Anomaly Detection with Machine Learning and Graph Databases in Fraud Management

Article *in* International Journal of Advanced Computer Science and Applications · January 2018

DOI: 10.14569/IJACSA.2018.091104

6 authors, including:

Shamil Magomedov
Moscow State Institute of Radio Engineering, Electronics and Automation
20 PUBLICATIONS **51** CITATIONS

Anomaly Detection with Machine Learning and Graph Databases in Fraud Management

Shamil Magomedov[1], Sergei Pavelyev[2], Irina Ivanova[3], Alexey Dobrotvorsky[4], Marina Khrestina[5]
Department «Automated control systems»
MIREA – Russian Technological University
Moscow, Russian Federation

Timur Yusubaliev[6]
Quality Software Solutions ltd Moscow, Russia

***Abstract*—In this paper, the task of fraud detection using the methods of data analysis and machine learning based on social and transaction graphs is considered. The algorithms for feature calculation, outlier detection and identifying specific sub-graph patterns are proposed. Software realization of the proposed algorithms is described and the results of experimental study of the algorithms on the sets of real and synthetic data are presented.**

***Keywords*—Data analysis; machine learning; graph database; fraud detection; anti-money laundering**

I. Introduction

At present fraud is a major threat that is increasing every year. The global economic crime survey of 2018 carried out by PricewaterhouseCoopers [1] found that almost half (49%) of the 7,200 companies they surveyed had experienced fraud of some kind. Experts from HSN Consultants predict online credit card fraud to soar to $32 billion in 2020 [2]. Beside direct financial losses, fraud also affects customer loyalty and conversions in both digital and physical environments. For instance, 20% of customers change their banks after experiencing frauds. Meanwhile, manual review remains prevalent among the means of fraud detection. According to the annual Fraud Benchmark Report by CyberSource [3] 79% of North American businesses conduct manual reviews, and on average, these businesses manually review 25% of orders. At the same time, the survey found that these businesses accepted 89% of orders following manual review. This means that more orders are subject to manual reviews than might be necessary. Since manual review is usually the costly aspect of fraud management operations, automated screening could make fraud management processes more efficient by leaving only the most suspect orders to manual reviews.

Machine Learning technologies have shown their effectiveness in solving such tasks as spam detection, image recognition, product recommendation, predictive analytics etc. In fraud management, Machine Learning can be used to predict fraud in a large volume of transactions by applying cognitive computing technologies to raw data. The prediction problem can be further divided into two types of tasks: classification and regression. Regression analysis is a popular, longstanding statistical technique that measures the strength of cause-and-effect relationships in structured data sets. Regression analysis tends to become more sophisticated when applied to fraud detection due to the number of variables and size of the data sets. It can provide value by assessing the predictive power of individual variables or combinations of variables as part of a larger fraud strategy. According to this technique, the authentic transactions are compared with the fraud ones to create an algorithm, which will then predict whether a new transaction is fraudulent or not. Classification problem can be solved with the help of Decision Tree algorithms. They are essentially a set of rules that are trained using examples of fraud that clients are facing. The creation of a tree ignores irrelevant features and does not require extensive normalization of the data. By inspecting a tree, it is possible to understand why a decision was made by following the list of rules triggered by a certain customer. Random Forest technique uses a combination of multiple decision trees to improve the performance of the classification or regression. It allows smoothing the error that might exist in a single tree and increases the overall performance and accuracy of the model while maintaining the ability to interpret the results and provide explainable scores to the users. Random forest runtimes are quite fast, and they are able to deal with unbalanced and missing data. Random Forest weaknesses are that when used for regression they cannot predict beyond the range in the training data and that they may over-fit data sets that are particularly noisy. Neural networks can be an excellent complement to other techniques, which improves with exposure to data. The neural network is a part of cognitive computing technology where the machine mimics how the human brain works and how it observes patterns. Neural networks can adapt to the change in the behavior of normal transactions and identify new patterns of fraud transactions. Data processing by neural networks is extremely fast which makes it possible to make decisions in real time.

Due to growing popularity of machine learning, many innovative enterprises are starting to implement these techniques in their fraud management processes. For example, PayPal uses a homegrown AI engine built with open-source tools to detect suspicious activity, and more importantly to separate false alarms from true fraud [4]. PayPal implements express assessment using linear models to separate uncertain transactions from ordinary ones. Then, all transactions that look suspicious are run through an ensemble of three models

comprising a linear model, a neural network, and a deep neural network. The three models then vote to arrive at the result with the higher accuracy. With the help of this human and AI solution, PayPal has decreased its false alarm rate to half. MasterCard integrated machine learning and AI to track and process such variables as transaction size, location, time, device, and purchase data [5]. The system assesses account behavior in each operation and provides real-time judgment on whether a transaction is fraudulent. The project aims at reducing the number of false declines in merchant payments. Feedzai [6], a FinTech company, claims that a fine-tuned machine learning solution can detect up to 95% of all fraud and minimize the cost of manual reconciliations, which accounts now for 25% of fraud expenditures. Capgemini [7] claims that fraud detection systems using machine learning and analytics minimize fraud investigation time by 70% and improve detection accuracy by 90%. These facts prove the benefits of using machine learning in anti-fraud systems. On the other hand, banks have been slow to adopt machine learning and AI solution at a large scale. The reasons for this include high infrastructural costs, strict regulations and risk of replacing existing technology.

Machine Learning technologies also have their own limitations. One of such limitations is their blindness to connections in data when the initial data set is relatively small. Machine learning models work on actions, behavior, and activity. For example, the model can overlook a seemingly obvious connection such as a shared card between two accounts. To counter this machine learning models can be enhanced with Graph databases. Graph database addresses Gartner's fifth layer of fraud prevention: entity link analysis [8]. Graph database allows looking beyond the individual data points of discrete analysis to the connections that link them. Thus, graph technique can find multiple bogus actors for every single one prevented through scoring. Graph databases allow blocking suspect and bogus accounts before they have taken any fraudulent action. Another important trait that makes graph database a valuable addition to any fraud prevention solution is its inherent speed in calculating relationships. Since the relationships in graph database are treated with as much value as the database records themselves, the engine that navigates the connections between nodes can do so efficiently, enabling millions of connections per second. Graph database enables quick extraction of new insight from large and complex databases to help uncover unknown interactions and relationships.

II. Anomaly Detection Algorithms for Graph Structures

A. Local Outlier Factor (LOF) Algorithm based on Local-Sensitive Hashing (LSH) Method

LOF is an outlier detection algorithm that calculates certain numeric value for each point, which allows identifying the point as normal or anomaly. LOF value close to one corresponds to normal points; otherwise, the points are considered anomalies. Exact threshold for anomaly detection is set after conducting data analysis. An algorithm for calculation of LOF based on LSH method has been developed. Pseudo code for the developed algorithm is listed below:

```
Input: points // set of all points
Output: result // set of nearest neighbors for each point
result = 0
nv = StartNumVectors
// creation of hash-table
hash_table = 0
for 1..NumTables do
    // clearing hash-table for each iteration
    hash_table = 0
    hash_vecs = get_random_vecs(nv)
    forall p ∈ points do
        hash = get_hash(hash_vecs, p)
        hash_table[hash] += p
    foreach cell ∈ hash_table, cell.size < const do
        forall point ∈ cell do
            result[point] += all points ∈ cell without the
point
    forall point ∈ result do
        result[point] = save_only_best_kNN_neighbors
    nv = nv + 1
foreach cell ∈ hash_table, cell.size > const do
    forall point ∈ cell do
        result[point] = save_any_kNN_neighbors ∈ cell
forall point ∈ result do
    result[point] = save_only_best_kNN_neighbors
foreach point : result[point].size < kNN do
    result[point] = brute_force[point]
```

LSH method was chosen due to necessity of fast identification of nearest neighbors for each point. The main principle of LSH method involves using special method of hashing when hash values are equal for the points close to each other. A set of hash-tables (*NumTables* = 100) has been generated with a set of vectors for each of those tables. For the first table a number of random vectors (*StartNumVectors* = 3) has been generated. For each subsequent table the number of random vectors increases by one with each iteration (number of iterations corresponds to the number of tables, one table is processed per iteration). Each vector consists of *d* random values generated according to normal distribution with mean value of zero and standard deviation of one. Hash is calculated for each point as a bit sequence where bit *i* equals one if scalar product of vector *i* of the processed table and the vector corresponding to the point in question (the point can be considered a vector) is equal or greater than 0. Otherwise bit *i* equals zero. Thus in each iteration of the algorithm all points are distributed among the cells of current hash-table.

Then the cells of current hash-table are considered with the size lower than *const* (4**kNN* in this example, where *kNN* is the number of nearest neighbors). For each point in such cells, all other points in this cell are added to the set of candidates for the point in question. Finally, for each point, duplicate candidates are removed and the nearest *kNN* neighbors are left.

At the last iteration, the cells with the size larger than *const* are considered and for each point in such cells *kNN* random points from the same cell are selected, duplicate candidates for each point are removed and the best *kNN* candidates are left. For each point, random *kNN* candidates are selected, because when the number of random vectors used for calculating hash is large enough each cell corresponds to a small part of n-dimensional space, which means all points from the same cell are close to each other. In the end for each point with the number of neighbors lower than *kNN* a naïve algorithm is used since the number of such points at the last iteration should be very small. In current realization, *kNN* was set equal to 10.

B. *"Volcano" and "Black Hole" Patterns*

So-called "volcano" and "black hole" patterns were described in [9]. "Black hole" refers to a sub-graph, which has only incoming edges from the vertices of the graph not included in this sub-graph. "Volcano" refers to a sub-graph which has only outgoing edges to the vertices of the graph not included in this sub-graph. The task of identifying "volcanoes" is inverse to the task of identifying "black holes".

An example of a "volcano" and a "black hole" is shown in Figure 1.

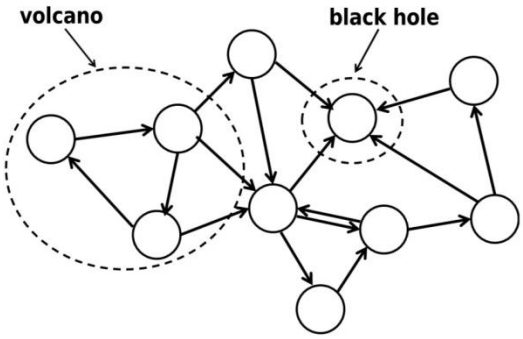

Fig. 1. An example of a "volcano" and a "black hole".

The task of identifying "volcanoes" and "black holes" is a combinatorial problem. In [9] two algorithms based on pruning schemes are proposed.

C. *Algorithm for Identifying "Volcanoes" and "Black Holes"*

An algorithm for identifying "black holes" has been developed. For identifying "volcanoes", the direction of all edges of the graph should be reversed and the same algorithm should be applied.

The set of *ancestors* for the vertex *v* is defined as a set of all vertices having at least one edge outgoing to *v*, as well as all ancestors of those vertices. The algorithm identifies "black holes":

- with diameter equal or lower than (*MaxIterCount*) and
- with the number of ancestors for each vertex in the sub-graph lower than *MaxSetSize* (considering only ancestors at the distance equal or lower than *MaxIterCount*).

The algorithm is described using vertex-centric [10] approach, but practical realization is carried out using resilient distributed dataset (RDD) API [11] without Pregel API in Apache Spark [12-15]. The algorithm is iterative with maximum number of iterations defined by *MaxIterCount*.

1) *Description of Handler Function*

Each vertex has a local buffer *send_buf*, which is cleared each time the handler is called. At the start of the algorithm, each vertex receives the message *init*, which is necessary for initialization of the algorithm. On receiving the message *init* the vertex puts its identifier *id* into the buffer *send_buf*.

Each vertex has its own set of ancestors – *ancestors*. Initially the set *ancestors* is empty for each vertex. Each vertex also has a flag *bad*, initially set to value *false*. This flag is set to value *true* if the vertex has too many ancestors. At each iteration, a vertex can acquire new ancestors in incoming messages. Acquired ancestors are added to the set *ancestors* if the size of resulting set does not exceed *MaxSetSize*, otherwise the set *ancestors* does not update and the flag *bad* is set to value *true*.

2) *Description of message-sending operation*

At each iteration of the algorithm after handler function finishes its work follows the message-sending step. At this step, all triplets of the graph corresponding to its edges are considered, and a message is sent along each edge.

For each triplet (edge):

- if the buffer *send_buf* of the initial vertex of the edge is empty or the flag *bad* of the terminal vertex equals *true*, then nothing should be sent along this edge;
- otherwise all elements from the buffer *send_buf* which are not present in the set *ancestors* of the terminal vertex are sent within a message along this edge.

3) *Detection of "black holes"*

In the end, each vertex owns a set including a number of its ancestors. A set of vertices *B* with a common ancestor *X* is considered a "black hole" if:

- *X* is not included in the buffer *send_buf* of any vertex from the set *B*. Otherwise some vertex could send *X* at the last iteration, which means there could be an outgoing edge from *B*, which contradicts the definition of a "black hole".
- *X* does not belong to the set ancestors of the initial vertex of each edge incoming to a vertex with the flag *bad* set to value *true*, and *X* is not the initial vertex of such edge. Otherwise (if such edge existed) some vertex (corresponding to such edge) could send *X* at the last iteration, but due to the defined limitations further transmission through the vertex with the flag *bad* set to value *true* would be impossible, and identifier *X* would not be able to reach any of the buffers *send_buf*.

Then the vertices should be grouped according to ancestors. In the Apache Spark realization, each pair (vertex, ancestor) should be mapped to a pair (ancestor, vertex) and then the pairs should be grouped by key (operations *map* and *groupBy*). Defined limitations guarantee that the sets of vertices acquired by the method described above does not include any outgoing edges.

An important feature of the developed algorithm is the ability to identify intersecting "black holes". Pseudo code for the developed algorithm is listed below:

```
MaxSetSize = 100 // constant limiting the size of sets ancestors and send_buf
MaxIterCount = 10 // constant defining the number of iterations
// vertex handler
(1) procedure handler(v: vertex, msgs: Vector[Long])
// message sending
(2) procedure sendMsg(triplet: EdgeTriplet)
v.send_buf = 0
// if initial message
if msgs.size = 1 and msgs(0) = -1 then
```

```
        vert.send_buf.add(vertex.id)
        return
if vert.ancestors.size + msgs.size ≤ MaxSetSize then
        // update set of ancestors
        vert.ancestors.add(msgs)
else
        vert.bad = true
        return
if msgs.size ≤ MaxSetSize then
        vert.send_buf.add(msgs)
else
        vert.bad = true
if !(triplet.srcAttr.send_buf.isEmpty or triplet.dstAttr.bad) then
        msgs = Vector[Long]()
        for x ← triplet.srcAttr.send_buf do
                if !triplet.dstAttr.ancestors.contains(x) then
                        msgs.add(x)
        send msgs along triplet
// merging messages
(3) function mergeMsg(a: Vector[Long], b: Vector[Long])
        // concatenation of two vectors
        return a + b
// main procedure
(4) procedure detect_blackholes(graph: Graph)
```

III. Experimental Study

Experimental study has been carried out on a set of data consisting of the database of all transactions (including 781 440 transactions and 15 034 710 involved entities) and the database of suspicious transactions (including 715 transactions and 349 involved entities). For evaluation of computational performance and scalability of the developed algorithms synthetic Erdos-Renyi [16] graphs of different sizes have also been used.

Software realization of the developed algorithms has been tested on a computational cluster consisting of 8 nodes connected with 1GBit Ethernet, each node running 8-core 2.2GHz E5-2660 processor, 64GB DDR3 memory, operation system SLES 11 SP4, Apache Spark 2.1.1 and Scala compiler sbt 0.13.13.

Software realization of the machine learning algorithm has been carried out using Scala, Spark 2.1.0 language and includes the following steps:

- Data input (database of all transactions, database of suspicious transactions and configuration data).
- Search for suspicious transaction in the database of all transactions (by comparing the fields DATA, SUME, ACC_B0 and Date, RealQty, AccClientOtpr accordingly).
- Creation of a graph with entities as its vertices and transactions as its edges (entities being taken from the database of all transactions).
- Selection of a set of edges for machine learning consisting of all suspicious transactions and the same number of normal transactions (selected randomly).
- Formation of egonets around the vertices for calculation of features.
- Calculation of features for each edge (transaction) (32 features total): amount and time of transaction; for sender and recipient egonets - minimum, maximum and average degrees, indegrees and outdegrees of vertices, number of vertices, “volcano” vertices, “black hole” vertices and other vertices; number of transactions and total transaction amount.
- Machine learning: method – random forest; MinMaxScaler method used to project each feature into [0, 1] line; random division of the learning set – 70% of samples for training (using cross-validation), 30% of samples for testing.

Graph created in the experimental study consists of 114 791 vertices (entities) and 781 440 edges. The learning set for machine learning includes 1430 transactions (50% normal and 50% suspicious). Training set consists of 993 transactions – 477 class 0 objects (normal transactions) and 516 class 1 objects (suspicious transactions). Testing set consists of 437 transactions – 238 class 0 objects (normal transactions) and 199 class 1 objects (suspicious transactions).

Resulting classification accuracy (share of correctly classified objects) reached 97.7%.

A measure of importance for each feature has been calculated. Top five most important features (listed with their respective “coefficient of importance”; the sum of coefficients for all features amounts to one) turned out to be the following:

- Amount of transaction – 0.38
- Degree of the sender vertex – 0.12
- Total amount of transactions (incoming and outgoing) corresponding to the sender – 0.09
- Number of vertices in the sender’s egonet – 0.07
- Outdegree of the sender vertex – 0.07

It is evident that amount of transaction has the greatest influence on classification results.

Other metrics of classification quality calculated during the experimental study (the closer to 1.0 the better):

- AUROC – 0.999
- Sensitivity (also known as “Recall”) – 1.0
- Specificity – 0.958
- Precision – 0.952
- NPV (negative predictive value = number of correctly classified normal transactions divided by total number of transactions classified as normal) – 1.0
- F1 score – 0.975

Sensitivity equaling 1.0 means that the algorithm did not miss any suspicious transactions within the testing set. This metric is especially important, since in anti-money laundering tasks it is necessary not to miss any fraudulent transactions.

False positive rate (FPR = (1-Specificity)*100%) on the testing set amounted to 4.2%, meaning that among 238 normal transactions 10 were falsely classified as suspicious.

Recent studies using a Random Forest Classifier in order to predict fraudulent transactions on raw data [17] have shown results for such metrics of classification quality as Precision, Recall, F1-score and AUROC all below 0.9 in most cases. Thus, it can be concluded that using the developed set of features allows improving the classification quality.

Additional experiment has been carried out using the database of all transactions excluding the suspicious ones as the testing set (780 725 transactions total). The share of transactions classified as suspicious equaled 4.9%. It is worth noting that the share of transactions with amount exceeding 100 000 classified as suspicious equaled 3.2% of the total number of transactions.

Quality of LOF algorithm based on LSH method is shown in Table I.

TABLE I. TABLE TYPE STYLES

THR	FN / P	FP / N
2.5	8.03%	1.2%
3	10.27%	0.9%
3.5	12.13%	0.8%

Here THR defines threshold value for anomaly detection, FN – number of false-negative objects, FP – number of false-positive objects, P – total number of anomalies, N – total number of normal objects. Results were calculated for 6000 random points corresponding to the edges of the real graph. Though the classification quality is lower in comparison with the trained random forest method applied above, the advantage of this approach is the ability to carry out fraud detection without prior knowledge of transaction history.

Using the algorithm for identifying “volcanoes” and “black holes” an edge was considered an anomaly if its initial vertex belonged to any of 5% largest “volcanoes” or “black holes”. Figure 2 shows the distribution of identified “volcanoes” and “black holes” for the real graph.

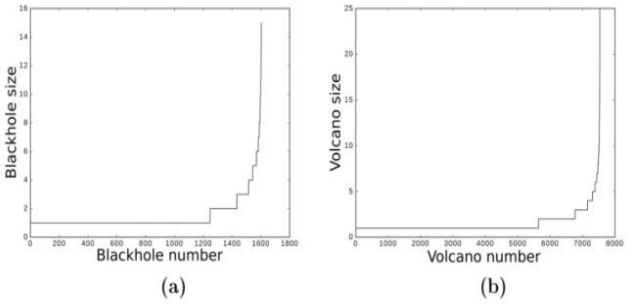

Fig. 2. Distribution of identified “volcanoes” and “black holes”.

Figure 3 illustrates strong scalability of the developed feature calculation algorithm, LOF algorithm based on LSH method and the algorithm for identifying “volcanoes” and “black holes”.

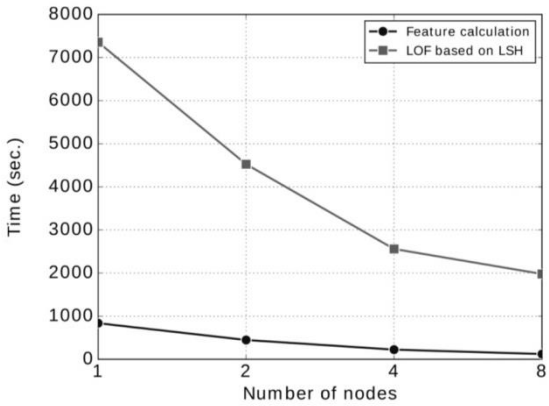

Fig. 3. Scalability of the developed algorithms.

Scalability of feature calculation algorithm was calculated for a synthetic Erdos-Renyi graph with 2^{19} vertices and 2^{22} edges. Scalability of LOF algorithm based on LSH method was calculated for the real graph. Scalability of the algorithm for identifying “volcanoes” and “black holes” was calculated for a synthetic Erdos-Renyi graph with 2^{21} vertices and 2^{24} edges.

Figure 4 illustrates the relative speedup of data processing with the developed algorithms based on the number of computational nodes involved in the processing of data.

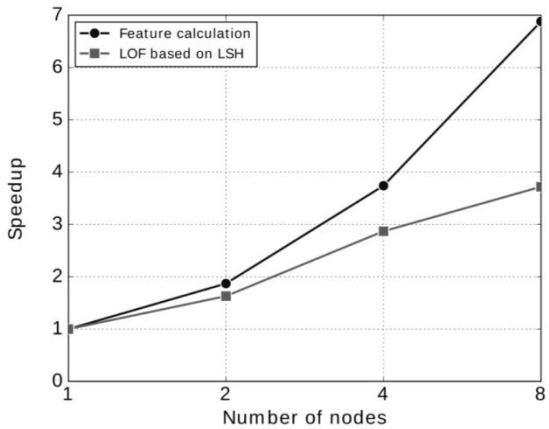

Fig. 4. Speedup of data processing according to the number of computational nodes.

IV. CONCLUSION

It can be concluded that the developed algorithm for feature calculation can be successfully implemented in conjunction with common machine learning methods to achieve high values of classification quality metrics. In particular, high value of sensitivity (and subsequently low False Negative Rate value) is important for anti-money laundering and other fraud management processes.

The results of experimental study have shown that the developed algorithms for anomaly detection demonstrate classification quality comparable with a trained random forest method when applied on a real transaction graph. This makes it possible to implement these algorithms in situations when prior knowledge of transaction history is not accessible, as well as for identification of new and unknown methods of fraud.

Software realization of the developed algorithms demonstrated high scalability, which makes it possible to significantly increase their performance using multi-processor computational clusters.

The developed algorithms are intended to be used in the framework of a software complex for automated fraud management in different areas of business.

ACKNOWLEDGMENT

The research is being conducted with the financial support of the Ministry of Education and Science of the Russian Federation (Contract №14.574.21.0142) Unique ID for Applied Scientific Research (project) RFMEFI57417X0142. The data presented, the statements made, and the views expressed are solely the responsibility of the authors.

REFERENCES

[1] D. Lavion et al., PwC's Global Economic Crime and Fraud Survey, 2018, URL: https://www.pwc.com/gx/en/forensics/global-economic-crime-and-fraud-survey-2018.pdf (retrieved 28 August 2018).

[2] The Nilson Report, 2016, issue 1096, URL: https://nilsonreport.com/upload/content_promo/The_Nilson_Report_10-17-2016.pdf (retrieved 01 October 2018).

[3] 2017 North America Online Fraud Benchmark Report, URL: https://www.cybersource.com/content/dam/cybersource/2017_Fraud_Benchmark_Report.pdf (retrieved 01 October 2018).

[4] A. Chelsea, PayPal's history of fighting fraud, Fin Newsletter, URL: https://fin.plaid.com/articles/paypals-history-of-fighting-fraud (retrieved 01 October 2018).

[5] M. Cochrane, How MasterCard is using AI to improve the accuracy of its fraud protection, Business Insider, 2017, URL: https://www.businessinsider.com/mastercard-artificial-intelligence-fraud-protection-2017-1 (retrieved 01 October 2018).

[6] Demystifying machine learning for banking, URL: https://hollandfintech.com/wp-content/uploads/2018/01/Feedzai-Demystifying-Machine-Learning-for-Banking-2017.pdf (retrieved 01 October 2018).

[7] Next-generation fraud management solutions, URL: https://www.capgemini.com/wp-content/uploads/2017/07/next-generation_fraud_management_2017.pdf (retrieved 01 October 2018).

[8] C. Pettey and R. Van der Meulen, Gartner Says a Layered Fraud Prevention Approach Can Thwart Malicious Attacks, Gartner, 2009, URL: https://www.gartner.com/newsroom/id/1254413 (retrieved 01 October 2018).

[9] Z. Li, H. Xiong, and Y. Liu, Detecting Blackholes and Volcanoes in Directed Networks, CoRR, 2010, URL: https://arxiv.org/pdf/1005.2179.pdf (retrieved 23.11.2016).

[10] R.R. McCune, T. Weninger, and G.R. Madey, Thinking Like a Vertex: a Survey of Vertex-Centric Frameworks for Distributed Graph Processing, CoRR, 2015, URL: http://arxiv.org/abs/1507.04405 (retrieved 23.11.2016).

[11] M. Zaharia, M. Chowdhurr, M.J. Franklin et al., Spark: Cluster Computing with Working Set, HotCloud, 2010, vol. 10, p. 7.

[12] M. Armbrust, T. Das, A. Davidson et al., Scaling spark in the real world: performance and usability, Proceedings of the VLDB Endowment, 2015, issue 8, vol. 12, pp. 1840-1843.

[13] Voit A., Stankus A., Magomedov Sh., Ivanova I. Big data processing for full-text search and visualization with elasticsearch International Journal of Advanced Computer Science and Applications. 2017. T. 8. № 12. C. 76-83. DOI: 10.14569/IJACSA.2017.081211

[14] Magomedov Sh. Organization of secured data transfer in computers using sign-value notation. ITM Web of Conferences. 2017. T. 10. DOI: 10.1051/itmconf/20171004004

[15] N. Chaimov, A. Malony, S. Canon et al., Scaling Spark on HPC Systems, 2016, pp. 97-110.

[16] P. Erdos and A. Renyi, On random graphs, Publicationes Mathematicae Debrecen, 1959, vol. 6, pp. 290-297.

[17] R. Pierre, Detecting Financial Fraud Using Machine Learning: Winning the War Against Imbalanced Data, 2018, URL: https://towardsdatascience.com/detecting-financial-fraud-using-machine-learning-three-ways-of-winning-the-war-against-imbalanced-a03f8815cce9 (retrieved 20.11.2018).