```
pip install scikit-learn
```

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

```
pip install sklearn-genetic
```

```
Collecting sklearn-genetic
  Downloading sklearn_genetic-0.5.1-py3-none-any.whl (11 kB)
Requirement already satisfied: multiprocess in /usr/local/lib/python3.7/dist-packages
Collecting deap>=1.0.2
  Downloading deap-1.3.1-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_
     |████████████████████████████████| 160 kB 9.7 MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: scikit-learn>=0.23 in /usr/local/lib/python3.7/dist-pa
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-packages
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: dill>=0.3.4 in /usr/local/lib/python3.7/dist-packages
Installing collected packages: deap, sklearn-genetic
Successfully installed deap-1.3.1 sklearn-genetic-0.5.1
```

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

```
Collecting sklearn-genetic-opt
  Downloading sklearn_genetic_opt-0.8.1-py3-none-any.whl (30 kB)
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-package
Requirement already satisfied: tqdm>=4.61.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
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-packages
Installing collected packages: sklearn-genetic-opt
Successfully installed sklearn-genetic-opt-0.8.1
```

```
#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()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 80256 entries, 0 to 30927
Data columns (total 79 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0    Destination Port             80256 non-null  int64
 1    Flow Duration                80256 non-null  int64
 2    Total Fwd Packets            80256 non-null  int64
 3    Total Backward Packets       80256 non-null  int64
 4   Total Length of Fwd Packets   80256 non-null  int64
 5    Total Length of Bwd Packets  80256 non-null  int64
 6    Fwd Packet Length Max        80256 non-null  int64
 7    Fwd Packet Length Min        80256 non-null  int64
 8    Fwd Packet Length Mean       80256 non-null  float64
 9    Fwd Packet Length Std        80256 non-null  float64
 10  Bwd Packet Length Max         80256 non-null  int64
 11   Bwd Packet Length Min        80256 non-null  int64
 12   Bwd Packet Length Mean       80256 non-null  float64
 13   Bwd Packet Length Std        80256 non-null  float64
 14  Flow Bytes/s                  80233 non-null  float64
 15   Flow Packets/s               80256 non-null  float64
 16   Flow IAT Mean                80256 non-null  float64
 17   Flow IAT Std                 80255 non-null  float64
 18   Flow IAT Max                 80255 non-null  float64
 19   Flow IAT Min                 80255 non-null  float64
 20  Fwd IAT Total                 80255 non-null  float64
 21   Fwd IAT Mean                 80255 non-null  float64
 22   Fwd IAT Std                  80254 non-null  float64
 23   Fwd IAT Max                  80254 non-null  float64
 24   Fwd IAT Min                  80254 non-null  float64
 25  Bwd IAT Total                 80254 non-null  float64
 26   Bwd IAT Mean                 80254 non-null  float64
 27   Bwd IAT Std                  80254 non-null  float64
 28   Bwd IAT Max                  80254 non-null  float64
 29   Bwd IAT Min                  80254 non-null  float64
 30  Fwd PSH Flags                 80254 non-null  float64
 31   Bwd PSH Flags                80254 non-null  float64
 32   Fwd URG Flags                80254 non-null  float64
 33   Bwd URG Flags                80254 non-null  float64
 34   Fwd Header Length            80254 non-null  float64
```

```
 35   Bwd Header Length            80254 non-null  float64
 36  Fwd Packets/s                 80254 non-null  float64
 37   Bwd Packets/s                80254 non-null  float64
 38   Min Packet Length            80254 non-null  float64
 39   Max Packet Length            80254 non-null  float64
 40   Packet Length Mean           80254 non-null  float64
 41   Packet Length Std            80254 non-null  float64
 42   Packet Length Variance       80254 non-null  float64
 43  FIN Flag Count                80254 non-null  float64
 44   SYN Flag Count               80254 non-null  float64
 45   RST Flag Count               80254 non-null  float64
 46   PSH Flag Count               80254 non-null  float64
 47   ACK Flag Count               80254 non-null  float64
 48   URG Flag Count               80254 non-null  float64
 49   CWE Flag Count               80254 non-null  float64
 50   ECE Flag Count               80254 non-null  float64
 51   Down/Up Ratio                80254 non-null  float64
 52   Average Packet Size          80254 non-null  float64
```

```
#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()
```

Total Total Total Total Fwd Fwd

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

```
(80256, 79)
```

30924 80 60857363 13 14 489 9479 423 0

```
#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,
```

```
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: 0.9999872697414485
```

```
#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 = 0.968288925948086
```

```
#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
```

```python

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

```python
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")
```

```
    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=0.9,
                                  mutation_proba=0.03,
                                  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      [ 0.971844  5.383333  0.003739] [ 0.032894  2.714723  0.021349] [ 0.
1       50      [-499.07301     3.283333  500.004994]   [ 2179.662138     1.984033  
2       51      [-499.071009    3.083333  500.001505]   [ 2179.662597     2.525151  
3       51      [-665.751288    3.683333  666.669658]   [ 2494.682903     2.831323  
4       54      [-165.696714    4.466667  166.668194]   [ 1280.317235     2.472965  
5       56      [-165.692732    3.733333  166.668417]   [ 1280.317753     2.151485  
6       52      [-165.689755    3.25      166.668483]   [ 1280.318141     1.649495  
7       50      [ 0.993259  3.733333  0.001807]         [ 0.002944  1.536952  0.0005
8       54      [ 0.99357   4.166667  0.001891]         [ 0.002261  1.416176  0.0008
9       54      [-332.373189    4.7       333.335195]   [ 1795.23323      2.147091  
10      54      [ 0.993833  4.333333  0.001787]         [ 0.003442  1.72884   0.0012
11      54      [-332.373046    6.25      333.335164]   [ 1795.233257     2.446937  
12      52      [ 0.995936  6.583333  0.001479]         [ 0.003891  1.417647  0.0015
13      54      [-332.369354    6.5       333.334283]   [ 1795.233942     2.04532   
14      53      [-165.686178    5.3       166.667548]   [ 1280.318607     1.6052    
15      52      [ 0.997484  4.183333  0.000862]         [ 0.000005  0.499722  0.0000
[ True 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 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 False False False False False  True  True 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 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=0.9,
                                 mutation_probability=0.03,
                                 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      0.967993        0.0134746       0.980959        0.951819
1       18      0.980202        0.00147787      0.980959        0.977246
2       20      0.980974        1.9926e-05      0.980989        0.98093
3       18      0.980986        8.9116e-06      0.980989        0.980959
4       19      0.98098         1.90207e-05     0.980989        0.98093
5       20      0.980968        2.31992e-05     0.980989        0.98093
6       20      0.980983        1.18806e-05     0.980989        0.980959
7       20      0.980986        8.91054e-06     0.980989        0.980959
8       19      0.98098         1.36129e-05     0.980989        0.980959
9       20      0.98098         1.36121e-05     0.980989        0.980959
10      20      0.980974        1.4852e-05      0.980989        0.980959
11      15      0.980983        1.18806e-05     0.980989        0.980959
12      20      0.980989        1.05886e-09     0.980989        0.980989
13      17      0.980983        1.18821e-05     0.980989        0.980959
14      20      0.980989        1.21307e-09     0.980989        0.980989
15      16      0.980989        1.05886e-09     0.980989        0.980989
GASearchCV(crossover_probability=0.9,
           cv=StratifiedKFold(n_splits=3, random_state=None, shuffle=True),
           estimator=RandomForestClassifier(bootstrap=False, max_depth=3,
                                            max_leaf_nodes=7,
                                            min_weight_fraction_leaf=0.0166861661192
                                            n_estimators=283),
           generations=15, keep_top_k=4, mutation_probability=0.03, n_jobs=-1,
           param_grid={'bootstrap': <sklearn_genetic.space...
                       '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)
```

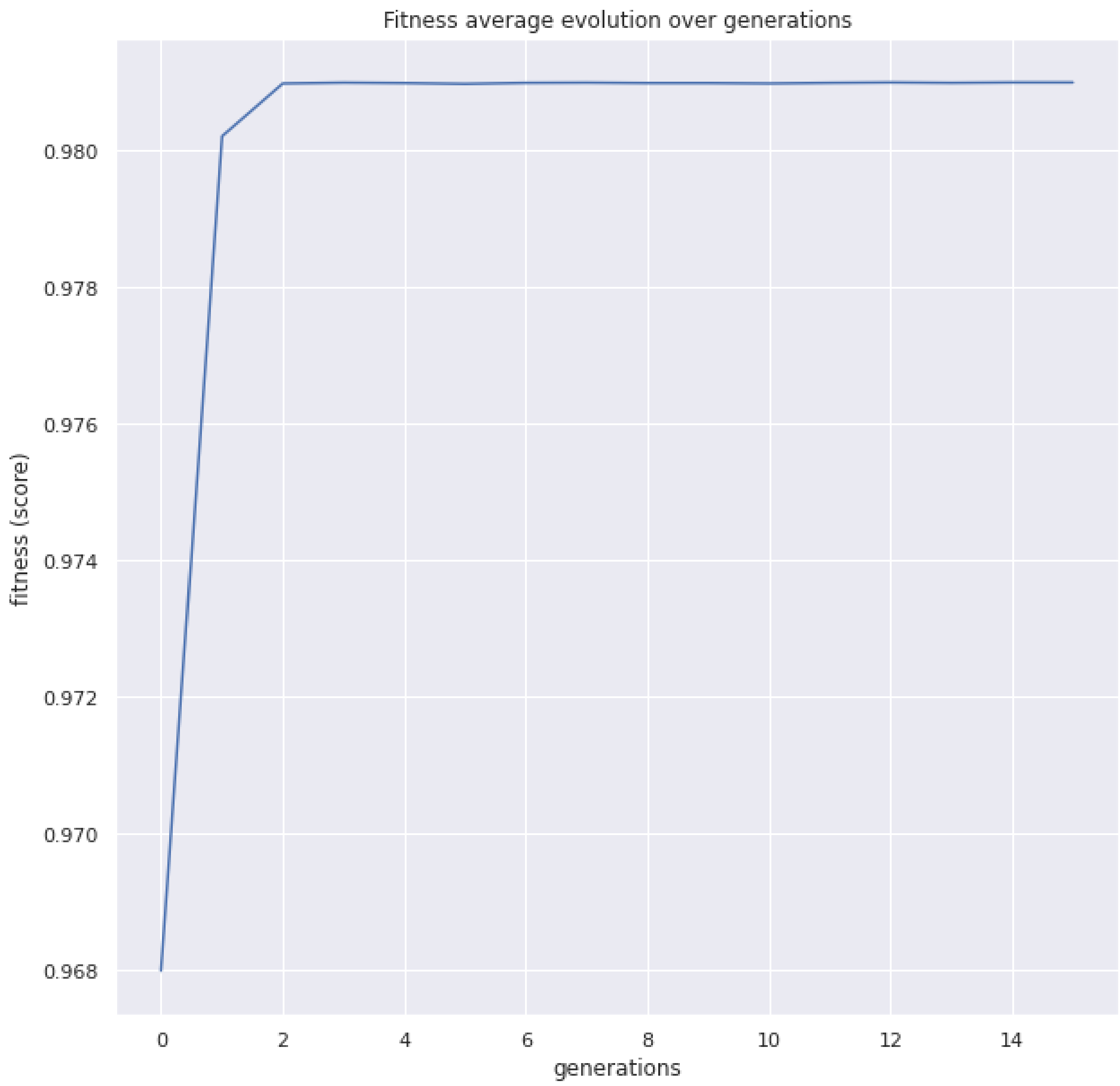
```
0.9820121446666582
```

```
evolved_estimator.best_params_
```

```
{'bootstrap': False,
 'max_depth': 3,
```

```
'max_leaf_nodes': 7,
'min_weight_fraction_leaf': 0.01668616611927558,
'n_estimators': 283}
```

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




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

```
-------------------------------------------------------------------------------------
bootstrap       cv_scores                                  fit_time
False           [0.95179112 0.95179112 0.95187595]         [4.78227305 4.94342732 3.5
True            [0.95179112 0.95179112 0.95187595]         [8.67175794 9.59350085 7.3
True            [0.95179112 0.95179112 0.95187595]         [7.86923122 7.73000383 4.7
True            [0.97540545 0.96462306 0.97495767]         [3.6987555  4.11532211 2.8
False           [0.9808412  0.98101943 0.98092862]         [10.8442347  10.71007919
False           [0.98093032 0.98101943 0.9808395 ]         [10.24782515  9.78313875
True            [0.98093032 0.9808412  0.98110685]         [11.78138614 11.31209278
False           [0.97522723 0.98066298 0.97584885]         [10.30359769  9.92831755
False           [0.9808412  0.98093032 0.98101773]         [11.50090957 11.59582496
False           [0.95179112 0.95179112 0.95187595]         [6.31854129 6.10418081 4.3
```

```
False          [0.98066298 0.98101943 0.98110685]     [11.0061307  10.68017983
True           [0.96462306 0.97594012 0.96470903]     [6.90935755 6.62814951 4.7
False          [0.98075209 0.98093032 0.98110685]     [11.72237492 12.07730556
False          [0.9808412  0.9808412  0.98110685]     [10.62314105 10.93279743
True           [0.9657815  0.96542506 0.95187595]     [3.0193522  3.04260421 2.1
True           [0.9749599  0.96515773 0.96497638]     [6.16820097 6.10399508 4.2
False          [0.95179112 0.95179112 0.95187595]     [5.19366217 5.21862507 3.5
True           [0.98093032 0.9808412  0.98101773]     [8.64323068 8.40267181 6.3
False          [0.95179112 0.95179112 0.95187595]     [9.39343095 9.62427354 6.9
True           [0.9808412  0.98093032 0.98110685]     [5.40753746 5.3602531  3.8
False          [0.98048476 0.96756371 0.98075038]     [10.24381924 10.45868492
False          [0.98066298 0.98093032 0.98110685]     [7.57224965 7.44484019 5.2
False          [0.98075209 0.98101943 0.98101773]     [10.69463968 10.89302301
True           [0.97353413 0.96560328 0.96533286]     [6.67789865 6.88230324 4.5
False          [0.95179112 0.95179112 0.95187595]     [6.35684204 5.92418337 4.1
False          [0.98093032 0.98093032 0.9808395 ]     [7.77910137 7.27595878 5.4
True           [0.95179112 0.95179112 0.95187595]     [5.00466466 4.78189111 3.3
False          [0.95179112 0.95179112 0.95187595]     [6.87747002 6.9822998  4.7
True           [0.9808412  0.98101943 0.98101773]     [8.59650898 8.15534425 6.0
True           [0.98075209 0.98101943 0.98101773]     [7.63027787 7.55121827 5.4
False          [0.9808412  0.98101943 0.98110685]     [10.83221936 10.79689431
True           [0.98075209 0.98093032 0.98110685]     [8.37610269 7.88974118 5.9
True           [0.98101943 0.98093032 0.98101773]     [8.20759916 8.29446149 5.7
False          [0.9808412  0.98093032 0.98101773]     [8.98901868 9.31965947 6.5
False          [0.97469257 0.9808412  0.97513591]     [11.57557535 11.95444965
True           [0.98101943 0.98093032 0.98101773]     [9.3908205  9.67287159 6.6
False          [0.98093032 0.98101943 0.98092862]     [11.78273439 11.49239206
True           [0.98101943 0.9808412  0.98101773]     [9.27734995 9.8231709  6.4
True           [0.98075209 0.98101943 0.98101773]     [6.75452924 6.51238227 4.5
True           [0.98101943 0.98101943 0.9808395 ]     [8.35056591 8.36855125 5.8
True           [0.98093032 0.98093032 0.98110685]     [9.92369366 9.49271941 6.8
False          [0.98101943 0.98093032 0.98092862]     [10.96311188 10.48230863
False          [0.96462306 0.97567279 0.98057214]     [7.86659455 7.58395958 5.4
True           [0.98101943 0.98093032 0.9808395 ]     [7.16497707 6.97475982 5.1
False          [0.98039565 0.97585101 0.97477943]     [10.25636411 10.2131393
True           [0.95179112 0.95179112 0.95187595]     [7.28839302 7.48228955 5.6
True           [0.98101943 0.9808412  0.98092862]     [9.6054852  9.14183378 6.9
False          [0.98093032 0.9808412  0.98110685]     [16.17269421 15.52221322 1
False          [0.98101943 0.98093032 0.98101773]     [14.01565456 13.84658241
False          [0.98075209 0.98093032 0.98110685]     [10.70868206 10.59957099
False          [0.9808412  0.9808412  0.98110685]     [10.6961143  10.59237862
False          [0.98093032 0.98101943 0.98101773]     [10.86771131 10.67186427
True           [0.98101943 0.98101943 0.98092862]     [9.89771748 9.33969808 6.7
False          [0.98093032 0.9808412  0.98110685]     [10.68506098 10.74067974
```

```
1 evolved_estimator.hof
```

```
{0: {'bootstrap': False,
  'max_depth': 3,
  'max_leaf_nodes': 7,
  'min_weight_fraction_leaf': 0.01668616611927558,
  'n_estimators': 283},
 1: {'bootstrap': False,
  'max_depth': 3,
  'max_leaf_nodes': 26,
  'min_weight_fraction_leaf': 0.01668616611927558,
  'n_estimators': 242},
 2: {'bootstrap': True,
```

```
  'max_depth': 23,
  'max_leaf_nodes': 26,
  'min_weight_fraction_leaf': 0.01668616611927558,
  'n_estimators': 242},
 3: {'bootstrap': False,
  'max_depth': 3,
  'max_leaf_nodes': 7,
  'min_weight_fraction_leaf': 0.01668616611927558,
  'n_estimators': 283}}
```

```
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,predictio
```

```
[0.9708477079169477, 0.9707585961070869]
[0.9710641223123242, 0.9710641223123242]
[0.9697019846473082, 0.9697019846473082]
[0.9709495499853602, 0.9709495499853602]
[0.9716115234300409, 0.9716115234300409]
[0.9693709979249678, 0.9693709979249678]
[0.9703130370577826, 0.9703130370577826]
[0.965170012602956, 0.965170012602956]
[0.9683271167237406, 0.9683271167237406]
[0.9698292872328237, 0.9698292872328237]
[0.9699820503354423, 0.9699820503354423]
[0.969905668784133, 0.969905668784133]
[0.9714714905859738, 0.9714714905859738]
[0.9685180706020139, 0.9685180706020139]
[0.9686835639631841, 0.9686835639631841]
Accuracy score after genetic algorithm is= 0.9999872697414485
```

```
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,predictio
```

```
[0.9712805367077005, 0.9706567540386746]
[0.9699565898183392, 0.9699565898183392]
[0.9681107023283643, 0.9681107023283643]
[0.9689508993927667, 0.9689508993927667]
[0.9672068539712042, 0.9672068539712042]
[0.9650172495003374, 0.9650172495003374]
[0.9638587959721462, 0.9638587959721462]
[0.968059781294158, 0.968059781294158]
[0.9657556044963274, 0.9657556044963274]
[0.9699311293012362, 0.9699311293012362]
[0.9665066897508688, 0.9665066897508688]
[0.9679833997428487, 0.9679833997428487]
[0.9715860629129378, 0.9715860629129378]
[0.9684416890507046, 0.9684416890507046]
[0.9702111949893703, 0.9702111949893703]
Accuracy score after genetic algorithm is= 0.9535600168039413
```

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=0.9,
    mutation_proba=0.03, n_generations=15,
    crossover_independent_proba=0.5,
    mutation_independent_proba=0.04,
    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', ' Fwd Packet Length Std', ' Bwd Packet
       ' Fwd URG Flags', ' Max Packet Length', ' Packet Length Mean',
       ' Packet Length Variance', ' Subflow Fwd Bytes',
       ' min_seg_size_forward'],
      dtype='object')
```

```
#split dataset in features and target variable
feature_cols = [' Destination Port', ' Fwd Packet Length Std', ' Bwd Packet Length Min'
       ' Fwd URG Flags', ' Max Packet Length', ' Packet Length Mean',
       ' Packet Length Variance', ' Subflow Fwd Bytes',
       ' min_seg_size_forward']
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: 0.994226855505254
```