Configuration Manual

MSc Practicum Part 2
MSc in Cybersecurity

Anusha Palakkattu East Madom Ramadas
Student ID: 23124903

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Anusha Palakkattu East Madom Ramadas
Student ID:	23124903
Programme:	Master of Science in Cybersecurity **Year:** 2024
Module:	MSc Practicum part 2
Lecturer:	Vikas Sahni
Submission Due Date:	12/12/2024
Project Title:	Dynamic intrusion detection system for improved cloud security
Word Count:	971 **Page Count:** 12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Anusha Palakkattu East Madom Ramadas

Date: 09/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Anusha Palakkattu East Madom Ramadas
23124903

1 Introduction

This configuration manual contains primary setup and tools required to replicate the project. The project builds a model using extreme learning machine (ELM) algorithm for intrusion detection in cloud environments. The dataset utilised is CSE-CIC-IDS2018[1]. This manual includes all the software and hardware requirements, configurations and execution flows.

2 Hardware Requirements

The hardware utilised for the development of this project is as follows:

- Processor: 13th Gen Intel(R) Core (TM) i5-1340P 1.90 GHz, 12 Cores
- RAM: 16.0 GB
- Operating System: Microsoft Windows 11 Home
- GPU: Intel(R) Iris(R) Xe Graphics
- Storage: 512 GB SSD

3 Software Requirements

The software tools utilised for the development of this project is as follows:

- Anaconda 2.6.0
- Jupyter Notebook
- Python 3.12.4

Anaconda navigator[2] is an open-source platform which has user friendly interface to manage packages and environments. It comes with Jupyter Notebook and required python setup to run machine learning (ML) projects. Anaconda 64-bit latest stable version was installed and setup on the windows 11 machine.

Jupyter notebook is a web-based interactive platform which can be used to create and share computing documents like codes, interactive dashboards and equations. It is a good platform to use for ML projects. It can be launched through anaconda navigator or can be launched through anaconda command prompt as shown in Figure 2.

[1] https://registry.opendata.aws/cse-cic-ids2018/

[2] https://www.anaconda.com/download

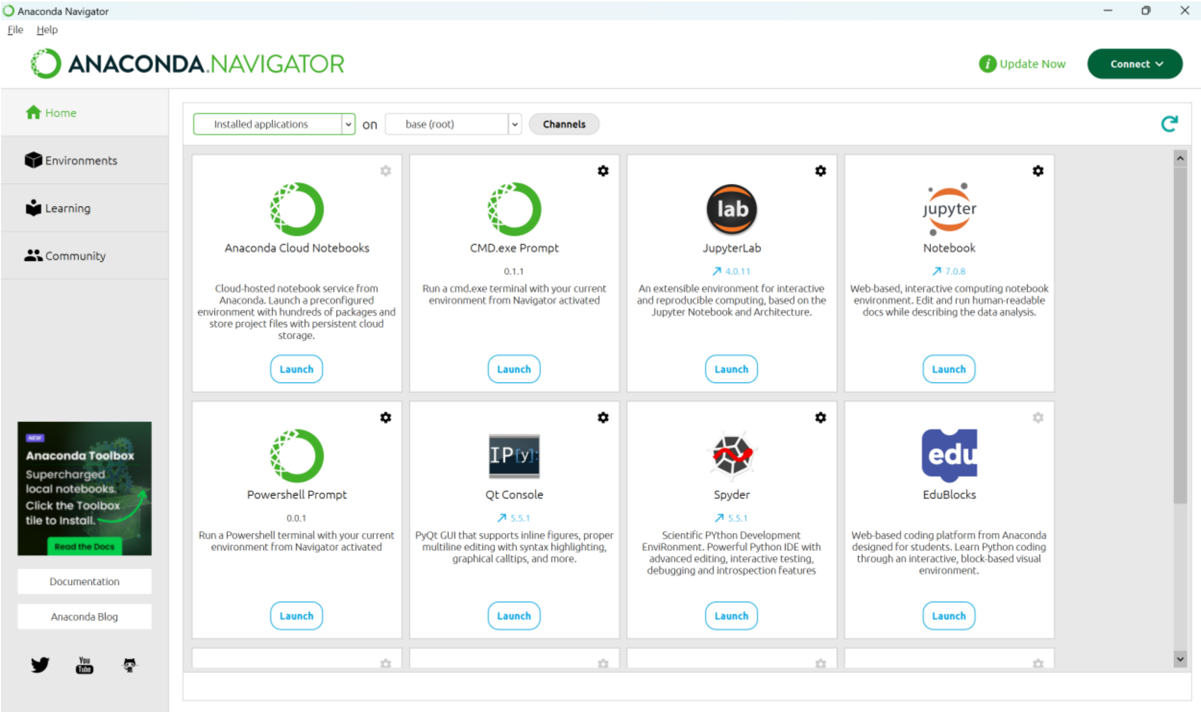

Figure 1: Anaconda navigator.

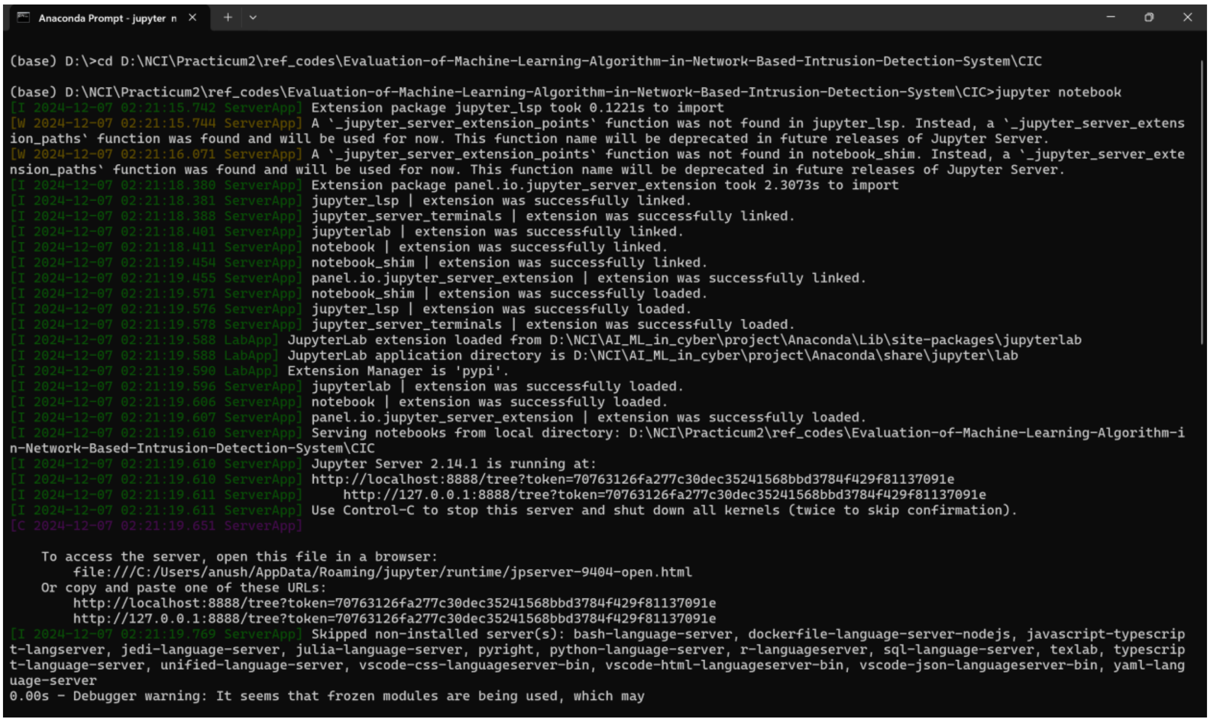

Figure 2: Jupyter notebook run through command line.

There were several python libraries utilised for building the project such as pandas, numpy, scikit-learn and matplotlib. To develop ELM model scikit-elm library was installed and used as shown in Figure 3.

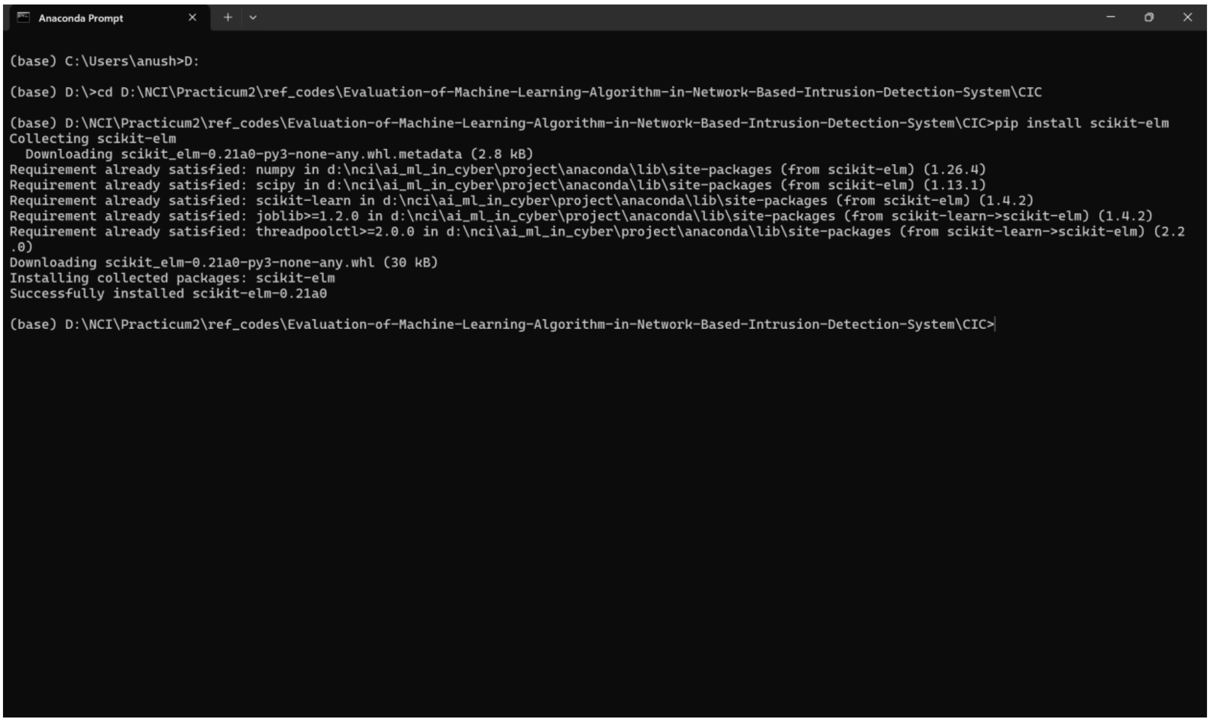

Figure 3: scikit-elm package installation.

4 Dataset Preparation

CSE-CIC-IDS2018 dataset was leveraged to develop the project because it has real time network traffic and different attack types. This dataset contains various cyberattack scenarios including DoS, DDoS, botnet, brute-force, web attacks, network infiltration and heartbleed. The dataset contains 80 extracted features of the real time network traffic captured from 420 machines and 30 servers. It can be downloaded using the aws command given on the website. Since python does not accept pcap files, csv files were used for processing. However, there is one file with 84 columns, those 4 columns were analysed and removed to maintain data consistency, as shown in Figure 4. The dataset was cleaned from null or missing values, infinite values, outliers and duplicates. Class imbalance was found in the dataset, as illustrated in Figure 5. Therefore, dataset was down sampled, and the result obtained is depicted in Figure 6. Furthermore, attack samples were relabelled to benign and malicious as shown in Figure 7.

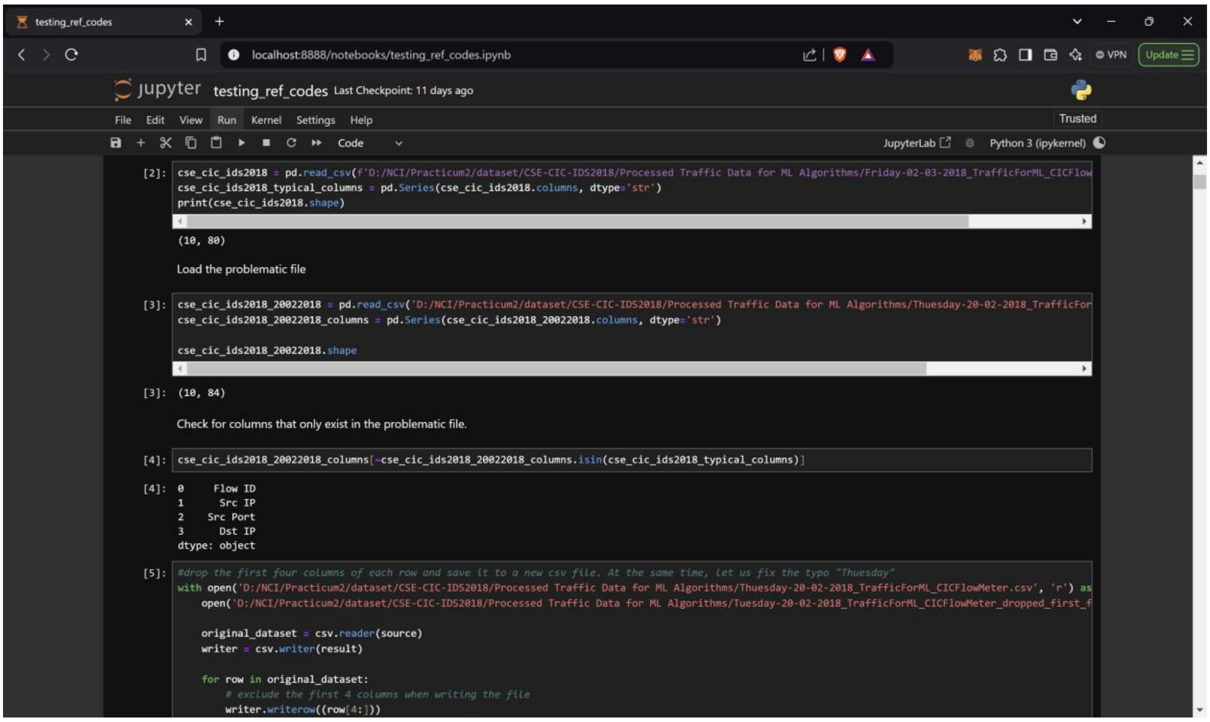

Figure 4: Analysing and removing extra 4 columns in one file.

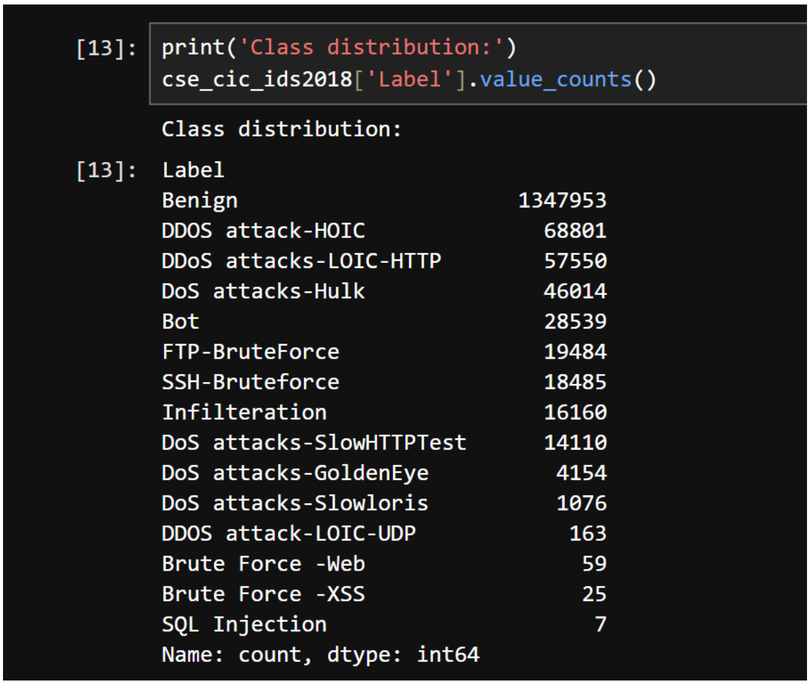

Figure 5: Class imbalance in data.

```
                sample_size = max_sample
            else:
                sample_size = count

            sample = dataset[dataset['Label'] == label].sample(n=
            dataset_downsampled = pd.concat([dataset_downsampled,

    return dataset_downsampled
```

```
[26]: ids2018_attack = downsample_dataset(cse_cic_ids2018, sample_c
      num_attack_sample = ids2018_attack.shape[0]
      ids2018_benign = cse_cic_ids2018[cse_cic_ids2018['Label'] ==
      ids2018_downsampled = pd.concat([ids2018_attack, ids2018_beni
      del ids2018_attack
      del ids2018_benign

      print('Distribution of class after downsampling')
      ids2018_downsampled['Label'].value_counts()

      Distribution of class after downsampling

[26]: Label
      Benign                      257791
      DDOS attack-HOIC             68628
      DDoS attacks-LOIC-HTTP       57550
      DoS attacks-Hulk             45691
      Bot                          28501
      SSH-Bruteforce               16312
      Infilteration                16034
      FTP-BruteForce               12368
      DoS attacks-SlowHTTPTest      7251
      DoS attacks-GoldenEye         4153
      DoS attacks-Slowloris         1049
      DDOS attack-LOIC-UDP           163
      Brute Force -Web                59
      Brute Force -XSS                25
      SQL Injection                    7
      Name: count, dtype: int64
```

Figure 6: Down sampling to reduce class imbalance.

```
[28]: # replace the label of all attack class to 'malicious'
      ids2018_downsampled.iloc[ids2018_downsampled['Label'] != 'Benign', -1] = 'malicious'
      ids2018_downsampled.iloc[ids2018_downsampled['Label'] == 'Benign', -1] = 'benign'
      ids2018_downsampled['Label'].value_counts()

[28]: Label
      malicious    257791
      benign       257791
      Name: count, dtype: int64

[29]: # drop the additional columns
      ids2018_downsampled = ids2018_downsampled.drop(['Protocol', 'Timestamp'], axis=1).copy()
      ids2018_downsampled.head()
```

[29]:

	Dst Port	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean	Fwd Pkt Len Std	...	Fwd Seg Size Min	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label
0	80	1693	2	0	0.0	0.0	0.0	0.0	0.0	0.0	...	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	malicious
1	80	2200	2	0	0.0	0.0	0.0	0.0	0.0	0.0	...	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	malicious
2	80	2200	2	0	0.0	0.0	0.0	0.0	0.0	0.0	...	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	malicious
3	80	5743	2	0	0.0	0.0	0.0	0.0	0.0	0.0	...	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	malicious
4	80	8655	2	0	0.0	0.0	0.0	0.0	0.0	0.0	...	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	malicious

5 rows × 78 columns

Figure 7: Relabelling of data.

5 Feature Selection

For implementing feature selection, a random forest (RF) classifier was utilised. Importance of each feature were analysed, and top 20 features were chosen according to importance score ranking as shown in Figure 8 and Figure 9. Additionally brute force method was implemented for feature reduction. This is illustrated in Figure 10.

```
[37]: # extract the importance score from the random forest classifier
score = np.round(rfc.feature_importances_, 3)
importance = pd.DataFrame({'feature': X_columns,
                           'importance': score})
importance = importance.sort_values('importance', ascending=False).set_index('feature')

print(f"Top 20 features: \n{importance[:20]}")

plt.rcParams['figure.figsize'] = (12, 4)
importance.plot.bar()
```

```
Top 20 features:
                   importance
feature
Init Fwd Win Byts       0.065
TotLen Fwd Pkts         0.059
Dst Port                0.055
Fwd Pkt Len Mean        0.050
Fwd Pkt Len Max         0.043
Subflow Fwd Byts        0.040
Fwd Header Len          0.035
Init Bwd Win Byts       0.033
Fwd Seg Size Min        0.029
Flow Pkts/s             0.028
Pkt Len Max             0.028
Fwd Seg Size Avg        0.027
Fwd IAT Tot             0.027
Fwd IAT Std             0.026
Fwd IAT Mean            0.025
Fwd IAT Max             0.025
Fwd Pkts/s              0.023
Flow IAT Mean           0.023
Flow Duration           0.022
Flow IAT Min            0.019
```

Figure 8: Top 20 important features.

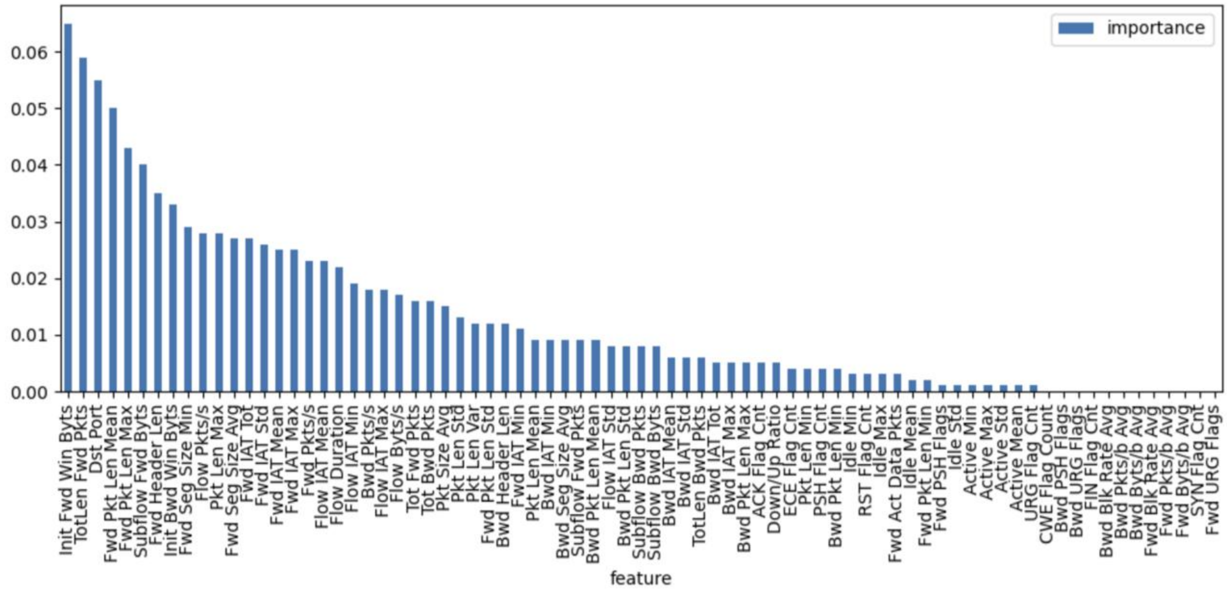

Figure 9: Feature importance ranking for all features.

Brute force feature reduction

```
[44]: columns = features.tolist() +  ['Label']

      ids2018 = ids2018[columns]
      ids2018.shape

[44]: (51558, 21)

[45]: ids2018_X = ids2018.drop('Label', axis=1).copy()
      ids2018_y = ids2018['Label'].copy()

      ids2018_train_X, ids2018_test_X, ids2018_train_y, ids2018_test_y = train_test_split(ids2018_X, ids2018_y, test_size=0.3)

[48]: # define ML models without optimized hyperparameter
      models = {
          'Decision Tree': tree.DecisionTreeClassifier(),
          'Random Forest': RandomForestClassifier(n_jobs=-1),
          'Support Vector Machine': SVC(),
          'Naive Bayes': GaussianNB(),
          'Artificial Neural Network': MLPClassifier(hidden_layer_sizes=(40), max_iter=500),
          'Deep Neural Network': MLPClassifier(hidden_layer_sizes=(12, 12, 12), max_iter=500),
          'Extreme Learning Machine': ELMClassifier(n_neurons=(12, 12, 12), ufunc='tanh', batch_size=1000),
      }

[49]: feature_set = []
      scalar = StandardScaler()
      scores = []

      for feature in features:
          feature_set.append(feature)
          print(f"Added feature {len(feature_set)} ({feature})...")\

          test_X = ids2018_test_X[feature_set]
          train_X = ids2018_train_X[feature_set]

          # scale the dataset
          train_X_scaled = scalar.fit(train_X)
          train_X_scaled = scalar.transform(train_X)
          test_X_scaled = scalar.transform(test_X)

          score_temp = [len(feature_set)]

          for model in models:
              clf = models[model]
              clf.fit(train_X_scaled, ids2018_train_y)
```

Figure 10: code snippet for feature reduction.

After selecting the final feature set, hyperparameter tuning for each model was performed using this final feature set using GridSearchCV. The code snippet and result for ELM is shown in Figure 11.

ELM

```
[85]: %%time

# Custom wrapper for ELMClassifier to define the missing `pairwise` tag
class CustomELMClassifier(ELMClassifier):
    def _get_tags(self):
        tags = super()._get_tags()
        # Add the missing pairwise tag
        tags['pairwise'] = False
        return tags

# Define the parameter space
parameter_space = {
    'n_neurons': [1000, 2000, 5000, 7000],
    'ufunc': ['tanh', 'relu', 'sigmoid', 'linear'],
    'alpha': [1e-7, 1e-5, 1e-3, 1e-1]
}

# Initialize the CustomELMClassifier
elm = CustomELMClassifier()

optimal_elm = GridSearchCV(
                        elm,
                        parameter_space,
                        cv=5,
                        n_jobs=-1,
                        verbose=0
)

# Perform grid search to find the best hyperparameters
optimal_elm.fit(ids2018_train_X_scaled, ids2018_train_y)

# Get the best parameters from GridSearchCV
elm_optimal_params = optimal_elm.best_params_
print(f"Optimum hyperparameters for ELM: \n{elm_optimal_params}")

Optimum hyperparameters for ELM:
{'alpha': 1e-05, 'n_neurons': 7000, 'ufunc': 'relu'}
CPU times: total: 1min 49s
Wall time: 10min 34s
```

Figure 11: Code snippet and result for hyperparameter tuning of ELM.

6 Classification

The ELM classification model was developed using the optimised hyperparameters. Five-fold cross validation of each model was performed using StratifiedKFold. Code snippet of cross validation of ELM and other models is shown in Figure 12.

```
[18]: # Custom wrapper for ELMClassifier to define the missing `pairwise` tag
class CustomELMClassifier(ELMClassifier):
    def _get_tags(self):
        tags = super()._get_tags()
        # Add the missing pairwise tag
        tags['pairwise'] = False
        return tags

[19]: models = {
    'Decision Tree': tree.DecisionTreeClassifier(criterion='entropy', ccp_alpha=1.4401469385343852e-05),
    'Random Forest': RandomForestClassifier(max_depth=20, min_samples_leaf=0.00001, min_samples_split=0.00001, n_estimators=350, n_jobs=-1,criterion='gir
    'Naive Bayes': GaussianNB(var_smoothing=1.0),
    'Artificial Neural Network': MLPClassifier(hidden_layer_sizes=(50,), activation='tanh', alpha=0.0001, solver='adam', max_iter=1000),
    'Deep Neural Network': MLPClassifier(hidden_layer_sizes=(15, 15, 15), activation='tanh', alpha=1e-05, solver='adam', max_iter=1000),
    'Extreme Learning Machine': CustomELMClassifier(n_neurons=7000, ufunc='relu', alpha=1e-05, batch_size=1000)
}

[20]: accuracy_scores = {}
accuracy_scores_mean = {}
accuracy_scores_std = {}

cv = StratifiedKFold(n_splits=5, shuffle=True)

for model in models:
    clf = models[model]

    accuracy_scores[model] = cross_val_score(clf,
                                             ids2018_X_scaled,
                                             ids2018_y,
                                             cv=cv,
                                             scoring='accuracy',
                                             n_jobs=-1)
    accuracy_scores_mean[model] = np.mean(accuracy_scores[model])
    accuracy_scores_std[model] = np.std(accuracy_scores[model])

    print(f"{'-'*25} {model} {'-'*25}")
    print(f"Accuracy: {accuracy_scores[model]}")
    print(f"mean: {accuracy_scores_mean[model]:.4f}\t\tstd: {accuracy_scores_std[model]:.4f}")
```

Figure 12: Code snippet of model cross validation.

```
------------------------- Decision Tree -------------------------
Accuracy: [0.96065945 0.96030386 0.95700663 0.95581057 0.95923711]
mean: 0.9586            std: 0.0019
------------------------- Random Forest -------------------------
Accuracy: [0.97297559 0.97249071 0.97287862 0.97294327 0.97407467]
mean: 0.9731            std: 0.0005
------------------------- Naive Bayes -------------------------
Accuracy: [0.65201228 0.65240019 0.65133344 0.65602069 0.65530952]
mean: 0.6534            std: 0.0019
------------------------- Artificial Neural Network -------------------------
Accuracy: [0.94397931 0.9437207  0.94362373 0.94504606 0.94388233]
mean: 0.9441            std: 0.0005
------------------------- Deep Neural Network -------------------------
Accuracy: [0.95445289 0.95532568 0.96764183 0.96938743 0.95345078]
mean: 0.9601            std: 0.0070
------------------------- Extreme Learning Machine -------------------------
Accuracy: [0.96725392 0.96728625 0.96877323 0.96796509 0.96670438]
mean: 0.9676            std: 0.0007
```

Figure 13: Cross validation results.

7 Evaluation

The built ELM model was evaluated based on accuracy, precision, recall and f1-score. Furthermore, the results of accuracy, precision, recall, f1-score and time consumption for prediction of ELM model was compared with other ML models. Confusion matrix was plotted. Code snippet of model building and evaluation is shown in Figure 14.

Model building

[10]:
```python
# Custom wrapper for ELMClassifier to define the missing `pairwise` tag
class CustomELMClassifier(ELMClassifier):
    def _get_tags(self):
        tags = super()._get_tags()
        # Add the missing pairwise tag
        tags['pairwise'] = False
        return tags
```

[11]:
```python
models = {
    'Decision Tree': tree.DecisionTreeClassifier(criterion='entropy', ccp_alpha=1.4401469385343852e-05),
    'Random Forest': RandomForestClassifier(max_depth=20, min_samples_leaf=0.00001, min_samples_split=0.00001, n_estimators=350, n_jobs=-1,criterion='
    'Naive Bayes': GaussianNB(var_smoothing=1.0),
    'Artificial Neural Network': MLPClassifier(hidden_layer_sizes=(50,), activation='tanh', alpha=0.0001, solver='adam', max_iter=1000),
    'Deep Neural Network': MLPClassifier(hidden_layer_sizes=(15, 15, 15), activation='tanh', alpha=1e-05, solver='adam', max_iter=1000),
    'Extreme Learning Machine': CustomELMClassifier(n_neurons=7000, ufunc='relu', alpha=1e-05, batch_size=1000)
}
```

[35]:
```python
trained_models = {}
prediction_time = {}
prediction_memory_usage = {}
accuracy_testing_dataset = {}
f_score_testing_dataset = {}

plt.rcParams.update({'font.size': 18})
fig, axes = plt.subplots(2, 3, figsize=(20, 10))

for i, (model, clf) in enumerate(models.items()):

    clf.fit(ids2018_train_X_scaled, ids2018_train_y)

    # save the trained model
    trained_models[model] = clf

    # Track memory and time during prediction
    start_memory_pred = memory_usage()[0]
    prediction_start_time =  time.time()
    prediction = clf.predict(ids2018_test_X_scaled)
    # save the time and memory consumption for prediction
    prediction_time[model] = time.time() - prediction_start_time
    # Adding a small delay after prediction to help memory capture
    time.sleep(0.1)
    prediction_memory_usage[model] = memory_usage()[0] - start_memory_pred

    model_report = metrics.classification_report(ids2018_test_y, prediction, digits=4, output_dict=True)

    # save the accuracy and the f1-score of each model
    accuracy_testing_dataset[model] = model_report['accuracy']
    f_score_testing_dataset[model] = model_report['weighted avg']['f1-score']

    print(f"{'-'*25} {model} {'-'*25}")
    print(metrics.classification_report(ids2018_test_y, prediction, digits=4))

    ConfusionMatrixDisplay.from_estimator(clf,
                            ids2018_test_X_scaled,
                            ids2018_test_y,
                            cmap=plt.cm.Blues,
                            ax=axes[math.floor(i/3)][i%3])

    axes[math.floor(i/3)][i%3].set_title(model)


fig.subplots_adjust(hspace=0.65, wspace=0.7)
fig.suptitle('Confusion matrix of each model on CIC-IDS2018 dataset', fontsize=24)
```

Figure 14: Model building and evaluation.