A New Machine Learning Approach for Anomaly Detection Using Metadata for Model Training

Alexander Prosak, Amitava Gangopadhyay and Hemant Garg

March 15, 2019

A New Machine Learning Approach for Anomaly Detection Using Metadata for Model Training

Alexander Prosak
Innovation Labs
SWIFT, Inc.
Manassas, VA 20110
alexander.prosak@swift.com

Amitava Gangopadhyay
Innovation Labs
SWIFT, Inc.
Manassas, VA 20110
amitava.gangopadhyay@swift.com

Hemant Garg
Innovation Labs
SWIFT, Inc.
Manassas, VA 20110
hemant.garg@swift.com

Abstract— ***We report a new approach to train machine learning (ML) models for binary classification in order to detect anomalies in application log records. Contrary to the common use of actual values of different log fields, we used metadata of the log records ("log schema") to train and test our ML models. Our objective was to use ML models to automatically detect anomalous log records that may carry sensitive or restricted information and thus prevent their inadvertent transfer ("leakage") from the source to destination environments. In addition to all the controls and measures currently in place to prevent such data leakage, our ML model approach provides an additional layer of data security to further reduce the possibility of potential data leaks.***

Several ML models (decision tree (DT), random forest (RF) and Gradient Boosted Tree (GBT)) were trained using a combination of real (class: "normal") and synthetic (class: "suspicious") metadata for approximately five million log records. The metadata for "normal" records were extracted from the schema of real historical log records that do not contain "sensitive" or "restricted" information. The metadata for likely "suspicious" records were simulated via artificially injecting structural violations that are not observed in the known "normal" log records. The final prediction ("normal" or "suspicious") for each new record was based upon the use of a voting classifier. The three ML models (DT, RF and GBT) in our solution all individually yield high average accuracy in predictions (1.0, 0.99 and 1.0, respectively) over multiple experimental runs. Accordingly, the voting classifier consistently yields highly accurate predictions (1). Combined, our results suggest that the use of a combination of real and synthetic metadata derived from log schema and a voting classifier can be successfully applied to build a robust ML model solution for anomaly detection in log records.

Keywords—anomaly detection, machine learning, decision tree, random forest, gradient boosted tree.

I. Introduction

Anomaly or outlier detection involves identification of one or more rare, unexpected or suspicious items, events or observations [1-3]. Some common anomaly detection use cases in the industry include abnormal traffic pattern detection in financial transactions, fraud detection in insurance claims, prevention of network intrusion, and monitoring of unusual system or user behavior. Traditionally, different statistical analytical models have been used for the purpose of anomaly detection in large datasets [3, 4]. The statistical methods usually involve computations of various statistical parameters of representative sample data (e.g., moving average, standard deviation), followed by formulation of rules based upon their computed values in known "normal" sample data. New samples that deviate from the defined "normal" patterns are usually identified as potential anomalies or outliers [4].

More recently, with the advent of "big data", it is now recognized that the use of machine learning (ML) models, in conjunction with dynamic statistical analyses, can provide more suitable real-time solution to most anomaly detection problems, particularly in cases where the distribution patterns of "normal" data in high-velocity massive datasets change frequently or evolve dynamically over time [5-9]. The ML modeling approach typically involves data preprocessing, training and testing of models, and finally detecting anomalies in new and commonly live streaming data using previously trained ML models.

The detection of sensitive data as "anomalous" and prevention of their unintended transfer from a secure source environment to a non-secure destination environment is usually called "DLP" ("Data Loss or Leak Prevention") [10, 11]. In this specific study, our objective was to detect leakage of sensitive log data during their transfer from the source environment (secure zone) to its destination (non-secure zone) environment (*Fig. 1*). The data from various application logs in the source data lake may contain sensitive information related to specific details on financial transactions. While it is necessary to move data from their source to the destination data lake, particularly for the purpose of log analytics and system monitoring, it is also critically important to prevent any accidental leakage of data in this process.

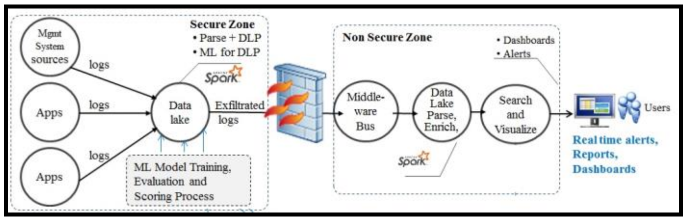

Fig. 1 The data sources and sinks for the data lakes in the secure zone and non-secure zone environments

Here we report the results of our new approach to train multiple ML models using a training dataset that was prepared via extracting the schema of known historical “normal” log records. The training dataset also included synthetic records with artificially injected schema violations labeled as “suspicious” records. The ML models in our approach collectively yield final predictions based upon simple majority voting on new records as either “normal” or “suspicious”. We demonstrate that our approach of using a combination of log metadata and voting classifier consistently yield accurate predictions.

II. Methodolodies

In *Fig. 2*, we show a schematic to outline the complete workflow for data preprocessing, and ML model training, testing and prediction. Here is a brief description of each of these stages:

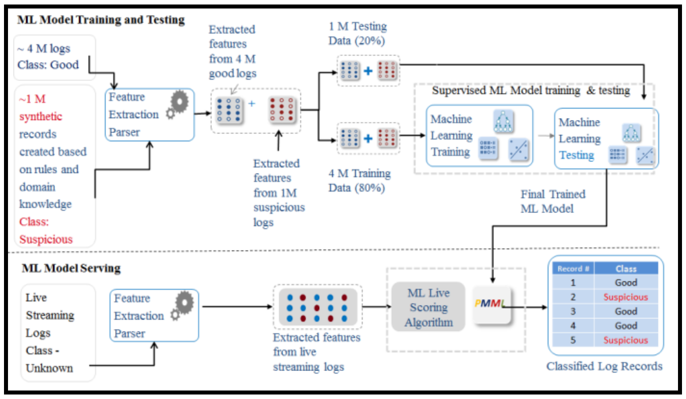
ML Model Training and Testing
~ 4 M logs
Class: Good
~1 M synthetic records created based on rules and domain knowledge
Class: Suspicious
Feature Extraction Parser
Extracted features from 4 M good logs
Extracted features from 1M suspicious logs
1 M Testing Data (20%)
4 M Training Data (80%)
Supervised ML Model training & testing
Machine Learning Training
Machine Learning Testing
Final Trained ML Model
ML Model Serving
Live Streaming Logs
Class - Unknown
Feature Extraction Parser
Extracted features from live streaming logs
ML Live Scoring Algorithm
PMML
Record #
Class
1
Good
2
Suspicious
3
Good
4
Good
5
Suspicious
Classified Log Records

Fig. 2 The ML workflow for data leak prevention

A. *Data Preprocessing*

In typical ML modeling, actual values of individual features are used in training datasets as part of data preprocessing. In our case, we have, however, used the schema (metadata) of log records in order to train and test our ML models. As such, our sample data were comprised of metadata extracted from known “normal” log records available in the data lake in the non-secure source environment. The raw log records extracted from the non-secure environment closely resemble the source log data in the secure zone environment, and, accordingly, our sample data are representative of the log records that our ML models are likely to ingest upon deployment in the secure zone environment.

First, the metadata of approximately four million historical log records were extracted from their immediate storage in HBase. These pipe-delimited raw log records were parsed via a custom-built parser to extract the individual fields of the log records (*Fig. 2*). Each field in a record was further broken down into multiple sub-fields based upon its schema elements (e.g., field length, alphabetic, numeric, special character; also see *Table 1*). For the field type schema element, we used a binary indicator to populate values for its data type fields (0 for no and 1 for yes). For example, if a given field in the log is of numeric data type for a specific record (e.g., “Field Length”; see Table 1), the value for that feature was populated with 1. If the value of the same feature is non-numeric for a different log record, the value for that record is populated with 0. This process was repeated for each feature and for each record in the training dataset. Further, Shannon entropy (discussed later; Table 1) was computed for each feature based upon its unencrypted value and also encrypted values with respect to simple, hexadecimal and base64 encryption methods. Finally, the sign of each field value in the log (positive or negative) was populated in the “Sign” field of the feature matrix. Finally, if a given log record had less than maximum number of features for the entire dataset, the missing feature values were padded with 0’s in the sample metadata matrix. Combined, our feature matrix had a total of 10 separate fields (“sub-features”) that describe the metadata for a single feature. These records were used as “normal” samples (class 0) in our sample dataset.

In the absence of known “suspicious” samples in the source log, we programmatically created synthetic (class 1) samples. This was done via artificially injecting records with structural violations that are not observed in “normal” samples. First, a custom program was used to create a “baseline” for all positive records for each combination of primary schema elements in the historical dataset. The baseline creation is important for our dataset because it ensures that all variabilities within a single field for a “normal” sample metadata are considered across all records prior to generation of synthetic suspicious samples. This, in turn, ensures that synthetic suspicious samples are indeed different from any of the existing normal samples in the log records. We used approximately four million class 0 and one million class 1 records for our sample data.

For each sample, a random unique identifier (UUID) was assigned and the target variable (class 0 or 1) was labeled. Further, the values of three primary schema elements repre-sentting categorical fields (Application name, log type and flow type) were converted to numerical values using the *StringIndexer*() Spark ML library function [12]. This function is similar to a hashing function that parses over each

Table 1 Representative sample data for the ML models*

UUID	App_Name	Log_type	Flow_Type	NOF	FL_1	Num_1	Alph_1	AlphaNum_1	Str_1	Unen_Entpy_1	En_Entpy_1	Hex_1	B64_1	Sign_1	…….	Target
g602337400	abc	ABcLog	REPLICATION	11	4	0	0	0	1	1.5	4.334962501	0	1	0	…….	0
g985347899	abc	ABcLog	REPLICATION	11	4	0	0	0	1	2	4.053508855	0	1	0	…….	0
FL49782052	abc	ABcLog	REPLICATION	11	4	0	0	0	1	1.780863225	4.401009816	0	1	0	…….	1
g531662446	abc	ABcLog	REPLICATION	11	4	0	0	0	1	1.5	4.053508855	0	1	0	…….	0
g1191718574	abc	ABcLog	REPLICATION	11	4	0	0	0	1	2	4.418295834	0	1	0	…….	0
g1791474902	abc	ABcLog	REPLICATION	11	4	0	0	0	1	2	4.251629167	0	1	0	…….	0
g1289576505	abc	ABcLog	REPLICATION	11	4	0	0	0	1	1.5	4.251629167	0	1	0	…….	0
FL464086029	abc	ABcLog	REPLICATION	11	4	0	0	0	1	1.692824713	3.997382869	0	1	0		1

(*NOF: No. of fields; FL: field length; Num: numerical; Alph: alpha-betic; AlphaNum: alphanumeric; En: encrypted; Unen: unencryptted; Entpy: entropy)

categorical feature column and assigns a numerical value to each categorical value. Next, all records (four million class 0 and one million class 1) were mixed and reshuffled to yield the final sample data that were used by our ML model training program and subsequently for model testing. We used an 80/20 split out of a total of approximately five million samples for our model training and testing.

B. ML Model Training

We used three different ML models for our solutions: Decision Tree (DT) [13, 14]; Random Forest (RF) [15, 16]; and Gradient Boosted Tree (GBT) classifier [17-19]. Further, we used a Simple Majority Voting Classifier to yield the final predictions for each new test sample.

1) Decision Tree (DT)

A decision tree is a structure that resembles an inverted tree with its root at the top and branches with nodes, arcs, and finally ending with leaves at the bottom [13, 14]. Nodes are assigned to feature attributes, whereas the arcs that emerge out of the nodes are assigned to the features in the dataset. The leaves of the tree are assigned to one of the target classes (0 or 1). The partitioning of sample data continues recursively until each subset ends up with a single leaf representing a single class. As a result, the "path" of nodes links all the relevant features in their decreasing order of importance from the top to the bottom of the tree and classify the sample data into their respective target classes. As noted, in our specific use case, only two classes were used (class 1 for "suspicious" and class 0 for "normal" samples). Thus, a complete partitioning of sample dataset via DT algorithm resulted in either 0 or 1 (and not both) as class values in the leaves.

2) Random Forest (RF)

The RF algorithm uses labeled training data to create a collection ("ensemble") of Decision Trees (hence called "forest") and merge them together to yield potentially more consistent predictions than expected from a single individual tree [15, 16]. Each tree was trained using a defined percentage of randomly selected subset from the original training dataset (66% with replacements). We used an ensemble of X trees in our RF model. The final prediction of the model (class 0 or 1) was based upon the built-in aggregation logic applied on the collection of predictions from each of the X trees constructed beforehand. The RF model usually tends to reduce the variance in predictions via the use of more trees.

3) Gradient-Boosted Tree (GBT)

Similar to RF, the GBT model also uses ensembles of decision trees. In contrast to the RF model which builds fully-grown trees, the GBT model, however, iteratively builds shallower (and hence weaker) trees. The GBT model uses the aggregation of predictions from these multiple shallow trees in order to make final predictions. Unlike RF, where reducing variance is the primary objective, the GBT model tends to reduce the bias in final predictions by using more trees [17-19].

C. ML Model Testing

In order to leverage the strengths of each of the three different models (DT, RF and GBT) we implemented a simple majority voting classifier to produce a single final prediction based upon the individual prediction from each model (described above). Each log record is scored by the three models, and thus yielding three predictions. These predictions (1's or 0's) were persisted in a SQL table for each new log record identified by their unique ID (UUID). The voting algorithm joins each of the three output tables corresponding to each model, and creates a new table with the predictions of each model in separate columns. The final voting score is calculated as the average of each row and rounded to the nearest integer or zero. For example, if the predictions for a single row are 0, 1, and 1, the rounded average for those predictions would be 1, essentially taking a majority vote (a tie results in 1 as the output). The voting algorithm uses model inputs dynamically, so new models can be added and removed without necessitating changes in the voting classifier.

Our ML model training program trained each model with the training subset and saved it as a PMML file in the HDFS. The trained ML models were tested via using a separate testing program. The testing program loaded the feature matrix of the test data that was prepared during the initial data preprocessing stage prior to model training (section II A). Next, the testing program loaded each of the three saved ML models from HDFS. These saved models were used to make predictions based upon the test data. The voting classifier subsequently made the final prediction (class 0 or 1) on each record. The test results were saved in a file in the HDFS for evaluation on model performance.

In accordance with confidentiality requirement, the details of scoring program and the source data used for the program cannot be reported here. Suffice to note that we have created a separate ML model scoring program where the feature matrix for new streaming log data will be ingested and the scoring program will make predictions in real time. This scoring program is similar to the testing program in all aspects except for its real-time data ingestion capability. The scoring program is capable of ingesting streaming data via use of Spark structured streaming and make predictions in real time.

D. ML Solution Implementation

As noted, the ML models were built using Spark ML/MLLib libraries [12], and were trained and tested using source-like data saved in the HDFS in a 6-node Spark cluster. The data pipeline and saved models were deployed in the source environment. A custom-built script is run to fetch sample data from the HBase, prepare a CSV file and save it locally in the cluster. A shell script was created that is capable of kick-starting the jobs of data preprocessing, model training and finally model testing. Once the mode test performance is evaluated to be satisfactory, the scoring program is run via a script in order to make real time predictions. In source environment, the periodic evaluations of model performances and required retraining of models are supported as part of ML model life cycle.

III. RESULTS

As noted in section 2, the performance of each model was evaluated via a testing program using approximately one million test data that comprised a 20% split of the original five million samples. The total training time for the three ML models was estimated to be approximately 2 hours and 30 mins using about 4 million training samples. As expected, the elapsed time for ML model testing, on the other hand, was much smaller (less than an hour). The test results and performance metrics for each model are shown in Table 2.

Table 2: Performance metrics* for the ML models and the voting classifier

Models	TP	TN	FP	FN	Accu.	Prec.	Rec.	F1	AUC
DT	799163	151948	0	0	1.0	1.0	1.0	1.0	1.0
RF	799130	151632	33	316	0.99	1.0	1.0	0.999	0.999
GBT	799163	151948	0	0	1.0	1.0	1.0	1.0	1.0
Voting	**799163**	**151948**	**0**	**0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**

(*TP/TN: True Positive/Negative; FP/FN: False Positive/ Negative; Accu.: Accuracy; Prec.: Precision; Rec.: Recall; F1: F1 Score; AUC: Area Under the ROC Curve)

IV. DISCUSSIONS

Three features in our historical log dataset (Application name, Flow type and Log type) mainly control the variability in the log schema, and are called "primary schema elements" here. As such, the metadata for a given combination of these primary schema elements vary only to limited extents across all positive log records. Conversely, two log records with different combination of primary predictors may have vastly different metadata.

As noted, our current modeled solution uses DT, RF and GBT models. However, we initially experimented with additional binary classification models (e.g., Naïve Bayes [21], Support Vector Machine [22]) using the same sample data. Our experimental results consistently showed that the use of DT, RF and GBT models followed by a majority voting classifier yielded the most consistent and highly accurate results (*Table 2*). Accordingly, here we limit our discussion to the latter three models.

The DT model uses a "greedy algorithm" that recursively partitions the training data into one of our binary classes (class 0 or 1) [23]. Each partition is greedily selected based upon the best split from a set of possible splits. The split that maximizes the "information gain" at a given node in the tree is identified as the best split. Further, the node impurity provides a measure of homogeneity of the labels at the node. The node impurity is usually determined via one of two popular attribute selection methods, "gini index" or entropy. Although we evaluated both methods in our current implementation, the gini index method seems to have yielded the most consistently accurate results for our sample dataset. This is expected because gini index is more suitable where samples are split into larger partitions and into smaller number of classes (only two classes in our case).

While DT models consistently yields accurate results over multiple runs in our testing, use of DT model alone can impose certain limitations [24]. For example, the DT algorithm trains only one single decision tree in modeling. Also, DT structure can be somewhat unstable because it depends heavily on the feature values in each node of the tree. As a result, the DT model can have a large impact on the overall structure of the tree due to small changes in the features values in the training dataset. Further, DT models are also prone to overfitting on the training data. For these reasons, we have used two additional "ensemble" methods ("bagging" and "boosting") in our modeling in order to build a more robust solution. The two ensemble models we have used for "bagging" and "boosting" are a random forest (RF) model, and a gradient-boosted tree (GBT) model, respectively.

The RF and GBT models attempt to address the issue of overfitting observed in the DT model via training multiple decision trees in parallel. In our RF model, we have used a 66% subset of training data with replacement in order to build an ensemble of X separate trees [25]. The use of more trees helps to improve the model performance in two divergent ways in RF and GBT models. Notably, the RF model reduces the variance in predictions by using more trees, whereas more trees in the GBT model tend to reduce the bias observed in a single tree.

Both RF and GBT models, however, have their strengths and weaknesses [26]. For example, GBTs train one tree at a time, whereas the RF models train multiple trees in parallel. As a result, GBT models can take longer to train than RF models using the same training data. Note that the GBT model, however, can use shallower trees than in RF models in order to reduce the training time. Further, RF models are usually easier to tune because model performance improves monotonically with addition of progressively larger number of trees. The performances of GBT models, on the other hand, can degrade if the number of trees continues to grow beyond its optimum number. The two algorithms, when used in combination via voting, thus leverage the strengths of both models. This is consistent with our results that RF model alone make small proportion, yet significant number of false positive (FP) and false negative (FN) predictions (33 and 316, respectively; see Table 2), whereas the voting classifier combines the results of all three models (including GBT) and makes 100% accurate predictions based upon our test data.

The DT and GBT models consistently yield accurate predictions over multiple runs (accuracy = 100%; see *Table 2*). Similarly, the accuracy in predictions from the RF model is also impressively high (> 99%). Such high accuracy in prediction from individual models in some cases may indicate possible "overfitting" of training samples. The issue of overfitting can be identified via comparative evaluation of model performance during training and testing. For example, if a highly complex ML model is used in training, it can result in high training accuracy, yet the same model is likely to fail to "generalize" on new samples. Each of our three ML models, on the other hand, yields highly accurate predictions on a large subset of testing samples (~ one million that were not used during model training). This suggests that our two

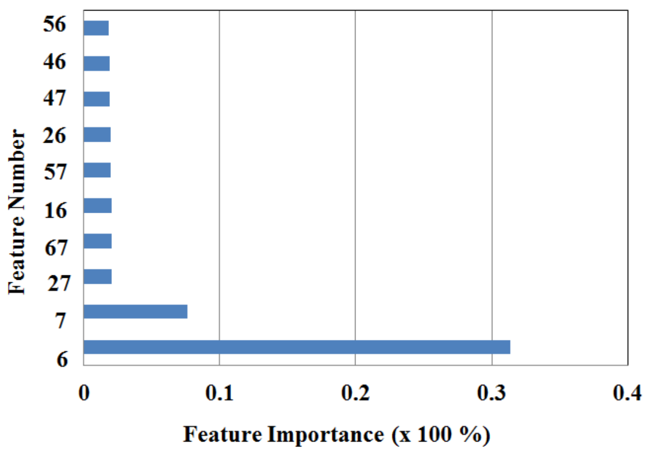

Fig. 3 Chart of the relative importance of 10 best features in the training dataset for the DT model. These 10 features collectively account for ~51% of the cumulative feature importance.

target classes (class 0 and 1) are linearly separable in the feature space of our sample dataset. This is consistent with our observation that some of the features with highest importance (e.g., Shannon entropy values) are distinct between the two classes. As noted, each field in the log record was broken down to 10 separate sub-features in our dataset (section II A). Also note in *Fig 3*, that the 10 features that together contribute to more than 50% of feature importance in our DT model are not the 10 "sub-features" of a single feature. Instead, the 10 important features include seven distinct features from the dataset (ranging from feature number 6 through 67). This suggests that the DT model predictions are not heavily biased based on one or two particular features. Further, except for features 6 and 7, the other features carry somewhat equal importance (~ 2%: see *Fig 3*). Combined, these results suggest that multiple distinctive features in the dataset allow the DT model to separate the two target classes (0 and 1) on the feature space with minimal or no overlap.

The RF model yields some incorrect predictions (FP = 33 and FN = 316; *Table 2*) that constitute a very small fractions of the total predictions (< 0.01%). Although such a low percentage of incorrect prediction is acceptable in most use cases, we chose to build a more robust solution for our use case. This is because the log records misclassified as "suspicious" (false positive (FP)) are otherwise candidates for blocking from the source to destination environment. Conversely, the 316 false negative (FN) predictions represent records that are actually suspicious, yet the RF model failed to classify them correctly. As such, the data leakage of these records from the source to destination environments cannot be prevented, and it can pose a data security risk.

Although both false positive and false negative predictions are undesirable, false negative predictions, as discussed above, pose greater risk of data leakage in our use case. Accordingly, one of the acceptance criteria for our ML models was to achieve minimal or no false negative predictions. To this end, we have leveraged multiple ML models (DT, RF and GBT), rather than relying solely on one single model. Our results suggest that the use of voting classifier to aggregate the predictions from these three individual models and make final prediction based on majority voting yield consistently highly accurate (100%) predictions.

V. CONCLUSIONS

In this study, we have demonstrated the effectiveness of a new approach of using log metadata to train machine learning (ML) models for binary classification in detecting suspicious records that may carry sensitive or restricted information. We have used a combination of real ("normal") and synthetic ("suspicious") metadata for approximately five million log records and trained three ML models (decision tree (DT), random forest (RF) and Gradient Boosted Tree (GBT)). These three ML models in our solution all individually yield high average accuracy in predictions (1, 0.99 and 1, respectively) over multiple experimental runs. The final predictions (class 0 for "normal" and class 1 for "suspicious") were obtained via a custom voting classifier based upon the aggregated majority voting on the predictions form each of the three ML models. The voting classifier consistently yields highly accurate predictions (accuracy = 1.0). Combined, our results suggest that the use of a combination of real and synthetic metadata derived from log schema and a voting classifier can be successfully applied to build a robust solution for anomaly detection in log records.

In the future, we intend to explore the efficacy of deep learning (DL) models (e.g., Autoencoder, Restricted Boltzmann Machine (RBM)) as additional methods for detecting anomalous log records. Similarly, different natural language processing (NLP) methods can also be potentially used to detect sensitive information in log records based on actual values of different fields rather than use of their metadata (as used in our current ML solution). Finally, in contrast to our current implementation, the different ML models and the voting classifier can be deployed as microservices that can run concurrently and also as self-contained services, rather than sequentially in a monolithic application. The use of microservices as ML models can significantly reduce the training and testing time of the models and enable the application to make both training and scoring at the same time.

Disclaimer

The views articulated in this paper are personal to the authors and do not represent the views of their employers or any other organization.

Acknowledgement

This research was funded and supported by the Innovation Labs and Summer Internship Program at SWIFT. We acknowledge the help and support from the Innovation Lab and OASIS Delivery team in productizing the solution. We thank Soumitra Dutta (Cornell Univ.) and Uwe Aickelin (Univ. of Melbourne) for their very constructive informal reviews of an earlier version of this manuscript.

REFERENCES

[1] Hodge, V. J.; Austin, "A Survey of Outlier Detection Methodologies" (PDF). Artificial Intelligence Review. 22 (2): 85–126, 2004

[2] Chandola, V.; Banerjee, A.; Kumar, V., "Anomaly detection: A survey". ACM Computing Surveys. 41 (3): 1–58, 2009

[3] Zimek, A., Schubert, E., "Outlier Detection", Encyclopedia of Database Systems, Springer New York, pp. 1–5, 2017

[4] Zimek, A., Filzmoser, P., "There and back again: Outlier detection between statistical reasoning and data mining algorithms". Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 8 (6): e1280, 2018

[5] Knorr, E. M.; Ng, R. T.; Tucakov, V., "Distance-based outliers: Algorithms and applications". The VLDB Journal the International Journal on Very Large Data Bases. 8 (3–4): 237–253, 2000

[6] Ramaswamy, S., Rastogi, R., Shim, K., Efficient algorithms for mining outliers from large data sets. Proceedings of the 2000 ACM SIGMOD international conference on Management of data – SIGMOD '00. p. 427., 2000

[7] Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; Sander, J., LOF: Identifying Density-based Local Outliers (PDF). Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. SIGMOD. pp. 93–104, 2000

[8] Angiulli, F.; Pizzuti, C., Fast Outlier Detection in High Dimensional Spaces. Principles of Data Mining and Knowledge Discovery. Lecture Notes in Computer Science. 2431. p. 15., 2002

[9] "Machine Learning Techniques for Anomaly Detection: An Overview," S. Omar, A. Ngadi, and H. H. Jebur, 2013, International Journal of Computer Applications, vol. 79, no. 2, pp. 33–41.

[10] Shabtai, A., Elovici, Y., Rokach, L., A Survey of Data Leakage Detection and Prevention Solutions, Springer-Verlag, 2012

[11] Ouellet, E., Magic Quadrant for Content-Aware Data Loss Prevention, Technical Report, RA4 06242010, Gartner RAS Core Research, 2012

[12] Apache Spark 2.2.0 MlLib Main Guide, Extracting, transforming and selecting features: StringIndexer, https://spark.apache.org/docs/2.2.0/ml-features.html#stringindexer, Accesses on 12/31/2018.

[13] Breiman, L., Friedman, J. H., Olshen, R. A., Stone, C. J., Classification and regression trees. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software, 1984

[14] Quinlan, J. R., Induction of Decision Trees. Machine Learning 1: 81-106, Kluwer Academic Publishers, 1986

[15] Ho T. K., Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282, 1995

[16] Ho T. K., "The Random Subspace Method for Constructing Decision Forests" (PDF). IEEE Transactions on Pattern Analysis and Machine Intelligence. 20 (8): 832–844, 1998

[17] Breiman, L., "Arcing The Edge", Technical Report 486. Statistics Department, University of California, Berkeley, 1997, https://statistics.berkeley.edu/sites/default/files/tech-reports/486.pdf

[18] Friedman, J. H., "Greedy Function Approximation: A Gradient Boosting Machine", IMS 1999 Reitz Lecture, version April 19, 2001, https://statweb.stanford.edu/~jhf/ftp/trebst.pdf

[19] Mason, L.; Baxter, J.; Bartlett, P. L.; Frean, Marcus (1999). "Boosting Algorithms as Gradient Descent", In S.A. Solla and T.K. Leen and K. Müller. Advances in Neural Information Processing Systems 12. MIT Press. pp. 512–518, 1999

[20] Quinlan, J. R., Induction of Decision Trees. Machine Learning 1: 81-106, Kluwer Academic Publishers, 1986

[21] Hand, D. J.; Yu, K., "Idiot's Bayes — not so stupid after all?". International Statistical Review. 69 (3): 385–399, 2001

[22] Cortes, Corinna; Vapnik, Vladimir N., "Support-vector networks". Machine Learning. 20 (3): 273–297, 1995

[23] Alkhalid A., Chikalov I., Moshkov M., Comparison of Greedy Algorithms for Decision Tree Optimization. In: Skowron A., Suraj Z. (eds) Rough Sets and Intelligent Systems - Professor Zdzisław Pawlak in Memoriam. Intelligent Systems Reference Library, vol 43. Springer, Berlin, Heidelberg, 2013

[24] Safavian, S. R., landgrebe, D., A Survey of Decision Tree Classifier Methodlogy, School of Electrical Engg., Purdue Univ., 1990, https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19910014493.pdf

[25] Dietterich, T., "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization". Machine Learning: 139–157, 2000

[26] Breiman L., Random forests. Machine learning, 45(1): 5-32, 2001