# Quantitative Analysis of Machine Learning Classification Algorithms for the Design of Intrusion Detection/Prevention Systems.

O. Cheche Agada, Ph.D.
Department of Information Sciences and Technology
College of Engineering and Computing
George Mason University
Fairfax, Virginia, USA
oagada@gmu.edu

***Abstract*—Intrusion Detection and Prevention systems play a very important role in preventing unwanted and unauthorized access to a computer network infrastructure. Traditional intrusion detection and prevention systems rely on static databases that are unable to cope with the dynamic sophistication of modern-day intrusions. This has led to the development of a new breed of systems that are reliant on the machine learning paradigm. Machine learning based systems are adaptive and therefore able to do more than a traditional system will do, as far as detecting and preventing unauthorized attempts to access protected domains. Several machine learning algorithms can be leveraged in the construction of intrusion detection systems. Choosing the right algorithm is not usually a trivial matter. In this study we created a two-phase evaluation procedure for comparing the performance of five machine learning algorithms. We used the NSL-KDD dataset to compare and rank the algorithms on their accuracy of detection. Based on the ranking, we selected the best two and compared their execution times using a 95 percent confident interval estimation. Of the five algorithms evaluated, the Decision Tree algorithm had the best performance in terms of accuracy and execution time. The other algorithms include Support Vector Machine, Logistic Regression, K-Nearest Neighbor, and Naïve Bayes. Our results validate previously produced work on the application of machine learning in intrusion detection.**

***Keywords*—intrusion detection system, intrusion prevention system, machine learning.**

## I. Introduction

Intrusion Detection Systems (IDSs) are devices or applications that monitor traffic on a computer network to detect malicious activities and security policy violations. They analyze network traffic to identify suspicious patterns with the intention to compromise the system [1]. A typical IDS will only detect malicious activities and communicate its suspicions to a human for analysis and possible action. On the other hand, an Intrusion Prevention System (IPS) will detect and attempt to prevent malicious activities from taking place. Over the years, substantial progress has been made in the design of IDS and IPS systems. One major achievement is the application of Machine Learning (ML) algorithms in the design of such systems. ML algorithms have enhanced the capability of IDS/IPS systems [2]. Modern IDSs are able to learn network traffic patterns and make intelligent decisions without human intervention. This has increased the robustness and reliability of such systems.

Classic intrusion detection literature describes 3 categories of IDSs. Ashoor and Gore [3] describe the categories of IDSs as Signature-based, Anomaly-based, and Specification-based. ML based IDSs are considered to belong to the Anomaly-based IDS family [4]. This is due to their ability to differentiate between normal and abnormal traffic patterns. However, they differ from other non-ML based IDSs within the Anomaly-based IDS family. Non-ML based IDSs use static databases to detect attack signatures and malicious activities. They are unable to deal with the dynamic and complex nature of current cyber intrusions and are therefore unable to detect previously unknown attacks. Hence the need for efficient adaptive methods and techniques which ML based IDSs provide. Apart from being able to deal with all the complexities of modern cyber-attacks, the use of ML algorithms in the design of IDSs can also result in higher detection rates, lower false alarm rates, and reasonable computation and communication costs [5].

The performance of an ML based IDS or IPS is largely dependent on the underlying algorithm. Such systems are built using classification algorithms. The algorithm is trained on a labelled dataset containing normal as well as attack data. When fully trained, an ML algorithm is able to differentiate between normal and abnormal or attack traffic to a certain degree of accuracy. Different design decisions and trade-offs are made when deciding on an algorithm for building an IDS. In this study, we evaluate common classification algorithms used for the design of IDSs. We used the NSL-KDD dataset for evaluating each algorithm. The NSL-KDD dataset is a benchmark dataset that is commonly used in the evaluation of ML based IDSs. It is an improvement over its predecessor, the KDD Cup 1999 intrusion dataset [2].

## II. Problem Description

The problem is to determine the most efficient ML classification algorithm to use in the design of an IDS. There are several machine learning classification algorithms, but not all algorithms can be applied to a particular problem. Technically

speaking, we may apply any classification algorithm to build an IDS, but we are unlikely to obtain the desired optimal performance, if the chosen algorithm is not the right one for the job.

The goal, therefore, is to quantitatively evaluate a group of classification algorithms, to determine the most efficient algorithm for designing an IDS. The algorithms to be considered are:

- Logistic Regression
- Naïve Bayes
- Decision Tree
- K-Nearest Neighbor, and
- Support Vector Machine

The metrics of choice are accuracy and speed (more on this later).

## III. Methodology

### A. Dataset: NSL-KDD

Before describing the procedure, we describe the dataset used in evaluating the algorithms. As previously mentioned, the NSL-KDD dataset is a benchmark dataset for intrusion detection research. It was first proposed by Tavallaee et al. [4], after they carried out a complete statistical analysis and revamp of the KDD CUP 99 dataset. Their analysis led to two main improvements:

(a) The removal of an unusually high number of redundant data, which caused learning algorithms to become biased towards more frequently encountered data, and

(b) The validation of the output labels by analyzing the difficulty level of the records in the dataset.

The revamp has led to a much better performance of learning algorithms as compared to the previous state of the dataset. However, NSL-KDD is still not a perfect representation of real-world intrusion dataset. According to Tavallaee et al. [4], the NSL-KDD dataset is still plagued with several problems described by McHugh [6]. The absence of actual intrusion datasets in public domain means the NSL-KDD dataset is the 'gold standard' for intrusion detection research.

The version of the dataset used in this project was published by Botes et al. [1]. [1]

### B. Data Preprocessing and Visualization

The NSL-KDD dataset is divided into training set and test set. The training set contains a total of **41** features and **125,973** data points of five traffic types (Normal, DOS, R2L, Probe, and U2R). Figure 1 shows the traffic types. Table 1 also shows a breakdown of the data. For this project, we decided to put all the attack traffic together in order to have only two categories of traffic – attack and normal. Figure 2 and Table 2 show the dataset after combining all attack data.

The test set contains the same number of features **(41)** as the training set, and **22,543** data points of five traffic types. Figure 3 shows the traffic types. Table 3 shows a breakdown of the

[1]Downloaded from github (https://github.com/InitRoot/NSLKDD-Dataset)

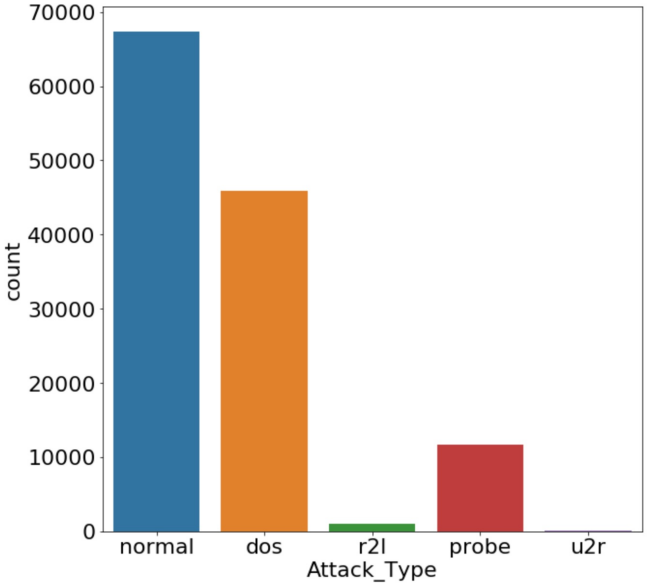


Fig. 1. Training Set.

TABLE I
Breakdown of traffic in the Training Set.

| Attack Type | Count |
|---|---|
| Normal | 67,343 |
| DOS (Denial of Service) | 45,927 |
| R2L (Remote to Local) | 11,656 |
| Probe | 995 |
| U2R (User to Root) | 52 |
| Total | 125,973 |

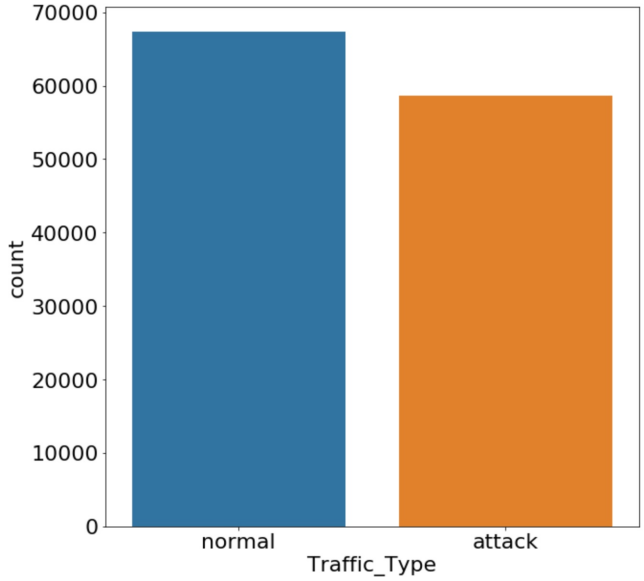


Fig. 2. Traffic Type - Training Set.

TABLE II
BREAKDOWN OF TRAFFIC TYPE - TRAINING SET.

| Traffic Type | Count |
|---|---|
| Normal | 67,343 |
| Attack | 58,630 |
| Total | 125,973 |

data. Figure 4 and Table 4 show the dataset after combining all attack data.

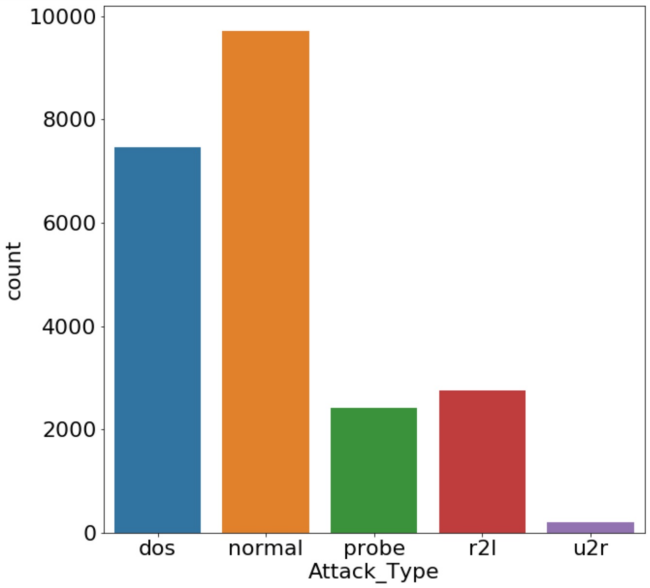


Fig. 3. Test Set.

TABLE III
BREAKDOWN OF TRAFFIC IN THE TEST SET.

| Attack Type | Count |
|---|---|
| Normal | 9,710 |
| DOS (Denial of Service) | 7,456 |
| R2L (Remote to Local) | 2,754 |
| Probe | 2,421 |
| U2R (User to Root) | 202 |
| Total | 22,543 |

TABLE IV
BREAKDOWN OF TRAFFIC TYPE - TEST SET.

| Traffic Type | Count |
|---|---|
| Normal | 9,710 |
| Attack | 12,833 |
| Total | 22,543 |

The features of the NSL-KDD dataset as discussed in iqbal et al. [7] are shown in Appendix A.

### C. Evaluation

The evaluation of the algorithms was carried out in two phases. Phase one measured the accuracy of each algorithm

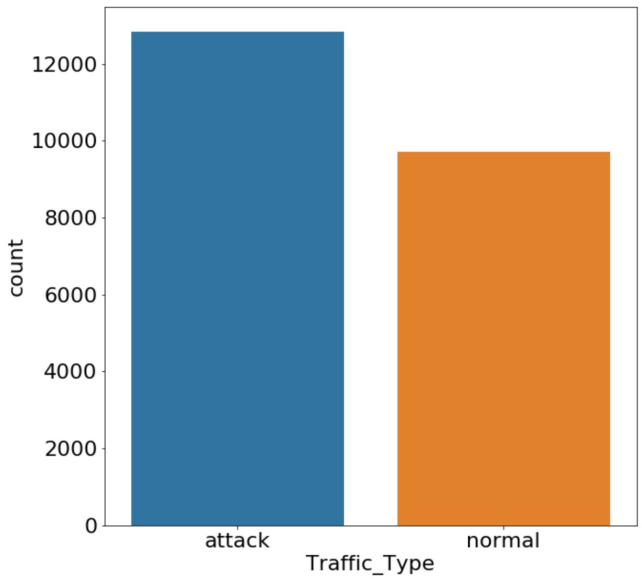


Fig. 4. Traffic Type - Test Set.

on predicting the test set of the NSL-KDD dataset. Phase two measured the execution time of a selected group of 2 algorithms on 50 samples of the NSL-KDD test set.

In phase one, we used the training set to train all five algorithms. Once trained, we used the test set to determine the prediction accuracy for each algorithm. We produced a ranking system containing all five algorithms from the highest to the lowest. Only the top two algorithms – determined by how high they predicted on the test set – made it to phase two.

In phase two, we used confidence interval estimations to perform a system comparison on the two algorithms. The objective was to determine how fast they performed in predicting the class of the datapoints in the test set. We started by training both algorithms with the training set. Then we randomly selected 50 samples out of the 22,543 data points in the test set. Each sample contained 4,000 data points. Using the same sample on both algorithms, we measured the time it took each algorithm to completely predict the class of each data point in the sample. This process produced two sets of execution times. We used the execution times to compare the performance of both algorithms by computing a 95 percent confidence interval estimation. The results are shown in the next section.

## IV. ANALYSIS OF RESULTS

### A. Phase One

The ML algorithms were implemented using python programming language, and the built-in Scikit-learn (SKLearn) library containing various functionalities. After training the algorithm with the training set and running it on the test set to generate predictions, we generated a classification report for each algorithm. (Appendix B)

A summary of the performance of each algorithm on the test set is shown in Table 5.

TABLE V
PREDICTION ACCURACY ON THE TEST SET.

| Algorithm | Accuracy |
|---|---|
| Decision Tree | 80% |
| Support Vector Machine | 80% |
| K-Nearest Neighbor | 79% |
| Naive Bayes | 77% |
| Logistic Regression | 75% |

The results show that Decision Tree (DT) and Support Vector Machine (SVM) performed better than the other three algorithms. This result means that both algorithms made correct predictions on the test set 80 percent of the time. With this performance, the DT and SVM algorithms are the candidates selected for phase two of the evaluation.

### *B. Phase Two*

We used a sample size of 4,000 to slow down the execution time of the DT algorithm. When a smaller sample size was used, we could not measure the execution time. The DT algorithm was executing the task in less than a unit of time. With sample sizes above 4000 data points, the execution time was large enough to be recorded. The execution times for both algorithms for all 50 samples are shown in Appendix C. Special care was taken to ensure that both algorithms were being evaluated using the same sample each time. This is to avoid testing them using unequal scales, which will result in incorrect comparison. A visual inspection of the execution times (Appendix C) show that the DT algorithm performs better, it completes execution on each of the 50 samples in less than a second. The SVM algorithm on the other hand takes at least a second to make predictions on each sample.

To support our observation, we used confidence interval estimations to compare the execution time of both algorithms, to determine the better algorithm of the two. The formula for computing confidence intervals for large samples (samples $>$ 30) is given below. The definitions of the quantities are also provided.

$$(\bar{X} - Z_{1-\alpha/2} \times \frac{S}{\sqrt{n}}, \bar{X} + Z_{1-\alpha/2} \times \frac{S}{\sqrt{n}})$$

- $\bar{X}$: sample mean
- S: sample standard deviation
- n: sample size
- $Z_{1-\alpha/2}$: $(1 - \alpha/2)$ - quantile of a unit normal variate (N(0,1).

Table 6 shows the confidence interval computations.

**Note:** DT – SVM is defined as execution time of the DT algorithm minus execution time of the SVM algorithm. It is calculated for each sample as shown on the fourth column of the table in Appendix C.

TABLE VI
95 PERCENT CONFIDENCE INTERVAL COMPUTATION.

| **Quantity** | **Value** | **Additional Comments** |
|---|---|---|
| x-bar (DT - SVM) | -1.1908781 | Mean of (DT - SVM) |
| s(DT - SVM) | 0.02275773 | Standard Dev of (DT - SVM) |
| n | 50 | Sample size |
| Alpha | 0.05 | Significance level |
| 1 - Alpha/2 | 0.975 | Confidence coefficient |
| Z1-Alpha/2 | 1.95996398 | Inverse of the CDF |
| 1/2CI | 0.006308 | Half confidence interval |
| **C1** | **-1.1971861** | Lower bound |
| **C2** | **-1.1845701** | Upper Bound |

C1 and C2 are the respective lower and upper bounds of the interval at 95 percent confidence level. Since zero (0) does not fall within the interval (-1.1971861 and -1.1845701), we can infer that both algorithms are not the same. Since both intervals are negative, the SVM algorithm has higher values than the DT algorithm. However, the comparison is based on time, and with time we always look for small values. We can therefore say that at 95 percent confidence level, the SVM algorithm takes longer to process the same sequence of samples as the DT algorithm. DT is a more efficient algorithm for building an IDS, because it is faster than SVM.

## V. CONCLUSION

In this study, we implemented a quantitative method to determine the most efficient ML classification algorithm for the design of an IDS. The algorithms considered are Decision Tree, Support Vector Machine, K-Nearest Neighbor, Naïve Bayes, and Logistic Regression. Our evaluation was conducted in two phases. In the first phase, we trained the algorithms using the NSL-KDD training dataset, consisting of 125,973 data points and 41 features. After training, we ranked the algorithms by applying them to predict class labels on the NSL-KDD test set, consisting of 22,543 data points and 41 features. We obtained a classification report for each algorithm and used accuracy as the ranking metric. Decision Tree and Support Vector Machine came out joint tops in the ranking table having scored 80 percent each. In the second phase, we created 50 samples from the NSL-KDD test set, each containing 4,000 data points. After training the algorithm using the training set as before, we ran both algorithms over each of the 50 samples to predict their class labels. We measured the execution time for both algorithms on each sample. Finally, we computed 95 percent confidence interval estimation to compare both algorithms. We can conclusively state that the most efficient of the five tested algorithms is the Decision Tree algorithm, and this is consistent with previously established facts in IDS literature [1] about the efficiency of the Decision Tree algorithm in Intrusion Detection Systems.

## REFERENCES


[1] F. Botes, L. Leenen, and R. De La Harpe, "Ant colony induced decision trees for intrusion detection," in *16th European Conference on Cyber Warfare and Security*. ACPI, 2017, pp. 53–62.

[2] Z. Lin, Y. Shi, and Z. Xue, "Idsgan: Generative adversarial networks for attack generation against intrusion detection," *arXiv preprint arXiv:1809.02077*, 2018.

[3] A. S. Ashoor and S. Gore, "Importance of intrusion detection system (ids)," *International Journal of Scientific and Engineering Research*, vol. 2, no. 1, pp. 1–4, 2011.

[4] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE symposium on computational intelligence for security and defense applications*. IEEE, 2009, pp. 1–6.

[5] M. Zamani and M. Movahedi, "Machine learning techniques for intrusion detection," *arXiv preprint arXiv:1312.2177*, 2013.

[6] J. McHugh, "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 (TISSEC)*, vol. 3, no. 4, pp. 262–294, 2000.

[7] A. Iqbal, S. Aftab, I. Ullah, M. A. Saeed, and A. Husen, "A classification framework to detect dos attacks." *International Journal of Computer Network & Information Security*, vol. 11, no. 9, 2019.

# Appendix A

FEATURES OF THE NSL-KDD DATASET

| # | Feature |
|---|---|
| 1 | duration |
| 2 | src_bytes |
| 3 | dst_bytes |
| 4 | Land |
| 5 | wrong_fragment |
| 6 | urgent |
| 7 | hot |
| 8 | num_failed_logins |
| 9 | logged_in |
| 10 | num_compromised |
| 11 | root_shell |
| 12 | su_attempted |
| 13 | num_root |
| 14 | num_file_creations |
| 15 | num_shells |
| 16 | num_access_files |
| 17 | num_outbound_cmds |
| 18 | is_host_login |
| 19 | is_guest_login |
| 20 | count |
| 21 | srv_count |
| 22 | serror_rate |
| 23 | srv_serror_rate |
| 24 | rerror_rate |
| 25 | srv_rerror_rate |
| 26 | same_srv_rate |
| 27 | diff_srv_rate |
| 28 | srv_diff_host_rate |
| 29 | dst_host_count |
| 30 | dst_host_srv_count |
| 31 | dst_host_same_srv_rate |
| 32 | dst_host_diff_srv_rate |
| 33 | dst_host_same_src_port_rate |
| 34 | dst_host_srv_diff_host_rate |
| 35 | dst_host_serror_rate |
| 36 | dst_host_srv_serror_rate |
| 37 | dst_host_rerror_rate |
| 38 | dst_host_srv_rerror_rate |
| 39 | protocol_type |
| 40 | service |
| 41 | flag |

## Appendix B

## CLASSIFICATION REPORTS FOR ALL FIVE ALGORITHMS

### Logistic Regression

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.92 | 0.61 | 0.73 | 12833 |
| 1 | 0.64 | 0.93 | 0.76 | 9710 |
| accuracy | | | 0.75 | 22543 |
| macro avg | 0.78 | 0.77 | 0.75 | 22543 |
| weighted avg | 0.80 | 0.75 | 0.75 | 22543 |

### Naïve Bayes

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.91 | 0.67 | 0.77 | 12833 |
| 1 | 0.68 | 0.92 | 0.78 | 9710 |
| accuracy | | | 0.77 | 22543 |
| macro avg | 0.79 | 0.79 | 0.77 | 22543 |
| weighted avg | 0.81 | 0.77 | 0.77 | 22543 |

### Decision Tree

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.97 | 0.68 | 0.80 | 12833 |
| 1 | 0.69 | 0.97 | 0.81 | 9710 |
| accuracy | | | 0.80 | 22543 |
| macro avg | 0.83 | 0.82 | 0.80 | 22543 |
| weighted avg | 0.85 | 0.80 | 0.80 | 22543 |

## K-Nearest Neighbor

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.97 | 0.64 | 0.77 | 12833 |
| 1 | 0.67 | 0.98 | 0.80 | 9710 |
| accuracy | | | 0.79 | 22543 |
| macro avg | 0.82 | 0.81 | 0.79 | 22543 |
| weighted avg | 0.84 | 0.79 | 0.78 | 22543 |

## SVM

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.98 | 0.66 | 0.79 | 12833 |
| 1 | 0.69 | 0.98 | 0.81 | 9710 |
| accuracy | | | 0.80 | 22543 |
| macro avg | 0.83 | 0.82 | 0.80 | 22543 |
| weighted avg | 0.85 | 0.80 | 0.80 | 22543 |

# Appendix C

## EXECUTION TIME (SECONDS)

| S/N | Decision Tree (A) | Support Vector Machine (B) | A - B |
|---|---|---|---|
| 1 | 0.000999 | 1.196236 | -1.19524 |
| 2 | 0.000998 | 1.218428 | -1.21743 |
| 3 | 0.000997 | 1.190477 | -1.18948 |
| 4 | 0.001 | 1.276542 | -1.27554 |
| 5 | 0.001032 | 1.186174 | -1.18514 |
| 6 | 0.000999 | 1.190952 | -1.18995 |
| 7 | 0.000994 | 1.232021 | -1.23103 |
| 8 | 0.000955 | 1.180771 | -1.17982 |
| 9 | 0.001001 | 1.18945 | -1.18845 |
| 10 | 0.001993 | 1.266891 | -1.2649 |
| 11 | 0.001995 | 1.239583 | -1.23759 |
| 12 | 0.000998 | 1.217741 | -1.21674 |
| 13 | 0.000996 | 1.193527 | -1.19253 |
| 14 | 0.000993 | 1.197766 | -1.19677 |
| 15 | 0.000996 | 1.198343 | -1.19735 |
| 16 | 0.000996 | 1.174868 | -1.17387 |
| 17 | 0.000998 | 1.167588 | -1.16659 |
| 18 | 0.000997 | 1.183265 | -1.18227 |
| 19 | 0.000998 | 1.174174 | -1.17318 |
| 20 | 0.001993 | 1.18479 | -1.1828 |
| 21 | 0.000996 | 1.169434 | -1.16844 |
| 22 | 0.000996 | 1.177195 | -1.1762 |
| 23 | 0.001011 | 1.172727 | -1.17172 |
| 24 | 0.000997 | 1.174639 | -1.17364 |
| 25 | 0.000998 | 1.179182 | -1.17818 |
| 26 | 0.000996 | 1.177313 | -1.17632 |
| 27 | 0.000998 | 1.185032 | -1.18403 |
| 28 | 0.000993 | 1.17402 | -1.17303 |
| 29 | 0.001001 | 1.172887 | -1.17189 |
| 30 | 0.001992 | 1.190658 | -1.18867 |
| 31 | 0.001038 | 1.178587 | -1.17755 |
| 32 | 0.000997 | 1.215846 | -1.21485 |
| 33 | 0.000999 | 1.187218 | -1.18622 |
| 34 | 0.001041 | 1.179813 | -1.17877 |
| 35 | 0.001039 | 1.198215 | -1.19718 |
| 36 | 0.000997 | 1.22086 | -1.21986 |
| 37 | 0.000998 | 1.183075 | -1.18208 |
| 38 | 0.000997 | 1.174665 | -1.17367 |
| 39 | 0.001994 | 1.199568 | -1.19757 |
| 40 | 0.00104 | 1.182919 | -1.18188 |
| 41 | 0.000997 | 1.185067 | -1.18407 |
| 42 | 0.000998 | 1.176059 | -1.17506 |
| 43 | 0.000996 | 1.188516 | -1.18752 |
| 44 | 0.000993 | 1.176179 | -1.17519 |
| 45 | 0.000998 | 1.196498 | -1.1955 |
| 46 | 0.000999 | 1.185117 | -1.18412 |
| 47 | 0.000996 | 1.173745 | -1.17275 |
| 48 | 0.000996 | 1.183049 | -1.18205 |
| 49 | 0.001 | 1.181945 | -1.18095 |
| 50 | 0.002005 | 1.200314 | -1.19831 |