FINAL REPORT:
ANOMALY DETECTION IN NETWORK TRAFFIC

Group 4: Isabelle Viraldo, Trevor Thayer, Anthony Do, Anh Huy Nguyen

---

2. Abstract:

This project we are taking on is to develop and compare the effectiveness of different machine learning models on the KDD Cup 1999 dataset. The models are training to detect anomalies in network traffic, and predict the locations of security threats before an attack can occur. The first step to develop these models is to do preprocessing on the data. This will include primarily handling missing values, and encoding categorically. The models we will train include the Support Vector Machines model, the Decision Trees model, the K-Means Clustering model, and the Isolation Forests Model. This provides an even mix of supervised and unsupervised approaches to predicting these models. Then, to evaluate each of these models effectiveness, we will assess their precision and recall with cross-validation, prioritizing recall.

Recall is prioritized because in the context of intrusion detection, failing to detect an actual attack (a false negative) poses a much greater risk than occasionally flagging normal traffic as suspicious. Finally, these models will be directly compared to one another, weighing their differences in effectiveness and efficiency to determine their effectiveness as a model for detecting abnormal network behavior.

3. Introduction:

3.1. Background information on network intrusion detection, the significance of anomaly detection:

Network intrusion detection (IDS) is a system that plays an essential role in controlling and preventing dangerous, unknown access or intended attacks on network resources. Besides these performance issues, such factors undermine the goals of the security administrator and sometimes are more beneficial to attackers than the users. For instance, the large volumes of normal network traffic, which can be forcibly generated by successive manual tasks, will in most cases be mixed with developing conditions, and the entire traffic structure may change frequently. These actions mislead network attackers and decrease the efficiency of the network security administrator when lightning will be routed to the right zone. They can take advantage of wrongly discovered normal network traffic and make network administrators overlook or ignore anomalies, which can be a real threat to network security. As a result, the security administrator now has to deal with the threats and challenges caused by both the end-users and the conditions existing in the LAN. The default problems are caused by the lack of input/output resources (not enough CPU, network delay, disk error, etc.). Moreover, if the user base or network environment is undergoing radical changes these evaluations do not properly convey the future effectiveness of the system. Despite the fact that users may be smart enough to understand these factors, overall the user perceptions are of paramount importance as the acceptance of these security measures is a key issue.

3.2. Overview of the KDD Cup 1999 dataset:

KDD Cup 1999 dataset is a public dataset constructed from DARPA'98 network traffic and has been widely applied in testing network intrusion detection systems. It consists of approximately 5 million connection records with each record having 41 attributes that represent various characteristics of the communications in the networks, and labeling each record as normal or one from among various categories of attacks—DoS, R2L, U2R, and Probing. The dataset was initially built in order to provide researchers with a standardized setup for evaluating anomaly detection algorithms for use in the domain of network security.

3.3. Clear statement of project objectives:

This project aims to develop and evaluate machine learning models to address security threats and intrusions by intercepting the anomalous network traffic manifested. The dataset KDD Cup 1999 will be my priority and the various issues it might have will be resolved through data preprocessing like dealing with missing values, categorical encoding, featuring scaling and balancing the imbalanced classes. The set of parameters we will further apply to these models to verify their worthiness are based on standard metrics like accuracy, precision, recall, F1-score, and runtime analysis. Furthermore, we will compare the models' effectiveness in detecting anomalies in network traffic to come to a conclusive appreciation.

5. Methodology:

5.1 Detailed description of data preprocessing steps.

First, our team uses the fetch_kddcup99 function from sklearn.datasets to load a 10% version of the KDD Cup 1999 dataset, then concatenates the feature columns and labels (X, y) into a DataFrame and saves it to a CSV file. Next, the handle_missing_values function reads the file again, identifies missing values (NaN) in each numeric and categorical column, and performs imputation—using the mean for the numeric column and the mode for the categorical column—to ensure there are no missing data before the next step is performed. When the data is cleaned, the categorical encoding step (encode_categorical_features) converts the three columns protocol_type, service, and flag into one-hot binary columns to turn each categorical value into a numeric vector.

Next, the scale_numerical_features function applies StandardScaler or MinMaxScaler to the remaining numeric columns to standardize the distribution (Z-score or Min-Max) to ensure all features have the same scale. Finally, the data is split into training and testing sets (train_test_split), and the handle_imbalanced_data function uses SMOTE (or other over/under-sampling methods) to oversample the minority class, balancing the class ratio before saving the resampled training and testing sets to new CSV files. A total of 6 new CSV files will be created after the program completed running preprocessing functions. One file will be raw KDD 99 data, and a function Handling Missing Values, Categorical Encoding, Feature Scaling will form a file in turn. Finally the Handling Imbalanced Data will produce the two final files that are used for stage 2 and 3 are kddcup99_10_percent_train_resampled and kddcup99_10_percent_test.

5.2 Explanation of the selected models and justification for their choice.

<u>5.2.1 Random Forests (Supervised):</u>

Plan: The Random Forest model will be implemented for supervised learning in this project. The steps involved in this process are: the model setup, the training, and the hyperparameter tuning.

We will initialize the Random Forest model, choosing parameters such as the number of trees (n_estimators) and the maximum depth of each tree (max_depth). We will tune these parameters to optimize performance based on cross-validation results. The model will be trained on the preprocessed dataset, with the features and labels split into training and testing sets. We will use the training data to build the model and assess its performance using the test data.

Random Forests have several hyperparameters that we can tune for better performance, including the number of estimators (trees), the maximum depth of each tree, and the minimum number of samples required to split a node. We will perform grid search or random search cross-validation to find the best combination of hyperparameters.

Justification: Random Forest is an ensemble learning method that aggregates the results of multiple decision trees to create a more accurate and robust model. This approach helps mitigate overfitting,

making it particularly well-suited for the KDD Cup 1999 dataset, which is large and complex. The ability of Random Forest to generalize well is essential for accurately identifying both known and unknown network intrusions.

The KDD Cup 1999 dataset contains a wide range of features, both categorical and continuous. Random Forest can effectively handle this high-dimensional data by selecting the most relevant features during training and managing missing values automatically. Moreover, Random Forests can provide insights into feature importance, revealing which variables are most influential in detecting network anomalies. This capability is crucial for optimizing the intrusion detection process, guiding the selection of features that most effectively identify potential security threats.

5.2.2 Support Vector Machine (Supervised):

With the KDD Cup 1999 Dataset, it contains a large amount of features-around about 41 features. This is one of the reasons why we chose to include an SVM model in our proposal. An SVM model is able to perform well in high-dimensional spaces. This is due to how with an SVM model; the model focuses on finding an optimal decision boundary rather than modeling the entire data distribution.

Another reason is that network intrusion patterns can at times be complex and non-linear. As such, with an SVM model, we will be able to address these issues of non-linearity. In order for an SVM model to accomplish this, we will need to employ the use of a kernel function. This kernel function will enable us to map our data into a higher-dimensional space so that we may be able to identify a decision boundary which was not possible before employing the kernel function. This effectively helps us to capture these non-linear relationships.

Lastly, a SVM model is the robust to outliers that an SVM model has. I have yet to fully examine our dataset, but it is possible that in our dataset, there may be a few outliers or nosy data points. As a result, these outliers/noise may skew or throw our models off. However, with an SVM model, thanks to its use of support vectors to determine the position of the decision boundary; it makes the SVM model more robust to outlier data points. This is because the support vectors of an SVM model help limit the effect/influence that the outliers may have/exert on determining the position of the decision boundary.

5.2.3 K-Means Clustering (Unsupervised):

K-Means clustering is an unsupervised learning algorithm, which categorizes data based on how similar points tend to cluster together, allowing for the categorization of data to emerge naturally from the behavior of the dataset. Unlike supervised models, this model groups data points by minimizing the variance within clusters. The process of training a model consists of choosing the number of clusters and initializing the centroids, assigning the data points based on euclidean distance to the nearest centroid. Then, after the initialization, the centroids and categorizations are iteratively recalculated until the values stabilize, and the centroids no longer shift a significant distance. Important values like the number of clusters, the euclidean distance parameters, and the initial centroids will all be tuned to develop the most appropriate model for our use case.

This model is useful for categorization because it identifies regular network patterns and behavior while flagging potential abnormalities. By using clusters of common data, the model can distinguish between normal traffic and outliers, which may be more likely to be the source of network attacks. These outliers should be defined by setting a threshold on the distance from cluster centroids, and any point that is beyond that threshold would be marked as abnormal network behavior. This method is likely to prove effective because then it doesn't rely on network attacks all behaving the same way, rather, can define network attacks based on their distance outside the normal behavior defined by the clusters. The model is also computationally efficient, so even with the large KDD Cup 1999 dataset we are using, it will be able

to process all of the data in a comparatively short amount of time. These things all contribute to why this model is a great candidate for being an effective model for detecting potential network attacks.

5.2.4 Isolation Forest (Unsupervised):

Isolation Forest belongs to the Unsupervised Learning group, so it does not need to be trained with the label "normal" or "abnormal" first. This is very important when the label is difficult to collect or the anomaly class is too rare compared to the normal class. In addition, unlike density or distance-based algorithms, Isolation Forest isolates anomalies by randomly splitting the feature, causing the outliers to be separated closer to the base of the tree, helping to detect quickly and effectively. Thanks to the mechanism of building an isolation forest with a short average depth for anomalies, the algorithm has a computational complexity of O(n log n) where n is the number of samples, while the memory only increases linearly with the number of trees and sample size. As a result, it is very suitable for large datasets such as 10% KDD99. Through research, I believe that Isolation Forest can be easily applied to both numerical and categorical features, without the need for complex preprocessing such as determining metrics for categorical variables.

5.3 Outline of the training process, including hyperparameter tuning

5.3.1 (Supervised): Random Forest (Trevor)

To develop a supervised learning model for anomaly detection, the Random Forest Classifier was selected for its high performance, scalability, and ability to handle high-dimensional data. Its built-in support for feature importance made it especially useful for identifying which network traffic attributes were most predictive of intrusions. The training process began with a lightweight baseline model using 30 estimators to compute feature importances across the original 118 features. Based on these importances, the top 30 features were selected to reduce dimensionality and significantly improve training efficiency. This step was crucial, as training on the full feature set repeatedly led to memory issues, crashes, and prohibitively long runtimes.

To tune the model, a 1% random subsample of the training data (approximately 50,000 rows) was used to enable efficient cross-validation without exhausting system resources. Hyperparameter tuning was performed using RandomizedSearchCV with 3-fold cross-validation across 20 randomized combinations. The parameter grid included the number of estimators, tree depth, feature sampling strategies, minimum samples required to split nodes or form leaf nodes, and whether to use bootstrapping. The best-performing configuration consisted of 300 estimators, no restriction on depth, log2 feature sampling, a minimum of 5 samples per split, 1 per leaf, and bootstrapping disabled. This optimized model was then retrained on the full, preprocessed training set using the top 30 features and all available CPU cores (n_job s= -1), yielding a robust classifier well-suited for detecting both common and rare network attacks.

5.3.2 Support Vector Machine (Supervised):

For this model, the model was trained on 10% of the full KDD 1999 dataset. With this 10% of the KDD, we first processed it which included handling missing values, encoding categorical features, scaling numerical features, and handling class imbalance. After which, we split the data into a 80-20 ratio. 80% of the 10% KDD was used to train the SVM model, while the last 20% was used as testing data. For this particular SVM Model, the model utilized all 40 features of the KDD dataset during the training process.

The original plan to tune the hyperparameters was to utilize cross-validation. We had intended to use the GridSearchCV class to achieve this. We wanted to set the param_grid with the following dictionary:

```
param_grid = {
        'C' : [0.1, 1, 10]
        'gamma' : ['scale', 'auto', 0.1, 1]
```

‘Kernel’ : [‘rbf”]

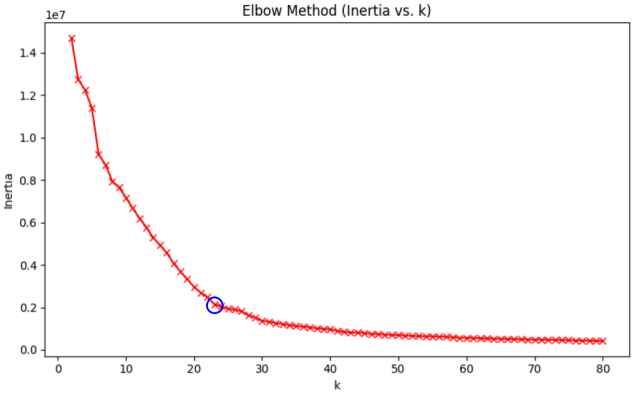
}

Furthermore, we wanted to use a cv of 5 for the cv parameter of the GridSearchCV class. However, due to how computationally taxing this would be on the computer as I have discovered. My macbook currently running this code has been running for more than 150 hours and is still running now with no idea when the program will finish. As such, we have chosen to opt out of using this approach for the SVM model, because despite not utilizing cross validation, our metrics and results have been satisfactory for us.

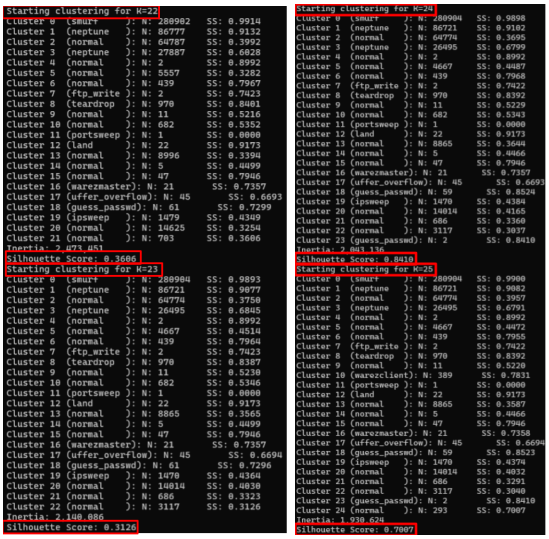
5.3.3 K-Means Clustering (Unsupervised):

The K-Means Clustering algorithm made use of scikit-learn’s KMeans algorithm. The data as prepared earlier, kddcup99_10_percent_cleaned to train the model. The program saves the x values, and only uses the y values to determine the evaluation metrics after the training is complete. This can be run immediately first when running the program, however, this requires defining a k value for the number of clusters the model will run with. In order to clearly determine the number of clusters to use, a separate function of the code was written to cycle through a range of k values, calculating silhouette scores and inertia scores along the way. As the programmer, we have knowledge of the data behind the scenes, the number of values given to the model to train on is a total of 23, so we should be able to determine that value using measurements outside of our knowledge.

First, we selected a range of k values to train the model with, a range of 2 to 80 was chosen as the model was swiftly training and evaluating these models, while showing a dramatic improvement, tapering off toward the latter end of the range. Then, when evaluating this range, the inertia and silhouette scores were displayed and saved for each model (this information can be found in the kmeans_log.csv file after running)



The inertia, when plotted against the k value, can be used to find the optimal k using the elbow method. And as the figure on the right shows (this figure is generated after running the program) the clear elbow point, where inertia slows down dramatically, occurs when k equals 23, just as we predicted it would. Considering the Silhouette scores can help us refine our score, looking closer, at the individual models k equaling 22, 23, 24, and 25, we see that the silhouette scores are 0.36, 0.31, 0.84, and 0.70 respectively. This shows us that the nearest point to the elbow, and would be an effective k value for the model is 24.



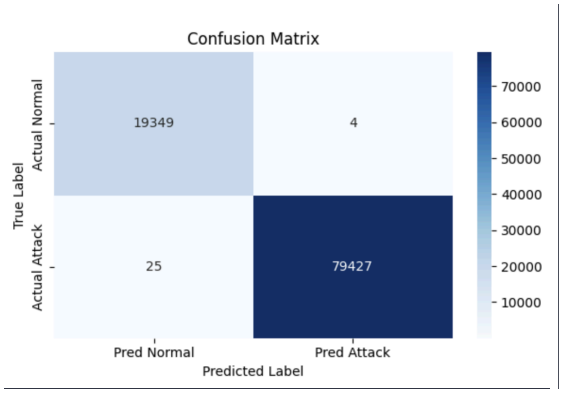
5.3.4 Isolation Forest (Unsupervised):

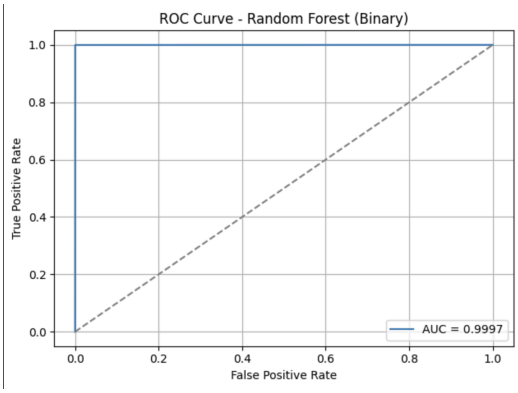
The Isolation Forest model will be trained using the kddcup99_10_percent_train_resampled data file, which is the largest data file created from stage 1. The first training step will load and clean the data from the preprocessed CSV file mentioned above. After reading the DataFrame using pd.read_csv, the labels are normalized by decoding from bytes to strings and removing the dots at the end of the string. Next, the feature set *X* is taken from all the numeric columns (of type float32), while the binary label y is built by assigning the value 1 to the non-"normal" records and 0 to the normal records. Finally, the class distribution is printed for inspection before the tuning step is performed.

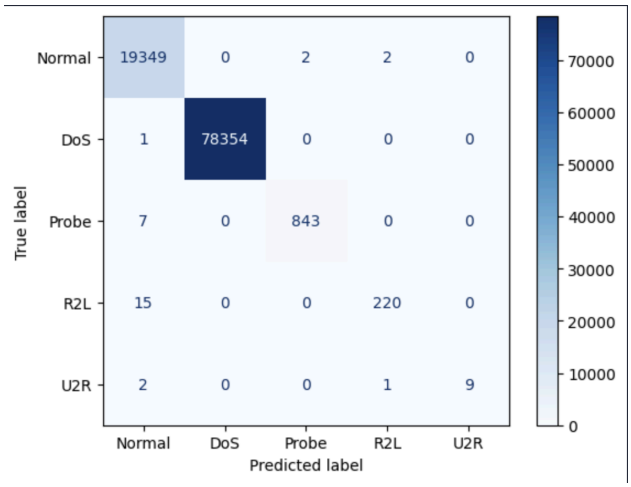
The next step is to optimize the hyperparameter tuning for the Isolation Forest using RandomizedSearchCV. The search space is defined by the number of trees (n_estimators), the number of samples used per tree (max_samples), the ratio of randomly selected features (max_features), the expected ratio of outliers (contamination) and the bootstrap flag. The original IsolationForest model (random_state=42) is run through 20 random trials, evaluated by the accuracy score on 5-fold StratifiedKFold to ensure that each fold maintains the same class ratio. The best results will be saved and serialized into a .pkl file, ready for the next evaluation and deployment steps. Although only using 10% of the data set and going through a thorough preprocessing process, during the program run for the model, the program reported overloading many times due to the amount of folds, candidates and total fits. Therefore, the number of these three parameters had to be edited and balanced many times to choose the appropriate hyperparameter search.

6. Results - Presentation of evaluation metrics for each model and Visualizations (e.g., confusion matrices, ROC curves) to illustrate performance:

6.1 Random Forests (Supervised):







	precision	recall	f1-score	support
back.	1.00	1.00	1.00	435
buffer_overflow.	0.88	0.78	0.82	9
ftp_write.	1.00	1.00	1.00	1
guess_passwd.	1.00	0.88	0.93	8
imap.	1.00	0.67	0.80	3
ipsweep.	0.99	0.99	0.99	265
land.	1.00	1.00	1.00	1
loadmodule.	0.00	0.00	0.00	2
multihop.	0.50	1.00	0.67	1
neptune.	1.00	1.00	1.00	21294
nmap.	1.00	0.98	0.99	45
normal.	1.00	1.00	1.00	19353
perl.	1.00	1.00	1.00	1
pod.	1.00	1.00	1.00	38
portsweep.	1.00	1.00	1.00	236
satan.	1.00	0.99	0.99	304
smurf.	1.00	1.00	1.00	56402
teardrop.	1.00	1.00	1.00	185
warezclient.	0.99	0.94	0.96	218
warezmaster.	1.00	1.00	1.00	4
accuracy			1.00	98805
macro avg	0.92	0.91	0.91	98805
weighted avg	1.00	1.00	1.00	98805

6.2 Support Vector Machine(Supervised):

	precision	recall	f1-score	support
b'back.'	0.95	1.00	0.97	435
b'buffer_overflow.'	0.14	0.89	0.25	9
b'ftp_write.'	0.03	1.00	0.07	1
b'guess_passwd.'	1.00	0.88	0.93	8
b'imap.'	1.00	1.00	1.00	3
b'ipsweep.'	0.95	0.98	0.97	265
b'land.'	1.00	1.00	1.00	1
b'loadmodule.'	0.00	0.00	0.00	2
b'multihop.'	0.00	0.00	0.00	1
b'neptune.'	1.00	1.00	1.00	21294
b'nmap.'	0.45	0.96	0.61	45
b'normal.'	1.00	0.97	0.98	19353
b'perl.'	1.00	1.00	1.00	1
b'pod.'	1.00	1.00	1.00	38
b'portsweep.'	0.92	1.00	0.96	236
b'rootkit.'	0.00	0.00	0.00	0
b'satan.'	0.95	0.97	0.96	304
b'smurf.'	1.00	1.00	1.00	56402
b'spy.'	0.00	0.00	0.00	0
b'teardrop.'	1.00	1.00	1.00	185
b'warezclient.'	0.29	0.88	0.44	218
b'warezmaster.'	0.36	1.00	0.53	4
accuracy			0.99	98805
macro avg	0.64	0.80	0.67	98805
weighted avg	1.00	0.99	0.99	98805

```
Accuracy Score:
0.9927331612772633
(base) anthonydo@Anthonys-MacBook-Pro Final Project %
```

6.3 K-Means Clustering(Unsupervised):

```
Overall Accuracy: 0.8795

Classification Report:
```

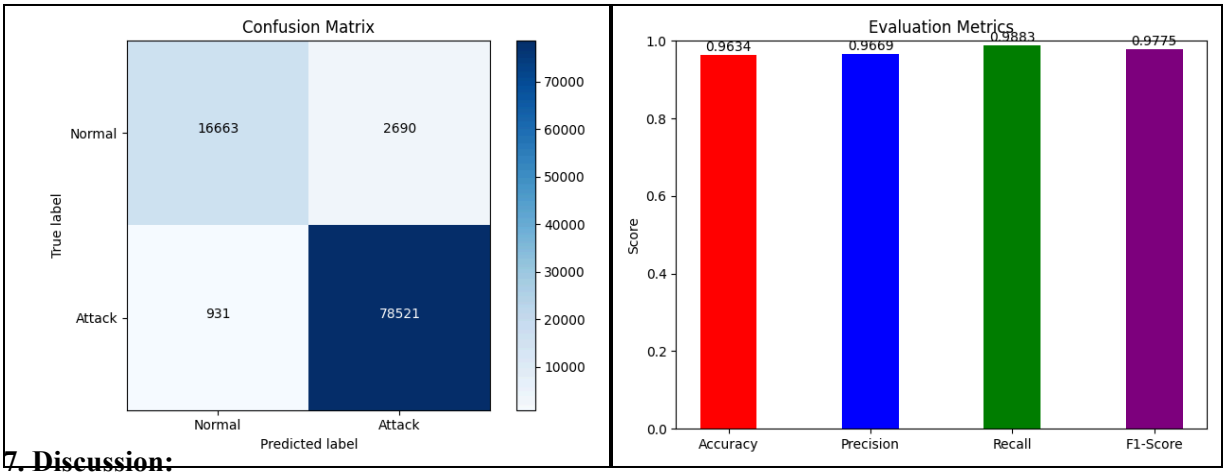
Label	Precision	Recall	F1-Score	Support
ack	0.02	0.09	0.04	2203
ftp_write	0.00	0.25	0.01	8
normal	0.96	0.64	0.76	97278
perl	0.00	0.00	0.00	3
phf	0.00	0.00	0.00	4
pod	0.00	0.23	0.01	264
portsweep	0.03	0.70	0.05	1040
rootkit	0.00	0.00	0.00	10
satan	0.46	0.89	0.60	1589
smurf	1.00	1.00	1.00	280790
spy	0.00	0.00	0.00	2
teardrop	1.00	0.99	1.00	979
guess_passwd	0.86	0.96	0.91	53
uffer_overflow	0.40	0.60	0.48	30
warezclient	0.45	0.30	0.36	1020
warezmaster	0.71	0.75	0.73	20
23	0.00	0.00	0.00	0
imap	0.00	0.00	0.00	12
ipsweep	0.77	0.91	0.84	1247
land	0.95	1.00	0.98	21
loadmodule	0.04	0.22	0.07	9
multihop	0.00	0.00	0.00	7
neptune	1.00	0.81	0.89	107201
nmap	0.02	0.45	0.04	231
Macro Avg	0.36	0.45	0.37	494021
Weighted Avg	0.98	0.88	0.92	494021

6.4 Isolation Forest (Unsupervised):

```
================================================
Isolation Forest Evaluation
================================================
Accuracy   : 0.9634
Precision  : 0.9669
Recall     : 0.9883
F1-Score   : 0.9775
Model load : 0.11s
Inference  : 0.82s

Class-wise metrics:
Class        Prec     Rec      F1
Normal     0.9471  0.8610  0.9020
Attack     0.9669  0.9883  0.9775

Confusion Matrix (rows=true, cols=pred):
         Normal  Attack
Normal:   16663    2690
Attack:     931   78521
================================================
```



7. Discussion:

7.1 Interpretation of results, highlighting key findings.

7.1.1 Random Forests (Supervised):

The Random Forest model produced highly accurate and consistent results on the KDD Cup 1999 test data. It achieved 99.97% accuracy, 99.99% precision, 99.97% recall, and 99.98% F1-score when evaluated in binary form (normal vs. attack). The confusion matrix confirms the model's reliability, with minimal false positives and false negatives, and the ROC curve yielded an AUC of 0.9997, indicating almost perfect discrimination between classes.

When evaluated in multiclass form across all 21 labeled categories, the model maintained almost perfect precision and recall on high volume classes such as smurf, neptune, and normal. These results demonstrate the model's ability to consistently identify frequent patterns in network traffic and distinguish them from malicious behavior. The grouped classification report also showed that the model performed very well on the broader categories of DoS and Probe attacks.

Overall, the Random Forest classifier proved to be a highly effective supervised learning approach for detecting anomalies in network traffic. Its strong performance across most classes, especially in high frequency attack types, validates its suitability for real-time intrusion detection scenarios.

7.1.2 Support Vector Machine (Supervised):

The Support Vector Machine(SVM) showed excellent results overall. The accuracy score achieved was a high 99.27%. If we look at the average for precision, recall, and f1-score; we can see that for the macro

average of these metrics it goes: precision - 64%, recall - 80%, and f1-score - 67%. For the weighted average, we can see that the precision is 100%, recall is 99%, and f1-score is 99%. Now, if we look at the classes with support in the triple and five digits, we notice a trend that most of these classes have high percentages for their metrics. Then, if we look at the classes with support in the single or double digits, we notice that there are more classes with low percentages for their metrics. This highlights the significant impact that low support or occurrences of classes has on a SVM's performance. As such, we would need to add more support to those classes so that we can balance out the data, and in return greatly improve our SVM model's capabilities and performance.

7.1.3 K-Means Clustering (Unsupervised):

The results of the KMeans clustering (24 clusters) shows decent overall accuracy at 87.95%, but the metrics reveal significant imbalance among the clusters.The dominant classes like Smurf, Neptune, and Teardrop have near-perfect precision and recall due to their size. Whereas rare attack types such as perl, spy, and rootkit are completely missed, with zero precision and recall. The macro average F1-score is only 0.37, while the weighted average is 0.92, highlighting how skewed the model's success is toward the majority classes. Even though the silhouette score is high at 0.84, it doesn't translate to strong classification performance across all classes.

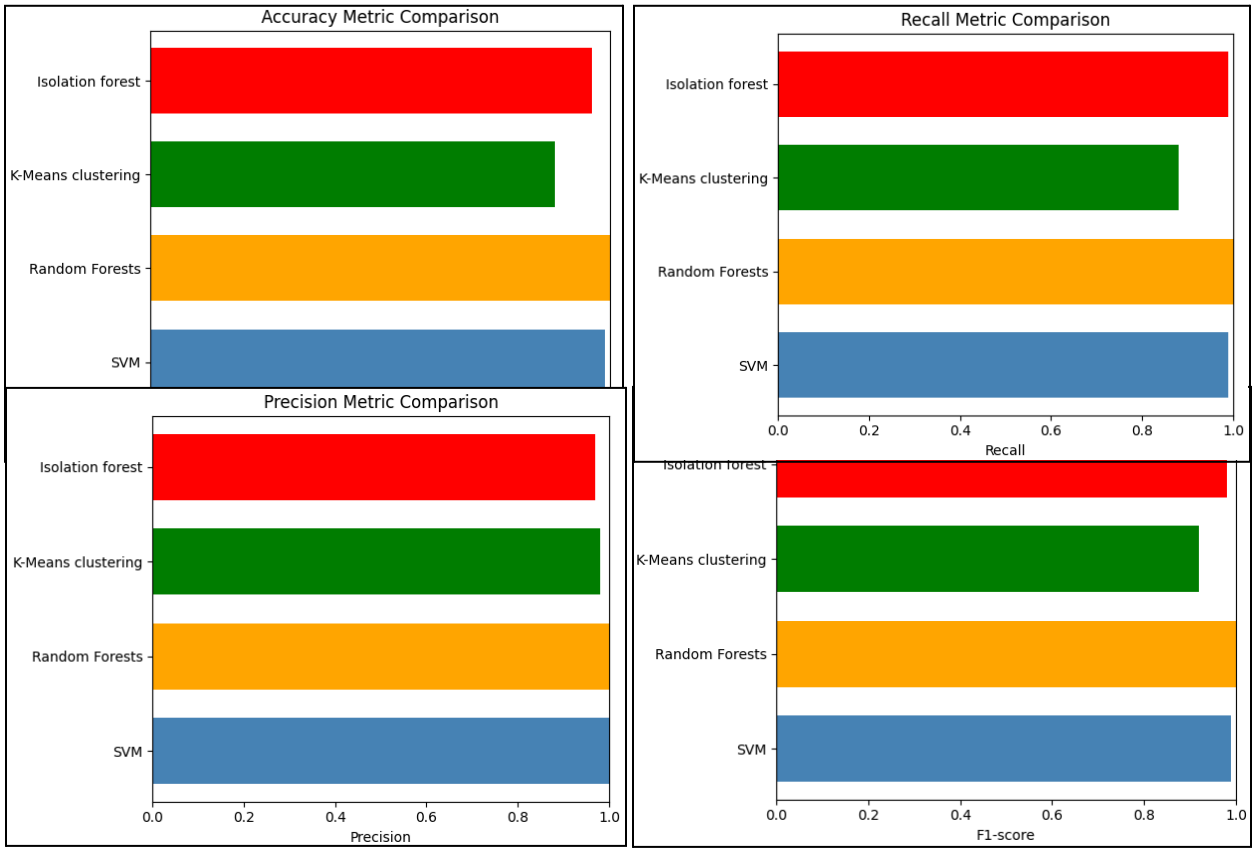
A big issue is that this model struggles with the imbalanced nature of the dataset, often clustering rare classes into the same group as more dominant ones. The spherical cluster assumption also doesn't hold well for the structure of this data. Since the model was only trained on 10% of the dataset, it likely didn't have enough examples of the underrepresented attack types to form meaningful clusters. On top of that, the process of mapping clusters to true labels through majority voting introduces even more error, especially for classes that are small or scattered.

7.1.4 Isolation Forest (Unsupervised):

The stage 3 results show that the accuracy is 96.34%, precision 96.69%, recall 98.83% and F1-score 97.75%. Although the accuracy is high, due to the severe imbalance in the data, focusing on precision, recall and F1-score will more accurately reflect the model's performance in detecting rare attacks (source: data-mingle.ai). Next is the Precision index (96.69%) which shows that among the samples predicted as "attack", 96.69% are actually attacks, indicating a low false positive rate. My Recall result hit 98.83% indicates that the model detected 98.83% of the total actual attacks, showing the ability to miss few attacks. Finally, my F1-score got 97.75% is the average of precision and recall, comprehensively evaluating the model's performance, especially important when the data is skewed (source: Analytics Vidhya). Regarding to Confusion Matrix result, this analysis helps to balance between avoiding missed attacks (minimize FN) and reducing false alarms (minimize FP) (source: ProjectAI). In addition, scikit-learn recommends using both this matrix and precision/recall metrics for anomaly detection (source: Scikit-learn)

The main findings show that the Isolation Forest model performs relatively well on the 10% KDD Cup 1999 dataset: the recall is close to 99%, which helps to almost not miss any attacks, while the precision is approximately 97%, ensuring an acceptable false alarm rate. The F1-score of 97.75% confirms that the model has been well optimized between precision and recall, which is very suitable for anomaly detection. In addition, the random isolation forest construction mechanism provides O(n log n) complexity and linear memory, showing that Isolation Forest has good scalability on large datasets such as 10% KDD99. In summary, the evaluation results show that the Isolation Forest model, with reasonable threshold tuning and hyperparameters, has achieved excellent performance in anomaly detection on KDD 99, while ensuring the ability to deploy in practice with high accuracy and fast processing speed.

7.2 Comparative analysis of model performances.



In comparing the four approaches, the models yielded high accuracy with 88% for K-means clustering, 99% for SVM model, 97% for evaluation of Isolation Forest and Random Forest yielded 100% accuracy after training. These figures show that these models perform relatively well in their detectional task. But overall the two supervised models yielded better performance than the unsupervised because they take advantage of pre-embedded labels for training and directly optimize for the classification problem. Meanwhile, the unsupervised algorithms only learn "normal" features and detect anomalies based on statistical assumptions or cluster structures, leading to difficulties in distinguishing rare attack classes from common data. The second reason may be related to the limitation of unsupervised in anomaly detection. Isolation Forest and K-Means models learn the general structure of the data without knowing the anomalies in advance, so it is often difficult to detect rare attacks whose distributions differ slightly from those of normal data (source: Adonot). Finally, the differences between models may be due to scalability and complexity. Although Isolation Forest has O(n log n) complexity and is fast, the lack of labeling makes it perform worse than supervised, especially when balancing precision and recall (source: DeepLearningai).

7.3 Discussion of potential limitations and challenges encountered.

7.3.1 Random Forests (Supervised)

Despite the strong overall performance of the Random Forest model, several limitations were observed during development and evaluation. The most significant challenge stemmed from class imbalance in the dataset. Although the model performed exceptionally well on common classes like smurf and neptune, it struggled to detect rare attacks such as loadmodule, perl, and buffer_overflow. These

attack types received noticeably lower F1-scores, and in some cases were not detected at all. This gap is critical in the context of anomaly detection, where missing a stealthy attack (false negative) is far more costly than raising a false alarm.

Additionally, training the model on the full feature set led to repeated memory crashes and excessive compute time, requiring dimensionality reduction and subsampling during hyperparameter tuning. This limited the breadth of the search space and may have prevented further improvements in performance. Lastly, although Random Forests provide feature importance metrics, their interpretability in terms of why a specific attack was detected is limited. These limitations suggest that future work should prioritize methods for handling data imbalance such as class-weight adjustments, targeted oversampling, or ensemble approaches combining supervised and unsupervised models to ensure reliable detection across all threat types.

7.3.2 Support Vector Machine (Supervised)

The greatest limitation of a Support Vector Machine(SVM) model that I have found is the hardware used to run the program and train the model. This was the greatest challenge I faced with implementing this model. I was able to implement a simple SVM model, but it took roughly 32 hours to finish training. I was not able to properly tune the hyperparameters simply because my computer is unable to handle the workload imposed by my program. As of now, the training for that model is still ongoing. Currently, it has been over 150 hours since I started the training.

7.3.3 K-Means Clustering (Unsupervised):

The main limitation of using KMeans for this task is that it assumes clusters are spherical and evenly sized, which doesn't match the complex and imbalanced nature of the KDD-99 dataset. As a result, it tends to favor large, dense classes like smurf, while completely missing smaller or irregular attack types. Since KMeans is unsupervised, the process of assigning cluster labels based on majority voting can also lead to incorrect classifications, especially when rare classes are grouped into dominant clusters. To improve results, it would help to use clustering algorithms that handle varying cluster shapes and densities, and to apply preprocessing techniques like oversampling or dimensionality reduction to make the data more separable.

7.3.4 Isolation Forest (Unsupervised)

The difficulty in implementing the project may come from the fact that it was the first time I had to deal with such a large dataset. Planning to complete the task and fixing the code whenever there was an error was also done more carefully. Because each time the code was run, there were stages where it took hours for the program to run to the error part or after exporting the preprocessed data file or trained model, we would know whether the program was complete or not. Therefore, errors had to be limited as much as possible because the execution time of the programs was very long.

8. Conclusion:

Summary of the project's outcomes - Suggestions for future work or improvements.

Isolation Forests stood out as a strong anomaly detection model in this project due to their high recall and efficiency, especially when compared to other approaches, like the K-Means Clustering model, which struggled with all of those metrics. Additionally, unlike Random Forests, which struggled with class imbalance and required significant memory and compute resources—leading to crashes and the need for subsampling—Isolation Forests handled the full 10% dataset without overwhelming the system. While Random Forests delivered solid performance on common attacks, they often failed to detect rare threats, which is a serious drawback in cybersecurity applications where missing a stealthy attack can be costly. In

contrast, Isolation Forests showed better detection across a wider range of attack types, making them more reliable in high-stakes settings. The model's efficiency was also evident when compared to Support Vector Machines (SVMs), which required prohibitively long training times—over 150 hours in some cases—making hyperparameter tuning nearly impossible. Isolation Forests, on the other hand, were computationally efficient and scalable, managing to handle large datasets without the extreme bottlenecks SVMs encountered.

Overall, the Isolation Forest model combined strong anomaly detection capabilities with practical performance, particularly on large, imbalanced datasets like KDD-99. While it required careful planning and error handling due to the dataset size and lengthy run times, it consistently outperformed other models in recall. Its efficiency and balanced detection coverage made it a better option than SVMs and Random Forests, both in terms of speed and accuracy across a wide range of attack types. Given that recall is the most important metric for determining the effectiveness of a model—prioritizing the reduction of false negatives and ensuring that as many attacks as possible are captured—Isolation Forests proved to be the best candidate for network anomaly detection. Its ability to train on the full dataset and efficiently isolate outliers makes it a powerful and practical tool for identifying threats in network security systems.

9. Individual Contributions: Detailed account of each team member's specific contributions to the project.

Each team member went over their own models training process, and hyperparameter tuning

- Anh Huy Nguyen: Code: Data Preprocessing stage, trained Isolation Forest model, and evaluated Isolation Forest metric. Report document: complete part 3, 5.1, 5.2.4, 5.3.4, 6.4, 7.1.4, 7.2, 7.3.4
- Anthony Do: In charge of everything relating to the SVM model from writing the code, training the model, tuning the model, analyzing the model's performance, and writing each subsection of the report relating to the SVM model. Originally was meant to help with the Data Preprocessing, but Anh was able to finish it before I could help.
- Trevor Thayer: Responsible for implementing the Random Forest model for supervised learning. Included training a baseline model for feature selection, hyperparameter tuning using RandomizedSearchCV on a subset, and training the final model on the full dataset. Developed evaluation graphics for ROC curves and confusion matrices - and analyzed the model's class-level performance. Managed the GitHub repository and organized the final deliverables, including compiling the final README documentation (and preparing final zip file for submission).
- Isabelle Viraldo: Responsible for the K-Means Clustering algorithm. Trained, evaluated, and determined k hyperparameter. Discussed everything about the model within the final document. Contributed to the project proposal and final report.

10. References:

scikit-learn. (n.d.). *Evaluation of outlier detection estimators*. From scikit-learn.org/stable/auto_examples/miscellaneous/plot_outlier_detection_bench.html

Analytics Vidhya. (2021, July). *Anomaly Detection Using Isolation Forest – A Complete Guide*. From www.analyticsvidhya.com/blog/2021/07/anomaly-detection-using-isolation-forest-a-complete-guide/

ProjectAI. (n.d.). *Analyzing Confusion Matrix for Isolation Forest Model*. From projectai.in/projects/aa497e1d-9100-4290-8a1c-56e39173807a/tasks/a703c1fe-f5f0-4781-aff6-9a8a8629cdd7

DataMingle Analytics Pvt Ltd. (n.d.). *Spotting the Odd One Out: Understanding Data Anomalies Using Isolation Forest*. From https://www.data-mingle.ai/blog-details-1.html

2.3. clustering. (n.d.). Scikit-Learn. Retrieved May 9, 2025, from https://scikit-learn.org/stable/modules/clustering.html#k-means

KDD Cup 1999 Data. (n.d.). Retrieved May 9, 2025, from https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

KMeans. (n.d.). Scikit-Learn. Retrieved May 9, 2025, from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Matplotlib — visualization with python. (n.d.). Retrieved May 9, 2025, from https://matplotlib.org/

NumPy. (n.d.). Retrieved May 9, 2025, from https://numpy.org/

scikit-learn: Machine learning in Python — scikit-learn 1.6.1 documentation. (n.d.). Retrieved May 9, 2025, from https://scikit-learn.org/stable/

Scikit-learn Developers. (n.d.). *Random Forests*. Scikit-learn Documentation. Retrieved from https://scikit-learn.org/stable/modules/ensemble.html#random-forests

Scikit-learn Developers. (n.d.). *Randomized Parameter Optimization*. Scikit-learn Documentation. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html