FEATURE SELECTION using Wrapper methods in Python

*Content copied from.... (https://towardsdatascience.com/feature-selection-using-wrapper-methods-in-python-f0d352b346f)

In order to perform any machine learning task or to get insights from such high dimensional data, feature selection becomes very important. Since some features may be irrelevant or less significant to the dependent variable so their unnecessary inclusion to the model leads to
• Increase in complexity of a model and makes it harder to interpret.
• Increase in time complexity for a model to get trained.
• Result in a dumb model with inaccurate or less reliable predictions.
Hence, it gives an indispensable need to perform feature selection. Feature selection is very crucial and must component in machine learning and data science workflows especially while dealing with high dimensional datasets.

What is Feature selection?

As the name suggests, it is a process of selecting the most significant and relevant features from a vast set of features in the given dataset.
For a dataset with **d** input features, ***the feature selection*** process results in **k** features such that k < d,where k is the smallest set of significant and relevant features.
So feature selection helps in finding the smallest set of features which results in :
• **Training** a machine learning algorithm faster.
• Reducing the **complexity** of a model and making it easier to **interpret.**
• Building a **sensible model** with **better prediction power.**
• **Reducing overfitting** by selecting the right set of features.

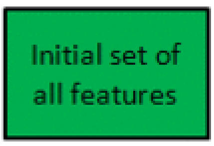
Feature selection methods For a dataset with d features, if we apply hit and trial method with all possible combinations of features then total $2^d — 1$ models need to be evaluated for a significant set of features. It is a time-consuming approach, therefore, we use feature selection techniques to find out the smallest set of features more efficiently.
There are three types of feature selection techniques :

1. Filter methods
2. Wrapper methods
3. Embedded methods
   Difference between Filter, Wrapper and Embedded methods

Filter methods	Wrapper methods	Embedded methods
Generic set of methods which do not incorporate a **specific machine learning algorithm.**	Evaluates on a **specific machine learning algorithm** to find optimal features.	Embeds (fix) features during **model building process.** Feature selection is done by observing each iteration of model training phase.
Much **faster** compared to Wrapper methods in terms of time complexity	**High computation time** for a dataset with many features	Sits **between Filter methods and Wrapper methods** in terms of time complexity
Less prone to **over-fitting**	High chances of **over-fitting** because it involves training of machine learning models with different combination of features	Generally used to reduce **over-fitting** by **penalizing** the coefficients of a model being too large.
Examples – **Correlation, Chi-Square test, ANOVA, Information gain** etc.	Examples - **Forward Selection, Backward elimination, Stepwise selection** etc.	Examples - **LASSO, Elastic Net, Ridge Regression** etc.

Wrapper methods

In wrapper *methods* , the feature selection process is based on a specific machine learning algorithm that we are trying to fit on a given dataset.
It follows a *greedy search approach* by evaluating all the possible combinations of features against *the evaluation criterion.*
The evaluation criterion is simply the performance measure which depends on the type of problem, for eg. *for regression* evaluation criterion can be p-values, R-squared, Adjusted R-squared.
Similarly *for classification* the evaluation criterion can be accuracy, precision, recall, f1-score, etc. Finally, it selects the combination of features that gives the optimal results for the specified machine learning algorithm.



Most commonly used techniques under wrapper methods are:

1. Forward selection
2. Backward elimination
3. Bi-directional elimination(Stepwise Selection)

Too much theory so far. Now let us discuss wrapper methods with an example of **Boston**

house prices dataset available in sklearn. The dataset contains 506 observations of 14 different features. The dataset can be imported using load_boston() function available in sklearn.datasets module.

In [3]:
```python
from sklearn.datasets import load_boston
boston = load_boston()
print(boston.data.shape)          # for dataset dimension
print(boston.feature_names)       # for feature names
print(boston.target)              # for target variable
print(boston.DESCR)               # for data description
```

```
(506, 13)
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.  18.9 21.7 20.4
 18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8
 18.4 21.  12.7 14.5 13.2 13.1 13.5 18.9 20.  21.  24.7 30.8 34.9 26.6
 25.3 24.7 21.2 19.3 20.  16.6 14.4 19.4 19.7 20.5 25.  23.4 18.9 35.4
 24.7 31.6 23.3 19.6 18.7 16.  22.2 25.  33.  23.5 19.4 22.  17.4 20.9
 24.2 21.7 22.8 23.4 24.1 21.4 20.  20.8 21.2 20.3 28.  23.9 24.8 22.9
 23.9 26.6 22.5 22.2 23.6 28.7 22.6 22.  22.9 25.  20.6 28.4 21.4 38.7
 43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8
 18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22.  20.3 20.5 17.3 18.8 21.4
 15.7 16.2 18.  14.3 19.2 19.6 23.  18.4 15.6 18.1 17.4 17.1 13.3 17.8
 14.  14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.4
 17.  15.6 13.1 41.3 24.3 23.3 27.  50.  50.  50.  22.7 25.  50.  23.8
 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.2
 37.9 32.5 26.4 29.6 50.  32.  29.8 34.9 37.  30.5 36.4 31.1 29.1 50.
 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.  22.6 24.4 22.5 24.4 20.
 21.7 19.3 22.4 28.1 23.7 25.  23.3 28.7 21.5 23.  26.7 21.7 27.5 30.1
 44.8 50.  37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.  24.  25.1 31.5
 23.7 23.3 22.  20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8
 29.6 42.8 21.9 20.9 44.  50.  36.  30.1 33.8 43.1 48.8 31.  36.5 22.8
 30.7 50.  43.5 20.7 21.1 25.2 24.4 35.2 32.4 32.  33.2 33.1 29.1 35.1
 45.4 35.4 46.  50.  32.2 22.  20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9
 21.7 28.6 27.1 20.3 22.5 29.  24.8 22.  26.4 33.1 36.1 28.4 33.4 28.2
 22.8 20.3 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21.  23.8 23.1
 20.4 18.5 25.  24.6 23.  22.2 19.3 22.6 19.8 17.1 19.4 22.2 20.7 21.1
 19.5 18.5 20.6 19.  18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.6
 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25.  19.9 20.8 16.8
 21.9 27.5 21.9 23.1 50.  50.  50.  50.  50.  13.8 13.8 15.  13.9 13.3
 13.1 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4 10.2 11.5 15.1 23.2
  9.7 13.8 12.7 13.1 12.5  8.5  5.   6.3  5.6  7.2 12.1  8.3  8.5  5.
 11.9 27.9 17.2 27.5 15.  17.2 17.9 16.3  7.   7.2  7.5 10.4  8.8  8.4
 16.7 14.2 20.8 13.4 11.7  8.3 10.2 10.9 11.   9.5 14.5 14.1 16.1 14.3
 11.7 13.4  9.6  8.7  8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6
 14.1 13.  13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20.  16.4 17.7
 19.5 20.2 21.4 19.9 19.  19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.3
 16.7 12.  14.6 21.4 23.  23.7 25.  21.8 20.6 21.2 19.1 20.6 15.2  7.
  8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.9
 22.  11.9]
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attri
bute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.f
```

```
t.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 oth
erwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (https://archi
ve.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at Carnegie M
ellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that ad
dress regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data
and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Pro
ceedings on the Tenth International Conference of Machine Learning, 236-243, Univ
ersity of Massachusetts, Amherst. Morgan Kaufmann.
```

In [4]:
```
#Let’s convert this raw data into a data frame including target variable and actual
import pandas as pd
bos = pd.DataFrame(boston.data, columns = boston.feature_names)
bos['Price'] = boston.target
X = bos.drop("Price", 1)       # feature matrix
y = bos['Price']               # target feature
bos.head()
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	P
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	

In [5]:
```
#Here, the target variable is Price. We will be fitting a regression model to pred
#through wrapper methods.
```

1. Forward selection :

In forward selection, we start with a null model and then start fitting the model with each individual feature one at a time and select the feature with the minimum p-value. Now fit a model with two features by trying combinations of the earlier selected feature with all other remaining features. Again select the feature with the minimum *p-value.* Now fit a model with three features by trying combinations of two previously selected features with other remaining features. Repeat this process until we have a set of selected features with a p-value of individual feature less than the *significance level* . In short, the steps for forward selection technique are as follows :

1. Choose a significance level (e.g. SL = 0.05 with a 95% confidence).
2. Fit all possible simple regression models by considering one feature at a time. Total ’n’ models are possible.
   Select the feature with the lowest p-value.
3. Fit all possible models with one extra feature added to the previously selected feature(s).
4. Again, select the feature with minimum p-value. if p_value < significance level then go to Step 3, otherwise terminate the process.

Implementing Forward selection using built-in functions in Python:
mlxtend library contains built-in implementation for most of the wrapper methods based feature selection techniques.
SequentialFeatureSelector() function comes with various combinations of feature selection techniques.

In [6]:
```
#importing the necessary libraries
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.linear_model import LinearRegression
# Sequential Forward Selection(sfs)
sfs = SFS(LinearRegression(), k_features=11, forward=True, floating=False, scoring
```

In [7]:
```
sfs.fit(X, y)
sfs.k_feature_names_
```

Out[7]:
```
('CRIM',
 'ZN',
 'CHAS',
 'NOX',
 'RM',
 'DIS',
 'RAD',
 'TAX',
 'PTRATIO',
 'B',
 'LSTAT')
```

SequentialFeatureSelector() function accepts the following major arguments :
• LinearRegression() as an estimator for the entire process. Similarly, it can be any classification based algorithm.
• k_features indicates the number of features to be selected. It can be any random value, but the optimal value can be found by analyzing and visualizing the scores for different numbers of features.
• forward and floating arguments for different flavors of wrapper methods, here, forward = True and floating = False are for forward selection technique.
• Scoring argument specifies the evaluation criterion to be used. For regression problems, there is only r2 score in default implementation. Similarly for classification, it can be accuracy, precision, recall, f1-score, etc.
• cv argument is for k-fold cross-validation.
Now let’s fit the above-defined feature selector on Boston house price dataset.

In [24]:
```
#Performing Mutiple Linear Regression on boston dataset using feature extracted fro
X = pd.DataFrame(bos[['CRIM','ZN','CHAS','NOX','RM','DIS','RAD','TAX','PTRATIO','B
y = pd.DataFrame(bos['Price'])
```

In [26]:
```
X.head()
```

Out[26]:

	CRIM	ZN	CHAS	NOX	RM	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	0.0	0.538	6.575	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	0.0	0.469	6.421	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	0.0	0.469	7.185	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	0.0	0.458	6.998	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	0.0	0.458	7.147	6.0622	3.0	222.0	18.7	396.90	5.33

In [27]:
```
y.head()
```

Out[27]:

	Price
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2

In [29]:
```
from sklearn.model_selection import train_test_split
#split the data in 80/20 proportion
X_train ,X_test, y_train, y_test = train_test_split(X,y,test_size =0.2, random_stat
```

In [31]:
```
from sklearn import linear_model
lm = linear_model.LinearRegression()
model = lm.fit(X_train,y_train)
print('coeficent',lm.coef_)
print('Intercept',lm.intercept_)
print('R square',lm.score(X_train,y_train))
```

```
coeficent [[-1.19265889e-01  4.51412980e-02  2.34282671e+00 -1.62978884e+01
   3.68549285e+00 -1.37731203e+00  2.43212756e-01 -1.08465147e-02
  -1.04568509e+00  8.03588534e-03 -4.96639689e-01]]
Intercept [38.16377346]
R square 0.7729811407453248
```

In [32]:
```
y_pred = lm.predict(X_test)
print('y_pred : ',y_pred[0:5]) ## Printing only first five y_pred elements
print(len(y_pred))
```

```
y_pred :  [[24.82340108]
 [23.77736709]
 [29.40699074]
 [12.141119  ]
 [21.41824566]]
102
```

In [ ]:
```

```