce the current methods for transaction fee predictions. Third, we describe what is missing in the current scenario and why our solution is unique

A. Bitcoin Transactions

Within the blockchain itself, there lies an additional embedded data structure. This data structure is the transaction-graph. The transaction graph represents the flow of all funds between pseudonymous actors (bitcoin outputs). Each transaction references previously created outputs (inputs) and subsequently creates new outputs (coins). Bitcoin embeds within the vertices of the transaction graph, a distributed NP Statement verification engine: Script. Script is a stack-based scripting system closely resembling Forth. Outputs are sent to a public key script, only redeemable by the corresponding signature script: one whose arguments when prepended to the public key script and executed return True [6]. Script allows for complex non-interactive smart contract execution directly upon the Bitcoin blockchain. Required by the agreed upon consensus code, the sum of all inputs in a transaction must be less than, or equal to than the sum of all corresponding outputs within a transaction. In the case where the sum of all inputs, is less than total output

value, the remaining satoshis are implicitly rewarded to miners in the form of fees when they solve the PoW for a block.

B. Bitcoin Transaction Fee Calculation

Bitcoin blocks are currently restricted to have a maximum size of 1MB. When selecting candidate transactions for inclusion into a block, miners must select between transactions with various sizes, each paying arbitrary fees. Therefore, the problem miners encounter when selecting transactions for block inclusion is equivalent to the Knapsack Problem. Wherein the weight of a transaction is its size in bytes, and the value of a transaction are the total fees paid to a miner. Accordingly, the metric the network has adopted when quantifying the fees for a particular transaction is satoshis per kilobyte. The size of Bitcoin transactions is highly dynamic. The size of a transaction grows proportionally to the number of inputs, outputs, and the complexity of the encoded smart contract.

Most commonly in Bitcoin wallet software, the estimated fee is hardcoded to 10,000 satoshis/KB. Obviously, in a dynamic peer-to-peer network one size does not fit all. Therefore, to maintain the user experience, it is critical that users be provided with accurate, up to date fee estimates. Current techniques widely deployed for estimating fees [3], simply depend on estimates calculated via moving averages. The commonly used estimatefee RPC in Bitcoin Core answers the question: What fee did 85% of the transactions which were confirmed in N blocks pay?. Alternative mechanisms like CoinTape [2] simulate miner fee selection via Monte Carlo simulations of the contents currently in the mempool.

C. Fee Context Information

We feel that the current techniques used for estimating fees are insufficient. They do not take valuable network, and transaction context into account, are lagging, and slow to adapt to the changing network conditions. In order to properly model fee estimation, we feel that is necessary to take into account miner policy when selecting transactions. For example, in Bitcoin Core 50KB within blocks are reserved for high priority transactions. Transactions with a high priority, may be confirmed in a timely manner regardless of the fee paid. Occasionally, a very high priority transaction may even be confirmed while paying zero fees.

In addition to context information regarding transactions, context information about the network should also be taken into consideration. For example, due to the probabilistic nature of solving PoW for blocks, the time between two successive blocks is exponentially distributed. It can therefore be observed that if the last block was 30 minutes ago, getting into the next block will be more competitive than if the last block was just 30 seconds ago. Continuing, several other factors such as the mempool size, which is the total number of pending unconfirmed transaction should be taken into consideration. Such metrics allow our model to gauge the congestion and current backlog of the network.

III. Related Work

At the time of our experiments, the only published work we know of using Machine Learning within the context of Bitcoin systems use [4] Bayesian Regression techniques to predict the price variation of bitcoin. However, this problem is orthogonal to predicting transaction fees, and the estimated number of block it may take for a transaction to confirm.

There exist several solutions that provides a potential range of (number of block to confirm) given a range of satoshis per byte (the fee rate) [3]. This is the most widely deployed solution, shipped as a default RPC in Bitcoin Core. Several wallet providers also utilize their own fee estimation techniques. However, as these estimations are proprietary, and server side they are not widely published.

Developers of full node Bitcoin software are actively iterating in order to introduce new features to the network giving users more control over fees paid. One such policy is Replace-by-Fee (RBF). RBF allows users to adjust the amount of fees they pay after the fact, by allocating less money to the destination or change address and more satoshis to the implicit fee. In the process, creating a double spend, which is accepted into the mempool. Without RBF, a node employs a first seen policy, only accepting the first version of a transaction disallowing any modifications or replacement. Another emerging policy is Child-Pays-For-Parent (CPFP). Child pays for parent allows a transaction depending on another unconfirmed transaction to pay fees for the dependant transaction. If widely employed, this can remove the burden of fees from the sender to the receiver. In a possible scenario merchants could quickly confirm pending payments to them, by creating a dependant with a very high fee, allowing them to pay the necessary fees for timely confirmation.

IV. Data Collection and Initial Analysis

As our work needs context information we cannot use the dataset available in use [23]. We need the most recent data for our experiments. Therefore we modified a Bitcoin client in order to gather data from the real bitcoin network. Once the data is ready we run the prediction algorithms offline and verify the predictions .

A. Feature Selection

In the following section, we enumerate the candidate features we utilize to train our model. Each feature is accompanied by a brief explanation of their significance. Logically, our fees can be divided into two distinct categories. Features conveying transaction context, and features conveying global network context.

1) ***Size of Transaction in Bytes***. The size of Bitcoin transaction is currently used to calculate fees use [24]. A larger transaction, sans priority, will require a larger transaction fee. This is due to the fact that a Bitcoin block can only contain a fixed amount of bytes and the larger transactions (in terms of bytes) will have to fight for fixed space
2) ***Transaction Priority***. Every transaction is given a priority, so that bitcoin miners can confirm transactions with high priority first. Priority is calculating using the following formula: $\sum_{input_i}.value \cdot \sum_{input_i}.total_age$.
3) ***Total Ancestral Fee***: This is the sum of transaction fees of the dependency graph for a transaction. A transaction at the end of the graph may have a low

fee, but may still be included in a block due to the cumulative fee a package will pay to the miner.

4) ***Transaction Fee per Kilobyte***: As transaction fees closely correlated with the size of transaction, this ratio provides more useful information.
5) ***Number of Parents***: The length of the dependant transaction graph. A transaction cannot be included in a block unless all its dependencies have been included in prior blocks, or they are included in the same block. A transaction with a large dependant sent will only be included if the sum of the fees paid across the entire set is sufficiently high.
6) ***Mempool Size***: Commonly, the size of a nodes mempool will grow unbounded, being pruned when unconfirmed transactions are included in a block. Therefore this proves to be a useful metric in order to gauge the level of congestion currently in the network, However, it doesnt provide information on the actual size occupied due to transactions.
7) ***Mempool Size in Bytes***: This will capture total kilobytes in mempool. Greater the size in mempool, more difficult it is to confirm the transaction into block chain as the network would be more congested. Additionally, due to the constrained size of each block, this accurately convey the level of backlog and the number of blocks needed to clear all pending unconfirmed transactions.
8) ***Number of Transactions in Last Block***. Occasionally, a block will be mined zero transactions. This is due to miners begging to mine on a received block before theyve fully verified it, resulting in a block with zero transactions. Such a block creates additional backlog, creating additional fee pressure for the next block.
9) ***Number of Seconds Since Last Block***: This is another metric to capture the context of the bitcoin at a given time period. A larger value suggest a greater backlog of transactions, which indicates that inclusion into the next block may be more competitive than average.
10) ***Incoming Transaction Rate***: Sometimes there will be a burst transactions that will appear like spikes in a temporal graph. During this period, it will be difficult to transact in the network as agents are flooding the mempools of miners with many transactions. We measure this rate via a sliding window of 10 transactions, recording the time delta between the first and last transaction in this window before incrementing.
11) ***Total Input Value***: The total value of all inputs in satoshis.
12) ***Total Output Value***: The total value of all outputs in satoshis.
13) ***Block Difficulty***: The expected number of hashes computed in order to solve the Proof of Work for the given block.
14) ***Number of Blocks to Confirm***. Total number of blocks that was taken to confirm the given transaction. If the transaction was confirmed quickly, it either could have been a small sized transaction or a large transaction with decent fees to incentivize the miners or it could be that the network is less congested.
15) ***Transaction Fee***. Total fee in satoshis paid by a given transaction.

In our experiments we did not use Transaction Fee as a feature, as we thought transaction fees and transaction size are closely related. We used Transaction Fee per kilobyte of transaction instead.

V. Label Collection

Due to the nature of our targeted features, we are unable to use the recorded history of the blockchain as training data. Instead, we must collect these features online, to ensure that all necessary context is collected. To collect our features, we created a modified btcd [12] node. When a transaction is accepted into the mempool of the node, we record all the features such as size, and fee per KB immediately. Once a verified block arrives which includes the transaction, we can then record a delta: the number of blocks that passed before the transaction was confirmed. Additionally at the point of transaction confirmation, we can record auxiliary features such as the number of seconds since the last block, the block difficulty etc. Features were stored in Bolt DB, to be processed by additional scripts, cleaning up and formatting the data to be fed into our machine learning models. For our offline phase, we collected data over 4 weeks during November 2015. This collection period resulted in 3 million labeled features to train our initial models.

VI. Methodology

Now we will detail the high level approach we followed in order to predict the transaction fees and blocks to confirm. Our dataset contains 14 features. We split the data in 80:20 split and trained the machine learning model on the 80 set and tested it in the 20 set. Some of the machine learning algorithms supported cross validation out of the box and therefore we do not specifically talk about cross validation here.

In the first phase of our work we predicted transaction fees and blocks from four weeks worth of transaction data which is about 2.9 million records. We tried many machine learning algorithms and compared them on their prediction accuracy and execution time. Initially we used linear regression use [8] for our offline experiments. But since the performance and robustness of the linear regression is very low, we have excluded the results from it. We then used decision tree based algorithms for more accurate predictions. The three algorithms used are Gradient Boosted Regression Trees use [1] , Random Forests use [7] and Extreme Gradient Boosting (XGBoost) use [17] . We compared the performance of these four algorithms and after careful consideration of performance metrics, we decided to use XGBoost.

In the second phase of our work, we designed an online model trained with XGBoost algorithm. Essentially, the model would continuously predict the transaction fee or transaction blocks as requested by the user of the system. We created two applications for online predictions. The first application gathers the data and transfers each block worth of data to the second application via sockets. The second application keeps listening for any data transfer and trains the model for every new block. The predictions are done by the second application.

This section is divided into the following sub sections. In the first subsection, we provide more details of the offline

model we designed. In the next subsection, we talk about the machine learning algorithms. In the next subsection we talk about online model we follow and lastly we talk about the bucketized rounding to enhance accuracy of predictions.

A. Offline Model

Our offline model consists of three phases. The first phase involves having a process collect data from the Bitcoin network in the format of our choice and store it in a temporary database. In the second phase, we export the data from the node running the data collection onto a text file where it is parsed in order to be fed into the machine learning algorithm for evaluation. In the last phase, we perform various checks on the data scanning for value integrity and structural homogeneity and then run machine learning algorithms on the clean dataset. Now we shall speak of the algorithms used in our work.

1) Linear Regression: Linear Regression is a very basic machine learning technique which performs regression by fitting a convex function using gradient descent It is very sensitive to outliers and overfitting. The accuracy of linear regression can be very low if the dataset has strong nonlinear relationships use [26].

2) Gradient Boosted Regression Trees (GBRT): Gradient boosted Regression trees are a class of decision tree algorithms which produce a prediction model in the form of an ensemble of suboptimal prediction models. The model is built is multiple stages and each stage performs optimization based on an arbitrary differentiable loss function use [10]. In our experiments we used mean squared error (MSE) as the loss function. Although prone to overfitting, its prediction accuracy is very high. As each stage is dependent on the previous sub optimal tree, the algorithm cannot be easily parallelized. Therefore, it takes relatively more execution time.

3) Random Forests (RF): Random Forests are also a class of decision tree algorithms very popular today due to their parallel execution. The most effective way of using it in practice is still a topic of discussion use [18] . Random Forests build several trees during training time phase [16] and averages the predictions from each tree during testing phase. Since it averages the predictions from multiple trees, it is less prone to overfitting. The term randomness in the algorithm is due to random sampling of the dataset. In the more sophisticated versions of Random Forests, even the features are sub sampled use [7]. In this [22], work is done to budget the cost incurred due to acquisition of features from dataset. For our experiments we use the library from sklearn[7].

4) Extreme Gradient Boosting (XGBoost): XGBoost stands for eXtreme Gradient Boosting [17]. It is a type of decision tree algorithm based on Gradient Boosting algorithm. It is most popular for its efficiency and scalability. It includes both a linear model solver and tree learning algorithm. It can perform the training in parallel and therefore runs much faster than GBRT algorithm while still maintaining the latters accuracy. In the experiment section, we will also talk more about the time taken by this algorithm to train datasets of size millions.

Another important reason why we chose this algorithm is that it has its own data type which can load the data much quickly compared to other basic data types like arrays in numpy package. It also takes the data in sparse format (libsvm) use [25] therefore saving precious space and time to load data.

This is also the model we choose to use in our online rolling model described in the next section due to its quick training time and accuracy.

VII. Online Modeling

In Online Model, we will predict the transaction fee market for real time transactions that are waiting to be confirmed. Essentially, as we gather new data, before we train the model, we first predict the number of blocks a transaction will take to confirm. We will discuss the method now.

A. Rolling Model on Entire Dataset

In this model, we train a machine learning model using an offline dataset and bring it up online for predicting live transaction fees or blocks. In the background a separate process will retrain N models with the new dataset that is being predicted. We train this background model every K blocks. The value of K is to be derived from performance metrics, it can be increased as the size of our dataset increases for improved performance. In our experiments we chose the value of K as 64. Once the K new-blocks are collected we retrain the model. However, as the dataset grows, the training time would also grow making it difficult to switch over the model quickly, which is why we allow for a variable value of K, such that as the size of our training-set grows, we roll over the new model less often.

B. Rolling Model with Training Last N Blocks

In this model, we train the model with only N blocks rather than the entire dataset plus N blocks. Clearly, this model will not leverage the power of the entire data set. However, by forgetting subsequent blocks, this allows the model to focus on the newly arising context within the network. In the experiment section, we examine the effect of this loss, which we term loss of accuracy, on accuracy of the predictions.

C. Bucketization Blocks to Confirm and Transaction Fees

The machine learning algorithms predict the blocks in floating points. As the blocks are supposed to be discrete values, we round the value using half round technique. Similarly, the transaction fees are also rounded to nearest satoshis.

VIII. Experiments

A. Data Collection

The dataset was collected from the real world peer to peer bitcoin network using modified bitcoin client, btcd [12]. We used Go language to perform data collection and Bolt DB [13] to store the data intermittently. At the time of experimentation the size of the dataset was about 2.9 million records. In many of the experiments we formatted the dataset in libsvm format, which is a sparse representation for large datasets.

B. Metrics

1) Mean Squared Error (MSE): The square of the difference between the predicted value and actual value. The values in this context refer to transaction fees or blocks to confirm the transaction.
2) Root Mean Squared Error (RMSE): The square root of Mean Square Error. This metric provides some robustness against outliers. This was mainly used as a loss function for XGBoost experiments.
3) Bucketized Prediction Accuracy (BPA): It is the percentage ratio of the rounded predicted value to actual value.
$BPA = bucketize((absolute(predictedValue - actualValue)) * 100/actualValue)$
4) Execution Time: The time taken by the algorithm to train the dataset. This metric is very important because we need to train the data as quickly as possible in an online phase.

C. Machine Learning Libraries

We used the libraries from python scikit learn for GBRT use [1] and Random Forest use [7]. There is an open source python implementation of XGBoost use [27] which we used for our experiments.

D. Experiment Setting

Dataset Size	2.9 Million
Number of Trees	200
Number of Threads (for RF and XGBoost)	4
Depth of each tree	10
Training Size to Test Size Ratio	80:20

E. Comparison of Execution Time

We ran the machine learning models for prediction both the transaction fees and blocks to confirm, and we found out that the time taken independent of whether you are predicting fees or blocks to confirm. From the table it is evident that GBRT takes more time to train the data set than the other two algorithms. It is because, GBRT builds sub optimal trees in each step and the structure of each tree is dependent on the previous trees. Therefore, it cannot be parallelized. Random forests on the other hand construct forests of trees which are not dependent on each other. Therefore, it can be run in parallel and hence in lesser time, XGBoost too works with parallel threads and therefore is faster.

F. Comparison of Bucketized Prediction Accuracy (%)

In the below figure, we can see that the bucketized prediction accuracy for predicting the transaction fees is higher for XGBoost compared to GBRT and RF has the least value. The low range of values is because of two reasons. The first, is due to the number of trees we are building, is limited to 200. We have seen that increasing the trees increases accuracy as well as the training time. Similarly, we observe that XGBoost outperforms both GBRT and RF while predicting blocks to confirm. However, the accuracy is very high, because the range of values is low for predicting blocks compared to fees. More importantly, when we took a closer look at the data we saw many outliers. We will discuss that in the next section.

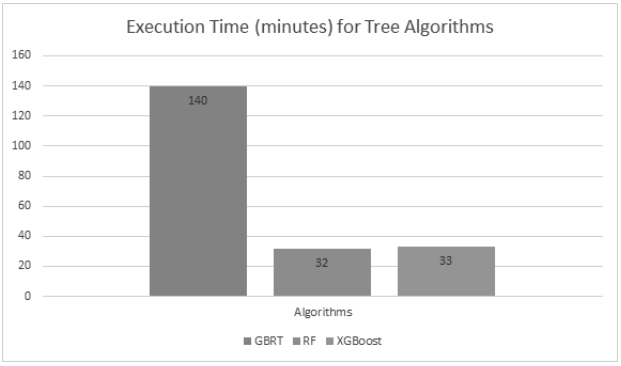

Fig. 1: Comparison of Execution Time (mins) for Decision Trees

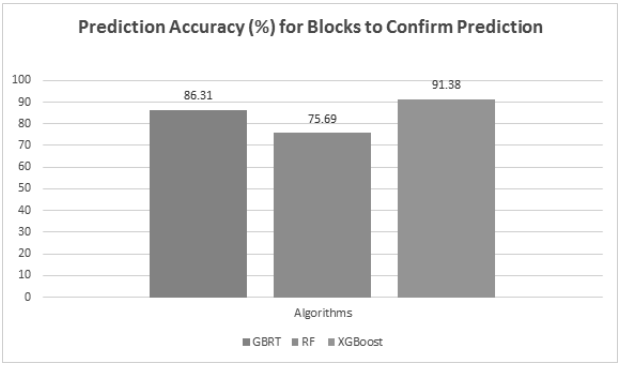

Fig. 2: Comparison of BPAs for predicting Blocks

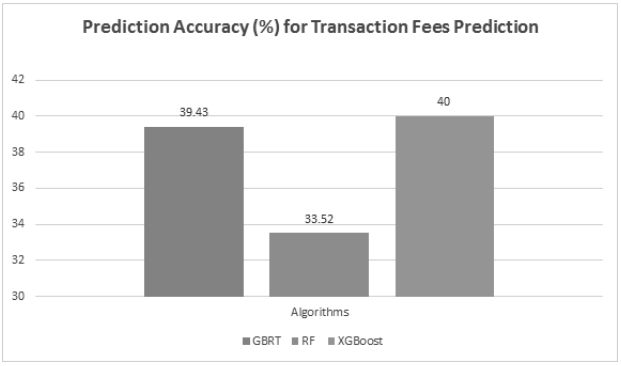

Fig. 3: Comparison of BPAs for predicting Transaction Fees

G. Impact of Outliers

We investigated the reason for low accuracy in predicting transaction fees. One of the reason it is so difficult to predict transaction fees is due to the vast range of values it takes. The minimum value of transaction fee is zero satoshis and maximum value of it is three hundred million satoshi. However, the values are not distributed uniformly. About 95% of the transaction fees are in the range of 10,000 to 15,000 per kilo byte. In a dataset of 2.9 million records the impact from the other 5% of transaction fee values is significant. We now perform new experiments where we remove these values from dataset and run the predictions again. The results are summarized in the next figure. As we can see there is a drastic

change in prediction accuracy.

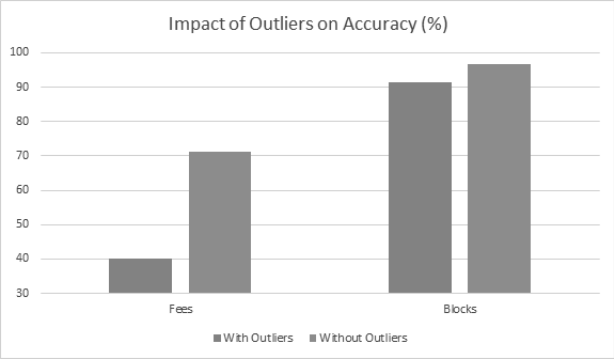

Fig. 4: Figure showing impact of outliers

H. Online Model Experiments

For the online phase of our model we utilized two custom applications. The first application is written in Go language, it's a modified btcd node which connects to the Bitcoin peer-to-peer network in order to collect data for our labels. The newly labeled is sent to the second application batched according to block number. The online machine learning application, currently has eight rolling online models. Our experiments restricted to total experiment length to only 100 new blocks. The accuracy measured from this model is reflected due to this data sparsity. However, we felt that this small window was sufficient. As shown in our online testing the accuracy will improve as the data set grows.

Every block we receive we count that as an iteration. For each block we receive we train 8 models, a historic model which uses all past data blocks for training, and 7 sliding window models which only use the last-x blocks as training data $x\epsilon\{1, 2, 4, 8, 16, 32, 64\}$. The graph below shows our percent-error (BPA) at iterations 1,8,16,24...100. Also the bar chart below shows the cumulative percent error of our models at the various sliding-window sizes over the first 100 blocks of data received. For this period of time all the models performed relatively similar. We also noticed that some blocks of data would be error-free (98%+ correct prediction rate) when being predicted by our algorithms and some would be littered with incorrect predictions where the correct prediction rate is around 3%. We suspect the reason for the existence of these bundled errors by block is primarily because of the existence of high-payers in terms of transactions where the user isnt trying to minimize transaction fee-rate in any serious manner (a course of action to correct this would be outlier detection and removal from prediction set) and because of our data-gathering node isnt picking up the transactions early enough in some cases (connection to the high-speed relay network would cause us to receive all transactions sooner to the time of original announcement) leading to less of a skew in our measured $numBlockstoConfirm$ feature. Also, different miners might be using different transaction-selection algorithms and therefore whenever a block is detected by a different mining pool a different set of transactions can potentially be chosen for different reasons.

I. Prediction using last N blocks vs entire dataset.

In this section, we discuss the comparison of predictions using entire dataset and predictions using last one million transactions. The reason we do this experiment is to understand and quantify the loss of accuracy due to using just the most recent historical data. For the transaction fees, the accuracy was down by about 2% and for blocks to confirm the accuracy was down by about 3%. This is acceptable for online modeling as the trade off by storing less data and by training the models quickly outweighs the loss of accuracy of about 3%.

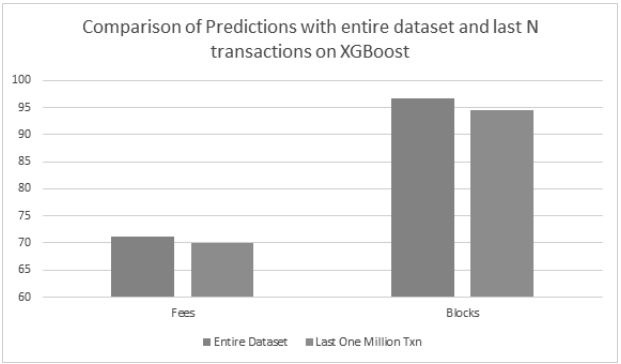

Fig. 5: Figure showing loss of accuracy due to reduced dataset

IX. Conclusion

To the best of our knowledge this is the first work which uses machine learning to predict the Bitcoin fee market by leveraging the network context in historical data. We have proposed both an offline and online model to address the lack of solutions in this space for this particular problem. Not all market actors act according to the supply-and-demand curves of the transaction fee market and do not try to minimize transaction fee or do not care about the confirmation time and because of that will greatly underpay or overpay. They might underpay if they do have a time-preference as to when the transaction will be confirmed or they might overpay if the transaction is time-sensitive and they want to guarantee their transaction will be confirmed by a certain margin. It is an open problem to predict and account for extreme behavior on either parts of this spectrum.

Such problems are important to the user experience of the Bitcoin network. Fees must be transparent to users, otherwise, they may become transparent as previously adequate fees are no longer sufficient in the fast moving Bitcion Fee Market. Future work includes: setting up an Open Source web service providing a querying interface, and adapting our techniques to gain insight into the policy of miners.

References

[1] Gradient Boosting, http://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting, 12 09 2015

[2] Cointape, www.cointape.com

[3] Bitcoin Fees http://bitcoinfees.com/

[4] Devavrat Shah and Kang Zhang, *Bayesian Regression and Bitcoin*

[?] LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE, *The Byzantine Generals Problem*, 1982

[5] Nicolas Houy *The Economics of Bitcoin Transaction Fees*, 2014

[6] Bitcoin Wiki, https://en.bitcoin.it/wiki/Script

[7] Random Forest Sklearn, http://scikit-learn.org/stable/modules/ensemble.html#forest, 12 09 2015

[8] Linear Regression Sklearn, http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html, 12 09 2015

[9] GBRT Wikipedia, https://en.wikipedia.org/wiki/Gradient_boosting, 12 09 2015

[10] J. H. Friedman, *Greedy function approximation: a gradient boosting machine* , The Annals of Statistics, Vol. 29, No. 5. (2001) Key: citeulike:9826044

[11] Pieter Wullie, http://bitcoin.sipa.be/work-ever.png

[12] https://github.com/btcsuite/btcd/

[13] https://github.com/boltdb/bolt

[14] Adam Back, http://www.hashcash.org/papers/hashcash.pdf, 2002

[15] Kerem Kakalolu, *Near Zero Bitcoin Transaction Fees Cannot Last Forever*, 2014

[16] Breiman, Leo *Random Forests. Machine Learning* ,45 (1): 532. doi:10.1023/A:1010933404324.(2001)

[17] XGBoost , https://cran.r-project.org/web/packages/xgboost/vignettes/xgboost.pdf, 12 09 2015

[18] Denil, Misha and Matheson, David and De Freitas, Nando *Narrowing the gap: Random forests in theory and in practice*, arXiv preprint arXiv:1310.1415 ,2013

[19] Satoshi Nakamoto, https://bitcoin.org/bitcoin.pdf, 2015

[20] Andrew Miller et. all, https://www.cs.umd.edu/ mwh/papers/gpads.pdf, 2014

[21] Leslie Lamport, http://research.microsoft.com/en-us/um/people/lamport/pubs/time-clocks.pdf, 1978

[22] Nan, Feng and Wang, Joseph and Saligrama, Venkatesh *Feature-Budgeted Random Forest*, arXiv preprint arXiv:1502.05925, 2015

[23] Bitcoin http://www.vo.elte.hu/bitcoin/downloads.htm, 12 09 2015

[24] Transaction Fees, https://en.bitcoin.it/wiki/Transaction_fees, 12 09 2015

[25] Chang, Chih-Chung and Lin, Chih-Jen *LIBSVM: A library for support vector machines*, ACM Transactions on Intelligent Systems and Technology (TIST), 2.3.27 2011 ACM

[26] Linear Regression Wikipedia, https://en.wikipedia.org/wiki/Linear_regression, 12 09 2015

[27] XGBoost github, https://github.com/dmlc/xgboost, 12 09 2015