Suspicious Network Event Recognition Leveraging on Machine Learning

Daniele Sartiano
IIT/CNR and
Dipartimento di Informatica
Università di Pisa
Via Moruzzi 1
Pisa, Italy
daniele.sartiano@iit.cnr.it

Giuseppe Attardi
Dipartimento di Informatica
Università di Pisa
Largo B. Pontecorvo 3
Pisa, Italy
attardi@di.unipi.it

Luca Deri
IIT/CNR and
Dipartimento di Informatica
Università di Pisa
Via Moruzzi 1
Pisa Italy
luca.deri@iit.cnr.it

Maurizio Martinelli
IIT/CNR
Via Moruzzi 1
Pisa, Italy
maurizio.martinelli@iit.cnr.it

***Abstract*— Network log events produced by network probes are used by security analyzers to detect traffic anomalies and threats. While it is relatively trivial for a probe to mark specific events as suspicious, it is much more challenging for log analyzers to create a comprehensive picture of the overall network. Machine learning can potentially help in this, however there is no specific solution for analyzing network event logs. This paper covers the experiments and design choices that have been made to create a machine learning-based tool able to analyze network event logs. The tool has been evaluated in the Suspicious Network Event Recognition Cup at IEEE BigData 2019, achieving an AUC (Area Under the Curve) of over 90%, making it accurate enough for deployment in real life scenarios.**

Keywords—machine learning, gradient boosting, network events, cyber-security

I. Introduction

The architecture of most network security tools is split into two main components: a network sensor that produces traffic logs, and a log analyser that analyses them. For network security events the two most popular tools are Suricata and Zeek. [1] is an open-source network IDS (Intrusion Detection System) able to create network event traces complemented with additional security information produced when matching signatures. Zeek, formerly known as Bro, [2] is also a network IDS that does not leverage signatures as it is mostly used to create network logs that are used by log analysers. While the network sensor side is monopolised by these two tools, the problem of analysing security event logs is still mostly open. In the past few years Machine Learning (ML) techniques have been exploited to solve specific problems such as detection of malware behind encrypted TLS streams [3]–[5] . Besides this, ML has also been used both in research [6], [7] and commercial products [8], [9] in order to forecast future usage and produce early warnings, and report unexpected changes in behaviours due to unusual deviations from past behaviour.

In order to validate the effectiveness of the most successful ML algorithms used in other domains, such as image recognition or text analysis, we decided to analyse an annotated dataset of network events, which was provided through a data mining challenge [10] organised in association with the IEEE BigData 2019 conference. Event annotation is not common in real life, as security tools only label basic events often using simple threshold-based techniques and network administrators are not usually keen to perform such task. However, annotated data were crucial for our assessment since they allowed us to train an ML system and to evaluate its results in comparison with those of other participants in the same challenge. As reported in table III, this work ranked 14th out of 82 scored submissions. If we group submissions by score, we note that the top 3 obtained an AUC of more than 0.92, 6 submissions an AUC of more than 0.91, and then 5 submissions, including ours, had an AUC over 0.90.

The rest of this paper is organized as follows. Section II describes the challenge for which this tool has been developed. Section III covers the various options that have been evaluated, as well the architecture of the tool developed. Section IV describes the experiments carried on validating this work. Section V evaluates the results and positions the developed tool against the other challenge participants. Finally, Section VI summarizes this work and gives an outlook of future work items.

II. Challenge description

The aim of the challenge was to detect suspicious events and false alarms analysing network traffic events. In order to achieve this goal, the organizers provided an annotated dataset of network events, where, for each alert, multiple features about network flows were provided such as statistical data, information about IP addresses, protocols, ports, etc. The dataset was enriched by a set of localized alerts and related log events. The challenge lasted three and a half months.

The evaluation was done using the AUC (Area Under the Curve), a metric that measures the area under the ROC (Receiver Operating Characteristic curve) curve. The ROC shows the performance of a classification model: it plots the TPR (True Positive Rate), also known as Recall, versus the FPR (False Positive Rate) where TPR and FPR are defined in (1) and (2).

$$TPR = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (1)$$

$$FPR = \frac{False\ Positive}{False\ Positive + True\ Negative} \quad (2)$$

AUC measures the area underneath the ROC curve from the (0,0) to (1,1), and the larger the area, the more accurate is the model is.

In order to provide an immediate feedback on the accuracy of each submission, the organizers made available an online evaluation tool with a public leader board. This tool was available throughout the challenge. This preliminary evaluation was done on a small subset of the test set.

In the following sections we provide a description of the data provided, comprising three datasets: aggregated alert events, localized alert events and log events. The datasets are cross referenced by the alert identifier.

A. *Aggregated alert events dataset*

The first dataset is a collection of aggregated alert events divided into training and test set. They are provided as compressed files consisting of records of pipe-separated values of 63 fields, extracted by the software described in [11], here a summary of the fields:

- *alert_ids*: alert identifier.
- *client_code*: encrypted client code.
- *notified*: the target column with binary values.
- *categoryname*: category of the alert.
- *ip, ipcategory_name, ipcategory_scope, parent_category, grandparent_category*: information about the encrypted IP address.
- *overallseverity*: estimation of the alert severity.
- *timestamp_dist, start_hour, start_minute, start_second, weekday*: time information.
- *correlatedcount*: number of records denoted by the system in an auxiliary table with localized alerts.
- *n1, n2, n3, n4, n5, n6, n7, n8, n9, n10*: binary results of analytical queries.
- *score*: score issued by an autonomous analytical model.
- *srcip_cd, dstip_cd, srcport_cd, dstport_cd*: numbers of different IP addresses (source and destination) and network ports (source and destination) in an auxiliary table with localized alerts.
- *alerttype_cd*: number of triggered alert types in the localized alerts.
- *direction_cd, eventname_cd, severity_cd, reportingdevice_cd, devicetype_cd, devicevendor_cd, domain_cd, protocol_cd, username_cd, srcipcategory_cd, dstipcategory_cd*: number of different fields denoted in an auxiliary table with localized alerts.
- *isiptrusted*: a binary field indicating whether the IP address corresponding to the alert is controlled by the customer.
- *untrustscore, flowscore, trustscore, enforcementscore*: various scores based on the network activity.
- *dstipcategory_dominate, srcipcategory_dominate, dstportcategory_dominate, srcportcategory_dominate*: most frequent information about IP addresses and ports denoted in an auxiliary table with localized alerts.
- *thrcnt_month, thrcnt_week, thrcnt_day*: number of records from an auxiliary table with threat watch alerts, denoted for the same IP address as the alert, during the previous month, week and day.
- *p6, p9, p5m, p5w, p5d, p8m, p8w, p8d*: results of analytical queries.

The test set file has the same format as the training set, except that the target column "*notified*" is missing. The training set is almost twice the size of the test set, as described in TABLE I. Only 5.8% of alerts are labelled as to be notified.

TABLE I. Aggregated alert events dataset

Dataset	**# Alerts**	**Notified**
Training set	39427	2276 (5.8%)
Test set	20000	-

B. *Localized alert events dataset*

Localized alerts describe intermediate data stored by the organizers systems [11] and are related to the alert in the training and test set. For each alert there is a list of events, with the following fields:

- *alert_ids*: alert identifier ('AAB', 'BXm', 'EHr', etc)
- *alerttype*: type of the alert ('Active Scan', 'DNS Alert', 'Failed Login High Rate', 'Suspicious Outbound anomaly - Company', etc)
- *devicetype*: type of the device ('AAA', 'Scan', 'AWS Flow', 'WAF', etc)
- *reportingdevice_code*: reporting device code ('tMU', 'vMn', 'rQE')
- *devicevendor_code*: device vendor code ('EF', 'NB', 'QZ', etc)
- *srcip*: anonymized source IP address ('172.KM.QP.85', '10.KT.ZT.17', 'AR.XY.10.50', etc)
- *dstip*: anonymized destination IP address ('FP.ZX.248.10', 'PN.ZU.2.16', 'DJ.FB.217.116', etc)
- *srcipcategory*: category of source IP address ('BENCH', 'INTERNET', 'PRIV-10', 'PRIV-192', 'LINK-LOCAL', etc)
- *dstipcategory*: category of destination IP address ('CURR_NET', 'MULTICAST', 'PRIV-172', 'PRIV-192', '6TO4', etc)
- *srcport*: source port (0, 443, 33143, 49814, etc)
- *dstport*: destination port (0, 443, 80, 8888, 21, 22, etc)
- *srcportcategory*: category of the source port (0, 1, 2, 3, 4)
- *dstportcategory*: category of the destination port (0, 1, 2, 3, 4)
- *direction*: direction (0, 1, 2, 3, 4, 5, 6, 7, 8)
- *alerttime*: relative alert time (0, 855, 6012, etc)
- *severity*: severity score (1, 2, 3, 4, 5, 8, 9)

- *count*: count (1, 2, 3, 10, 272, etc)
- *domain*: binary value
- *protocol*: network protocol ('udp/6054', 'tcp/9908', 'ssh', 'SAMBA', etc)
- *username*: binary value
- *signature*: binary value

The number of localized alerts for each aggregated alert is highly variable: it ranges from 1 to 916,736. The dataset consists of 8,690,705 entries.

C. *Log events dataset*

Log events are individual events logged by security system's software. This dataset was available only to the team that achieved a preliminary score greater than 0.85. The dataset is composed by the following fields: *alert id, src ip, dst ip, src ipcategory, srcport, dst ipcategory, dstport, relative timestamp, report device code, device type, device vendor code, event name code, usr code, dev id code, dev severity, dev rule, object code, srccountrycd code, dstcountrycd code, direction, disposition, protocol, asnid code, event code, domain code, logontype, targetuser*. Again, the number of log events for each alert is highly variable. The total number of entries is about 4 billion for a total of about 21 GB of compressed files.

III. Approach and Implementation

We investigated two approaches, the first one based on gradient boost algorithm and the second one based on a deep learning architecture.

A. *XGboost*

Gradient boosting is a supervised technique that uses ensembles of decision trees [12]. This is an iterative method that, for each step, the gradient of the cost function is calculated with respect to the predicted value of the ensembles and the trees which are then added in order to move in the direction of the gradient. Extreme Gradient Boosting, better known as XGBoost, is a scalable end-to-end tree boosting system [13]. It works very well on structured data and it can handle missing values.

The classifier was implemented through the Python library XGBoost[1]. We defined the class *Reader*, able to read and extract features from the aggregated, localized and log events datasets. The features were extracted from the pipe-separated values files, that are elaborated through the *Pandas*[2] library. In this approach we used aggregated data for each sample, so we collected multiple features for a single alert. We extracted all the features from the aggregated dataset, whereas for the other two datasets, we extracted aggregated information, such as statistical features. We processed non-numerical variables as categorical and we converted them into dummy variables, implemented using the *Pandas*'s function *get_dummies*[3], that transforms a categorical feature input into an indicator variable. In addition to the fields in the aggregation dataset, we created new features, such as the concatenation or the normalization of some fields. The features extracted from the localized alerts dataset are statistical information about a subset of available fields, such as mean, minimum, maximum.

Considering the large size of the log events dataset, we built a script to split the information as requirements a single file for each alert. In this way we collected 53472 pipe-separated files named by the alert id. These files were processed by another script able to extract and aggregate information in parallel. The result of this processing was a single pipe-separated file with the alert id and a list of new fields with statistical information, such as the mean, max, min, sum of fields; we also added information about the source and destination countries involved in the logs, both as a list of unique country and as the list of the pair country and number of time the this country appear in the logs. For example, for the alert id "*BpB*" the script created three new fields "*logevents_srccountry*", "*logevents_dstcountry*", "*logevents_countries*":

- *logevents_srccountry*: "AM_HD_JJ" means that "AM", "HD" and "JJ" appeared as values in the field "*srccountrycd_code*" of the log events.
- *logevents_dstcountry*: "HD_JJ" means that "HD" and "JJ" appeared as values in the field "*dstcountrycd_code*" of the log events.
- *logevents_countries*: "AM+462_HD+19794_JJ+20256" means that "AM" appeared 462 times in the fields "*srccountrycd_code*" and "*dstcountrycd_code*" of the log events.

The dataset created was joined with the aggregated alert dataset and the localized events dataset. In this way we obtained about 9000 features with our best configuration described in the section IV.A.

B. *Deep learning*

The main idea behind the deep learning approach is to build a classifier using network layers capable of remembering sequences of data, considering that the localized alert events dataset and the log events dataset are collections of sequences. We decided to focus on the localized alert events dataset, which is more manageable with respect to the log events dataset.

The input is a sequence of localized alerts, where we divided the features into numerical and categorical. Numerical fields are mapped together into a vector and then the sequence of vectors is passed into a LSTM [14] model. Categorical fields are mapped into trainable Embeddings layers and then passed into LSTM layers. In this way we obtained a total of 17 Input fields, 1 for the numerical fields and 16 for categorical variables (categorical: *alerttype*, *devicetype*, *reportingdevice_code*, *devicevendor_code*, *srcipcategory*, *dstipcategory*, *srcport*, *dstport*, *srcportcategory*, *dstportcategory*, *protocol*, *severity*, *direction*, *username*, *signature*, *domain*; numerical: *alerttime*, *count*). The results of all the LSTM layers are concatenated together into a single layer and then passed into a Dense layer. We used a binary cross entropy as loss function and Adam [15] as optimizer. During the experiments, various

---

[1] https://xgboost.readthedocs.io/en/latest/python/python_intro.html
[2] https://pandas.pydata.org/
[3] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html

models were tested, for example using a bidirectional LSTM [16], or introducing intermediate dense layers.

The classifier was implemented in TensorFlow 2.0[4], a well-known framework for building deep neural networks. We used Keras[5], a high-level API for building and training deep learning models. As in the XGBoost classifier, we created a *Reader* class in order to process the dataset. Categorical features were transformed into indexes through the *LabelEncoder* class of *Scikit-Learn* library; the start value of these indexes was one, in order to use the masking function of LSTM, implemented in Keras[6]. Considering the variability in length of the sequences, we decided to consider only the last part of sequences.

IV. Experiments

In our experiments, we randomly split the dataset into a development set (80%) and a test set (20%). The development set was split in turn into a training set (80%) and a validation set (20%). The training of the models was done using the training set and we validated the models through the validation set. The test set was used to evaluate the solutions. The results were scored using the AUC metric described in the previous sections.

The XGBoost experiments were run on two machines, equipped with the processor Intel i7-7700K, respectively with 32 and 64 GB of RAM. The deep learning experiments were conducted on a server equipped with 4 GPUs Nvidia Tesla Pascal 100.

A. XGBoost

We applied a feature engineering approach, following this method:

- we started the experiments with the aggregated events dataset and doing feature engineering, including almost all the features available (discarding the "alert id") and creating other new features described in the previous section;
- we did feature selection in order to find the best set of features;
- we added the features extracted from the localized events dataset and we continued the feature selection;
- after having reached the AUC of 0.8800, we downloaded the log events dataset and extracted the features;
- we did feature engineering with all the features from the three datasets.

In order to tune the XGBoost hyperparameters we did randomized search with stratified k-folds cross validation passing the following parameters:

- "*min_child_weight*" with values [1, 5, 10];
- "*gamma*" with values [0.5, 1, 1.5, 2, 5]
- "*subsample*" with values [0.6, 0.8, 1.0];
- "*colsample_bytree*" with values [0.6, 0.8, 1.0];
- "*max_depth*" with values [3, 4, 5].

the parameters used are shown in TABLE II, included the best values found by the randomized search.

TABLE II. XGBost parameters used during the experiments

Parameter	Value
booster	dart
eta	0.02
objective	binary:logisitc
eval	auc
min_child_weight	1
gamma	5
subsample	0.6
colsample_bytree	0.6
max_depth	5
early_stopping_round	250
epochs	5000
scale_pos_weight	$\frac{\#neg\ samples}{\#pos\ samples} = \frac{37151}{2276} = 16.32$

The "*scale_pos_weight*" parameter was used in order to manage unbalanced dataset: the value was calculated as the division of the number of negative samples over the number of the positive samples. We did feature selection by a trial and error method, and we found the following best configuration:

- aggregated dataset: all field except for "alert_ids", "start_second", "start_minute", "timestamp_dist", "thrcnt_month", "thrcnt_week", "thrcnt_day". We added the following features: first element of "ip" field; the concatenation of first and second element of "ip" field;
- concatenation of the following fields of the aggregated dataset: "protocol_cd" and "srcport_cd"; "protocol_cd" and "dstport_cd"; "overallsecurity" and "score"; "reportingdevice_cd", "devicetype_cd" and "devicevendor_cd"; "untrustscore", "flowscore", "trustscore" and "enforcementscore"; "srcportcategory_dominate" and "dstportcategory_dominate";
- localized dataset: number of entries in the localized set, maximum and mean of "alerttime"; maximum, minimum and mean of "severity"; sum and mean of "count"; mean of "domain", "signature" and "direction"; the dummies values of the following fields: "protocol", "alerttype", "devicetype", "reportingdevice_code", "devicevendor_code", "srcportcategory", "dstportcategory", "username", "direction", "srcipcategory", "dstipcategory"; first element of "srcip" field; the concatenation of first and second element of "srcip" field; first element of "dstip"

[4] https://www.tensorflow.org/
[5] https://www.tensorflow.org/guide/keras
[6] https://www.tensorflow.org/guide/keras/masking_and_padding

field; the concatenation of first and second element of "dstip" field;

- log events dataset: number of entries.

B. *Deep learning*

The experiments with the deep learning classifier involved mostly exploring different neural network architectures, in contrast to the XGBoost classifier, in which feature engineering was the main activity. In addition to the architecture described in section III.B, as shown in Fig. 1, we tried also a Bidirectional LSTM and a version with an added convolutional layer on top of the LSTM layer. We do not report the results of these architectures since they did not improve. In order to handle overfitting we added dropout layers [17] and regularization [18]. Unbalanced dataset was managed by setting class weights.

The best score achieved on our test set with a deep learning architecture was about 0.62 AUC. We have been unable to find a good way to mitigate overfitting issues even though in the training phase AUC was about 0.95 on our validation set. Due to a lack of time, this will be an investigation we plan to carry on as future work item.

V. Results

We submitted our two best solutions achieved on the preliminary results with XGBoost classifier. The first one with the same configuration described in the experiments section, the second one as the first one except for the features extracted from aggregated alert events dataset. In addition to the excluded fields described, we removed "ip" related and network ports. The second one achieved the best solution shown in TABLE *III*.

TABLE III. Official results of our submission compared to the top three.

Rank	Team Name	Score
1	hieuvq	0.931743
2	test_123	0.930295
3	HSOC	0.926885
14	**Pisa**	**0.902961**

VI. Future Work

The results of our experiments were encouraging in terms of the final ranking achieved using XGBoost. However we hoped to achieve even better results with the deep learning model, since it is capable of taking into account the ordering of events, that might be valuable for improving the accuracy, rather than using just the aggregated data exploited by gradient boosting. The deep learning model though run into overfitting and we did not have time to investigate its causes. Our plan is to also evaluate whether combining the two approaches used in this work into an ensemble, achieves a better score that what we obtained with the one we used in this competition.

VII. Conclusions

This paper described the design and implementation of a tool developed by the authors for participating to the data

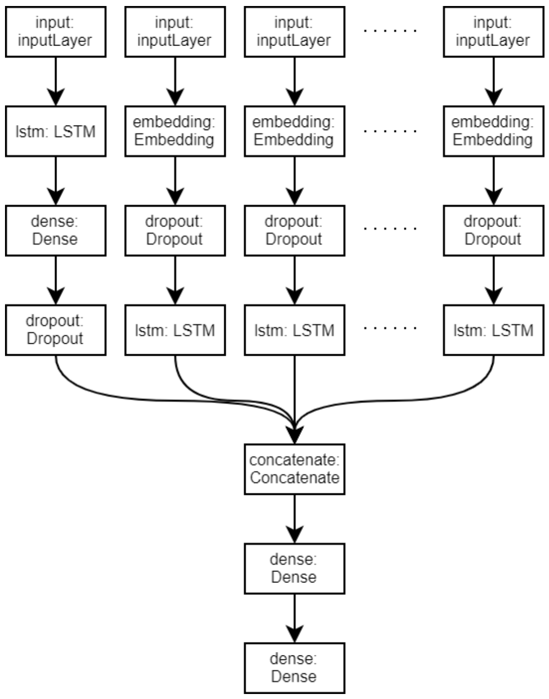

Fig. 1 Deep Learning Classifier

mining challenge organized in association with the IEEE BigData 2019 conference. Our best results were achieved using an XGBoost classifier that outperformed an experiment carried out with deep learning techniques. With respect with other submissions to the challenge, our submission scored in the 14th position.

Source code

The source code of the tool is available at https://gitlab.tools.iit.cnr.it/cyberlab/suspicious-network-event-recognition-2019 and it is released under the GPLv3 license.

Acknowledgments

The experiments were conducted on a server with 4 Nvidia Tesla Pascal 100 GPUs, acquired with partial funding from Grandi Attrezzature 2016 by the Università di Pisa.

References

[1] W. Park and S. Ahn, "Performance comparison and detection analysis in snort and suricata environment," Wirel. Pers. Commun., vol. 94, no. 2, pp. 241–252, 2017.

[2] V. Paxson, S. Campbell, J. Lee, and others, "Bro intrusion detection system," 2006.

[3] D. Bekerman, B. Shapira, L. Rokach, and A. Bar, "Unknown malware detection using network traffic classification," in 2015 IEEE Conference on Communications and Network Security (CNS), 2015, pp. 134–142.

[4] Y. Yang, C. Kang, G. Gou, Z. Li, and G. Xiong, "TLS/SSL Encrypted Traffic Classification with Autoencoder and Convolutional Neural Network," in 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2018, pp. 362–369.

[5] B. Anderson, S. Paul, and D. McGrew, "Deciphering malware's use of TLS (without decryption)," J. Comput. Virol. Hacking Tech., vol. 14, no. 3, pp. 195–211, 2018.

[6] V. Gustavsson, "Machine Learning for a Network-based Intrusion Detection System: An application using Zeek and the CICIDS2017 dataset." 2019.

[7] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization.,” in ICISSP, 2018, pp. 108–116.

[8] Splunk Inc, “Machine Learning Toolkit and Advisory Program,” 2019. [Online]. Available: https://www.splunk.com/en_us/software/splunk-enterprise/machine-learning.html.

[9] Steve Dodson, “Introducing Machine Learning for the Elastic Stack.” [Online]. Available: https://www.elastic.co/blog/introducing-machine-learning-for-the-elastic-stack.

[10] A. Janusz, D. Kałuża, Chądzyńska-Krasowska, B. Konarski, J. Holland, and D. Ślęzak, “IEEE BigData 2019 Cup: Suspicious Network Event Recognition,” in 2019 {IEEE} International Conference on Big Data, BigData 2019, Los Angeles, CA, USA, December 9-12, 2019, 2019.

[11] D. Ślezak, A. Chadzyńska-Krasowska, J. Holland, P. Synak, R. Glick, and M. Perkowski, “Scalable cyber-security analytics with a new summary-based approximate query engine,” in 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 1840–1849.

[12] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” Ann. Stat., pp. 1189–1232, 2001.

[13] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, 2016, pp. 785–794.

[14] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997.

[15] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv Prepr. arXiv1412.6980, 2014.

[16] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” IEEE Trans. Signal Process., vol. 45, no. 11, pp. 2673–2681, 1997.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” J. Mach. Learn. Res., vol. 15, no. 1, pp. 1929–1958, 2014.

[18] A. Y. Ng, “Feature selection, L 1 vs. L 2 regularization, and rotational invariance,” in Proceedings of the twenty-first international conference on Machine learning, 2004, p. 78.