Step Forward, Step Backward and Exhaustive Feature Selection (Wrapping Method.)

Wrapping Method.

- In the wrapping method we choose the subset of feature and train the model by using this.Based on the inferences that we draw from previous model we decide to add or remove feature from our subset thats means the feature subset. The problem is essential reduce the search problem and this method are usually coputationally very expensive.Moreever we can say the wrapper uses the combination variable to determine the predictive power of the model and it also find the best combination of variable. It is computationally expensive than the filter method although it is quite helpfull to get the better accuracy than the filter method.It is not recommanded the higher number of features.It will make the huge combination of feature and practically little,very difficult to implement

- 1.Use the combinations of variables to determine predictive power.
- 2.Find the best combination of variable.
- 3.Computationally expnsive than the filter method.
- 4.Perform the better than the filter method.
- 5.Not recommended on high number of features.

- Wrapper methods are based on greedy search algorithm as the evaluate all possible combination of feature and select the combination that produce the best result for a specific machine learning algorithms.

- In filtering method we have not incuded the machine learning algorithm while selecting the features but in wrapper method we include the machine learning algorithm while selecting the features.The downside of this approach is that is testing the all possible cobination of the features can be computationally very expensive particularly when when feature set is vary large.

- It divide into the three categories :-
  - 1.Subset Selection (Exhaustic Feature Selection).
  - 2.Foreward Step Selection.
  - 3.Backward Step Selection (Recursive Feature Selection).

1.Foreward Step Selection or Step Foreward Selection.

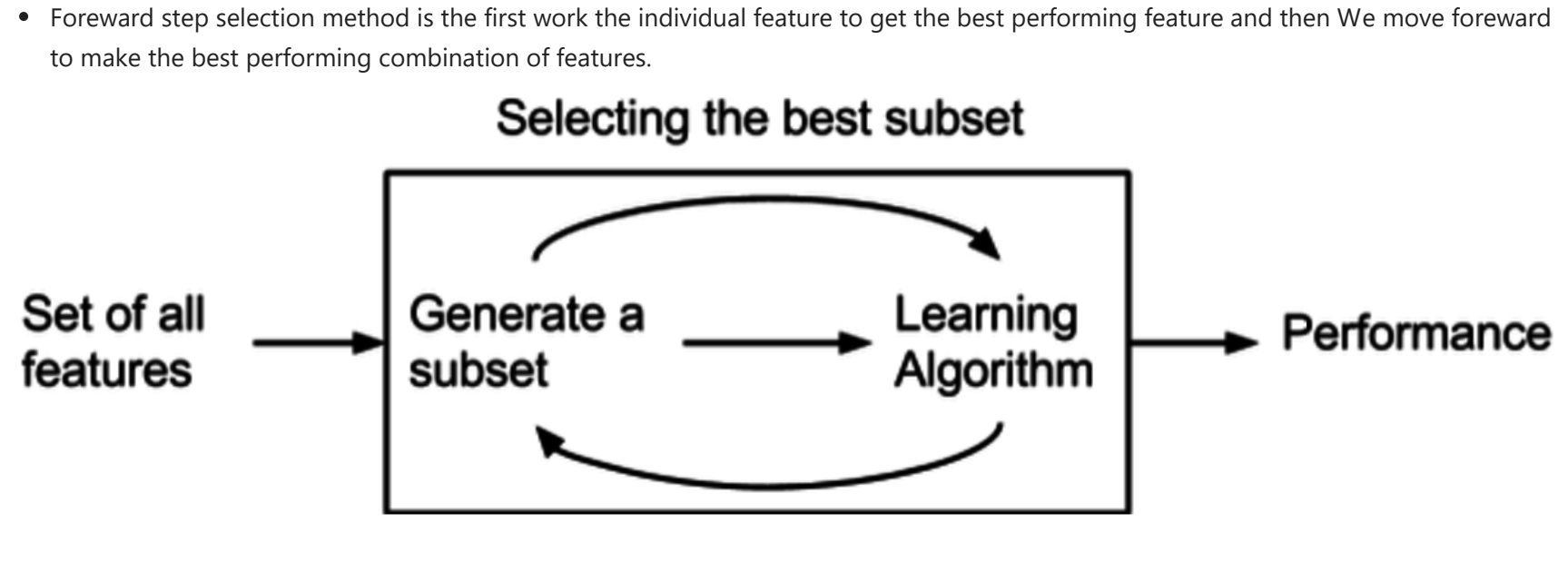
Step 1:

- In the first step of foreward step selection process the performace of classifier is evaluated on each of this features particularly and then best performing features are selected.

Step 2:

- First feature tried the combination of all other feature and then the combination two features that yeild the best possible performance is selected and then process continue until we reaches our desire number features are avalible and performace.

- Foreward step selection method is the first work the individual feature to get the best performing feature and then We move foreward to make the best performing combination of features.



2.Backward Step Selection or Step Backward Selection(Recursive Feature Selection).

It's name is showing that it will work exact opposite as we do in step foreward selection.

Step 1:

- One feature is removed in round rabin fashion from the total featuresets and then the performace of classifier is eveluated and this process goes on and we romoving the one feature which is least important or least performing in featurespace.It keeps on untill we reaches the desired performance,accuracy,number of featureset.

- This is used to balance of most important features in datasets.It works like first time we containing the large features in daatsetes in combination with other features and we keep removing those who are the least important one by one in step format and check at each wheather we got the required importance or not.

- If we got these feature with required then we stop removing the feature and if not then we have to keep going untill and unless we will not get best performing features combination.

3. Subset Selection (Exhaustive Feature Selection)

- In the Exhaustive feature selection method (subset selection method):Performace of machine learning algorithm is evaluated against all possible combinations of the features in the features set. The feature subset that yeild the best performances selected the exhaustive search alogorithm is the most gridy algorithm all the wrapper method.Since it tries the all combinations of features and select the best one.The downside of exaustive feature selection is that it can slower infact
- It is slower comapare to step foreward and step backward method.Since it evaluate all features combinations.

- lets suppose that if our modle or data have the N features then it evaluate took 2*(2^(N-1)) of feature set.

- For example if your model have 10 feature space then it will run atleast 124 times and model having feature 1024 feature.

- Now we see that we only having the only 10 features but our model will run for alleast 1024 times.

- So,this is computatinaly expensive method (exahaustive feature selcetion method)

Points To Remember:-

- 1.Fits the modle with each possible combinations of N features.
- 2.Required massive computational power.
- (Y=BO, Y= BO+B1X1, Y= CO+C1*X2, Y=D0+D1*X1 + D2*X2)
- 3.Use Test error to evaluate the model performance.

Use Of mixtend in wrapper method.

Things has been become very easy by using the mixtend in wrapper method.

In [1]:
```
#we have to install the mixtend by visisting the site otherwise we get it by running the ! pip install mixtend
#! pip install mlxtend
```

- More imformation avalible at http://rasbt.github.io/mlxtend/
- If we check the API of the mixtend liberary in that we get the feature selection API in that we again get SubAPI that is Sequential feature selection.

- It used to reduce the initial diamensinaity (d) to the k diamensinality , were k is less than d (d>k)

- The motivation behaind the feature selection algorithm is to automatically select subset of the feature that is most relevant to the problem and the goal of the feature selection two float:

  - 1.we want to improve the computatinal efficiency.

  - 2.Reduce the generalization error of the model by removing the irrelevant features.

- In nutshell,

- Sequential features selection remove or add one feature at time based on the classifier performace until feature subset of the desired size k is reached.

- There are four diffrent flavors of SFAs(sequential feature selection algorithm) via SequentialFeatureSelector:

- 1.Sequential Foreward Selection(SFS)
- 2.Sequential Backward Selection(SBS)
- 3.Sequential Foreward Floating Selection(SFFS)
- 4.Sequential Backward Floating Selection(SBFS)

In [1]:
```
#One by One we learn feature selection by using the above API
```

Step Foreward Selection(SFS)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
```

In [3]:
```
#although we are using the tree based algorithm and it doesn't depends on the normalization and standardizatio
#StandarScaler - Standardize the data on to the common scale
#Normalization- it is techinique is useto data preparation for machine learning.Its goal is to change the nume
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
```

In [4]:
```
data = datasets.load_wine()
```

In [5]:
```
#through which we can get the all information about the data.
data.keys()
```

Out[5]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])
```

In [6]:
```
#see the Description of the wine datasets
print(data.DESCR)
# in that we are getting the instances(rows)of the data = 178
#columns = 13
#features = attributes avaliable.
```

```
.. _wine_dataset:

Wine recognition dataset
------------------------

**Data Set Characteristics:**

    :Number of Instances: 178 (50 in each of three classes)
    :Number of Attributes: 13 numeric, predictive attributes and the class
    :Attribute Information:
            - Alcohol
            - Malic acid
            - Ash
            - Alcalinity of ash
            - Magnesium
            - Total phenols
            - Flavanoids
            - Nonflavanoid phenols
            - Proanthocyanins
            - Color intensity
            - Hue
            - OD280/OD315 of diluted wines
            - Proline

    - class:
            - class_0
            - class_1
            - class_2

    :Summary Statistics:

    ============================= ==== ===== ======= =====
                                   Min   Max   Mean     SD
    ============================= ==== ===== ======= =====
    Alcohol:                      11.0  14.8    13.0   0.8
    Malic Acid:                   0.74  5.80    2.34  1.12
    Ash:                          1.36  3.23    2.36  0.27
    Alcalinity of Ash:            10.6  30.0    19.5   3.3
    Magnesium:                    70.0 162.0    99.7  14.3
    Total Phenols:                0.98  3.88    2.29  0.63
    Flavanoids:                   0.34  5.08    2.03  1.00
    Nonflavanoid Phenols:         0.13  0.66    0.36  0.12
    Proanthocyanins:              0.41  3.58    1.59  0.57
    Colour Intensity:              1.3  13.0     5.1   2.3
    Hue:                          0.48  1.71    0.96  0.23
    OD280/OD315 of diluted wines: 1.27  4.00    2.61  0.71
    Proline:                       278  1680     746   315
    ============================= ==== ===== ======= =====

    :Missing Attribute Values: None
    :Class Distribution: class_0 (59), class_1 (71), class_2 (48)
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

The data is the results of a chemical analysis of wines grown in the same
region in Italy by three different cultivators. There are thirteen different
measurements taken for different constituents found in the three types of
wine.

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository
[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,
School of Information and Computer Science.

.. topic:: References

  (1) S. Aeberhard, D. Coomans and O. de Vel,
  Comparison of Classifiers in High Dimensional Settings,
  Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Technometrics).

  The data was used with many others for comparing various
  classifiers. The classes are separable, though only RDA
  has achieved 100% correct classification.
  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
  (All results using the leave-one-out technique)

  (2) S. Aeberhard, D. Coomans and O. de Vel,
  "THE CLASSIFICATION PERFORMANCE OF RDA"
  Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Journal of Chemometrics).
```

If we observed at the mean of the data is varing preety much that so that we required the standardization and normalization to get the feature columnss on to the common scale.It basically required when we use the regression machine learning model to predict the data accuracy.It is preprocess of the data before going to use in model.

But here,If we are not going to use the regression machine learning algorithm instad of that we are going to use classification then there is no such requirement to use of standardization and normalization.

Because the classifiaction is never dependd upon the Standardization and Normalization.

In [7]:
```
x = pd.DataFrame(data.data)
y = data.target

x.shape,y.shape
```

Out[7]:
```
((178, 13), (178,))
```

In [8]:
```
x.columns = data.feature_names
x.head()
```

Out[8]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32

In [9]:
```
x.isnull().sum()
```

Out[9]:
```
alcohol                         0
malic_acid                      0
ash                             0
alcalinity_of_ash               0
magnesium                       0
total_phenols                   0
flavanoids                      0
nonflavanoid_phenols            0
proanthocyanins                 0
color_intensity                 0
hue                             0
od280/od315_of_diluted_wines    0
proline                         0
dtype: int64
```

In [10]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size =0.8,random_state=0,stratify=y)

x_train.shape,x_test.shape
```

Out[10]:
```
((142, 13), (36, 13))
```

In [11]:
```
#now our task reduce the 13 features space and find out the best combination of these feature set to predict t
#if we havinng the large datasets with the large feature space that time its been diffict to use them as well
#the more time to train the model.So,for that this techinique really helpfull to select the such features those
#predict accuracy of maodel without leading to get more training time.
```

Step Foreward Feature Selection(SFS)

- Here,we use the SFS = sequential feature selector in that we use the RandomForestClassfier.

- k_features = this number of featuers we wanted (desired number features),we use here the subset of the 6 features which are maximumly performing.

- foreward = this means that the we are using the foreward step selection features if its true.

- floating =we having two which are the plan and float here we are going to use only plan that is fload = False.

- verbose = we want the log while traing the model we are gone set teh true.

- scoring = method for the scoring we are going to use here is accuracy.

- n_job = how many core can while selecting the features and -1 says that use all the avaliable core in this computer.

- cv =cross validation.

- Then fit the model to availd the overfitting.

- After the run the below code we will get the train of the model in the Backend LokyBackend with the 4 concurrent workers. That means it is using the four core in my computer.

In [12]:
```
sfs = SFS(RandomForestClassifier(n_estimators=100,random_state=0,n_jobs= -1),
          k_features=6,
           forward=True,
           floating=False,
           verbose=2,
           scoring='accuracy',
           cv=4,
           n_jobs=-1
         ).fit(x_train,y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  13 out of  13 | elapsed:    8.8s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  13 out of  13 | elapsed:    8.8s finished

[2021-09-08 22:03:02] Features: 1/6 -- score: 0.7609126984126985[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:    4.4s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:    4.4s finished

[2021-09-08 22:03:07] Features: 2/6 -- score: 0.9303571428571429[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  11 out of  11 | elapsed:    4.0s finished

[2021-09-08 22:03:11] Features: 3/6 -- score: 0.9652777777777778[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    3.9s finished

[2021-09-08 22:03:15] Features: 4/6 -- score: 0.9583333333333333[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   7 out of   9 | elapsed:    3.1s remaining:    0.8s
[Parallel(n_jobs=-1)]: Done   9 out of   9 | elapsed:    3.9s finished

[2021-09-08 22:03:19] Features: 5/6 -- score: 0.9718253968253968[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   6 out of   8 | elapsed:    2.8s remaining:    0.9s
[Parallel(n_jobs=-1)]: Done   8 out of   8 | elapsed:    2.9s finished

[2021-09-08 22:03:22] Features: 6/6 -- score: 0.9789682539682539
```

- Since, It is step foreward feature selection technique.Now we will see that it selected the 1st a single feature out of the 6 then it find out the maximumly performace in the score format .Similarly,one by one select the feature upto the 6 with the increasing the accuracy score.like 1/6,1/2,1/3 upto the 6/6.

- We can do the same thing with the 13 also.

- k__features = 13 then,

- From that we can understand that ,the if we use the large subset of feature it will no give us good accuracy by watching the senorios of the 13 feature in k_features.

- There wil be some subset of feature that actually giving us the better accuracy if we comapare withn the others fetaure subset.

- By observing the accuracy score of the feature subset, we can say that instaed of adding all feature together we some technique where we can actually subset of the features or total number of features and we ca improve the accuracy and we can reduce the total training time.

- We can get the maximum select feature feature*names* by using the code :-

- sfs.k_feature*names*

- At the k_faeture pair 13/13 as well as we will accuracy also at this feature subset.

In [13]:
```
sfs.k_feature_names_
```

Out[13]:
```
('alcohol', 'malic_acid', 'ash', 'magnesium', 'flavanoids', 'color_intensity')
```

- We will get the the feature index of selected features by using the the below code sfs.k_feature_names.

In [14]:
```
sfs.k_feature_idx_
```

Out[14]:
```
(0, 1, 2, 4, 6, 9)
```

- To get the accuracy with the those features which are selected by using the code that is sfs.k*score*

- It is represnting that the maximum score can be attend by using the selected features subset

In [15]:
```
sfs.k_score_
```

Out[15]:
```
0.9789682539682539
```

- Now go ahead and see in the pandas DataFrame throgh which we will get behaviour of the faeture accuracy changing.

- pd.DataFrame.from_dict(sfs.get_metric_dict()).T

- It will give us the dataframe of the behaviour of that features subset one by one.

In [16]:
```
pd.DataFrame(sfs.get_metric_dict()).T
```

Out[16]:

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
1	(6,)	[0.7222222222222222, 0.75, 0.7428571428571429,...	0.760913	(flavanoids,)	0.0647163	0.0403725	0.0233091
2	(6, 9)	[0.8888888888888888, 0.8611111111111112, 0.971...	0.930357	(flavanoids, color_intensity)	0.0915652	0.0571218	0.0329793
3	(4, 6, 9)	[0.9166666666666666, 0.9444444444444444, 1.0, ...	0.965278	(magnesium, flavanoids, color_intensity)	0.0578426	0.0360844	0.0208333
4	(1, 4, 6, 9)	[0.9166666666666666, 0.9166666666666666, 1.0, ...	0.958333	(malic_acid, magnesium, flavanoids, color_inte...	0.0667909	0.0416667	0.0240563
5	(0, 1, 4, 6, 9)	[0.9722222222222222, 0.9722222222222222, 0.942...	0.971825	(alcohol, malic_acid, magnesium, flavanoids, c...	0.0323914	0.0202069	0.0116665
6	(0, 1, 2, 4, 6, 9)	[0.9444444444444444, 1.0, 0.9714285714285714, ...	0.978968	(alcohol, malic_acid, ash, magnesium, flavanoi...	0.0370199	0.0230944	0.0133336

- It will give us the representation of the feature subset with the respective accuracy score,feature_idx,feature_names etc.
- We can select these features manually also,but we do it same thing by programactically.
- It will gives the best the performing feature subset in the range of feature (1,2,4,6,9).

- After that we will get the result with the feature subset with the accuracy as previous.

- Now if we see that it will give us best the performing feature so, there no need to select the best performing feature manually that we had discussed.

In [17]:
```
sfs = SFS(RandomForestClassifier(n_estimators=100,random_state=0,n_jobs= -1),
          k_features=(1,8),
           forward=True,
           floating=False,
           verbose=2,
           scoring='accuracy',
           cv=4,
           n_jobs=-1
         ).fit(x_train,y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  13 out of  13 | elapsed:    5.2s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  13 out of  13 | elapsed:    5.2s finished

[2021-09-08 22:03:28] Features: 1/8 -- score: 0.7609126984126985[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:    4.3s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:    4.3s finished

[2021-09-08 22:03:32] Features: 2/8 -- score: 0.9303571428571429[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  11 out of  11 | elapsed:    4.2s finished

[2021-09-08 22:03:37] Features: 3/8 -- score: 0.9652777777777778[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    4.0s finished

[2021-09-08 22:03:41] Features: 4/8 -- score: 0.9583333333333333[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   7 out of   9 | elapsed:    2.9s remaining:    0.7s
[Parallel(n_jobs=-1)]: Done   9 out of   9 | elapsed:    3.7s finished

[2021-09-08 22:03:44] Features: 5/8 -- score: 0.9718253968253968[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   6 out of   8 | elapsed:    2.8s remaining:    0.9s
[Parallel(n_jobs=-1)]: Done   8 out of   8 | elapsed:    3.0s finished

[2021-09-08 22:03:48] Features: 6/8 -- score: 0.9789682539682539[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   4 out of   7 | elapsed:    1.7s remaining:    1.2s
[Parallel(n_jobs=-1)]: Done   7 out of   7 | elapsed:    2.8s finished

[2021-09-08 22:03:50] Features: 7/8 -- score: 0.9720238095238095[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   6 | elapsed:    1.4s remaining:    1.4s
[Parallel(n_jobs=-1)]: Done   6 out of   6 | elapsed:    2.4s finished

[2021-09-08 22:03:53] Features: 8/8 -- score: 0.9859126984126985
```

In [18]:
```
#in similar fashion we have to get the best k_score_
```

In [19]:
```
sfs.k_score_
```

Out[19]:
```
0.9859126984126985
```

- After the run the sfs.k_feature*names* it will give us the only those features who are the best performing in (1,8).

- LAST time we had discuss how select the best feature set with programing so in this we got the best performing featureset.

- We got the same featuers those who are the best faetures had selected while we choosen the k_features =6 last time.

In [20]:
```
sfs.k_feature_names_
```

Out[20]:
```
('alcohol',
 'malic_acid',
 'ash',
 'magnesium',
 'flavanoids',
 'nonflavanoid_phenols',
 'color_intensity',
 'hue')
```

In [21]:
```
sfs.k_features
```

Out[21]:
```
(1, 8)
```

Step Backward Selection (SBS)

- Here if we see that the we had done the coding with the foreward = False that means it is in backward step selection.

- Backward step :- it will work as feature selection by backward moment.

- First it will start to give the accuracy of the all faetures subset i.e.13 Then it will give the accuracy of the feature subset of 12th then 11th,10th and so on upto the 1st that is last feature avalible in the datasets.

In [22]:
```
sbs = SFS(RandomForestClassifier(n_jobs=-1,n_estimators=100,random_state=0),
          k_features=(1,8),
          floating=False,
          forward=False,
          verbose=2,
          n_jobs=-1,
          scoring='accuracy',
          cv=4).fit(x_train,y_train)

#here if we see that the we had done the coding with the foreward = False that means it is in backward step se
#backward step :- it will work as feature selection by backward moment.
#first it will start to give  the accuracy of the all faetures subset i.e.13
#then it will give the accuracy of the feature subset of 12th then 11th,10th and so on upto the 1st that is la
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  13 out of  13 | elapsed:    5.4s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  13 out of  13 | elapsed:    5.4s finished

[2021-09-08 22:04:11] Features: 12/1 -- score: 0.9720238095238095[Parallel(n_jobs=-1)]: Using backend LokyBacke
nd with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:    4.3s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:    4.3s finished

[2021-09-08 22:04:16] Features: 11/1 -- score: 0.9720238095238095[Parallel(n_jobs=-1)]: Using backend LokyBacke
nd with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  11 out of  11 | elapsed:    4.1s finished

[2021-09-08 22:04:20] Features: 10/1 -- score: 0.9789682539682539[Parallel(n_jobs=-1)]: Using backend LokyBacke
nd with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    4.0s finished

[2021-09-08 22:04:24] Features: 9/1 -- score: 0.9930555555555556[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   7 out of   9 | elapsed:    3.0s remaining:    0.8s
[Parallel(n_jobs=-1)]: Done   9 out of   9 | elapsed:    3.8s finished

[2021-09-08 22:04:28] Features: 8/1 -- score: 0.9789682539682539[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   6 out of   8 | elapsed:    2.8s remaining:    0.9s
[Parallel(n_jobs=-1)]: Done   8 out of   8 | elapsed:    3.0s finished

[2021-09-08 22:04:31] Features: 7/1 -- score: 0.9720238095238095[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   4 out of   7 | elapsed:    1.7s remaining:    1.2s
[Parallel(n_jobs=-1)]: Done   7 out of   7 | elapsed:    2.8s finished

[2021-09-08 22:04:34] Features: 6/1 -- score: 0.9720238095238095[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   6 | elapsed:    1.4s remaining:    1.4s
[Parallel(n_jobs=-1)]: Done   6 out of   6 | elapsed:    2.4s finished

[2021-09-08 22:04:36] Features: 5/1 -- score: 0.9720238095238095[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    2.0s finished

[2021-09-08 22:04:39] Features: 4/1 -- score: 0.9720238095238095[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   4 out of   4 | elapsed:    1.2s finished

[2021-09-08 22:04:40] Features: 3/1 -- score: 0.9511904761904761[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:    1.1s finished

[2021-09-08 22:04:41] Features: 2/1 -- score: 0.9303571428571429[Parallel(n_jobs=-1)]: Using backend LokyBacken
d with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   2 | elapsed:    0.9s finished

[2021-09-08 22:04:42] Features: 1/1 -- score: 0.7609126984126985
```

- This will train the modle in the reversed manner of the foreward step selection method.

- That means it will give us the first accuracy based of the all feature set 13/13 then start decreasing 12/13,11/13,10/13 and so one upto the 1/13 feature subset.

- Make the sbs = sfs if we sometimes forgot to write the sbs.

In [23]:
```
sbs= sfs
```

In [24]:
```
#to get the accuracy score
sbs.k_score_
```

Out[24]:
```
0.9859126984126985
```

- While selecting the backward step selection method we will get the feature which are most imporatnt by waching there highest avg_score.

- for an example we got the 6 features those contributing more accuracy in features sets. Because it predicted the feature selction from the bottom of the feature set.i.e.start from 1/1,2/2,3/3,4/4 upto the 13/13.

In [25]:
```
#to get the feature names.
sbs.k_feature_names_
```

Out[25]:
```
('alcohol',
 'malic_acid',
 'ash',
 'magnesium',
 'flavanoids',
 'nonflavanoid_phenols',
 'color_intensity',
 'hue')
```

- However,we got the optimum feature that can give us the maximum accuracy.

Exhaustive Feature Selection (SFS)

In [26]:
```
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
```

- Here,the exahustive feature selection techinique

- min_features = We wanted exahustive feature so that is why we required minimum features from where we want to make the selection.here we had taken the min_featutes =4 that means will start to predict the accuracy of the features with the faeture subset of 4.

- max_faetures = It gone select the feature combination UPTo featureset 5.

- cv = None -becasuse it is exahaustive feature selection and this is going to take lot of the time.it can take the 2 to 3 hrs to check the training time.Because exahaustive feature selection is the most exapnsive feature selection technique.It takes lot of the time.

In [27]:
```
efs = EFS(RandomForestClassifier(random_state=0,n_jobs=-1,n_estimators=100),
         min_features=4,
         max_features=5,
         scoring='accuracy',
         n_jobs=-1,
         cv=None).fit(x_train,y_train)
```

```
Features: 2002/2002
```

- It can gives the output like feature 2002/2002 = It is representing that the model is run for the 2002 times.

- Mean time had calcalted the combination

- C(13,4) = Here we can see that the feature avalible from 4 faeture subset out of 13 feature there are possible combination is 715

- similarly,

- C(13,5) = here,we can see that the feature avalible from the 5 feature subset out of 13 feature there is possible combination is 1287 C(13,4) + C(13,5) = 715 + 1287 If we sum this we got the 2002 features.

- So,from that we can say that for the min features and max feature subset it has been train for the 2002 feature subsets.

In [28]:
```
715 + 1287
```

Out[28]:
```
2002
```

- If we want to know more about the efs then try :-

- help(efs)

- Then we will get the detail documentation on the exahaustive feature selection for classification and regression.

In [29]:
```
#here we will get the best score
efs.best_score_
```

Out[29]:
```
1.0
```

- It will give us the best subset of features that produce the maximum result.

- Whatever the feature we will get these feature will gives us the result in the terms of best*score* and maximum result is the one.

In [ ]:
```
#thoght which we will get the bset performing features
```

In [30]:
```
efs.best_feature_names_
```

Out[30]:
```
('alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash')
```

- We will get the index of the best features.

In [31]:
```
efs.best_idx_
```

Out[31]:
```
(0, 1, 2, 3)
```

Now we want to plot

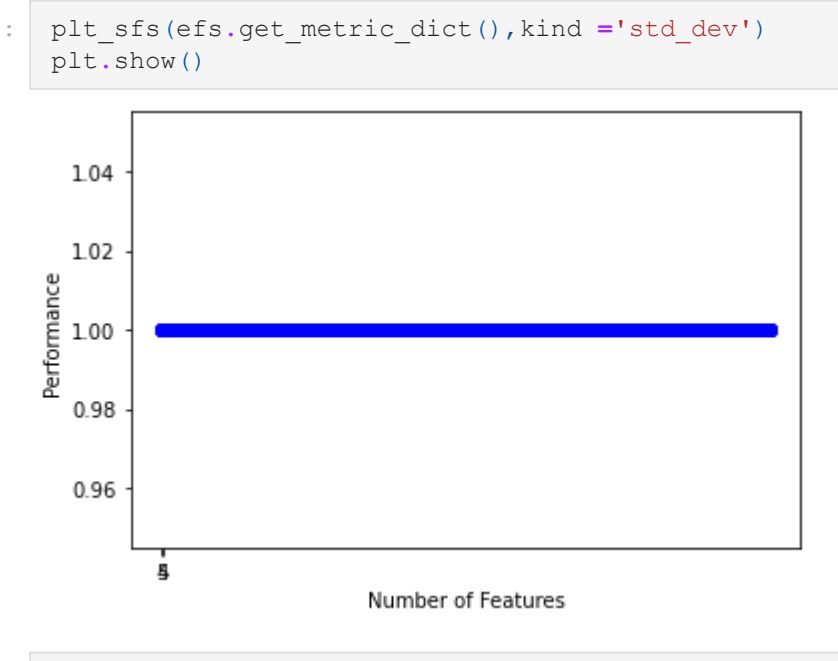
- For plotting,

- We had imported the plot_sequential_feature_selection by uisng API plotting and refered them with the the name as plot sfs.

- Here ,We are going to put the data as efs in the dictionary format and use the shaaded as std = standard deviation.

In [37]:
```
from mlxtend.plotting import plot_sequential_feature_selection as plt_sfs
```

In [39]:
```
plt_sfs(efs.get_metric_dict(),kind ='std_dev')
plt.show()
```



In [23]:
```
#here we will get to know homw any features are go there performace at the 1.
```