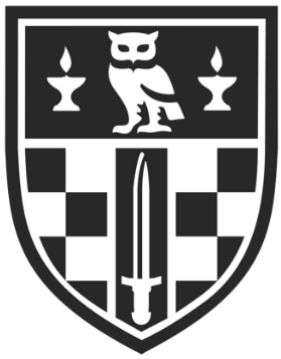


CSIS

Module Name :
Applied Machine Learning

Assignment Type :
Individual Practical Project

Assignment Title:
Cyber Threat Detection Competition using Machine Learning

Course Name:
MSc. Advance Computing Technologies

Course Tutor:
Dr. Paul Yoo

Student Name: Rohan Khanolkar
Student Number: 13199041
Word Count of Report: 2858

A network of computer system whose security is compromised by various (anomalies) kind of attacks which are known and unknown to users is known as cyber-attacks[1]. National Institute of Standards and Technology, USA depicts cyber threat as [2]

"*An attack, via cyberspace, targeting an enterprise's use of cyberspace for the purpose of disrupting, disabling, destroying, or maliciously controlling a computing environment / infrastructure; or destroying the integrity of the data or stealing controlled information.*".

As cyber security is a vast topic, we will be focusing on the section where we will detect anomalies and classifying it as normal or attack, this is known as

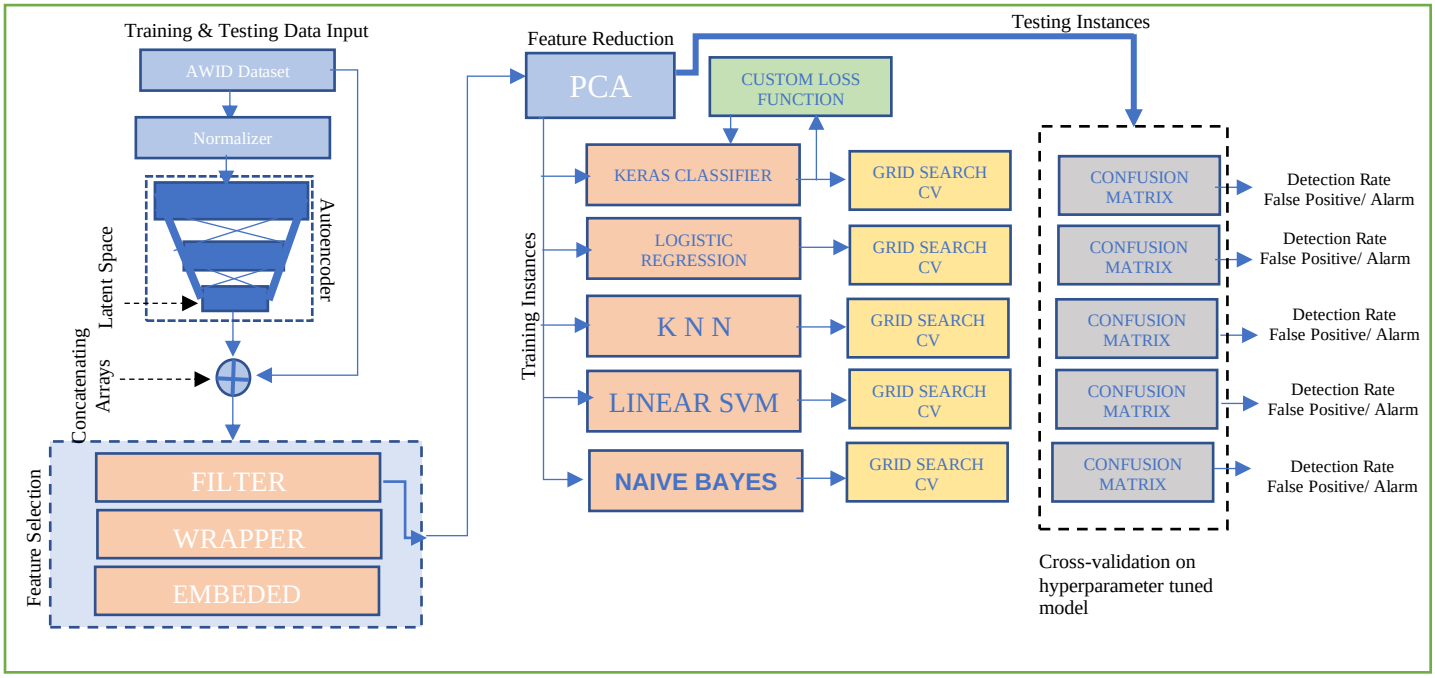
"*Deep Extraction and Mutual Information Selection (DEMISe)*" by Parkar et al [3].

In this paper they used a stacked autoencoder which was deep structured for feature extraction before information share by every feature and labels. This paper also shows that how wrapper method was achieved using tree based entropy and how this method lowered the computational cost improved interpretability of models to achieve high detection rate.

The IMPACT [4] paper also used stacked autoencoder to extract features and later use is on linear support vector machine using the gradient decent which demonstrated better performance and lower false alarm rate than the DEMISe paper as more training time was given and it also created a benchmark.

In this coursework, we will use the same AWID dataset for intrusion detection which has 154 features with 1 target variable. To avoid unfair predictions feature number 4 & 7 are removed from the dataset, so the dataset now has 152 features and 1 target variable.

Flow of the Course work:



1) Dataset and Loading the training & testing dataset:

The AWID datasets provided to us is balanced, where there are no missing values and target variable is marked appropriately. We will first import the 'pandas' to initialize library, then we locate the file name and use 'read_csv()' and initialized the file name. We have not used 'delimiter' as the data is already in an array format and there was no need to use it. Apart from that, we have not used label of the dataset as 'pandas data frame' do not used labels/header as default [5]. Further we have separated 152 feature (as X and X_t) and target variable (as Y and Y_t). Refer 1.1

1) Loading training and testing dataset

```
[ ]  # Loading the Training Data
     filename = 'train_imperson_without4n7_balanced_data.csv'
     train_dataframe = read_csv(filename)
     array = train_dataframe.values
     # separate array into input and output components
     X = array[:,0:152]
     Y = array[:,152]
```

```
     # Loading the Testing Data
     filename = 'test_imperson_without4n7_balanced_data.csv'
     test_dataframe = read_csv(filename)
     array = test_dataframe.values
     # separate array into input and output components
     X_t = array[:,0:152]
     Y_t = array[:,152]
```

1.1 : Codes used for loading data

1) (a) Data Description:

In this section 'dataframe.describe()' function was used to summarise the distribution, shape, central tendency and dispersion label of the training and testing dataset, it also excludes 'NaN' values and gives the total count, the mean, the standard deviation, the min & max values along with percentile 25 (lower) 50(==median) 75(higher) as default [6]. Refer 1.2 &1.3

```
set_option('display.width', 100)
set_option('precision', 3)
description_train = train_dataframe.describe()
description_train
```

	1	2	3	5	6	8	9	10	11	12	13	14	15	
count	97044.0	97044.0	97044.0	9.704e+04	9.704e+04	97044.000	97044.000	97044.0	97044.0	97044.0	97044.0	97044.000	97044.000	970
mean	0.0	0.0	0.0	6.252e-03	6.252e-03	0.194	0.194	0.0	0.0	0.0	0.0	1.000	1.000	
std	0.0	0.0	0.0	1.554e-02	1.554e-02	0.354	0.354	0.0	0.0	0.0	0.0	0.015	0.015	
min	0.0	0.0	0.0	2.860e-06	2.860e-06	0.000	0.000	0.0	0.0	0.0	0.0	0.000	0.000	
25%	0.0	0.0	0.0	1.442e-03	1.442e-03	0.038	0.038	0.0	0.0	0.0	0.0	1.000	1.000	
50%	0.0	0.0	0.0	3.706e-03	3.706e-03	0.038	0.038	0.0	0.0	0.0	0.0	1.000	1.000	
75%	0.0	0.0	0.0	5.916e-03	5.916e-03	0.055	0.055	0.0	0.0	0.0	0.0	1.000	1.000	
max	0.0	0.0	0.0	9.784e-01	9.784e-01	1.000	1.000	0.0	0.0	0.0	0.0	1.000	1.000	

8 rows × 153 columns

1.2 : Summary of the Training dataset

```
set_option('display.width', 100)
set_option('precision', 3)
description_test = test_dataframe.describe()
description_test
```

	1	2	3	5	6	8	9	10	11	12	13	14	15	
count	40158.0	40158.0	40158.0	4.016e+04	4.016e+04	40158.000	40158.000	40158.0	40158.0	40158.0	40158.0	40158.000	40158.000	401
mean	0.0	0.0	0.0	5.080e-03	5.080e-03	0.201	0.201	0.0	0.0	0.0	0.0	1.000	1.000	
std	0.0	0.0	0.0	1.619e-02	1.619e-02	0.371	0.371	0.0	0.0	0.0	0.0	0.017	0.017	
min	0.0	0.0	0.0	5.810e-06	5.810e-06	0.000	0.000	0.0	0.0	0.0	0.0	0.000	0.000	
25%	0.0	0.0	0.0	1.362e-03	1.362e-03	0.017	0.017	0.0	0.0	0.0	0.0	1.000	1.000	
50%	0.0	0.0	0.0	2.262e-03	2.262e-03	0.017	0.017	0.0	0.0	0.0	0.0	1.000	1.000	
75%	0.0	0.0	0.0	3.163e-03	3.163e-03	0.060	0.060	0.0	0.0	0.0	0.0	1.000	1.000	
max	0.0	0.0	0.0	9.048e-01	9.048e-01	1.000	1.000	0.0	0.0	0.0	0.0	1.000	1.000	

8 rows × 153 columns

1.3 : Summary of Testing dataset

2) Data Preprocessing: Normalizing the training and testing dataset:

Data preprocessing is a list of techniques used to process data from its raw format to a efficient format which can be used to process in various machine learning algorithms [7]. Normalization is a part of data 'sklearn' data preprocessing which samples every individual unit of the dataset, this usually samples each row of the dataset and rescales the data independently that its norm is equals to an approximate one [8]. Performing this normalizer function helps in avoid overfitting in the dataset when it is compiled with machine learning models. Refer 2.1

▾ 2) Normilizing the training and testing dataset

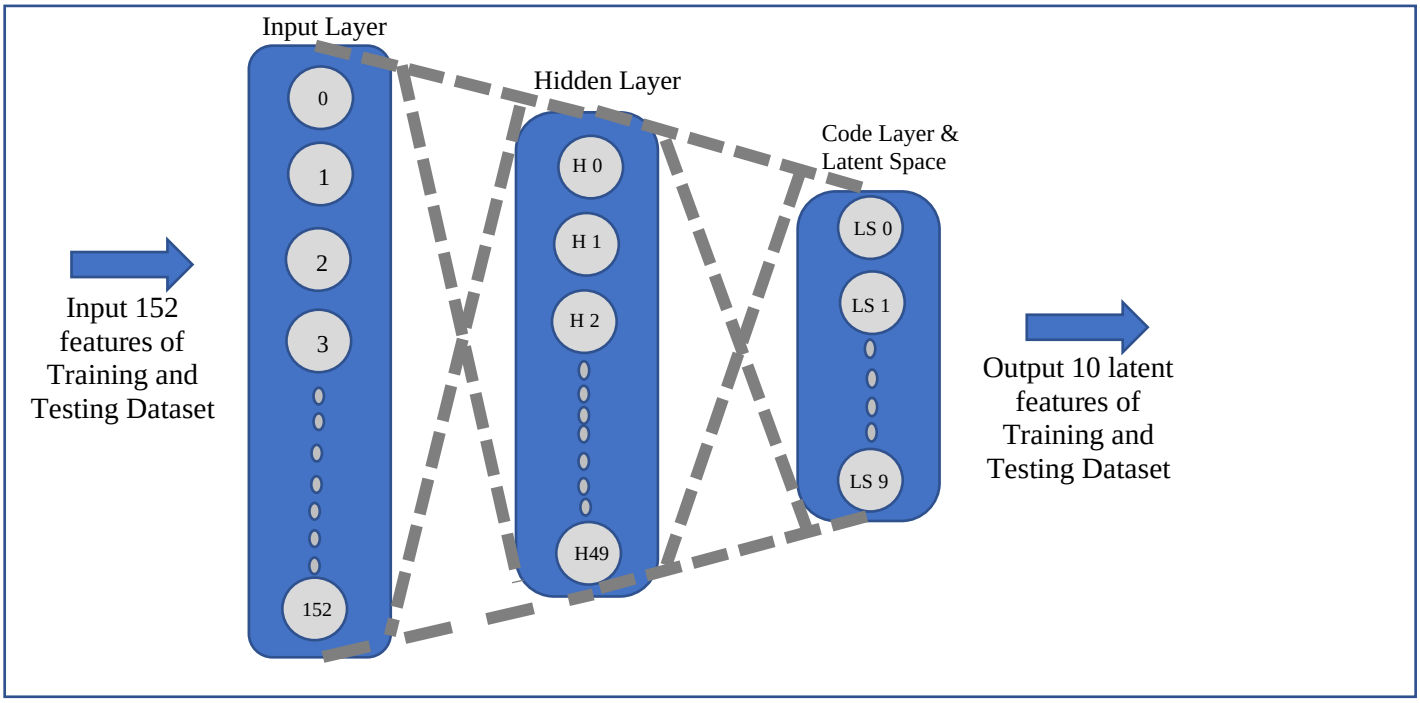
```
[ ] # Here we normilize the the New traning dataset (with 162 features)
    train_scaler = Normalizer().fit(X)
    train_normalizedX = train_scaler.transform(X)
```

```
# Here we normilize the the New traning dataset (with 162 features)
test_scaler = Normalizer().fit(X_t)
test_normalizedX = test_scaler.transform(X_t)
```

2.1 : Codes used to apply Normalizer on datasets.

3) Autoencoder & adding extra features to the training and testing dataset:

The autoencoder layer is used to extract 10 informative and important (latent) features form its latent space. These 10 latent features are actually a compressed version of the 152 existing features with 50 hidden layers. Refer 3.1



3.1: Representation of Autoencoder

Here we can note from below codes (Refer 3.2) we can see that we have used the rectified leaner unit (relu) activation function because it will reflect all the positive input values and zero all the negative values in the dataset to minimize gradient issues and uplift the performance. We can note below that we have extracted 10 latent features from both training and testing dataset [9].

3) Applying Autoencoder on the new normilized Training and Testing data to get the 10 additional features

```
[8]  # Applying Autoencoder on normilized Training data set
     input_size = 152
     hidden_size= 50
     code_size = 10
     input_dat= Input(shape=(input_size,))
     hidden_1 = Dense(hidden_size, activation='relu')(input_dat)
     code = Dense(code_size, activation='relu', activity_regularizer=l1(10e-6))(hidden_1)
     encoder = Model(input_dat, code)
     encoded_train_vals = encoder.predict(train_normalizedX)
```

```
[9]  # Applying Autoencoder on normilized Testing data set
     encoded_test_vals = encoder.predict(test_normalizedX)
```

```
[10] # To check the encoder output of 10 features for normilized Training dataset
     encoded_train_vals.shape

     (97044, 10)
```

```
     # To check the encoder output of 10 features for normilized Testing dataset
     encoded_test_vals.shape

     (40158, 10)
```

3.2: Codes used to extract latent features from the datasets

4) Adding 10 latent features in the dataset:

We will used ‘numpy’ array concatenate function to add the 10 latent features extracted from the autoencoder layer. The training and testing dataset will now become 162 features and 1 target variable. Refer 4.1

4) Adding the 10 new features acquired from the normilized Training and Testing dataset and combining these features with the normilized training and testing datasets to acquiring 162 features in both of them.

```
[12] # To add the 10 new Training features in the Normalized Training data set X
     Train_array1 = np.array(train_normalizedX)
     Train_array2 = np.array(encoded_train_vals)
     Train_array_X = np.concatenate((Train_array1, Train_array2), axis=1)

[13] Train_array_X.shape

     (97044, 162)

[14] # To add the 10 new Testing features in the Normalized Testing data set X_t
     Test_array1 = np.array(test_normalizedX)
     Test_array2 = np.array(encoded_test_vals)
     Test_array_X = np.concatenate((Test_array1, Test_array2), axis=1)

[15] Test_array_X.shape

     (40158, 162)
```

4.1: Codes used to concatenate the 10 latent features with the dataset [10]

5) Feature Selection using Filter, Wrapper & Embedded method

5)(a) Filter Method using SelectKBest:

‘SelectKBest’ function is a part of ‘sklearn’ feature selection library. This method selects best 50 best features from the 162 features by using Chi squared statistics as its function [11]. Refer 5.a.1 , we will note that the function had selected 50 features out of the 162 features of the training dataset. Further, we used the same SelectKBest model which we had fit the training data and transformed it on the testing dataset to get 50 best features from it.

5) Now we will used Filter, Wrapper and Embeded method for Feature selection. To select the best features in the new test and train dataset with 162 features.

(a) Filter method - using SelectKBest :

```
[16] #Select k best on train set
     selectModel = SelectKBest(score_func=chi2, k=50)
     kBesttrain = selectModel.fit(Train_array_X,Y)
     selectTrainFeatures = kBesttrain.transform(Train_array_X)

[17] #Select k best on test set
     selectTestFeatures = kBesttrain.transform(Test_array_X)

     selectTrainFeatures.shape

     (97044, 50)

     + Code   + Text

[19] selectTestFeatures.shape

     (40158, 50)
```

5.a.1: Codes used to extract 50 features from the 162 features of training and testing dataset

5)(b) Wrapper method using Recursive feature elimination (RFE):

The Wrapper method uses recursive feature elimination (RFE) which is a backward selection process building a model to compute important scores for every predictor [11]. This model uses logistic regression from 'sklearn' library with a 'liblinear' solver which helps to select 50 features from 162 input features of training and testing dataset. Further, we used to same RFE model which we had fit the training data and transformed it on the testing dataset to get 50 best features from it. Refer 5.b.1

(b) Wrapper method - using Recursive feature elimination (RFE)

```
[ ]  #RFE on train set
     model_rfe = LogisticRegression(solver='liblinear')
     rfeModel = RFE(model_rfe, 50)
     rfeTrain = rfeModel.fit(Train_array_X,Y)
     rfeTrainFeatures = rfeTrain.transform(Train_array_X)
```

```
[ ]  rfeTrainFeatures.shape
```

```
     (97044, 50)
```

```
[ ]  #RFE on test set
     rfeTestFeatures = rfeTrain.transform(Test_array_X)
```

```
[ ]  rfeTestFeatures.shape
```

```
     (40158, 50)
```

5.b.1: Codes used to extract 50 features from the 162 features of training and testing dataset

5)(c) Embedded method using ExtraTreesClassifier:

ExtraTreeClassifier is the method used in embedded method of feature selection. This method computes impurities based on the importance of the features, which helps in identifying the important features and eliminating the irrelevant features [11]. We had set n_estimators as 50 to select 50 features but it only selected 36 features out of 162 features. Refer to 5.c.1, where we had fit the training data and transformed it on the testing dataset to get best features from it.

(c) Embedded method - using ExtraTreesClassifier

```
[24] #ExtraTree on train set
     extraTreeModel = ExtraTreesClassifier(n_estimators=50)
     exTreeTrain = extraTreeModel.fit(Train_array_X, Y)
     clf_model = SelectFromModel(exTreeTrain, prefit=True)
     exTreeTrainFeatures = clf_model.transform(Train_array_X)
```

```
[25] exTreeTrainFeatures.shape
```

```
     (97044, 36)
```

```
[26] #ExtraTree on test set
     exTreeTestFeatures = clf_model.transform(Test_array_X)
```

```
[27] exTreeTestFeatures.shape
```

```
     (40158, 36)
```

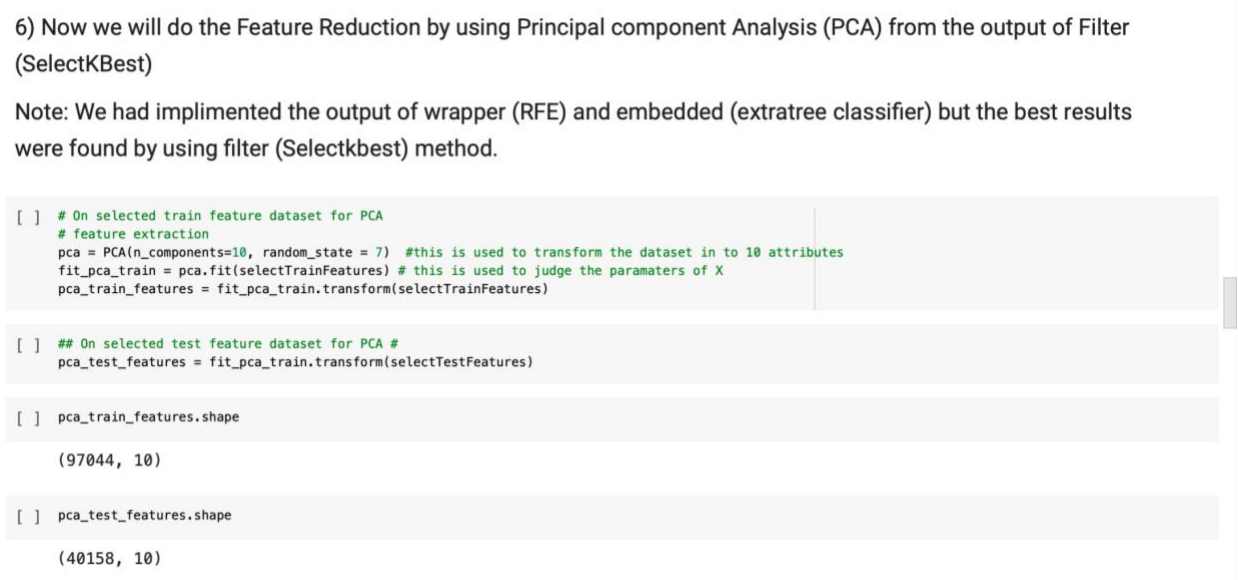
5.c.1: *Codes used to extract 36 features from the 162 features of training and testing*

NOTE:

We had used the output of the Filter, Wrapper & Embedded methods and found out that the model accuracy and detection rate of wrapper and embedded method was below 70% and the output of filter method was showing an remarkable performance the model accuracy and detection rate. Thus we went ahead providing the filter output to the Principal Component analysis (PCA). Please find the Output of Wrapper & Embedded method in Appendix 1 & 2

6) Feature Reduction by using Principal component Analysis (PCA):

We have used Principal Component Analysis (PCA) (a part of 'sklearn' decomposition library) as feature reductions technique because it finds the eigenvectors within the arrayed dataset and finds its eigenvalues to project the best ranking data into a new subspace with lower dimensions (or features) [12]. Refer 6.1, we will note that we have given 'n_components = 10' because we want to extract 10 features from the output of SelectKBest (i.e. 50 features). In this you will note that we have fit the training data and transformed it on the testing dataset to get best 10 features from it.



6.1: Codes used to extract 10 features from the 50 features of SelectKBest training and testing output [12]

7) Building and Training Models:

First, we have built five machine learning models and Trained them with various parameters and found the model accuracy on training dataset. Then we used 'GridSearchCV' which is a part of 'Sklearn" model selection library to find the best parameters which gives the best accuracy for each model. Below every model codes you will find the codes for 'GridSearchCV, which passes a param grid to identify the parameter of the model [13].

7)(a) Model 1- Deep learning model using KerasClassifier:

The 'KerasClassifier' is a part of 'keras.wrappers.scikit_learn' library which uses the numerical library of 'tenserflow'. The 'KerasClassifier' has multiple ways of tuning hyperparameter i.e creating a model within a model and it also supports tuning loss function. Refer to 7.a.1, the codes used for building Keras model.

```
# Grid Search Deep Learning Model Parameters
# create a function to build a model, required for KerasClassifier

optimizers = ['adam', 'rmsprop']
inits = ['uniform', 'glorot_uniform']
epochs = [5, 7, 10]
batches = [20, 30, 40]

# Custom Loss funtion
def custom_loss(y_true, y_pred):
    alpha = 7
    beta = 3
    loss_mean = tf.keras.losses.mean_squared_error(y_true, y_pred)
    loss_binary = tf.keras.losses.binary_crossentropy(y_true, y_pred)
    loss_KL = tf.keras.losses.kullback_leibler_divergence(y_true, y_pred)
    loss = loss_mean+(alpha*loss_binary)+(beta*loss_KL)
    return loss

def create_model(optimizer=optimizers, init=inits):
  # create model
  mlp_model = Sequential()
  mlp_model.add(Dense(12, input_dim=10, activation='relu'))
  mlp_model.add(Dense(10, activation='relu'))
  mlp_model.add(Dense(1, activation='sigmoid'))

  # Compile model
  mlp_model.compile(loss = custom_loss, metrics=["accuracy"])
  return mlp_model

# create model
model_keras = KerasClassifier(build_fn=create_model, verbose=0)

# grid search epochs, batch size and optimizer
param_grid = dict(optimizer=optimizers, epochs=epochs, batch_size=batches, init=inits)
grid = GridSearchCV(estimator=model_keras, param_grid=param_grid)
grid_result = grid.fit(pca_train_features,Y)

# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.970137 using {'batch_size': 40, 'epochs': 7, 'init': 'glorot_uniform', 'optimizer': 'rmsprop'}
```

7.a.1: Codes used to build and train KerasClassifier with GridSearchCV [14] [15]

In the above code block we had first started created a KerasClassifier model only with activation function 'relu', metrices as "accuracy", 'verbose=0' (to run multiple line per epoch) and tried to fit it on training dataset, this returned with an error asking for missing input parameters. So we created another model within the 'KerasClassifier' model and name it as 'create_model' and we sequentially added input variables with activation function 'relu' (i.e. rectified linear unit) which identifies all positive values and zero out all negative values. We then sequentially added sigmoid function as output of model which will either give 1 value between 0. The accuracy was 0.6708, so we added a 'custom_loss' function to identify if we can increase the accuracy of the model. Thus we identified that all the values will be positive in this models we first tried 'mean_squared_error' which did not affect the accuracy of the model. So we added, the KL function known as the 'kullback_leibler_divergence' the accuracy showed minimum changes, after that we added the 'tenserflow.keras' binary crossentrophy for losses, the accuracy showed a drastic difference. We then tired add tuning values to the KL loss function and Binary loss function added and created a stable loss function for the Keras model.

i.e. *custom loss function = Mean Squared Error + (Tune_value_alpha * Binary Loss Function) + (Tune_value_alpha * KL Divergence Function)*

This function gave us a 0.9701 accuracy with its best parameters given by the 'GridSearchCV'

7)(b) Model 2 – Logistic Regression:

This method was selected because it can handle sparse data as well as dense data. The 'LogisticRegression' is a part of 'Sklearn' leanier model libereay, its main function is to predict the values close to 1 and 0 by learning the from the training dataset [16] and C value is passed as a array of multiple values to check the optimal performance with 'GridSearchCV' along with panelty as 'l2' which support 'lbfg' solvers if active (it do not support 'liblinear' solver). Thus 'liblinear' is specified as a parameter for 'GridSerachCV'. Refer 7.b.1, the accuracy of the model is 0.9683 along with the best parameter to achieve it.

(b) Logistic regression

```
solvers = ['lbfgs', 'liblinear']
penalty = ['l2']
c_values = [100,10, 1.0, 0.1, 0.01]
param_grid = dict(solver=solvers,penalty=penalty,C=c_values)
model = LogisticRegression(max_iter=2000)
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid.fit(pca_train_features, Y)
print(grid.best_score_)
print(grid.best_estimator_)
```

```
0.9683431260397679
LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=2000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

7.b.1: Codes used to build and train LogisticRegression with GridSearchCV and its best results.

7)(c) Model 3 – K- Nearest Neighbors (KNN):

The K-Nearest Neighbor is known as 'KNeighborsClassifier' from the 'Sklearn' neighbors library function. This identifies the nearest neighbors of each point with as simple used of majority vote and stores the training data instances. It is given with 'n_neghbors' as range and metric as 'euclidean', 'manhattan', 'minkowski' [17] and weight as 'uniform', 'distance', so that 'GridSearchCV' can evaluate parameter of the model. Refer 7.c.1, show accuracy as 0.9931 along with the best parameter to achieve it.

(c) K - Nearest neighbor (KNN)

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    model = KNeighborsClassifier()
    n_neighbors = range(10, 15, 5)
    weights = ['uniform', 'distance']
    metric = ['euclidean', 'manhattan', 'minkowski']
    param_grid_NB = dict(n_neighbors=n_neighbors,weights=weights,metric=metric)
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid_NB, n_jobs=-1)
    grid_result = grid_search.fit(pca_train_features, Y)
    print(grid_result.best_score_)
    print(grid_result.best_estimator_)
```

```
0.9931059983476274
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=10, p=2,
                     weights='uniform')
```

7.c.1: Codes used to build and train KNeighborsClassifier' with GridSearchCV and its best results.

7)(d) Model 4 - Linear Support Vector Machine (SVM):

The Linear Support Vector Machine (SVM) is a know as 'SVC' model from 'Sklearn' svm liberary. This performs a multi-class classification on the training dataset [18] , for linear 'SVC' we need to set the 'kernel' as 'linear'. Refer 7.d.1, show accuracy as 0.9426 along with the best parameter to achieve it.

(d) Linear SVM

```
[ ]  model = SVC()
     kernel = ['linear']
     C = [1.0, 0.1, 0.01]
     gamma = ['scale']
     grid = dict(kernel=kernel,C=C,gamma=gamma)
     grid_search_SVM = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1)
     grid_result_SVM = grid_search_SVM.fit(pca_train_features, Y)
     print(grid_result_SVM.best_score_)
     print(grid_result_SVM.best_estimator_)
```

```
0.942601262717252
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

7.d.1: Codes used to build and train linear 'SVC' with GridSearchCV and its best results.

7)(e) Model 5 - Naive Bayes Classifier:

The Naive Bayes Classifier is known as 'GaussianNB' from the 'Sklearn' naïve bayes library. It predicts the condition of every pair of features in the given dataset [19]. Refer 7.e.1, show accuracy as 0.9517 along with the best parameter to achieve it.

(e) Naive Bayes Classifier

```
[ ]  param_grid = {}
     model = GaussianNB()
     grid = GridSearchCV(estimator=model, param_grid = param_grid)
     grid.fit(pca_train_features, Y)
     print(grid.best_score_)
     print(grid.best_estimator_)
```

```
0.9517936107398184
GaussianNB(priors=None, var_smoothing=1e-09)
```

7.e.1: Codes used to build and train 'GaussianNB' with GridSearchCV and its best results.

8) Comparing the Accuracy of best trained model:

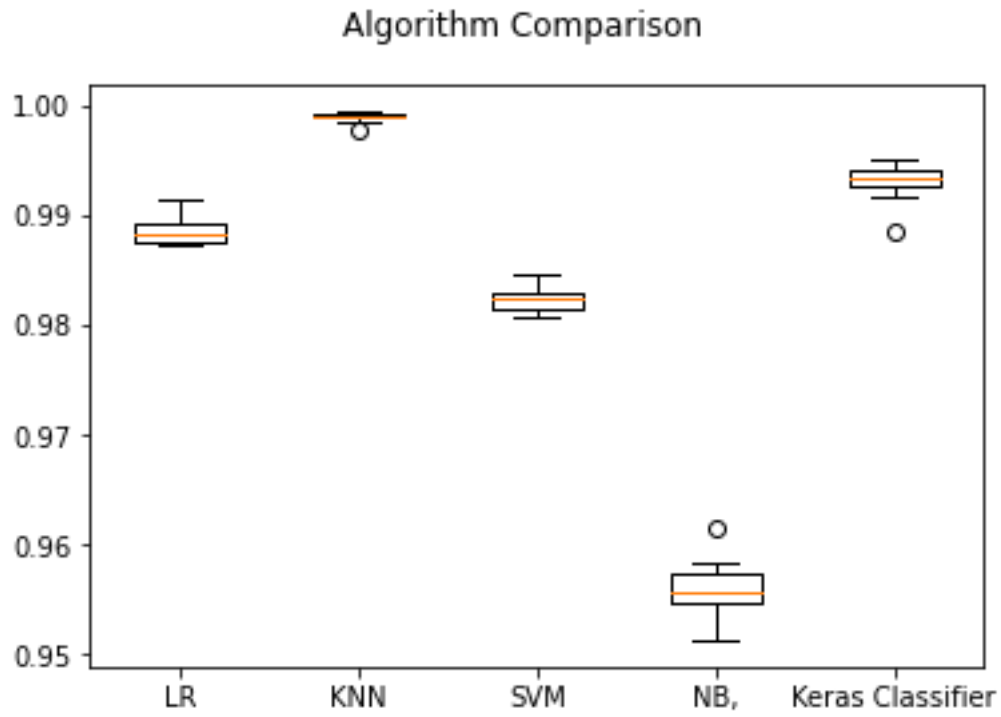
After achieving the results of the 'GridSearchCV' we can use the crossvalidation on hyperparameter tuned model and compare their performance accuracy. Refer 8.1 and 8.2 for the codes used and results achieved from it.

8) Comparing best models :

(a) Using Crossvalidation on hyperparameter tuned models and compare the performance accuracy

```
models = []
models.append(('LR', LogisticRegression(solver='liblinear',penalty='l2',C=100, max_iter=2000)))
models.append(('KNN', KNeighborsClassifier(n_neighbors=10,metric='euclidean',weights='uniform')))
models.append(('SVM', SVC(kernel='linear', C=1.0, gamma='scale')))
models.append(('NB,', GaussianNB(priors=None, var_smoothing=1e-09)))
models.append(('Keras Classifier', KerasClassifier(build_fn=create_model, batch_size= 40, epochs= 7)))
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
  kfold = KFold(n_splits=10, random_state=7,shuffle=True)
  cv_results = cross_val_score(model, pca_train_features, Y, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
# boxplot algorithm comparison
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

8.1: Codes used to cross-validate on hyperparameter tuned models



```
LR: 0.988541 (0.001213)
KNN: 0.998887 (0.000414)
SVM: 0.982379 (0.001111)
NB,: 0.955680 (0.002946)
Keras Classifier: 0.993024 (0.001804)
```

8.1: Comparison of model accuracy results obtained.

We can note that KNN and Keras Classifier are giving 99% accuracy followed by Linear Regression and SVM at 98% and Naïve Bayes at 95%

9) Predicting labels of test data instances and evaluating model:

Now we will predict the labels on test data instances and evaluate each model based the confusion matrix obtained and understand the following:
(i) Time taken to build model
(ii) Time taken to test model
(iii) Model Accuracy
(iv) Model Error Rate
(v) Detection Rate
(vi) False Positive/Alarm
(vii) Matthews correlation coefficient (MCC)
Refer 9.1 for the codes used to calculate all five models performance on test data instances.
Refer 9.2 for results obtained

```
models = []
models.append(('Logistic Regression', LogisticRegression(solver='liblinear',penalty='l2',C=100, max_iter=2000, random_state=0)))
models.append(('K - Nearest neighbor (KNN)', KNeighborsClassifier(n_neighbors=10,metric='euclidean',weights='uniform')))
models.append(('Linear SVM', SVC(kernel='linear', C=1.0, gamma='scale', random_state=0)))
models.append(('Naive Bayes Classifier', GaussianNB(priors=None, var_smoothing=1e-09)))
models.append(('Keras Classifier', KerasClassifier(build_fn=create_model, batch_size= 40, epochs= 7, init = 'glorot_uniform', optimizer='rmsprop')))


for name, model in models:
  print("-----------------------------------------------")
  print("Model : ", name)
  model_start = time.time()
  model.fit(pca_train_features, Y)
  print("Time to build model (sec) : %.4f " % round(time.time()-model_start,4))
  start = time.time()
  predicted = model.predict(pca_test_features)
  print("Time to test model (sec) : %.4f " % round(time.time()-start,4))
  matrix = confusion_matrix(Y_t, predicted)
  print("Time elapsed (sec): %.4f " % round(time.time()-model_start, 4))
  print(matrix)

  TN1 = matrix[0][0]
  FN1 = matrix[1][0]
  FP1 = matrix[0][1]
  TP1 = matrix[1][1]

  DetectionRate_LR = TP1/(TP1+FN1)
  Alarm_LR =   FP1/(FP1+TN1)

  # To built a MCC for LR
  MCC_num_LR= (TP1*TN1)-(FP1*FN1)
  MCC_din_LR= math.sqrt((TP1 + FP1)*(TP1+FN1)*(TN1 + FP1)*(TN1+FN1))

  MCC_LR = MCC_num_LR / MCC_din_LR
  Acc_LR = (TP1 + TN1) / (TP1+FP1+FN1+TN1)
  Err_LR = 1 - Acc_LR
  print("Model Accuracy : %s"%(Acc_LR))
  print("Model Error Rate : %s"%(Err_LR))
  print("DetectionRate :%s"%(DetectionRate_LR))
  print("False Positive/Alarm :%s"%(Alarm_LR))
  print("Matthews correlation coefficient (MCC):%s"%(MCC_LR))
```

9.1: Codes used to evaluate all five models and obtain the confusion matrix to draw conclusion

```
------------------------------------------------
Model :  Logistic Regression
Time to build model (sec) : 0.2251
Time to test model (sec) : 0.0023
Time elapsed (sec): 0.3018
[[19239   840]
 [  396 19683]]
Model Accuracy : 0.9692215747796205
Model Error Rate : 0.030778425220379546
DetectionRate :0.9802779022859704
False Positive/Alarm :0.041834752726729416
Matthews correlation coefficient (MCC):0.9386726687779009
------------------------------------------------
Model :  K - Nearest neighbor (KNN)
Time to build model (sec) : 0.1837
Time to test model (sec) : 2.0728
Time elapsed (sec): 2.2867
[[19967   112]
 [18651  1428]]
Model Accuracy : 0.5327705563026047
Model Error Rate : 0.4672294436973953
DetectionRate :0.07111907963544001
False Positive/Alarm :0.005577967030230589
Matthews correlation coefficient (MCC):0.17064777880156473
------------------------------------------------
Model :  Linear SVM
Time to build model (sec) : 39.9580
Time to test model (sec) : 4.2119
Time elapsed (sec): 44.2048
[[18904  1175]
 [   68 20011]]
Model Accuracy : 0.9690472633099257
Model Error Rate : 0.030952736690074256
DetectionRate :0.9966133771602171
False Positive/Alarm :0.058518850540365555
Matthews correlation coefficient (MCC):0.93952348348598
------------------------------------------------
Model :  Naive Bayes Classifier
Time to build model (sec) : 0.0235
Time to test model (sec) : 0.0081
Time elapsed (sec): 0.0622
[[19625   454]
 [ 1610 18469]]
Model Accuracy : 0.9486030180785896
Model Error Rate : 0.0513969819214104
DetectionRate :0.9198167239404352
False Positive/Alarm :0.022610687783256138
Matthews correlation coefficient (MCC):0.8986966837350798
------------------------------------------------
Model :  Keras Classifier
Epoch 1/10
3235/3235 [==============================] - 4s 1ms/step - loss: 0.3192 - accuracy: 0.9224
Epoch 2/10
3235/3235 [==============================] - 3s 1ms/step - loss: 0.0370 - accuracy: 0.9893
Epoch 3/10
3235/3235 [==============================] - 4s 1ms/step - loss: 0.0355 - accuracy: 0.9895
Epoch 4/10
3235/3235 [==============================] - 4s 1ms/step - loss: 0.0322 - accuracy: 0.9903
Epoch 5/10
3235/3235 [==============================] - 4s 1ms/step - loss: 0.0321 - accuracy: 0.9905
Epoch 6/10
3235/3235 [==============================] - 4s 1ms/step - loss: 0.0312 - accuracy: 0.9905
Epoch 7/10
3235/3235 [==============================] - 4s 1ms/step - loss: 0.0287 - accuracy: 0.9914
Epoch 8/10
3235/3235 [==============================] - 4s 1ms/step - loss: 0.0259 - accuracy: 0.9925
Epoch 9/10
3235/3235 [==============================] - 4s 1ms/step - loss: 0.0259 - accuracy: 0.9926
Epoch 10/10
3235/3235 [==============================] - 4s 1ms/step - loss: 0.0261 - accuracy: 0.9927
Time to build model (sec) : 36.5900
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/sequential.py:450: Us
  warnings.warn('`model.predict_classes()` is deprecated and '
Time to test model (sec) : 0.7704
Time elapsed (sec): 37.3929
[[19695   384]
 [18644  1435]]
Model Accuracy : 0.5261716220927337
Model Error Rate : 0.4738283779072663
DetectionRate :0.07146770257482943
False Positive/Alarm :0.01912445838936202
Matthews correlation coefficient (MCC):0.12585369653296963
```

9.2: Summary of Results obtained

10) Results Summarization and Observations:

	Logistic Regression	***K - Nearest Neighbors (KNN)***	***Linear SVM***	***Naive Bayes Classifier***	***Keras Classifier***
Time to build model (sec)	0.2251	0.1837	39.9580	0.0235	36.5900
Time to test model (sec)	0.0023	2.0728	4.2119	0.0081	0.7704
Time elapsed (sec)	0.3018	2.2867	44.2048	0.0622	37.3929
Model Accuracy	0.9692	0.5327	0.9690	0.9486	0.5261
Model Error Rate	0.0307	0.4672	0.0309	0.0513	0.4738
Detection Rate	0.9802	0.0711	0.9966	0.9198	0.0714
False Positive/Alarm	0.0418	0.0055	0.0585	0.0226	0.0191
Type II Error	0.0197	0.9288	0.0033	0.0801	12.9923
Matthews correlation coefficient (MCC)	0.9386	0.1706	0.9395	0.8986	0.1258

10.1: Summary of Results achieved on models by the testing instances

10)(a) Observation:

(i) Time to Taken to build model (TBM): With the above observation we can note that 'Nive Bayes Classifier" model had the quickest time to build followed by 'Logistic Regression' model, whereas 'Linear SVM' took the longest to build followed by 'Keras Classifier' model.

(ii) Time to Test model (TTM): With the above observation we can note that 'Nive Bayes Classifier" model had the quickest time to build followed by 'Logistic Regression' model, whereas 'Linear SVM' models took the longest to build followed by 'K- Nearest Neighbors' model.

(iii) Model Accuracy and Error Rate: With the above observation we can note that 'Linear SVM' & 'Logistic Regression' have a similar and highest accuracy and error rate, whereas 'Keras Classifier' & 'K- Nearest Neighbors' models performed poorly.

(iv) Detection Rate: With the above observation we can note that 'Naive Bayes Classifier" model had the highest detection rate followed by 'Logistic Regression', whereas Keras Classifier' & 'K- Nearest Neighbors' models performed poorly.

(v) False Positive/Alarm and Type II Error: With the above observation we can note that 'Nive Bayes Classifier" model had the highest False alarm rate and Lowest Type II Error followed by 'Logistic Regression', whereas Keras Classifier' & 'K- Nearest Neighbors' models performed poorly.

(vi) *Matthews correlation coefficient (MCC):* With the above observation we can note that 'Linear SVM', 'Logistic Regression' and 'Naïve Bayes Classifier' have a values closer to 1, this means that it represents a perfect predictions. On the other hand, 'Keras Classifier' & 'K-Nearest Neighbors' have a value closer to 0, this means that it is equivalent to random predictions.

10)(b) Summary:

From the above observations we can note that 'Linear SVM' and 'Logistic Regression' models are performing best on the given dataset. The Impact Paper also by Seo et al [4] mentions that

"In comparison with other mod- els using different classifiers or SVM, (providing higher training time on the AWID impersonation dataset), the IMPACT demonstrated better performance including much lower FAR compared to DEMISe models."

Apart from that, I observed that Keras Classifier Model was giving different results for very execution. The reason being that the classifier randomly assigns weights to the neurons and thus it tends to give inconsistent results because of its stochastic nature. Although we implemented a 'custom_loss' function and tuned its parameter, it did not reap fruitful benefits after spent a lot of time creating/customizing it.

Reference:

[1] N. Ye, Y. Zhang, and C. M. Borror, “Robustness of the Markov-chain model for cyber-attack detection,” *IEEE trans. reliab.*, vol. 53, no. 1, pp. 116–123, 2004.

[2] C. C. Editor, “Cyber Attack - Glossary,” Nist.gov. [Online]. Available: https://csrc.nist.gov/glossary/term/Cyber_Attack. [Accessed: 11-Jan-2021].

[3] L. R. Parker, P. D. Yoo, T. A. Asyhari, L. Chermak, Y. Jhi, and K. Taha, “DEMISe: Interpretable deep extraction and mutual information selection techniques for IoT intrusion detection,” in *Proceedings of the 14th International Conference on Availability, Reliability and Security - ARES ’19*, 2019.

[4] S. J. Lee et al., “IMPACT: Impersonation attack detection via edge computing using deep autoencoder and feature abstraction,” IEEE Access, vol. 8, pp. 65520–65529, 2020.

[5] “pandas.read_csv — pandas 1.2.0 documentation,” Pydata.org. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html. [Accessed: 12-Jan-2021].

[6] “pandas.DataFrame.describe — pandas 1.2.0 documentation,” Pydata.org. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html. [Accessed: 12-Jan-2021].

[7] C. Ordonez, “Data set preprocessing and transformation in a database system,” Intell. Data Anal., vol. 15, no. 4, pp. 613–631, 2011.

[8] “sklearn.preprocessing.Normalizer — scikit-learn 0.24.0 documentation,” Scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html. [Accessed: 12-Jan-2021].

[9] Keras Team, “Variational AutoEncoder,” Keras.io. [Online]. Available: https://keras.io/examples/generative/vae/. [Accessed: 15-Jan-2021].

[10] “numpy.concatenate — NumPy v1.19 Manual,” Numpy.org. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.concatenate.html. [Accessed: 12-Jan-2021].

[11] “1.13. Feature selection — scikit-learn 0.24.0 documentation,” Scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/feature_selection.html. [Accessed: 12-Jan-2021].

[12] “sklearn.decomposition.PCA — scikit-learn 0.24.0 documentation,” Scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html. [Accessed: 12-Jan-2021].

[13] “sklearn.model_selection.GridSearchCV — scikit-learn 0.24.0 documentation,” Scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. [Accessed: 12-Jan-2021].

[14] “tf.keras.wrappers.scikit_learn.KerasClassifier,” *Tensorflow.org*. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/wrappers/scikit_learn/KerasClassifier. [Accessed: 12-Jan-2021].

[15] E. Zakkay, “Advanced Keras — constructing complex custom losses and metrics,” Towards Data Science, 10-Jan-2019. [Online]. Available: https://towardsdatascience.com/advanced-keras-constructing-complex-custom-losses-and-metrics-c07ca130a618. [Accessed: 13-Jan-2021].

[16] “sklearn.linear_model.LogisticRegression — scikit-learn 0.24.0 documentation,” Scikit-learn.org. [Online]. Available: https://scikit-

learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. [Accessed: 13-Jan-2021].

[17] "1.6. Nearest Neighbors — scikit-learn 0.24.0 documentation," Scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html. [Accessed: 13-Jan-2021].

[18] "sklearn.svm.SVC — scikit-learn 0.24.0 documentation," Scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. [Accessed: 13-Jan-2021].

[19] "1.9. Naive Bayes — scikit-learn 0.24.0 documentation," Scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html. [Accessed: 13-Jan-2021].

Appendix 1: Wrapper Method Output

RFE output

```
---------------------------------------------------
Model :  LR
Time to build model (sec) : 2.3170
Time to test model (sec) : 0.0015
Time elapsed (sec): 2.3849
[[19653   426]
 [11328  8751]]
Model Accuracy : 0.707306140744061
Model Error Rate : 0.29269385925593905
DetectionRate :0.9787838039743015
False Positive/Alarm :0.5641715224861796
Matthews correlation coefficient (MCC):0.49372618456099016
---------------------------------------------------
Model :  KNN
Time to build model (sec) : 0.2112
Time to test model (sec) : 3.3507
Time elapsed (sec): 3.5973
[[19861   218]
 [19870   209]]
Model Accuracy : 0.4997758852532497
Model Error Rate : 0.5002241147467503
DetectionRate :0.9891428856018726
False Positive/Alarm :0.9895911150953732
Matthews correlation coefficient (MCC):-0.0021850622995254233
---------------------------------------------------
Model :  SVM
Time to build model (sec) : 11.6375
Time to test model (sec) : 1.7474
Time elapsed (sec): 13.4242
[[19506   573]
 [16014  4065]]
Model Accuracy : 0.5869565217391305
Model Error Rate : 0.4130434782608695
DetectionRate :0.9714627222471238
False Positive/Alarm :0.797549678768863
Matthews correlation coefficient (MCC):0.27206481775610425
---------------------------------------------------
Model :  NB,
Time to build model (sec) : 0.0271
Time to test model (sec) : 0.0092
Time elapsed (sec): 0.0718
[[19577   502]
 [ 1831 18248]]
Model Accuracy : 0.9419044773146072
Model Error Rate : 0.05809552268539275
DetectionRate :0.9749987549180736
False Positive/Alarm :0.09118980028885901
Matthews correlation coefficient (MCC):0.8857512891822353
---------------------------------------------------
Model :  Keras Classifier
Epoch 1/5
3235/3235 [==============================] - 4s 1ms/step - loss: 1.7435 - accuracy: 0.9534
Epoch 2/5
3235/3235 [==============================] - 4s 1ms/step - loss: 0.2729 - accuracy: 0.9893
Epoch 3/5
3235/3235 [==============================] - 4s 1ms/step - loss: 0.2274 - accuracy: 0.9913
Epoch 4/5
3235/3235 [==============================] - 4s 1ms/step - loss: 0.1990 - accuracy: 0.9921
Epoch 5/5
3235/3235 [==============================] - 4s 1ms/step - loss: 0.1692 - accuracy: 0.9930
Time to build model (sec) : 20.4691
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWa
after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,  if your model does m
activation).* `(model.predict(x) > 0.5).astype("int32")`,  if your model does binary classification  (e.g.
  warnings.warn('`model.predict_classes()` is deprecated and '
Time to test model (sec) : 0.7566
Time elapsed (sec): 21.2638
[[19703   376]
 [18667  1412]]
Model Accuracy : 0.5257980975148164
Model Error Rate : 0.47420190248518357
DetectionRate :0.981273967827083
False Positive/Alarm :0.9296777727974501
Matthews correlation coefficient (MCC):0.12507776105360394
```

Appendix 2: Embedded Method output

```
Extra Tree Classifier :
-------------------------------------------------
Model :  LR
Time to build model (sec) : 3.7359
Time to test model (sec) : 0.0015
Time elapsed (sec): 3.8034
[[19492   587]
 [18484  1595]]
Model Accuracy : 0.5251008516360377
Model Error Rate : 0.4748991483639623
DetectionRate :0.970765476368345
False Positive/Alarm :0.9205637730962697
Matthews correlation coefficient (MCC):0.11073337826785293
-------------------------------------------------
Model :  KNN
Time to build model (sec) : 0.2403
Time to test model (sec) : 3.4531
Time elapsed (sec): 3.7339
[[19915   164]
 [18704  1375]]
Model Accuracy : 0.5301558842571841
Model Error Rate : 0.4698441157428159
DetectionRate :0.9918322625628766
False Positive/Alarm :0.9315204940485083
Matthews correlation coefficient (MCC):0.15708124011098562
-------------------------------------------------
Model :  SVM
Time to build model (sec) : 86.1739
Time to test model (sec) : 3.3022
Time elapsed (sec): 89.5125
[[19471   608]
 [18751  1328]]
Model Accuracy : 0.517929179740027
Model Error Rate : 0.48207082025997305
DetectionRate :0.9697196075501768
False Positive/Alarm :0.933861248070123
Matthews correlation coefficient (MCC):0.083699548611644
-------------------------------------------------
Model :  NB,
Time to build model (sec) : 0.0292
Time to test model (sec) : 0.0096
Time elapsed (sec): 0.0739
[[19745   334]
 [ 5452 14627]]
Model Accuracy : 0.8559191194780617
Model Error Rate : 0.14408088052193835
DetectionRate :0.9833657054634195
False Positive/Alarm :0.2715274665072962
Matthews correlation coefficient (MCC):0.7361539796934943
-------------------------------------------------
Model :  Keras Classifier
Epoch 1/5
3235/3235 [==============================] - 4s 1ms/step - loss: 1.9467 - accuracy: 0.9228
Epoch 2/5
3235/3235 [==============================] - 4s 1ms/step - loss: 0.5266 - accuracy: 0.9746
Epoch 3/5
3235/3235 [==============================] - 4s 1ms/step - loss: 0.4428 - accuracy: 0.9798
Epoch 4/5
3235/3235 [==============================] - 4s 1ms/step - loss: 0.3650 - accuracy: 0.9852
Epoch 5/5
3235/3235 [==============================] - 4s 1ms/step - loss: 0.3061 - accuracy: 0.9884
Time to build model (sec) : 20.3823
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserW
be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your
`softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does bina
activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
Time to test model (sec) : 1.0351
Time elapsed (sec): 21.4611
[[19461   618]
 [18756  1323]]
Model Accuracy : 0.5175556551621097
Model Error Rate : 0.48244434483789034
DetectionRate :0.9692215747796205
False Positive/Alarm :0.9341102644554011
Matthews correlation coefficient (MCC):0.08185553533464951
```