A Multiple Model Cost-Sensitive Approach for Intrusion Detection

Wei Fan[1], Wenke Lee[2], Salvatore J. Stolfo[1], and Matthew Miller[1]

[1] Department of Computer Science, Columbia University
1214 Amsterdam Avenue Room 450, New York, NY 10027-7003, USA
{wfan,sal,mmiller}@cs.columbia.edu

[2] Department of Computer Science, North Carolina State University
Raleigh, NC 27695-7550, USA
wenke@csc.ncsu.edu

Abstract. Intrusion detection systems (IDSs) need to maximize security while minimizing costs. In this paper, we study the problem of building cost-sensitive intrusion detection models to be used for real-time detection. We briefly discuss the major cost factors in IDS, including consequential and operational costs. We propose a multiple model cost-sensitive machine learning technique to produce models that are optimized for user-defined cost metrics. Empirical experiments in off-line analysis show a reduction of approximately 97% in operational cost over a single model approach, and a reduction of approximately 30% in consequential cost over a pure accuracy-based approach.

1 Introduction

Intrusion Detection (ID) is an important component of infrastructure protection mechanisms. Many intrusion detection systems (IDSs) are emerging in the market place, following research and development efforts in the past two decades. They are, however, far from the ideal security solutions for customers. Investment in IDSs should bring the highest possible benefit and maximize user-defined security goals while minimizing costs. This requires ID models to be sensitive to cost factors. Currently these cost factors are ignored as unwanted complexities in the development process of IDSs.

We developed a data mining framework for building intrusion detection models. It uses data mining algorithms to compute activity patterns and extract predictive features, and applies machine learning algorithms to generate detection rules [7]. In this paper, we report the initial results of our current research in extending our data mining framework to build cost-sensitive models for intrusion detection. We briefly examine the relevant cost factors, models and metrics related to IDSs. We propose a multiple model cost-sensitive machine learning technique that can automatically construct detection models optimized for given cost metrics. Our models are learned from training data which was acquired from an environment similar to one in which a real-time detection tool may be deployed. Our data consists of network connection records processed from raw `tcpdump` [5] files using MADAM ID (a system for Mining Audit Data for Automated Models for Intrusion Detection) [7].

The rest of the paper is organized as follows: Section 2 examines major cost factors related to IDSs and outlines problems inherent in modeling and measuring the relationships among these factors. Section 3 describes our multiple model approach to reducing operational cost and a MetaCost [3] procedure for reducing damage cost and response cost. In Section 4, we evaluate this proposed approach using the 1998 DARPA Intrusion Detection Evaluation dataset. Section 5 reviews related work in cost-sensitive learning and discusses extensions of our approach to other domains and machine learning algorithms. Section 6 offers conclusive remarks and discusses areas of future work.

2 Cost Factors, Models, and Metrics in IDSs

2.1 Cost Factors

There are three major cost factors involved in the deployment of an IDS. Damage cost, *DCost*, characterizes the maximum amount of damage inflicted by an attack when intrusion detection is unavailable or completely ineffective. Response cost, *RCost*, is the cost to take action when a potential intrusion is detected. Consequential cost, *CCost*, is the total cost caused by a connection and includes DCost and RCost as described in detail in Section 2.2. The operational cost, *OpCost*, is the cost inherent in running an IDS.

2.2 Cost Models

The cost model of an IDS formulates the total expected cost of the IDS. In this paper, we consider a simple approach in which a prediction made by a given model will always result in some action being taken. We examine the cumulative cost associated with each of these outcomes: false negative (FN), false positive (FP), true positive (TP), true negative (TN), and misclassified hits. These costs are known as *consequential costs* (CCost), and are outlined in Table 1.

FN Cost is the cost of not detecting an intrusion. It is therefore defined as the damage cost associated with the particular type of intrusion i_t, DCost(i_t).

TP Cost is the cost incurred when an intrusion is detected and some action is taken. We assume that the IDS acts quickly enough to prevent the damage of the detected intrusion, and therefore only pay RCost(i_t).

FP Cost is the cost incurred when an IDS falsely classifies a normal connection as intrusive. In this case, a response will ensue and we therefore pay RCost(i), where i is the detected intrusion.

TN Cost is always 0, as we are not penalized for correct normal classification.

Misclassified Hit Cost is the cost incurred when one intrusion is incorrectly classified as a different intrusion – when i is detected instead of i_t. We take a pessimistic approach that our action will not prevent the damage of the intrusion at all. Since this simplified model assumes that we always respond to a predicted intrusion, we also include the response cost of the detected intrusion, RCost(i).

Table 1. Consequential Cost (CCost) Matrix

Outcome	CCost(c)
Miss (FN)	DCost(i_t)
False Alarm (FP)	RCost(i)
Hit (TP)	RCost(i_t)
Normal (TN)	0
Misclassified Hit	RCost(i) + DCost(i_t)
c: connection, i_t: true class, i: predicted class	

2.3 Cost Metrics

Cost-sensitive models can only be constructed and evaluated using given cost metrics. Qualitative analysis is applied to measure the relative magnitudes of the cost factors, as it is difficult to reduce all factors to a common unit of measurement (such as dollars). We have thus chosen to measure and minimize CCost and OpCost in two orthogonal dimensions.

An intrusion taxonomy must be used to determine the damage and response cost metrics which are used in the formulation of CCost. A more detailed study of these cost metrics can be found in our on-going work [8]. Our taxonomy is the same as that used in the DARPA evaluation, and consists of four types of intrusions: probing (PRB), denial of service (DOS), remotely gaining illegal local access (R2L), and a user gaining illegal root access (U2R). All attacks in the same category are assumed to have the same DCost and RCost. The relative scale or metrics chosen are shown in Table 2a.

Table 2. Cost Metrics of Intrusion Classes and Feature Categories

Category	DCost	RCost
U2R	100	40
R2L	50	40
DOS	20	20
PRB	2	20
normal	0	0

(a)

Category	OpCost
Level 1	1 or 5
Level 2	10
Level 3	100

(b)

The operational cost of running an IDS is derived from an analysis of the computational cost of computing the features required for evaluating classification rules. Based on this computational cost and the added complexity of extracting and constructing predictive features from network audit data, features are categorized into three relative levels. Level 1 features are computed using at most the first three packets of a connection. Level 2 features are computed in the middle of or near the end of a connection using information of the current connection only. Level 3 features are computed using information from all connections within a given time window of the current connection. Relative magnitudes are assigned to these features to represent the different computational costs as measured in a prototype system we have developed using NFR [10]. These costs are shown in Table 2b. The cost metrics chosen incorporate the computational cost as well as the availability delay of these features. It is important to note that level 1 and level 2 features must be computed individually. However, because all level 3 features require iteration through the entire set of connections in a given time window, all level 3 features can be computed at the same time, in a single iteration. This saves operational cost when multiple level 3 features are computed for analysis of a given connection.

3 Cost-Sensitive Modeling

In the previous section, we discussed the consequential and operational costs involved in deploying an IDS. We now explain our cost-sensitive machine learning methods for reducing these costs.

3.1 Reducing Operational Cost

In order to reduce the operational cost of an IDS, the detection rules need to use low cost features as often as possible while maintaining a desired accuracy level. Our approach is to build multiple rulesets, each of which uses features from different cost levels. Low cost rules are always evaluated first by the IDS, and high cost rules are used only when low cost rules can not predict with sufficient accuracy. We propose a multiple ruleset approach based on RIPPER, a popular rule induction algorithm [2].

Before discussing the details of our approach, it is necessary to outline the advantages and disadvantages of two major forms of rulesets that RIPPER computes, *ordered* and *un-ordered*. An ordered ruleset has the form **if** $rule_1$ **then** $intrusion_1$ **elseif** $rule_2$ **then** $intrusion_2$, . . . , **else** *normal*. To generate an ordered ruleset, RIPPER sorts class labels according to their frequency in the training data. The first rule classifies the most infrequent class, and the end of the ruleset signifies prediction of the most frequent (or default) class, *normal*, for all previously unpredicted instances. An ordered ruleset is usually succinct and efficient, and there is no rule generated for the most frequent class. Evaluation of an entire ordered ruleset does not require each rule to be tested, but proceeds from the top of the ruleset to the bottom until any rule evaluates to *true*. The features used by each rule can be computed one by one as evaluation proceeds. An un-ordered ruleset, on the other hand, has at least one rule for each class and there are usually many rules for frequently occurring classes. There is also a default class which is used for prediction when none of these rules are satisfied. Unlike ordered rulesets, all rules are evaluated during prediction and all features used in the ruleset must be computed before evaluation. Ties are broken by using the most accurate rule. Un-ordered rulesets are less efficient in execution, but there are usually several rules of varying precision for the most frequent class, *normal*. Some of these *normal* rules are usually more accurate than the default rule for the equivalent ordered ruleset.

With the advantages and disadvantages of ordered and un-ordered rulesets in mind, we propose the following multiple ruleset approach:

- We first generate multiple training sets T_{1-4} using different feature subsets. T_1 uses only cost 1 features. T_2 uses features of costs 1 and 5, and so forth, up to T_4, which uses all available features.
- Rulesets R_{1-4} are learned using their respective training sets. R_4 is learned as an ordered ruleset for its efficiency, as it may contain the most costly features. R_{1-3} are learned as un-ordered rulesets, as they will contain accurate rules for classifying *normal* connections.
- A *precision* measurement p_r[1] is computed for *every rule*, r, except for the rules in R_4.
- A threshold value τ_i is obtained for every single class, and determines the tolerable precision required in order for a classification to be made by any ruleset except for R_4.

In real-time execution, the feature computation and rule evaluation proceed as follows:

- All cost 1 features used in R_1 are computed for the connection being examined. R_1 is then evaluated and a prediction i is made.
- If $p_r > \tau_i$, the prediction i will be fired. In this case, no more features will be computed and the system will examine the next connection. Otherwise, additional features required by R_2 are computed and R_2 will be evaluated in the same manner as R_1.
- Evaluation will continue with R_3, followed by R_4, until a prediction is made.
- When R_4 (an ordered ruleset) is reached, it computes features as needed while evaluation proceeds from the top of the ruleset to the bottom. The evaluation of R_4 does not require any firing condition and will always generate a prediction.

The OpCost for a connection is the total computational cost of all unique features used before a prediction is made. If any level 3 features (of cost 100) are used at all, the cost is counted only once since all level 3 features are calculated in one function call.

This evaluation scheme is further motivation for our choice of learning R_{1-3} as un-ordered rulesets. If R_{1-3} were learned as ordered rulesets, a *normal* connection could not be predicted until R_4 since the default *normal* rules of these rulesets would be less accurate than the default rule of R_4. OpCost is thus reduced, resulting in greater system throughput, by only using low cost features to predict normal connections.

[1] Precision describes how accurate a prediction is. Precision is defined as $p = \frac{|P \cap W|}{|P|}$, where P is the set of predictions with label i, and W is the set of all instances with label i in the data set.

The precision and threshold values can be obtained during model training from either the training set or a separate hold-out validation set. Threshold values are set to the precisions of R_4 on that dataset. Precision of a rule can be obtained easily from the positive, p, and negative, n, counts of a rule, $\frac{p}{p+n}$. The threshold value will, on average, ensure that the predictions emitted by the first three rulesets are not less accurate than using R_4 as the only hypothesis.

3.2 Reducing Consequential Cost

The MetaCost algorithm, introduced by Domingos [3], has been applied to reduce CCost. MetaCost re-labels the training set according to the cost-matrix and decision boundaries of RIPPER. Instances of intrusions with $DCost(i) < RCost(i)$ or a low probability of being learned correctly will be re-labeled as *normal*.

Table 3. Intrusions, Categories and Sampling

U2R		R2L		DOS		PRB	
buffer_overflow	1	ftp_write	4	back	1	ipsweep	1
loadmodule	2	guess_passwd	1	land	1	nmap	1
multihop	6	imap	2	neptune	$\frac{1}{20}$	portsweep	1
perl	6	phf	3	pod	1	satan	1
rootkit	2	spy	8	smurf	$\frac{1}{20}$		
		warezclient	1	teardrop	1		
		warezmaster	1				

4 Experiments

4.1 Design

Our experiments use data that were distributed by the 1998 DARPA evaluation, which was conducted by MIT Lincoln Lab. The data were gathered from a military network with a wide variety of intrusions injected into the network over a period of 7 weeks. The data were then processed into connection records using MADAM ID. The processed records are available from the UCI KDD repository as the 1999 KDD Cup Dataset [11]. A 10% sample was taken which maintained the same distribution of intrusions and normal connections as the original data.[2] We used 80% of this sample as training data. For infrequent intrusions in the training data, those connections were repeatedly injected to prevent the learning algorithm from neglecting them as statistically insignificant and not generating rules for them. For overwhelmingly frequent intrusions, only 1 out of 20 records

[2] The full dataset is around 743M. It is very difficult to process and learn over the complete dataset in a reasonable amount of time with limited resources given the fact that RIPPER is memory-based and MetaCost must learn multiple bagging models to estimate probabilities.

were included in the training data. This is an ad hoc approach, but produced a reasonable ruleset. The remaining 20% of our sample data were left unaltered and used as test data for evaluation of learned models. Table 3 shows the different intrusions present in the data, the category within our taxonomy that each belongs to, and their sampling rates in the training data.

We used the training set to calculate the precision for each rule and the threshold value for each class label. We experimented with the use of a hold-out validation set to calculate precisions and thresholds. The results (not shown) are similar to those reported below.

4.2 Measurements

We measure expected operational and consequential costs in our experiments. The expected OpCost over all occurrences of each connection class and the average OpCost per connection over the entire test set are defined as $\frac{\sum_{c \in S_i} OpCost(c)}{|S_i|}$ and $\frac{\sum_{c \in S} OpCost(c)}{|S|}$, respectively, where S is the entire test set, i is a connection class, and S_i represents all occurrences of i in S. In all of our reported results, $OpCost(c)$ is computed as the sum of the feature computation costs of all unique features used by all rules evaluated until a prediction is made for connection c. CCost is computed as the cumulative sum of the cost matrix entries, defined in Table 1, for all predictions made over the test set.

4.3 Results

In all discussion of our results, including all tables, "RIPPER" is the single model learned over the original dataset, "Multi-RIPPER" is the respective multiple model, "MetaCost" is the single model learned using RIPPER with a MetaCost re-labeled dataset, and "Multi-MetaCost" is the respective multiple model.

As shown in Table 5, the average OpCost per connection of the single MetaCost model is 191, while the Multi-MetaCost model has an average OpCost of 5.78. This is equivalent to the cost of computing only a few level 1 features per connection and offers a reduction of 97% from the single ruleset approach. The single MetaCost model is 33 times more expensive. This means that in practice we can classify most connections by examining the first three packets of the connection at most 6 times. Additional comparison shows that the average OpCost of the Multi-RIPPER model is approximately half as much as that of the single RIPPER model. This significant reduction by Multi-MetaCost is due to the fact that R_{1-3} accurately filter *normal* connections (including low-cost intrusions relabeled as *normal*), and a majority of connections in real network environments are *normal*. Our multiple model approach thus computes more costly features only when they are needed to detect intrusions with $DCost > RCost$. Table 4 lists the detailed average OpCost for each connection class. It is important to note that the difference in OpCost between RIPPER and MetaCost models is explainable by the fact that MetaCost models do not contain (possibly costly) rules to classify intrusions with $DCost < RCost$.

Table 4. Average OpCost per Connection Class

IDS	RIPPER	Multi-RIPPER	MetaCost	Multi-MetaCost
back	223	143	191	1
buffer_overflow	172	125.8	175	91.6
ftp_write	172	113	146	71.25
guess_passwd	198.36	143	191	87
imap	172	107.17	181	108.08
ipsweep	222.98	100.17	191	1
land	132	2	191	1
loadmodule	155.33	104.78	168.78	87
multihop	183.43	118.43	182.43	100.14
neptune	223	100	191	1
nmap	217	119.63	191	1
normal	222.99	111.14	190.99	4.99
perl	142	143	151	87
phf	21	143	191	1
pod	223	23	191	1
portsweep	223	117.721	191	1
rootkit	162	100.7	155	63.5
satan	223	102.84	191	1
smurf	223	143	191	1
spy	131	100	191	46.5
teardrop	223	23	191	1
warezclient	223	140.72	191	86.98
warezmaster	89.4	48.6	191	87

Table 5. Average OpCost per Connection

	RIPPER	Multi-RIPPER	MetaCost	Multi-MetaCost
OpCost	222.73	110.64	190.93	5.78

Table 6. CCost and Error Rate

	RIPPER	Multi-RIPPER	MetaCost	Multi-MetaCost
CCost	42026	41850	29866	28026
Error	0.0847%	0.1318%	8.24%	7.23%

Table 7. Precision and Recall for Each Connection Class

		RIPPER	Multi-RIPPER	MetaCost	Multi-MetaCost
back	TP	1.0	1.0	0.0	0.0
	p	1.0	1.0	na	na
buffer_overflow	TP	1.0	1.0	0.8	0.6
	p	1.0	1.0	0.67	0.75
ftp_write	TP	1.0	0.88	0.25	0.25
	p	1.0	1.0	1.0	1.0
guess_passwd	TP	0.91	0.91	0.0	0.0
	p	1.0	1.0	na	na
imap	TP	1.0	0.83	1.0	0.92
	p	1.0	1.0	1.0	1.0
ipsweep	TP	0.99	0.99	0.0	0.0
	p	1.0	1.0	na	na
land	TP	1.0	1.0	0.0	0.0
	p	1.0	1.0	na	na
load_module	TP	1.0	1.0	0.44	0.67
	p	0.9	1.0	1.0	1.0
multihop	TP	1.0	1.0	1.0	0.86
	p	0.88	0.88	0.88	1.0
neptune	TP	1.0	1.0	na	na
	p	1.0	1.0	na	na
nmap	TP	1.0	1.0	0.0	0.0
	p	1.0	1.0	na	na
normal	TP	0.99	0.99	0.99	0.99
	p	0.99	0.99	0.92	0.93
perl	TP	1.0	1.0	1.0	1.0
	p	1.0	1.0	1.0	1.0
phf	TP	1.0	1.0	0.0	0.0
	p	1.0	1.0	na	na
pod	TP	1.0	1.0	0.0	0.0
	p	0.98	0.98	na	na
portsweep	TP	0.99	0.99	0.0	0.0
	p	1.0	1.0	na	na
rootkit	TP	1.0	0.6	0.5	0.2
	p	0.77	1.0	0.83	1.0
satan	TP	1.0	0.98	0.0	0.0
	p	0.99	0.99	na	na
smurf	TP	1.0	1.0	0.0	0.0
	p	1.0	1.0	na	na
spy	TP	1.0	1.0	0.0	0.0
	p	1.0	1.0	na	na
teardrop	TP	1.0	1.0	0.0	0.0
	p	1.0	1.0	na	na
warezclient	TP	0.99	0.99	0.0	0.9
	p	1.0	1.0	na	1.0
warezmaster	TP	0.6	0.6	0.0	0.0
	p	1.0	1.0	na	na

Table 8. Comparison with fcs-RIPPER

	Multi-MetaCost	MetaCost	fcs-RIPPER									
			$\omega = .1$	.2	.3	.4	.5	.6	.7	.8	.9	1.0
OpCost	5.78	191	151	171	191	181	181	161	161	171	171	171

Our CCost measurements are shown in Table 6. As expected, both MetaCost and Multi-MetaCost models yield a significant reduction in CCost over RIPPER and Multi-RIPPER models. These reductions are both approximately 30%. The consequential costs of the Multi-MetaCost and Multi-RIPPER models are also slightly lower than those of the single MetaCost and RIPPER models.

The detailed precision and TP[3] rates of all four models are shown in Table 7 for different connection classes. The values for the single classifier and multiple classifier methods are very close to each other. This shows that the coverages of the multiple classifier methods are identical to those of the respective single classifier methods. It is interesting to point out that MetaCost fails to detect *warezclient*, but Multi-MetaCost is highly accurate. The reason is that R_4 completely ignores all occurrences of *warezclient* and classifies them as *normal*.

The error rates of all four models are also shown in Table 6. The error rates of MetaCost and Multi-MetaCost are much higher than those of RIPPER and Multi-RIPPER. This is because many intrusions with $DCost < RCost$ are relabeled as *normal* by the MetaCost procedure. Multi-RIPPER misclassified such intrusions more often than RIPPER, which results in its slightly lower CCost and slightly higher error rate. Multi-MetaCost classifies more intrusions correctly (*warezclient*, for example) and has a lower CCost and error rate than MetaCost.

4.4 Comparison with fcs-RIPPER

In previous work, we introduced a feature cost-sensitive method, "fcs-RIPPER", to reduce OpCost [8,9]. This method favors less costly features when constructing a ruleset. Cost sensitivity is controlled by the variable $\omega \in [0, 1]$ and sensitivity increases with the value of ω. We generated a single ordered ruleset using different values of ω with fcs-RIPPER. In Table 8, we compare the average OpCost over the entire test set for the proposed multiple classifier method with that of fcs-RIPPER. We see that fcs-RIPPER reduces the operational cost by approximately 10%, whereas Multi-MetaCost reduces this value by approximately 97%. The expected cost of Multi-MetaCost is approximately 30 times lower than that of fcs-RIPPER, RIPPER, and MetaCost. This difference is significant.

5 Related Work

Much research has been done in cost-sensitive learning, as indicated by Turney's online bibliography [13]. Within the subset of this research which focuses on multiple models, Chan and Stolfo proposed a meta-learning approach to reduce consequential cost in credit card fraud detection [1]. MetaCost is another approach which uses bagging to estimate probabilities. Fan et al. proposed a

[3] Unlike precision, TP rate describes the fraction of occurrences of a connection class that were correctly labeled. Using the same notation as in the definition of precision, $TP = \frac{P \cap W}{W}$.

variant of AdaBoost for misclassification cost-sensitive learning [4]. Within research on feature-cost-sensitive learning, Lavrac et al. applied a hybrid genetic algorithm effective for feature elimination [6].

Credit card fraud detection, cellular phone fraud detection and medical diagnosis are related to intrusion detection because they deal with detecting abnormal behavior, are motivated by cost-saving, and thus use cost-sensitive modeling techniques. Our multiple model approach is not limited to IDSs and is applicable in these domains as well.

In our study, we chose to use an inductive rule learner, RIPPER. However, the multiple model approach is not restricted to this learning method and can be applied to any algorithm that outputs a precision along with its prediction.

6 Conclusion and Future Work

Our results using a multiple model approach on off-line network traffic analysis show significant improvements in both operational cost (a reduction of 97% over a single monolithic model) and consequential costs (a reduction of 30% over accuracy-based model). The operational cost of our proposed multiple model approach is significantly lower than that of our previously proposed fcs-RIPPER approach. However, it is desirable to implement this multiple model approach in a real-time IDS to get a practical measure of its performance. Since the average operational cost is close to computing at most 6 level 1 features, we expect efficient real-time performance. The moral of the story is that computing a number of specialized models that are accurate and cost-effective for particular subclasses is demonstrably better than building one monolithic ID model.

6.1 Future Work

It was noted in Section 2.2 that we only consider the case where a prediction made by a given model will always result in an action being taken. We have performed initial investigation into the utility of using an additional decision module to determine whether action is necessary based upon whether $DCost > RCost$ for the predicted intrusion. Such a method would allow for customizable cost matrices to be used, but may result in higher OpCost, as the learned model would make cost-insensitive predictions.

In off-line experiments, rulesets are evaluated using formatted connection records such that rulesets are evaluated after all connections have terminated. In real-time execution of ID models, a major consideration is to evaluate rulesets as soon as possible for timely detection and response. In other words, we need to minimize the detection delay. To achieve this, we can first translate each of the rulesets produced by our multiple model approach, each using different levels of features, into multiple modules of a real-time IDS. Since features of different levels are available and computed at different stages of a connection, we can evaluate our multiple models in the following manner: as the first packets arrive, level 1 features are computed and R_1 rules are evaluated; if a rule evaluates to *true* and that rule has sufficient precision, then no other checking for the connection is done. Otherwise, as the connection proceeds, either on a per-packet basis or multi-packet basis, level 2 features are computed and R_2 rules are evaluated. This process will continue through the evaluation of R_4 until a prediction is made. Our current single model approach computes features and evaluates rulesets at the end of a connection. It is thus apparent that this multiple model approach will significantly reduce the detection delay associated with the single model approach. However, it remains to be seen whether additional operational cost will be incurred because we must trigger the computation of various features at different points throughout a connection. We plan to experiment in the real-time evaluation of our multiple model approach using both NFR and Bro [12], two network monitoring tools used for real-time intrusion detection.

References

1. P. Chan and S. Stolfo. Towards scalable learning with non-uniform class and cost distribution: A case study in credit card fraud detection. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, August 1999. 151
2. W. W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, Lake Taho, CA, 1995. Morgan Kaufmann. 145
3. P. Domingos. MetaCost: A general method for making classifiers cost-sensitive. In *Proc. of the Fifth ACM SIGKDD Int'l. Conf. on Knowledge Discovery & Data Mining*, pages 155–164, San Diego, CA, 1999. ACM. 143, 147
4. W. Fan, S. Stolfo, J. X. Zhang, and P. K. Chan. Adacost: misclassification cost-sensitive learning. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 97–105, Bled, Slovenia, 1999. Morgan Kaufmann. 152
5. V. Jacobson, C. Leres, and S. McCanne. `tcpdump`. available via anonymous ftp to ftp.ee.lbl.gov, June 1989. 143
6. N. Lavrac, D. Gamberger, and P. Turney. Cost-sensitive feature reduction applied to a hybrid genetic algorithm. In *Proceedings of the Seventh International Workshop on Algorithmic Learning Theory*, pages 127–134, Sydney, Australia, 1996. 152
7. W. Lee. *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*. PhD thesis, Columbia University, June 1999. 142, 143
8. W. Lee, M. Miller, and S. Stolfo et al. Toward cost-sensitive modeling for intrusion detection. Technical Report CUCS-002-00, Computer Science, Columbia University, 2000. 144, 151
9. M. Miller. Learning cost-senstitive classification rules for network intrusion detection using ripper. Technical Report CUCS-035-99, Computer Science Department, Columbia University, December 1999. 151
10. Network Flight Recorder Inc. Network flight recorder. http://www.nfr.com/, 1997. 145
11. University of California at Irvine. UCI KDD archive. http://kdd.ics.uci.edu/. 147
12. V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proc. of the Seventh USENIX Security Symposium*, San Antonion, TX, 1998. 153

13. P. Turney. Cost-sensitive learning bibliographies. http://ai.iit.nrc.ca/bibiographies/cost-sensitive.html. 151