Mach Learn (2015) 101:59–84
DOI 10.1007/s10994-014-5473-9

Analysis of network traffic features for anomaly detection

Félix Iglesias · Tanja Zseby

Received: 9 December 2013 / Accepted: 16 October 2014 / Published online: 4 December 2014

Abstract Anomaly detection in communication networks provides the basis for the uncovering of novel attacks, misconfigurations and network failures. Resource constraints for data storage, transmission and processing make it beneficial to restrict input data to features that are (a) highly relevant for the detection task and (b) easily derivable from network observations without expensive operations. Removing strong correlated, redundant and irrelevant features also improves the detection quality for many algorithms that are based on learning techniques. In this paper we address the feature selection problem for network traffic based anomaly detection. We propose a multi-stage feature selection method using filters and stepwise regression wrappers. Our analysis is based on 41 widely-adopted traffic features that are presented in several commonly used traffic data sets. With our combined feature selection method we could reduce the original feature vectors from 41 to only 16 features. We tested our results with five fundamentally different classifiers, observing no significant reduction of the detection performance. In order to quantify the practical benefits of our results, we analyzed the costs for generating individual features from standard IP Flow Information Export records, available at many routers. We show that we can eliminate 13 very costly features and thus reducing the computational effort for on-line feature generation from live traffic observations at network nodes.

Keywords Feature selection · Anomaly detection · Network security · Data preprocessing · Supervised classification

Editors: Vadim Strijov, Richard Weber, Gerhard-Wilhelm Weber, and Süreyya Ozogur Akyüz.

F. Iglesias (✉) · T. Zseby
Institute of Telecommunications, Vienna University of Technology,
Gusshausstrae 25 / E389, 1040 Wien, Austria
e-mail: felix.iglesias@nt.tuwien.ac.at

T. Zseby
e-mail: tanja.zseby@tuwien.ac.at

1 Introduction

Today's communication networks evolve quickly. The same is true for network attacks. New vulnerabilities appear every day and are quickly exploited in zero-day attacks. Whereas signature-based detection fails to detect previously unknown attacks, anomaly detection techniques can find deviations from normal traffic patterns and therefore are an important instrument to improve network security in today's communication networks.

Although there is a significant amount of technical and scientific literature on anomaly detection methods for network traffic, the valuable step of feature selection is often underrepresented and treated inattentively in the literature. Tavallaee et al. (2010) point out three commonly defective aspects in anomaly detection research: the employed data sets, the characteristics of the performed experiments, and the methods used for performance evaluation. Within the characteristic of the experiments, the authors underline data preprocessing as one of the phases usually skipped in related papers, pointing out that *feature selection* is habitually freely undertaken without a proper justification. This is quite unfortunate because removing trivial and redundant features does not only reduce resource consumption for processing, storing and transmitting data, it also enhances the modelling of the phenomena under analysis and, therefore, it is a determinant step in the detection of network anomalies.

Guyon and Elisseeff (2003) expose this idea when they say:

> The objective of variable selection is threefold: (1) improving the prediction performance of the predictors, (2) providing faster and more cost-effective predictors, and (3) providing a better understanding of the underlying process that generated the data.[1]

As far as anomaly detection is concerned, the importance of (1) is out of question. (2) is a prerequisite for the fast installation of countermeasures to prevent the massive propagation of malware. (3) is necessary in order to generalize solutions and obtain new knowledge that enables future enhancements and refinements.

The objective of our work is to rank and select network traffic features with regard to their contribution to the detection of anomalies. The starting point is the widely-adopted set of 41 features, used in the NSL-KDD (2009), the DARPA (1998) and the KDD cup (1999) data sets, which provided the basis for numerous studies on network anomaly detection. We argue that using a smaller subset of those features is not only more resource efficient than using all features but also leads to better detection results for a broad range of typical classifiers.

We label features as highly relevant, medium relevant and negligible; and discuss their roles in the posterior classification processes. Our feature selection achieved sound conclusions on feature subsets by smartly combining different filters and stepwise regression techniques, implemented in wrappers.

In order to evaluate the selection decision, simple models of different classification paradigms were utilized: Decision Trees (DTC), k-Nearest Neighbor models (kNN), Naïve Bayes classifiers, Least Absolute Selection and Shrinkage Operator with Least Angle Regression (LASSO-LAR), Artificial Neural Networks (ANN) and Support Vector Machines (SVM). The first four models were also deployed for feature subset validation and refinement.

We are aware that the network research community is still short on openly accessible representative labeled data sets for network traffic analysis. Severe critics have been given to classical data sets (McHugh 2000), which have been used in many previous works on anomaly detection. In our research we used the more recent NSL-KDD (2009) database, that

[1] Enumeration marks – (1), (2), (3) – are ours.

has been provided to overcome shortcomings of existing data sets. In addition, we addressed critical points described in Tavallaee et al. (2010) when conducting our experiments.

2 Related work

Among the copious amount of studies facing anomaly detection in networks, there are some works that deal with feature selection prior to testing detection and classification techniques. For a comparison of related work we show the feature subsets proposed by others (if revealed) together with our results in Fig. 1.

Despite the critiques on the DARPA'98 and the KDD Cup'99 data sets, they embrace a high percentage of revised studies about anomaly detection, also for feature selection. The DARPA'98 data set was selected in Sung and Mukkamala (2003) for research on feature selection and classification with SVM and ANN. The feature selection process is carried out with a backward elimination wrapper. Authors establish subsets of important and secondary features depending on the type of attack to detect, as well as a union of important feature subsets that accounts for 30 features. Differences between evaluations with the important subset only and adding the secondary subset do not exceed a 0.3 % in the accuracy measure. Unfortunately, authors do not specify which features are selected. Later, in Sung and Mukkamala (2004), the same authors carry out feature selection with algorithms based on SVM, Multivariate Adaptive Regression Splines (MARS) and Linear Genetic Programming (LGP). They conclude on 6 different imperative features for every method. Switching from 41 to 6 features involves an accuracy performance degradation of 1.74 % on the worst case. If we superimpose the selection of the three methods, a subset of 11 most important features is obtained.

For the same data set, in Khor et al. (2009) two algorithms called Correlation-based Feature Selection Subset Evaluator (CFSE) and Consistency Subset Evaluator (CSE) select features for a posterior Bayes classifier. The final proposal consists of 7 main features. Also with DARPA'98, in Chebrolu et al. (2005) Bayesian networks, classification trees and regression trees discriminate imperative features.

For the KDD Cup'99 data, authors in Nguyen et al. (2010) reduce from 41 features to a minimum of 1 and a maximum of 22 depending on the type of attack to identify. They use correlation-based methods and check relevance and redundancy. Unfortunately, features are

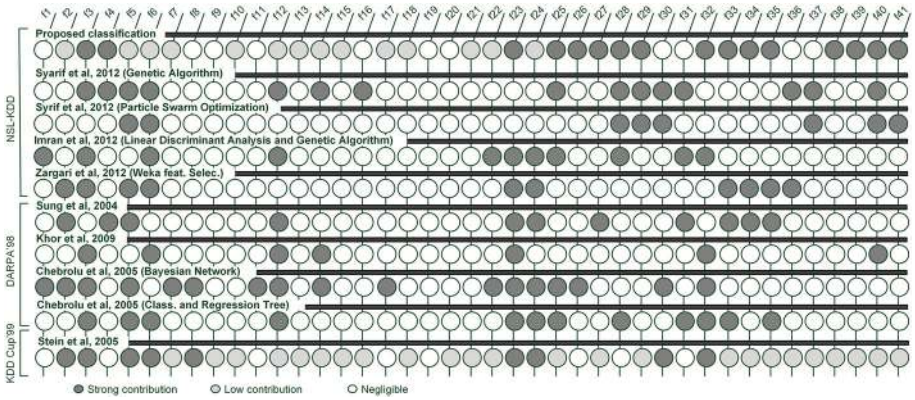

Fig. 1 Comparison of feature selection studies for network traffic anomaly detection

not specified. For the same data set, in Stein et al. (2005) a genetic algorithm (GA) wrapper with a DTC as a validation model looks for relevant features. They are shown for the DoS type of attack case.

There are also some works concerning feature selection for the NSL-KDD data set. In Syarif et al. (2012), GA and Particle Swarm Optimization (PSO) are utilized for feature selection, counting on Naïve Bayes, DTC, *k*NN and Rule Induction as learning models for validation. GA selects 14 features and PSO 9, both obtaining better performances than by using the original 41 feature subset, at least clearly when using the *k*NN classifier. In Imran et al. (2012), authors carry out feature selection by means of Linear Discriminant Analysis (LDA) and GA, concluding in an optimal subset of 11 features when a Radial Basis Function (RBF) is in charge of classification. Previously, they directly removed nominal features (i.e., *protocol_type, service* and *flag*) supposedly to enable linear analysis. Finally, two feature selection methods extracted from Weka (Correlation-Stepwise-based, and Information-Gain-based) are applied in Zargari and Voorhis (2012), reaching the best performance with a subset of 10 features.

All investigated papers are based on the same initial set of 41 features, but even if they use exactly the same data, they all propose different feature subsets for the detection of anomalies (Fig. 1 shows the discrepancies). We found several reasons for the disagreement of previous feature selection work in literature and explicitly address such reasons in the present paper.

1. *Different objectives, if not mixed or confusing.* Some of the introduced works look for differentiating specific type of attacks in the data sets, while others mix anomaly detection and attack identification in the same process. Others even deal with *normal traffic* as if it were another threat type, leading to a misleading interpretation of results. It is obvious that different classification objectives lead to different feature sets. Our work focuses on *anomaly detection* (separating normal from anomalous data), not on attack identification.
2. *Different attack distributions.* Although data sets all use the same 41 features, they introduce different collections of attacks, differently distributed. Attack distribution affects the feature selection, but even more the performance of feature selection validation methods. This explains differences between studies of the different data sets and of studies that use different validation techniques. In our work we use the most recent KDD-NSL (2009), for which defects of previous data sets have been fixed. Furthermore, we validate our results with different classifier types to avoid bias.
3. *High correlation among features*. In our analysis we found several highly correlated features. For instance, *f13* and *f16* show a linear correlation of 0.998 (features are described in Appendix, Table 6); or the lower linear correlation index among *f25*, *f26*, *f38* and *f39* equals 0.975. Hence, if a selection method opts for *f13* as relevant, it will probably consider *f16* as redundant, therefore neglected. Another method may favor *f16*, and then surely reject *f13*. To decide among high correlated features some methods opt for the feature most correlated to labels, whereas other approaches check the power of the discussed features in combination with the pre-selected subsets. Such methodological differences lead to different results. In our work, we propose a multi-stage feature selection that combines different methods. Furthermore, we use multiple validation paradigms, including models less sensitive to redundancy (e.g., DTC, *k*NN).
4. *Performance indices*. Many of the cited papers use *accuracy* as the only performance index. Considering the non-normal attack type distribution, trusting only the *accuracy* index is not optimal. For the NSL-KDD database *normal traffic* and *neptune attack* accounts for more than 80 % of samples (similarly in the precedent data sets). The habitual high *accuracy* values obtained by classifiers make it difficult to elucidate which option

under test is the best (in this case: which feature subset). Additional indices are necessary, such as *false positive rate*, *false negative rate*, *precision*, *recall* or *ROC curves* (represented by the area under the curve—AUC). In our work we use multiple performance indices and display besides *accuracy* also attack *precision*, attack *recall*, and *AUC*.

5. *Insufficient validation techniques.* Finally, a common source of lack in statistical significance comes from mixing the evaluation of feature selection with the evaluation of classifiers or detection systems (IDS's). Such misunderstanding leads researchers to deploy the original, proposed train/test data set division also to check feature selection. Since evaluation results are not cross-validated, and considering the narrow accuracy range where classifiers coexist (mainly in classic data sets), final assessments are not prone to be representative. To reach maximum significance, we have utilized the whole NSL-KDD database for feature selection, executing fivefold cross-validation in the decisive evaluations.

In our research we aimed at a general feature subset not bound to a specific classification approach. The idea is that future researchers can use the reduced feature set instead of the full 41 features without degradation of the classifier performance. Furthermore, we looked at the costs for feature generation and show which of the costly feature can be omitted for on-line feature generation in IDS's.

Our contribution can be summarised as follows:

- We reduce the network feature set from 41 commonly used features to only 16 features without (or with minimal) anomaly detection power degradation.
- We introduce a multi-stage feature selection with different techniques to eliminate bias introduced by specific methods and understand feature contribution.
- We consider the costs for feature generation and show which of the costly features are negligible.
- We evaluate our results with six very different classifier types and present all relevant performance indicators for our experiments.

3 Network data for anomaly detection

Network anomaly detection is challenging due to the dynamic nature of network traffic. Proposed solutions cover techniques inherited from statistics, data mining and machine learning. The recent work by Bhuyan et al. (2013) offers a comprehensive survey that compares a considerable number of network IDS's.

Anomalies appear due to network intrusions and attacks, but also because of malfunctioning devices or network overloads. According to Thottan and Ji (2003), we define an *anomaly* as any circumstance that makes network traffic deviate from normal behaviour. Since we work with labeled data, we consider an *anomaly* any vector not labeled as *normal traffic* in the database, i.e. labeled as a *type of attack* (Appendix: Table 6). We differentiate between *anomaly detection* and *anomaly identification*, concepts that sometimes seem to be mixed in the literature (e.g., references in Sect. 2). Our paper focuses exclusively on anomaly detection. Note that determinant features for anomaly detection are not necessarily the same as the features selected for identifying the type of anomaly.

3.1 The NSL-KDD data set

We are aware that due to the variety of networks, traffic profiles and attacks, the representativeness of any network traffic data set can be questioned. So finding suitable labelled

datasets is difficult. Many of the published anomaly detection and feature selection proposals still utilize DARPA'98 and KDD'99 Cup, although the antiquity of those data sets and the strong, popular critics received advise against their use (McHugh 2000). In this work we used the more recent NSL-KDD data set (2009). The NSL-KDD data set (2009) has been proposed by Tavallaee et al. (2009) to provide a data set for IDS testing that overcomes some of the inherent problems found in the KDD Cup'99 database (also present in DARPA'98). NSL-KDD solves the problems concerning the huge number of redundant records and the duplication of records in the test data; furthermore, a more challenging attack distribution is provided. The NSL-KDD data set has been already adopted by the IDS research community, e.g., Panda et al. (2011), Salama et al. (2011).

Like their predecessors, NSL-KDD contains 41 features split into three groups: *basic features*, extracted from TCP/IP connections; *traffic features*, related to the same host or the same service; and *content features*, based on data from packet contents. Features are listed in Table 1 and described in the Appendix, Table 7.

3.2 Attack types

Table 6 in the Appendix displays the labeled attacks with presence rate in the training and test sets. Threats are clustered according to attack type: *Denial of service (Dos)*, where some resource is swamped, causing DoS to legitimate users. *Probes*, gathering network information to bypass security. *Remote to local (R2L) attacks* that exploit remote system vulnerabilities to get access to a system. *User to root (U2R) attacks* that attempt to gain root access to a system.

The NSL-KDD data set is released divided into two groups: training (125973 samples) and test (22544 samples). The division of the data is chosen in a way most demanding for classifiers. Train and test data do not show the same attack probability distribution; moreover, 16 out of the 38 labeled threats are only present in the test data set. In addition, the presence of some attacks is very scarce (2 or 3 samples in the whole database).

4 Feature selection

Reducing the number of features brings obvious benefits in terms of computational resources; i.e., less stored data, faster processing and easier data extraction. Moreover, data mining and machine learning techniques profit from dimensionality reduction in more complex ways. In this section, we present *feature selection* methods, their benefits for classification (beyond resource optimization), and their implications dealing with network anomaly detection databases.

4.1 Problems of high-dimensionality for classification

From a theoretical perspective, including meaningless or redundant variables to describe a phenomenon should not entail a degradation in the performance of classifiers. But from the practical application, we find that learning algorithms by default ignore the underlying distribution of the data and are habitually forced to discover a solution by approximating NP-hard optimization problems due to the presence of redundant features (Kohavi and John 1997). This issue can be dramatic for neural networks based classifiers or decision trees looking for optimality.

Another problem related to a feature excess is known as *curse of dimensionality*, i.e., the complexity of the multi-dimensional input space exponentially increases with every new

Table 1 Features in the NSL-KDD data set, their relevance for anomaly detection (rel), effort to generate them (eff) and the required IPFIX Information elements

Id	Scale	feature	Rel	Eff	IPFIX IE
f1	Integer	B: duration	○	S	flowDurationMicroseconds
f2	Nominal	B: protocol_type	·	S	protocolIdentifier
f3	Nominal	B: service	●	S	destinationTransportPort
f4	Nominal	B: flag	●	H	state keeping
f5	Integer	B: src_bytes	·	S	octetTotalCount
f6	Integer	B: dst_bytes	●	S	octetTotalCount
f7	Binary	B: land	·	M	comparison of IEs
f8	Integer	B: wrong_fragment	●	H	checksum check
f9	Integer	B: urgent	○	M	tcpUrgTotalCount
f10	Integer	C: Hot	·	H	content inspection
f11	Integer	C: num_failed_logins	○	H	content inspection
f12	Binary	C: logged_in	·	H	content inspection
f13	Integer	C: num_compromised	·	H	content inspection
f14	Binary	C: root_shell	·	H	content inspection
f15	Binary	C: su_attempted	·	H	content inspection
f16	Integer	C: num_root	○	H	content inspection
f17	Integer	C: num_file_creations	·	H	content inspection
f18	Integer	C: num_shells	·	H	content inspection
f19	Integer	C: num_access_files	○	H	content inspection
f20	Integer	C: num_outbounds_cmds	○	H	content inspection
f21	Binary	C: is_hot_login	○	H	content inspection
f22	Binary	C: is_guest_login	·	H	content inspection
f23	Integer	T: count	●	M	comparison of IEs
f24	Integer	T: srv_count	·	M	comparison of IEs
f25	Real	T: serror_rate	●	H	state keeping
f26	Real	T: srv_serror_rate	●	H	state keeping
f27	Real	T: rerror_rate	·	H	state keeping
f28	Real	T: srv_rerror_rate	●	H	state keeping
f29	Real	T: same_srv_rate	●	M	comparison of IEs
f30	Real	T: diff_srv_rate	○	M	comparison of IEs
f31	Real	T: srv_diff_host_rate	○	M	comparison of IEs
f32	Integer	T: dst_host_count	●	M	comparison of IEs
f33	Integer	T: dst_host_srv_count	●	M	comparison of IEs
f34	Real	T: dst_host_same_srv_rate	●	M	comparison of IEs
f35	Real	T: dst_host_diff_srv_rate	●	M	comparison of IEs
f36	Real	T: dst_host_same_src_port_rate	○	M	comparison of IEs
f37	Real	T: dst_host_srv_diff_host_rate	○	M	comparison of IEs
f38	Real	T: dst_host_serror_rate	●	H	state keeping

Table 1 continued

Id	Scale	feature	Rel	Eff	IPFIX IE
f39	Real	T: `dst_host_srv_serror_rate`	•	H	state keeping
f40	Real	T: `dst_host_rerror_rate`	•	H	state keeping
f41	Real	T: `dst_host_srv_rerror_rate`	◦	H	state keeping

In column 3, 'B' stands for *basic*, 'C' for *content* and 'T' for *traffic*. Column 4 (rel) shows our results for feature relevance. ◦ *negligible*, ◦ *low contribution* and • *strong contribution* (Sect. 4). Column 5 (eff) shows the effort to generate the feature: S-small, M-medium, H-high. Column 6 (IPFX IE) shows the IPFIX information elements that can be used to generate this feature or explains which additional steps (comparison of IEs or state keeping of connections) are necessary

variable, thus vectors become sparse and dissimilar in the huge universe and the exploration by classifiers becomes harder. This problem has a strong effect on some classification techniques, yet in clustering and anomaly detection in general (Zimek et al. 2012).

The performance degradation caused by irrelevant or redundant features varies depending on the classification technique. Naïve Bayes classifiers are robust dealing with irrelevant variables, but very vulnerable to the addition of correlated features (even if they are relevant) (Kohavi and John 1997). In contrast, SVM and *k*NN models are severely impaired by noisy and irrelevant variables. Also, some lineal models like LAR are very sensitive to noise data as well as correlated variables (Efron et al. 2004). In order to reduce the bias introduced by specific classifiers, we have used 3 different classification paradigms for validation (DTC, *k*NN and Naïve Bayes) and up to 5 for evaluation (adding ANN and SVM models to the previous group). Additionally, we also deploy a LASSO-LAR-based classifier when using this specific method for feature selection.

4.2 Feature selection versus feature extraction

A habitual misconception related to feature selection comes from the subtle difference between the two methods that are set to carry out *dimensionality reduction*, i.e., feature selection and feature extraction, also called feature construction (Guyon and Elisseeff 2003). *Feature selection* performs the removal of features that are irrelevant or redundant in posterior processes for data representation and classification. On the other hand, *feature extraction* consists of the projection of the original data set into a new space where the linear dependence of features (axis or variables of the new space) is minimized, causing therefore a reduction in the number of required features. An example within the intended application field can be seen in Wen and Chen (2012). Note that, in feature extraction, the resulting features are different from the original ones.

In spite of the fact that feature extraction—like PCA (principal component analysis), ICA (independent component analysis) or SVD (singular value decomposition)—can be used for feature weighting and therefore selection, it is important to notice that measurements from all the original variables contribute to define the final subspace, as well as the fact that only linear relationships among features are considered, i.e., higher order interactions are ignored. Such aspects make feature extraction methods usually not the most appropriate for feature selection. In any case, feature selection and feature extraction are not exclusive of each other; quite the opposite. The elimination of irrelevant features (feature selection) is a useful step before projecting features onto optimized spaces (feature extraction). Our work deals only with feature selection, not feature extraction.

4.3 Types of feature selection techniques

Feature selection techniques are embraced within three different groups (Guyon and Elisseeff 2003):

- *Wrappers*. A wrapper consists of a search algorithm for progressively selecting feature subsets and a predictive model (i.e., a classifier) to validate the selection, thus searching the subset with the best performance. Wrappers demand high computational cost and have a risk of overfitting; nevertheless, they are reliable in general and have the benefit of considering feature dependencies.
- *Filters*. With a much lower computational cost than wrappers, filters are fast and straightforward; they measure intrinsic properties of the data and are completely independent of the classification algorithm. Not being affected by subsequent inference phases can be an advantage, but it is also the main disadvantage of filters, as they are *blind* to see the influence of the selected feature subset in the classification performance.
- *Hybrids and/or Embedded*. In the middle of the two previous options, embedded solutions are those selection techniques which have been coupled to the classifier construction in the training phase. As a results, data exploitation is optimized as the predictive model is not retrained for every new subset.

The type of data to deal with is determinant to decide the appropriate feature selection method. In the intended case, the original data set contains mixed types, i.e., some features are nominal—e.g., the protocol type: tcp, udp, icmp—, and others belong to numerical ratio scales—e.g., time—(the complete list of features is shown in Table 1 and Table 7). This characteristic a priori makes approaches based on linear models not suitable, for instance: selection based on LASSO or SVM. Nevertheless, to enable numerical methods it is always possible to map nominal features in a continuous space, transforming every nominal value in a new feature that can equal 0 or 1 (*dummy coding*). The main drawback of such mapping is that it can cause a dramatic increase of dimensionality: data become more sparse, processing times in classifiers are extended and the original balance among features is distorted. In our application case for instance the feature *service* alone transforms into 69 new dummy variables if converted.

Finally, our goal was not only reducing the number of features, but also to understand their effect. As recommended in Guyon and Elisseeff (2003), we used filters to obtain and compare variable rankings and baseline results. Later on, we refined our search using wrappers, which made the most of the filters' inferences.

Considering the introduced aspects, we opted for the following methods:

- *Weight by Maximum Relevance (WMR)*
  WMR is a simple filter which measures the dependence between every feature x and the classification feature y (i.e., the label) using Fisher F-test scores, Pearson's linear correlation and mutual information (Blum and Langley 1997). A high dependence score reveals feature *relevance*. The metric to be used depends on the type of feature. Two numerical features are scored by linear correlation; F-test score is applied to compare a numerical feature with a nominal feature, finally, for two nominal features, mutual information is calculated.
- *Minimum Redundancy Maximum Relevance (MRMR)*
  A step forward with respect to WMR is carried out by the MRMR filter. MRMR performs a sequential forward exploration, iteratively adding features to the subset by considering the following aspects: *a)* the new feature must contain the *maximum relevance* regarding the label and *b)* the *minimum redundancy* with regard to the feature subset already selected.

As for the WMR case, relevance and redundancy assessments are calculated using Fisher F-test scores, Pearson's correlation and mutual information measurements. As for the parametrization in the conducted tests, the quotient of relevance and redundancy established the relevance redundancy relation. The interested reader is referred to Ding and Peng (2003) for additional information.

– *Significance Analysis for Microarrays (SAM)*
SAM is a type of filter that assigns a score to each feature according to the significance of its changes related to the classification labels. It is done by generating a series of t-test and taking into account the standard deviations of repeated measurements for each feature. SAM uses permutations of such repetitions in order to estimate the percentage of vectors identified by chance, establishing therefore a false discovery rate (Tusher et al. 2001).
An important characteristic of SAM is that it does not assume a priory any statistical structure in the data set, i.e., the data may not follow a normal distribution. As for the parametrization, s_0, a small constant that ensures the independence of the variance, was set to 0.1.
– *Least Absolute Selection and Shrinkage Operator (LASSO)*
Linear classification models can be also useful to perform feature selection by checking the coefficient vector β, which manifests the feature relevancy. The LASSO operator has been widely deployed in statistics and data mining. It was firstly proposed by Tibshirani (1994) as a linear estimator that provides interpretable models with feature coefficients that are exactly zero (therefore such features can be ignored).
For the experiments, we use a LAR algorithm for finding LASSO solutions where the shrinkage operator λ has been adjusted to 12.07 after a set of 100 exploratory fivefold cross validation analysis. LASSO is a hybrid feature selection method.
– *Stability Selection*
Stability selection has been recently proposed (Meinshausen and Bühlmann 2010) as a general method for feature selection that can be applied to many different kind of scenarios. It can be considered as a meta-technique that embeds any feature selection approach and improves the selection process by providing statistical consistency to the nested method. In short, it works by submitting the underlying selection method to randomized subsamples of half-size (with regard to the dataset under analysis); features are finally selected according to the rate of appearance throughout the conducted runs.
For our experiments we have followed the work by Meinshausen and Bühlmann (2010), deploying bootstraping (200 runs), and embedding WMR, MRMR, SAM and LASSO.
– *Stepwise Regression*
In stepwise regression the choice of suitable features is carried out by an automatic procedure, adding the best feature (or removing the worst) based on a greedy algorithm. The greedy algorithm wraps a classification model for the comparison of feature subsets, hence stepwise regression is a general term for designating forward selection and backward elimination wrappers.
For the experiments, we used both *forward selection* (FS) and *backward elimination* (BE), which equally deployed three different classification models—DTC, *k*NN and Naïve Bayes—for every test.

5 Feature generation costs

The 41 features collected in DARPA'98, KDD Cup'99 and NSL-KDD data sets represent typical observations used in IDS's and were generated from traffic observations in the network

and auditing processes on hosts. When creating the data set, the costs for feature generation was not an issue. Sniffers were used to capture full packet traces of the traffic in the network and auditing tools were installed on hosts. The observed data was stored and could be analysed offline. Nevertheless, in a live system the costs for feature generation is relevant. Network operators need to be able to generate those features in a timely and cost-efficient manner and the 41 features are not equal with regard to the effort required to generate them. If we manage to remove costly features and still achieve good detection results, we can achieve a higher resource reduction than when omitting features that are less costly to generate. In order to check if there is a relation between the usefulness of a feature and the cost to generate it, we provide a categorisation of features with regard to the costs for feature generation. We furthermore checked which of the features can be generated from standard flow measurements on routers and which features require additional analysis functions only available in IDS's.

5.1 Feature categorization

The NSL-KDD data set distinguishes basic, content and traffic features. Features of the category *basic* (*f1–f9*) can be deduced from packet headers (IP and transport header) of observed packets in the network. Nevertheless, the effort for generating them (in a live system) is not equal. Some features need only a simple inspection of header fields (e.g., *f2* `protocol`) and are standard features reported by flow measurements on routers (see Sect. 5.2). Others require a comparison of values (e.g., *f7* `land`) or even state keeping about the connection status (*f4* `flag`). Such functions demand additional effort, usually implemented in IDS's, e.g. Bro.[2]

Features of category *content* (*f10–f22*) require processing of packet content. Capturing whole packets is much more costly than just capturing packet headers. Furthermore, the packet payload must be inspected and application data needs to be reassembled. Thus deriving content features from network measurements is very costly. In addition, if communication is encrypted end to end, the packet payload can only be decrypted on the communicating hosts and not in the network. This requires measurements at all hosts in the network, i.e. even more effort.

Features of the category *traffic* (*f23–f41*) take statistics about previous connections into account. *f23–f31* contain statistics about connections to the same destination within the last 2 s, whereas *f32–f41* look at the last 100 connections to the same destination, regardless of the time period. All these features require to keep counters and tables for the different characteristics of the connections. But some of these features can be generated by inspecting packet headers (e.g. *f23* `count`, *f24* `srv_count`, *f32* `dst_host_count`), whereas others require reassembly of the segments of a connection (e.g., *f25* `serror_rate`).

5.2 Extracting features from standard IPFIX measurements

Typical tools for network analysis allow either packet capturing or flow measurements. Packet capturing collects all packets observed at an observation point that is typically installed on a router. Most tools allow to configure a snap size that defines how much of the packet payload is included in the packet capture. Flow measurements aggregate packets with common properties (e.g., common source, destination, protocol, ports) into a flow (Claise et al. 2013). Since packets are aggregated into flows, much less data needs to be stored, but also information

[2] The Bro Network Security Monitor, http://www.bro.org.

from individual packets gets lost. Cisco NetFlow is a broadly deployed flow measurement protocol provided on Cisco routers. The IP Flow Information Export (IPFIX) protocol is the future standard for flow measurements and provides a flexible flow definition (Claise et al. 2013). Since NetFlow (and in future IPFIX) is already available on many routers, it is beneficial if a feature can be deduced directly from IPFIX or NetFlow measurements.

We look at all features derived from network measurements and reason if they can be deduced from IPFIX or if any additional analysis is required. IPFIX flows are formed using a flow key, typically derived from packet header fields (Claise et al. 2013). The flow key definition in IPFIX is quite flexible and can include many different characteristics that allow to aggregate packets into flows. Nevertheless, most common is to use a flow key that distinguishes packets based on the 5-tuple from IP and transport header fields: source and destination IP addresses, source and destination port numbers and protocol type. Characteristics of a specific flow are then reported to a collector using IPFIX information elements. An overview of all currently defined IPFIX elements can be found in the IANA Registry.[3]

The classical 5-tuple flow key allows to distinguish TCP connections using IPFIX measurements. One difference is that IPFIX looks at unidirectional flows, whereas a TCP connection establishes a bidirectional communication. But based on addresses and port numbers (used in flow keys), flows can be matched to TCP connections (Trammell and Boschi 2008). `protocol_type` (*f2*) is part of the flow key. `service` (*f3*) can be derived from the port number. Those two features are easy to obtain from IPFIX. The duration of a flow (*f1*) and the number of bytes transmitted (*f5*, *f6*) can be reported in standard IPFIX information elements (`flowDurationMicroseconds`, `octetTotalCount`). `land` (*f7*) is not directly reported by IPFIX, but can be deduced by a simple comparison of addresses and ports in the flow key, i.e. some additional effort is necessary. Also some extra effort is needed to count the number of urgent packets (*f9*). Although IPFIX provides an information element for this (`tcpUrgTotalCount`), the router must check flags in the TCP header of each observed packet. `status` (*f4*) needs to keep the connection state of TCP connections and is not directly available from IPFIX records. `wrong_fragment` (*f8*) requires to count packets with wrong checksum within a TCP connection. This information is not directly available from IPFIX either.

Content features (*f10–f22*) require processing of packet content. This is not part of IPFIX and demands additional effort for capturing, storing and processing.

The feature `count` (*f23*) describes the number of connections to the same host in the past 2 s. This can be deduced by comparing flow keys of the observed flows and start times of flows (`flowStartSysUpTime`). So it can be done based on IPFIX, but requires some additional effort for keeping the statistics about past flows. Also the number of flows to the same host and to the same service (`same_srv_rate`, *f29*), the flows to the same service in the past seconds (`srv_count`, *f24*), the flows to the same host and different services (`diff_srv_rate`, *f30*) and flows to the same service and different hosts (`srv_diff_host_rate`, *f31*) can be equally deduced. The same is true for the features looking at services and destination hosts from the last 100 connections (*f32–f37*). For finding out the amount of connections with SYN errors (*f25, f26, f38, f39*) or REJ errors (*f27, f28, f40, f41*) the connection state has to be maintained, so these features are more costly.

We categorize the effort to generate a feature as small, medium or high. Table 1 shows the results of the feature selection process (relevance) together with our categorization of features

[3] IANA Registry for IP Flow Information Export (IPFIX) Protocol Information Elements, www.iana.org/assignments/ipfix?.

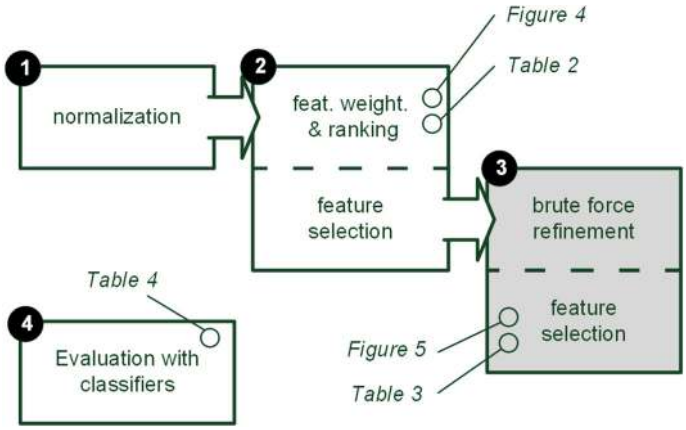

Fig. 2 Experiment scheme and steps

with regard to the effort to generate it. We also show which IPFIX information elements (IEs) can be used to report specific features.

Ideal are features that strongly contribute to anomaly detection and are easy to generate, i.e., *f3* (`service`). Furthermore, if a feature is costly to generate but contributes little to the detection, it is beneficial to omit it, i.e., *f8* (`wrong_fragment`).

6 Conducted experiments

Experiments have been carried out with MATLAB[4] and RapidMiner,[5] on a machine with the following characteristics: Intel(R) Core(TM) i5-2405S 2.50GHz, 8Gb RAM, Windows 7–64 bits. We focused on the analysis and selection of features for *anomaly detection*, postponing *anomaly identification* to subsequent work. Therefore, there were two possible labels for any vector: *normal* or *attack*.

6.1 Experiment phases

Figure 2 reflects the undertaken steps. They are as follows:

1. *Normalization*
   Databases were normalized using Z-transformation (statistical normalization), in a way that every feature got $\bar{x} = 0$ and $\sigma^2 = 1$.
2. *Feature weighting and ranking*
   Taking into account the whole database, different feature ranks were obtained with WMR, MRMR, SAM and LASSO, with and without applying *stability selection*. The obtained subsets as well as the original set containing 41 features were assessed by using five-fold cross validation (representative cases are shown in Table 2). Cross validation tests were repeated wrapping every of the introduced classification models able to deal with mixed nominal and numerical data, i.e. DTC, *k*NN and Naïve Bayes. In the LASSO case,

[4] MATLAB, http://www.mathworks.com.

[5] RapidMiner, http://rapidminer.com.

Table 2 Validation of subsets (WMR, MRMR, SAM, LASSO) after stability selection and fivefold cross validation

Method (#f)	Class.	Accuracy (%)	Precision (%)	Recall (%)	AUC	Time
WMR (15)	DTC	89.67 ± 0.10	96.98 ± 0.20	81.13 ± 0.18	0.905 ± 0.001	$3'50''$
	*k*NN	96.66 ± 0.09	97.01 ± 0.17	96.03 ± 0.07	0.978 ± 0.000	$26'10''$
	Bayes	88.31 ± 0.10	90.92 ± 0.21	84.12 ± 0.30	0.941 ± 0.001	$2''$
MRMR (11)	DTC	85.47 ± 0.27	99.78 ± 0.07	69.97 ± 0.57	0.913 ± 0.033	$3'27''$
	*k*NN	95.48 ± 0.05	96.14 ± 0.15	94.39 ± 0.07	0.968 ± 0.001	$15'24''$
	Bayes	87.33 ± 0.09	92.09 ± 0.21	80.60 ± 0.41	0.920 ± 0.001	$< 1''$
SAM (15)	DTC	95.51 ± 0.10	98.48 ± 0.25	92.11 ± 0.20	0.984 ± 0.000	$4'31''$
	*k*NN	98.45 ± 0.06	98.42 ± 0.08	98.37 ± 0.18	0.991 ± 0.001	$21'56''$
	Bayes	86.62 ± 0.15	93.81 ± 0.34	77.28 ± 0.59	0.946 ± 0.001	$2''$
LASSO (16)	DTC	95.72 ± 0.04	97.45 ± 0.13	93.56 ± 0.10	0.987 ± 0.000	$5'29''$
	*k*NN	98.91 ± 0.04	98.92 ± 0.08	98.80 ± 0.06	0.994 ± 0.000	$27'25''$
	Bayes	87.67 ± 0.16	90.77 ± 0.26	82.79 ± 0.23	0.945 ± 0.001	$2''$
	LASSO[a]	94.41 ± 0.16	96.45 ± 0.16	91.76 ± 0.22	0.976 ± 0.000	$50''$

[a] *Since LASSO-LAR is a hybrid method, we deploy it also as classifier after performing its own feature selection*

also a LASSO-based classifier was cross validated. By averaging filter results, we established a tentative baseline of features with a *strong contribution*. Results are shown in Fig. 4.

3. *Refinement with brute force search*
   Analysis with filters allowed us to reduce the scope so as to tackle an exhaustive search by means of stepwise regression. This step was done twofold:

   a) On one side, starting with the 41-feature set and running backward elimination (BE). The stop criterion was stated with the performance maximum.
   b) On the other side, taking the obtained *strong contribution* subset as baseline, forward selection (FS) added new features until the accuracy improvement was below a certain threshold $\Delta\gamma = 0.0001$ (being γ the classification accuracy with $\gamma = 1$ for a classification without errors). Previously, 2-step BE was also conducted to refine the *strong contribution* subset.

   Again, the three classifiers were models for validation. In order to reduce computational costs, stepwise regression runs with simple validation and splits the dataset into new train and test groups (70 % and 30 % of samples). After significant performances (turning points), fivefold cross validation was conducted to ensure the correctness of results obtained by simpler validations (Table 3). In short, this step refined the solution discovered by filters and generated the definitive assessment of features (Fig. 5).
4. *Evaluation with classifiers*
   In this final step we actually evaluated our proposed feature subsets with different classifiers comparing them to the original set with 41 features (Table 5). For this we used the original train/test division proposed by NSL-KDD authors.
   In addition to obtain a further validation of the proposed subsets, the aim of this final step was to state a precedent for further studies and show the rates achieved by simple classifiers with the proposed features. Future IDS's could use these results as benchmarks to build more sophisticated classifiers.

Table 3 Validation of significant subsets after fivefold cross validation

Subset (#f)	Class.	Accuracy (%)	Precision (%)	Recall (%)	AUC	Time
All (41)	DTC	98.93 ± 0.58	99.57 ± 0.05	98.20 ± 1.20	0.997 ± 0.001	$45'10''$
	*k*NN	99.21 ± 0.04	99.20 ± 0.03	99.16 ± 0.12	0.996 ± 0.000	$69'45''$
	Bayes	88.46 ± 0.11	87.90 ± 0.32	88.15 ± 0.46	0.947 ± 0.001	$3''$
SLC (30)	DTC	99.06 ± 0.11	99.36 ± 0.08	98.68 ± 0.23	0.997 ± 0.001	$32'47''$
	*k*NN	99.19 ± 0.04	99.16 ± 0.09	99.16 ± 0.09	0.996 ± 0.000	$37'39''$
	Bayes	88.84 ± 0.09	94.89 ± 0.60	81.18 ± 0.46	0.946 ± 0.001	$2''$
SC (16)	DTC	96.74 ± 0.47	95.28 ± 1.02	98.10 ± 0.44	0.995 ± 0.000	$11'52''$
	*k*NN	98.86 ± 0.07	98.77 ± 0.03	98.85 ± 0.15	0.994 ± 0.000	$21'29''$
	Bayes	88.84 ± 0.09	94.89 ± 0.60	81.18 ± 0.46	0.946 ± 0.001	$2''$

SLC strong and low contribution, *SC* strong contribution feature datasets

6.2 Classification models

– *Decision Tree Classifier (DTC)*
A DTC is a multistage decision making model, i.e., an approach that splits a complex decision into various simpler decisions, deploying a tree-like graph to schematize and represent the decision making process (Quinlan 1986). We here used a simple DTC in which pruning and pre-pruning was performed, with a confidence level of 0.25 for the pessimistic error calculation of pruning and 3 as pre-pruning alternatives. The criterion for splitting was based on entropy, adjusting the information gain of each feature to allow the breadth and uniformity of feature values. The minimal size for splitting was 4 samples, whereas the minimal leaf size was 2. The minimal gain required to split a node was fixed to 0.1. The maximal tree depth allowed was 20 nodes.
– *Naïve Bayes Classifier*
A naive Bayes classifier is a probabilistic classification model based on Bayes's theorem, which *naively* assumes that features are independent given class (Lewis 1998). Due to the data nature, we used *laplace correction* to prevent a strong effect of zero probabilities (Zadrozny and Elkan 2001).
– *k-Nearest Neighbor Classifier (kNN)*
A kNN classifier is a simple nonparametric classifier which links a new vector x to an existing class based on the votes (classification) of the k nearer training vectors to x (Samworth 2012). As for the tests, k was set to three neighbours, being the votes weighted based on the distance between neighbors. The metric for the distance was Euclidean-based.
– *Artificial Neural Network (ANN)*
An ANN is a computational model with a strongly parallel architecture that emulates biological neural networks. Among the multiple possibilities, we opted for a classic, widely used ANN consisting of a feed-forward multi-layer perceptron trained by a back propagation algorithm (Riedmiller 1994). We used a network with a hidden layer which size was $(\#features + \#classes)/2 + 1$; the training cycles were 500, the learning rate 0.3, the momentum 0.2, and the optimization was stopped if the error rate was below 1×10^{-5}.
– *Support Vector Machine (SVM)*
In short, SVM classify samples by trying to fix cluster boundaries in regions of data space where there is little data (Chapelle et al. 2002). Parameterization is very sensitive to the

subset under test. For the results shown in Table 5, we used a SVM with a radial kernel function; after running parameter optimization, we found the best performance when the kernel gamma equaled 0.01, the tolerance parameter C was 220.0, and the convergence epsilon was set to 0.01.

6.3 Performance indices

The validation of feature selection depended on the performances obtained by classifiers when working with the subsets under test. Scores were based on:

- *Accuracy*, i.e., percentage of correctly classified vectors.

$$accuracy = 100 \times \frac{\text{correctly identified vector}}{\text{total vectors}} \tag{1}$$

- *Attack precision* (or positive predictive value for attacks), i.e., the fraction of predicted attacks that are actually real attacks.

$$precision = 100 \times \frac{\text{correctly identified attacks}}{\text{vectors identified as attacks}} \tag{2}$$

- *Attack recall* (or sensitivity for attacks), i.e., the fraction of real attacks correctly predicted.

$$recall = 100 \times \frac{\text{correctly identified attacks}}{\text{real attacks}} \tag{3}$$

- *Area Under ROC Curve (AUC)*. The AUC is a metric to compare classifiers. In general terms, AUC is suitable for problems with biased or unbalanced class distribution. Moreover, the AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive sample higher than a randomly chosen negative sample (Fawcett 2006). It is important to remark that, even in situations where distinct classification approaches obtain even performances, binary classifiers (DTC, *k*NN) get worse AUC than probabilistic classifiers (e.g., Bayes-based). Moreover, as smartly analyzed in Hand (2009), AUC values can give potentially misleading results, hence they should be interpreted together with other indices.
- *Time*. We provide the time required by each classifier to carry out the fivefold cross validation, as well as the original train/test classification task, both with the diverse feature subset selections (Table 2 and Table 5 respectively). The objective is to show the relationship between *processing time*, *classification performance* and *number of features*.

7 Results

We here discuss our results on feature ranking, performance of selected subsets, benchmarking and processing time.

7.1 Feature weighting and ranking

Based on its inherent metrics and indices (Sect. 4.3), filters weight the contribution of every feature for the anomaly identification task. We normalize filters weights according to the maximum and minimum values and rearrange features in ascending order. Later on, nominal categories are visually stated according to pronounced changes in the weight slopes obtained by each filter, e.g WMR and SAM cases are shown in Fig. 3.

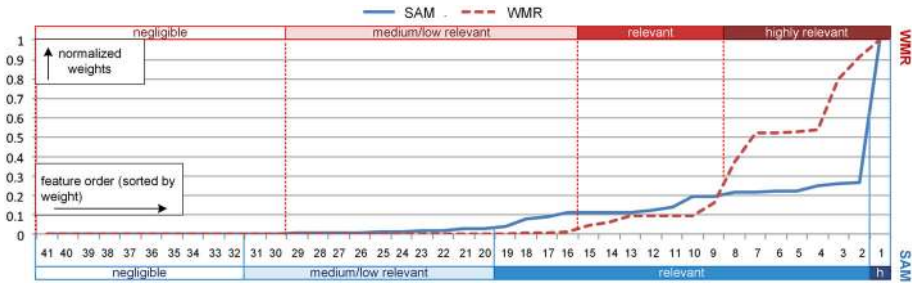

Fig. 3 Weighted, ranked features for the WMR and SAM cases

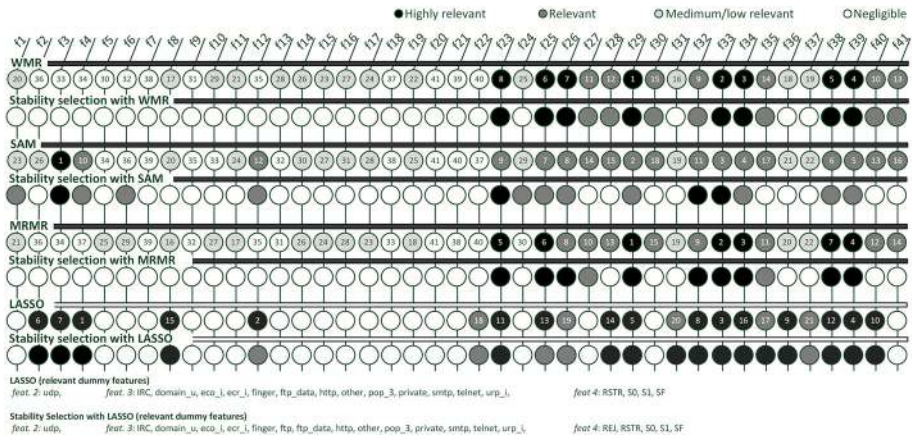

Fig. 4 Feature ranking and weighting by WMR, MRMR, SAM and LASSO methods, with and without *stability selection*

The feature rankings achieved with WMR, MRMR, SAM filters and LASSO, with and without *stability selection* are shown in Fig. 4. WMR and SAM filters are quick and straightforward, requiring less than 1 s to perform their rankings, whereas MRMR takes 18 s and LASSO 29 s to carry out their selection. Elapsed times for *stability selection* are obviously longer: 1′17″ for WMR, 27″ for SAM, 9′45″ for MRMR and 14′56″ for LASSO.

An initial statistical analysis of the data revealed obvious irrelevant features. For instance, feature *f20* (number of outbound commands) equals 0 for the whole data set. Thus all filters agreed ranking *f20* as the most irrelevant variable (rank 41 in Fig. 4). *f20* is usually scorned also in related works, except for Stein et al. (2005), which utilizes the KDD Cup'99 (where also *f20* is 0 for all observations).

Beyond this aspect, according to Fig. 4 the estimations of the diverse feature selection methods showed strong similarities, yet some disagreements too. All methods agreed about finding *traffic* features mostly relevant and *content* features mostly irrelevant, whereas they disagreed mainly when assessing *basic* features. In this respect, SAM and LASSO emphasized the contribution of *basic* features and WMR and MRMR completely ignored them. Table. 2 displays classification performances with the subsets obtained by WMR, MRMR, SAM and LASSO after *stability selection*. Classification results disclose that LASSO and SAM selections slightly outperformed WMR and MRMR, yet it is worth remarking the notable performance of the *k*NN classifier only with 11 features in the MRMR case.

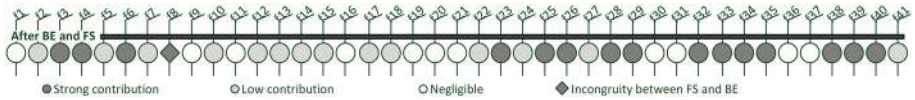

Fig. 5 Final feature classification after BE and FS refinement: 11 negligible, 14 low/medium contribution, 16 strong contribution

7.2 Refinement with brute force search

We used a subset based on a conservative estimation of filters and LASSO rankings for *strong contribution* (i.e., *f23, f25, f26, f29, f32, f33, f34, f38* and *f39*) and the 41 feature set as baselines for FS and BE stepwise regression respectively.

The final feature categorization after wrapper refinement is shown in Fig. 5. The meaning of the final three-color categories is as follows: The imperative feature subset (dark gray) led to an acceptable classification performance. From here on, the addition of new features (light gray) implied low improvements until a performance peak was finally reached. Then, the addition of new features (white) worsened the performance. Note that we have joined *relevant* and *highly relevant* features in the final *strong contribution* category.

Stepwise regression suggested that filters were able to correctly discover imperative features, but for some incongruities or disagreements due to the highly correlated nature of some features. In other words, there are features and combinations of features that provide very similar information and therefore exclude each other. This phenomenon is also the origin of the disagreement between BE and FS with regard to *f8*; BE labels it as irrelevant, whereas FS considers it as relevant. Specific cross-validation tests confirmed the inclusion of *f8* into the strong contribution group before other possible feature combinations. The same reason caused that, except for *f20*, filters arbitrarily ranked negligible and low contribution features. In this respect, it is important to remark that, beyond the contribution of a few set of features (the conservative subset), the inclusion of new features has a relative value, i.e. it is always possible to replace such new features with others that contain analogous information and lead to quite similar performances.

As for the required time, brute force refinement is a high time demanding process compared to filters (for instance, BE stepwise regression with normal validation using the *k*NN classifier requires 5h and 45′ to remove only one feature of the 16-feature subset; this time is $\times 5$ for the fivefold cross validation case). In the conducted tests, fivefold cross validation checks around turning points were run after visual inspection of normal validation results of the three classifiers together. Considering that such decisions are made in an automated manner, and taking into account that classifiers tests are serialized, the estimated time for the complete brute force refinement accounts for about 10 days of processing.

7.3 Comparison of the final subsets

Table 3 shows the validation of the final subset selections. Some interesting reasoning can be inferred by comparing performance outcomes:

- As a general rule, the high scores obtained by the selected classifiers regardless of the feature subset confirmed their suitability for feature selection validation (Bayes classifier remains under discussion).
- Starting from the 41-feature set, the removal of negligible features improved the classification up to taking into account 30 features. The 41- and 30-feature performances can be considered equivalent in terms of detection power.

Table 4 Subset competency based on DTC and *k*NN performance indices (*1st* indicates the best performance)

	AUC	Accuracy	Recall	Precision
1st	all (41)/SLC (30)	SLC (30)	SLC (30)	all (41)
2nd	SC (16)	all (41)	all (41)	SLC (30)
3rd	LASSO (16)	SC (16)	SC (16)	SAM (15)

SLC strong and low contribution, *SC* for strong contribution feature datasets. The number of included features is provided within brackets

– The performance degradation from 30 to 16 features is significantly low, hence the contribution for anomaly detection of the rejected features is also low.

Obviating the less reliable Bayes classifier, feature subsets of Tables 2 and 3 can be compared based on the provided indices. Such comparison is displayed in Table 4. AUC and *accuracy* are global evaluators, whereas a good *recall* performance is usually considered more important for anomaly detection than *precision* (i.e. in terms of security *false positives* are less harmful or preferable before *false negatives*).

In short, obtained outcomes suggest that, with less data to analyze, classifiers work in a simpler, less noisy space, improve criteria and reduce randomness in classification; on the other hand, there is a little information loss that otherwise could be useful to detect specific attacks.

7.4 Evaluation with classifiers

Results show that the original data set with 41 features contains redundant and irrelevant features. As a consequence, researchers that use the 41 features to detect anomalies should get similar or even better results when using the 30-feature subset, and minor degradation (or even non-existing) when using the 16-feature subset. We performed some tests to validate this statement, i.e., assessing classifiers working with the selected 41, 30 or 16 features and comparing results (Table 5). For this we used the originally proposed train/test NSL-KDD division.

Test outcomes confirmed the proposed feature selection classification introduced in Sects. 7.1, 7.3 and Table 3, as classifiers working with the 30-feature subset improve the performance if compared to the 41-feature case. Accuracy indices are not so excellent compared to some previous studies. Reasons for this have been already introduced: a) The more demanding NSL-KDD data set compared to DARPA'98 and KDD Cup'99, b) the original train/test partitioning, designed to challenge data mining studies c) our simple, straightforward classification models (since optimizing classifiers is not the focus of this work).

Noteworthy is the evolution of the Bayes classifier, which improved the performance by exchanging *precision* and *recall* rates. With 41 features, it was prone to consider almost any new vector as an attack, generating many false positives. With 16 features, false positives were reduced, but losing power to identify some anomalies. It does not imply that missing threats were distinguished with 41 features, performance indices suggest that the Naïve Bayes classifier was superficially dividing the input space, unable to dive in a deeper level to classify the data.

SVM are optimal for binary problems that involve high dimensional input spaces (i.e., many features) and myriads of samples (Joachims 1998). SVM showed specially sensitive to changes in the parameterization, demanding a new adjustment for every different subset

Table 5 Evaluation of significant subsets with the original train/test division

Subset (#f)	Classifier	Acc. (%)	Prec. (%)	Rec. (%)	AUC	Time
all (41)	DTC	75.53	78.12	79.19	0.799	$9'02''$
	*k*NN	77.55	96.62	62.76	0.753	$13'57''$
	Bayes	57.01	57.00	99.66	0.815	$1''$
	ANN	77.74	66.34	92.79	0.895	$145'50''$
	SVM	51.90	92.55	16.85	0.648	$21'11''$
SLC (30)	DTC	78.68	73.49	85.53	0.780	$2'35''$
	*k*NN	77.84	95.98	63.74	0.770	$6'06''$
	Bayes	75.61	92.46	62.23	0.883	$< 1''$
	ANN	75.99	92.04	63.84	0.802	$121'02''$
	SVM	74.40	88.54	63.21	0.884	$22'18''$
SC (16)	DTC	78.22	79.81	82.65	0.772	$53''$
	*k*NN	76.65	91.90	64.69	0.734	$3'48''$
	Bayes	73.38	92.61	57.84	0.879	$< 1''$
	ANN	79.25	90.35	70.96	0.885	$85'15''$
	SVM	78.83	87.12	73.70	0.855	$34'08''$

SLC strong and low contribution, SC strong contribution feature datasets

selection. In Table 5, results correspond to the set-up that obtained the best performance with the minimum number of features, after running parameter optimization with an evolutionary computation approach.

For the ANN and SVM tests nominal features were transformed to numerical features, significantly increasing the number of dimensions and the sparsity in the feature space. The inclusion or removal of nominal features implies a determinant effect on the input space overall shape. Irrespective of the relevance of the contained information, it can be either beneficial or detrimental when establishing partition boundaries in the SVM case (depending on which features are already selected). Considering feature by feature, density estimations revealed that both attack and normal clusters show strong deviation as a general rule, introducing an environment with non-determinant fluctuations in density and where areas of existence of both normal and attack classes frequently superimpose. Moreover, normal traffic accounts for very diverse types of operations, and could be even considered as a background noise wrapping attack profiles. On the other hand, the attack group is obviously not homogeneous, and is arranged in little sub-clusters that can also show high dispersion in several features. Taking into account how SVM establish classification boundaries, such characteristics suggest a very demanding scenario. In any case, a specific parameterization for every subset revealed a tendency to improve the performance as features were reduced.

7.5 Processing time

We provide the time required by filter feature selection validation (Table 2) as well as for the classification benchmarks (Table 5). As expected, as a consequence of utilizing less features by the involved algorithms computational costs were highly reduced without impairing classification performances, which kept acceptable rates, showed minor degradation or even improved.

The reasons behind the long-time periods required by ANN have to do with the database size growth (it is more than double in storage) when transforming nominal data to numerical data for enabling LASSO, ANN and SVM processing (explained in Sect. 4.3). On the other hand, input dimensionality does not have a high impact in overall complexity for the SVM, hence performance times depended more directly on sample size and convergence criteria inside training algorithms.

8 Conclusions

The main objective of this paper is to set a precedent where future IDS's can resort to when selecting network features to be deployed in their analysis. To that end, we have ranked features from a network traffic database according to a combination of feature selection filters and wrappers, concluding in: 16 features that shows a *strong contribution* for the anomaly detection task, 14 features with a *low contribution* and 11 features considered as *negligible*. Our experiments revealed the existence of irrelevant features, as well as a high redundancy among some features and inter-dependencies that account for a considerable group of anomalies.

Considering the recent critics and the disorder attributed to an appreciable part of the related literature, we have rigorously designed and performed our experiments, carrying out comparisons with previous approaches, discussing the reasons behind errors and drifts, and installing precautions to overcome such drawbacks. Finally, we show IDS benchmarks based on our selected subsets and five well-known classification models. The purpose is to state a baseline for future evaluation of IDS solutions.

Appendix

See Tables 6 and 7.

Table 6 Types of attacks and distributions in the NSL-KDD dataset—normal traffic accounts for 53.46 % (train) and 44.54 % (test)

Type	Attack	Train (%)	Test (%)	Description
Dos	back	0.76	1.75	Denial of service attack against apache webserver where a client requests a URL with many backslashes
	land	0.01	0.03	Denial of service where a remote host is sent a UDP packet with the same source and destination
	neptune	32.72	21.36	Syn flood denial of service on one or more ports
	pod	0.16	0.19	Denial of service ping of death
	smurf	2.10	3.05	Denial of service icmp echo reply flood
	teardrop	0.71	0.06	Denial of service where mis-fragmented UDP packets cause some systems to reboot
	processtable	–	0.14	The attack is launched against network services allocating a new process for each incoming TCP/IP connection (UNIX systems)
	mailbomb	–	1.34	The attacker sends many messages to a server, overflowing server's mail queue and possibly causing system failure

Table 6 continued

Type	Attack	Train (%)	Test (%)	Description
Probe	ipsweep	2.86	0.65	Surveillance sweep performing either a port sweep or ping on multiple host addresses
	nmap	1.19	0.33	Network mapping using the nmap tool. Mode of exploring network will vary – options include SYN
	portsweep	2.33	0.72	Surveillance sweep through many ports to determine which services are supported on a single host
	satan	2.88	3.77	Network probing tool (perl and C programs) which looks for well-known weaknesses
	saint	–	1.46	SAINT (Security Administrator's Integrated Network Tool). Although it is not conceived as an attack tool, it provides security information that is useful to an attacker
	mscan	–	4.57	Uses DNS zone transfers and brute force scanning of IP addresses to locate and test machines for vulnerabilities
R2L	guess_passwd	0.04	5.65	Guess passwords for a valid user using simple variants of the account name over a telnet connection
	ftp-write	0.01	0.01	Remote FTP user creates .rhost file in world writable anonymous FTP directory and obtains local login
	imap	0.01	0.00	Remote buffer overflow using imap port leads to root shell
	multihop	0.01	0.08	Multi-day scenario in which a user first breaks into one machine
	phf	0.00	0.01	Exploitable CGI script which allows a client to execute commands on a machine with a misconfigured web server
	spy	0.00	–	Multi-day scenario in which a user breaks into a machine to find important data trying to avoid detection
	warezclient	0.71	–	Users downloading illegal software which was previously posted via anonymous FTP by the warezmaster
	warezmaster	0.02	4.33	Anonymous FTP upload of Warez (usually illegal copies of copywrited software) onto FTP server
	xlock	–	0.04	A remote attacker gains local access by fooling a legal user who has left their X console ans password unprotected
	snmpgetattack	–	0.82	SNMP sweeps in order to find information about a specific system or compromising the remote device
	snmpguess	–	1.52	Action of "guessing" SNMP strings to obtain an access the attacker would not normally have
	named	–	0.08	Attack that exploits a buffer overflow in BIND version 4.9 releases prior to 4.9.7 and BIND 8 releases prior to 8.1.2
	sendmail	–	0.06	Attack that exploits a buffer overflow in sendmail v8.8.3 and allows a remote attacker to execute commands with superuser privileges

Table 6 continued

Type	Attack	Train (%)	Test (%)	Description
R2L	*worm*	–	0.01	Standalone malware program that replicates itself to spread to other computers, relying on security failures on the target computer to access it
	xsnoop	–	0.02	An attacker watches the keystrokes processed by an unprotected X server to gain information that can allow local access to the victim system
U2R	buffer_overflow	0.02	0.09	Buffer overflow using eject program on Solaris, the ffbconfig or the fdformat UNIX system command
	loadmodule	0.01	0.01	Non-stealthy loadmodule attack which resets IFS for a normal user and creates a root shell
	perl	0.00	0.01	Perl attack which sets the user id to root in a perl script and creates a root shell
	rootkit	0.01	0.06	Multi-day scenario where a user installs one or more components of a rootkit
	httptunnel	–	0.61	The attacker gains local access to a machine and then configures an http client to periodically query a web server setup at some remote host
	ps	–	0.07	Takes advantage of a race condition in the version of 'ps' distributed with Solaris 2.5 and allows an attacker to execute arbitrary code with root privilege
	xterm	–	0.06	Exploits a buffer overflow in the Xaw library distributed with some operating systems and allows an attacker to execute instructions with root privilege
	sqlattack	–	0.01	Code injection technique in which malicious SQL statements are inserted in an entry field for execution

In *italics* attacks only present in the test dataset. Attack descriptions have been mainly obtained from: http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/docs/attackDB.html (25 Sept. 2013)

Table 7 Description of features (information taken from *GureKddcup database description*, http://www.sc.ehu.es/acwaldap/gureKddcup/README.pdf, 10 Nov. 2013)

#	Name	Type	Description
f1	duration	Integer	Duration of the connection
f2	protocol_type	Nominal	Protocol type of the connection: TCP, UDP, ICMP
f3	service	Nominal	http, ftp, smtp, telnet. . . and others
f4	flag	Nominal	Connection status: SF, S0, S1, S2, S3, OTH, REJ, RSTO, RSTOS0, SH, RSTRH, SHR
f5	src_bytes	Integer	Bytes sent in one connection
f6	dst_bytes	Integer	Bytes received in one connection
f7	land	Binary	If source and destination IP addresses and port numbers are equal, this variable is 1, else 0
f8	wrong_fragment	Integer	Sum of bad checksum packets in a connection
f9	urgent	Integer	Sum of urgent packets in a connection (urgent bit activated)
f10	hot	Integer	Sum of hot actions in a connection such as: entering a system directory, creating programs and executing programs
f11	num_failed_logins	Integer	Number of incorrect logins in a connection

Table 7 continued

#	Name	Type	Description
f12	logged_in	Binary	If the login is correct then 1, else 0
f13	num_compromised	Integer	Sum of not found error appearances in a connection
f14	root_shell	Binary	If the root gets the shell then 1, else 0
f15	su_attempted	Binary	If the "su" command has been used then 1, else 0
f16	num_root	Integer	Sum of operations performed as root in a connection
f17	num_file_creations	Integer	Sum of file creations in a connection
f18	num_shells	Integer	Number of logins of normal users
f19	num_access_files	Integer	Sum of operations in control files in a connection
f20	num_outbound_cmds	Integer	Sum of outbound commands in a ftp session
f21	is_hot_login	Binary	If the user is accessing as root or adm
f22	is_guest_login	Binary	If the user is accessing as guest, anonymous or visitor
f23	count	Integer	Sum of connections to the same destination IP address
f24	srv_count	Integer	Sum of connections to the same destination port number
f25	serror_rate	Real	The percentage of connections that have activated the flag (f4) s0, s1, s2 or s3, among the connections aggregated in count (f23)
f26	srv_serror_rate	Real	The percentage of connections that have activated the flag (f4) s0, s1, s2 or s3, among the connections aggregated in srv_count (f24)
f27	rerror_rate	Real	The percentage of connections that have activated the flag (f4) REJ, among the connections aggregated in count (f23)
f28	srv_error_rate	Real	The percentage of connections that have activated the flag (f4) REJ, among the connections aggregated in count (f23)
f29	same_srv_rate	Real	The percentage of connections that were to the same service, among the connections aggregated in count (f23)
f30	diff_srv_rate	Real	The percentage of connections that were to different services, among the connections aggregated in count (f23)
f31	srv_diff_host_rate	Real	The percentage of connections that were to different destination machines among the connections aggregated in srv_count (f24)
f32	dst_host_count	Integer	Sum of connections to the same destination IP address
f33	dst_host_srv_count	Integer	Sum of connections to the same destination port number
f34	dst_host_same_srv _rate	Real	The percentage of connections that were to the same service, among the connections aggregated in dst_host_count (f32)
f35	dst_host_diff_srv_rate	Real	The percentage of connections that were to different services, among the connections aggregated in dst_host_count (f32)
f36	dst_host_same_src _port_rate	Real	The percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count (f33)

Table 7 continued

#	Name	Type	Description
f37	dst_host_srv_diff _host_rate	Real	The percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count (f33)
f38	dst_host_serror_rate	Real	The percentage of connections that have activated the flag (f4) s0, s1, s2 or s3, among the connections aggregated in dst_host_count (f32)
f39	dst_host_srv_serror _rate	Real	The percent of connections that have activated the flag (f4) s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_count (f33)
f40	dst_host_rerror_rate	Real	The percentage of connections that have activated the flag (f4) REJ, among the connections aggregated in dst_host_count (f32)
f41	dst_host_srv_error _rate	Real	The percentage of connections that have activated the flag (f4) REJ, among the connections aggregated in dst_host_srv_count (f33)

References

Bhuyan, M., Bhattacharyya, D., & Kalita, J. (2013) Network anomaly detection: Methods, systems and tools. *IEEE Communication Surveys and Tutorials, 16*(1), 1–34.

Blum, A. L., & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, *97*(1–2), 245–271.

Chapelle, O., Vapnik, V., Bousquet, O., & Mukherjee, S. (2002). Choosing multiple parameters for support vector machines. *Machine Learning*, *46*(1–3), 131–159.

Chebrolu, S., Abraham, A., & Thomas, J. P. (2005). Feature deduction and ensemble design of intrusion detection systems. *Computers & Security*, *24*(4), 295–307.

Claise, B., Trammell, B., & Aitken, P. (2013). Specification of the IP Flow Information Export (IPFIX) protocol for the exchange of flow information. In *RFC 7011 (INTERNET STANDARD)*. http://www.ietf.org/rfc/rfc7011.txt.

DARPA intrusion detection evaluation data sets. (1998). http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/.

Ding, C., & Peng, H. (2003). Minimum redundancy feature selection from microarray gene expression data. In *Proceedings of the IEEE bioinformatics conference* (pp. 523–528).

Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. *Annals of Statistics*, *32*, 407–499.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, *3*, 1157–1182.

Hand, D. (2009). Measuring classifier performance: A coherent alternative to the area under the ROC curve. *Machine Learning*, *77*(1), 103–123.

Imran, H. M., Abdullah, A. B., Hussain, M., Palaniappan, S., & Ahmad, I. (2012). Intrusions detection based on optimum features subset and efficient dataset selection. *International Journal of Engineering and Innovative Technology*, *2*(6), 265–270.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In C. Nédellec, & C. Rouveirol (Eds.), *Machine learning: ECML-98, Lecture Notes in Computer Science* (vol. 1398, pp. 137–142). Berlin: Springer.

KDD cup '99 data. (1999). https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

Khor, K. C., Ting, C. Y., & Amnuaisuk, S. P. (2009). A feature selection approach for network intrusion detection. In *International conference on information management and engineering (ICIME'09)* (pp. 133–137).

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, *97*(1–2), 273–324.

Lewis, D. D. (1998). Naive (bayes) at forty: The independence assumption in information retrieval. In C. Nédellec, & C. Rouveirol (Eds.), *Machine learning.: ECML-98, Lecture Notes in Computer Science* (vol. 1398, pp. 4–15). Berlin: Springer.

McHugh, J. (2000). Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security*, *3*(4), 262–294.

Meinshausen, N., & Bühlmann, P. (2010). Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *72*(4), 417–473.

Nguyen, H. T., Franke, K., & Petrovic, S. (2010). Towards a generic feature-selection measure for intrusion detection. In *20th International conference on pattern recognition (ICPR)* (pp. 1529–1532).

Panda, M., Abraham, A., Das, S., & Patra, M. R. (2011). Network intrusion detection system: A machine learning approach. *Int Decision Technologies*, *5*(4), 347–356.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106.

Riedmiller, M. (1994). Advanced supervised learning in multi-layer perceptrons from backpropagation to adaptive learning algorithms. *Computer Standards & Interfaces*, *16*(3), 265–278.

Salama, M. A., Eid, H. F., Ramadan, R. A., Darwish, A.,& Hassanien, A. E. (2011). Hybrid intelligent intrusion detection scheme. In A. Gaspar-Cunha, R. Takahashi, G. Schaefer, & L. Costa (Eds.), *Soft computing in industrial applications, no. 96 in advances in intelligent and soft computing* (pp. 293–303). Berlin: Springer.

Samworth, R. J. (2012). Optimal weighted nearest neighbour classifiers. *The Annals of Statistics*, *40*(5), 2733–2763.

Stein, G., Chen, B., Wu, A. S., & Hua, K. A. (2005). Decision tree classifier for network intrusion detection with GA-based feature selection. In *Proceedings of the 43rd annual Southeast regional conference, ACMACM-SE 43* (vol. 2, pp. 136–141). New York, NY, USA.

Sung, A.,& Mukkamala, S. (2003). Identifying important features for intrusion detection using support vector machines and neural networks. In *Proceedings of the symposium on applications and the internet* (pp. 209–216).

Sung, A.,& Mukkamala, S. (2004). The feature selection and intrusion detection problems. In *Proceedings of the 9th Asian computing science conference on advances in computer science (ASIAN'04)* (pp. 468–482). Berlin: Springer.

Syarif, I., Prugel-Bennett, A., & Wills, G. (2012). Data mining approaches for network intrusion detection: From dimensionality reduction to misuse and anomaly detection. *Journal of Information Technology Review*, *3*(2), 70–83.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. (2009). A detailed analysis of the KDD CUP 99 data set. In *IEEE symposium on computational intelligence for security and defense applications (CISDA)* (pp. 1–6).

Tavallaee, M., Stakhanova, N., & Ghorbani, A. (2010). Toward credible evaluation of anomaly-based intrusion-detection methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *40*(5), 516–524.

The NSL-KDD dataset. (2009). http://nsl.cs.unb.ca/NSL-KDD/.

Thottan, M., & Ji, C. (2003). Anomaly detection in IP networks. *IEEE Transactions on Signal Processing*, *51*(8), 2191–2204.

Tibshirani, R. (1994). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, *58*, 267–288.

Trammell, B., & Boschi, E. (2008). *Bidirectional Flow Export Using IP Flow Information Export (IPFIX)*. RFC 5103 (Proposed Standard). http://www.ietf.org/rfc/rfc5103.txt.

Tusher, V. G., Tibshirani, R., & Chu, G. (2001). Significance analysis of microarrays applied to the ionizing radiation response. *Proceedings of the National Academy of Sciences of the United States of America*, *98*(9), 5116–5121.

Wen, B., & Chen, G. (2012). Principal component analysis of network security data based on projection pursuit. In J. Lei, F. L. Wang, M. Li, & Y. Luo (Eds.) *Network computing and information security, no. 345 in communications in computer and information science* (pp. 380–387). Berlin: Springer.

Zadrozny, B., & Elkan, C. (2001). Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Proceedings of the eighteenth international conference on machine learning* (pp. 609–616), Morgan Kaufmann.

Zargari, S., Voorhis, D. (2012). Feature selection in the corrected KDD-dataset. In *2012 Third international conference on emerging intelligent data and web technologies (EIDWT)* (pp. 174–180).

Zimek, A., Schubert, E., & Kriegel, H. P. (2012). A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining*, *5*(5), 363–387.