```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-package
```

```
pip install sklearn-genetic
```

```
Requirement already satisfied: sklearn-genetic in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: deap>=1.0.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scikit-learn>=0.23 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: multiprocess in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: dill>=0.3.4 in /usr/local/lib/python3.7/dist-packages
```

```
pip install sklearn-genetic-opt
```

```
Requirement already satisfied: sklearn-genetic-opt in /usr/local/lib/python3.7/dist-
Requirement already satisfied: deap>=1.3.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scikit-learn>=0.21.3 in /usr/local/lib/python3.7/dist
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: tqdm>=4.61.1 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-package
```

```
#Import libraries
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import metrics
```

```
#Load the data
df1 = pd.read_csv('/content/Tuesday-WorkingHours.pcap_ISCX.csv')
```

```
df2 = pd.read_csv('/content/Wednesday-workingHours.pcap_ISCX.csv')
df3 = pd.read_csv('/content/Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv')
```

```
#Combining of three Dataframes into one Dataframe
frames = [df1,df2,df3]
df = pd.concat(frames)
```

```
#It gives an overview about a Dataframe columns
df.info()
```

```
 24   Fwd IAT Min                  89556 non-null  float64
 25  Bwd IAT Total                 89556 non-null  float64
 26   Bwd IAT Mean                 89556 non-null  float64
 27   Bwd IAT Std                  89556 non-null  float64
 28   Bwd IAT Max                  89556 non-null  float64
 29   Bwd IAT Min                  89556 non-null  float64
 30  Fwd PSH Flags                 89555 non-null  float64
 31   Bwd PSH Flags                89555 non-null  float64
 32   Fwd URG Flags                89555 non-null  float64
 33   Bwd URG Flags                89555 non-null  float64
 34   Fwd Header Length            89555 non-null  float64
 35   Bwd Header Length            89555 non-null  float64
 36  Fwd Packets/s                 89555 non-null  float64
 37   Bwd Packets/s                89554 non-null  float64
 38   Min Packet Length            89554 non-null  float64
 39   Max Packet Length            89554 non-null  float64
 40   Packet Length Mean           89554 non-null  float64
 41   Packet Length Std            89554 non-null  float64
 42   Packet Length Variance       89554 non-null  float64

 43  FIN Flag Count                89554 non-null  float64
 44   SYN Flag Count               89554 non-null  float64
 45   RST Flag Count               89554 non-null  float64
 46   PSH Flag Count               89554 non-null  float64
 47   ACK Flag Count               89554 non-null  float64
 48   URG Flag Count               89554 non-null  float64
 49   CWE Flag Count               89554 non-null  float64
 50   ECE Flag Count               89554 non-null  float64
 51   Down/Up Ratio                89554 non-null  float64
 52   Average Packet Size          89554 non-null  float64
 53   Avg Fwd Segment Size         89554 non-null  float64
 54   Avg Bwd Segment Size         89554 non-null  float64
 55   Fwd Header Length.1          89554 non-null  float64
 56  Fwd Avg Bytes/Bulk            89554 non-null  float64
 57   Fwd Avg Packets/Bulk         89554 non-null  float64
 58   Fwd Avg Bulk Rate            89554 non-null  float64
 59   Bwd Avg Bytes/Bulk           89554 non-null  float64
 60   Bwd Avg Packets/Bulk         89554 non-null  float64
 61  Bwd Avg Bulk Rate             89554 non-null  float64
 62  Subflow Fwd Packets           89554 non-null  float64
 63   Subflow Fwd Bytes            89554 non-null  float64
 64   Subflow Bwd Packets          89554 non-null  float64
 65   Subflow Bwd Bytes            89554 non-null  float64
 66  Init_Win_bytes_forward        89554 non-null  float64
 67   Init_Win_bytes_backward      89554 non-null  float64
 68   act_data_pkt_fwd             89554 non-null  float64
 69   min_seg_size_forward         89554 non-null  float64
 70  Active Mean                   89554 non-null  float64
 71   Active Std                   89554 non-null  float64
```

```
 72   Active Max                   89554 non-null  float64
 73   Active Min                   89554 non-null  float64
 74  Idle Mean                     89554 non-null  float64
 75   Idle Std                     89554 non-null  float64
 76   Idle Max                     89554 non-null  float64
 77   Idle Min                     89554 non-null  float64
 78   Label                        89554 non-null  object
dtypes: float64(65), int64(13), object(1)
memory usage: 54.7+ MB
```

```
#By default the head function returns the first 5 rows
df.head()
```

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Pa Le
0	88	640	7	4	440	358	220	0	62.85
1	88	900	9	4	600	2944	300	0	66.66
2	88	1205	7	4	2776	2830	1388	0	396.57
3	88	511	7	4	452	370	226	0	64.57
4	88	773	9	4	612	2944	306	0	68.00

5 rows × 79 columns

```
#By default the tail function returns the first 5 rows
df.tail()
```

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min
30923	443	60485447	13	16	1184	4304	517	0
30924	80	60857363	13	14	489	9479	423	0
30925	80	60784103	10	10	795	340	747	0
30926	80	115496215	20	19	4714	644	1545	0
30927	80	115569259	16	14	472	257	388	0

5 rows × 79 columns

```
#Count the number of rows and column in the data set
```

```
df.shape
```

```
(89557, 79)
```

```
#Explore the data
df.columns
```

```
Index([' Destination Port', ' Flow Duration', ' Total Fwd Packets',
       ' Total Backward Packets', 'Total Length of Fwd Packets',
       ' Total Length of Bwd Packets', ' Fwd Packet Length Max',
       ' Fwd Packet Length Min', ' Fwd Packet Length Mean',
       ' Fwd Packet Length Std', 'Bwd Packet Length Max',
       ' Bwd Packet Length Min', ' Bwd Packet Length Mean',
       ' Bwd Packet Length Std', 'Flow Bytes/s', ' Flow Packets/s',
       ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min',
       'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max',
       ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std',
       ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd PSH Flags', ' Bwd PSH Flags',
       ' Fwd URG Flags', ' Bwd URG Flags', ' Fwd Header Length',
       ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s',
       ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean',
       ' Packet Length Std', ' Packet Length Variance', 'FIN Flag Count',
       ' SYN Flag Count', ' RST Flag Count', ' PSH Flag Count',
       ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count',
       ' ECE Flag Count', ' Down/Up Ratio', ' Average Packet Size',
       ' Avg Fwd Segment Size', ' Avg Bwd Segment Size',
       ' Fwd Header Length.1', 'Fwd Avg Bytes/Bulk', ' Fwd Avg Packets/Bulk',
       ' Fwd Avg Bulk Rate', ' Bwd Avg Bytes/Bulk', ' Bwd Avg Packets/Bulk',
       'Bwd Avg Bulk Rate', 'Subflow Fwd Packets', ' Subflow Fwd Bytes',
       ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward',
       ' Init_Win_bytes_backward', ' act_data_pkt_fwd',
       ' min_seg_size_forward', 'Active Mean', ' Active Std', ' Active Max',
       ' Active Min', 'Idle Mean', ' Idle Std', ' Idle Max', ' Idle Min',
       ' Label'],
      dtype='object')
```

```
df = df.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
```

```
df.rename({' Label':'Attacks'},axis=1,inplace=True)
```

```
from sklearn.preprocessing import LabelEncoder

Attacks_encoder = LabelEncoder()
df['Attacks']=Attacks_encoder.fit_transform(df['Attacks'].astype(str))

label=df['Attacks']
```

```
#splitting the model into training and testing set
X_train, X_test, y_train, y_test = train_test_split(df,
                                                    label, test_size=0.30,
                                                    random_state=101)
```

```
clf = DecisionTreeClassifier()
```

```

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 1.0
```

```
#training a logistics regression model
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
predictions = logmodel.predict(X_test)
print("Accuracy = "+ str(accuracy_score(y_test,predictions)))
```

```
Accuracy = 0.9879467117783887
```

```
#defining various steps required for the genetic algorithm
def initilization_of_population(size,n_feat):
    population = []
    for i in range(size):
        chromosome = np.ones(n_feat,dtype=np.bool)
        chromosome[:int(0.3*n_feat)]=False
        np.random.shuffle(chromosome)
        population.append(chromosome)
    return population

def fitness_score(population):
    scores = []
    for chromosome in population:
        logmodel.fit(X_train.iloc[:,chromosome],y_train)
        predictions = logmodel.predict(X_test.iloc[:,chromosome])
        scores.append(accuracy_score(y_test,predictions))
    scores, population = np.array(scores), np.array(population)
    inds = np.argsort(scores)
    return list(scores[inds][::-1]), list(population[inds,:][::-1])

def selection(pop_after_fit,n_parents):
    population_nextgen = []
    for i in range(n_parents):
        population_nextgen.append(pop_after_fit[i])
    return population_nextgen

def crossover(pop_after_sel):
    population_nextgen=pop_after_sel
    for i in range(len(pop_after_sel)):
        child=pop_after_sel[i]
        child[3:7]=pop_after_sel[(i+1)%len(pop_after_sel)][3:7]
        population_nextgen.append(child)
    return population_nextgen

def mutation(pop_after_cross,mutation_rate):
```

```
    population_nextgen = []
    for i in range(0,len(pop_after_cross)):
        chromosome = pop_after_cross[i]
        for j in range(len(chromosome)):
            if random.random() < mutation_rate:
                chromosome[j]= not chromosome[j]
        population_nextgen.append(chromosome)
    #print(population_nextgen)
    return population_nextgen

def generations(size,n_feat,n_parents,mutation_rate,n_gen,X_train,
                                   X_test, y_train, y_test):
    best_chromo= []
    best_score= []
    population_nextgen=initilization_of_population(size,n_feat)
    for i in range(n_gen):
        scores, pop_after_fit = fitness_score(population_nextgen)
        print(scores[:2])
        pop_after_sel = selection(pop_after_fit,n_parents)
        pop_after_cross = crossover(pop_after_sel)
        population_nextgen = mutation(pop_after_cross,mutation_rate)
        best_chromo.append(pop_after_fit[0])
        best_score.append(scores[0])
    return best_chromo,best_score
```

Print the selected features

```
from sklearn.svm import LinearSVC
from __future__ import print_function
from sklearn.preprocessing import LabelEncoder
from sklearn import datasets, linear_model

from genetic_selection import GeneticSelectionCV


def main():
    #Combining of three Dataframes into one Dataframe
    frames = [df1,df2,df3]
    df = pd.concat(frames)
    # Some noisy data not correlated
    e = np.random.uniform(0, 0.2, size=(len(df), 30))
    df = df.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)

    df.rename({' Label':'Attacks'},axis=1,inplace=True)
    Attacks_encoder = LabelEncoder()
    df['Attacks']=Attacks_encoder.fit_transform(df['Attacks'].astype(str))

    X = np.hstack((df, e))
    y = df['Attacks']

    estimators = linear_model.LogisticRegression(solver="liblinear", multi_class="ovr")
```

```

    selectors = GeneticSelectionCV(estimators,
                                  cv=6,
                                  verbose=2,
                                  scoring="accuracy",
                                  max_features=10,
                                  n_population=60,
                                  crossover_proba=0.6,
                                  mutation_proba=0.2,
                                  n_generations=15,
                                  crossover_independent_proba=0.6,
                                  mutation_independent_proba=0.06,
                                  tournament_size=4,
                                  n_gen_no_change=20,
                                  caching=True,
                                  n_jobs=-2)
    selectors = selectors.fit(X, y)

    print(selectors.support_)


if __name__ == "__main__":
    main()
```

```
Selecting features with genetic algorithm.
gen     nevals  avg                                 std                                 min
0       60      [ 0.948173  5.183333  0.007084] [ 0.099109  2.717178  0.0445   ] [ 0.
1       39      [-999.133958    5.533333 1000.000259]   [ 3000.288681     3.836086
2       45      [-1165.824202     6.033333  1166.668395]        [ 3210.532885     3.
3       35      [-1832.546218     7.366667  1833.334406]        [ 3869.768527     3.
4       45      [-2332.591903     8.683333  2333.334274]        [ 4229.934877     3.
5       39      [-1832.540998     8.4       1833.335217]        [ 3869.771        2.
6       37      [-1665.85409      8.616667  1666.667658]        [ 3727.143358     2.
7       36      [-1665.853206     8.95      1666.667269]        [ 3727.143753     2.
8       40      [-2165.90174      9.433333  2166.667054]        [ 4120.137991     2.
9       38      [-2499.2676       9.566667  2500.000374]        [ 4330.54987      2.
10      36      [-999.121116    8.666667 1000.000447]           [ 3000.292961     1.
11      40      [-1665.853133     9.25      1666.667074]        [ 3727.143786     2.
12      48      [-1165.804056     8.7       1166.667106]        [ 3210.540206     1.
13      39      [-1499.169944     9.066667  1500.000425]        [ 3571.062908     2.
14      38      [-2165.90196      9.216667  2166.667048]        [ 4120.137876     2.
15      41      [-2332.584653     9.55      2333.333715]        [ 4229.938877     2.
[False False False False False False  True False False  True False False
 False False False False False False False False False False False False
 False False False  True False  True False False False False False False
 False False False False False False  True False False False False False
 False False False False  True False False False False False False False
 False False False False False False False False False False False False
 False False False False  True False False False False False False False
 False False False False False False False False False False False False
 False False  True False False False False False False False False False
 False]
```

```
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")

#Load the data
data1 = pd.read_csv('/content/Tuesday-WorkingHours.pcap_ISCX.csv')
data2 = pd.read_csv('/content/Wednesday-workingHours.pcap_ISCX.csv')
data3 = pd.read_csv('/content/Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv')

#Combining of three Dataframes into one Dataframe
frames = [data1,data2,data3]
data = pd.concat(frames)

data = data.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)

data.rename({' Label':'Attacks'},axis=1,inplace=True)

Attacks_encoder = LabelEncoder()
data['Attacks']=Attacks_encoder.fit_transform(data['Attacks'].astype(str))

n_samples = len(data)
X = data
y = data['Attacks']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7)

clf = RandomForestClassifier()
```

data is fitted with the help of GASearchCV

```
from sklearn_genetic import GASearchCV
from sklearn_genetic.space import Continuous, Categorical, Integer
from sklearn_genetic.plots import plot_fitness_evolution, plot_search_space
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt

param_grid = {'min_weight_fraction_leaf': Continuous(0.01, 0.5, distribution='log-unifo
              'bootstrap': Categorical([True, False]),
              'max_depth': Integer(2, 30),
              'max_leaf_nodes': Integer(2, 35),
              'n_estimators': Integer(100, 300)}

cv = StratifiedKFold(n_splits=3, shuffle=True)

evolved_estimator = GASearchCV(estimator=clf,
                               cv=cv,
                               scoring='accuracy',
                               population_size=10,
```

```
                                 generations=15,
                                 tournament_size=3,
                                 elitism=True,
                                 crossover_probability=0.6,
                                 mutation_probability=0.05,
                                 param_grid=param_grid,
                                 criteria='max',
                                 algorithm='eaMuPlusLambda',
                                 n_jobs=-1,
                                 verbose=True,
                                 keep_top_k=4)
```

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

```
gen	nevals	fitness 	fitness_std	fitness_max	fitness_min
0  	10    	0.973849	0.0155173  	0.992891   	0.961179
1  	14    	0.98018 	0.0155143  	0.992891   	0.961179
2  	12    	0.989705	0.00950861 	0.992891   	0.961179
3  	18    	0.992883	2.23326e-05	0.992891   	0.992816
4  	13    	0.992891	0          	0.992891   	0.992891
5  	15    	0.992891	0          	0.992891   	0.992891
6  	12    	0.992891	0          	0.992891   	0.992891
7  	14    	0.992891	0          	0.992891   	0.992891
8  	8     	0.992891	0          	0.992891   	0.992891
9  	11    	0.992891	0          	0.992891   	0.992891
10 	9     	0.992891	0          	0.992891   	0.992891
11 	15    	0.992891	0          	0.992891   	0.992891
12 	12    	0.992891	0          	0.992891   	0.992891
13 	12    	0.992891	0          	0.992891   	0.992891
14 	15    	0.992891	0          	0.992891   	0.992891
15 	12    	0.992891	0          	0.992891   	0.992891
GASearchCV(crossover_probability=0.6,
           cv=StratifiedKFold(n_splits=3, random_state=None, shuffle=True),
           estimator=RandomForestClassifier(bootstrap=False, max_depth=22,
                                            max_leaf_nodes=25,
                                            min_weight_fraction_leaf=0.0128178317035
                                            n_estimators=275),
           generations=15, keep_top_k=4, mutation_probability=0.05, n_jobs=-1,
           param_grid={'bootstrap': <sklearn_genetic.sp...
                       'max_depth': <sklearn_genetic.space.space.Integer object at 0
                       'max_leaf_nodes': <sklearn_genetic.space.space.Integer object
                       'min_weight_fraction_leaf': <sklearn_genetic.space.space.Cont
                       'n_estimators': <sklearn_genetic.space.space.Integer object a
           population_size=10, return_train_score=True, scoring='accuracy')
```

```
y_predicy_ga = evolved_estimator.predict(X_test)
accuracy_score(y_test,y_predicy_ga)
```

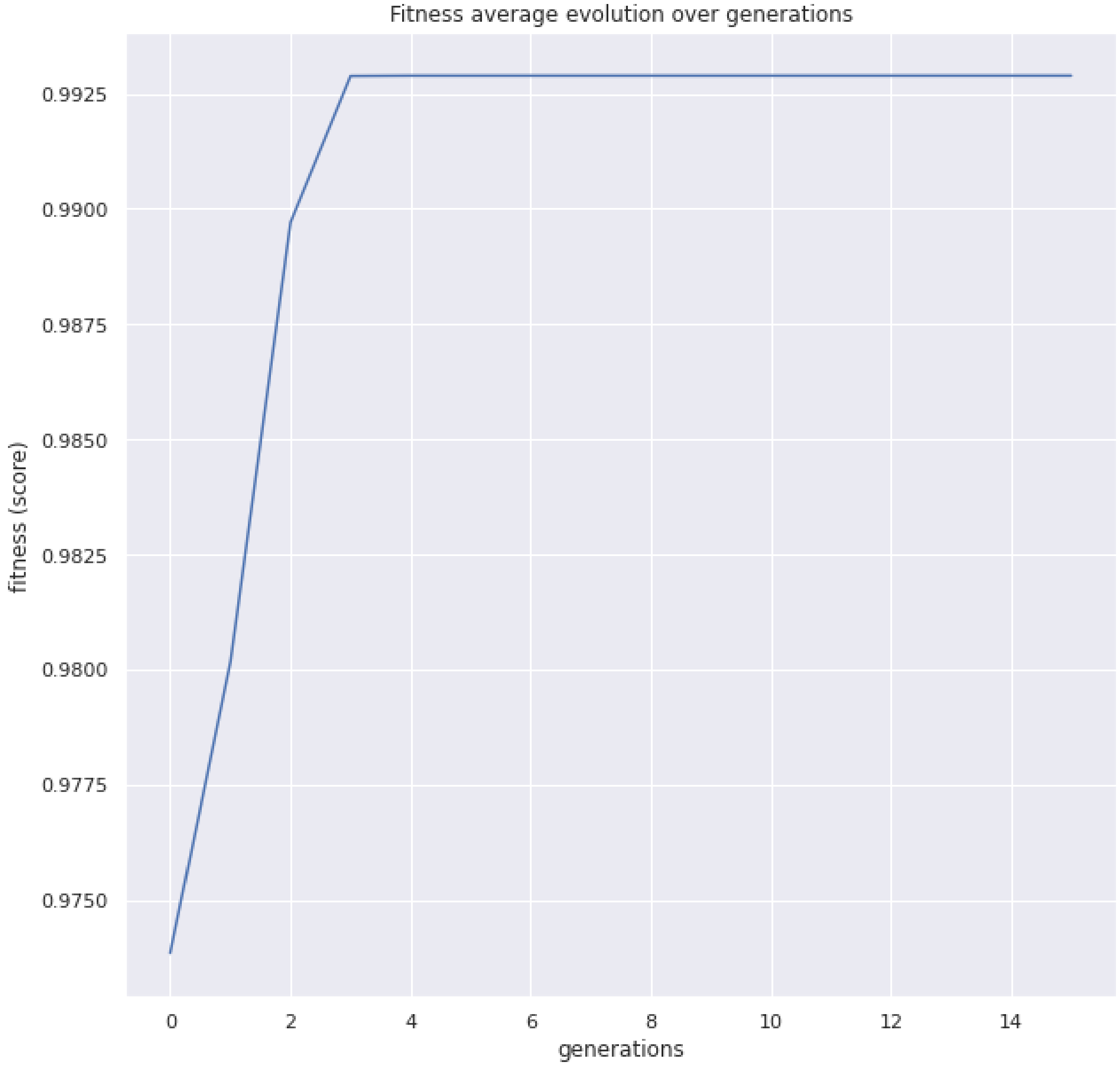
```
0.992853724676982
```

```
evolved_estimator.best_params_
```

```
{'bootstrap': False,
 'max_depth': 22,
```

```
        'max_leaf_nodes': 25,
        'min_weight_fraction_leaf': 0.012817831703580702,
        'n_estimators': 275}
```

```
plot_fitness_evolution(evolved_estimator)
plt.show()
```




```
print(evolved_estimator.logbook)
```

```
     False          [0.99296561 0.99285395 0.99285315]       [11.31169343 11.21477723
     False          [0.99285395 0.99285395 0.99296482]       [11.46857238 11.52505732
     False          [0.99285395 0.99285395 0.99296482]       [11.17141342 11.46613693
     False          [0.99296561 0.9927423  0.99285315]       [11.18666863 10.63165712
     False          [0.99296561 0.99285395 0.99262982]       [11.35737205 11.37384152
     False          [0.99296561 0.99285395 0.99285315]       [11.86456132 11.51538396
     False          [0.99296561 0.9927423  0.99285315]       [11.24379635 11.61328721
     False          [0.99296561 0.9927423  0.99296482]       [11.57952404 11.79654026
     False          [0.99296561 0.9927423  0.99296482]       [11.16064072 11.62353134
     False          [0.99285395 0.99285395 0.99296482]       [11.55695295 11.2514255
     False          [0.9927423  0.99285395 0.99296482]       [9.23811841 9.31808352 6.
     False          [0.99285395 0.99285395 0.99296482]       [9.50731421 9.42682409 6.
     False          [0.99296561 0.99285395 0.99285315]       [9.45780444 9.68561411 6.
```

```
False           [0.99296561 0.99285395 0.99285315]      [11.38922763 11.32546091
False           [0.99285395 0.99285395 0.99296482]      [9.21806002 9.61907005 6.
False           [0.99296561 0.99285395 0.99285315]      [11.51435018 11.73599863
False           [0.99296561 0.99285395 0.99285315]      [9.46440578 9.3861115  6.
False           [0.99296561 0.9927423  0.99296482]      [11.8011384  11.11341572
False           [0.99296561 0.99285395 0.99274149]      [9.4444921  9.45029473 6.
False           [0.99296561 0.99285395 0.99285315]      [9.66726208 9.72219086 6.
False           [0.99296561 0.9927423  0.99296482]      [11.91321135 12.28313637
False           [0.99285395 0.99285395 0.99296482]      [11.50712705 11.37994719
False           [0.99285395 0.9927423  0.99296482]      [11.0775218  10.9252584
False           [0.99296561 0.99285395 0.99285315]      [8.89219093 9.26939082 6.
False           [0.99296561 0.9927423  0.99296482]      [11.52015042 11.30007887
False           [0.99285395 0.99285395 0.99296482]      [9.10022092 9.47356224 6.
False           [0.99296561 0.99285395 0.99285315]      [11.2474761  11.56699014
False           [0.99285395 0.99285395 0.99296482]      [11.66397333 11.56912279
False           [0.99296561 0.9927423  0.99296482]      [9.35355353 9.40592241 6.
False           [0.99296561 0.9927423  0.99296482]      [9.58159232 9.0182445  6.
False           [0.99296561 0.9927423  0.99285315]      [8.99550533 9.12076735 6.
False           [0.99296561 0.99285395 0.99285315]      [10.88423014 11.71231222
False           [0.99285395 0.99285395 0.99296482]      [9.3780551  9.13492179 6.
False           [0.99296561 0.99285395 0.99285315]      [11.43497872 11.57727838
False           [0.99296561 0.9927423  0.99296482]      [9.05820084 9.46480513 6.
False           [0.99296561 0.99285395 0.99285315]      [11.5822916  11.53190398
False           [0.99285395 0.99285395 0.99296482]      [11.71440887 11.13487339
False           [0.99296561 0.9927423  0.99296482]      [11.57959342 11.3814795
False           [0.99296561 0.9927423  0.99285315]      [11.51244712 11.76514101
False           [0.99296561 0.99285395 0.99285315]      [9.52094579 9.25528908 6.
False           [0.99285395 0.99285395 0.99296482]      [8.85333228 9.50748825 6.
False           [0.99296561 0.99251898 0.99296482]      [11.54541135 11.69236279
False           [0.99285395 0.99285395 0.99296482]      [11.66804338 11.56215715
False           [0.99296561 0.9927423  0.99296482]      [9.47266603 9.35538363 6.
False           [0.99285395 0.99285395 0.99285315]      [11.72685289 10.73224568
False           [0.99296561 0.99285395 0.99285315]      [9.30805755 9.70774174 6.
False           [0.99285395 0.9927423  0.99296482]      [9.38373017 9.54240799 6.
False           [0.99296561 0.9927423  0.99296482]      [11.17252564 11.41401482
False           [0.99296561 0.9927423  0.99296482]      [9.61609244 9.44495535 6.
False           [0.99296561 0.99285395 0.99285315]      [11.51811266 11.48861408
False           [0.99285395 0.99285395 0.99285315]      [11.53081036 11.74502921
False           [0.99296561 0.9927423  0.99296482]      [9.42242479 9.16007471 6.
False           [0.99285395 0.9927423  0.99296482]      [8.95766878 9.34241819 6.
False           [0.99285395 0.99285395 0.99296482]      [9.31093812 9.5622654  6.
False           [0.99296561 0.9927423  0.99296482]      [11.06297445 11.91531849
False           [0.99296561 0.99285395 0.99285315]      [11.61893344 11.71284342
False           [0.99285395 0.99285395 0.99296482]      [9.32737732 9.22695136 6.
```

```
1 evolved_estimator.hof
```

```
{0: {'bootstrap': False,
  'max_depth': 22,
  'max_leaf_nodes': 25,
  'min_weight_fraction_leaf': 0.012817831703580702,
  'n_estimators': 275},
 1: {'bootstrap': False,
  'max_depth': 5,
  'max_leaf_nodes': 25,
  'min_weight_fraction_leaf': 0.012817831703580702,
  'n_estimators': 275},
 2: {'bootstrap': False,
```

```
  'max_depth': 20,
  'max_leaf_nodes': 8,
  'min_weight_fraction_leaf': 0.012817831703580702,
  'n_estimators': 275},
 3: {'bootstrap': False,
  'max_depth': 22,
  'max_leaf_nodes': 25,
  'min_weight_fraction_leaf': 0.012817831703580702,
  'n_estimators': 275}}
```

```
chromo,score=generations(size=20,n_feat=79,n_parents=10,mutation_rate=0.10,
                     n_gen=15,X_train=X_train,X_test=X_test,y_train=y_train,y_test=y_te
clf.fit(X_train.iloc[:,chromo[-1]],y_train)
predictions = clf.predict(X_test.iloc[:,chromo[-1]])
print("Accuracy score after genetic algorithm is= "+str(accuracy_score(y_test,predictic
```

```
[0.9763279629925028, 0.9758175147551443]
[0.9747168607433403, 0.9747168607433403]
[0.9748125697878449, 0.9748125697878449]
[0.9756101451587175, 0.9756101451587175]
[0.9752911150103685, 0.9752911150103685]
[0.9748763758175147, 0.9748763758175147]
[0.9737597702982932, 0.9737597702982932]
[0.9755463391290476, 0.9755463391290476]
[0.9794704099537407, 0.9794704099537407]
[0.9752911150103685, 0.9752911150103685]
[0.9766788961556867, 0.9766788961556867]
[0.9755144361142128, 0.9755144361142128]
[0.9740947519540597, 0.9740947519540597]
[0.9748125697878449, 0.9748125697878449]
[0.977253150422715, 0.977253150422715]
Accuracy score after genetic algorithm is= 0.9925506460360504
```

```
chromo,score=generations(size=20,n_feat=79,n_parents=10,mutation_rate=0.10,
                     n_gen=15,X_train=X_train,X_test=X_test,y_train=y_train,y_test=y_te
logmodel.fit(X_train.iloc[:,chromo[-1]],y_train)
predictions = logmodel.predict(X_test.iloc[:,chromo[-1]])
print("Accuracy score after genetic algorithm is= "+str(accuracy_score(y_test,predictic
```

```
[0.9892154789596109, 0.9887925565658702]
[0.989074504828364, 0.989074504828364]
[0.9883696341721294, 0.9883696341721294]
[0.989074504828364, 0.989074504828364]
[0.9889335306971171, 0.9889335306971171]
[0.989074504828364, 0.989074504828364]
[0.9884401212377528, 0.9884401212377528]
[0.9885810953689997, 0.9885810953689997]
[0.9885106083033763, 0.9885106083033763]
[0.9892859660252343, 0.9892859660252343]
[0.9893564530908578, 0.9893564530908578]
[0.9886515824346233, 0.9886515824346233]
[0.9893564530908578, 0.9893564530908578]
[0.9891449918939874, 0.9891449918939874]
[0.9878057376471417, 0.9878057376471417]
Accuracy score after genetic algorithm is= 0.9869598928596602
```

we can select the features with the help of the genetic selection function

```
from genetic_selection import GeneticSelectionCV
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
import numpy as np

df1 = pd.read_csv('/content/Tuesday-WorkingHours.pcap_ISCX.csv')
df2 = pd.read_csv('/content/Wednesday-workingHours.pcap_ISCX.csv')
df3 = pd.read_csv('/content/Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv')

frames = [df1,df2,df3]
df = pd.concat(frames)

df = df.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
df.rename({' Label':'Attacks'},axis=1,inplace=True)
Attacks_encoder = LabelEncoder()
df['Attacks']=Attacks_encoder.fit_transform(df['Attacks'].astype(str))

x = df.drop('Attacks',axis=1)
Y = df['Attacks']
estimators = DecisionTreeClassifier()
models = GeneticSelectionCV(
    estimators, cv=5, verbose=0,
    scoring="accuracy", max_features=10,
    n_population=100, crossover_proba=0.6,
    mutation_proba=0.05, n_generations=15,
    crossover_independent_proba=0.6,
    mutation_independent_proba=0.05,
    tournament_size=3, n_gen_no_change=10,
    caching=True, n_jobs=-1)
models = models.fit(x, Y)
print('Feature Selection:', x.columns[models.support_])
```

```
Feature Selection: Index([' Destination Port', ' Flow IAT Min', 'Fwd IAT Total', ' F
       ' Bwd URG Flags', ' CWE Flag Count', ' Fwd Header Length.1',
       ' Subflow Bwd Bytes', 'Init_Win_bytes_forward',
       ' Init_Win_bytes_backward'],
      dtype='object')
```

```
#split dataset in features and target variable
feature_cols = [' Destination Port', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Min',
       ' Bwd URG Flags', ' CWE Flag Count', ' Fwd Header Length.1',
       ' Subflow Bwd Bytes', 'Init_Win_bytes_forward',
       ' Init_Win_bytes_backward']
x = df[feature_cols] # Features
Y = df['Attacks'] # Target variable
```

```
# Split dataset into training set and test set
x_train, x_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.3, random_state=1
```

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(x_train,Y_train)

#Predict the response for test dataset
Y_pred = clf.predict(x_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred))
```

```
Accuracy: 0.9988158719002582
```

✓ 5s completed at 5:07 PM