Artificial Intelligence Project Report

Group Members

Farhan Abbasi (20P-0044)

Usama Yazdani (20P-0598)

Section

BCS-6C

Introduction

This AI project is about running the data set on different classification algorithms. The data set deals with the types of cyber attacks having 43 feature columns and 125973 rows in them. The name and types of attacks are provided in a separate data set. The data set will be pre-processed which includes data cleaning and handling unnecessary details. After the pre-processing, the data in the form of vectors will be passed to the classifiers to be trained and produce predictions. The metrics score will be calculated based on predictions produced by the classifiers through which the performance of the classifiers can be evaluated.

Executive Summary

The cyber attack data set consists of two files one is the actual data set which needs to be trained and the other file contains the target label. The data visualization and cleaning are performed using Python's pandas library.

Correlation analysis of features is done against the target label to find out which features are more contributing to the target label.

After the data is preprocessed and cleaned, the data is split into training and testing data with 70 training and 30 percent testing data.

After splitting, 4 different classifying algorithms are being used which are listed below

Decision Tree: it is a type of classification algorithm used for the supervised data set. It produces a tree-like structure in the processing of the data set to reach a specific label.

K-Nearest Neighbour: This classification algorithm is a nonparametric supervised learning algorithm that classifies the data points based on the closest K number of data points.

Artificial Neural Network (MultiLayer Perceptron): it is a supervised machine learning algorithm that has multiple layers of interconnected perceptrons which compute the weight of its inputs and apply an activation function to it (sigmoid, ReLu, etc)

K-Means Algorithm: It is a clustering-based machine learning algorithm used for unsupervised data sets. It clusters similar data points together based on the features given to it.

The Metric Scores (Accuracy, F1 Score, Recall Score, Precision Score) is being calculated for the first three algorithms i.e Decision Tree, KNN, and ANN, and results are visualized in the line graph

Libraries Used:

Numpy

Pandas

Matplotlib

Sklearn

Data Preprocessing (Extraction and Cleaning)

In data preprocessing, we used the pandas library for the visualization of the data set. The data set contained no null values.

```
df.isnull().sum() # no null values
```

```
duration                     0
protocol_type                0
service                      0
flag                         0
src_bytes                    0
dst_bytes                    0
land                         0
wrong_fragment               0
urgent                       0
hot                          0
num_failed_logins            0
logged_in                    0
num_compromised              0
root_shell                   0
su_attempted                 0
num_root                     0
num_file_creations           0
num_shells                   0
num_access_files             0
num_outbound_cmds            0
is_host_login                0
is_guest_login               0
count                        0
srv_count                    0
serror_rate                  0
srv_serror_rate              0
rerror_rate                  0
srv_rerror_rate              0
same_srv_rate                0
diff_srv_rate                0
srv_diff_host_rate           0
dst_host_count               0
dst_host_srv_count           0
```

Since there were no null values, we encoded the categorical values and string values. The column ‘protocol_type’ contained 3 categorical values, so they were being encoded to [0,1,2]

```
df['protocol_type'] = df['protocol_type'].replace("tcp" ,0)  # encoding
df['protocol_type'] = df['protocol_type'].replace("udp" , 1)  #  encoding
df['protocol_type'] = df['protocol_type'].replace("icmp",2)  # encoding
df['protocol_type'].unique()
```

And other columns i.e 'flag' and 'service' containing string values were replaced by numerical values

```
for i in range(len(df['service'].unique())):  # converitng string to numeric
    df['service'] = df['service'].replace(df['service'].unique()[i] ,i)
```

```
for i in range(len(df['flag'].unique())):  # converitng string to numeric
    df['flag'] = df['flag'].replace(df['flag'].unique()[i] ,i+100)
```

Since target labels were in different text files, a separate dictionary was made containing target labels of the data set which were mapped against the column 'attack_category' which contained the names of the attacks.

```
#dictionary for mapping of attack categories
cyber_attack_types = {'normal':'normal','neptune':'dos',
            'warezclient':'r21' , 'ipsweep':'probe' ,
          'teardrop':'dos' , 'portsweep':'probe' ,
                     'nmap':'probe' , 'satan':'probe' ,
                     'smurf':'dos' , 'pod':'dos' , 'back':'dos' ,
                     'guess_passwd':'r21' , 'ftp_write':'r21' ,
                     'multihop':'r21' , 'rootkit':'u2r' ,
                     'httptunnel':'u2r' ,'snmpguess':'r21' ,
                     'imap':'r21' , 'spy':'r21' , 'warezmaster':'r21' ,
                     'phf':'r21' , 'named':'r21' ,'sendmail':'r21' ,
                     'xlock':'r21' , 'xsnoop':'r21', 'worm':'r21',
                     'mscan':'probe' ,'saint':'probe' , 'worm' :'probe' ,
                     'buffer_overflow':'u2r' , 'loadmodule':'u2r',
                     'land':'dos' ,'perl':'u2r'
         }
```

```
df['target_label'] = df['attack_category'] # adding target label to the data frame
```

```
df['target_label'] = df['target_label'].map(cyber_attack_types) #mapping values in dataframe
```

After mapping, the encoding of labels was performed into 5 categories and mapped to the target label

```
#encoding the target labels
enc_labels = {
    'normal' : 0 ,
    'dos' : 1 ,
    'r2l' : 2 ,
    'probe' : 3 ,
    'u2r' : 4
}
```

```
df['target_label'] = df['target_label'].map(enc_labels) # mapping the encoded labels
```

Scaling:

The feature having large values are downscaled for better accuracy.

Columns 'service', 'src_bytes' 'flag', 'dst_bytes', and 'occurrence' are standard scaled using a built-in scalar function from the sklearn preprocessing library.

Scaling Features

```
scaler = StandardScaler()
df[['service','src_bytes','flag','dst_bytes','occurance']]=scaler.fit_transform(df[['service','src_bytes','flag','dst_bytes','occ
```

Feature Engineering

The feature engineering Is done using correlation analysis.

Correlation Analysis:

Correlation analysis was performed to find out the importance of the features helping to determine the target label. Feature correlation values are sorted in descending order. We picked the top 7 features for the model training.

All other features are dropped from the data set to reduce the data set.

```
correlation_matrix = df.corr()
corr_with_target = correlation_matrix['target_label']

sorted_features = corr_with_target.abs().sort_values(ascending=False)

print(sorted_features)
```

```
target_label                 1.000000
dst_host_srv_count           0.546923
logged_in                    0.538431
occurance                    0.532578
dst_host_diff_srv_rate       0.480732
dst_host_same_srv_rate       0.442043
dst_host_same_src_port_rate  0.403252
flag                         0.398071
same_srv_rate                0.394552
dst_host_srv_rerror_rate     0.346545
srv_rerror_rate              0.345624
rerror_rate                  0.343777
diff_srv_rate                0.332775
dst_host_srv_diff_host_rate  0.323976
dst_host_rerror_rate         0.321275
```

Selecting Top 7 features

```
features_and_target = ['dst_host_srv_count', 'logged_in', 'occurance',
            'dst_host_diff_srv_rate' , 'dst_host_same_srv_rate'
            ,'dst_host_same_src_port_rate','flag','target_label']
```

Dropping unimportant features

```
df = df.drop(columns=[col for col in df.columns if col not in features_and_target])

df
```

These are the selected features

```
features = ['dst_host_srv_count', 'logged_in', 'occurance',
            'dst_host_diff_srv_rate' , 'dst_host_same_srv_rate'
            ,'dst_host_same_src_port_rate','flag']  # separating features from target
```

The data is now cleaned!

Use of Classification And Clustering Algorithms

Before passing the data into the model we first need to split the data into training and testing data. Our split ratio was 70/30 i.e. 70 percent training and 30 percent testing data with a random state of 10 for shuffling data to prevent biases.

```
X = df[features]  # features / attributes
y = df['target_label']   # target label
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)   # splitting on data 70/30 ratio
```

Decision Tree Algorithm

The decision tree algorithm is implemented on the data using the sklearn decision tree classifier. it is a supervised learning model, that makes a tree based on information gain and selects the nodes in a tree accordingly.

An object for the decision tree is made with parameter values passed to it. The criterion was set to ‘entropy’ which measures the impurity of a node, max depth = 4 which means a decision tree can have the maximum level of 4, and random seed = 10 for shuffling.

Now X_train and y_train are passed to the fit function of the object which trains the model and predictions are generated.

Decision Tree

```
Decision_tree_classifier = DecisionTreeClassifier(criterion = 'entropy' ,max_depth = 4 , random_state = 10 )
```

```
Decision_tree_classifier.fit(X_train , y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=10)
```

```
dct_prediction = Decision_tree_classifier.predict(X_test)
```

A visual representation of a tree is generated using matplotlib where the root is ‘flag’ having the highest information gain.


flag <= -0.119
entropy = 1.388
samples = 88181
value = [47182, 32199, 665, 8099, 36]
occurance <= -0.002
entropy = 0.827
samples = 52470
value = [44386, 3249, 627, 4173, 35]
dst_host_diff_srv_rate <= 0.145
entropy = 0.894
samples = 35711
value = [2796, 28950, 38, 3926, 1]
dst_host_same_src_port_rate <= 0.415
entropy = 1.705
samples = 14754
value = [7020, 3049, 627, 4023, 35]
logged_in <= 0.5
entropy = 0.085
samples = 37716
value = [37366, 200, 0, 150, 0]
dst_host_same_srv_rate <= 0.575
entropy = 0.49
samples = 31756
value = [2699, 28806, 37, 213, 1]
dst_host_same_srv_rate <= 0.055
entropy = 0.394
samples = 3955
value = [97, 144, 1, 3713, 0]
occurance <= -1.31
entropy = 1.182
samples = 8432
value = [6104, 1628, 251,
logged_in <= 0
entropy = 1.62
samples = 63
value = [916, 1421, 376
dst_host_same_src
entropy
samples
value = [6175,
value = [3
occurance <= -0.875
entropy = 0.15
samples = 29193
value = [327, 28659, 1, 206
occurance <= -1.747
entropy = 0.454
samples = 2563
value = [2372, 147, 36,
dst_host_diff_srv_rate <= 0.1
entropy = 0.063
samples = 3647
value = [8, 16, 0, 3623, 0]
flag <= 0.856
entropy = 1.589
samples = 308
value = [89, 128, 1, 90, 0]
entropy
samples
value = [450, 1
value = [56
entropy
samples
value = [686, 1
entropy
samples
value = [230,
value = [36
entropy = 1.312
samples = 181
value = [81, 11, 1, 88, 0]

K-Nearest Neighbour:

This classification algorithm is a non-parametric supervised learning algorithm that classifies the data points based on the closest K number of data points.

Repeating the same process as above, we create an object of the KNN model and initially, we take k = 5, meaning 5 nearest neighbors of a data point. Now we trained the data on the fit function. The accuracy was 0.967

K-Nearest Neighbour (KNN)

```
X = df[features]  # features / attributes
y = df['target_label']   # target label
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

```
KNN = KNeighborsClassifier(n_neighbors=5)
```

```
KNN.fit(X_train , y_train)
```

```
KNeighborsClassifier()
```

```
knn_prediction = KNN.predict(X_test)
```

```
accuracy = accuracy_score(y_test, knn_prediction)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9670829805249789
```

Now we searched for the optimal value of K, in a loop and trained the data on different values of K ranging from 1 to 10, and accuracy was calculated on each iteration

```
accuracies = []
k_values = []
for i in range(1,11):
    KNN = KNeighborsClassifier(n_neighbors=i)
    KNN.fit(X_train , y_train)
    knn_prediction = KNN.predict(X_test)


    #Accuracy
    accuracy = accuracy_score(y_test, knn_prediction)
    accuracies.append(accuracy)
    k_values.append(i)
    print("Accuracy at k = ",i," ", accuracy)
```

```
Accuracy at k =  1   0.9730366215071973
Accuracy at k =  2   0.968088484335309
Accuracy at k =  3   0.9698613463166808
Accuracy at k =  4   0.9684060118543607
Accuracy at k =  5   0.9671094411515665
Accuracy at k =  6   0.9658657917019475
Accuracy at k =  7   0.9642516934801016
Accuracy at k =  8   0.9634314140558848
Accuracy at k =  9   0.9631403471634208
Accuracy at k =  10   0.9624259102455546
```

We cant take k= 1 because setting k to 1 can result in overfitting to the training data and make the model more susceptible to noise and outliers in the data. Increasing the value of k can improve the generalization ability of the model, but it may also result in a higher bias and lower model complexity

So we took k = 3, the second highest accuracy result.

ANN (Multi-Layer Perceptron):

After splitting data, we made an object of the Multilayer Perceptron and adjusted the values of the hyperparameters as follows: solver = 'lbfgs' which is an optimizer of the quasi-newton family for the weights that are calculated, and alpha = 1e-5 which is a very small value, it is a regularization term which prevents the data from overfitting.

hidden_layer_sizes = (5,3,3) which means we have 3 layers the first one will have 5 perceptrons and the other two have 3 perceptrons each. After that data is trained and predictions are made.

ANN (Multilayer Perceptron)

```
X = df[features]  # features / attributes
y = df['target_label']   # target label
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)   # splitting on data 70/30 ratio
```

```
Multilayer_perceptron = MLPClassifier(solver='lbfgs' ,alpha = 1e-5, hidden_layer_sizes = (5,3,3) , random_state = 10)
```

```
Multilayer_perceptron.fit(X_train , y_train)
```

```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 3, 3), random_state=10,
              solver='lbfgs')
```

```
mlp_prediction =  Multilayer_perceptron.predict(X_test)
```

The following metrics scores were calculated after the above training:

```
Accuracy = accuracy_score(y_test, mlp_prediction)
F1_score = f1_score(y_test, mlp_prediction,average = 'weighted')
Recall_score = recall_score(y_test, mlp_prediction,average = 'weighted')
Precision_score = precision_score(y_test, mlp_prediction, average = 'weighted')

print("Accuracy score: ", Accuracy )
print("F1 score: ",F1_score)
print("Recall Score: ",Recall_score)
print("Precision score: ",Precision_score)
```

```
Accuracy score:  0.9154847586790855
F1 score:  0.9120403925019586
Recall Score:  0.9154847586790855
Precision score:  0.9116028961209518
```

Fine Tuning:

We changed the hyperparameters to fine-tune the model. Solver = ‘adam’ which is a more accurate optimizer and hidden layer sizes were changed to (4,4,6).

Fine Tuning

```
Multilayer_perceptron = MLPClassifier(solver='adam' ,alpha = 1e-5, hidden_layer_sizes = (4,4,6) , random_state = 1)
```

```
Multilayer_perceptron.fit(X_train , y_train)
```

```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(4, 4, 6), random_state=1)
```

```
mlp_prediction =  Multilayer_perceptron.predict(X_test)
```

The metrics score changed from 0.90 to 0.95

Metric Scores(After Tuning)

```
Accuracy = accuracy_score(y_test, mlp_prediction)
F1_score = f1_score(y_test, mlp_prediction,average = 'weighted')
Recall_score = recall_score(y_test, mlp_prediction,average = 'weighted')
Precision_score = precision_score(y_test, mlp_prediction, average = 'weighted')

print("Accuracy score: ", Accuracy )
print("F1 score: ",F1_score)
print("Recall Score: ",Recall_score)
print("Precision score: ",Precision_score)
```

```
Accuracy score:  0.9583245131244708
F1 score:  0.9581890827029542
Recall Score:  0.9583245131244708
Precision score:  0.9580755478271874
```

K-Means Clustering Algorithm:

We first dropped the target column from the data set.

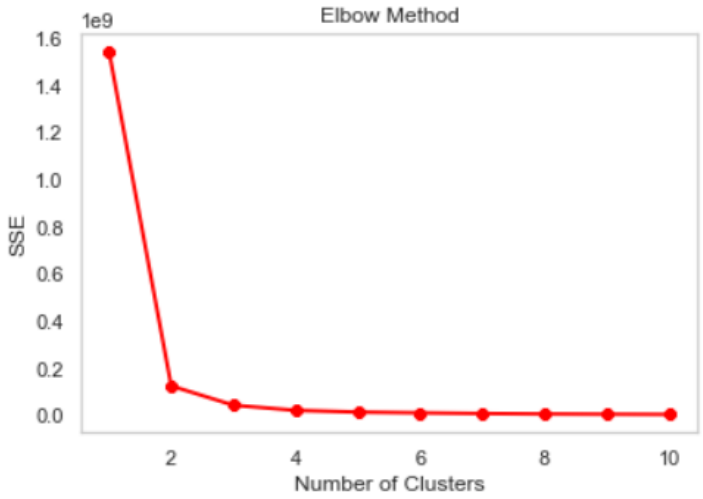
```
df1 = df.drop(columns='target_label') # dropping the target label
```

We made a list of sse (Sum Of Squared Error) and appended means inertia which is a metric used to evaluate the quality of clusters produced by the k-means algorithm. It is a measure of how internally coherent the clusters are. We ran a loop a from 1 to 10 which was the value of k(clusters)

```
sse = []
for k in range(1,11):
    kmeans = KMeans(n_clusters=k, random_state = 10)
    kmeans.fit(df1)
    sse.append(kmeans.inertia_)
```

Now using the elbow method we plot a graph for the optimum value of K where the error is less and selected the value of K.

```
number_of_clusters = range(1,11)
plt.grid()
plt.plot(number_of_clusters , sse , linewidth = 2 , color = 'red' , marker = "8")
plt.title('Elbow Method')
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```



Since we had 7 features we performed PCA (Principal Component Analysis) to reduce the dimension of the features to 2 so they can be plotted in a scatter graph. Where red dots are centroids which represent the center of the cluster

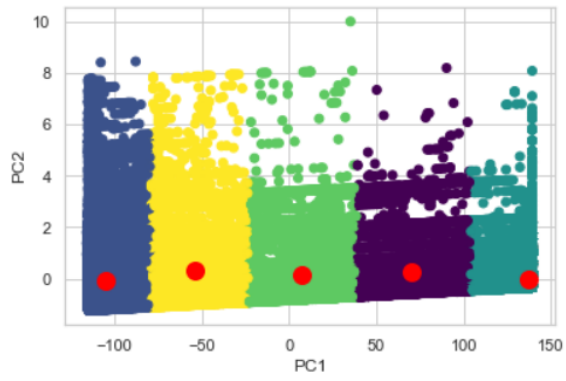
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(df1)  # Perform PCA to reduce the number of dimensions to 2


kmeans = KMeans(n_clusters=5 , init='k-means++', random_state=10)
kmeans.fit(X_pca)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_    # Perform K-means clustering

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis')
plt.xlabel('PC1')
plt.ylabel('PC2')

# Add cluster centroids to the plot
plt.scatter(centroids[:, 0], centroids[:, 1], marker='o', s=100, linewidths=3, color='red', zorder=10)

plt.show()
```



Comparison and Performance Evaluation

Performance Evaluation of Algorithms

Accuracy, F1 Score, Recall Score, and Precision Score are calculated based on predictions produced by the models.

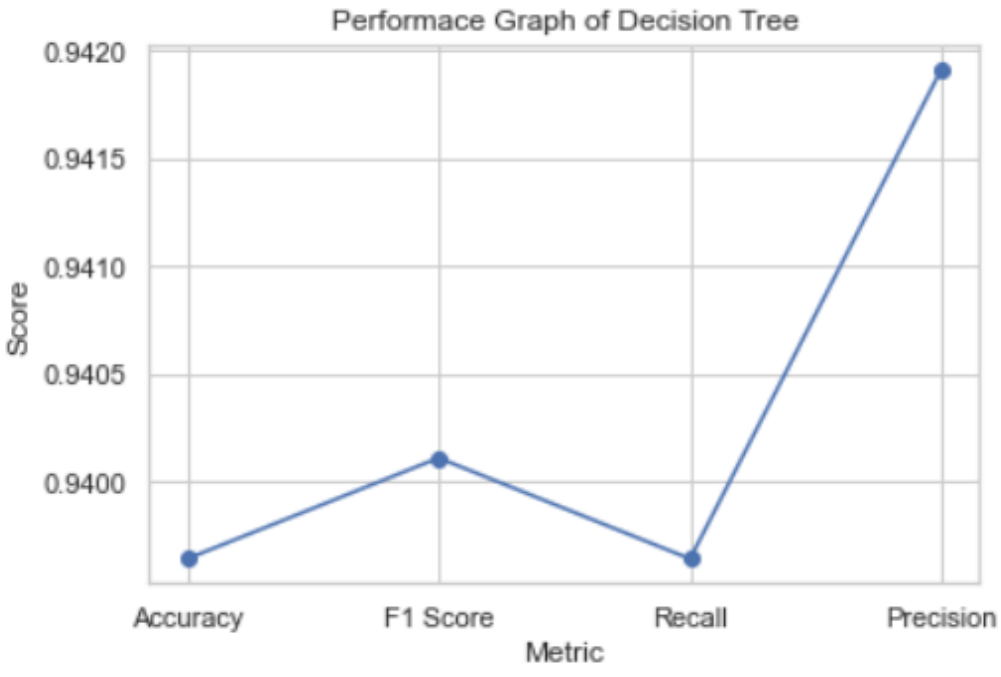
Decision Tree:

```
Accuracy = accuracy_score(y_test, dct_prediction)
F1_score = f1_score(y_test, dct_prediction,average = 'weighted')
Recall_score = recall_score(y_test, dct_prediction,average = 'weighted')
Precision_score = precision_score(y_test, dct_prediction, average = 'weighted')

print("Accuracy score: ", Accuracy )
print("F1 score: ",F1_score)
print("Recall Score: ",Recall_score)
print("Precision score: ",Precision_score)
```

```
Accuracy score:  0.9400137595258256
F1 score:  0.9406491114246106
Recall Score:  0.9400137595258256
Precision score:  0.9425587802646751
```

Line Graph:



KNN:

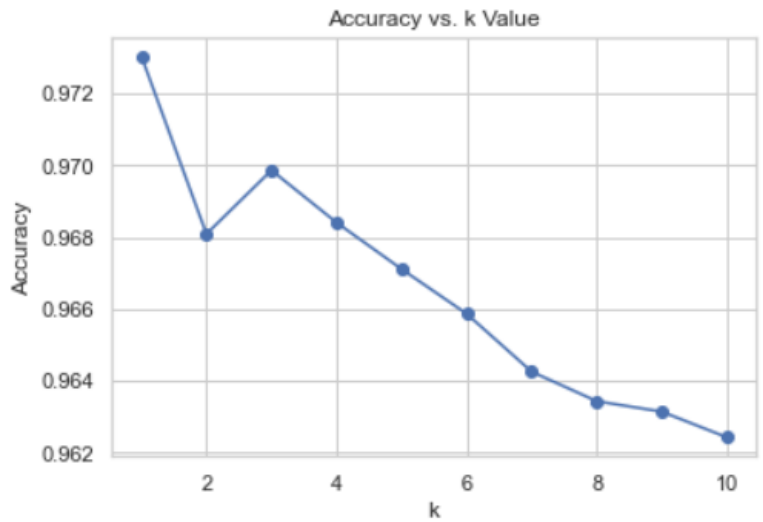
Accuracies of different values of K

```
accuracies = []
k_values = []
for i in range(1,11):
    KNN = KNeighborsClassifier(n_neighbors=i)
    KNN.fit(X_train , y_train)
    knn_prediction = KNN.predict(X_test)


    #Accuracy
    accuracy = accuracy_score(y_test, knn_prediction)
    accuracies.append(accuracy)
    k_values.append(i)
    print("Accuracy at k = ",i," ", accuracy)
```

```
Accuracy at k =  1   0.9730366215071973
Accuracy at k =  2   0.968088484335309
Accuracy at k =  3   0.9698613463166808
Accuracy at k =  4   0.9684060118543607
Accuracy at k =  5   0.9671094411515665
Accuracy at k =  6   0.9658657917019475
Accuracy at k =  7   0.9642516934801016
Accuracy at k =  8   0.9634314140558848
Accuracy at k =  9   0.9631403471634208
Accuracy at k =  10   0.9624259102455546
```

Line Graph:



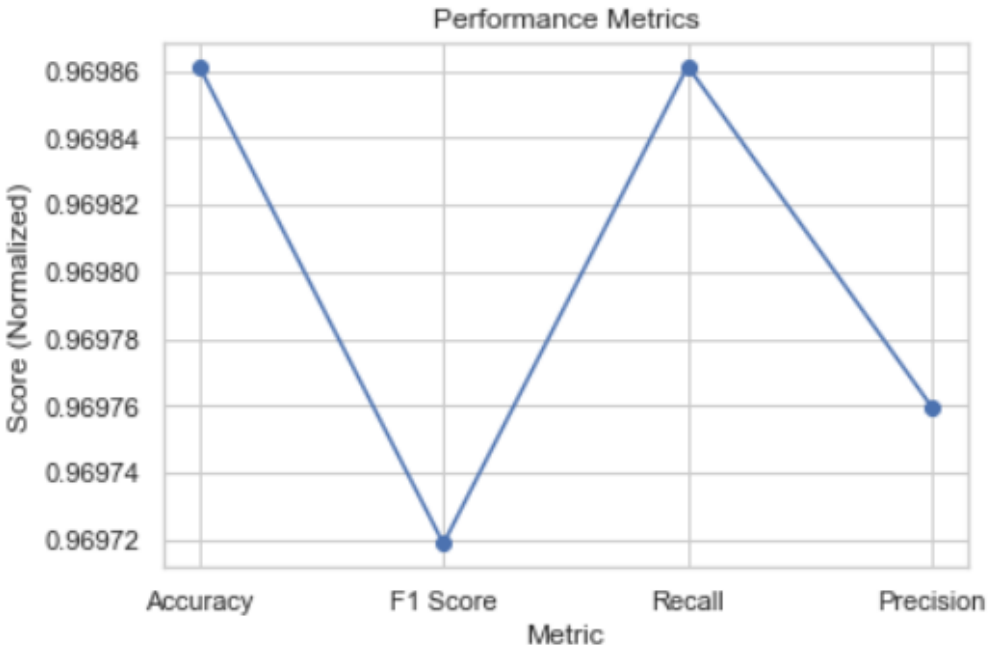
Performance on K = 3:

```
Accuracy = accuracy_score(y_test, knn_prediction)
F1_score = f1_score(y_test, knn_prediction,average = 'weighted')
Recall_score = recall_score(y_test, knn_prediction,average = 'weighted')
Precision_score = precision_score(y_test, knn_prediction, average = 'weighted')

print("Accuracy score: ", Accuracy )
print("F1 score: ",F1_score)
print("Recall Score: ",Recall_score)
print("Precision score: ",Precision_score)
```

```
Accuracy score:  0.9698613463166808
F1 score:  0.9697187634971518
Recall Score:  0.9698613463166808
Precision score:  0.9697594826259062
```

Line Graph:



Multi-Layer Perceptron:

Before fine Tuning

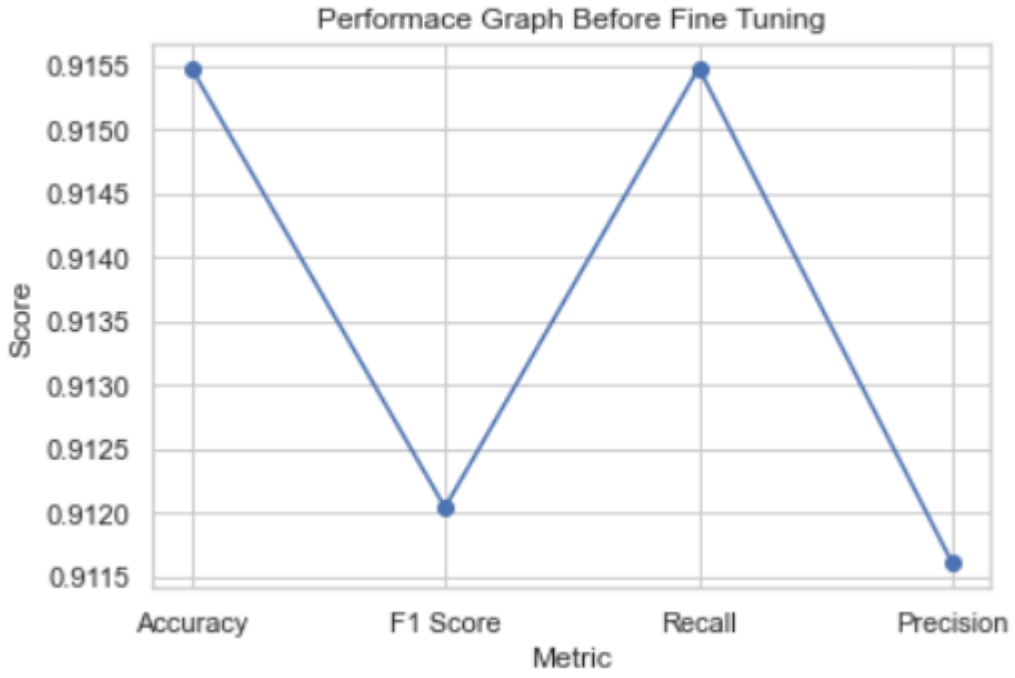
Metric Scores(MLP Classifier)

```
Accuracy = accuracy_score(y_test, mlp_prediction)
F1_score = f1_score(y_test, mlp_prediction,average = 'weighted')
Recall_score = recall_score(y_test, mlp_prediction,average = 'weighted')
Precision_score = precision_score(y_test, mlp_prediction, average = 'weighted')

print("Accuracy score: ", Accuracy )
print("F1 score: ",F1_score)
print("Recall Score: ",Recall_score)
print("Precision score: ",Precision_score)
```

```
Accuracy score:  0.9154847586790855
F1 score:  0.9120403925019586
Recall Score:  0.9154847586790855
Precision score:  0.9116028961209518
```

Line Graph:



After fine-tuning:

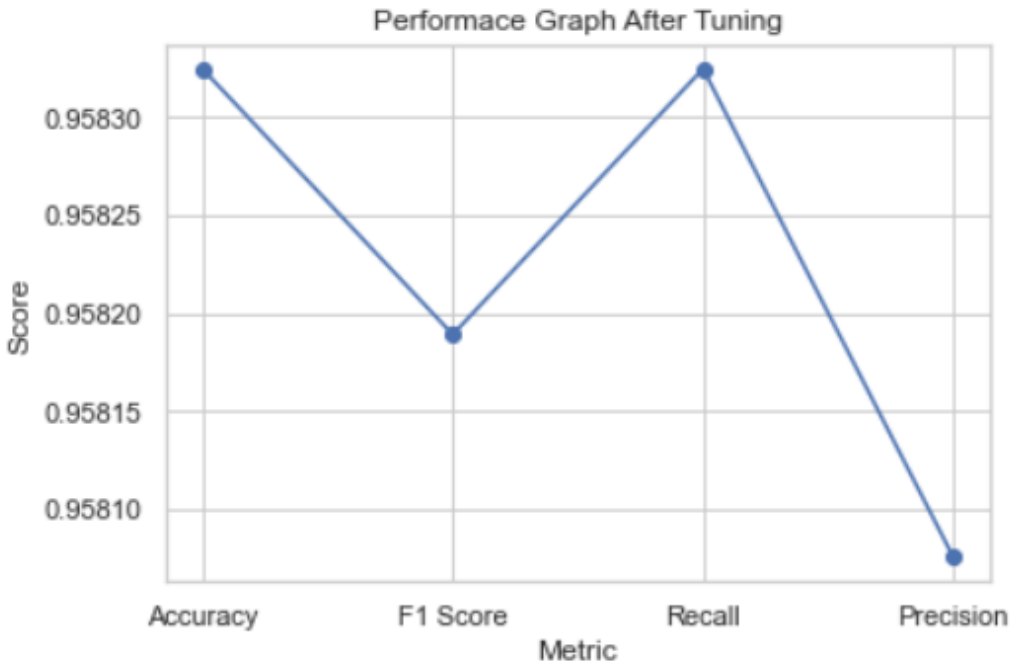
Metric Scores(After Tuning)

```
: Accuracy = accuracy_score(y_test, mlp_prediction)
  F1_score = f1_score(y_test, mlp_prediction,average = 'weighted')
  Recall_score = recall_score(y_test, mlp_prediction,average = 'weighted')
  Precision_score = precision_score(y_test, mlp_prediction, average = 'weighted')

  print("Accuracy score: ", Accuracy )
  print("F1 score: ",F1_score)
  print("Recall Score: ",Recall_score)
  print("Precision score: ",Precision_score)
```

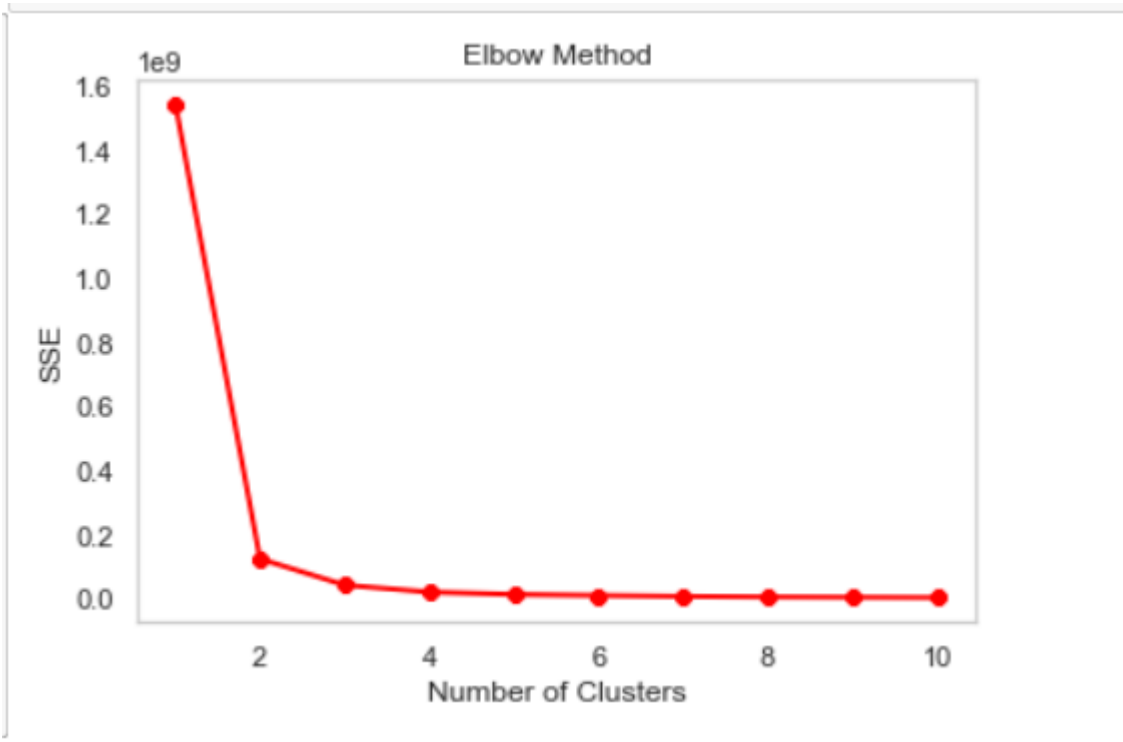
```
Accuracy score:  0.9583245131244708
F1 score:  0.9581890827029542
Recall Score:  0.9583245131244708
Precision score:  0.9580755478271874
```

Line Graph:

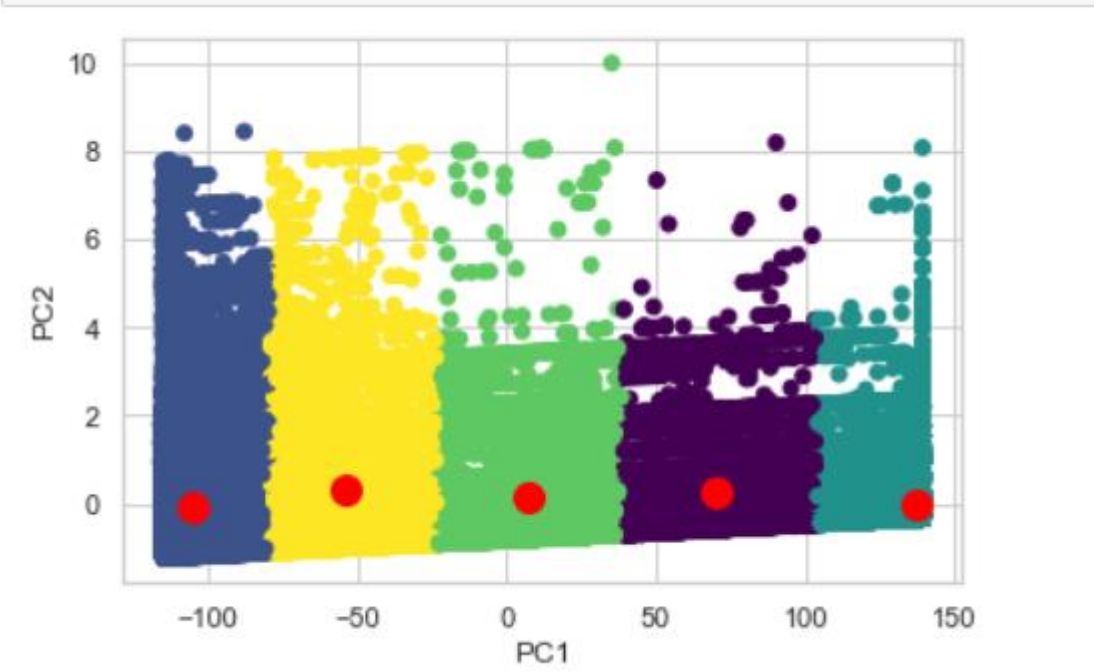


KMeans :

At an optimum value of K, the error started reducing



So at k= 5, there is no error, and clusters are made from the data points.



Comparison Table for Algorithms:

Algorithm	Accuracy	F1 Score	Recall Score	Precision Score
Decision Tree	0.9396433107535986	0.9401115580304753	0.939643310753598	0.9419162525026081
KNN at K=3	0.9698613463166808	0.9697187634971518	0.9698613463166808	0.9697594826259062
MLP(Before Tuning)	0.9154847586790855	0.9120403925019586	0.9154847586790855	0.9116028961209518
MLP(After Tuning)	0.9583245131244708	0.9581890827029542	0.9583245131244708	0.9580755478271874

Conclusion

we can conclude that the KNN algorithm with k=3 achieved the highest accuracy score of 0.9699, as well as the highest F1 score, recall score, and precision score, indicating that it performed the best out of the four algorithms tested in this experiment.

The Decision Tree algorithm also performed well with an accuracy score of 0.9396, but its scores were lower than those of KNN in all other metrics.

The MLP algorithm before tuning had the lowest accuracy score of 0.9155 and the lowest scores in all other metrics, indicating that it performed the worst out of the four algorithms. However, after tuning, its accuracy score increased significantly to 0.9583, and its F1 score, recall score, and precision score also improved, indicating that it could potentially perform better with further optimization.

In summary, KNN with k=3 performed the best out of the four algorithms tested in this experiment, while the MLP algorithm showed promise after tuning, but still did not perform as well as KNN. Decision Tree performed relatively well but was outperformed by KNN.