```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn 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: 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: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-packag
```

```
pip install sklearn-genetic
```

```
Collecting sklearn-genetic
  Downloading sklearn_genetic-0.5.1-py3-none-any.whl (11 kB)
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
Collecting deap>=1.0.2
  Downloading deap-1.3.1-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux
     |████████████████████████████████| 160 kB 13.9 MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
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
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: scikit-learn>=0.21.3 in /usr/local/lib/python3.7/dist
Requirement already satisfied: tqdm>=4.61.1 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: deap>=1.3.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packag
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
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-package
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
```

```
df = pd.read_csv('/content/KDDTrain+_20Percent.txt')
```

```
##  adding column names to data frame

columns = (['duration'
,'protocol_type'
,'service'
,'flag'
,'src_bytes'
,'dst_bytes'
,'land'
,'wrong_fragment'
,'urgent'
,'hot'
,'num_failed_logins'
,'logged_in'
,'num_compromised'
,'root_shell'
,'su_attempted'
,'num_root'
,'num_file_creations'
,'num_shells'
,'num_access_files'
,'num_outbound_cmds'
,'is_host_login'
,'is_guest_login'
,'count'
,'srv_count'
,'serror_rate'
,'srv_serror_rate'
,'rerror_rate'
,'srv_rerror_rate'
,'same_srv_rate'
,'diff_srv_rate'
,'srv_diff_host_rate'
,'dst_host_count'
,'dst_host_srv_count'
,'dst_host_same_srv_rate'
,'dst_host_diff_srv_rate'
,'dst_host_same_src_port_rate'
,'dst_host_srv_diff_host_rate'
,'dst_host_serror_rate'
,'dst_host_srv_serror_rate'
,'dst_host_rerror_rate'
,'dst_host_srv_rerror_rate'
,'attack'
,'level'])

df.columns = columns
```

```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25191 entries, 0 to 25190
Data columns (total 43 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   duration                     25191 non-null  int64  
 1   protocol_type                25191 non-null  object 
 2   service                      25191 non-null  object 
 3   flag                         25191 non-null  object 
 4   src_bytes                    25191 non-null  int64  
 5   dst_bytes                    25191 non-null  int64  
 6   land                         25191 non-null  int64  
 7   wrong_fragment               25191 non-null  int64  
 8   urgent                       25191 non-null  int64  
 9   hot                          25191 non-null  int64  
 10  num_failed_logins            25191 non-null  int64  
 11  logged_in                    25191 non-null  int64  
 12  num_compromised              25191 non-null  int64  
 13  root_shell                   25191 non-null  int64  
 14  su_attempted                 25191 non-null  int64  
 15  num_root                     25191 non-null  int64  
 16  num_file_creations           25191 non-null  int64  
 17  num_shells                   25191 non-null  int64  
 18  num_access_files             25191 non-null  int64  
 19  num_outbound_cmds            25191 non-null  int64  
 20  is_host_login                25191 non-null  int64  
 21  is_guest_login               25191 non-null  int64  
 22  count                        25191 non-null  int64  
 23  srv_count                    25191 non-null  int64  
 24  serror_rate                  25191 non-null  float64
 25  srv_serror_rate              25191 non-null  float64
 26  rerror_rate                  25191 non-null  float64
 27  srv_rerror_rate              25191 non-null  float64
 28  same_srv_rate                25191 non-null  float64
 29  diff_srv_rate                25191 non-null  float64
 30  srv_diff_host_rate           25191 non-null  float64
 31  dst_host_count               25191 non-null  int64  
 32  dst_host_srv_count           25191 non-null  int64  
 33  dst_host_same_srv_rate       25191 non-null  float64
 34  dst_host_diff_srv_rate       25191 non-null  float64
 35  dst_host_same_src_port_rate  25191 non-null  float64
 36  dst_host_srv_diff_host_rate  25191 non-null  float64
 37  dst_host_serror_rate         25191 non-null  float64
 38  dst_host_srv_serror_rate     25191 non-null  float64
 39  dst_host_rerror_rate         25191 non-null  float64
 40  dst_host_srv_rerror_rate     25191 non-null  float64
 41  attack                       25191 non-null  object 
 42  level                        25191 non-null  int64  
dtypes: float64(15), int64(24), object(4)
memory usage: 8.3+ MB
```

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

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragmen
0	0	udp	other	SF	146	0	0	
1	0	tcp	private	S0	0	0	0	
2	0	tcp	http	SF	232	8153	0	
3	0	tcp	http	SF	199	420	0	
4	0	tcp	private	REJ	0	0	0	

5 rows × 43 columns

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

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fr
25186	0	tcp	exec	RSTO	0	0	0	
25187	0	tcp	ftp_data	SF	334	0	0	
25188	0	tcp	private	REJ	0	0	0	
25189	0	tcp	nnsp	S0	0	0	0	
25190	0	tcp	finger	S0	0	0	0	

5 rows × 43 columns

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

```
(25191, 43)
```

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

```
Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
       'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
       'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',
       'su_attempted', 'num_root', 'num_file_creations', 'num_shells',
       'num_access_files', 'num_outbound_cmds', 'is_host_login',
       'is_guest_login', 'count', 'srv_count', 'serror_rate',
       'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
       'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
       'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
       'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
```

```
        'dst_host_srv_rerror_rate', 'attack', 'level'],
       dtype='object')
```

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

```
from sklearn.preprocessing import LabelEncoder

protocol_type_encoder = LabelEncoder()
df['protocol_type']=protocol_type_encoder.fit_transform(df['protocol_type'].astype(str)

service_encoder = LabelEncoder()
df['service']=service_encoder.fit_transform(df['service'].astype(str))

flag_encoder = LabelEncoder()
df['flag']=flag_encoder.fit_transform(df['flag'].astype(str))

attack_encoder = LabelEncoder()
df['attack']=attack_encoder.fit_transform(df['attack'].astype(str))

label=df['attack']
```

```
#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: 0.9997353797300873
```

```
#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 = 0.8641174913998412
```

```
#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():
    df = pd.read_csv('/content/KDDTrain+_20Percent.txt')

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

    ##  adding column names to data frame

    columns = (['duration','protocol_type','service','flag','src_bytes','dst_bytes','la
    ,'logged_in','num_compromised','root_shell','su_attempted','num_root','num_file_cre
    ,'is_host_login','is_guest_login','count','srv_count','serror_rate','srv_serror_rat
    ,'diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_count','dst_ho
    ,'dst_host_same_src_port_rate','dst_host_srv_diff_host_rate','dst_host_serror_rate'
    ,'attack','level'])

    df.columns = columns

    protocol_type_encoder = LabelEncoder()
    df['protocol_type']=protocol_type_encoder.fit_transform(df['protocol_type'].astype(

    service_encoder = LabelEncoder()
    df['service']=service_encoder.fit_transform(df['service'].astype(str))

    flag_encoder = LabelEncoder()
    df['flag']=flag_encoder.fit_transform(df['flag'].astype(str))

    attack_encoder = LabelEncoder()
    df['attack']=attack_encoder.fit_transform(df['attack'].astype(str))


    X = df
    y = df['attack']

    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=0.05,
```

```
                                  n_generations=15,
                                  crossover_independent_proba=0.6,
                                  mutation_independent_proba=0.05,
                                  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      [ 0.787703  5.683333  0.00862 ] [ 0.12829   3.196309  0.024058] [ 0.
1       38      [-832.522646    7.716667  833.336318]   [ 2764.098424     2.608267
2       46      [-1499.219947     8.066667  1500.0028  ]        [ 3571.041902     2.
3       32      [-499.10737     8.133333  500.002445]           [ 2179.654255     1.
4       36      [-165.729093    8.616667  166.668888]           [ 1280.31302      1.
5       45      [-332.407688    9.4       333.335684]           [ 1795.226824     0.
6       38      [-165.723174    9.316667  166.669031]           [ 1280.31379      0.
7       33      [-999.134936    9.616667 1000.002024]           [ 3000.288355     0.
8       42      [-1332.4962       9.966667  1333.335346]        [ 3399.674693     0.
9       28      [-832.445804   10.        833.335298]           [ 2764.121592     0.
10      38      [-999.127851   10.05     1000.001874]           [ 3000.290716     0.
11      36      [-665.761445   10.05      666.668665]           [ 2494.680189     0.
12      33      [-832.443197   10.066667  833.335187]           [ 2764.122378     0.
13      44      [-1165.808066    10.1       1166.668212]         [ 3210.538748     0.
14      31      [-165.71077    10.016667  166.668277]           [ 1280.315405     0.
15      34      [-665.759083   10.183333  666.668184]           [ 2494.68082      0.
[False  True False  True False False False  True False  True False False
 False False False False False False False False False False False False
 False  True False False False False False False  True False False  True
 False  True False  True False  True 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
data = pd.read_csv('/content/KDDTrain+_20Percent.txt')

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

columns = (['duration','protocol_type','service','flag','src_bytes','dst_bytes','land',
,'logged_in','num_compromised','root_shell','su_attempted','num_root','num_file_creatic
```

```
,'is_host_login','is_guest_login','count','srv_count','serror_rate','srv_serror_rate','
,'diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_count','dst_host_s
,'dst_host_same_src_port_rate','dst_host_srv_diff_host_rate','dst_host_serror_rate','ds
,'attack','level'])

data.columns = columns

protocol_type_encoder = LabelEncoder()
data['protocol_type']=protocol_type_encoder.fit_transform(data['protocol_type'].astype(

service_encoder = LabelEncoder()
data['service']=service_encoder.fit_transform(data['service'].astype(str))

flag_encoder = LabelEncoder()
data['flag']=flag_encoder.fit_transform(data['flag'].astype(str))

attack_encoder = LabelEncoder()
data['attack']=attack_encoder.fit_transform(data['attack'].astype(str))

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

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      0.886952        0.03248         0.952627        0.863438
1       14      0.923541        0.0352381       0.954215        0.869657
2       12      0.938454        0.0305511       0.954215        0.869657
3       14      0.953646        0.000388286     0.954215        0.953156
4       11      0.953857        0.000360213     0.954215        0.953421
5       11      0.953804        0.000292621     0.954215        0.953421
6       14      0.95399         0.000237084     0.954215        0.953553
7       9       0.954096        0.00018192      0.954215        0.953818
8       16      0.953844        0.000453789     0.954215        0.952759
9       14      0.954228        0.000983952     0.956994        0.953288
10      12      0.954082        0.000144958     0.954215        0.953818
11      13      0.953791        0.000582242     0.954215        0.952759
12      12      0.953778        0.000458587     0.954215        0.952891
13      13      0.954294        0.000555776     0.955406        0.953024
14      10      0.954373        0.000442061     0.955406        0.953553
15      14      0.954625        0.000524158     0.955406        0.954215
GASearchCV(crossover_probability=0.6,
           cv=StratifiedKFold(n_splits=3, random_state=None, shuffle=True),
           estimator=RandomForestClassifier(max_depth=16, max_leaf_nodes=32,
                                            min_weight_fraction_leaf=0.0165794432350
                                            n_estimators=163),
           generations=15, keep_top_k=4, mutation_probability=0.05, n_jobs=-1,
           param_grid={'bootstrap': <sklearn_genetic.space.space.Catego...
                       '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')
```

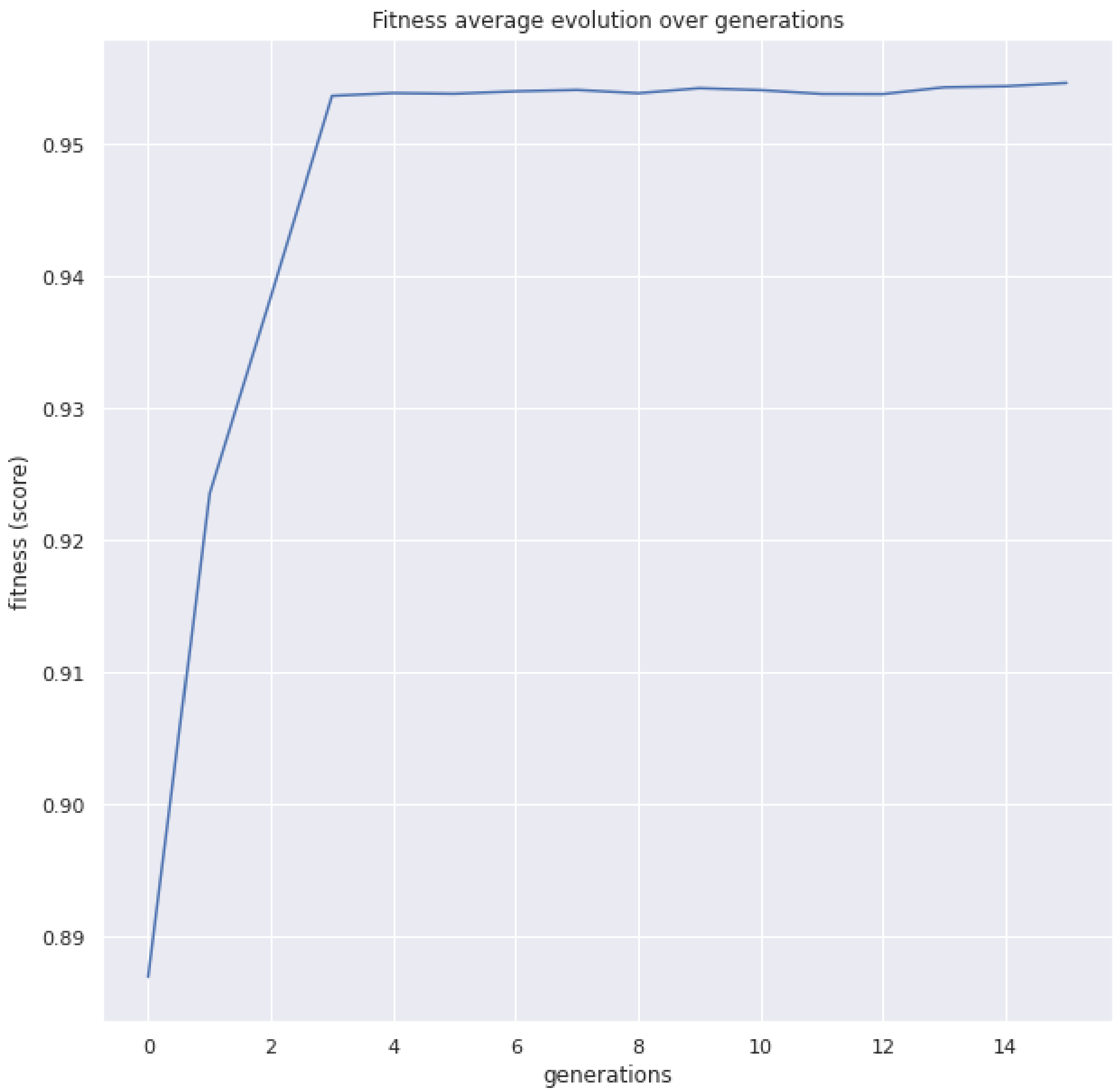
```
y_predicy_ga = evolved_estimator.predict(X_test)
accuracy_score(y_test,y_predicy_ga)
```

```
0.95349892253601
```

```
evolved_estimator.best_params_
```

```
{'bootstrap': True,
 'max_depth': 16,
 'max_leaf_nodes': 32,
 'min_weight_fraction_leaf': 0.016579443235020405,
 'n_estimators': 163}
```

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




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

```
True            [0.95156808 0.95514093 0.95275903]      [0.73261237 0.72189474 0.
True            [0.9511711  0.95514093 0.953553  ]      [0.73891521 0.68889713 0.
True            [0.95474395 0.953553   0.94918618]      [0.71425247 0.71916962 0.
True            [0.95156808 0.95315601 0.95514093]      [0.71835661 0.70720339 0.
True            [0.9511711  0.95275903 0.95236205]      [0.69053078 0.67033362 0.
True            [0.95077412 0.95434696 0.95474395]      [0.71357417 0.70152211 0.
False           [0.95236205 0.95037713 0.95553791]      [0.78613067 0.79555917 0.
True            [0.9511711  0.9511711  0.95514093]      [0.69562197 0.69906998 0.
True            [0.95394998 0.95434696 0.95434696]      [0.7202816  0.70668674 0.
True            [0.95474395 0.95156808 0.95315601]      [0.68723583 0.68857026 0.
False           [0.94998015 0.95553791 0.95315601]      [0.77535462 0.78875351 0.
True            [0.95196507 0.95434696 0.94918618]      [0.6602447  0.67793798 0.
True            [0.95156808 0.95514093 0.953553  ]      [0.73755121 0.71438289 0.
False           [0.95275903 0.95275903 0.95633188]      [0.78970265 0.79137039 0.
False           [0.953553   0.95037713 0.95633188]      [0.81618905 0.79903483 0.
True            [0.95156808 0.9511711  0.95434696]      [0.71718574 0.71382928 0.
True            [0.95315601 0.95553791 0.9511711  ]      [0.72318649 0.70953918 0.
True            [0.95275903 0.95037713 0.94958317]      [0.70658398 0.70182562 0.
True            [0.94720127 0.95593489 0.95394998]      [0.7381382  0.72267032 0.
True            [0.95514093 0.95275903 0.95156808]      [0.7061708  0.6912694  0.
False           [0.9511711  0.95672886 0.94720127]      [0.7671814  0.75271177 0.
```

```
True         [0.953553   0.9511711  0.95394998]     [0.70185852 0.7107482  0.
True         [0.95196507 0.95394998 0.95315601]     [0.71169019 0.727669   0.
True         [0.95236205 0.95672886 0.95474395]     [0.91105366 0.94501448 0.
True         [0.95315601 0.95633188 0.95394998]     [0.69120979 0.70325494 0.
True         [0.95275903 0.95434696 0.95434696]     [0.68302965 0.68959379 0.
False        [0.95077412 0.95077412 0.95434696]     [0.7926228  0.77438855 0.
False        [0.95315601 0.95037713 0.95394998]     [0.73924041 0.73852563 0.
True         [0.95394998 0.95077412 0.94918618]     [0.71785903 0.71489215 0.
True         [0.953553   0.95315601 0.95275903]     [0.67320776 0.69262862 0.
True         [0.95831679 0.95236205 0.95553791]     [0.72399449 0.69885159 0.
True         [0.95315601 0.95315601 0.94998015]     [0.68663287 0.69830298 0.
False        [0.95712584 0.953553   0.94720127]     [0.77276778 0.78042078 0.
True         [0.95633188 0.94561334 0.94998015]     [0.92638612 0.93259907 0.
True         [0.95196507 0.95394998 0.95474395]     [0.92959428 0.93491387 0.
True         [0.95752283 0.95315601 0.94918618]     [0.68942666 0.70939708 0.
True         [0.94759825 0.95553791 0.953553  ]     [0.71622443 0.69854999 0.
True         [0.953553   0.95553791 0.94799524]     [0.88636518 0.87674236 0.
True         [0.95474395 0.95077412 0.95553791]     [0.91750073 0.91578627 0.
True         [0.95394998 0.95514093 0.95394998]     [0.7543292  0.74735594 0.
True         [0.95434696 0.95394998 0.94998015]     [0.68835974 0.6869669  0.
True         [0.95633188 0.95156808 0.95315601]     [0.72477341 0.70462346 0.
True         [0.95474395 0.95196507 0.95315601]     [0.68243766 0.68169999 0.
True         [0.95196507 0.95236205 0.95434696]     [0.72714615 0.76782417 0.
True         [0.95275903 0.95672886 0.95196507]     [0.70624733 0.71791482 0.
True         [0.95514093 0.95077412 0.95196507]     [0.96272516 0.94558096 0.
True         [0.95196507 0.9511711  0.95593489]     [0.7135911  0.71982265 0.
True         [0.95474395 0.95315601 0.95275903]     [0.93427515 0.95922709 0.
True         [0.95236205 0.95434696 0.94799524]     [0.74803424 0.74034023 0.
True         [0.95156808 0.95275903 0.95434696]     [0.71379876 0.71195126 0.
True         [0.95236205 0.95434696 0.95474395]     [0.95361996 0.9597013  0.
True         [0.95434696 0.95275903 0.95037713]     [0.94534373 0.94430661 0.
True         [0.95236205 0.953553   0.95593489]     [0.92454123 0.90064979 0.
True         [0.95394998 0.95275903 0.94998015]     [0.93591022 0.94301176 0.
True         [0.95037713 0.953553   0.95434696]     [0.9234376  0.94976282 0.
True         [0.95514093 0.95315601 0.95077412]     [0.69577026 0.6942203  0.
True         [0.95156808 0.95672886 0.95394998]     [0.94114923 0.9274137  0.
```

```
evolved_estimator.hof
```

```
{0: {'bootstrap': True,
  'max_depth': 16,
  'max_leaf_nodes': 32,
  'min_weight_fraction_leaf': 0.016579443235020405,
  'n_estimators': 163},
 1: {'bootstrap': True,
  'max_depth': 16,
  'max_leaf_nodes': 32,
  'min_weight_fraction_leaf': 0.019245437578258293,
  'n_estimators': 216},
 2: {'bootstrap': True,
  'max_depth': 16,
  'max_leaf_nodes': 32,
  'min_weight_fraction_leaf': 0.019245437578258293,
  'n_estimators': 163},
 3: {'bootstrap': True,
  'max_depth': 16,
  'max_leaf_nodes': 29,
```

```
        'min_weight_fraction_leaf': 0.019245437578258293,
        'n_estimators': 163}}
```

```
chromo,score=generations(size=20,n_feat=43,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
```

```
[0.8517069297947147, 0.8501190881252126]
[0.9026879891119428, 0.9026879891119428]
[0.9104003629352387, 0.9104003629352387]
[0.8943518203470568, 0.8943518203470568]
[0.9522513326528298, 0.9522513326528298]
[0.9614381308835205, 0.9614381308835205]
[0.973857321084269, 0.973857321084269]
[0.9748780764432347, 0.9748780764432347]
[0.9772598389474878, 0.9772598389474878]
[0.9810026085970285, 0.9810026085970285]
[0.9318362254735171, 0.9318362254735171]
[0.909719859362595, 0.909719859362595]
[0.9489622320517183, 0.9489622320517183]
[0.9455597141884995, 0.9455597141884995]
[0.9426675740047635, 0.9426675740047635]
Accuracy score after genetic algorithm is= 0.9417602359079051
```

```
chromo,score=generations(size=20,n_feat=43,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
```

```
[0.8480775774072814, 0.8464897357377793]
[0.8592491777248498, 0.8592491777248498]
[0.86049676760803, 0.86049676760803]
[0.9275263695134399, 0.9275263695134399]
[0.9131790858568675, 0.9131790858568675]
[0.9132925031189747, 0.9132925031189747]
[0.8867528637858683, 0.8867528637858683]
[0.8990586367245095, 0.8990586367245095]
[0.9080753090620393, 0.9080753090620393]
[0.8912328456391063, 0.8912328456391063]
[0.9176590677101055, 0.9176590677101055]
[0.9208914596801633, 0.9208914596801633]
[0.9242372689123285, 0.9242372689123285]
[0.9388680957241692, 0.9388680957241692]
[0.9371668367925599, 0.9371668367925599]
Accuracy score after genetic algorithm is= 0.9358058296472723
```

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

df = pd.read_csv('/content/KDDTrain+_20Percent.txt')

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

columns = (['duration','protocol_type','service','flag','src_bytes','dst_bytes','land',
,'logged_in','num_compromised','root_shell','su_attempted','num_root','num_file_creatic
,'is_host_login','is_guest_login','count','srv_count','serror_rate','srv_serror_rate','
,'diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_count','dst_host_s
,'dst_host_same_src_port_rate','dst_host_srv_diff_host_rate','dst_host_serror_rate','ds
,'attack','level'])

df.columns = columns

protocol_type_encoder = LabelEncoder()
df['protocol_type']=protocol_type_encoder.fit_transform(df['protocol_type'].astype(str)

service_encoder = LabelEncoder()
df['service']=service_encoder.fit_transform(df['service'].astype(str))

flag_encoder = LabelEncoder()
df['flag']=flag_encoder.fit_transform(df['flag'].astype(str))

attack_encoder = LabelEncoder()
df['attack']=attack_encoder.fit_transform(df['attack'].astype(str))

x = df.drop('attack',axis=1)
Y = df['attack']
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=0.5,
    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(['service', 'flag', 'src_bytes', 'is_host_login', 'same_srv
       'diff_srv_rate', 'dst_host_count', 'dst_host_srv_diff_host_rate',
       'dst_host_serror_rate', 'level'],
      dtype='object')
```

```
#split dataset in features and target variable
feature_cols = ['service', 'flag', 'src_bytes', 'is_host_login', 'same_srv_rate',
       'diff_srv_rate', 'dst_host_count', 'dst_host_srv_diff_host_rate',
```

```
       'diff_srv_rate', 'dst_host_count', 'dst_host_srv_diff_host_rate',
       'dst_host_serror_rate', 'level']
x = df[feature_cols] # Features
Y = df['attack'] # 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: 0.9949722148716592
```

✓ 0s completed at 7:27 AM