**Project was to detect ddos attack using machine learning algorithm. Now write psedocode algorithm of bellow mentioned random forest code. a sample image is give for the referense.. you can follow the writing style**

**rf\_model = RandomForestClassifier(n\_estimators=50, random\_state=42)**

**rf\_model.fit(X\_train, y\_train)**

**rf\_pred = rf\_model.predict(X\_test)**

**rf\_accuracy = accuracy\_score(y\_test, rf\_pred)**

**rf\_f1 = f1\_score(y\_test, rf\_pred)**

**rf\_precision = precision\_score(y\_test, rf\_pred)**

**rf\_recall = recall\_score(y\_test, rf\_pred)**

**print('\nRandom Forest Metrics:')**

**print(f'Accuracy: {rf\_accuracy:.4f}')**

**print(f'F1 Score: {rf\_f1:.4f}')**

**print(f'Precision: {rf\_precision:.4f}')**

**print(f'Recall: {rf\_recall:.4f}')**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

|  |
| --- |
|  |
| 1: function DETECT\_DDOS\_ATTACK\_LOGISTIC\_REGRESSION(X\_train, y\_train, X\_test, y\_test)  2: Input: X\_train, y\_train, X\_test, y\_test  3: Output: result  4:  5: lr\_model ← LogisticRegression(random\_state=42)  6: lr\_model.fit(X\_train, y\_train)  7: lr\_pred ← lr\_model.predict(X\_test)  8:  9: lr\_accuracy ← accuracy\_score(y\_test, lr\_pred)  10: lr\_f1 ← f1\_score(y\_test, lr\_pred)  11: lr\_precision ← precision\_score(y\_test, lr\_pred)  12: lr\_recall ← recall\_score(y\_test, lr\_pred)  13:  14: result ← (lr\_accuracy, lr\_f1, lr\_precision, lr\_recall)  15:  16: return result  17: end function |

|  |
| --- |
|  |
| 1: function DETECT\_DDOS\_ATTACK(X\_train, y\_train, X\_test, y\_test)  2: Input: X\_train, y\_train, X\_test, y\_test  3: Output: result  4:  5: rf\_model ← RandomForestClassifier(n\_estimators=50, random\_state=42)  6: rf\_model.fit(X\_train, y\_train)  7: rf\_pred ← rf\_model.predict(X\_test)  8:  9: rf\_accuracy ← accuracy\_score(y\_test, rf\_pred)  10: rf\_f1 ← f1\_score(y\_test, rf\_pred)  11: rf\_precision ← precision\_score(y\_test, rf\_pred)  12: rf\_recall ← recall\_score(y\_test, rf\_pred)  13:  14: result ← (rf\_accuracy, rf\_f1, rf\_precision, rf\_recall)  15:  16: return result  17: end function |

|  |
| --- |
|  |
| 1: function DETECT\_DDOS\_ATTACK\_WITH\_KNN(X\_train, y\_train, X\_test, y\_test)  2: Input: X\_train, y\_train, X\_test, y\_test  3: Output: result  4:  5: knn\_model ← KNeighborsClassifier(n\_neighbors=5)  6: knn\_model.fit(X\_train, y\_train)  7: knn\_pred ← knn\_model.predict(X\_test)  8:  9: knn\_accuracy ← accuracy\_score(y\_test, knn\_pred)  10: knn\_f1 ← f1\_score(y\_test, knn\_pred)  11: knn\_precision ← precision\_score(y\_test, knn\_pred)  12: knn\_recall ← recall\_score(y\_test, knn\_pred)  13:  14: result ← (knn\_accuracy, knn\_f1, knn\_precision, knn\_recall)  15:  16: return result  17: end function |

|  |
| --- |
|  |
| 1: function DETECT\_DDOS\_ATTACK\_USING\_SVM(X\_train, y\_train, X\_test, y\_test)  2: Input: X\_train, y\_train, X\_test, y\_test  3: Output: result  4:  5: svm\_model ← SVC(probability=True, random\_state=42)  6: svm\_model.fit(X\_train, y\_train)  7: svm\_pred ← svm\_model.predict(X\_test)  8:  9: svm\_accuracy ← accuracy\_score(y\_test, svm\_pred)  10: svm\_f1 ← f1\_score(y\_test, svm\_pred)  11: svm\_precision ← precision\_score(y\_test, svm\_pred)  12: svm\_recall ← recall\_score(y\_test, svm\_pred)  13:  14: result ← (svm\_accuracy, svm\_f1, svm\_precision, svm\_recall)  15:  16: return result  17: end function |

|  |
| --- |
|  |
| 1: function DETECT\_DDOS\_WITH\_DECISION\_TREE(X\_train, y\_train, X\_test, y\_test)  2: Input: X\_train, y\_train, X\_test, y\_test  3: Output: result  4:  5: dt\_model ← DecisionTreeClassifier(random\_state=42)  6: dt\_model.fit(X\_train, y\_train)  7: dt\_pred ← dt\_model.predict(X\_test)  8:  9: dt\_accuracy ← accuracy\_score(y\_test, dt\_pred)  10: dt\_f1 ← f1\_score(y\_test, dt\_pred)  11: dt\_precision ← precision\_score(y\_test, dt\_pred)  12: dt\_recall ← recall\_score(y\_test, dt\_pred)  13:  14: result ← (dt\_accuracy, dt\_f1, dt\_precision, dt\_recall)  15:  16: return result  17: end function |

|  |
| --- |
|  |
| 1: function DETECT\_DDOS\_ATTACK\_WITH\_STACKING(X, y)  2: Input: X, y  3: Output: result  4:  7: gb\_model ← GradientBoostingClassifier(random\_state=42)  8: xgb\_model ← XGBClassifier(random\_state=42)  9:  10: meta\_model ← LogisticRegression()  11:  12: stacked\_model ← StackingClassifier(  13: estimators=[('gb', gb\_model), ('xgb', xgb\_model)],  14: final\_estimator=meta\_model,  15: cv=5  16: )  17:  18: stacked\_model.fit(X\_train, y\_train)  19:  20: stacked\_pred ← stacked\_model.predict(X\_test)  21:  22: stacked\_accuracy ← accuracy\_score(y\_test, stacked\_pred)  23: stacked\_f1 ← f1\_score(y\_test, stacked\_pred)  24: stacked\_precision ← precision\_score(y\_test, stacked\_pred)  25: stacked\_recall ← recall\_score(y\_test, stacked\_pred)  26:  27: result ← (stacked\_accuracy, stacked\_f1, stacked\_precision, stacked\_recall)  28:  29: return result  30: end function |

|  |
| --- |
|  |
| 1: function DETECT\_DDOS\_ATTACK\_WITH\_NN(X\_train, y\_train, X\_test, y\_test)  2: Input: X\_train, y\_train, X\_test, y\_test  3: Output: result  4:  5: nn\_model ← MLPClassifier(hidden\_layer\_sizes=(10,), max\_iter=10, random\_state=42)  6: nn\_model.fit(X\_train, y\_train)  7: nn\_pred ← nn\_model.predict(X\_test)  8:  9: nn\_accuracy ← accuracy\_score(y\_test, nn\_pred)  10: nn\_f1 ← f1\_score(y\_test, nn\_pred)  11: nn\_precision ← precision\_score(y\_test, nn\_pred)  12: nn\_recall ← recall\_score(y\_test, nn\_pred)  13:  14: result ← (nn\_accuracy, nn\_f1, nn\_precision, nn\_recall)  15:  16: return result  17: end function |

Detailed Description Table

**Detailed Description Table**

|  |  |  |
| --- | --- | --- |
| **No** | **Name** | **Description** |
| 1 | Init\_Win\_bytes\_forward | Initial window size in bytes for the forward direction. |
| 2 | act\_data\_pkt\_fwd | Number of actual data packets sent in the forward direction. |
| 3 | Subflow Fwd Bytes | Number of bytes in the forward direction for a subflow. |
| 4 | Avg Fwd Segment Size | Average size of the forward segments. |
| 5 | Fwd Packet Length Max | Maximum length of packets in the forward direction. |
| 6 | Bwd Packet Length Max | Maximum length of packets in the backward direction. |
| 7 | Fwd IAT Std | Standard deviation of the inter-arrival time between packets in the forward direction. |
| 8 | Fwd Packet Length Mean | Mean length of packets in the forward direction. |
| 9 | Total Length of Fwd Packets | Total length of all packets in the forward direction. |
| 10 | Fwd IAT Mean | Mean inter-arrival time between packets in the forward direction. |
| 11 | Subflow Fwd Packets | Number of packets in the forward direction for a subflow. |
| 12 | Fwd Packet Length Std | Standard deviation of the packet lengths in the forward direction. |
| 13 | Fwd IAT Total | Total inter-arrival time between packets in the forward direction. |
| 14 | Fwd Header Length | Header length of packets in the forward direction. |
| 15 | Packet Length Variance | Variance in the length of packets. |
| 16 | Average Packet Size | Average size of packets. |
| 17 | Bwd Header Length | Header length of packets in the backward direction. |
| 18 | Total Length of Bwd Packets | Total length of all packets in the backward direction. |
| 19 | Avg Bwd Segment Size | Average size of the backward segments. |
| 20 | Max Packet Length | Maximum length of packets in the flow. |
| 21 | Fwd IAT Max | Maximum inter-arrival time between packets in the forward direction. |
| 22 | Destination Port | Port number of the destination. |
| 23 | Flow IAT Std | Standard deviation of the inter-arrival time between packets in the flow. |
| 24 | Subflow Bwd Bytes | Number of bytes in the backward direction for a subflow. |
| 25 | Fwd Header Length.1 | Duplicate column for forward header length. |
| 26 | Bwd Packet Length Mean | Mean length of packets in the backward direction. |
| 27 | Init\_Win\_bytes\_backward | Initial window size in bytes for the backward direction. |
| 28 | Bwd Packet Length Std | Standard deviation of the packet lengths in the backward direction. |
| 29 | Bwd Packets/s | Number of backward packets per second. |
| 30 | Bwd Packet Length Min | Minimum length of packets in the backward direction. |
| 31 | Total Backward Packets | Total number of packets in the backward direction. |
| 32 | Packet Length Mean | Mean length of packets in the flow. |
| 33 | Subflow Bwd Packets | Number of packets in the backward direction for a subflow. |
| 34 | Down/Up Ratio | Ratio of downlink to uplink traffic. |
| 35 | Packet Length Std | Standard deviation of the packet lengths in the flow. |
| 36 | Flow Duration | Total duration of the flow. |
| 37 | Bwd IAT Max | Maximum inter-arrival time between packets in the backward direction. |
| 38 | Bwd IAT Total | Total inter-arrival time between packets in the backward direction. |
| 39 | Fwd IAT Min | Minimum inter-arrival time between packets in the forward direction. |
| 40 | Bwd IAT Std | Standard deviation of the inter-arrival time between packets in the backward direction. |
| 41 | min\_seg\_size\_forward | Minimum segment size observed in the forward direction. |
| 42 | Flow IAT Max | Maximum inter-arrival time between packets in the flow. |
| 43 | Fwd Packets/s | Number of forward packets per second. |
| 44 | Min Packet Length | Minimum length of packets in the flow. |
| 45 | Fwd Packet Length Min | Minimum length of packets in the forward direction. |

**Overview of the Dataset**

The CIC-IDS-2017 dataset, provided by the Canadian Institute for Cybersecurity at the University of New Brunswick, is a comprehensive collection of network traffic data designed for the evaluation of intrusion detection systems (IDS). This dataset encompasses a variety of attack scenarios, including Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks, alongside normal network traffic, capturing the behavior of 12 different machines under realistic conditions. The dataset features more than 80 network flow characteristics, such as packet lengths, inter-arrival times, and header information, meticulously extracted using the CICFlowMeter tool. It spans five days of data collection, with detailed logs of benign activities and attack instances, thus providing a robust foundation for developing and benchmarking IDS algorithms. The dataset's completeness, including labeled data and a mix of network protocols, ensures its utility for diverse research applications in cybersecurity.