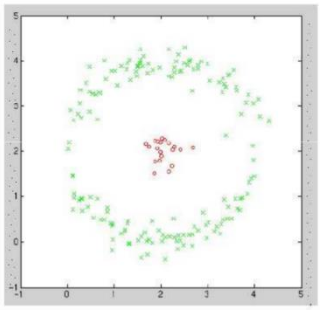
The following model estimates distribution of the data to distinguish between normal and less frequent abnormal observations. It consists from 2 steps – first transformation of features with Kernel Principal Component Analysis (KPCA) and second is density-based clustering of the observations.

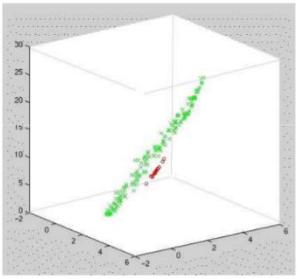
Source: Rizvi, B., Belatreche, A., Bouridane, A., & Watson, I. (2020). Detection of stock price manipulation using kernel based principal component analysis and multivariate density estimation. IEEE Access, 8, 135989–136003. https://doi.org/10.1109/ACCESS.2020.3011590

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

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

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

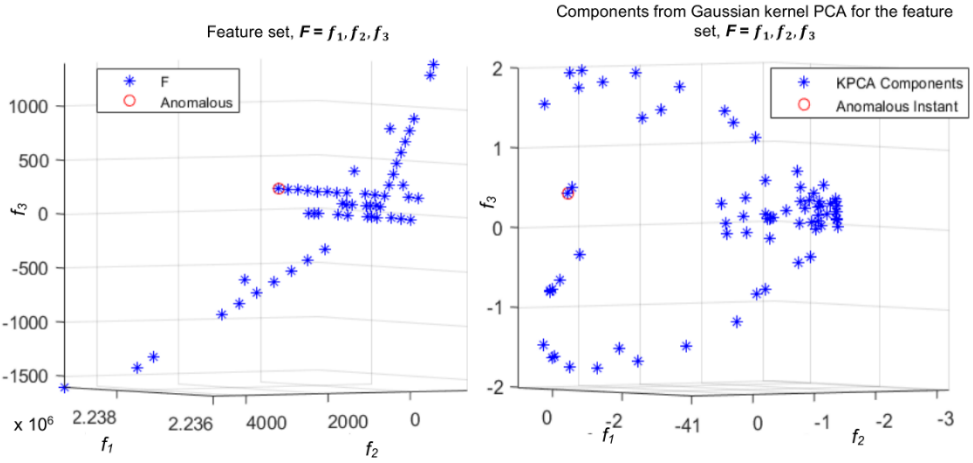
Alternatively, kernel defines similarity between a pair of observations. There are a lot of kernel functions that suit different applications and produce different mappings to higher dimensional space, so they should be chosen specifically for the task.

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).

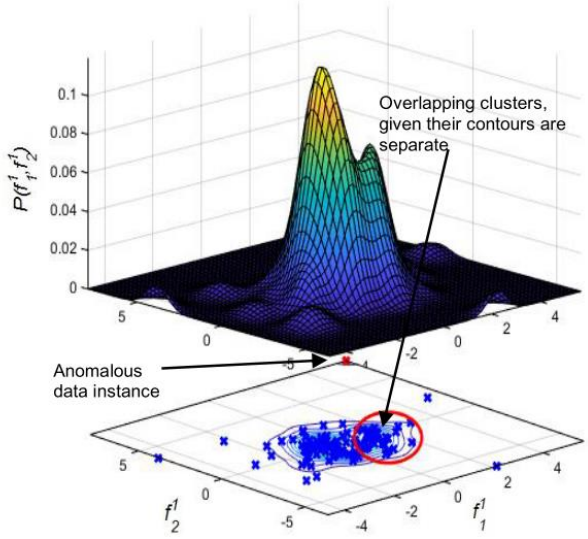
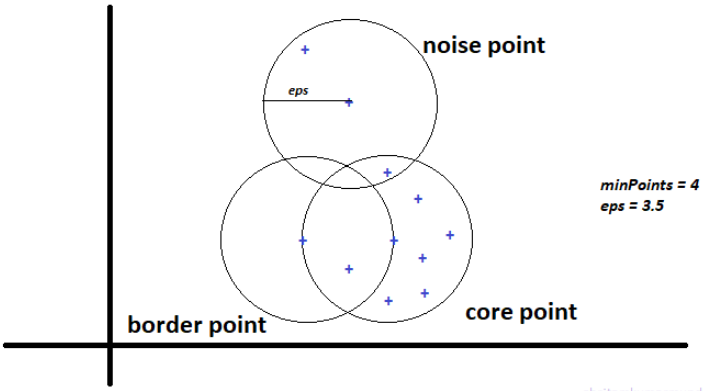


FIGURE 5. Probability distribution using kernel density estimate for a 2-D feature set $\epsilon \{f_1^1, f_2^1\}$ of Apple Stock for 100 data points along with its contour

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.

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