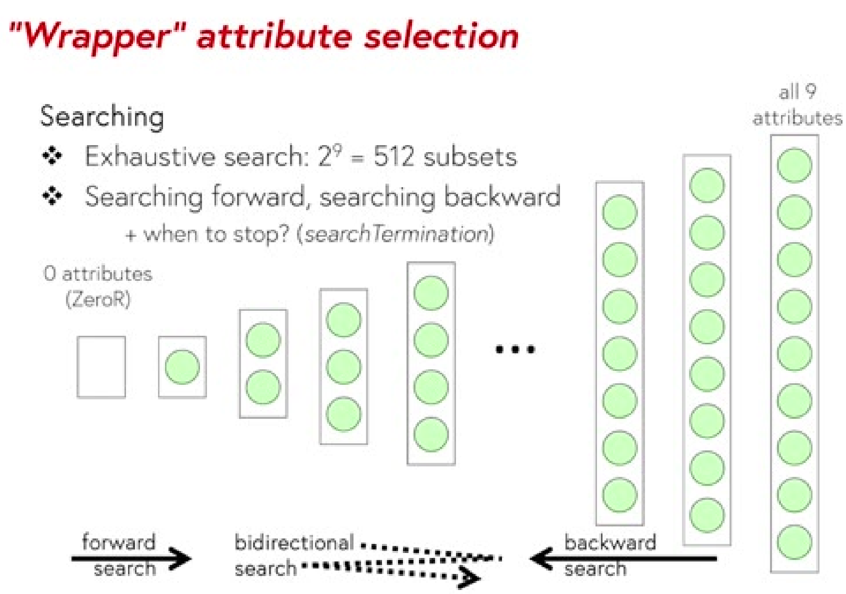
What is Feature Selection?

Feature Selection is the method of reducing the input variable to your model by using only relevant data and getting rid of noise in data. It is the process of automatically choosing relevant features for your machine learning model based on the type of problem you are trying to solve.

- **Types of Feature Selection**

Wrapper Method

Wrapper methods for feature selection are a type of feature selection methods that involve using a predictive model to score the combination of features. They are called "wrapper" methods because they "wrap" this type of model-based evaluation around the feature selection process.

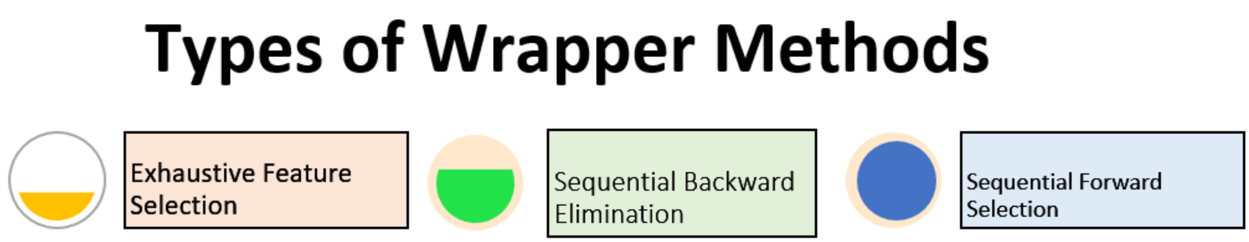


Here's how wrapper methods work in general:

1. **Subset Generation**: First, a subset of features is generated. This can be done in a variety of ways. For example, you might start with one feature and gradually add more, or start with all features and gradually remove them, or generate subsets of features randomly. The subset generation method depends on the specific type of wrapper method being used.

2. **Subset Evaluation**: After a subset of features has been generated, a model is trained on this subset of features, and the model's performance is evaluated, usually through cross validation. The performance of the model gives an estimate of the quality of the features in the subset.

3. **Stopping Criterion**: This process is repeated, generating and evaluating different subsets of features, until some stopping criterion is met. This could be a certain number of subsets evaluated, a certain amount of time elapsed, or no improvement in model performance after a certain number of iterations



1. Exhaustive Feature Selection/Best Subset Selection

Exhaustive feature selection, also known as best subset selection, is a method used in machine learning and statistics to identify the best combination of features (predictors) for a given predictive model. The goal is to select a subset of features that yields the most accurate and interpretable model.

The exhaustive feature selection technique evaluates all possible feature combinations and selects the one that achieves the best performance according to a chosen evaluation criterion, such as accuracy, mean squared error, or area under the curve. The process involves systematically evaluating models with different subsets of features and comparing their performance to determine the optimal subset.

Here's a general outline of the exhaustive feature selection process:

1. **Define the feature space:** Start by defining the set of features available for selection. This could include numerical, categorical, or binary variables.

2. **Generate all possible feature subsets:** Enumerate all possible combinations of features from the defined feature space. This can be done using combinatorial techniques such as power set generation.

3. **Build and evaluate models:** For each feature subset, build a predictive model using the selected algorithm (e.g., linear regression, decision tree, or support vector machine). Train the model using a suitable training dataset and evaluate its performance using an appropriate evaluation metric.

4. **Select the best subset:** Compare the performance of models obtained from different feature subsets and select the one that maximizes the chosen evaluation metric. This could be the subset with the highest accuracy or the lowest error, depending on the problem at hand.

5. **Validate the selected subset:** After identifying the best subset using the training dataset, it is crucial to assess the selected subset's performance on a separate validation dataset. This step helps verify that the selected subset generalizes well to unseen data.

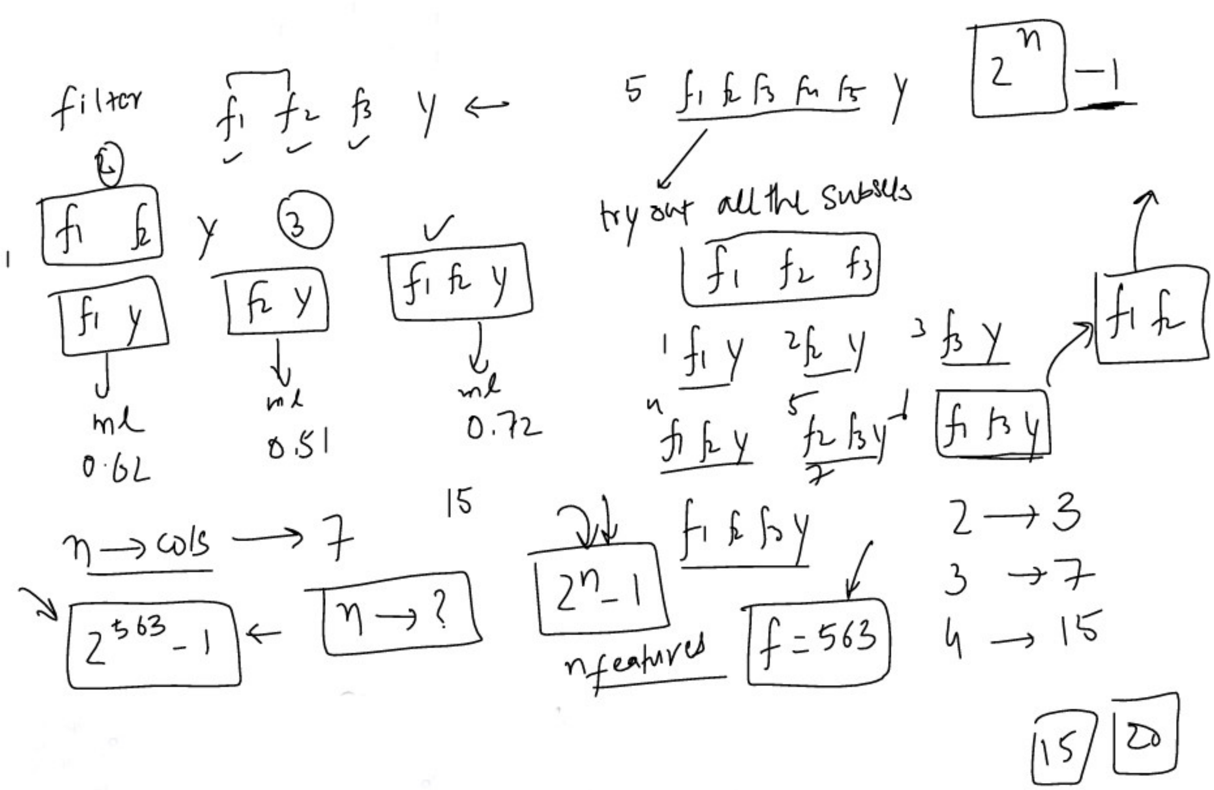
6. **Interpret the results:** Once the best subset is determined, analyze the selected features to gain insights into their importance and relationships with the target variable. Interpretability is one of the advantages of best

It's important to note that exhaustive feature selection can be computationally expensive, especially when the feature space is large. As the number of features increases, the number of possible subsets grows exponentially. Therefore, it may be necessary to employ strategies like early stopping or use heuristics to reduce the search space or limit the subset size.

Additionally, alternative feature selection techniques, such as forward selection (adding features one by one) or backward elimination (removing features one by one), can be considered to strike a balance between computational complexity and performance.

Overall, exhaustive feature selection can be a powerful approach for finding the optimal subset of features, but it requires careful consideration of computational resources and proper validation to ensure reliable results.

Explanation:




```
In [1]: # Code
```

In [2]:
```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.model_selection import cross_val_score
import pandas as pd
from sklearn.model_selection import cross_val_score
```

In [3]:
```
!pip install --upgrade scikit-learn mlxtend
```

```
Requirement already satisfied: scikit-learn in c:\users\user\anaconda3\lib\site-packages (1.
2.2)
Requirement already satisfied: mlxtend in c:\users\user\anaconda3\lib\site-packages (0.22.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\user\anaconda3\lib\site-packages (f
rom scikit-learn) (1.20.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\user\anaconda3\lib\site-pack
ages (from scikit-learn) (2.2.0)
Requirement already satisfied: scipy>=1.3.2 in c:\users\user\anaconda3\lib\site-packages (fr
om scikit-learn) (1.7.1)
Requirement already satisfied: joblib>=1.1.1 in c:\users\user\anaconda3\lib\site-packages (f
rom scikit-learn) (1.2.0)
Requirement already satisfied: setuptools in c:\users\user\anaconda3\lib\site-packages (from
mlxtend) (58.0.4)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\user\anaconda3\lib\site-package
s (from mlxtend) (3.4.3)
Requirement already satisfied: pandas>=0.24.2 in c:\users\user\anaconda3\lib\site-packages
(from mlxtend) (1.3.4)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\user\anaconda3\lib\site-packages
(from matplotlib>=3.0.0->mlxtend) (3.0.4)
Requirement already satisfied: pillow>=6.2.0 in c:\users\user\anaconda3\lib\site-packages (f
rom matplotlib>=3.0.0->mlxtend) (8.4.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\user\anaconda3\lib\site-pack
ages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: cycler>=0.10 in c:\users\user\anaconda3\lib\site-packages (fr
om matplotlib>=3.0.0->mlxtend) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\user\anaconda3\lib\site-package
s (from matplotlib>=3.0.0->mlxtend) (1.3.1)
Requirement already satisfied: six in c:\users\user\anaconda3\lib\site-packages (from cycler
>=0.10->matplotlib>=3.0.0->mlxtend) (1.16.0)
Requirement already satisfied: pytz>=2017.3 in c:\users\user\anaconda3\lib\site-packages (fr
om pandas>=0.24.2->mlxtend) (2021.3)
```

In [4]:
```
# Read the CSV file from the given URL

df = pd.read_csv('https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/0e7a9b0a5

df.head()
```

Out[4]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [5]:
```
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS

lr = LogisticRegression()

sel = EFS(lr, max_features=4, scoring='accuracy', cv=5)
```

In [6]:
```
# Fit the model using the first four columns of the DataFrame as features and the 'species' co


model = sel.fit(df.iloc[:,:4],df['species'])
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/sta
ble/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://s
cikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
C:\Users\user\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/sta
ble/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://s
cikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
C:\Users\user\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/sta
ble/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://s
cikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
Features: 15/15
```

In [7]:
```
# Retrieve the best score from the model

model.best_score_
```

Out[7]:
```
0.9733333333333334
```

In [8]:
```
# Print the best feature names from the model

model.best_feature_names_
```

Out[8]:
```
('sepal_width', 'petal_length', 'petal_width')
```

In [9]:
```
# detailed output

model.subsets_
```

Out[9]:
```
{0: {'feature_idx': (0,),
  'cv_scores': array([0.66666667, 0.73333333, 0.76666667, 0.76666667, 0.83333333]),
  'avg_score': 0.7533333333333333,
  'feature_names': ('sepal_length',)},
 1: {'feature_idx': (1,),
  'cv_scores': array([0.53333333, 0.56666667, 0.53333333, 0.53333333, 0.63333333]),
  'avg_score': 0.5599999999999999,
  'feature_names': ('sepal_width',)},
 2: {'feature_idx': (2,),
  'cv_scores': array([0.93333333, 1.        , 0.9       , 0.93333333, 1.        ]),
  'avg_score': 0.9533333333333334,
  'feature_names': ('petal_length',)},
 3: {'feature_idx': (3,),
  'cv_scores': array([1.        , 0.96666667, 0.9       , 0.93333333, 1.        ]),
  'avg_score': 0.96,
  'feature_names': ('petal_width',)},
 4: {'feature_idx': (0, 1),
  'cv_scores': array([0.73333333, 0.83333333, 0.76666667, 0.86666667, 0.9       ]),
  'avg_score': 0.8200000000000001,
  'feature_names': ('sepal_length', 'sepal_width')},
 5: {'feature_idx': (0, 2),
  'cv_scores': array([0.93333333, 1.        , 0.9       , 0.93333333, 1.        ]),
  'avg_score': 0.9533333333333334,
  'feature_names': ('sepal_length', 'petal_length')},
 6: {'feature_idx': (0, 3),
  'cv_scores': array([0.93333333, 0.96666667, 0.93333333, 0.93333333, 1.        ]),
  'avg_score': 0.9533333333333334,
  'feature_names': ('sepal_length', 'petal_width')},
 7: {'feature_idx': (1, 2),
  'cv_scores': array([0.93333333, 1.        , 0.9       , 0.93333333, 1.        ]),
  'avg_score': 0.9533333333333334,
  'feature_names': ('sepal_width', 'petal_length')},
 8: {'feature_idx': (1, 3),
  'cv_scores': array([0.93333333, 0.96666667, 0.9       , 0.93333333, 0.96666667]),
  'avg_score': 0.9400000000000001,
  'feature_names': ('sepal_width', 'petal_width')},
 9: {'feature_idx': (2, 3),
  'cv_scores': array([0.96666667, 0.96666667, 0.93333333, 0.93333333, 1.        ]),
  'avg_score': 0.96,
  'feature_names': ('petal_length', 'petal_width')},
 10: {'feature_idx': (0, 1, 2),
  'cv_scores': array([0.93333333, 1.        , 0.9       , 0.93333333, 1.        ]),
  'avg_score': 0.9533333333333334,
  'feature_names': ('sepal_length', 'sepal_width', 'petal_length')},
 11: {'feature_idx': (0, 1, 3),
  'cv_scores': array([0.9       , 0.96666667, 0.93333333, 0.93333333, 1.        ]),
  'avg_score': 0.9466666666666667,
  'feature_names': ('sepal_length', 'sepal_width', 'petal_width')},
 12: {'feature_idx': (0, 2, 3),
  'cv_scores': array([0.96666667, 0.96666667, 0.93333333, 0.96666667, 1.        ]),
  'avg_score': 0.9666666666666668,
  'feature_names': ('sepal_length', 'petal_length', 'petal_width')},
 13: {'feature_idx': (1, 2, 3),
  'cv_scores': array([0.96666667, 1.        , 0.93333333, 0.96666667, 1.        ]),
  'avg_score': 0.9733333333333334,
  'feature_names': ('sepal_width', 'petal_length', 'petal_width')},
 14: {'feature_idx': (0, 1, 2, 3),
  'cv_scores': array([0.96666667, 1.        , 0.93333333, 0.96666667, 1.        ]),
  'avg_score': 0.9733333333333334,
  'feature_names': ('sepal_length',
   'sepal_width',
   'petal_length',
   'petal_width')}}
```

In [10]:
```
# Create a DataFrame from the metric dictionary returned by the model and Transpose

metric_df = pd.DataFrame.from_dict(model.get_metric_dict()).T

"""In the above line, a DataFrame named metric_df is created by converting the metric dictiona
   returned by the model.get_metric_dict() function into a DataFrame using the pd.DataFrame.
  from_dict() method. The .T at the end is used to transpose the DataFrame, swapping the rows


metric_df
```

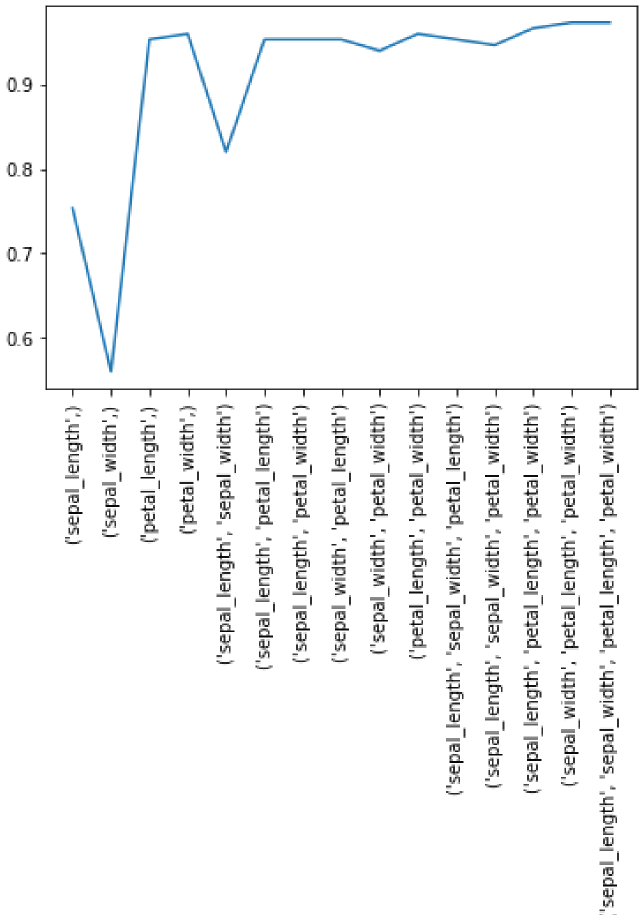
Out[10]:

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
0	(0,)	[0.6666666666666666, 0.7333333333333333, 0.766...	0.753333	(sepal_length,)	0.069612	0.05416	0.02708
1	(1,)	[0.5333333333333333, 0.5666666666666667, 0.533...	0.56	(sepal_width,)	0.049963	0.038873	0.019437
2	(2,)	[0.9333333333333333, 1.0, 0.9, 0.9333333333333...	0.953333	(petal_length,)	0.051412	0.04	0.02
3	(3,)	[1.0, 0.9666666666666667, 0.9, 0.9333333333333...	0.96	(petal_width,)	0.049963	0.038873	0.019437
4	(0, 1)	[0.7333333333333333, 0.8333333333333334, 0.766...	0.82	(sepal_length, sepal_width)	0.079462	0.061824	0.030912
5	(0, 2)	[0.9333333333333333, 1.0, 0.9, 0.9333333333333...	0.953333	(sepal_length, petal_length)	0.051412	0.04	0.02
6	(0, 3)	[0.9333333333333333, 0.9666666666666667, 0.933...	0.953333	(sepal_length, petal_width)	0.034274	0.026667	0.013333
7	(1, 2)	[0.9333333333333333, 1.0, 0.9, 0.9333333333333...	0.953333	(sepal_width, petal_length)	0.051412	0.04	0.02
8	(1, 3)	[0.9333333333333333, 0.9666666666666667, 0.9, ...	0.94	(sepal_width, petal_width)	0.032061	0.024944	0.012472
9	(2, 3)	[0.9666666666666667, 0.9666666666666667, 0.933...	0.96	(petal_length, petal_width)	0.032061	0.024944	0.012472
10	(0, 1, 2)	[0.9333333333333333, 1.0, 0.9, 0.9333333333333...	0.953333	(sepal_length, sepal_width, petal_length)	0.051412	0.04	0.02
11	(0, 1, 3)	[0.9, 0.9666666666666667, 0.9333333333333333, ...	0.946667	(sepal_length, sepal_width, petal_width)	0.043691	0.033993	0.016997
12	(0, 2, 3)	[0.9666666666666667, 0.9666666666666667, 0.933...	0.966667	(sepal_length, petal_length, petal_width)	0.027096	0.021082	0.010541
13	(1, 2, 3)	[0.9666666666666667, 1.0, 0.9333333333333333, ...	0.973333	(sepal_width, petal_length, petal_width)	0.032061	0.024944	0.012472
14	(0, 1, 2, 3)	[0.9666666666666667, 1.0, 0.9333333333333333, ...	0.973333	(sepal_length, sepal_width, petal_length, peta...	0.032061	0.024944	0.012472

In [11]:
```
import matplotlib.pyplot as plt

# Plotting the average scores

plt.plot([str(k) for k in metric_df['feature_names']],metric_df['avg_score'])
plt.xticks(rotation=90)
plt.show()
```



For Regression Example

In [12]:
```
df = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv
df.head()
```

Out[12]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

In [13]:
```
from sklearn.model_selection import train_test_split


# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,:-1], df['medv'], test_size=0.2
```

In [14]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(404, 13)
(102, 13)
```

In [15]:
```
X_train.head()
```

Out[15]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat
42	0.14150	0.0	6.91	0	0.448	6.169	6.6	5.7209	3	233	17.9	383.37	5.81
58	0.15445	25.0	5.13	0	0.453	6.145	29.2	7.8148	8	284	19.7	390.68	6.86
385	16.81180	0.0	18.10	0	0.700	5.277	98.1	1.4261	24	666	20.2	396.90	30.81
78	0.05646	0.0	12.83	0	0.437	6.232	53.7	5.0141	5	398	18.7	386.40	12.34
424	8.79212	0.0	18.10	0	0.584	5.565	70.6	2.0635	24	666	20.2	3.65	17.16

In [16]:
```
from sklearn.preprocessing import StandardScaler

# Create an instance of the StandardScaler class
sc = StandardScaler()

# Apply the fit_transform method to the training data
X_train = sc.fit_transform(X_train)

# Apply the transform method to the test data
X_test = sc.transform(X_test)
```

In [17]:
```
# baseline model

import numpy as np
from sklearn.metrics import r2_score
model = LinearRegression()


print("training",np.mean(cross_val_score(model, X_train, y_train, cv=5, scoring='r2')))
print("testing",np.mean(cross_val_score(model, X_test, y_test, cv=5, scoring='r2')))
```

```
training 0.7025123301096212
testing 0.6514899901155405
```

In [18]:
```
# Create a LinearRegression object

lr = LinearRegression()

# Create an ExhaustiveFeatureSelector object
exh = EFS(lr, max_features=13, scoring='r2', cv=10, print_progress=True,n_jobs=-1)

# Fit the selector to the training data
sel = exh.fit(X_train, y_train)
```

```
Features: 8191/8191
```

In [19]:
```
# Get the best score from sel
sel.best_score_
```

Out[19]:
```
0.6827988156800064
```

In [20]:
```
# Retrieve the best feature names
sel.best_feature_names_
```

Out[20]:
```
('0', '1', '4', '5', '7', '8', '9', '10', '11', '12')
```

In [21]:
```

# Create a DataFrame from the metric dictionary and transpose it
metric_df = pd.DataFrame.from_dict(sel.get_metric_dict()).T
metric_df

""" The metric_df DataFrame has several columns:
    feature_idx: It represents the indices of the selected features.
    cv_scores: It contains a list of cross-validated scores for each feature combination.
    avg_score: It represents the average score for each feature combination.
    feature_names: It contains the names of the selected features.
    ci_bound: It represents the confidence interval bound for each feature combination.
    std_dev: It represents the standard deviation of the scores for each feature combination.
    std_err: It represents the standard error of the scores for each feature combination. """
```

Out[21]:
```
' The metric_df DataFrame has several columns:\n    feature_idx: It represents the indices o
f the selected features.\n    cv_scores: It contains a list of cross-validated scores for ea
ch feature combination.\n    avg_score: It represents the average score for each feature com
bination.\n    feature_names: It contains the names of the selected features.\n    ci_bound:
It represents the confidence interval bound for each feature combination.\n    std_dev: It r
epresents the standard deviation of the scores for each feature combination.\n    std_err: I
t represents the standard error of the scores for each feature combination. '
```

In [22]:
```
def adjust_r2(r2, num_examples, num_features):
    coef = (num_examples - 1) / (num_examples - num_features - 1)
    return 1 - (1 - r2) * coef
"""Adjusts the R-squared value based on the number of examples and features.

    Args:
        r2 (float): The original R-squared value.
        num_examples (int): The number of examples in the dataset.
        num_features (int): The number of features in the dataset.

    Returns:
        float: The adjusted R-squared value"""
```

Out[22]:
```
'Adjusts the R-squared value based on the number of examples and features.\n\n    Args:\n
r2 (float): The original R-squared value.\n        num_examples (int): The number of example
s in the dataset.\n        num_features (int): The number of features in the dataset.\n\n
Returns:\n        float: The adjusted R-squared value'
```

In [23]:
```
# Add comments to explain what the code is doing.
# Set the value of 'observations' column in 'metric_df' to 404.
metric_df['observations'] = 404

# Calculate the number of features for each row and store the result in 'num_features' column.
metric_df['num_features'] = metric_df['feature_idx'].apply(lambda x:len(x))

# Calculate the adjusted R2 score using the 'avg_score', 'observations', and 'num_features' co
# and store the result in 'adjusted_r2' column.
metric_df['adjusted_r2'] = adjust_r2(metric_df['avg_score'],metric_df['observations'],metric_
```

In [24]:
```

# Sort the metric_df dataframe in descending order based on the 'adjusted_r2' column
metric_df.sort_values('adjusted_r2',ascending=False)
```

Out[24]:

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err	observations	r
7975	(0, 1, 4, 5, 7, 8, 9, 10, 11, 12)	[0.8855189158291971, 0.5742220049707853, 0.437...	0.682799	(0, 1, 4, 5, 7, 8, 9, 10, 11, 12)	0.096995	0.130595	0.043532	404	
7408	(0, 1, 4, 5, 7, 8, 9, 10, 12)	[0.8717831363927702, 0.581930780098259, 0.4623...	0.680483	(0, 1, 4, 5, 7, 8, 9, 10, 12)	0.090811	0.122269	0.040756	404	
8141	(0, 1, 2, 4, 5, 7, 8, 9, 10, 11, 12)	[0.8792702841985806, 0.5752245789381275, 0.438...	0.681125	(0, 1, 2, 4, 5, 7, 8, 9, 10, 11, 12)	0.096068	0.129348	0.043116	404	
8150	(0, 1, 3, 4, 5, 7, 8, 9, 10, 11, 12)	[0.8734082301119793, 0.5381382515761797, 0.461...	0.680994	(0, 1, 3, 4, 5, 7, 8, 9, 10, 11, 12)	0.098795	0.133019	0.04434	404	
8153	(0, 1, 4, 5, 6, 7, 8, 9, 10, 11, 12)	[0.8853169531726774, 0.5751761822045904, 0.434...	0.680914	(0, 1, 4, 5, 6, 7, 8, 9, 10, 11, 12)	0.097075	0.130703	0.043568	404	
...	...	...	...	...	...	...	...	...	
53	(3, 11)	[0.07227421305699011, -0.026141441832760126, 0...	0.073485	(3, 11)	0.069934	0.09416	0.031387	404	
11	(11,)	[0.1200629474726852, 0.03143835749752166, -0.0...	0.068712	(11,)	0.071116	0.095752	0.031917	404	
49	(3, 7)	[-0.0371219722713414, -0.16717603954280014, 0....	0.057453	(3, 7)	0.09446	0.127183	0.042394	404	
7	(7,)	[0.004822573124353857, -0.09518844023749029, -...	0.038815	(7,)	0.066813	0.089958	0.029986	404	
3	(3,)	[-0.07110886674980432, -0.08269807310551558, 0...	-0.025663	(3,)	0.055426	0.074627	0.024876	404	

8191 rows × 10 columns

In [25]:
```
X_train_sel = sel.transform(X_train)
X_test_sel = sel.transform(X_test)

# With 10 columns - (0, 1, 4, 5, 7, 8, 9, 10, 11, 12)
```

In [26]:
```
model = LinearRegression()

# Print the mean R2 score for training & testing data
print("training",np.mean(cross_val_score(model, X_train_sel, y_train, cv=5, scoring='r2')))
print("testing",np.mean(cross_val_score(model, X_test_sel, y_test, cv=5, scoring='r2')))
```

```
training 0.7100327839218561
testing 0.7205819296124483
```

Disadvantages of Exhaustive Feature Selection

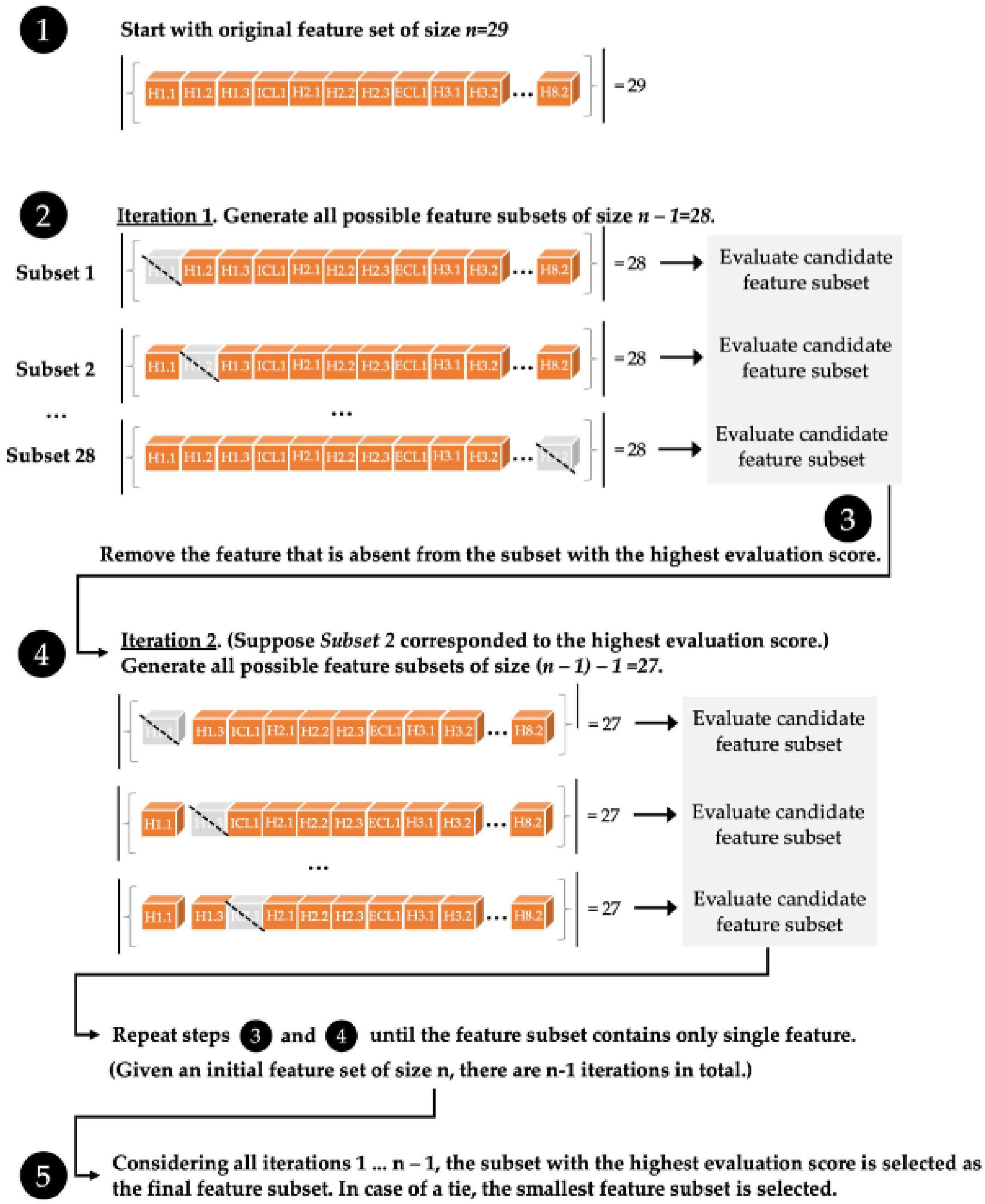
- **Computational Complexity**: The biggest drawback is its computational cost. If you have n features, the number of combinations to check is 2^n. So, as the number of features grows, the number of combinations grows exponentially, making this method computationally expensive and time-consuming. For datasets with a large number of features, it may not be practical.

- **Risk of Overfitting**: By checking all possible combinations of features, there's a risk of overfitting the model to the training data. The feature combination that performs best on the training data may not necessarily perform well on unseen data.

- **Requires a Good Evaluation Metric**: The effectiveness of exhaustive feature selection depends on the quality of the evaluation metric used to assess the goodness of a feature subset. If a poor metric is used, the feature selection may not yield optimal results.

2.Sequential Backward Selection/Elimination

Sequential Backward Selection (SBS), also known as Sequential Backward Elimination (SBE), is a feature selection technique that starts with the full set of features and iteratively removes one feature at a time until a desired number of features is reached. The goal is to eliminate the least informative features while maintaining or improving the performance of the model.



Here's a general outline of the Sequential Backward Selection (SBS) process:

1. **Define the feature space:** Start by defining the set of features available for selection. This could include numerical, categorical, or binary variables.
2. **Initialize the feature subset:** Begin with the full set of features.
3. **Build and evaluate the model:** Train a predictive model using the selected algorithm (e.g., linear regression, decision tree, or support vector machine) on the current feature subset. Evaluate the model's performance using an appropriate evaluation metric.
4. **Feature removal:** Remove one feature at a time from the current feature subset, evaluate the model's performance, and keep track of the performance change.
5. **Select the feature to remove:** Identify the feature whose removal results in the smallest performance change (e.g., the smallest increase in error or decrease in accuracy).

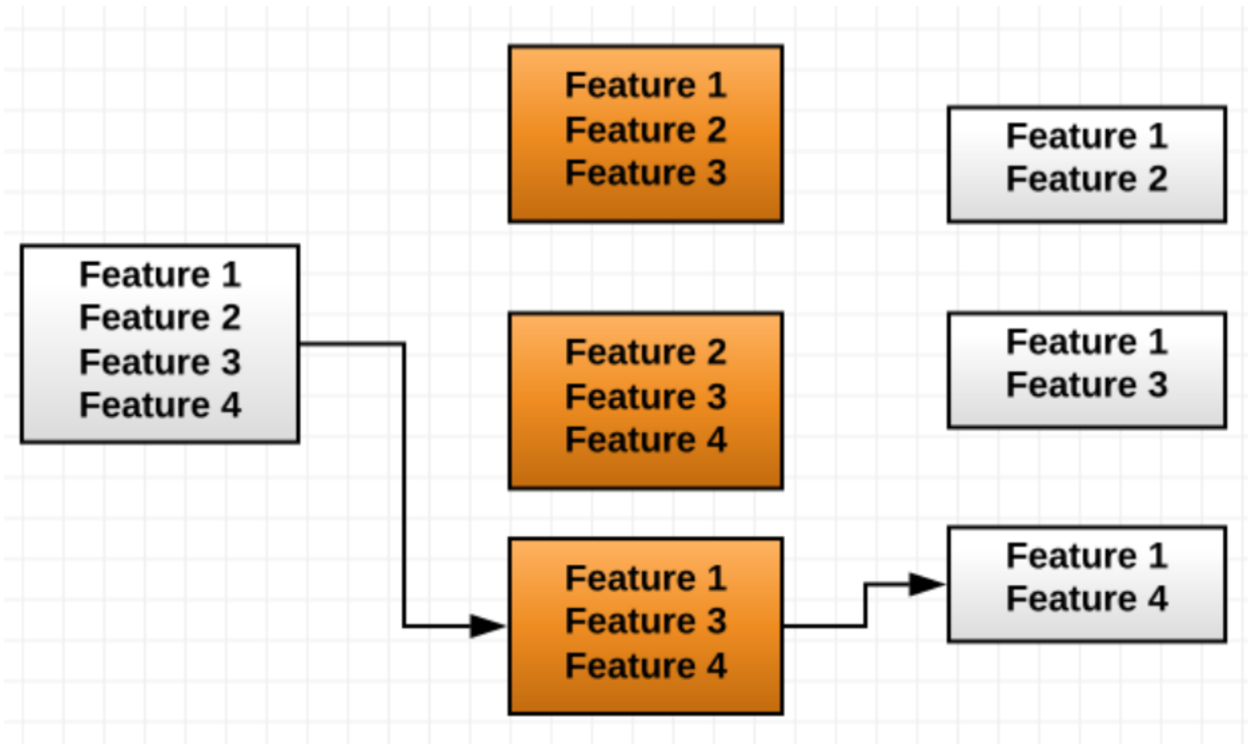
6. **Stopping criterion:** Check if the desired number of features has been reached or if the performance falls below a predefined threshold. If the criterion is not met, repeat steps 3 to 6.

7. **Finalize the feature subset:** Once the stopping criterion is met, finalize the selected feature subset.

It's important to note that the order in which features are removed can impact the performance of the algorithm. SBS typically removes one feature at a time based on the performance change observed when that feature is removed. The process continues until the desired number of features is obtained or the performance falls below the defined threshold.

Scikit-learn provides an implementation of Sequential Backward Selection as part of the `mlxtend` package. You can use the `SequentialFeatureSelector` class from `mlxtend.feature_selection` to perform Sequential Backward Selection.

Explanation :



In [27]:
```
# Code

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from mlxtend.feature_selection import SequentialFeatureSelector as SFS

# Load the dataset
data = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.cs


data
```

Out[27]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	11.9

506 rows × 14 columns

In [28]:
```
# separate the target variable
X = data.drop("medv", axis=1)
y = data['medv']

# split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

In [29]:
```
print(X_train.shape)
```

```
(404, 13)
```

In [30]:
```
# Importing the necessary libraries
from sklearn.preprocessing import StandardScaler

# Creating an instance of the StandardScaler class
sc = StandardScaler()

# Scaling the training data
X_train = sc.fit_transform(X_train)

# Scaling the testing data
X_test = sc.transform(X_test)
```

In [31]:
```
model = LinearRegression()

print("training",np.mean(cross_val_score(model, X_train, y_train, cv=5, scoring='r2')))
print("testing",np.mean(cross_val_score(model, X_test, y_test, cv=5, scoring='r2')))
```

```
training 0.7025123301096212
testing 0.6514899901155405
```

In [32]:
```
### backward elimination


lr = LinearRegression()

# perform backward elimination
sfs = SFS(lr, k_features='best', forward=False, floating=False, scoring='r2',cv=5)

sfs.fit(X_train, y_train)
```

Out[32]:
```
▸ SequentialFeatureSelector
▸ estimator: LinearRegression
    ▸ LinearRegression
```

In [33]:
```
# Get the indices of the selected features in the Focal cell
sfs.k_feature_idx_
```

Out[33]:
```
(0, 1, 4, 5, 7, 8, 9, 10, 11, 12)
```

In [34]:
```
# Create a DataFrame from the metric dictionary
metric_df = pd.DataFrame.from_dict(sfs.get_metric_dict()).T

# Add additional columns to the DataFrame
metric_df['observations'] = 404

"""This code will assign the value 404 to every row in the 'observations' column of the 'metri


metric_df['num_features'] = metric_df['feature_idx'].apply(lambda x: len(x))

"""In this code, the apply function is applied to each element in the 'feature_idx' column usi
The lambda function calculates the length of each feature index list x.
The resulting lengths are then assigned to the 'num_features' column in the 'metric_df' DataFr


metric_df['adjusted_r2'] = adjust_r2(metric_df['avg_score'],
                                     metric_df['observations'],
                                     metric_df['num_features'])

metric_df
```

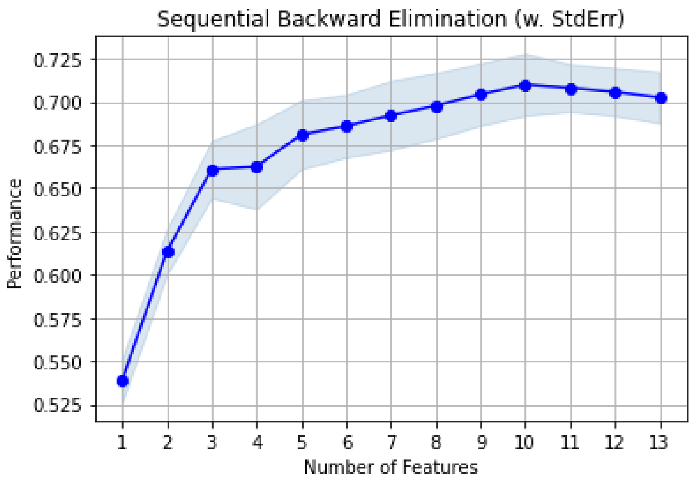
Out[34]:

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err	observations	num_
13	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)	[0.7535027170817177, 0.6920238509138779, 0.682...	0.702512	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)	0.038207	0.029727	0.014863	404	
12	(0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12)	[0.7532855958710692, 0.6944570477695304, 0.693...	0.70581	(0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12)	0.035641	0.02773	0.013865	404	
11	(0, 1, 3, 4, 5, 7, 8, 9, 10, 11, 12)	[0.7547108925568491, 0.6959627893665097, 0.701...	0.708109	(0, 1, 3, 4, 5, 7, 8, 9, 10, 11, 12)	0.035367	0.027516	0.013758	404	
10	(0, 1, 4, 5, 7, 8, 9, 10, 11, 12)	[0.7769593921905562, 0.6884741223718953, 0.702...	0.710033	(0, 1, 4, 5, 7, 8, 9, 10, 11, 12)	0.046075	0.035848	0.017924	404	
9	(0, 1, 4, 5, 7, 8, 9, 10, 12)	[0.7706104220711025, 0.6854023389684325, 0.690...	0.704324	(0, 1, 4, 5, 7, 8, 9, 10, 12)	0.046449	0.036139	0.018069	404	
8	(0, 1, 4, 5, 7, 8, 10, 12)	[0.7681719744800459, 0.6822126526818693, 0.670...	0.697727	(0, 1, 4, 5, 7, 8, 10, 12)	0.04882	0.037984	0.018992	404	
7	(0, 1, 4, 5, 7, 10, 12)	[0.7671638009750725, 0.6812300799626649, 0.661...	0.692234	(0, 1, 4, 5, 7, 10, 12)	0.051644	0.040181	0.02009	404	
6	(1, 4, 5, 7, 10, 12)	[0.7519120213497092, 0.6756087674652563, 0.646...	0.686004	(1, 4, 5, 7, 10, 12)	0.046845	0.036447	0.018224	404	
5	(4, 5, 7, 10, 12)	[0.7525552802357769, 0.6665033988504306, 0.639...	0.681065	(4, 5, 7, 10, 12)	0.051233	0.039861	0.019931	404	
4	(5, 7, 10, 12)	[0.7384743962575442, 0.6401188507668829, 0.587...	0.662544	(5, 7, 10, 12)	0.063384	0.049315	0.024658	404	
3	(5, 10, 12)	[0.7215896884753016, 0.6288372046797153, 0.633...	0.661012	(5, 10, 12)	0.04259	0.033136	0.016568	404	
2	(5, 12)	[0.6330856272904801, 0.5779812120755249, 0.586...	0.613259	(5, 12)	0.034066	0.026505	0.013252	404	
1	(12,)	[0.5472998394577442, 0.49002001493399727, 0.53...	0.538451	(12,)	0.032755	0.025485	0.012742	404	

In [35]:
```
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs


fig1 = plot_sfs(sfs.get_metric_dict(), kind='std_err',)

plt.title('Sequential Backward Elimination (w. StdErr)')
plt.grid()
plt.show()
```



In [36]:
```
# Transform the features of the training set
X_train_sel = sfs.transform(X_train)

# Transform the features of the test set
X_test_sel = sfs.transform(X_test)

# Create a linear regression model
model = LinearRegression()

# Print the mean R^2 score for training set
print("Training R^2:", np.mean(cross_val_score(model, X_train_sel, y_train, cv=5, scoring='r2

# Print the mean R^2 score for test set
print("Testing R^2:", np.mean(cross_val_score(model, X_test_sel, y_test, cv=5, scoring='r2'))
```

```
Training R^2: 0.7100327839218561
Testing R^2: 0.7205819296124483
```

In [37]:
```
# Display the shape of X_train_sel
X_train_sel.shape
```

Out[37]:
```
(404, 10)
```

In [38]:
```
### Using Sk-Learn

from sklearn.feature_selection import SequentialFeatureSelector as SFS


sfs2 = SFS(model,
          n_features_to_select=5,
          direction='forward',
          scoring='r2',
          n_jobs=-1,
          cv=5)

sfs2 = sfs2.fit(X_train, y_train)
```

In [39]:
```
# Retrieve the indices of the selected features
np.arange(X.shape[1])[sfs2.support_]
```

Out[39]:
```
array([ 5,  7, 10, 11, 12])
```

https://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/ (https://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/)

3. Sequential Forward Selection

Sequential Forward Selection (SFS) is a feature selection technique that starts with an empty set of features and iteratively adds one feature at a time until a desired number of features is reached. The goal is to select the most informative features while maintaining or improving the performance of the model.

Here's a general outline of the Sequential Forward Selection (SFS) process:

1. **Define the feature space:** Start by defining the set of features available for selection. This could include numerical, categorical, or binary variables.

2. **Initialize the feature subset:** Begin with an empty set of features.

3. **Build and evaluate the model:** Train a predictive model using the selected algorithm (e.g., linear regression, decision tree, or support vector machine) on the current feature subset. Evaluate the model's performance using an appropriate evaluation metric.

4. **Feature selection:** Add one feature at a time from the remaining set of features to the current feature subset. Evaluate the model's performance after adding each feature and keep track of the performance change.

5. **Select the best feature:** Identify the feature whose addition results in the largest performance improvement (e.g., the largest decrease in error or increase in accuracy).

6. **Stopping criterion:** Check if the desired number of features has been reached or if the performance improvement falls below a predefined threshold. If the criterion is not met, repeat steps 3 to 5.

7. **Finalize the feature subset:** Once the stopping criterion is met, finalize the selected feature subset.

Scikit-learn does not provide a built-in implementation of Sequential Forward Selection, but you can implement it manually using the available feature selection techniques. Here's an example of how to perform Sequential Forward Selection:

Explanation:

features → $\boxed{f_3\ f_4}$ → $\boxed{\text{best}}$

1 $\boxed{f_1}$ $\boxed{f_2}$ $\boxed{f_3}$ $\boxed{f_4}$ → model → 0.89

2
- f1 $\boxed{f_2}$ $\boxed{f_3}$ $\boxed{f_4}$ → model → 0.81
- $\boxed{f_1}$ f2 $\boxed{f_3}$ $\boxed{f_4}$ → m → 0.71
- $\boxed{f_1}$ $\boxed{f_2}$ f3 $\boxed{f_4}$ → m → $\boxed{0.91}$
- $\boxed{f_1}$ $\boxed{f_2}$ $\boxed{f_3}$ f4 → m → 0.65

3
- f1 $\boxed{f_2}$ $\boxed{f_4}$ → 0.79
- $\boxed{f_1}$ f2 $\boxed{f_4}$ → 0.81
- $\boxed{f_1}$ $\boxed{f_2}$ f4 → 0.81

4
- f1 $\boxed{f_2}$ → 0.63
- $\boxed{f_1}$ f2 → 0.53

f1 f2 f3 f4	0.89
f1 f2 f4	0.91
f1 f2	0.83
f2	0.63

$\boxed{f_1\ f_2\ f_4}$ → best feature subset

model train → result

faster → $n(n+1)$ → $1+2+3+\dots n$

Ex → 2^n-1

$\frac{13 \times 14}{2} = 13 \times 7$ (81)

- 4 → 4
- 3 → 3
- 2 → 2
- 1 → 1

n → n
n-1
n-2
⋮
1

(13) cols

sussy per itera

$1+2+3+\dots+n = \boxed{\frac{n(n+1)}{2}}$

$\boxed{100}$ cols

$2^{100}-1$

n^2 →

$\frac{100 \times 101}{2} = 50 \times 101$ $\boxed{5050}$

In [42]:
```
### Forward Selection
from mlxtend.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LinearRegression

lr = LinearRegression()

# perform Forward Selection
sfs = SFS(lr, k_features='best', forward=True, floating=False, scoring='r2',cv=5) # Change , F

sfs.fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_28640/3641867140.py in <module>
      6
      7 # perform Forward Selection
----> 8 sfs = SFS(lr, k_features='best', forward=True, floating=False, scoring='r2',cv=5) #
Change , Forward = True
      9
     10 sfs.fit(X_train, y_train)

TypeError: __init__() got an unexpected keyword argument 'k_features'
```

In [43]:
```
from mlxtend.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LinearRegression

lr = LinearRegression()

# Create an instance of SequentialFeatureSelector for Forward Selection
sfs = SequentialFeatureSelector(lr,
                                k_features=5,  # Adjust the desired number of features
                                forward=True,
                                floating=False,
                                scoring='r2',
                                cv=5)

# Perform Forward Selection
sfs.fit(X_train, y_train)
```

Out[43]:
```
SequentialFeatureSelector
SequentialFeatureSelector(estimator=LinearRegression(), k_features=(5, 5),
                          scoring='r2')
estimator: LinearRegression
LinearRegression()
LinearRegression
LinearRegression()
```

In [44]:
```
# Get the indices of the selected features in the Focal cell
sfs.k_feature_idx_
```

Out[44]: (5, 7, 10, 11, 12)

In [45]:

```
# Create a DataFrame from the metric dictionary
metric_df = pd.DataFrame.from_dict(sfs.get_metric_dict()).T

# Add additional columns to the DataFrame
metric_df['observations'] = 404

"""This code will assign the value 404 to every row in the 'observations' column of the 'metri


metric_df['num_features'] = metric_df['feature_idx'].apply(lambda x: len(x))

"""In this code, the apply function is applied to each element in the 'feature_idx' column usi
The lambda function calculates the length of each feature index list x.
The resulting lengths are then assigned to the 'num_features' column in the 'metric_df' DataFr


metric_df['adjusted_r2'] = adjust_r2(metric_df['avg_score'],
                                     metric_df['observations'],
                                     metric_df['num_features'])

metric_df
```

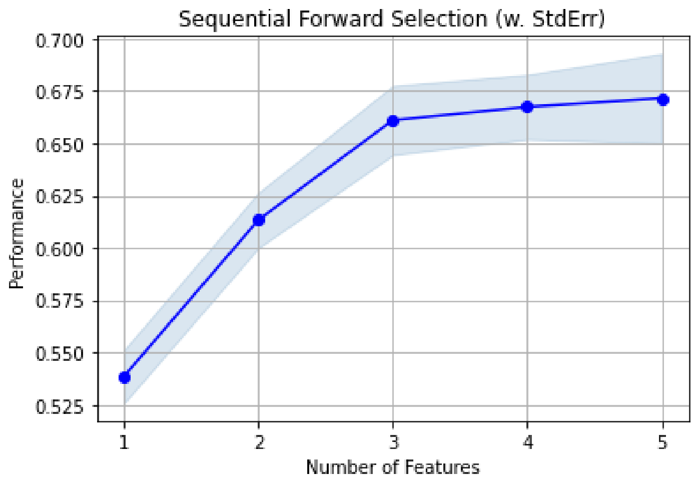
Out[45]:

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err	observations	num_f
1	(12,)	[0.5472998394577442, 0.49002001493399727, 0.53...	0.538451	(12,)	0.032755	0.025485	0.012742	404	
2	(5, 12)	[0.6330856272904801, 0.5779812120755249, 0.586...	0.613259	(5, 12)	0.034066	0.026505	0.013252	404	
3	(5, 10, 12)	[0.7215896884753016, 0.6288372046797153, 0.633...	0.661012	(5, 10, 12)	0.04259	0.033136	0.016568	404	
4	(5, 10, 11, 12)	[0.725877216548624, 0.6342604286872173, 0.6558...	0.667383	(5, 10, 11, 12)	0.039611	0.030819	0.01541	404	
5	(5, 7, 10, 11, 12)	[0.7440756174774326, 0.6473449858158778, 0.614...	0.671496	(5, 7, 10, 11, 12)	0.055057	0.042836	0.021418	404	

In [51]:
```
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs


fig1 = plot_sfs(sfs.get_metric_dict(), kind='std_err',)

plt.title('Sequential Forward Selection (w. StdErr)')
plt.grid()
plt.show()
```



In [47]:
```
# Transform the features of the training set
X_train_sel = sfs.transform(X_train)

# Transform the features of the test set
X_test_sel = sfs.transform(X_test)

# Create a linear regression model
model = LinearRegression()

# Print the mean R^2 score for training set
print("Training R^2:", np.mean(cross_val_score(model, X_train_sel, y_train, cv=5, scoring='r2

# Print the mean R^2 score for test set
print("Testing R^2:", np.mean(cross_val_score(model, X_test_sel, y_test, cv=5, scoring='r2')))
```

```
Training R^2: 0.6714957524062923
Testing R^2: 0.7505189069289498
```

In [48]:
```
# Display the shape of X_train_sel
X_train_sel.shape
```

Out[48]: (404, 5)

In [49]:
```
### Using With Sk-learn
```

In [50]:
```
from sklearn.feature_selection import SelectKBest, f_regression

# Assuming you have X as your feature matrix and y as your target variable

num_features = 5  # Adjust the desired number of features

selected_features = []
remaining_features = list(X.columns)

for _ in range(num_features):
    best_score = -float('inf')
    best_feature = None

    for feature in remaining_features:
        current_features = selected_features + [feature]
        X_subset = X[current_features]

        selector = SelectKBest(score_func=f_regression, k=1)
        selector.fit(X_subset, y)

        score = selector.scores_[0]

        if score > best_score:
            best_score = score
            best_feature = feature

    selected_features.append(best_feature)
    remaining_features.remove(best_feature)

print("Selected Features:", selected_features)
```

```
Selected Features: ['lstat', 'crim', 'zn', 'indus', 'chas']
```

In [ ]: