Programmed By : Rithik Tripathi

Connect with me on Linkedin (https://www.linkedin.com/in/rithik-tripathi-data-scientist/)

In this notebook, We will implement Backward feature elimination & Forward feature selection for Reducing Dimenisons

Dimensionality Reduction => BACKWARD FEATURE ELIMINATION

```
Steps Involved  :
- Train model with all variables
- Calculate performance of model
- Eliminate 1 feature, Train model again on remaining features
- Calculate performance of model on this new dataframe after elimination
- Identify the feature for which even after removal, the performance was not impacted much
- Keep repeating above steps until we have mentioned number of features finally left as passed in function call.
```

In [1]:

```
import warnings

warnings.filterwarnings("ignore")
```

In [2]:

```
import pandas as pd

# reading the dataset
df = pd.read_csv('Dataset/Dimensionality_Reduction/backward_feature_elimination.csv')
df.head()

# the dataset is all about to predict the count of bicycles to be rented using the available features
```

Out[2]:

	ID	season	holiday	workingday	weather	temp	humidity	windspeed	count
0	AB101	1	0	0	1	9.84	81	0.0	16
1	AB102	1	0	0	1	9.02	80	0.0	40
2	AB103	1	0	0	1	9.02	80	0.0	32
3	AB104	1	0	0	1	9.84	75	0.0	13
4	AB105	1	0	0	1	9.84	75	0.0	1

In [3]:

```
df.shape # checking dimensionality
```

Out[3]:

```
(12980, 9)
```

In [4]:

```
df.isnull().sum() # checking for Nulls
```

Out[4]:

```
ID            0
season        0
holiday       0
workingday    0
weather       0
temp          0
humidity      0
windspeed     0
count         0
dtype: int64
```

In [5]:

```
# removing ID feature as it is a unique identifier and does not adds any value

df.drop(['ID'], axis=1, inplace=True)
```

In [6]:

```
# separating Target Feature
x = df.drop(['count'], axis=1)
y = df['count']

# validiating the shapes
x.shape, y.shape
```

Out[6]:

```
((12980, 7), (12980,))
```

To use External Module for Forward & Backward Feature Selection Tehiniques => pip install mlxtend

```
!pip install mlxtend
```

In [7]:

```
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.linear_model import LinearRegression
```

In [8]:

```
LR = LinearRegression()

Backward_eliminator = sfs(LR, k_features= 4, forward= False, verbose= 2, scoring= 'neg_mean_squared_error')
```

```

# LR => which model we want to use
# k_features => how many features we want in the output
# forward = False => we want to implement backward feature elimination
# scoring = negative mean squared error

In machine learning, "verbose" is an optional parameter in many algorithms and models that controls the level of output or information displayed while the model is training.

When verbose is set to a high level, such as verbose=1, the algorithm may print out progress updates for each iteration of training, including the current epoch number, the training loss, and other metrics. This can be useful for monitoring the progress of the training and for debugging purposes.

On the other hand, when verbose is set to a low level, such as verbose=0, the algorithm may produce limited or no output during training, making it less distracting and allowing for faster execution.

The exact behavior of the verbose parameter can vary depending on the specific algorithm or model being used.
```

In [9]:

```
# Training model
Backward_eliminator = Backward_eliminator.fit(x,y)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.1s finished

[2023-02-09 03:22:04] Features: 6/4 -- score: -21618.48384431219[Parallel(n_jobs=1)]: Using backend SequentialBackend w
ith 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.1s finished

[2023-02-09 03:22:04] Features: 5/4 -- score: -21555.45783585459[Parallel(n_jobs=1)]: Using backend SequentialBackend w
ith 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished

[2023-02-09 03:22:04] Features: 4/4 -- score: -21477.988573502793
```

In [10]:

```
Backward_eliminator.k_score_
# this is the Negative Mean Squared Error
```

Out[10]:

```
-21477.988573502793
```

Looking at the Selected Features

In [11]:

```
selected_features = list(Backward_eliminator.k_feature_names_)
selected_features
```

Out[11]:

```
['holiday', 'workingday', 'temp', 'humidity']
```

```
 all we need to do now is simply filter these set of features out and make a new dataframe
```

In [13]:

```
new_df = df[selected_features]

# adding back the Target Feature to new dataframe
new_df['count'] = df['count']
new_df.head()
```

Out[13]:

	holiday	workingday	temp	humidity	count
0	0	0	9.84	81	16
1	0	0	9.02	80	40
2	0	0	9.02	80	32
3	0	0	9.84	75	13
4	0	0	9.84	75	1

=> Feature Selection has been done using Backward Feature Selection. We can go ahead for further modeling on these new Features

In [ ]:

```

```

```
lets see another technique which is just reverse of it
```

Dimensionality Reduction => FORWARD FEATURE SELECTION ¶

```
Steps :
- Train model on each feature individually & calculate performance
- choose feature with best performance
- keeping this selected variable fixed, add 1 more feature at a time and train and calculate performance
- variable giving highest performance will be selected
- Keep repeating above steps until we get mentioned number of features finally left as passed in function call.
```

In [14]:

```
# reading the dataset
df = pd.read_csv('Dataset/Dimensionality_Reduction/forward_feature_selection.csv')
df.head()

# the dataset is all about to predict the count of bicycles to be rented using the available features
```

Out[14]:

	ID	season	holiday	workingday	weather	temp	humidity	windspeed	count
0	AB101	1	0	0	1	9.84	81	0.0	16
1	AB102	1	0	0	1	9.02	80	0.0	40
2	AB103	1	0	0	1	9.02	80	0.0	32
3	AB104	1	0	0	1	9.84	75	0.0	13
4	AB105	1	0	0	1	9.84	75	0.0	1

In [15]:

```
# removing ID feature as it is a unique identifier and does not adds any value

df.drop(['ID'], axis=1, inplace=True)
```

In [16]:

```
# separating Target Feature
x = df.drop(['count'], axis=1)
y = df['count']

# validiating the shapes
x.shape, y.shape
```

Out[16]:

```
((12980, 7), (12980,))
```

In [17]:

```
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.linear_model import LinearRegression
```

In [20]:

```
# the only change will be in FORWARD parameter

LR = LinearRegression()

Forward_Selector = sfs(LR, k_features= 4, forward= True, verbose= 2, scoring= 'neg_mean_squared_error')
```

In [21]:

```
# Training model
Forward_Selector = Forward_Selector.fit(x,y)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.0s finished

[2023-02-09 03:28:52] Features: 1/4 -- score: -23364.955503180994[Parallel(n_jobs=1)]: Using backend SequentialBackend
with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s finished

[2023-02-09 03:28:52] Features: 2/4 -- score: -21454.899339219788[Parallel(n_jobs=1)]: Using backend SequentialBackend
with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished

[2023-02-09 03:28:52] Features: 3/4 -- score: -21458.27878856439[Parallel(n_jobs=1)]: Using backend SequentialBackend w
ith 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s finished

[2023-02-09 03:28:53] Features: 4/4 -- score: -21477.988573502793
```

In [22]:

```
Forward_Selector.k_score_
# this is the Negative Mean Squared Error
```

Out[22]:

```
-21477.988573502793
```

In [23]:

```
selected_features = list(Forward_Selector.k_feature_names_)
selected_features
```

Out[23]:

```
['holiday', 'workingday', 'temp', 'humidity']
```

In [24]:

```
new_df = df[selected_features]

# adding back the Target Feature to new dataframe
new_df['count'] = df['count']
new_df.head()
```

Out[24]:

	holiday	workingday	temp	humidity	count
0	0	0	9.84	81	16
1	0	0	9.02	80	40
2	0	0	9.02	80	32
3	0	0	9.84	75	13
4	0	0	9.84	75	1

In [ ]:

```
1
```