Anomaly Detection for Insider Threats: An Objective Comparison of Machine Learning Models and Ensembles

Filip Wieslaw Bartoszewski, Mike Just, Michael A. Lones, Oleksii Mandrychenko

Anomaly Detection for Insider Threats: An Objective Comparison of Machine Learning Models and Ensembles*

Filip Wieslaw Bartoszewski[1], Mike Just[1], Michael A. Lones[1], and Oleksii Mandrychenko[2]

[1] Heriot-Watt University, MACS department, Edinburgh, Scotland
[2] Fortinet UK Ltd, Edinburgh, Scotland

Abstract. Insider threat detection is challenging due to the wide variety of possible attacks and the limited availability of real threat data for testing. Most previous anomaly detection studies have relied on synthetic threat data, such as the CERT insider threat dataset. However, several previous studies have used models that arguably introduce bias, such as the selective use of metrics, and reusing the same dataset with the prior knowledge of the answer labels. In this paper, we create and test a host of models following some guidelines of good conduct to produce what we believe to be a more objective comparison of these models. Our results indicate that majority voting ensembles are a simple and cost-effective way of boosting the quality of results from individual machine learning models, both on the CERT data and on a version augmented with additional attacks. We include a comparison of models with their hyperparameters optimized for different target metrics.

Keywords: Anomaly detection · insider threat · machine learning · ensembles.

1 Introduction

An insider threat is a computer security issue involving attacks from inside the defensive perimeter of organisations. According to the Ponemon Institute, the average remediation cost of an insider credential threat was $871,686, with the average annual cost per company rising by 31% over the last two years [7]. While insider threats are an important security issue, there are challenges with studying them, including the lack of real world data. Companies are rarely willing to share information about attacks or behavioral logs for various reasons, including data privacy. As a result, researchers use synthetic datasets to train and test machine learning models. The CERT dataset [5] is the primary dataset for studying insider threats, though it may lack the variety required for extending our understanding of insider threats since each release contains only a handful of scenarios.

* Supported by The Datalab, https://www.thedatalab.com/

Furthermore, recent research on anomaly detection (including insider threat detection) has highlighted some apparent bias in several studies [4]. Practices contributing to this bias include selectively choosing metrics to present results, and designing models that are explicitly informed by the answer labels from the data. While there are examples of good practice, such as studies that use independent teams to create the attack data and detection models [20, 14], little research has taken an objective view of modeling insider threat attacks.

Our aim is to present an objective analysis and comparison of a variety of machine learning models and ensembles (combinations of models with a voting function) for detecting insider threats. Our work with ensembles focuses on heterogenous approaches, which combine different types of machine learning models. We compare several machine learning models for anomaly detection, including different versions with parameter choices optimized for separate quality metrics. We also compare the un-optimized and out-of-the box classifiers with their optimized counterparts, showing how optimizing models within an ensemble changes classifications. We present results of the models developed both on the CERT 4.2 release (*original dataset*) and four different attack scenarios that we recreated and inserted into test data (*augmented dataset*). To avoid bias, we do this *after* creating our models. We believe that our work contrasts the common practice in insider threat literature, where, in our opinion, models have been over-fitted by design and chosen after analysing the answer files.

2 Background

Researchers seem to agree that modeling insider threats is a complicated task [11, 12, 5, 2], leading many to instead model normal behavior and then detect outliers that could represent malicious activity. There are numerous examples of different types of anomaly detection for identifying insider threats, e.g., Hidden Markov Models (HMM), Naive Bayes, and Random Forest [12, 11, 10, 3, 14].

Researchers studying insider threat detection use several machine learning models that most frequently use either sequences of activities or feature vectors for their input. Those who represent the data as sequences of activities commonly use algorithms based on HMM, first introduced for detecting insider threats by Rashid et al. [14], who created sequences from week-long fragments of the CERT dataset. Other examples include Dahmane and Foucher [3], who prepared sequences for their HMM model with a clustering algorithm, and Lo et al. [12] who compared the performance of an HMM model against various distance measurement methods comparing sequence data. Yuan et al. [18] fed sequences of data into a Long Short-Term Memory (LSTM) neural network to receive a fixed size matrix and used a convolutional neural network for anomaly detection.

A different approach to represent input for machine learning models is to use feature vectors or their derivatives. Here, the input becomes a set of vectors that represent features, such as number of times an event repeats, day of the week, event type etc. Haidar and Gaber [6] split the types of features into

frequency, time-based, and nominal. They used these with One-Class Support Vector Machine (OCSVM) and Isolation Forest (IF) algorithms for ensemble-based anomaly detection. Other work with feature vectors includes Le et al.'s work on Self-Organising Maps (SOM) [9]. In the same paper, they used HMMs and decision trees to analyze the problem using both supervised and unsupervised models, using both sequences and feature vectors. They concluded that the unique characteristics of every model warrant further study. Legg et al.'s [11] work with user modeling involved comparing daily behavior for each user. The models formed tree-like structures with hierarchy reflecting how often an action was performed. Some authors, who wanted to capture more information from the data than simple activity counts, tried creating synthetic features that enumerate possible actions. Tuor et al. [16] created a 414-element feature vector, which was then used to create a feature matrix containing behaviors of all the users in the CERT dataset. It was subsequently used as input for Deep Neural Network (DNN) models, including a DNN and an LSTM-Recurrent Neural Network (RNN) combination, similar to Yuan et al.'s model [18].

Previous work on ensembles was done by Parveen et al. [13] and Haidar et al. [6]. In both cases, the ensembles were homogenous, created out of the same types of model, with each trained and tested on different folds of the dataset. Parveen et al. used ensembles made out of either one- or two-class SVM models and combined them to create an ensemble with a smaller false positive rate. Haidar et al.'s ensembles aggregated votes of OCSVM or IF models with oversampling techniques in order to flag non-malicious anomalies. Knowledge of those non-malicious anomalies was used while retraining the model, further decreasing false positive rates.

While anomaly detection seems to be the most feasible way of detecting insider threat, some researchers have created models of insider attacks. Agrafiotis et al. [1] analyzed numerous cases of insider attacks and tried to pinpoint common denominators linking the attack scenarios, ultimately creating several attack archetypes. Ruttenberg et al. [15] used such attack archetypes in an architecture similar to a Bayesian network (BN). Indicators and detectors assessed the probability of user behaviors being a part of a known archetype. The authors highlighted ease of transferability of the system between clients, but warned there was a "conceptual gap" where the indicators and detectors could not capture the concept of less common attacks. They also mentioned using optimization methods to improve the model's performance. While lacking a more detailed description of the optimization process, their chosen single metric, utility, improved almost twice in comparison to the original model's.

2.1 Critique of previous work

Previous research into anomaly detection of insider threat has often suffered from three key shortcomings. Firstly, results are often presented with a limited selection of metrics. Emmott et al. [4] highlighted such biased results where AUC (Area Under the ROC Curve) was used as the main, and sometimes only metric (e.g., [19]), or renaming existing metrics (e.g., referring to the commonly

accepted metric, *recall*, as a *true detection rate* [8]). Emmott et al. considered anomaly detection across different fields, with insider threat detection being given as an example of a high point difficulty problem — meaning that anomalies might be very similar to baseline behavior. Secondly, models are often created with prior knowledge of answers in the dataset. Recent research [20, 14] provides a nice counter-example to this practice whereby models were tested on attack data developed by a team separate to the one responsible for creating the detection models. Thirdly, results are not consistently presented alongside the limitations of the approach.

We posit that these pitfalls can be easily addressed by following the guidelines of good conduct [4]. Firstly, multiple metrics should be used to describe outcomes of the model. Secondly, models should ideally be created without prior knowledge of answers in the dataset. Finally, informing the readers about any shortcomings of the experiment design is vital, e.g., Lo et al. [12] highlight that their model was only tested for users known to contain attacks in their behavior.

3 Methodology

In this section we describe our dataset and feature extraction (Section 3.1), our machine learning models and their optimization (Section 3.2) as well as our ensemble creation (Section 3.3), our approach to adding attack examples to create an augmented dataset (Section 3.4) and our reporting metrics (Section 3.5).

3.1 Original dataset and feature extraction

The CERT dataset [5] is a synthetic dataset that was created as part of a project at Carnegie Mellon University to answer a specific problem in the insider threat field: the difficulty of obtaining data. The dataset was created using several interdependent systems to create log behaviors of a virtual organisation. The attacks in the CERT data were developed manually after consultations with counter-intelligence experts and included during the dataset creation. The authors highlighted that despite their efforts to introduce inconsistencies, the data was still cleaner than real-world equivalents.

There are several releases of the CERT dataset, varying in number of attacks, users, length, how advanced the text generation methods were, and others. We decided to use the release 4.2 (which we refer to as the *original dataset*), with 501 days of activity logs and 1000 users — 70 of whom performed attacks of 3 possible scenarios. The attacks could be events either occurring during a single day, or contain actions scattered over two months of user's activity. While there are newer releases of the CERT data available, r4.2 is often chosen due to the number of anomalies it contains. For example, at the time or writing, the newest dataset, r6.2, contains only 5 attacks (we recreate the scenarios introduced in r6.2 as our additional attack insertions for our *augmented dataset* — see Section 3.4). We believe that using this release will allow us, and other academics, to more easily compare results.

The CERT dataset (r4.2) contains behavior data for 1000 users over a period of a year and a half. We separated data into parts containing actions of single users. Each part consisted of samples representing a 24-hour day of a user's actions. We refer to samples and days of behavior interchangeably. A *sample* is a sequence of actions, sorted by the timestamps in the CERT dataset. We enumerated and encoded actions as numbers in a sequence. The sequence was changed into a fixed length feature vector containing counts of every action performed, for models requiring other type of input. The possible actions were: logging in, logging off, connecting a removable drive, disconnecting a removable drive, email events, file manipulation events, http access events (including browsing, downloading and uploading) and a parsing error event accounting for any unknown actions or dirty data. The first 21 days of every user's activity were used as the training set, with the rest of the data being the test set. We confirmed that no attacks occurred in those 21 days. This number was chosen as the average number of working days in a month.

3.2 Models and their optimization

We used several anomaly detection models (from the scikit-learn Python package [17]) for our analysis: Local Outlier Factor (LOF), One-Class Support Vector Machine (OCSVM/SVM), and Isolation Forest (IF). HMM models were implemented using the hmmlearn package. We chose LOF for its explainability and simplicity, SVM for its sensitivity to outliers, and HMM for its ability do identify sequences of events. The inclusion of the IF model was inspired by its use by Emmott et al. [4], since they recommended it as a "go-to model" that scaled up better than other solutions. Every model generated a label for every single sample it encounters: 1 for normal behavior and -1 for an anomaly.

Local Outlier Factor models calculate the local density of each sample based on the distance to its neighbors. Comparing those densities for each sample lets the model decide which cases are inliers (normal behavior) and which are outliers (anomalies, therefore suspected attacks).

One-Class Support Vector Machine models approximate a hyperplane that separates samples belonging to different classes during the training. During the classification step the labels are assigned based on which side of the plane the samples lie. The one-class variant can be used for anomaly detection, with the hyperplane encapsulating the inliers.

Hidden Markov Models assume that samples are created by a Markov process containing a number of hidden states. Every new part of the sample sequence corresponds to a change of the hidden state.

Isolation Forest models recursively partition the data, assuming that anomalies are easier to separate from the set than inliers. The model iteratively selects a feature, chooses a random value between minimum and maximum of that feature, and separates the data based on this value. The idea behind it is that, with samples representing common behavior forming a cluster, one needs more partitions to reach them than anomalies.

Suffix	Optimization
no suffix	Default
_acc	Accuracy
_auc	Area under the ROC curve
_rec	Recall
_f1	Fscore, (just SVM model)

Table 1. Model suffixes

Name	Models used
ensFirst	HMM_tt, LOF, SVM
ensBest	LOF_auc, LOF_rec, SVM
ensWorst	HMM_wb, SVM_acc, SVM_rec
ensSec	LOF, SVM, IF
ensSecAuc	IF_auc, LOF_auc, SVM_f1
ensSecAcc	IF_acc, LOF_acc, SVM_acc
ensSecRec	IF_rec, LOF_rec, SCM_rec
ensThird	HMM_tt, HMM_wb, IF, LOF, SVM
ensAll	HMM_tt, HMM_wb, IF, IF_acc, IF_auc, LOF_acc, LOF_auc, LOF_rec, SVM, SVM_acc, SVM_rec

Table 2. Ensembles

Motivated by the observation that previous research on insider threat detection has rarely used optimization, we decided to compare the results of our models when optimized for different performance metrics. To optimize our models we used gridsearchCV from scikit-learn library [17]: a method that exhaustively searches through given sets of model parameters to find the best combination for a chosen metric, e.g., accuracy. We optimized the LOF, SVM and IF models for accuracy, AUC and recall. Due to implementation issues with the library for the out-of-the-box AUC optimizer for SVM, optimizing the SVM model for AUC was unsuccessful — hence, we used F1 score. Two pairs of models, despite being optimized for different metrics had the same sets of parameters: Accuracy and F1 score for the SVM models, and Accuracy and Recall for the Isolation Forest models. We speculate that this was caused by a declared parameter space becoming too small when running the optimizer. When describing models or ensembles used in our work we use the optimized metric as a suffix to identify which version is used (see Table 1). All models were optimized on the original dataset.

The HMM models could not be optimized using the gridsearchCV methods due to not being implemented in the sciki-learn library. Rather, our code used the hmmlearn library, which was not compatible with the gridsearchCV optimizer method. To address this, we created two different versions of the HMM model, HMM_tt and HMM_wb which had different anomaly thresholds. HMM_tt calculates the threshold based only on the training samples, while HMM_wb uses both training and test data.

3.3 Ensembles

Ensembles are classifiers that aggregate results of other classifiers. They can be used to reduce the bias or variance of the models to create a new model more powerful than the sum of its parts. We decided to build on the work of Parveen et al. [13] and Haidar and Gaber [6] and created several ensembles that group different types of models instead of just one (with separate instances learning on different folds of the training set).

We aimed to create an ensemble that was computationally effective and worked in a way that was intuitive and transparent for the user. In this study we

used *majority voting*, since this was the easiest to interpret scheme for combining the outputs of individual models. The ensemble added up the classifications of the models. The majority decided a final verdict. We decided to create ensembles with only odd number of models that could vote. This was done to avoid a tied vote between the models. Some of the ensembles utilized optimized versions of models (see Section 3.2). For the full list of the created ensembles and what models were used, see Table 2.

ensFirst The first trial ensemble we designed, with the goal of quickly putting together models with diverse ways of learning from data.

ensBest and ensWorst Those two ensembles were built after we ran experiments for single models described in Section 4.1. We wanted to create ensembles out of the three best and worst models respectively.

ensSec We wanted to create an ensemble with every single model in it controlling for versions optimized for a different metric. We created four versions of this ensemble, one default with unoptimized models, and three with models optimized for accuracy, AUC and recall.

ensThird This ensemble was created to utilize all unoptimized types of machine learning models we present in this study in one ensemble: LOF, SVM, IF and HMM.

ensAll This ensemble was created to test if an ensemble that aggregated results of a larger number of models performed better, including several versions of the same model types optimized for different metrics.

3.4 Creating an augmented dataset

In order to test our models on new attacks not present in the original CERT dataset release, we developed a flexible generation and insertion pipeline to produce an *augmented dataset*. Firstly, we randomly selected a set of 28 benign users from the original (r4.2) dataset. We chose 28 users since we created 28 combinations of attack scenarios and anomaly frequency (described below).

Secondly, we set the number of insertions to be present in the data, which corresponded to the desired anomaly frequency rate — in our case between 1 and 7 attacks per user. The insertions were a sequence of events that represented the attack in the data. We read the benign user data and randomly chose a number of days, corresponding to the anomaly frequency, as the desired days of attacks.

Thirdly, we selected a scenario to generate and insert into the data. We selected three attack scenarios to recreate from a different release of the CERT dataset (Scenarios 1, 4 and 5 from r6.2), keeping their original numbering, as well as creating our own, Scenario 6. Note that we had already created our models prior to becoming aware of the details of the r6.2 dataset, thereby providing some independence, and reducing bias, in our model creation. Scenario 1 involves a person copying sensitive files on a pen drive and uploading them to Wikileaks. Scenario 4 is a person sending confidential files to their private email. Scenario 5 is a person browsing through restricted files and uploading them to Dropbox. Scenario 6 is a person copying confidential files onto a pen drive. Our generation

function created sequences representing the scenario and inserted them into a chosen sample. Every attack was generated separately, adding noise to the attack (e.g., different times of attack or number of sensitive files stolen), even within actions of the same user. We kept track of which samples were altered to create ground truth labels for the anomaly detection models. At this stage the data was ready for the model to be trained and tested. The augmented dataset consists of actions of only 28 malicious users. Each of the four scenarios has between one and seven days inserted with attack actions. We reused CERT data to create it.

3.5 Result metrics

We wanted to ensure that our results were easy to compare and recreate, especially for researchers interested in generating their own synthetic data insertions. With this in mind, we chose to use the area under the ROC curve (AUC), Accuracy and Recall. AUC is a number describing the area under the plot of true positive rate against the false positive rate, with 1 characterising a perfect model and 0.5 being analogous to a random choice. AUC is the most commonly reported metric in anomaly-based insider threat detection. Accuracy (Acc) was chosen as a metric affected by anomaly threshold of the model and not calculated solely based on the internal scoring function like AUC, and recall (Rec) as a measure of how many attacks were detected, without information about false positives.

Our choice of reporting additional metrics was inspired by existing guidelines laid out by Emmott et al. [4]. In particular, we wanted to be transparent on the documented results of our models even if they did not perform well.

4 Results

Every model has a separate instance for every user from the data. We will refer to these instances as user models. The results presented in Tables 3 and 4 show the means calculated from the results of all the user models in the corresponding dataset. The tables contain results from both default and optimized models and have the models sorted from top to bottom based on their AUC.

4.1 Evaluation of models on original data

Default SVM has the best AUC and recall, though it also has the second worst accuracy (with SVM_acc scoring lower). The LOF models have the most consistent scores across the non-ensemble models. EnsFirst has the highest average of all the metrics in comparison to the other models and ensembles. IF and HMM models have comparable results, with mediocre AUC between 0.651 and 0.629, but high accuracy, comparable to LOF models and ensembles.

4.2 Evaluation of models on augmented data

Since we designed the models with knowledge of the original dataset, to follow one of the guidelines laid out in Section 2.1, we evaluated the models on the

Classifier	AUC	Recall	Accuracy
SVM	0.956	0.992	0.238
LOF_rec	0.937	0.784	0.962
LOF_auc	0.935	0.785	0.961
ensFirst	0.916	0.882	0.919
ensAll	0.913	0.512	0.956
ensThird	0.901	0.639	0.945
ensBest	0.895	0.785	0.961
ensSec_acc	0.859	0.633	0.928
ensSec	0.857	0.848	0.899
LOF	0.843	0.810	0.927
LOF_acc	0.839	0.871	0.926
ensSec_auc	0.836	0.570	0.950
ensSec_rec	0.826	0.105	0.969
IF	0.651	0.225	0.928
IF_auc	0.640	0.082	0.969
IF_acc/rec	0.639	0.038	0.987
HMM_wb	0.629	0.034	0.979
HMM_tt	0.629	0.626	0.962
ensWorst	0.480	0.109	0.880
SVM_acc/f1	0.391	0.708	0.216
SVM_rec	0.148	0.103	0.830

Table 3. Model and ensemble results for original dataset, ordered by AUC

Classifier	AUC	Recall	Accuracy
ensFirst	0.966	1.000	0.909
LOF_auc	0.959	0.814	0.947
LOF_rec	0.958	0.814	0.947
SVM	0.955	1.000	0.265
ensThird	0.944	0.699	0.937
ensAll	0.943	0.585	0.942
LOF_acc	0.925	0.854	0.919
ensBest	0.912	0.814	0.947
LOF	0.863	0.693	0.917
ensSec	0.862	0.731	0.889
ensSec_acc	0.826	0.582	0.915
ensSec_auc	0.813	0.582	0.936
ensSec_rec	0.810	0.019	0.953
HMM_wb	0.767	0.112	0.904
HMM_tt	0.767	0.959	0.963
IF_auc	0.596	0.025	0.958
IF_acc/rec	0.590	0.008	0.971
IF	0.579	0.076	0.917
ensWorst	0.412	0.019	0.840
SVM_acc/f1	0.368	0.714	0.174
SVM_rec	0.052	0.025	0.843

Table 4. Model and ensemble results for augmented dataset, ordered by AUC

augmented dataset, whose creation is described in Section 3.4. In contrast to the earlier experiment (see Section 4.1), this dataset was much smaller (28 users instead of 1000) and contained only malicious users. Results of the experiment on this data can be found in Table 4. The model with the best AUC is ensFirst, and LOF_auc is the best non-ensemble. EnsFirst is tied with SVM for the best recall, but the latter still has poor accuracy. SVM_acc still has the worst accuracy of all the models. LOF models still have the best, most consistent scores out of non-ensemble models.

4.3 Comparison between default and optimized implementations

For results of optimized and default models, see Tables 3 and 4. The comparison of the ROC curves for each model family are shown in Figure 1. Since both HMM models have the same values for the scoring function, their ROC curves are overlapping. The internal scoring function of ensembles was the vote count, meaning that their ROC curve could have only two or three points of inflection, depending on the number of models that contributed to the ensemble, so we decided not to show them. The ROC curves for models run on the augmented dataset were sufficiently similar to those for the original dataset that we have also not included those curves.

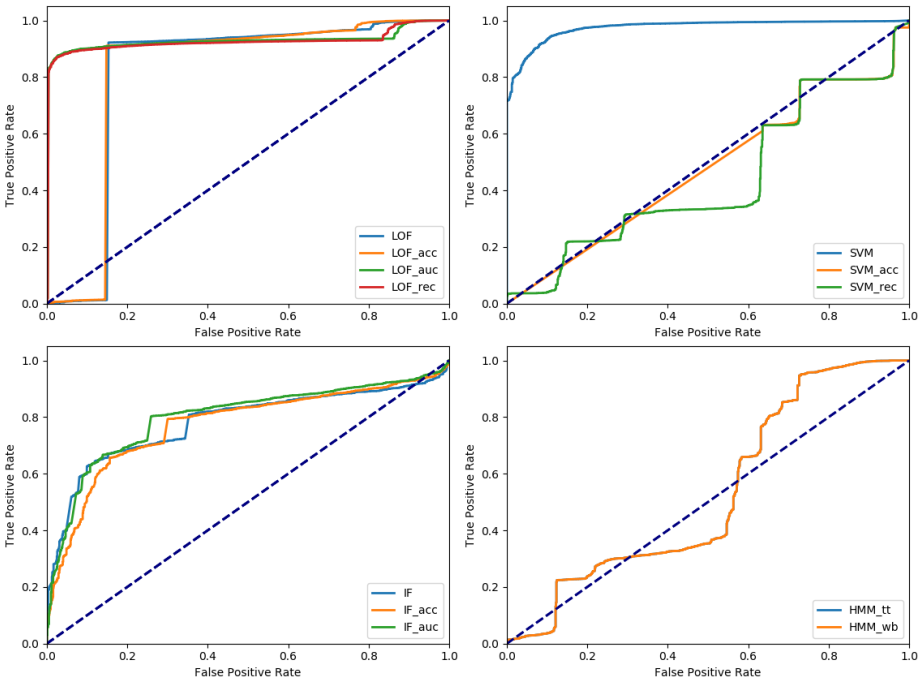

Fig. 1. Comparison of ROC curves of models tested on original CERT data

5 Discussion of Results

The model with the best results on the original dataset demonstrates the importance of reporting multiple metrics. For example, the default implementation of the SVM model showed high AUC=0.956 and Recall=0.992 and simultaneously one of the lowest accuracy scores, Acc=0.238, compared to the other models. This suggests that the SVM can be an overly sensitive classifier and we do not recommend using it to solve high point difficulty anomaly detection problems. The other versions of the SVM model were found to have similarly low results as both SVM models had the lowest AUC. SVM_acc had the lowest accuracy, and SVM_rec even produced a result of Rec=0.103, with only 3 other models displaying lower recall: IF_acc/rec, IF_auc and HMM_wb.

The ensemble models were among the highest scoring considering all the metrics, excluding the ensWorst ensemble, which was made specifically using the three models with the lowest AUC in the original experiment. The best performing, ensFirst, consisting of default versions of LOF, SVM and HMM_tt models, was also our first experiment in aggregating results from single models. We believe that using such ensembles could be an easy and cost-effective way of boosting detection rates of insider attacks. Implementing a voting system has a negligible cost in comparison to training and using singular models. Depending

on what sort of models the user needs, they can change the models voting in the ensemble in order to get a more precise or sensitive ensemble.

Overall, we rank the ensembles and LOF models as the best performing in our experiment, followed by IF and HMM, with SVMs being the least reliable family of classifiers in our work.

5.1 Verifying models on augmented data

The experiments we conducted to verify the models on inserted attacks have demonstrated similar results to the original CERT set. The results presented in Table 4 show similar trends to the original experiment in Table 3. Default SVM was no longer a model with best AUC and recall, overtaken by the ensemble ensFirst and models LOF_auc and LOF_rec. When looking at the results of detecting insertions we highlight that attacks were inserted randomly, therefore we did not account for the regular behavior of a user, as opposed to attacks originally present in the CERT dataset. Still, we hope that this limitation is somewhat offset by the fact that the attacks we recreated are either from different releases of CERT data (r6.2, where the original dataset uses r4.2) or have been designed to incur a minimal footprint in the behavior logs.

A further issue is that ens_first and default SVM had a recall of 1.0 — a perfect score. We assume this was potentially caused by a smaller size of the dataset used for the insertion experiment, i.e., 28 users as opposed to 1000. All 28 users had attacks inserted in their data, exhibiting a different rate of anomalies when compared to 70 attackers in the original CERT set. On average, the difference between metrics, depending on the dataset used, was 0.047 for AUC, 0.015 for Accuracy, and 0.068 for Recall. We believe that the differences were small enough (less than 5%) suggesting that our models should perform well on a different dataset.

5.2 Difference between default and optimized models

The models most positively affected by optimizing were LOF, and to a smaller degree, IF. Optimizing the LOF model for AUC improved the metric by 11%, optimizing for recall improved by 17% and optimizing for accuracy raised it by 3%, all in comparison to the default versions of the model. The IF model showed an improvement in AUC of 3% and in accuracy of 5% in respective optimizations. We highlight that the recall of the optimized models was lower than the default model, with the version specifically optimized for recall demonstrating worse results by 89%.

SVM and an optimized version of the ensSec ensemble scored poorer in the metric they were optimized for than the default counterpart. We believe that weaker performance of different variants of the ensemble was potentially caused by the votes of the SVM models. The only improvement in the SVM model was a better accuracy, increasing from 0.265 of the default model to 0.843 of the model optimized for recall. We want to emphasize that ROC curves shown in Figure 1 were plotted using aggregated labels and scoring functions from user

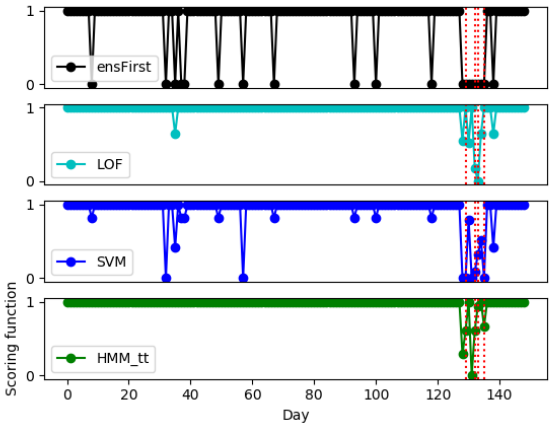

Fig. 2. Scoring function comparison for user KPC0073. Attacks in red.

models. This meant the AUC of plotted curves was different from the averaged AUC we reported in our results. Both of the HMM models had the same set of parameters, causing their scoring function values to be the same, see Figure 1.

5.3 Case study

In order to provide some more insight into our work we have included the following case study. We focused on user `KPC0073` from the original dataset who was a permanent employee of the simulated company. According to the backgroud readme information for the dataset, the attack performed by the user was a "new habit" of using removable drives and working after hours, finally resulting in using a pen drive and uploading data to a restricted website *wikileaks.org* [5]. Figure 2 shows a comparison of scoring functions of ensemble First and the models for this user. The ensembles performed relatively better, compared to standalone models. It is worth noting that the ensemble's scoring function is much less compressed in comparison to the models, as seen in Figure 2. All the votes at value '0' are unanimous, meaning that all three models creating the ensemble voted a sample anomalous. All of the votes at value '1' had two of the models voting benign and one voting malicious. Referring to the results from Table 3, we can assume that SVM is an oversensitive model that is being "kept in check" by more accurate LOF and HMM_tt models. Looking at the SVM model scoring function we can assume that the best way to improve that model in the future might be by adjusting its anomaly threshold.

6 Limitations and future work

We decided against using average precision of the models as one of our metrics, because depending on the choice of majority class, it would be near 1 or 0 for every single model. Similarly, we did not consider sliding models, where the model

re-learns based on new samples, which may be used to take behavioral drift into account. Our initial investigation of sliding models indicated poor performance and high computational cost. We would encourage others to investigate this topic, possibly on a real dataset. For the original dataset we used the fact that there are no attacks in training data. While we were able to confirm this using the CERT labels, we suggest that future modeling attempts should consider the implications of training with data that might contain insider threats. Our augmented dataset consists of only malicious users. Whilst this differs from the original dataset, we use augmented data to validate models on a per user basis, rather than comparing false positive rates with benign data.

Our feature extraction was based on a relatively small number of features. We encourage others to consider different feature sets, data representations, or deep learning models to investigate insider threats. Limited options could also affect our optimization work. GridsearchCV, while conducting an exhaustive search of model parameters, does so only within the dictionaries of parameter space passed to it. We suspect this might have affected some of the SVM models performing worse in the metric they were optimized for. Our approach to generating and inserting attacks was an initial investigation into addressing the problem with bias introduced by reusing the CERT data. Any future research should consider structured models for incorporating realistic attack data into datasets.

7 Conclusions

In this study, we compared several types of anomaly detection models, with versions optimized for different quality metrics. We also created a selection of majority voting ensembles using combinations of the anomaly detection models presented. We tested the models on the synthetic CERT dataset and a separate, smaller dataset created from a random selection of benign CERT users with new attacks inserted into it. The ensembles and LOF models had the best and most consistent results among all reported metrics.

We advocate the importance of following a set of guidelines while working on insider threat detection models. We would particularly encourage future studies to: report on multiple result metrics; design the models without prior knowledge of answers and labels or using a different dataset to create and evaluate the models; keep track and inform on any limitations of the approach discovered. In the future, we want to expand on this study by creating more feature sets for our models, expanding our ML model suite with neural net models e.g., autoencoders and their translation error for anomaly detection, creating ensembles that combine models using different feature sets and weighted voting schemes, conducting uncertainty and sensitivity analysis for our models and testing them on a non-synthetic dataset.

References

1. Agrafiotis, I., Nurse, J.R., et al.: Identifying attack patterns for insider threat detection. Computer Fraud & Security **2015**(7), 9–17 (jul 2015)

2. Cappelli, D., Moore, A., Trzeciak, R.: The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes. Addison-Wesley Professional (2012)
3. Dahmane, M., Foucher, S.: Combating Insider Threats by User Profiling from Activity Logging Data. In: ICDIS. pp. 194–199 (2018)
4. Emmott, A., Das, S., Dietterich, T., Fern, A., Wong, W.K.: A Meta-Analysis of the Anomaly Detection Problem (mar 2015), https://arxiv.org/abs/1503.01158
5. Glasser, J., Lindauer, B.: Bridging the gap: A pragmatic approach to generating insider threat data. Proceedings - IEEE CS Security and Privacy (2013)
6. Haidar, D., Gaber, M.M.: Adaptive One-Class Ensemble-based Anomaly Detection: An Application to Insider Threats. In: 2018 International Joint Conference on Neural Networks (IJCNN). pp. 1–9 (2018)
7. IBM: Cost of Insider Threats — ObserveIT (2020), https://www.observeit.com/cost-of-insider-threats/
8. Kim, J., Park, M., Kim, H., Cho, S., Kang, P.: Insider threat detection based on user behavior modeling and anomaly detection algorithms. Applied Sciences (Switzerland) **9**(19) (2019)
9. Le, D.C., Zincir-Heywood, A.N.: Evaluating Insider Threat Detection Workflow Using Supervised and Unsupervised Learning. In: 2018 IEEE Security and Privacy Workshops (SPW). pp. 270–275 (2018)
10. Le, D.C., Zincir-Heywood, N.: Exploring anomalous behaviour detection and classification for insider threat identification. International Journal of Network Management (July 2019), 1–19 (2020)
11. Legg, P.A., Buckley, O., Goldsmith, M., Creese, S.: Automated Insider Threat Detection System Using User and Role-Based Profile Assessment. IEEE Systems Journal **11**(2), 503–512 (2017)
12. Lo, O., Buchanan, W.J., et al.: Distance measurement methods for improved insider threat detection. Security and Communication Networks **2018**(January) (2018)
13. Parveen, P., Weger, Z.R., Thuraisingham, B., Hamlen, K., Khan, L.: Supervised Learning for Insider Threat Detection Using Stream Mining. In: 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence. pp. 1032–1039 (2011)
14. Rashid, T., Agrafiotis, I., Nurse, J.R.: A New Take on Detecting Insider Threats pp. 47–56 (2016)
15. Ruttenberg, B., Blumstein, D., Druce, J., et al.: Probabilistic Modeling of Insider Threat Detection Systems BT - Graphical Models for Security. pp. 91–98. Springer International Publishing (2018)
16. Tuor, A., Kaplan, S., Hutchinson, B., Nichols, N., Robinson, S.: Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams (oct 2017), http://arxiv.org/abs/1710.00811
17. Varoquaux, G., Buitinck, L., Louppe, G., et al.: Scikit-learn: Machine Learning in Python. GetMobile: Mobile Computing and Communications (1), 29–33 (2015)
18. Yuan, F., Cao, Y., Shang, Y., Liu, Y., Tan, J., Fang, B.: Insider Threat Detection with Deep Neural Network BT - Computational Science – ICCS 2018. pp. 43–54. Springer International Publishing (2018)
19. Yuan, F., Shang, Y., Liu, Y., Cao, Y., Tan, J.: Attention-Based LSTM for Insider Threat Detection, vol. 1116 CCIS. Springer Singapore (2019)
20. Zhang, H., Agrafiotis, I., Erola, A., et al.: A State Machine System for Insider Threat Detection. In: Graphical Models for Security. pp. 111–129. Springer (2019)