National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Shibi Murugesan
Student ID: X18170871

School of Computing
National College of Ireland

Supervisor: Dr.Pierpaolo Dondio

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Shibi Murugesan
Student ID:	X18170871
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr.Pierpaolo Dondio
Submission Due Date:	12/12/2019
Project Title:	Configuration Manual
Word Count:	512
Page Count:	11

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:	
Date:	12th December 2019

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

Shibi Murugesan
X18170871

1 Introduction

This configuration manual explains most of the system setup, software & hardware requirements and finally programming code required to implement the research project: *"Application of Machine Learning Models for Network Intrusion Detection Systems Based on Feature Selection Approach"*

2 Systems Configurations

2.1 Hardware Requirements

- **Processor:** Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
- **RAM:** 16.0 GB
- **System Type:** 64-bit Operating System, x64-based processor

2.2 Software Requirements

- **Pycharm Community Edition:** Much of the data aggregation, merging, cleaning, etc. was done in python.
- **Packages Required:** sklearn, yellowbrick, matplotlib, pandas and zipfile.

3 Dataset Preparation

The *NDL-KDD* dataset is downloaded from `https://www.unb.ca/cic/datasets/nsl.html`. The downloaded file would be zipped, hence we need to use *zipfile* python package to unzip it and convert the required set to the pandas dataframe format.

3.1 *NSL-KDD* unzip and converting to dataframe

3.1.1 Extraction

Extract the downloaded *NSL-KDD.zip* using *zipfile* python package in given location.

```
"""
Extract required dataset from zipfile
"""
import zipfile
import os
```

```
import pandas as pd

PATH_TO_ZIP_FILE = 'NSL-KDD.zip'
PATH_TO_ARFF_FILE = 'NSL-KDD/'
KDD_TRAIN_PATH = 'KDDTrain+.arff'
KDD_TEST_PATH = 'KDDTest+.arff'
IMAGE_DIR = 'images/'
LATEX_DIR = 'latex/'
CM_DIR = IMAGE_DIR + 'cm/'
ROC_DIR = IMAGE_DIR + 'roc/'
REPORT_DIR = IMAGE_DIR + 'report/'

def extractzipfile():
    """
    Unzip the zip file to a folder
    Also make images directory if not exist.
    """
    # Create the Image directory to save any plots
    if not os.path.exists(IMAGE_DIR):
        os.makedirs(IMAGE_DIR)

    # Extract the zipfile
    if os.path.exists(PATH_TO_ZIP_FILE):
        with zipfile.ZipFile(PATH_TO_ZIP_FILE, 'r') as zip_ref:
            zip_ref.extractall(PATH_TO_ARFF_FILE)
            return True

    # If ZipFile is not found
    return False
```

After unzipping, we might see below files in extracted folder

1. NSL-KDD/index.html
2. *NSL-KDD/KDDTest+.arff $->$ **We will use this as an Input to our algorithm**
3. NSL-KDD/KDDTest+.txt
4. NSL-KDD/KDDTest1.jpg
5. NSL-KDD/KDDTest-21.arff
6. NSL-KDD/KDDTest-21.txt
7. *NSL-KDD/KDDTrain+.arff $->$ **We will use this as an Input to our algorithm**
8. NSL-KDD/KDDTrain+.txt
9. NSL-KDD/KDDTrain+_20Percent.arff
10. NSL-KDD/KDDTrain+_20Percent.txt
11. NSL-KDD/KDDTrain1.jpg

3.1.2 Conversion to Dataframe

Convert the extracted **NSL-KDD/KDDTrain+.arff** and **NSL-KDD/KDDTest+.arff** into corresponding dataframes

```
def loaddataframe(filename, display=False):
    """
    Extract the required arff file into a dataframe.
    :return: <pd.Dataframe> Dataframe related with KDD data
    """

    dataset = pd.read_table(os.path.join(PATH_TO_ARFF_FILE, filename), sep=',',
                            index_col=False, header=None, names=COLUMNS)

    if display:
        print("Size of %s: " % filename)
        print("\tSamples Count: %d" % dataset.shape[0])
        print("\tAttributes Count: %d" % dataset.shape[1])
    return dataset

# Load the Dataset
traindata = extract.loaddataframe('KDDTrain+.txt')
testdata = extract.loaddataframe('KDDTest+.txt')
```

4 Dataset Pre-processing

Dataset Preprocessing follows below 3 steps:

1. Aggregating different attacks into 5 attack types
2. Label Encoding and Resampling minority Classes
3. Normalisation

4.1 Aggregating different attacks into 5 attack types

Here we will create a new column **class_group** which maps different attacks into different classes.

```
"""
Aggregate different types of attacks into different groups.
"""

def aggregateintogroups(value):
    """
    We will map different classes into 5 different groups.
    :return: <String> Return class group type
    """
    mapping = {'Probe': ['ipsweep', 'satan', 'nmap', 'portsweep', 'saint', 'mscan'],
               'DoS': ['teardrop', 'pod', 'land', 'back', 'neptune', 'smurf', 'mailbomb',
                       'udpstorm', 'apache2', 'processtable'],
               'U2R': ['perl', 'loadmodule', 'rootkit', 'buffer_overflow', 'xterm', 'ps',
                       'sqlattack', 'httptunnel'],
               'R2L': ['ftp_write', 'phf', 'guess_passwd', 'warezmaster', 'warezclient',
                       'imap', 'spy', 'multihop', 'named', 'snmpguess', 'worm', 'snmpgetattack',
                       'xsnoop', 'xlock', 'sendmail'],
               'normal': ['normal']
               }

    # return the group type
    for grp, names in mapping.items():
        for name in names:
            if name == value:
                return grp
    return None
```

4.2 Label Encoding and Re-sampling minority Classes

We will convert the categorical columns into a **Label Encoded** format. This is then followed by up-sampling the *U2R* and *R2L* samples. And then followed by down-sampling the *Normal* samples.

```
def createfeatureslabelset(dataframe, train=True):
    """
    Label encodes the 'object' type columns
    :param dataframe: <pd.Dataframe> Dataframe having NSL-KDD records
    """
    # Drop the class column
    dataframe = dataframe.drop('class', axis=1)

    encoder = LabelEncoder()
    # extract categorical attributes from dataframe
    # encode the categorical attributes
    categorical = dataframe.select_dtypes(include=['object']).apply(encoder.↘
        fit_transform)
    # Now let us merge both scaled and encoded feature dataframes
    dataframe = pd.concat([dataframe.drop(['protocol_type', 'service',
                                           'flag', 'class_group'],
                                          axis=1), categorical],
                          axis=1)

    if train:
        # separate minority and majority classes
        DoSsection = dataframe[dataframe['class_group'] == 0]
        Probesection = dataframe[dataframe['class_group'] == 1]
        R2Lsection = dataframe[dataframe['class_group'] == 2]
        U2Rsection = dataframe[dataframe['class_group'] == 3]
        Normalsection = dataframe[dataframe['class_group'] == 4]

        # upsample minority
        R2L_upsampled = resample(R2Lsection,
                                 # sample with replacement
                                 replace=True,
                                 # match number in majority class
                                 n_samples=len(Probesection),
                                 # reproducible results
                                 random_state=27)

        U2R_upsampled = resample(U2Rsection,
                                 replace=True,
                                 n_samples=len(Probesection),
                                 random_state=27)

        # downsample majority
        Normal_downsample = resample(Normalsection,
                                     replace=False,
                                     n_samples=len(Probesection),
                                     random_state=27)

        # combine majority and upsampled minority
        dataframe = pd.concat([DoSsection, Probesection,
                               R2L_upsampled,
                               U2R_upsampled,
                               Normal_downsample])

    features = dataframe.drop('class_group', axis=1)
    labels = pd.DataFrame()
    labels['labels'] = dataframe['class_group']

    return features, labels
```

4.3 Normalisation

We will use *MinMaxScalar* from sklearn to adjust all continuous values from dataset between 0 and 1. Also, some features with binary results like either 0 or 1 should not be considered for normalisation. Those features with binary values are *'land', 'logged_in', 'root_shell', 'su_attempted', 'is_host_login', 'is_guest_login'*

```
def normalisation(dataframe):
    """
    Apply Minmaxscalar on all int64 and float64 columns from dataframe.
    """
    # Extract if any binary columns present in dataframe
    binarycolumns = ['land', 'logged_in',
                     'root_shell', 'su_attempted',
                     'is_host_login', 'is_guest_login']

    bc = []
    for col in dataframe.columns:
        if col in binarycolumns:
            bc.append(col)
    binarydf = dataframe[bc]
    dataframe = dataframe.drop(bc, 1)
    dataframe = MinMaxScaler().fit(dataframe)

    categoricaldf = dataframe[dataframe.select_dtypes(include=['int32',
                                                               'int64',
                                                               'float64']).columns]
    result = pd.concat([binarydf, categoricaldf], axis=1)

    return result
```

5 Feature Selection Techniques

5.1 Embedded Technique

A two different feature selection techniques were applied i.e. *LASSO* and *Elastic-Net*

```
"""
This program implements different embedded/wrapper techniques for feature selection
"""
from sklearn.linear_model import Lasso, ElasticNet
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


class LassoSelection:
    """
    Class to apply LASSO feature selection Technique.
    """
    def __init__(self, X, y):
        """
        Initialize feature columns and label column
        :param X: <DataFrame> Features from Sensor Dataframe
        :param y: <DataFrame> Label column from Sensor Dataframe
        """
        self.X = X
        self.y = y
        self.coef = None

    def applyLasso(self):
        lasso = Lasso(alpha=0.1)
        lasso.fit(self.X, self.y)
```

```
        self.coef = lasso.coef_
        return np.abs(self.coef)


class Elasticnet(LassoSelection):
    """
    Class to apply Elastic-Net feature selection Technique. It is hybrid
    of LASSO and Ridge
    """
    def applyElasticNet(self):
        regr = ElasticNet(alpha=0.01)
        regr.fit(self.X, self.y)
        return np.abs(regr.coef_)
```

5.1.1 Creating Feature Ranking table

A ranks table is created which arranges the features in ranked order of their feature importance on the class column.

```
def rank_to_dict(ranks, names=trainX.columns, order=1):
    """
    Apply minmax scaler to normalise score or rank values of features
    """
    minmax = MinMaxScaler()
    ranks = minmax.fit_transform(order*np.array([ranks]).T).T[0]
    ranks = map(lambda x: round(x, 2), ranks)
    return dict(zip(names, ranks))


def create_ranks_dataframe(ranks):
    """
    This function does the following manipulations:
        1. Converts dictionary of ranks to a dataframe 'ranks_dataframe'.
        2. Creates a new column 'Mean' in 'ranks_dataframe' which specifies the
        mean along the rows.
        3. Sorts the 'ranks_dataframe' in descending order of 'Mean' Value.
        4. Creates a another column 'Rank' based on 'Mean' Column.

    :param ranks: <Dict> Dictionary of feature importance scores from all feature↘
         ranking
    methods
    :return: <DataFrame> Converted ranks to a dataframe format
    """
    mean_ranks = {}

    for name in trainX.columns:
        mean_ranks[name] = np.mean([ranks[method][name] for method in ranks.keys↘
            ()])

    methods = sorted(ranks.keys())
    ranks["Mean"] = mean_ranks
    methods.append("Mean")

    ranks_dataframe = pd.DataFrame(ranks).sort_values(by="Mean", ascending=False)

    # Consider index of 'sensorX' inside column name
    ranks_dataframe.reset_index(level=0, inplace=True)

    # Create new column called as Rank and sequentially increment its value
    ranks_dataframe["Rank"] = ranks_dataframe.index + 1
    return ranks_dataframe


#*************************************************************************
#                  APPLY EMBEDDED METHOD FOR FEATURE RANKING             #
#*************************************************************************
def callembeddedmethod(X=trainX, y=trainy):
    """
    In this function, various embedded feature rankings/selection methods are ↘
        implemented
```

```
    side-by-side and the feature importance values are returned to a output dict for each
    of those methods.

    :param X: <Numpy> Array of features from training set
    :param y: <Numpy> Array of labels from training set
    :return: <Dict> Dictionary of feature importance scores from all feature ranking methods
    """
    # Create objects for respective embedded methods
    lasso = featureselection.LassoSelection(X, y)
    net = featureselection.Elasticnet(X, y)

    dict_rank = {"Lasso": rank_to_dict(lasso.applyLasso()),
                 "Elastic-Net": rank_to_dict(net.applyElasticNet())}
    return dict_rank
```

5.2 Filter-based Technique

Pearson Correlation coefficient matrix is used to remove highly correlated features.

```
class Correlation(LassoSelection):
    """
    Brute Force Method to find Correlation between features
    """
    def correlation(self, threshold=0.8):
        # Set of all names of correlated columns
        data = pd.concat([self.X, self.y], axis=1)
        data = data.drop(['num_outbound_cmds',
                          'is_host_login'], axis=1)
        col_corr = set()
        corr_mat = data.corr()
        for i in range(len(corr_mat.columns)):
            for j in range(i):
                if abs(corr_mat.iloc[i, j]) > threshold:
                    colname = corr_mat.columns[i]
                    col_corr.add(colname)
        # plot the heatmap
        CorrMtx(corr_mat)
        return col_corr

#*****************************************************************************
#                APPLY FILTER METHOD FOR FEATURE SELECTION                    #
#*****************************************************************************
def callfiltermethod(X=trainX, y=trainy):
    """
    In this function, pearson correlation coefficient used to eliminate features
    """
    # Create objects for respective filter method
    corr = featureselection.Correlation(X, y)
    droppedfeatures = corr.correlation()
    return droppedfeatures
```

5.2.1 To Plot heatmap of Correlation matrix

```
def CorrMtx(df, dropDuplicates = True):

    # Your dataset is already a correlation matrix.
    # If you have a dateset where you need to include the calculation
    # of a correlation matrix, just uncomment the line below:
    # df = df.corr()

    # Exclude duplicate correlations by masking uper right values
    if dropDuplicates:
        mask = np.zeros_like(df, dtype=np.bool)
        mask[np.triu_indices_from(mask)] = True

    # Set background color / chart style
    sns.set_style(style = 'white')
```

```
15
16      # Set up  matplotlib figure
17      f, ax = plt.subplots(figsize=(11, 9))
18
19      # Add diverging colormap from red to blue
20      cmap = sns.diverging_palette(250, 10, as_cmap=True)
21
22      # Draw correlation plot with or without duplicates
23      if dropDuplicates:
24          map = sns.heatmap(df, mask=mask, cmap=cmap,
25                            square=True,
26                            linewidth=.5, cbar_kws={"shrink": .8}, ax=ax)
27      else:
28          map = sns.heatmap(df, cmap=cmap,
29                            square=True,
30                            linewidth=.5, cbar_kws={"shrink": .8}, ax=ax)
31      map.set_yticklabels(map.get_yticklabels(),
32                          horizontalalignment='right',
33                          fontweight='light',
34                          fontsize='small')
35
36      map.set_xticklabels(map.get_xticklabels(), rotation=60,
37                          horizontalalignment='right',
38                          fontweight='light',
39                          fontsize='small')
40      plt.show(block=True)
```

6 Implementation of Classical Machine Learning Algorithms

A series of different machine learning algorithms are applied to evaluate performance on validation and test set.

- Logistic Regression
- K-Nearest Neighbours
- Decision Tree
- Random Forest
- Linear-SVM
- PCA with Linear-SVM

```
1   def fitmodels(dataset, train_label, test_label, val_label, model,
2                 model_name='logistic_regression'):
3       """
4       This model implements each algorithm
5       """
6       Xtrain, Ytrain, Xval, Yval, Xtest, Ytest = dataset
7       test_accuracy = []
8       valid_accuracy = []
9       classes = ['DoS', 'Probe', 'R2L', 'U2R', 'normal']
10      precision = {'DoS': None, 'Probe': None,
11                   'R2L': None, 'U2R': None, 'normal': None}
12      recall = {'DoS': None, 'Probe': None,
13                'R2L': None, 'U2R': None, 'normal': None}
14      f1score = {'DoS': None, 'Probe': None,
15                 'R2L': None, 'U2R': None, 'normal': None}
16
17      for attack_type in range(5):
```

```
18          model.fit(Xtrain, train_label[attack_type])
19          y_pred = model.predict(Xtest)
20          stacked_train_df[attack_type][model_name] = model.predict(Xtrain)
21          stacked_test_df[attack_type][model_name] = y_pred
22          y_pred_val = model.predict(Xval)
23          stacked_val_df[attack_type][model_name] = y_pred_val
24          valid_accuracy += [metrics.accuracy_score(val_label[attack_type],
25                                                    y_pred_val)]
26          test_accuracy += [metrics.accuracy_score(test_label[attack_type],
27                                                   y_pred)]
28
29          precision[classes[attack_type]] = \
30              np.round(metrics.average_precision_score(test_label[attack_type],
31                                                       y_pred)*100, 2)
32
33          recall[classes[attack_type]] = \
34              np.round(metrics.recall_score(test_label[attack_type],
35                                            y_pred,
36                                            average='weighted')*100, 2)
37          f1score[classes[attack_type]] = \
38              np.round(metrics.f1_score(test_label[attack_type],
39                                        y_pred,
40                                        average='weighted')*100, 2)
41          cm = metrics.confusion_matrix(test_label[attack_type], y_pred)
42          plot_confusion_matrix(cm, labels=[classes[attack_type], 'rest'],
43                                title=extract.CM_DIR +
44                                      model_name + '_{}.jpeg'.format(classes[↘
                                        attack_type]))
45
46      mean_test_accuracy = np.mean(test_accuracy)
47      mean_valid_accuracy = np.mean(valid_accuracy)
48      print("Mean Accuracy score "
49            "on Validation Set:\t{} %".format(np.round(mean_valid_accuracy*100,
50                                                      2)))
51      print("Mean Accuracy score "
52            "on Test Set:\t\t{} %".format(np.round(mean_test_accuracy*100,
53                                                   2)))
54      print("Average Precision score "
55            "on Test Set:\t{} ==> {} %".format(precision,
56                                               np.round(np.array([precision[k]
57                                                                  for k in
58                                                                  precision]).mean↘
                                                                      (),
59                                                        2)))
60      print("Average Recall score "
61            "on Test Set:\t\t{} ==> {} %".format(recall,
62                                                 np.round(np.array([recall[k]
63                                                                    for k in
64                                                                    recall]).mean()↘
                                                                        ,
65                                                          2)))
66      print("Average f1-score "
67            "on Test Set:\t\t\t{} ==> {} %".format(f1score,
68                                                   np.round(np.array([f1score[k]
69                                                                      for k in
70                                                                      f1score]).↘
                                                                          mean(),
71                                                            2)))
72
73
74  def applyModel(dataset, selected_columns=columns):
75      """
76      Apply different model on dataset
77      """
78      Xtrain, Ytrain, Xval, Yval, Xtest, Ytest = dataset
79      Xtrain = splitdataset.normalisation(Xtrain[selected_columns])
80      Xtest = splitdataset.normalisation(Xtest[selected_columns])
81      Xval = splitdataset.normalisation(Xval[selected_columns])
82      scalar = MinMaxScaler()
83      scalar.fit(Xtrain)
84      Xtrain = scalar.transform(Xtrain)
85      Xtest = scalar.transform(Xtest)
86      Xval = scalar.transform(Xval)
```

```
87
88      dataset = Xtrain, Ytrain, Xval, Yval, Xtest, Ytest
89      train_label = pd.DataFrame()
90      test_label = pd.DataFrame()
91      val_label = pd.DataFrame()
92
93      # One hot encode the class column
94      for attack_type in range(5):
95          train_label[attack_type] = Ytrain['labels'].apply(lambda x: int(x ==
                attack_type))
96          test_label[attack_type] = Ytest['labels'].apply(lambda x: int(x ==
                attack_type))
97          val_label[attack_type] = Yval['labels'].apply(lambda x: int(x ==
                attack_type))
98
99      print("\n**********LogisticRegression**********")
100     fitmodels(dataset, train_label, test_label, val_label,
101               model=LogisticRegression(C=0.2, solver='liblinear'))
102
103     print("\n**********SGDClassifier**********")
104     fitmodels(dataset, train_label, test_label, val_label,
105               model=SGDClassifier(), model_name='SGDClassifier')
106
107     print("\n**********KNearestNeighbours**********")
108     fitmodels(dataset, train_label, test_label, val_label,
109               model=KNeighborsClassifier(n_neighbors=30),
110               model_name='KNearestNeighbours')
111
112     print("\n**********DecisionTree**********")
113     fitmodels(dataset, train_label, test_label, val_label,
114               model=DecisionTreeClassifier(max_depth=8),
115               model_name='DecisionTree')
116
117     print("\n**********RandomForest**********")
118     fitmodels(dataset, train_label, test_label, val_label,
119               model=RandomForestClassifier(max_features='sqrt',
120                                            max_depth=8,
121                                            n_estimators=900),
122               model_name='RandomForest')
123
124     print("\n**********Support Vector Classifier
            **********")
125     fitmodels(dataset, train_label, test_label, val_label,
126               model=LinearSVC(C=100),
127               model_name='SupportVectorClassifier')
128
129     print("\n**********Principal Component Analysis
            **********")
130     pca = PCA(n_components=10)
131     pca.fit(Xtrain)
132     Xtrain = pca.transform(Xtrain)
133     Xtest = pca.transform(Xtest)
134     Xval = pca.transform(Xval)
135     print("Shape of PCA Components: {}".format(pca.components_.shape))
136     dataset = Xtrain, Ytrain, Xval, Yval, Xtest, Ytest
137     fitmodels(dataset, train_label, test_label, val_label,
138               model=LinearSVC(C=100),
139               model_name='PCAwithSVC')
140
141     print("\n**********XGBClassifier**********")
142     fitmodels(dataset, train_label, test_label, val_label,
143               model=XGBClassifier(max_depth=6, learning_rate=0.01, n_estimators
                    =300, nthread=4,
144                                   min_child_weight=4, subsample=0.7,
                                        colsample_bytree=0.7),
145               model_name='XGBClassifier')
146
147     print("\n**********StackedAccuracy**********")
148     stacked_accuracy = []
149
150     for attack_type in range(5):
151         clf = KNeighborsClassifier(n_neighbors=30)
152         clf.fit(stacked_train_df[attack_type], train_label[attack_type])
```

```
153             y_pred = clf.predict(stacked_test_df[attack_type])
154             stacked_accuracy += [metrics.accuracy_score(test_label[attack_type], ↘
                    y_pred)]
155 
156         mean_stacked_accuracy = np.mean(stacked_accuracy)
157         print("Mean Accuracy score on Test Set: {}".format(mean_stacked_accuracy))
```