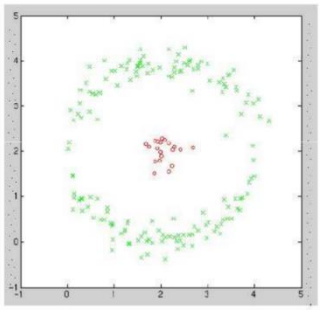
The following model estimates distribution of the data to distinguish between normal and less frequent abnormal observations. It consists from 2 steps – first transformation of features with Kernel Principal Component Analysis (KPCA) and second is density-based clustering of the observations.

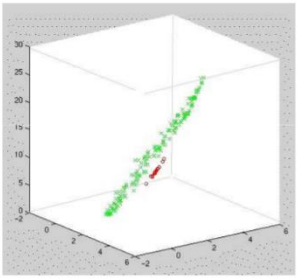
Source: Rizvi, B., Belatreche, A., Bouridane, A., & Watson, I. (2020). Detection of stock price manipulation using kernel based principal component analysis and multivariate density estimation. IEEE Access, 8, 135989–136003. https://doi.org/10.1109/ACCESS.2020.3011590

Step 1 - KPCA

Kernel function is basically a function that maps the data from d-dimensional space to d+k-dimensional space, where k can be large (or even infinite for certain choices of kernel). Thus, if in the original space distinct data groups (clusters) could not be separated by a straight line, it is possible that when they are mapped into higher dimensions, they can be linearly separated (shown in the figure below). To be more precise, if we have N points that are not separated in d < N dimensions, they can be linearly separated in d+k ≥ N dimensions.



These classes are linearly inseparable in the input space.



We can make the problem linearly separable by a simple mapping

$$\Phi : \mathbf{R}^2 \to \mathbf{R}^3$$

$$(x_1, x_2) \mapsto (x_1, x_2, x_1^2 + x_2^2)$$

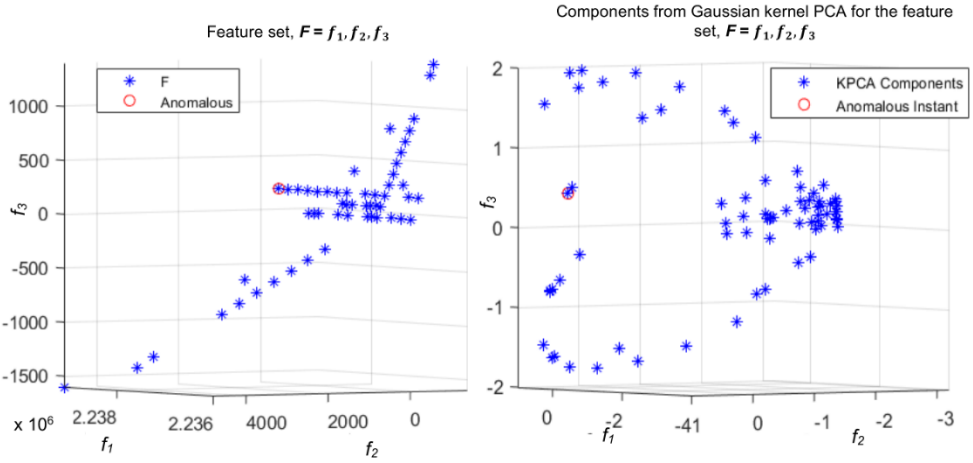
Alternatively, kernel defines similarity between a pair of observations. There are a lot of kernel functions that suit different applications and produce different mappings to higher dimensional space, so they should be chosen specifically for the task.

KPCA uses non-linear kernels to project data into higher-dimensional space that is non-linearly related to the original space. To implement it, the data should be centered (have zero mean and standard deviation of 1), otherwise the results might be misleading. First, the data is projected into high-dimensional space with kernel functions. Depending on the choice of kernel, in this projection anomalous points can be made more distinguishable (more distant) from normal ones. That is, the

distance between observations with high and low density is amplified, in order to make separation between normal and abnormal points easier.

Then Principal Component Analysis (PCA) is applied. In PCA data is normally projected into lower dimensions. This projection is called principal components (PCs) (there are several of them), which are linear combinations of original features with different weights, called loadings. If we have p features initially, then PCA produces M ≤ p PCs, where M is specified by the researcher. The intuition is that the 1st PC captures the largest amount of variance in the data, 2nd PC – the 2nd largest amount and so on. When estimating subsequent PCs, the requirement that data should vary the most along its vector of loadings still holds, but it is also necessary that each PC is orthogonal to the preceding ones. Thus, all PCs are uncorrelated.

However, in the article the authors do the contrary: they map 5 features into 7 PCs via KPCA. So, they first increase the distance between normal and abnormal observations via the use of kernels, and then represent data in even higher dimensions to make observations more distinguishable and more easily separable. They also claim that using kernels for non-linear mapping of the data allows to account for structure and patterns in it.



The result after applying KPCA. Left – initial data, right – data after KPCA.

Step 2 – clustering principal components

Clustering is applied to the resulting PCs. It is based on density estimation and does not require specifying the number of clusters (data groups) beforehand. Some points are not assigned to clusters, and we label them as manipulative trades. Further I describe an algorithm which is developed by authors on the basis of some preexisting algorithms. In univariate case, the mean of dataset is calculated, and the points, for which the difference between the mean and their values is less than some pre-defined threshold, are grouped into one cluster. For the remaining points, the new mean is calculated and threshold is chosen, and the process continues similarly. In multidimensional case, several thresholds (one for each dimension) are chosen and several means are calculated. The process is analogous to the single dimensional case, but while grouping points, all dimensions are accounted for. The problems with the algorithm, as mentioned by authors, are as follows. Since in larger number of dimensions points can be more distant from each other than in smaller number of dimensions, in multidimensional case small dataset may yield many clusters

(because the points are distant and so form their own cluster). Besides, some observations may belong to more than one cluster, in which case these clusters should be merged into one (given that the point is not anomalous). Another problem arises when there are many anomalous points, so they form a separate cluster. This conflicts with the idea that we define unclustered points as manipulative. However, this situation can be avoided by using robust features and appropriate sample size, which should not be too big to contain many anomalous points (so it makes sense to use sliding window of relatively small size).

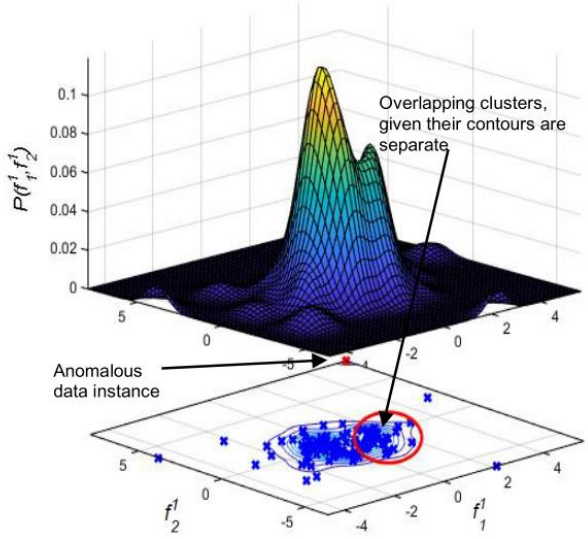
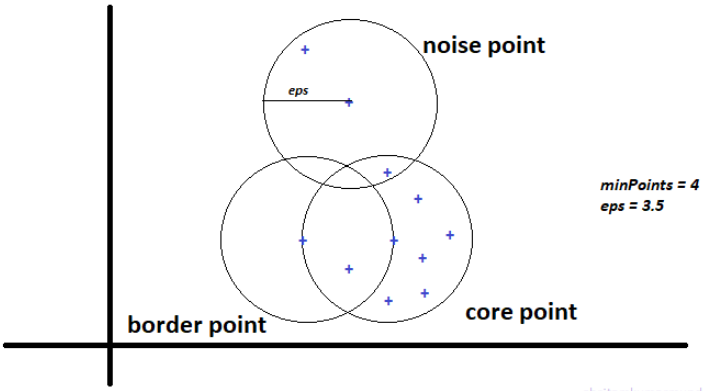


FIGURE 5. Probability distribution using kernel density estimate for a 2-D feature set $\epsilon \{f_1^1, f_2^1\}$ of Apple Stock for 100 data points along with its contour

On the picture above is the idea of clustering algorithm of authors – normal points have higher density and form a cluster, anomalous – have low density and are distant from other points.

Since this algorithm is developed by authors, I have not found its implementation neither in libraries nor on open-source websites, so I propose to use a widely used algorithm called DBSCAN, which has a similar idea. This algorithm leaves some points unclustered. It has two parameters: *eps*, which is the maximum distance between points to be in the same cluster, and *minPoints,* the minimum number of points required to form a cluster.

The basic idea of the algorithm is that when we plot a dataset, we can spot 3 types of points: core points, border points and noise points. Core point has more than *minPoints* observations within a radius of *eps*. Border point has less than *minPoints* observations within a radius of *eps*, but lies in the neighborhood of *eps* of a core point. Noise points are the remaining points.



First, the algorithm groups all core points into clusters. That is, if two core points have distance less than *eps*, then they are merged into cluster. Then, border points are assigned to the cluster which corresponds to the closest core point. Noise points are not assigned to the clusters, so they represent some kind of outliers – anomalous trades.

Comparison with Hidden Markov model

The authors compared their method to Hidden Markov model (HMM) and found that their result has larger number of correctly detected manipulative trades. For comparison the authors use Area Under Curve (AUC) metrics, which is 1 for the perfect model and 0.5 for model that makes random predictions. Overall, KPCA has AUC of 0.91 and HMM – of 0.73. In the original article on HMM, AUC varied from 0.73 to 0.89.

The authors of clustering approach also define a drawback of HMM model – it requires the data to come from certain distribution, which is not an easy task to define. In particular, the authors of HMM model assume multivariate normal distribution, an assumption which clearly fails in reality (I checked the prices for normality previously but did not include it in the text, since it is rather obvious that they are not normal).

Features

The authors of both HMM and clustering models note that "high frequency components in financial data are more prone to manipulation activities"[1], so they are more indicative of manipulation. They use different methods from the area of signal processing, in particular wavelet transform (a denoising technique) to extract high frequency component from the price series. To remove trend and seasonality of the data, they also include several other features such as differences and derivatives of the price and high-frequency component, along with price series itself.

We need to include relatively small number of features. If clustering model is used, then the number of features will be enlarged by projecting into higher-dimensional space. HMM is a model that initially has not been developed for multivariate case, so large number of features will lead to significant increase in computations. I propose the following variables (although probably the list should be reduced):

- The authors use prices and their derivatives, but due to non-stationarity of prices it would be better to use returns. However, the authors of HMM model propose a trick of how to use non-stationary data relatively safely – they suggest to use a sliding window. I am not quite sure if this can remedy non-stationarity. So I suggest using stationary returns and volume and its derivative with respect to time. Also we can extract high-frequency component with wavelet transform and use it and possibly its derivative as inputs.

[1] Rizvi, B., Belatreche, A., Bouridane, A., & Watson, I. (2020). Detection of stock price manipulation using kernel based principal component analysis and multivariate density estimation. IEEE Access, 8, 135989–136003. https://doi.org/10.1109/ACCESS.2020.3011590

- Realized volatility – the sum of squared returns. As has been claimed by several authors, volatility rises in times of manipulation, so this feature may be quite important.
- There is low liquidity before manipulation period, since it is easier then to move price. However, in times of manipulation liquidity rises. So we could calculate liquidity ratio - volume necessary to result in 1% price change. (Dimpfl, T. (2017). Bitcoin Market Microstructure. SSRN Electronic Journal)

$$LR = \frac{\sum_j \sum_t p_{j,t} V_{j,t}}{\sum_j \sum_t (|\Delta p_{j,t}| \cdot 100)}$$

where $p_{j,t}$ is the transaction price on day j at time t, $V_{j,t}$ the corresponding Bitcoin volume, and $\Delta p_{j,t} = p_{j,t} - p_{j,t-1}$.

- Liquidity ratio captures volume aspect of liquidity and we can capture price aspect of liquidity by calculating quoted spread:

$$\text{Quoted Spread} = \frac{\text{ask} - \text{bid}}{\text{midpoint}} \times 100$$

- Order imbalance, since several strategies (such as spoofing or ramping) are based upon it.

$$Order\ imbalance = \frac{bid\ volume - ask\ volume}{bid\ volume + ask\ volume}$$

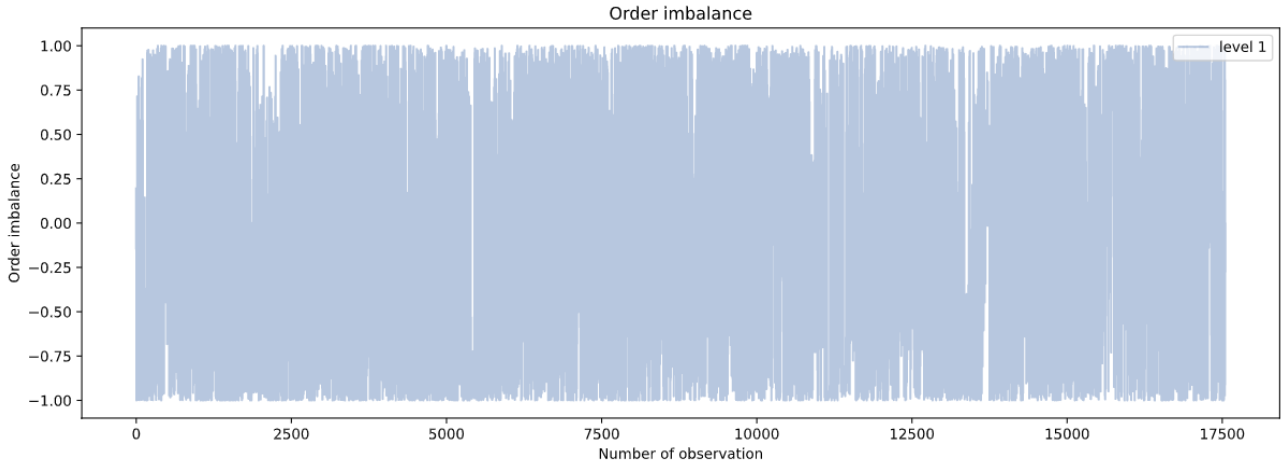
Articles by financial experts

I have searched for similar articles from financial experts on the topic, but there are only few of them. In particular, I have not found any where DBSCAN or KPCA are used. What concerns application of clustering method, I found two articles where authors use Gaussian Mixture model. These are a research by financial services company IHS Markit (*Using a "Human + Machine" Approach to Determine Financial Market Regimes*) and article *Gold and silver manipulation: What can be empirically verified?* by J. A. Batten, B. M. Lucey and M. Peat. IHS Markit experts used clustering for a different purpose – they clustered implied volatility according to the market state (crisis, transition, etc.) it came from. However, they also encountered the problem of how to assess a model, since their data was also unlabeled. They propose two methods: visual inspection and computing silhouette coefficient (the one I wrote about in the previous email). The second article uses Gaussian mixture model to detect manipulation in gold and silver markets. This model consists in finding parameters for a mixture (combination) of normal distributions, so that the estimated density is most likely to have generated the data. The parameters are chosen for each normal distribution in the mixture. These normal distributions are called components. The authors find the best result with 3 components, and they obtain parameters for these components and their weights in the mixture. The component with largest volatility, smallest weight and lowest mean is claimed to mark manipulative trades. When a trade arrives, a Bayes classifier assigns this trade to the component by which it is most likely to be generated. Then the authors separate the component to which most observations are assigned and call it control group of normal trades. They run several statistical tests, comparing statistics of control group and of manipulative component.

Finally, there is a comprehensive overview of manipulation detection methods in various areas (*Statistical Methods for Fighting Financial Crimes* by Sudjianto, Agus, et al.). The authors briefly describe the models and then show their application on some tasks. In particular they used Hidden Markov model for detection of checking account overdraft fraud. They also mention PCA and density-based outlier detection, similar to DBSCAN.

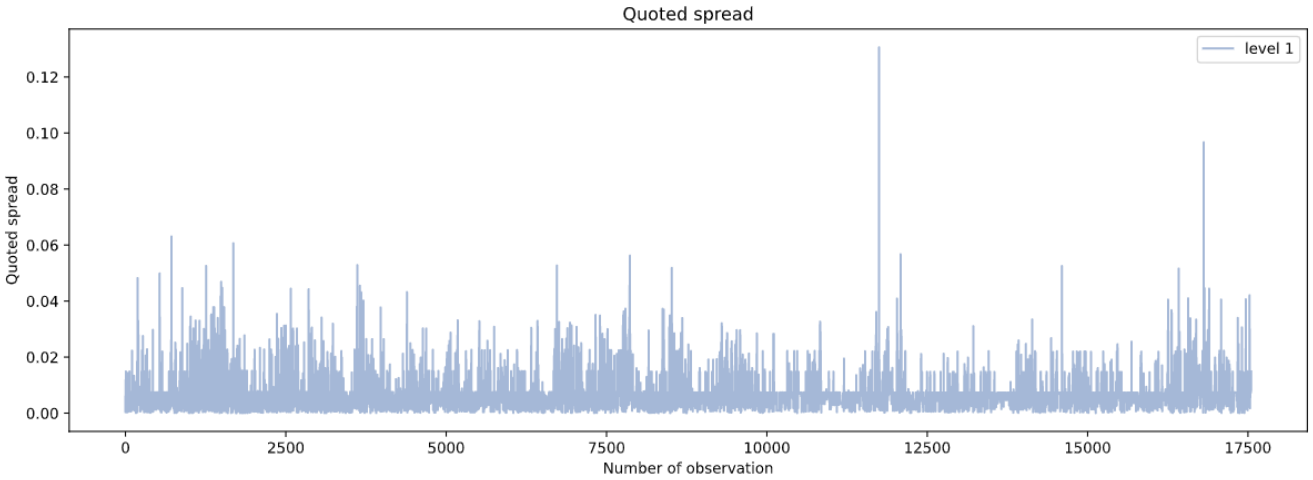
Features

Order imbalance = $\frac{bid\ volume - ask\ volume}{bid\ volume + ask\ volume}$

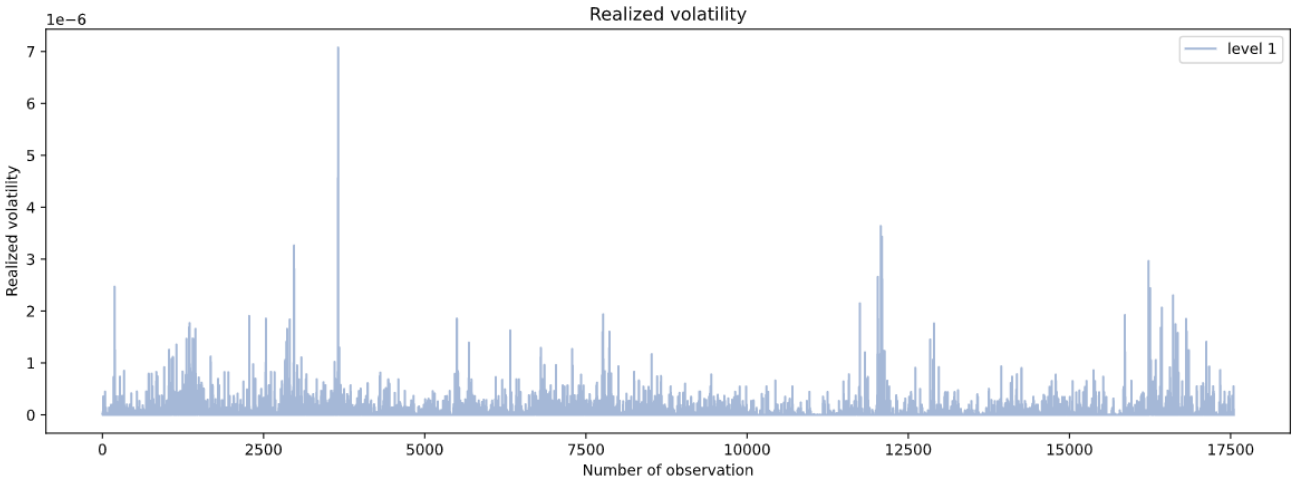


Quoted spread

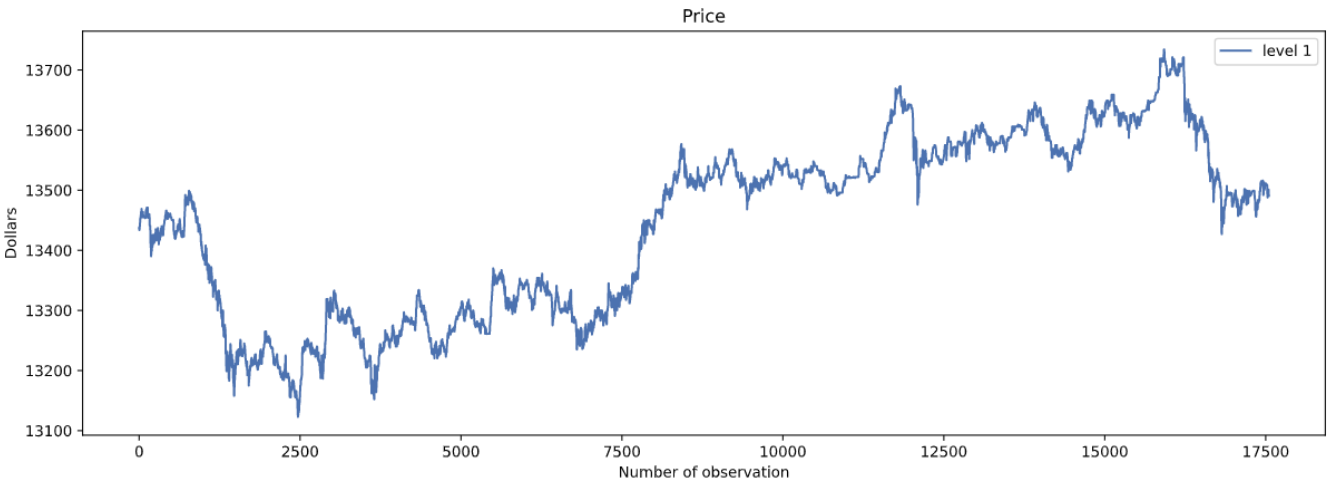
$$\text{Quoted Spread} = \frac{\text{ask} - \text{bid}}{\text{midpoint}} \times 100$$



Realized volatility – proxied by squared returns

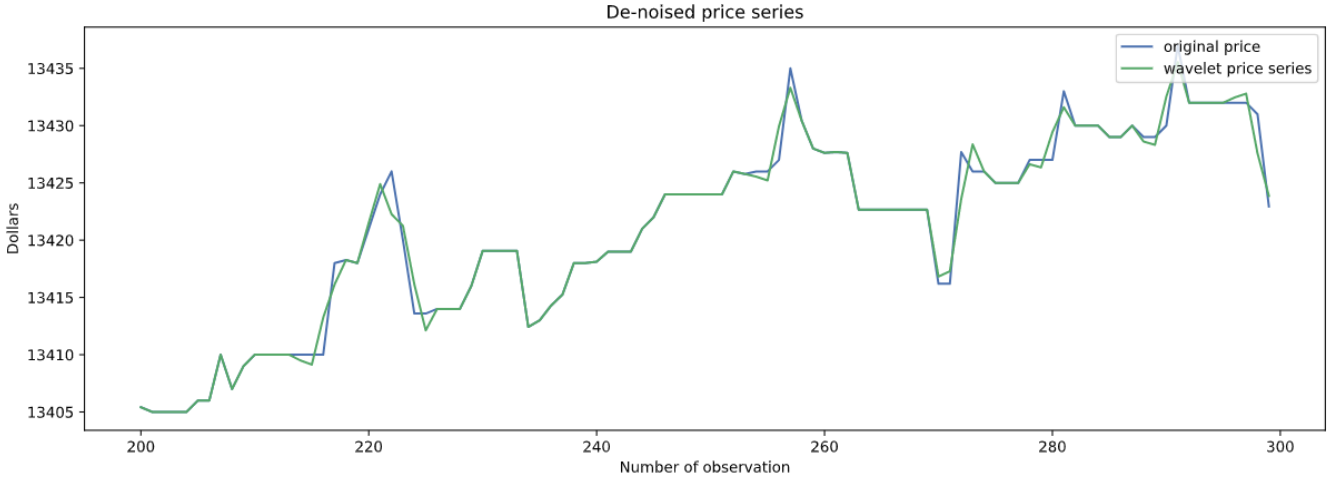


Price

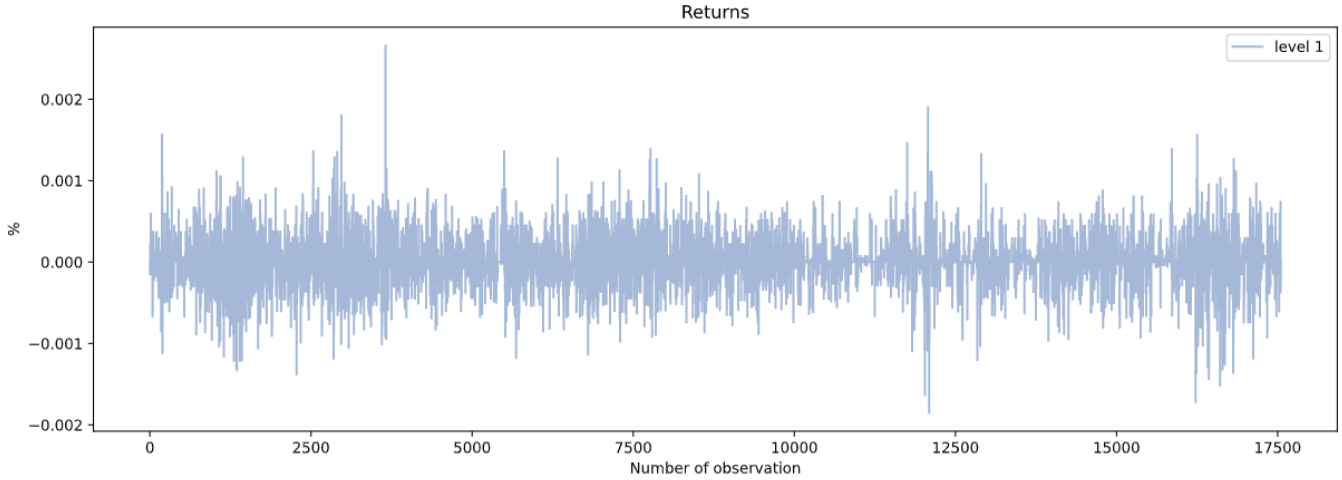


De-noised price (with discrete wavelet transform)

De-noised time series constructed with wavelets is smoother than the original one:



Returns of original price



Returns of de-noised price

Motivation

A noisy signal X, in our case it is BTC price, can be represented as a composition of Gaussian noise E and original signal S:

$$X = E + S, E \sim N(0, \sigma^2)$$

If wavelet transform is applied to X, then it produces one set of coefficients from S and other set from E. Large coefficients would likely be produced by original signal, and small ones – by noise.

Apart from being decomposed into noise and original signal, time series can be decomposed into low and high frequency components. It is claimed that high frequency component is more prone to manipulations activities. This is because manipulation causes prices to change dramatically, and fast change in a function is represented by high frequency component.

Manipulation usually triggers short-term price fluctuation around equilibrium price level. This oscillation is usually seen as data contamination. However, for investigation of time series it is more helpful to know true signal without noise. By applying discrete wavelet transform, we can receive true signal from high frequency component and remove additional noise. Discrete wavelet transform is a time-frequency representation of discretely sampled signal, which is continuous in time. This transform produces what is called detail and approximate coefficients, representing high and low frequency components, respectively.

Universal threshold method

Taking the detail coefficients, we can transform them via universal threshold method to remove noise from the series. For this, we need to find a threshold value, below which each coefficient will be set to zero. If the threshold value is too small, then considerable noise remains, and if the threshold value is too large, then some important feature of signal may be filtered out. The following formulas are used to determine threshold λ (W is detail coefficients, σ is average variance of noise and N is the length of the input signal – of time series):

$$\sigma = \frac{median\ (|W|)}{0.6745}$$

$$\lambda = \sigma\sqrt{2\ln\ (N)}$$

Then hard thresholding is applied to the detail coefficients. This method is used both by the authors of clustering and Hidden Markov models, but they apply it differently. In clustering model with DBSCAN and KPCA a more classical approach is used which is to remove small coefficients:

$$\widehat{W} = \begin{cases} W, & |W| \geq \lambda \\ 0, & |W| < \lambda \end{cases}$$

Since noise is usually represented by small coefficients, it is removed by hard thresholding. However, in paper on Hidden Markov model the authors remove large coefficients:

$$\widehat{W} = \begin{cases} W, & |W| \leq \lambda \\ 0, & |W| > \lambda \end{cases}$$

By this they claim to remove noise and low frequency components, which is not consistent with the theory. Noise is retained since it is represented with small coefficients, and low frequency component is not

removed since the method is applied to high frequency component, which is a separate set of coefficients. So, from my limited experience with wavelets, the first approach seems to be more consistent with theory.

Next, the time series is reconstructed from obtained coefficients using inverse wavelet transform, so we receive a series of prices without noise in high frequency component. The authors do not use universal threshold method for low frequency component, possibly because it does not contain useful information for manipulation detection.

Difference with Fourier transform

Fourier transform does not capture time information, it only captures information about frequencies. On the other hand, wavelet transform captures frequency information and also tells when these frequencies occur in time. So, if a signal varies with time, Fourier transform does not capture it, whereas wavelet transform does. Besides, it is claimed that wavelet transform is better for using with finite, non-periodic and/or non-stationary signals, which is likely to characterize price series. What concerns mathematical representation, wavelet transform uses wavelet functions, which have limited domain, while Fourier transform uses sin(x) and cos(x) functions with unlimited domain, and this difference allows wavelet transform to capture time information.

On kernel choice

The authors mention in the paper which kernel they use - it is a modification of Gaussian process kernel, which is a version of widely used radial basis function (RBF) kernel, often used as default due to its good empirical performance. Alternatively, we could try fitting different kernels (and our choice is restricted to nonlinear kernels only) and evaluate the performance of the model by the technique called cross validation. Cross validation consists in dividing the dataset into K parts, and iteratively training the model on all but i-th part, which is used for performance evaluation, where i ranges from 1 to K. However, the performance will be affected not only by the choice of kernel, but also by chosen number of principal components, as well as parameters for DBSCAN clustering algorithm. So it is possible to tune the best combination of parameters for the whole model but we cannot assess the choice of kernel function alone. Also, sometimes the choice of kernel depends on the task, since some kernels are more suitable for specific cases (neural networks, regression, sparse data, etc.). RBF kernel is suitable when there is no prior knowledge about the data (as in our case). To conclude, we do not know a priori which kernel is the best, but we can try to approximate it by the kernel which performs best on the data.

On model assessment

clustering is based on the idea that the distance (similarity) between the points inside the same cluster should be minimized, while distance between points in different clusters should be maximized. There are some performance metrics that are based on this.The most widely used ones, Davies-Bouldin and Dunn indices and silhouette coefficient, are discussed briefly here: https://medium.com/@ODSC/assessment-metrics-for-clustering-algorithms-4a902e00d92d. If one of the anomalous trades is included into a cluster with normal ones, then the performance will drop, since these trades are not similar. By optimizing these metrics, we could separate anomalous trades from the clusters of normal trades.

Alternatively, sometimes when authors do not know which trades are really manipulative, they generate artificial instances of such trades and insert them into the data. However, they do not provide a description of how exactly such trades are generated. If no public data on manipulative trades for the period of our data is available (or for the period in which BTC price had similar levels), then the generation of trades becomes rather subjective. For instance, we could generate manipulative trades with extreme values of some features, whereas in reality manipulative trades can have medium values of features but some unusual combinations of them. Even if we took some known manipulation cases in BTC market in the past, it is not guaranteed that the same manipulation patterns can be present in our dataset. For instance, now BTC fluctuates a lot, and traders can undertake non-manipulative strategies by placing orders with more extreme price/volume compared to the usual levels, whereas several months ago if they placed orders with similarly extreme prices/volumes, then their trades could look suspicious, since there was no such fluctuation in the market. So I do not clearly understand how to implement this approach without introducing bias into the model.