Security Analytics Final Report

Network Intrusion Detections using Machine Learning Methods

Mengchu Li, Meihan Lin, Shuyao Tan, Yuyang Zhou

May 10, 2022

1 Project Overview

1.1 Problem Definition and Motivation

A network intrusion refers to any forcible or unauthorized activity on a digital network. These unauthorized activities almost always imperil the security of networks and their data. Nowadays, online brands and companies are the usual subjects of these attacks. Huge financial losses could be caused. However, traditional solutions are quite limited in finding out potential attacks, especially novel attacks. Therefore, we want to use machine learning models to predict if an incoming connection is malicious.

1.2 Data Source

The dataset is adapted from the KDD-CUP-99 contest. In this project, we propose to use the dataset of Network Intrusion Detection. This dataset has 41 features and can be used to classify if a connection is normal.

In this project, we are trying to distinguish the bad connections, called intrusions or attacks, and normal connections. This may provide us with some insight into detecting the potential malicious connection attempts.

For each TCP/IP connection, 41 quantitative and qualitative features are obtained from normal and attack data (3 qualitative and 38 quantitative features).

There are three types of features:

- Basic features of individual TCP connections
- Content features within a connection suggested by domain knowledge
- Traffic features computed using a two-second time window

The target class variable has 5 categories:

- 0 - DOS: denial-of-service, e.g. syn flood; (attack)

- 1 - normal (no attack)
- 2 - probing: surveillance and other probing, e.g., port scanning (attack)
- 3 - R2L: unauthorized access from a remote machine, e.g. guessing password (attack)
- 4 - U2R: unauthorized access to local superuser (root) privileges, e.g., various "buffer overflow" attacks (attack)

Here's the link of our dataset: https://www.kaggle.com/code/abaojiang/network-intrusion-detection-a-simp
data

2 System Approach

In our project, we follow the process of conducting data science projects. Given the Network Intrusion Detection dataset, we want to find ways to identify novel attacks in order to mitigate the risk of financial losses.

First, we conduct an exploratory data analysis (EDA), in order to gain preliminary understandings to our data. Next, we conduct data pre-processing to convert raw data into cleaned data. Next, we try to fit our data with different Machine Learning Algorithms that belongs to anomaly detection approach and conventional classification approach, in the hope of finding models with accurate predictions to the attacks. After that, we will evaluate our models based on their performance and propose what are the suitable models we should deploy to prevent network intrusions.

3 Methods Demonstration

3.1 Exploratory Data Analysis

In this section, we analyzed the data to see if there were any discrepancies, intriguing patterns, data correlations, and so on, which refers as the Exploratory Data Analysis(EDA).

3.1.1 Dataset

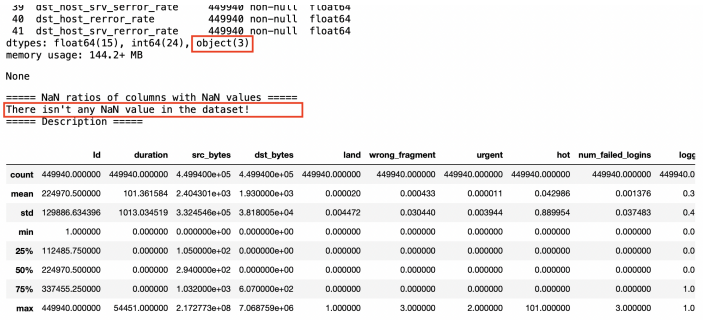
- **training.csv**: Training data containing 42 features and 1 column of ground-truths.
- **testing.csv**: Testing data containing 42 features and no labels.

  We'll solely utilize training.csv to find out the classifiers' generalization abilities. That is, the testing.csv will not be used.

3.1.2 Basic Description

Basic information about the training data is shown in this section, including data appearance, data types of features, etc.

From the basic analysis of the dataset(Figure 1), Several characteristics with binary values and a few with redundancy were discovered. And object data types for a few features must be encoded into numerical values as well.



```
39  dst_host_srv_serror_rate     449940 non-null  float64
40  dst_host_rerror_rate         449940 non-null  float64
41  dst_host_srv_rerror_rate     449940 non-null  float64
dtypes: float64(15), int64(24), object(3)
memory usage: 144.2+ MB

None

===== NaN ratios of columns with NaN values =====
There isn't any NaN value in the dataset!
===== Description =====
```

	Id	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logg
count	449940.000000	449940.000000	4.499400e+05	4.499400e+05	449940.000000	449940.000000	449940.000000	449940.000000	449940.000000	449940.0
mean	224970.500000	101.361584	2.404301e+03	1.930000e+03	0.000020	0.000433	0.000011	0.042986	0.001376	0.3
std	129886.634396	1013.034519	3.324546e+05	3.818005e+04	0.004472	0.030440	0.003944	0.889954	0.037483	0.4
min	1.000000	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	112485.750000	0.000000	1.050000e+02	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
50%	224970.500000	0.000000	2.940000e+02	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
75%	337455.250000	0.000000	1.032000e+03	6.070000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	1.0
max	449940.000000	54451.000000	2.172773e+08	7.068759e+06	1.000000	3.000000	2.000000	101.000000	3.000000	1.0

Figure 1: Part of Basic Description of Data

3.1.3 Feature Analysis

In this section, we used graphs to find distributions, relationships, and any other patterns that aren't obvious when looking at raw data.

- Unique Values

  See Figure 2, we can see the details of different unique values in categorical features:

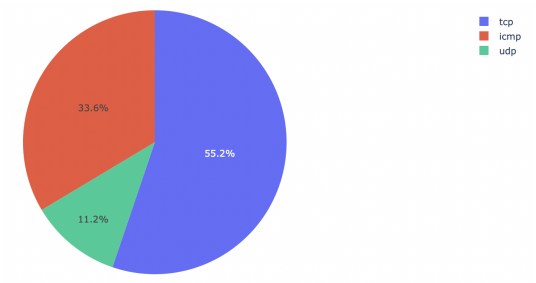
```
=====Unique values of protocol_type=====
['tcp' 'icmp' 'udp']
Number of unique values: 3

=====Unique values of service=====
['private' 'ecr_i' 'http' 'ntp_u' 'ftp_data' 'finger' 'other' 'pop_3'
 'smtp' 'ftp' 'eco_i' 'domain_u' 'supdup' 'discard' 'auth' 'bgp' 'Z39_50'
 'shell' 'netbios_ns' 'hostnames' 'telnet' 'uucp_path' 'mtp' 'http_443'
 'nnsp' 'time' 'urp_i' 'exec' 'iso_tsap' 'netbios_dgm' 'name' 'systat'
 'IRC' 'ldap' 'netstat' 'printer' 'link' 'gopher' 'ssh' 'efs' 'sunrpc'
 'daytime' 'X11' 'imap4' 'pop_2' 'uucp' 'whois' 'rje' 'sql_net' 'kshell'
 'vmnet' 'login' 'ctf' 'domain' 'echo' 'csnet_ns' 'courier' 'klogin'
 'urh_i' 'netbios_ssn' 'nntp' 'tim_i' 'remote_job' 'icmp' 'red_i' 'tftp_u']
Number of unique values: 66

=====Unique values of flag=====
['REJ' 'SF' 'S0' 'RSTO' 'S2' 'RSTR' 'S1' 'OTH' 'SH' 'S3' 'RSTOS0']
Number of unique values: 11
```

Figure 2: Overview of Unique Values

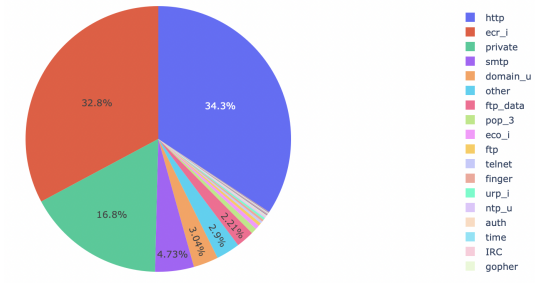
- Ratio of Each Unique Value

  In this part, we plotted several pie charts and bar charts to see if we can find some properties of these features. Refer to Figure 3 and 4 and below. Here's the description of some mentioned features:

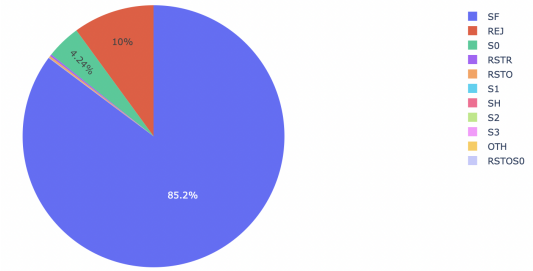
  - **counts:** normal or error status of the connection number of connections to the same host as the current connection in the past two seconds.
  - **srv_count:** number of connections to the same service as the current connection in the past two seconds.
  - **flag:** normal or error status of the connection.



(a) Distribution of Protocols

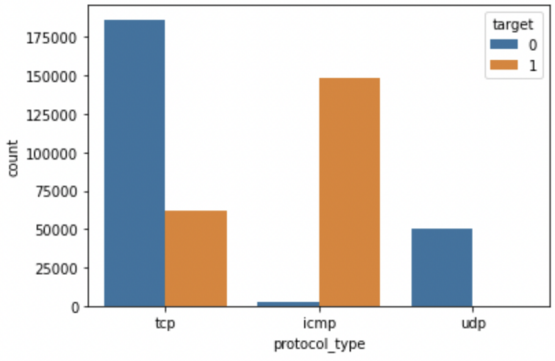


(b) Distribution of Services

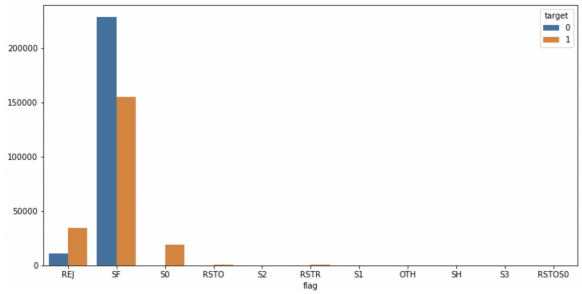


(c) Distribution of Flags

Figure 3: Distribution of Unique Values



(a) Target Count of Protocols

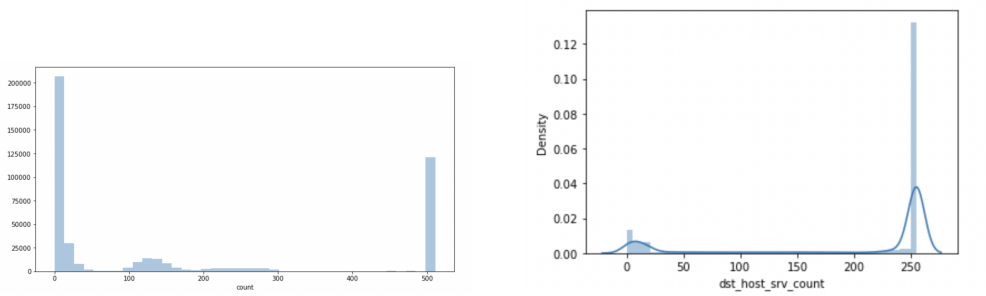


(b) Target Count of Flags

Figure 4: Target Count of Unique Values

- Ratio of other numeric values

  Refer to Figure 5.



(a) Ratio of Counts (b) Ratio of Host Services

Figure 5: Ratio of Some Numeric Values

- Ground-truth Distribution

  Refer to Figure 6

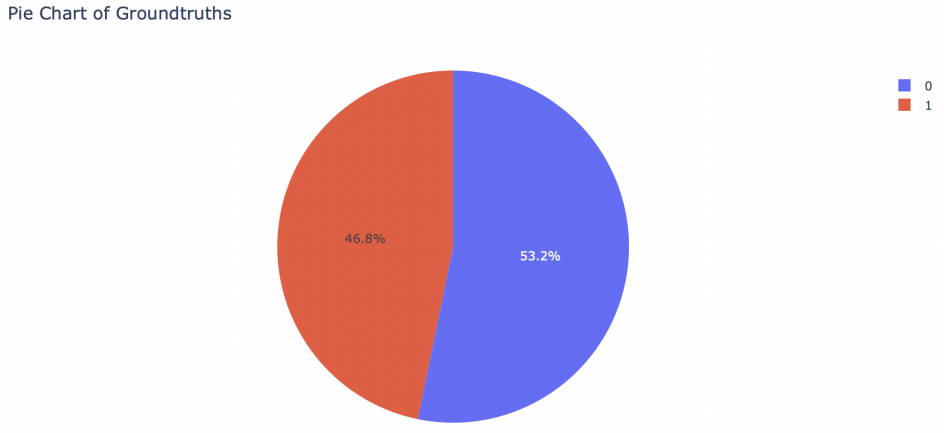


Figure 6: Attack Types Distribution

- Heatmap of original dataset

  Refer to Figure 7. And we also found two features highly correlated to the target, see in Figure 8.

3.1.4 Observations from Data Analysis

- In our dataset's class column "target," we discovered a slightly imbalance. However, it is insignificant; otherwise, we would resort to oversampling.
- 55.2% of traffic belongs TCP while 33.6% belongs to ICMP and rest to UDP.
- The majority of ICMP traffic was anomaly; most of UDP traffic was normal; and the proportion of normal connections in TCP is slightly higher.

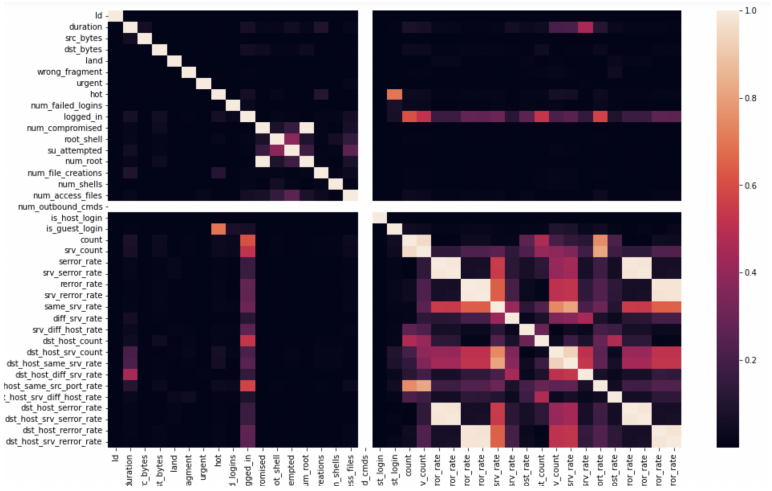


Figure 7: Features Heatmap

```
corr_with_target = X.corrwith(y).apply(abs)
corr_with_target[corr_with_target>0.7]
```

```
logged_in    0.707370
count        0.803544
dtype: float64
```

Figure 8: Two Highly Correlation with Target Features

- Most of the services are HTTP, ECR_i and private services.
- The traffic distribution based on flags was likewise unbalanced, with the majority of it having SF (Sign Flag).
- Most of the traffic with SF was normal, while that had S0 and REJ flag had anomaly connections.
- Most of the traffic recorded was unique.
- The number of connections with the same destination host and service was either extremely low or extremely high.

3.1.5 Observation From Heatmap

- We can observe from the correlation heatmap above that most of the data has a very low correlation. This is a good quality to have in our Machine Learning PPRCess.
- Few features, especially logged_in and count, have a good correlation with our target class, which will be useful for our model.

3.2 Data Pre-processing & Feature Selection

We used several typical data preparation procedures after the EDA. It was carried out everywhere we thought it would have an impact on our workflow.

We propose to conduct our analyses using several approaches covered in the class and do the data pre-processing(including data cleaning, encoding and etc.) and feature ranking & selections.

In our project, we are encoding the target class to 0s and 1s, so that it can be used for further analysis and training:

- 0 for "normal" connections – target = 1;
- 1 for "anomaly" connections – target in {0,2,3,4};

3.2.1 Data Pre-processing

Some methods used here:

- **Data cleaning** - drop duplicates, handling missing values, finding outliers etc.
- **Data Encoding** - Most of models are unable to process strings or objects. As a result, the data must be translated into numerical data. Data encoding is the term for this procedure (also data transformation).

First, we do the **data cleaning** part. At the very beginning, we want to drop those useless columns and to check whether there are any duplicate and missing values in our dataset. If there is any, we will need to drop them.

From the previous EDA part, we find that the "Id" column looks like useless, so we drop it. Then we can also find in the EDA that our dataset doesn't have missing values in each features, so we ignore this step and proceed to drop the duplicates. After dropping the duplicates and dropping useless features, the shape of data reduces from (449940, 43) to (235464, 42). From the pie chart of the new target(see Figure 9), we can see that a lot amount of anomaly data are the same, so the remaining data is largely normal data, which is pretty typical for anomaly detection.

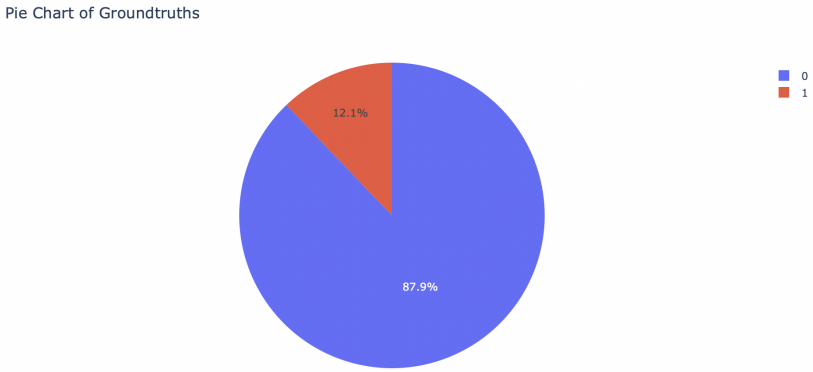


Figure 9: Distribution of Target after Dropping duplicates

Then, after data cleaning, let's **check the outliers** in our datasets. Such as deleting the extraneous 2 in feature which must has 0 or 1.

Based on the describe data(Figure 10). We first create a table for each potentially anomalous characteristic (such as su_attemped with a max value of 2), grouping by feature values and target type counts. We could simply locate and delete outliers using this table.

```
df.describe()
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_con
count	235464.000000	2.354640e+05	2.354640e+05	235464.000000	235464.000000	235464.000000	235464.000000	235464.000000	235464.000000	2354
mean	140.371360	3.654130e+03	3.627808e+03	0.000038	0.000713	0.000021	0.077311	0.001869	0.731216	
std	1266.381212	4.594754e+05	5.271162e+04	0.006182	0.040692	0.005452	1.225303	0.043871	0.443328	
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	1.470000e+02	1.050000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	2.320000e+02	4.760000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	
75%	0.000000	3.130000e+02	2.265000e+03	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	
max	54451.000000	2.172773e+08	7.068759e+06	1.000000	3.000000	2.000000	101.000000	3.000000	1.000000	74

8 rows × 39 columns

Figure 10: Feature Describe Information

Here's all the tables from which we find outliers, refer to Figure 11 and Figure 12:

	su_attempted	target
0	0	235435
1	1	14
2	2	15

(a) su_attempted

	src_bytes	target
0	0	32854
1	1	474
2	2	1
3	4	6
4	5	42
...	...	...
4410	7847476	1
4411	11396904	1
4412	14584085	1
4413	24418776	1
4414	217277339	1

(b) src_bytes

	num_root	target
0	0	234314
1	1	505
2	2	31
3	3	1
4	4	18
5	5	41
6	6	236
7	9	291
8	10	1
9	17	1
10	26	2
11	39	1
12	41	1
13	54	1
14	278	1
15	401	1

(c) num_root

	num_compromised	target
0	0	234954
1	1	366
2	2	46
3	3	27
4	4	20
5	5	5
6	6	7
7	7	3
8	9	3
9	10	2
10	12	3
11	13	1
12	14	1
13	16	2
14	21	2
15	22	1
16	27	1
17	38	1
18	41	1
19	281	1
20	381	1
21	7479	1

(d) num_compromised

Figure 11: Tables of Features Containing Outliers(Part1)

Based on what we observed from the feature count table, we did these things:

- Delete outlier "2" in **su_attempted** feature.
- Delete outlier "7479" in **num_compromised** feature.
- Delete outlier "217277339" in **src_bytes** feature.
- Delete outlier "101" in **hot** feature.

22	24	49
23	25	1
24	28	220
25	30	49
26	33	1
27	101	1

(a) hot

	num_outbound_cmds	target
0	0	235445

(b) num_outbound_cmds

Figure 12: Tables of Features Containing Outliers(Part2)

- Delete outlier "401" in **num_root** feature.
- From (b) of Figure 12, we found that **num_outbound_cmds** have only one unique value i.e., 0. This introduces redundancy, as a feature with only 1 value won't affect our model. We can remove it and reduce the size of the data and hence improve the training process.

Finally, we do the **data encoding** part.

Here, we propose a Custom Label Encoder for handling unknown values, refer to Figure 13. Then the data is successfully cleaned and encoded, and we will do further process in the Feature Selection part.

```
# Custom Label Encoder for handling unknown values
class LabelEncoderExt(object):
    def __init__(self):
        self.label_encoder = LabelEncoder()

    def fit(self, data):
        self.label_encoder = self.label_encoder.fit(list(data) + ['Unknown'])
        self.classes_ = self.label_encoder.classes_
        return self

    def transform(self, data):
        new_data = list(data)
        for unique_item in np.unique(data):
            if unique_item not in self.label_encoder.classes_:
                new_data = ['Unknown' if x==unique_item else x for x in new_data]
        return self.label_encoder.transform(new_data)
```

Figure 13: Custom Label Encode

3.2.2 Feature Selection

In this section, we explored the correlations between 40 features after data pre-processing and then using different kinds of approach to filter out the best 10 features, which are commits to every feature selection we have used.

- **Heatmap**

  We generate the heatmap to overall look through the correlations between the features.

- **Sorting Correlation Table**

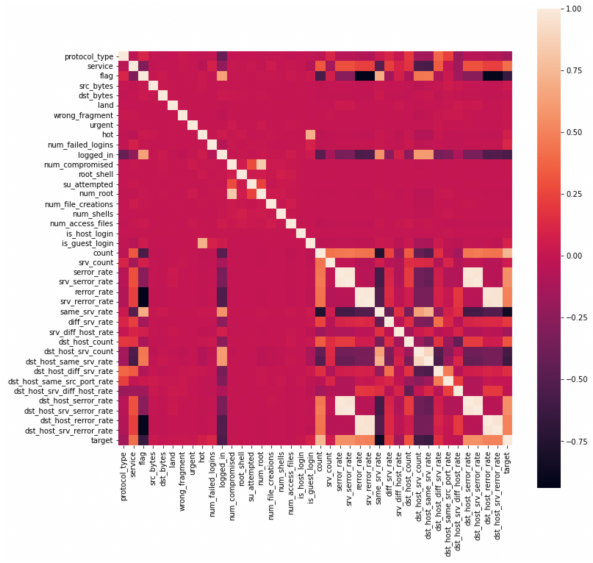


Figure 14: Heatmap of Features

From the Figure 14, we notice that there some dark and light areas in the heatmap, which means we have some features that are highly correlated. In order to see these features and correspond correlations clear, we plot out the sorted correlation using histogram. We set the correlation threshold as 0.8 and we filtered out the correlated features that has cor bigger than the threshold.

Figure 15: Plot of Correlated Features

From the Figure 15, we can see there are so many difference pairs features with strong correlations, it is not easy to handle these pairs of features one by one using PCA, which is also not a efficient way to get important features. Then we tried to implement some feature selections techniques to find the most important features.

- **Order the importance of all the features**
  - **Feature Selection Using Random Forest** Random forest feature selection falls under the area of Embedded techniques. Filter and wrapper methods are combined in embedded

methods. Algorithms with built-in feature selection methods are used to implement them. Based on the performance of Random Forest Classifier, we got a bar graph showing the importance of all the feature:

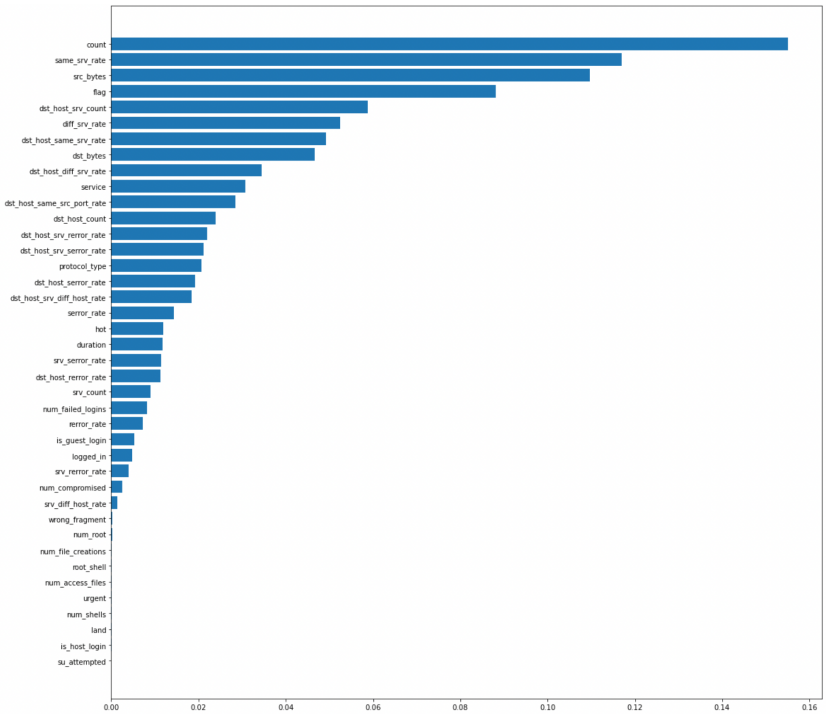


Figure 16: Feature Order by Random Forest Classifier

From Figure 16, we can see several important features selected are: *count, same_srv_rate, src_bytes, flag, dst_host_srv_count, diff_srv_rate, dst_host_same_srv_rate, dst_bytes, dst_host_diff_srv_rate, service* .

– **Univariate Feature Selection**

As for univariate feature selection, Each attribute is compared to the target variable to check if there is a statistically significant relationship between the two. Analysis of variance is another name for it (ANOVA). We disregard the other features while analyzing the link between one feature and the target variable. That is why it is referred to as "univariate." Each feature has a test score associated with it[5]. For our analysis, we choose chi-squared to be our statistical test tool and we got the best ten features with the highest chi-squared score.

From Figure 17, we can see there are some features overlapped with the best ten features we got from random forest, which is: *src_bytes, count, dst_bytes, dst_host_srv_count, service, flag*

– **Select the Best set of features according to RFE**

RFE(Recursive Feature Elimination) is a feature selection algorithm with a wrapper. This means that in the core of the technique, a different machine learning algorithm is given

	Features	Score
4	src_bytes	7.694078e+08
21	count	2.153583e+07
5	dst_bytes	6.923755e+06
31	dst_host_srv_count	4.248411e+06
30	dst_host_count	1.590108e+06
2	service	1.927078e+05
3	flag	7.248479e+04
37	dst_host_srv_serror_rate	6.821255e+04
24	srv_serror_rate	6.718275e+04
23	serror_rate	6.696301e+04

Figure 17: Feature Order by Chi-Square

and used, which is wrapped by RFE and used to help choose features. Filter-based feature selections, on the other hand, score each feature and select the features with the highest (or lowest) score. RFE works by searching for a subset of features in the training dataset, starting with all of them and successfully deleting them until the target number of features remains[1].

We use Random Forest Classifier to be the estimator of our RFE. And the best ten features we got are: *service, flag, src_bytes, dst_bytes, count, same_srv_rate, diff_srv_rate, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate*

– **The Best Ten Features**

Based on all the algorithms we used to do the feature selections, we got the best ten features: *service, flag, src_bytes, dst_bytes, count, same_srv_rate, diff_srv_rate, dst_host_srv_count, dst_host_srv_serror_rate, dst_host_diff_srv_rate*

- Standardization & Normalization We use StandardScaler from sklearn to do the Standardization.

3.3 Anomaly Detection Methods

Anomaly detection models are known to be able to predict if connections are abnormal, even though the models have not been trained on them. Since our dataset is greatly imbalanced, with more data in normal connections comparing to abnormal connections, we find it an appropriate use case for anomaly detection. Also, in our expectation, anomaly detection may be able to shed some lights on identifying novel attacks[6].

3.3.1 Nearest Neighbor

Nearest Neighbor is an ideal model to be used for anomaly detection. The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these[3]. Based on this principle, we can extract only normal data from the training file as the training samples, and only calculate the distance to the nearest point

in the normal dataset. We also need to set a suitable threshold for the distances, which is used to predict whether it's normal or intrusive. Because the training samples are only the normal data, if the distance of the new point is very small (i.e., smaller than the threshold), meaning it's more "similar" to a normal data point, conversely, if the distance of the new point is very large (i.e., larger than the threshold), meaning it differs from the normal data point, we can assume it as an intrusive data.

In our experiments, we split the data set into training, valid and testing set: training set includes 70% normal data, valid set includes 15% normal data and 50% intrusive data, and testing set includes 15% normal data and 50% intrusive data. The distance we use is the standard Euclidean distance for this model, which is the most common choice.

To find the best threshold, we use the f1 score as a metric. We first calculate the maximum distance of the intrusive data points in the valid dataset, and set a threshold range from 0 to this maximum value, then iterate the threshold in this range to predict the label and calculate the f1 score for each threshold. The best threshold is obtained when the corresponding f1 score reaches it maximum value. From Figure 18, the best threshold is 0.01555.

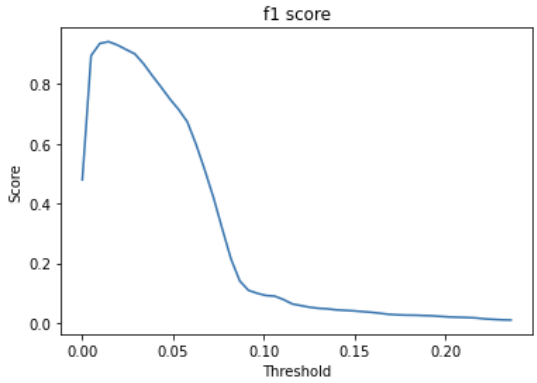


Figure 18: F1 score for each threshold under Nearest Neighbor

Figure 19 shows the PRC curve with the AUCPR value.

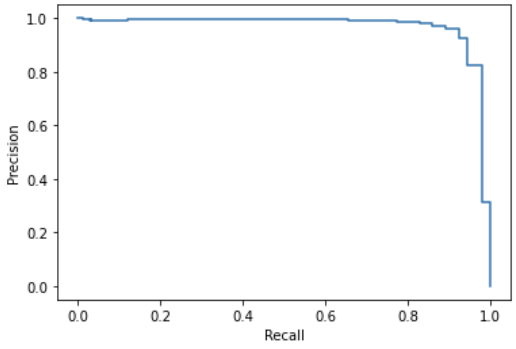


Figure 19: PRC curve with the AUCPR for Nearest Neighbor

The accuracy of this model is around 96.3%, with a confustion matrix as follows Figure 20:

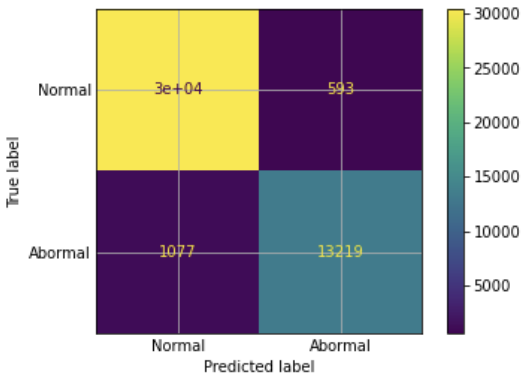


Figure 20: Confusion Matrix for Nearest Neighbor

3.3.2 Gaussian Mixture Model

Gaussian mixture model (GMM) is another desirable model for anomaly detection. A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters[2]. If the normal data is Gaussian with a mean and variance, points with low probability assignments under the given prior may be labeled as anomalous. We still use the normal data extracted from the training dataset to fit Gaussian mixture model, and calculate the corresponding log-likelihood of each new point under this model. Because the value of log-likelihood represents the probability of occurrence under the distribution constructed from the normal dataset, the larger this value is, the more normal this point is. Therefore, we can set a threshold of log-likelihood to predict the labels of new data, any value is less than the set threshold will be identified as an anomalous point.

In our experiments, we split the data set into training, valid and testing set: training set includes 70% normal data, valid set includes 15% normal data and 50% intrusive data, and testing set includes 15% normal data and 50% intrusive data. We also try out different parameter *n_components* from 1 to 3, it can get the highest accuracy when *n_components* is set to 2.

The method of finding best threshold is similar to the previous one. We first separate the normal data and intrusive data in the valid dataset, and then calculate the average log-likelihood for each category. After we get these two average value, we can obtain a threshold range, in which the final threshold is chosen based on the corresponding f1 score. The best threshold is obtained when the corresponding f1 score reaches it maximum value, it is -5.45740.

Figure 21 shows the PRC curve with the AUCPR value.

The accuracy of this model is around 94.4%, with a confusion matrix as follows Figure 22:

3.3.3 One Class Support Vector Machine

We also used One Class Support Vector Machine (OCSVM) to conduct novelty detection. Novelty detection is the problem of identifying test data that differ in some way from the data available during

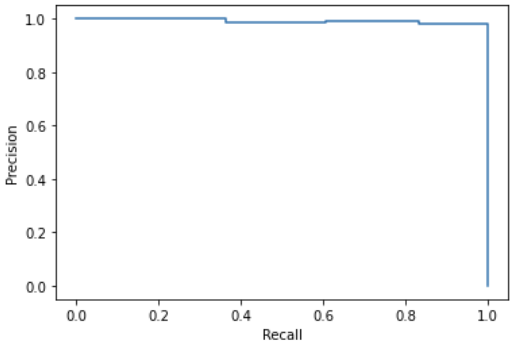


Figure 21: PRC curve with the AUCPR for Gaussian Mixture Model

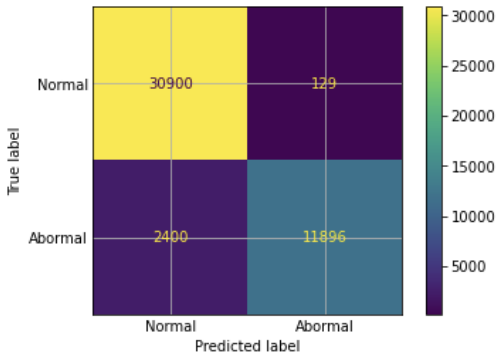


Figure 22: Confusion Matrix for Gaussian Mixture Model

training is known[4]. This may be seen as "one-class classification", in which a model is constructed to describe "normal" training data.

The method we propose to use, One Class SVM, is a type of SVM that can be utilized for anomaly identification in an unsupervised scenario. What separates it with the normal SVMs are that, instead of separating data with different classes, the boundary separates positive examples with negative ones. The one-class SVM finds a hyperplane that separates the given dataset from the origin and is as near as possible to the data points.

In this project, we propose to use RBF as the kernel of our OCSVM model. RBF kernel can provide a soft-margin which can provide us with more flexibility in classifying our data. There are two parameters of our concern, namely the kernel coefficient γ which determines the radius of the kernels, and ν which controls the training errors and the number of support vectors. We use an exhaustive search approach in order to find the best parameter set ν and γ. We used library GridSearchCV, which utilizes K fold cross validation with K set to 5. The best parameter set we get is $\nu = 0.001$, and $\gamma = 0.25$ (see Figure 23).

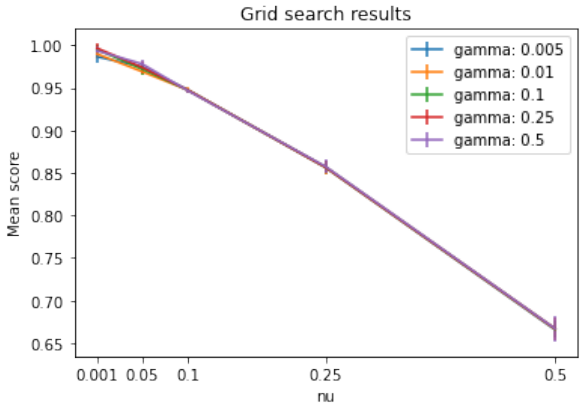


Figure 23: Exhaustive Search for best parameter set

We trained our model using data only with normal transactions, and test using data with both normal and abnormal transactions. There are 164,811 entries of data in the training set, and 76,353 entries of data in the testing set. A detailed split of the data can be found in Figure 24.

Next, we train our OCSVM model with the fine-tuned parameter set. The accuracy we get is 72% with a confusion matrix provided (Figure 25).

The PRC curve with the AUC of 94% (Figure 26).

3.4 Misuse Detection Methods

In this project, we also train some Machine Learning algorithms to classify the type of connections in a supervised manner. The models in our concern includes Naive Bayes, Ridge Classifier, Random Forest, XGBoost, and K Nearest Neighbors.

```python
# get normal and abnormal data in df
y = df[df['target'] == 1]['target']
X = df[df['target'] == 1].drop(['target'], axis=1, inplace=False)

y_outlier = df[df['target'] == -1]['target']
X_outlier = df[df['target'] == -1].drop(['target'], axis=1, inplace=False)

# split data into train and test
# 50% normal only data for training
X_train, X_test_normal, y_train, y_test_normal = train_test_split(X, y, test_size=0.3, random_state=42)

# 50% normal + 50% abnormal data for testing
_, X_test_ab, _, y_test_ab = train_test_split(X_outlier, y_outlier, test_size=0.5, random_state=42)

# # # prepare data for testing
X_test = pd.concat([X_test_normal, X_test_ab])
y_test = pd.concat([y_test_normal, y_test_ab])
```

Figure 24: Code for data split

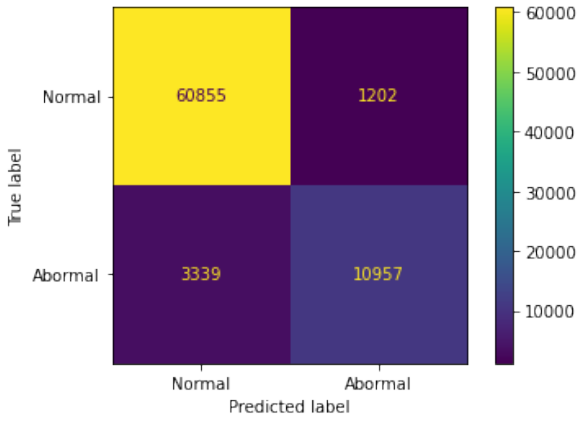


Figure 25: Confusion Matrix for One Class SVM

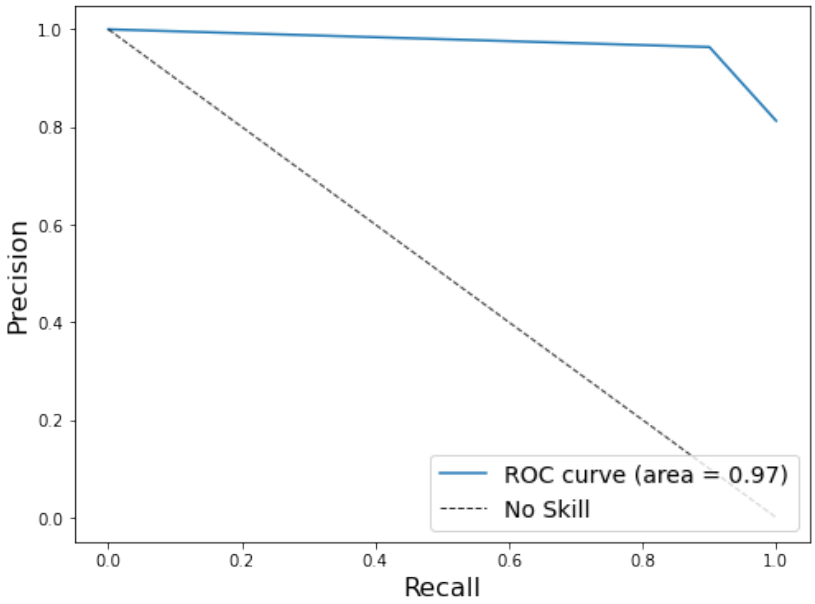


Figure 26: PRC curve with the AUC for One Class SVM

3.4.1 Naive Bayes Classifier

Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features[7]. We split our dataset with 70% of training data and 30% of testing data, containing 164,811 training data and 70,633 testing data. We use K Fold cross validation with five folds for training and validation, and then verifying the result using the testing set. K-Fold cross validation could provide us with the best assessment of our model given a variety of accuracy values for (relatively) distinct data sets. The number of data in training set is 178,157, and the number of data in testing set is 70,633. The accuracy score for our model is around 97%, with a confusion matrix (Figure 27).

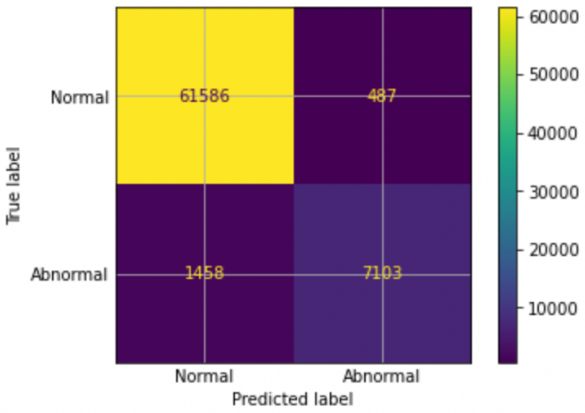


Figure 27: Confusion Matrix for Naive Bayes

The PRC curve with the mean AUC of 96% is obtained (Figure 28).

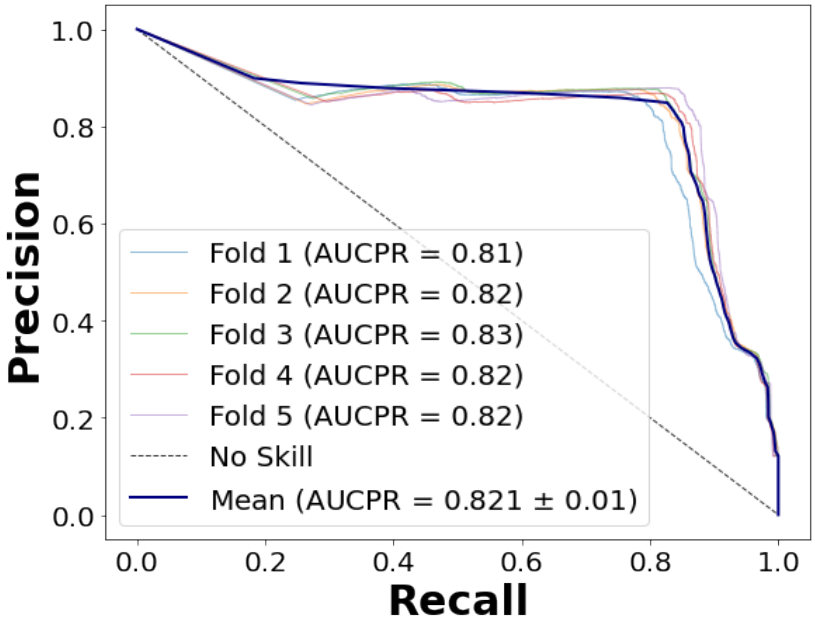


Figure 28: PRC curve with the AUC for Naive Bayes

More detailed classification reports can be seen in Figure 29:

```
============================ Naive Bayes Model Evaluation ============================

Cross Validation Mean Score:
 0.9516

Model Accuracy:
 0.9474

Classification report:
               precision    recall  f1-score   support

           0       0.98      0.96      0.97      4349
           1       0.76      0.88      0.81       651

    accuracy                           0.95      5000
   macro avg       0.87      0.92      0.89      5000
weighted avg       0.95      0.95      0.95      5000
```

Figure 29: Classification report for Naive Bayes

3.4.2 Linear Model: Ridge Classifier

Ridge Classifier uses Ridge regression. This classifier changes the goal values to -1, 1 before treating the problem as a regression problem (multi-output regression in the multi-class case). We split our dataset with 70% of training data and 30% of testing data, containing 164,811 training data and 70,633 testing data. We use K Fold cross validation with five folds for training and validation, and then verifying the result using the testing set. The accuracy score for our model is around 97.5%, with a confusion matrix (Figure 30).

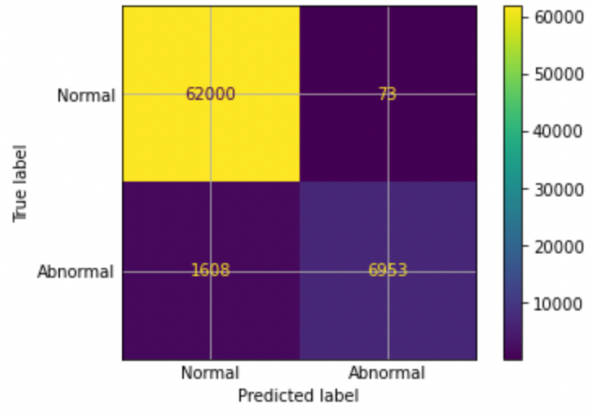


Figure 30: Confusion Matrix for Ridge Classifier

The PRC curve with the AUC is 95% (Figure 31).

More detailed classification reports can be seen in Figure 32:

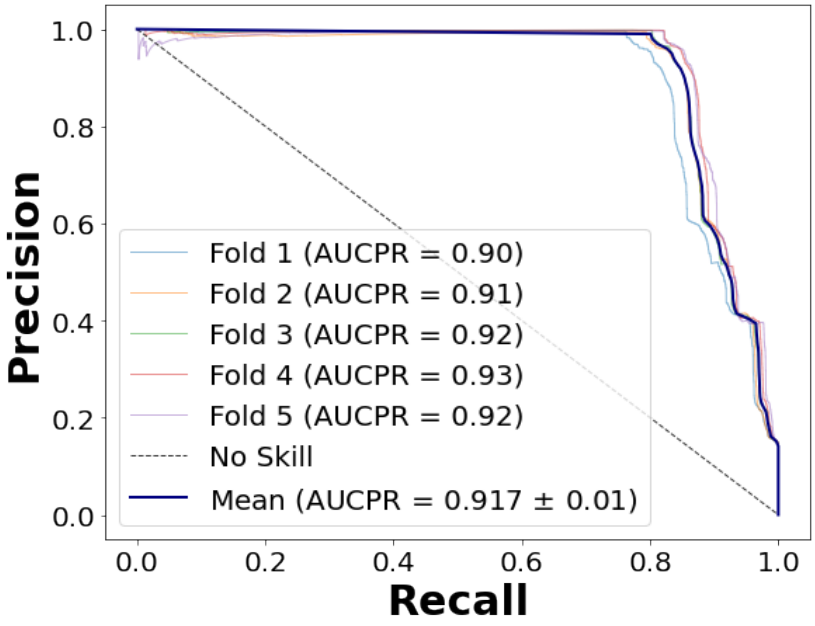


Figure 31: PRC curve with the AUC for Ridge Classifier

```
============================== Ridge Classifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.9729023027483574

Model Accuracy:
 0.9743466319336297

Classification report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     62073
           1       0.93      0.86      0.89      8561

    accuracy                           0.97     70634
   macro avg       0.95      0.92      0.94     70634
weighted avg       0.97      0.97      0.97     70634
```

Figure 32: Classification report for Ridge Classifier

3.4.3 Random Forest

Random Forest is a popular supervised learning models. It creates decision trees from various samples, using the majority vote for classification and the average for regression. Comparing to Decision Tree, Random Forest are less sensitive to the impact caused by the imbalance of the data set, so we think it fits our use case better. We splitted our dataset with 70% of training data and 30% of testing data, containing 164,811 training data and 70,633 testing data. We use K Fold cross validation with five folds for training and validation, and then verifying the result using the testing set.

The accuracy score for our model is around 99%, with a confusion matrix (Figure 33). More detailed classification reports can be seen in Figure 35:

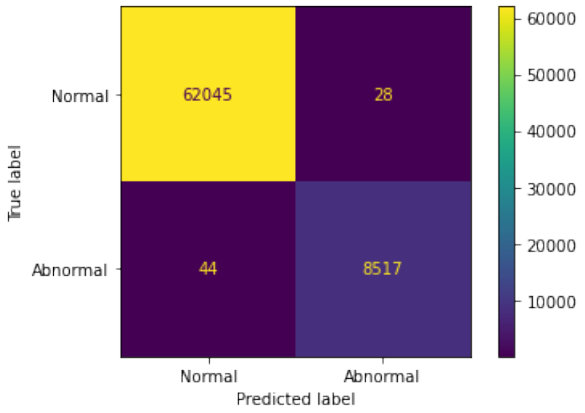


Figure 33: Confusion Matrix for Random Forest

The PRC curve with an average AUC of 99% is obtained (Figure **??**). More detailed classification reports can be seen in Figure 35:

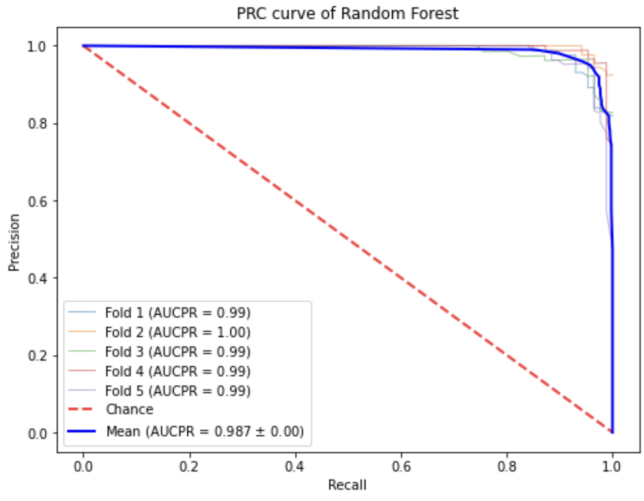


Figure 34: PRC curve with the AUCPR for Random Forest

```
============================== Random Forest Model Evaluation ==============================

Cross Validation Mean Score:
 0.9990837997174215

Model Accuracy:
 0.9989806608715349

Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     62073
           1       1.00      0.99      1.00      8561

    accuracy                           1.00     70634
   macro avg       1.00      1.00      1.00     70634
weighted avg       1.00      1.00      1.00     70634
```

Figure 35: Classification report for Random Forest

3.4.4 XGBoost

Since Random Forest has provided us with satisfying result, we further fits the dataset to XGBoost. XGBoost is a gradient boosting-based ensemble Machine Learning technique that leverages decision trees, which is proved to be memory efficient and accurate. We split our dataset with 70% of training data and 30% of testing data, containing 164,811 training data and 70,633 testing data. We use K Fold cross validation with five folds for training and validation, and then verifying the result using the testing set.

The accuracy score for our model is around 99%, with a confusion matrix (Figure 36).

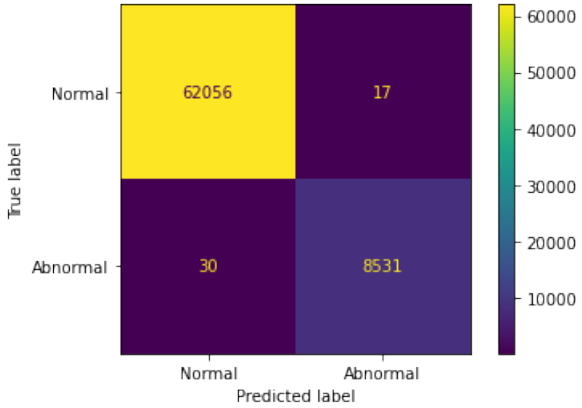


Figure 36: Confusion Matrix for XGBoost

The PRC curve with an average AUC of 99% is obtained (Figure 37). More detailed classification reports can be seen in Figure 38:

3.4.5 K-Nearest Neighbors

Neighbors-based classification is a type of instance-based learning in that it does not aim to build a generic internal model but instead just saves instances of the training data. Classification is calculated by a simple majority vote of each point's nearest neighbors: a query point is assigned the data class

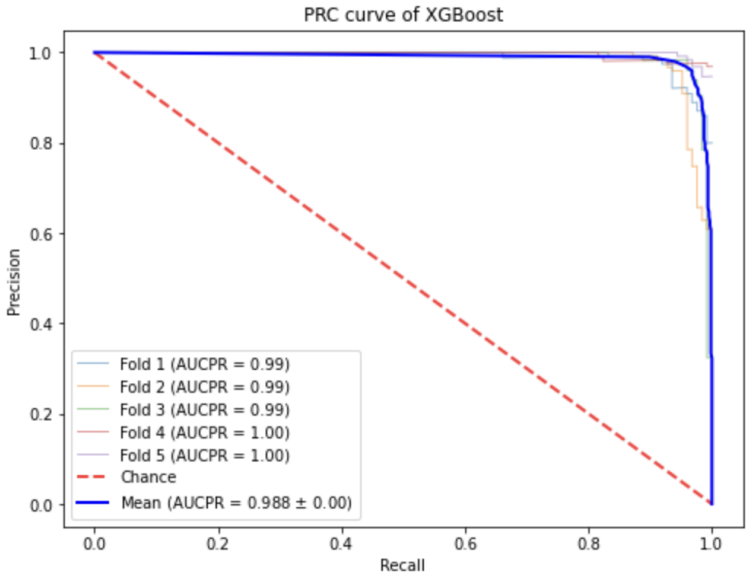


Figure 37: PRC curve with the AUCPR for XGBoost

```
============================== XGBoost Model Evaluation ==============================

Cross Validation Mean Score:
 0.9992718936274452

Model Accuracy:
 0.9993345980689187

Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     62073
           1       1.00      1.00      1.00      8561

    accuracy                           1.00     70634
   macro avg       1.00      1.00      1.00     70634
weighted avg       1.00      1.00      1.00     70634
```

Figure 38: Classification report for XGBoost

having the most representation among its nearest neighbors.

The basic theory of the k-neighbors classification in $KNeighborsClassifier$ is the same as the nearest neighbor mentioned in section 3.3.1. They both use the distance to the nearest neighbors to label new points. But they also have differences in training dataset and the number of neighbors. For the nearest neighbor model used in anomaly detection, it is trained by only normal data, and we just calculate the nearest neighbor (i.e., parameter k = 1). For the k nearest neighbor classification used in misuse detection, the training dataset includes both normal data and intrusive data, and the parameter k is optimal. The optimal choice of the value k is highly data-dependent: in general a larger k suppresses the effects of noise, but makes the classification boundaries less distinct.

In our experiments, We split our dataset into training and testing dataset: 70% normal data and 70% abnormal data for training set, 30% normal data and 30% abnormal data for testing set. The distance we use is standard Euclidean distance. We use K Fold cross validation with five folds for training and validation, and then verifying the result using the testing set. We also try different parameter k from 2 to 10, and the model can achieve highest accuracy when k is set to 3. The accuracy score for our model is around 99.7%, with a confusion matrix as follows Figure 39.

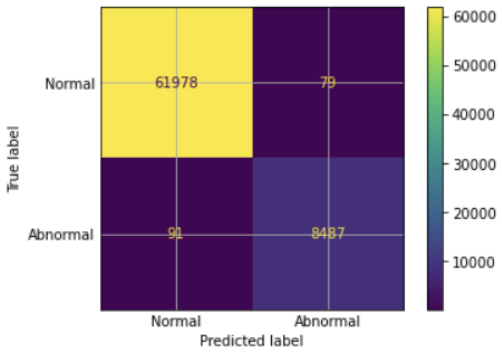


Figure 39: Confusion Matrix for K-Nearest Neighbors

The PRC curve is shown in Figure 40. More detailed classification reports can be seen in Figure 41.

3.5 Model Evaluation & Comparison

For model evaluation, we used K-fold cross-validation to evaluate our model. Additionally, confusion matrix and PRC-AUC curve are provided to better measure our work. From the results we generate for each model, we found that all three models can generate satisfactory results, while Nearest Neighbor have slightly better prediction for Anomaly Detection.

On the other hand, K Nearest Neighbor, Random Forest and XGBoost both provides satisfying results for classifying normal and abnormal attacks. The extremely high accuracy may partly due to that most of the data are normal connections. The high AUPRC may due to that models could have overfitted since training and testing data are similar to each other.

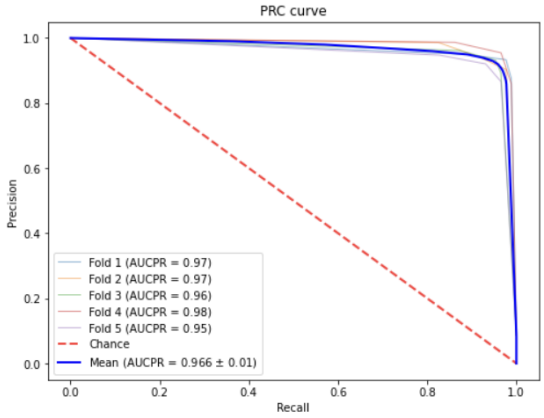


Figure 40: PRC curve with the AUCPR for K-Nearest Neighbors

```
============================== KNN Model Evaluation ==============================

Cross Validation Mean Score:
 0.9971603664826162

Model Accuracy:
 0.9975932611311673

Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     62057
           1       0.99      0.99      0.99      8578

    accuracy                           1.00     70635
   macro avg       0.99      0.99      0.99     70635
weighted avg       1.00      1.00      1.00     70635
```

Figure 41: Classification report for K-Nearest Neighbor

4 Conclusion & Future Work

In this project, we conducted a data science project by following the process of Exploratory Data Analysis, data cleaning and selected top 10 features, then tried out different Anomaly Detection and Misuse Detection Algorithms. We conclude that *service, flag, src_bytes, dst_bytes, count, same_srv_rate, diff_srv_rate, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate* are the best 10 features. Among the models we tested, Nearest Neighbors is a better as anomaly detection model, and K Nearest Neighbors, Random Forest and XGBoost are better as Classification Models.

There are a few things we can do to improve our work in the future. First, we may try to train our models with features we selected using different methods, and compare the prediction results. Moreover, we may try to combine our findings in the anomaly detection and misuse detection, such as filtering out part of the data that is concluded as "normal" by the anomaly detection models. We can also explore other state-of-the-art neural networks and compare their results.

References

[1] Jason Brownlee. Recursive feature elimination (rfe) for feature selection in python. https://machinelearningmastery.com/rfe-feature-selection-in-python/. Published May 25, 2020.

[2] Scikit-Learn Official Document. Gaussian mixture models. https://scikit-learn.org/stable/modules/mixture.html. Accessed April 28, 2022.

[3] Scikit-Learn Official Document. Nearest neighbors. https://scikit-learn.org/stable/modules/neighbors.html. Accessed April 28, 2022.

[4] Scikit-Learn Official Document. Novelty and outlier detection. https://scikit-learn.org/stable/modules/outlier_detection.html. Accessed April 28, 2022.

[5] Richard Liang. Feature selection using python for classification problems. https://towardsdatascience.com/feature-selection-using-python-for-classification-problem-b5f00a1c7028. Published August 6, 2019.

[6] AVI Networks. What is anomaly detection. https://avinetworks.com/glossary/anomaly-detection. Accessed April 28, 2022.

[7] WikiPedia. Naive bayes classifier. https://en.wikipedia.org/wiki/Naive_Bayes_classifier. Accessed April 28, 2022.