One-Class SVM for Anomaly Detection

1.1 Definition

One-class support vector machines aim to detect any anomalies or noise within datasets *(Erfani, 2016)* as they are trained to properly recognise what is considered "normal" within a network. This is through the process of identifying specific features *(Mounish, 2024),* enabling the model to detect any irregularities to successfully label attacks such as DDoS and SQL injections. In further detail, this method is implemented through the use of hyperplanes, which acts as a separator for the attack and normal samples *(Wang, 2023)* within a dataset. As demonstrated in figure x, it is conveyed how the hyperplane is a line that separates the anomalous and normal samples, making it a boundary where the data is best separated *(Sapien, 2024).*

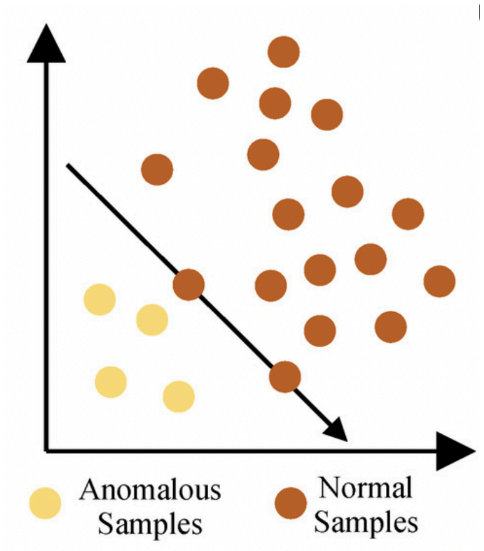
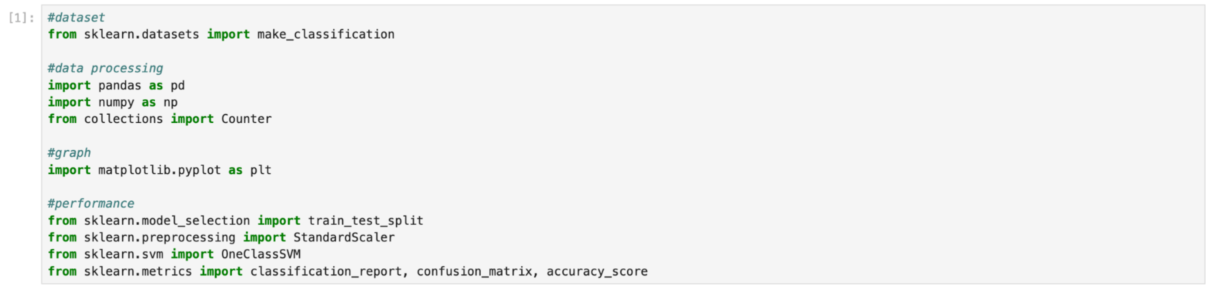


Figure 1

A kernel function within the model is utilised to map these samples, transforming the input data into a higher dimensional space *(Turing, 2022)*. This enables the model to easily identify patterns of normal behaviour in a non-linear manner *(Erfani, 2016).* By corresponding data points in an unsupervised manner *(Yang, 2021)*, one class support vector machines are able to process more complex data by organising the input into specific clusters *(Google Cloud, 2025).* Thus, these models are effective at handling anomaly based intrusion detection tasks in machine learning due to their flexibility with defining complex nonlinear boundaries between anomalous and normal data *(Yang, 2021).*

1.2 Key Components

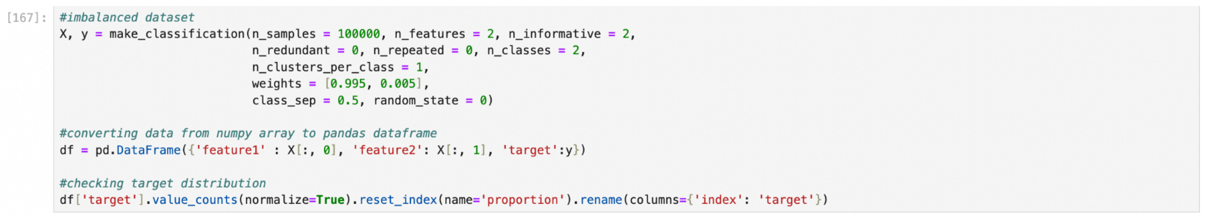
1. Importing libraries



```
[1]: #dataset
     from sklearn.datasets import make_classification

     #data processing
     import pandas as pd
     import numpy as np
     from collections import Counter

     #graph
     import matplotlib.pyplot as plt

     #performance
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.svm import OneClassSVM
     from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

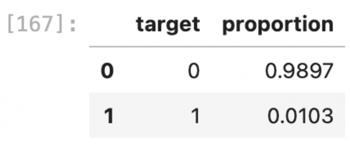
Figure 2

2. **Generating a synthetic imbalanced dataset for testing**: This helps evaluate the model's performance through simulating the task of intrusion detection. There are 100,000 records generated with 2 specific features. There are 0.5% anomalies which are labeled as "class 1", and there are 99.5% normals which are labeled as "class 0".



```
[167]: #imbalanced dataset
       X, y = make_classification(n_samples = 100000, n_features = 2, n_informative = 2,
                                  n_redundant = 0, n_repeated = 0, n_classes = 2,
                                  n_clusters_per_class = 1,
                                  weights = [0.995, 0.005],
                                  class_sep = 0.5, random_state = 0)

       #converting data from numpy array to pandas dataframe
       df = pd.DataFrame({'feature1' : X[:, 0], 'feature2': X[:, 1], 'target':y})

       #checking target distribution
       df['target'].value_counts(normalize=True).reset_index(name='proportion').rename(columns={'index': 'target'})
```

Figure 3

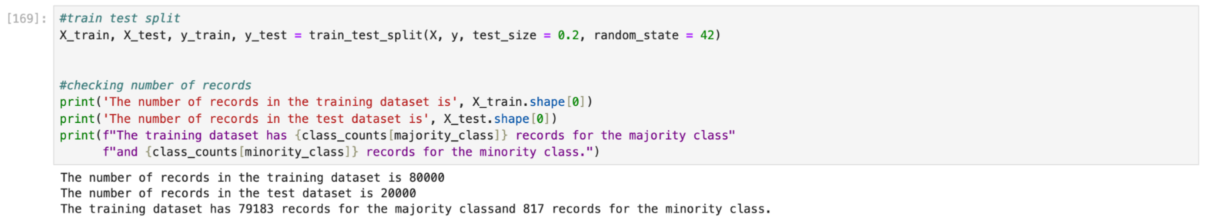


[167]:

	target	proportion
0	0	0.9897
1	1	0.0103

Figure 1

3. **Splitting training and test sets:** 80% for training and 20% for testing

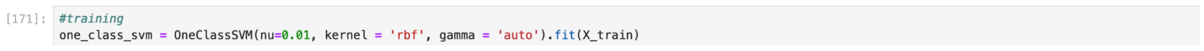


```
[169]: #train test split
       X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)


       #checking number of records
       print('The number of records in the training dataset is', X_train.shape[0])
       print('The number of records in the test dataset is', X_test.shape[0])
       print(f"The training dataset has {class_counts[majority_class]} records for the majority class"
             f"and {class_counts[minority_class]} records for the minority class.")
```

```
The number of records in the training dataset is 80000
The number of records in the test dataset is 20000
The training dataset has 79183 records for the majority classand 817 records for the minority class.
```

Figure 4

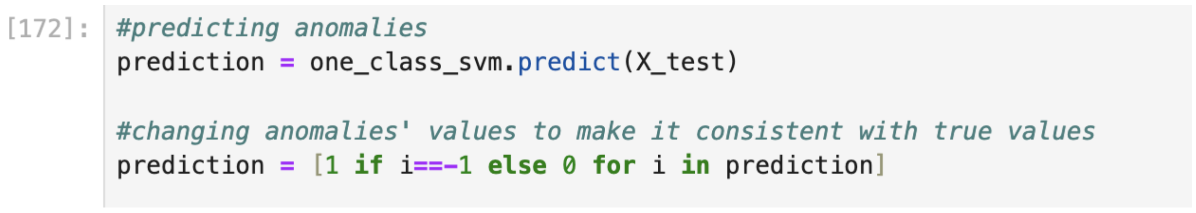
4. **Training the model using a kernel function:** It enables the model to determine the boundary between the normal and anomalous data by capturing non-linear patterns of normal behaviour.



```
[171]: #training
       one_class_svm = OneClassSVM(nu=0.01, kernel = 'rbf', gamma = 'auto').fit(X_train)
```

Figure 5

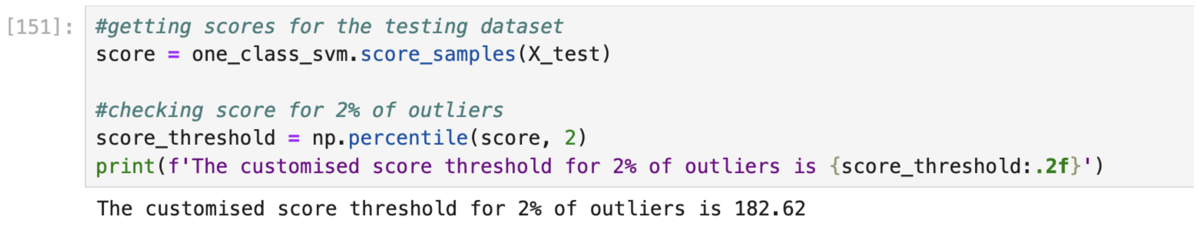
5. **Predicting**



```
[172]: #predicting anomalies
       prediction = one_class_svm.predict(X_test)

       #changing anomalies' values to make it consistent with true values
       prediction = [1 if i==-1 else 0 for i in prediction]
```

Figure 6

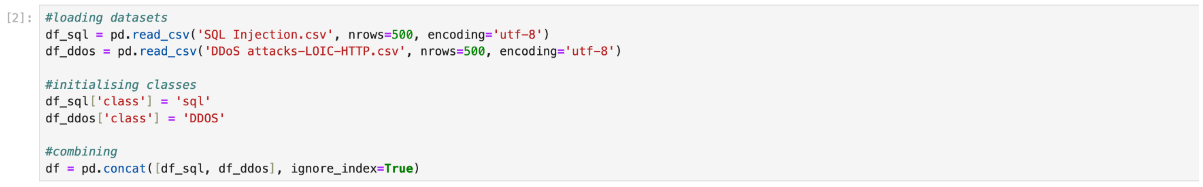
6. **Identifying outliers:** This is to reduce any outliers that may negatively skew the performance and accuracy of the model as it heavily influences calculations.



```
[151]: #getting scores for the testing dataset
       score = one_class_svm.score_samples(X_test)

       #checking score for 2% of outliers
       score_threshold = np.percentile(score, 2)
       print(f'The customised score threshold for 2% of outliers is {score_threshold:.2f}')

       The customised score threshold for 2% of outliers is 182.62
```

Figure 7

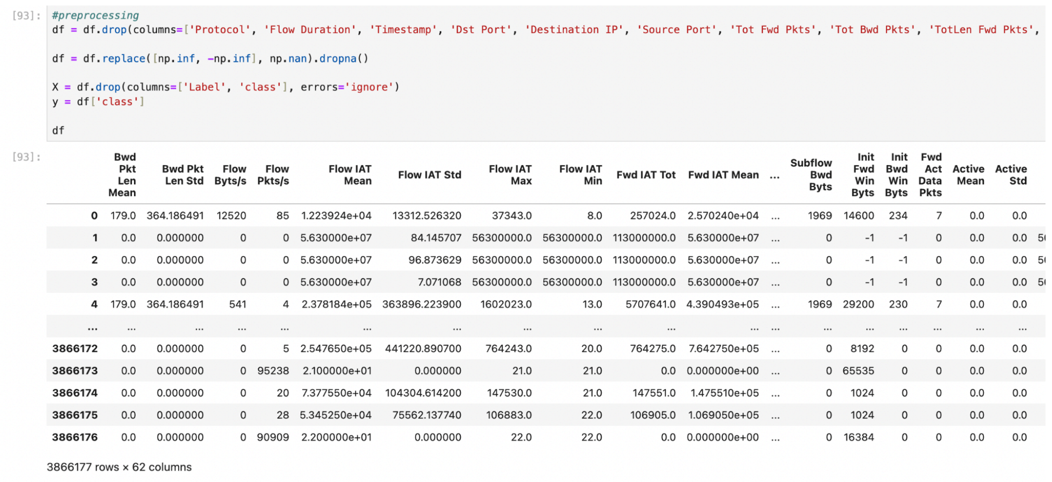
7. **Loading datasets:** Once the simulation is complete, specific datasets are used such as "SQL Injection.csv" and "DDoS attacks-LOIC-HTTP.csv" which are part of the large CSE-CIC-IDS2018 dataset that samples network traffic with embedded normal and attack data.



```
[2]: #loading datasets
     df_sql = pd.read_csv('SQL Injection.csv', nrows=500, encoding='utf-8')
     df_ddos = pd.read_csv('DDoS attacks-LOIC-HTTP.csv', nrows=500, encoding='utf-8')

     #initialising classes
     df_sql['class'] = 'sql'
     df_ddos['class'] = 'DDOS'

     #combining
     df = pd.concat([df_sql, df_ddos], ignore_index=True)
```

Figure 8

8. **Preprocessing:** By dropping any unnecessary columns to properly extract important features from the dataset



```
[93]: #preprocessing
      df = df.drop(columns=['Protocol', 'Flow Duration', 'Timestamp', 'Dst Port', 'Destination IP', 'Source Port', 'Tot Fwd Pkts', 'Tot Bwd Pkts', 'TotLen Fwd Pkts',

      df = df.replace([np.inf, -np.inf], np.nan).dropna()

      X = df.drop(columns=['Label', 'class'], errors='ignore')
      y = df['class']

      df
```

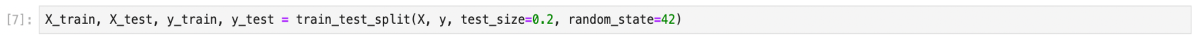
[93]:

	Bwd Pkt Len Mean	Bwd Pkt Len Std	Flow Byts/s	Flow Pkts/s	Flow IAT Mean	Flow IAT Std	Flow IAT Max	Flow IAT Min	Fwd IAT Tot	Fwd IAT Mean	...	Subflow Bwd Byts	Init Fwd Win Byts	Init Bwd Win Byts	Fwd Act Data Pkts	Active Mean	Active Std
0	179.0	364.186491	12520	85	1.223924e+04	13312.526320	37343.0	8.0	257024.0	2.570240e+04	...	1969	14600	234	7	0.0	0.0
1	0.0	0.000000	0	0	5.630000e+07	84.145707	56300000.0	56300000.0	113000000.0	5.630000e+07	...	0	-1	-1	0	0.0	0.0
2	0.0	0.000000	0	0	5.630000e+07	96.873629	56300000.0	56300000.0	113000000.0	5.630000e+07	...	0	-1	-1	0	0.0	0.0
3	0.0	0.000000	0	0	5.630000e+07	7.071068	56300000.0	56300000.0	113000000.0	5.630000e+07	...	0	-1	-1	0	0.0	0.0
4	179.0	364.186491	541	4	2.378184e+05	363896.223900	1602023.0	13.0	5707641.0	4.390493e+05	...	1969	29200	230	7	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3866172	0.0	0.000000	0	5	2.547650e+05	441220.890700	764243.0	20.0	764275.0	7.642750e+05	...	0	8192	0	0	0.0	0.0
3866173	0.0	0.000000	0	95238	2.100000e+01	0.000000	21.0	21.0	0.0	0.000000e+00	...	0	65535	0	0	0.0	0.0
3866174	0.0	0.000000	0	20	7.377550e+04	104304.614200	147530.0	21.0	147551.0	1.475510e+05	...	0	1024	0	0	0.0	0.0
3866175	0.0	0.000000	0	28	5.345250e+04	75562.137740	106883.0	22.0	106905.0	1.069050e+05	...	0	1024	0	0	0.0	0.0
3866176	0.0	0.000000	0	90909	2.200000e+01	0.000000	22.0	22.0	0.0	0.000000e+00	...	0	16384	0	0	0.0	0.0

3866177 rows × 62 columns

Figure 9

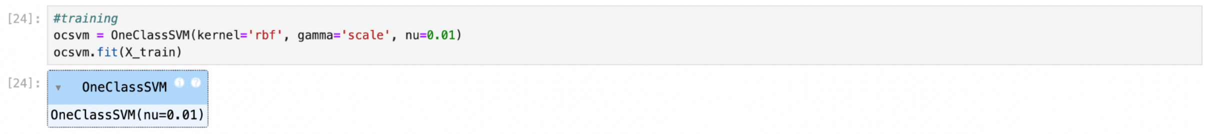
9. **Splitting the dataset:** 80% for the training data and 20% for the testing data.



```
[7]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 10

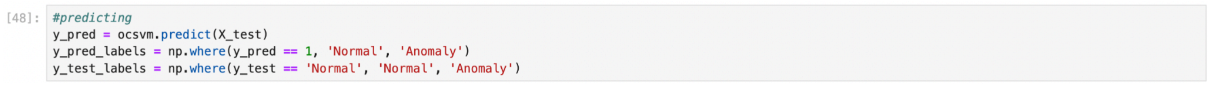
10. **Training and fitting the model**



```
[24]: #training
      ocsvm = OneClassSVM(kernel='rbf', gamma='scale', nu=0.01)
      ocsvm.fit(X_train)

[24]: ▾ OneClassSVM
      OneClassSVM(nu=0.01)
```

Figure 11

11. **Making predictions:** 1 is classed as “normal” and -1 is classed as an anomaly

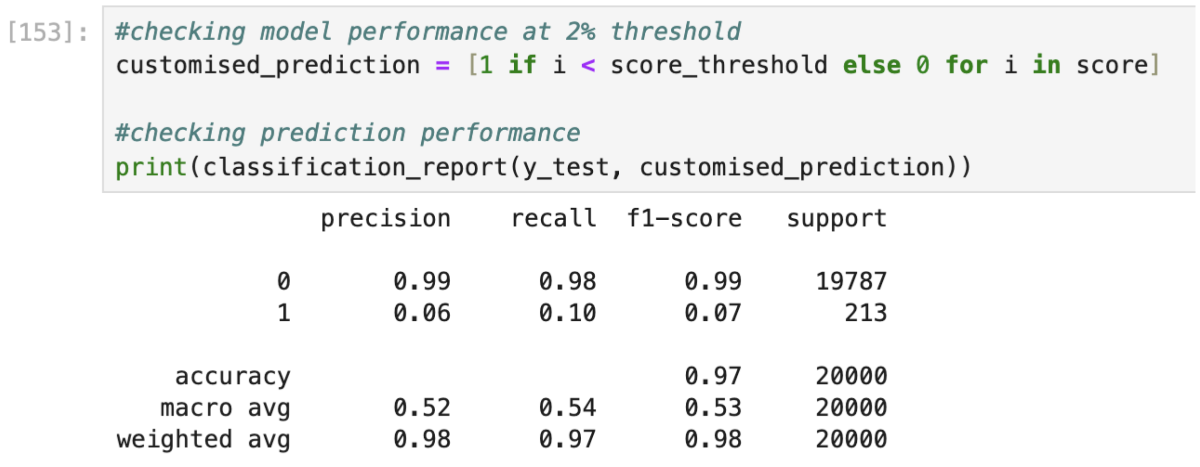


```
[48]: #predicting
      y_pred = ocsvm.predict(X_test)
      y_pred_labels = np.where(y_pred == 1, 'Normal', 'Anomaly')
      y_test_labels = np.where(y_test == 'Normal', 'Normal', 'Anomaly')
```

Figure 12

1.3 OCSVM Metrics and results

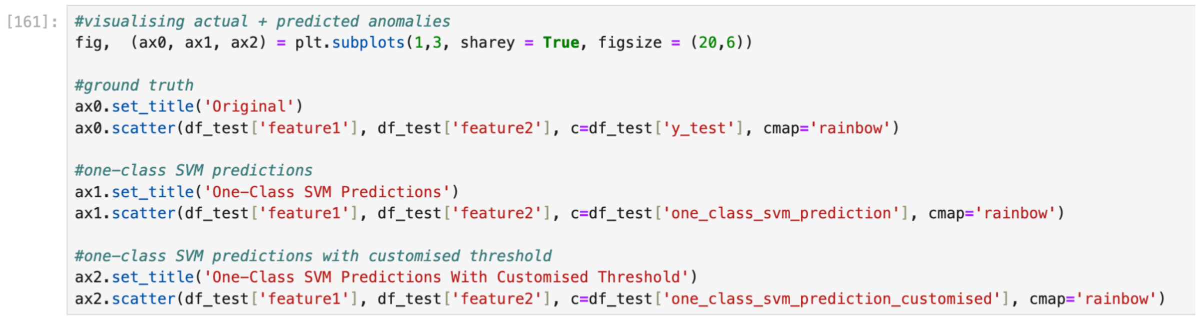
For the first simulation where a synthetic imbalanced dataset was utilised to estimate the performance of a OCSVM model:



```
[153]: #checking model performance at 2% threshold
       customised_prediction = [1 if i < score_threshold else 0 for i in score]

       #checking prediction performance
       print(classification_report(y_test, customised_prediction))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99     19787
           1       0.06      0.10      0.07       213

    accuracy                           0.97     20000
   macro avg       0.52      0.54      0.53     20000
weighted avg       0.98      0.97      0.98     20000
```

Figure 13



```
[161]: #visualising actual + predicted anomalies
       fig,  (ax0, ax1, ax2) = plt.subplots(1,3, sharey = True, figsize = (20,6))

       #ground truth
       ax0.set_title('Original')
       ax0.scatter(df_test['feature1'], df_test['feature2'], c=df_test['y_test'], cmap='rainbow')

       #one-class SVM predictions
       ax1.set_title('One-Class SVM Predictions')
       ax1.scatter(df_test['feature1'], df_test['feature2'], c=df_test['one_class_svm_prediction'], cmap='rainbow')

       #one-class SVM predictions with customised threshold
       ax2.set_title('One-Class SVM Predictions With Customised Threshold')
       ax2.scatter(df_test['feature1'], df_test['feature2'], c=df_test['one_class_svm_prediction_customised'], cmap='rainbow')
```

Figure 14

```
[161]: <matplotlib.collections.PathCollection at 0x179bf99d0>
```

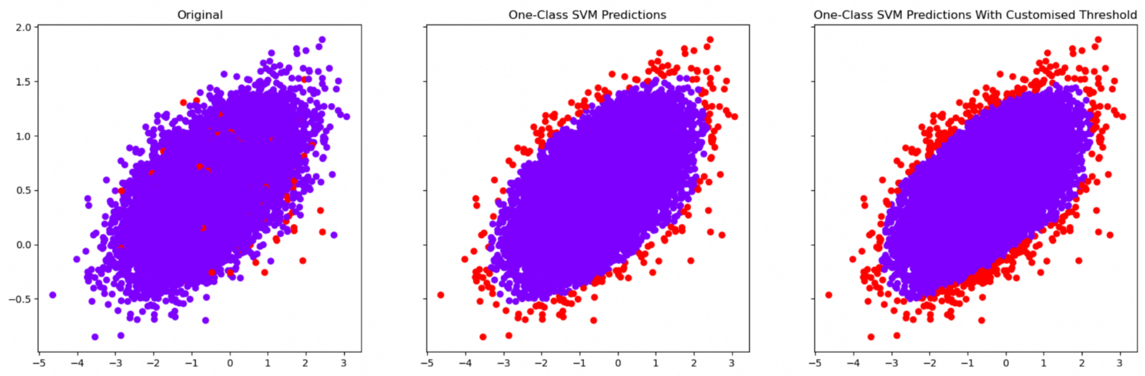
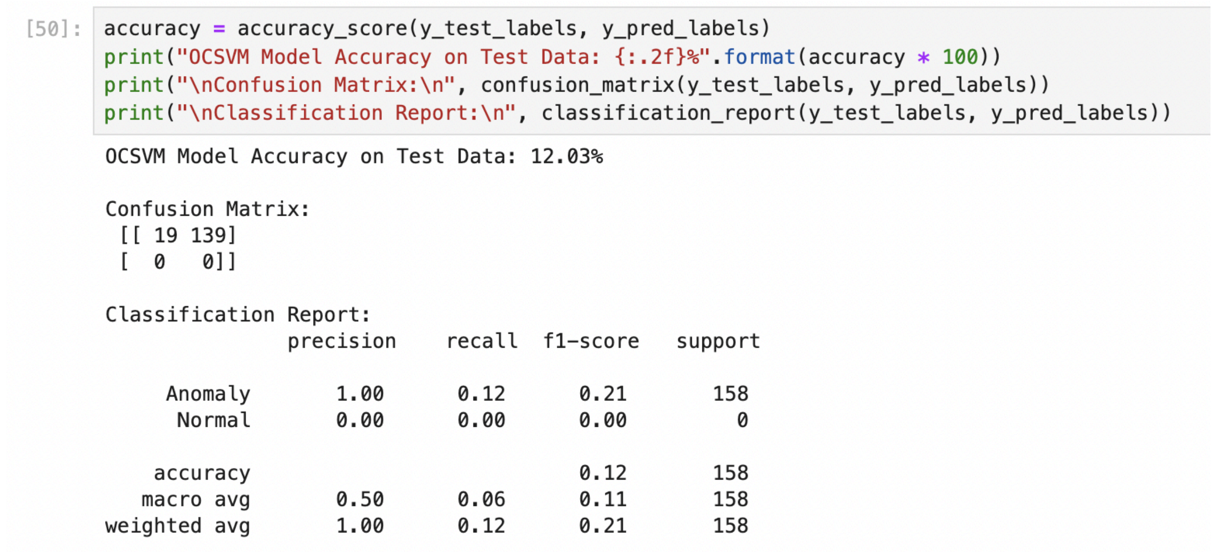


Figure 15

These results have demonstrated how the OCSVM model has achieved an accuracy score of 97.56% due to its mostly successful separation of the anomalous and normal data using a decision boundary.

Furthermore, with the utilisation of the "SQL Injection.csv" and "DDoS attacks-LOIC-HTTP.csv" datasets, this model has gained an accuracy of 12.03%. This is due to the model failing to properly learn from the anomalous data as they are usually trained with only normal data. The size of the datasets were also adjusted into a smaller variation as the OCSVM model is unable to properly process and withstand large datasets, consequently affecting its performance accuracy due to the low number of data. This can also cause the model to simplify features and cause it to cluster huge amounts of data without properly identifying it.



```
[50]: accuracy = accuracy_score(y_test_labels, y_pred_labels)
      print("OCSVM Model Accuracy on Test Data: {:.2f}%".format(accuracy * 100))
      print("\nConfusion Matrix:\n", confusion_matrix(y_test_labels, y_pred_labels))
      print("\nClassification Report:\n", classification_report(y_test_labels, y_pred_labels))
```

```
OCSVM Model Accuracy on Test Data: 12.03%

Confusion Matrix:
 [[ 19 139]
 [  0   0]]

Classification Report:
               precision    recall  f1-score   support

     Anomaly       1.00      0.12      0.21       158
      Normal       0.00      0.00      0.00         0

    accuracy                           0.12       158
   macro avg       0.50      0.06      0.11       158
weighted avg       1.00      0.12      0.21       158
```

Figure 16

It was observed how the OCSVM faced numerous challenges regarding the processing of a large dataset due to the immense amount of memory it requires. This affected model performance and accuracy, highlighting how it is only useful for small scale datasets which hinders their functional use in network environments that contain numerous amounts of data.

1.4 References

1. Amer, M., Goldstein, M., & Abdennadher, S. (2013). Enhancing one-class support vector machines for unsupervised anomaly detection. Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description - ODD '13. https://doi.org/10.1145/2500853.2500857
2. Chen, Y., Qian, J., & Venkatesh Saligrama. (2013). A new one-class SVM for anomaly detection. ArXiv (Cornell University). https://doi.org/10.1109/icassp.2013.6638322 Designing of different kernels in Machine Learning and Deep Learning. (2022, March 24). Www.turing.com. https://www.turing.com/kb/designing-of-different-kernels-in-machine-learning-deep-learning
3. Erfani, S., Rajasegarar, S., Karunasekera, S., & Leckie, C. (2016). High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning. Pattern Recognition, 58, 121–134. https://doi.org/10.1016/j.patcog.2016.03.028
4. Google Cloud. (2024). What is unsupervised learning? Google Cloud. https://cloud.google.com/discover/what-is-unsupervised-learning
5. Hejazi, M., & Singh, Y. P. (2013). ONE-CLASS SUPPORT VECTOR MACHINES APPROACH TO ANOMALY DETECTION. Applied Artificial Intelligence, 27(5), 351–366. https://doi.org/10.1080/08839514.2013.785791
6.
7. Mohamed, A. (2024). CSE-CIC-IDS2018. Mendeley Data, 1. https://doi.org/10.17632/29hdbdzx2r.1
8. Palo Alto. (2015). What is an Intrusion Detection System? Palo Alto Networks. https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-detection-system-ids
9. Panigrahi, R., & Borah, S. (2019). Classification and Analysis of Facebook Metrics Dataset Using Supervised Classifiers. Social Network Analytics, 1–19. https://doi.org/10.1016/b978-0-12-815458-8.00001-3
10. https://www.analyticsvidhya.com/blog/2024/03/one-class-svm-for-anomaly-detection/
11. Wang, C., Sun, Y., Sicai Lv, Wang, C., Liu, H., & Wang, B. (2023). Intrusion Detection System Based on One-Class Support Vector Machine and Gaussian Mixture Model. Electronics, 12(4), 930–930. https://doi.org/10.3390/electronics12040930