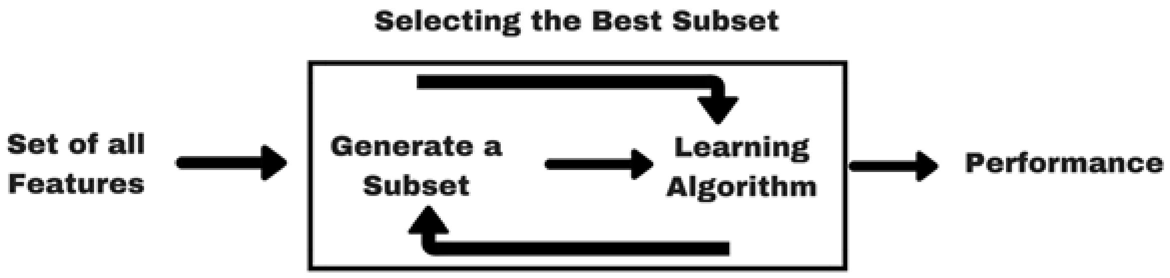
2 Wrapper method:



In wrapper methods, we try to use a subset of features and train a model using them. Based on the inferences that we draw from the previous model, we decide to add or remove features from your subset. The problem is essentially reduced to a search problem. These methods are usually computationally very expensive.

Some common examples of wrapper methods are

2.1. forward feature selection

2.2. backward feature elimination

2.3. recursive feature elimination

2.4. Boruta feature elimination

Forward Selection

Forward selection is an iterative method in which we start with having no feature in the model. In each iteration, we keep adding the feature which best improves our model till an addition of a new variable does not improve the performance of the model.

In [4]:

```
# include necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold
```

dataset link: https://www.kaggle.com/c/bnp-paribas-cardif-claims-management/data (https://www.kaggle.com/c/bnp-paribas-cardif-claims-management/data)

In [5]:

```
# read the dataset
paribas_data = pd.read_csv('paribas.csv',nrows = 20000)
```

In [6]:

```
paribas_data.shape
```

Out[6]:

```
(20000, 133)
```

In [7]:

```
# remove the correlated features with threshold greater than 0.8
```

In [8]:

```
num_cols = ['int16','int32','int64','float16','float32','float64']
```

In [10]:

```
numerical_cols = list(paribas_data.select_dtypes(include = num_cols).columns)
paribas_data = paribas_data[numerical_cols]
```

In [12]:

```
train_features , test_features , train_labels , test_labels = train_test_split(
    paribas_data.drop(labels=['target','ID'],axis =1),
paribas_data['target'],
test_size = 0.2,
random_state = 41)
```

In [13]:

```
correlated_features = set()
def corr_matrix(dataset):
    return dataset.corr()
```

In [14]:

```
correlation_matrix = corr_matrix(paribas_data)
```

In [15]:

```
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i,j]) > 0.8:
            col_name = correlation_matrix.columns[i]
            correlated_features.add(col_name)
```

In [17]:

```
len(correlated_features)
```

Out[17]:

```
55
```

In [18]:

```
train_features.drop(labels = correlated_features ,axis = 1, inplace = True)
test_features.drop(labels = correlated_features , axis = 1, inplace = True)
```

In [57]:

```
# Implementing Random ForestCLassifer on the entire features

from sklearn.ensemble import RandomForestRegressor,RandomForestClassifier
from sklearn.metrics import roc_auc_score
clf_before = RandomForestClassifier(n_estimators = 100 , random_state = 41 ,
                             max_depth = 3
                            )
clf_before.fit(train_features.fillna(0),train_labels )

train_predict = clf_before.predict_proba(train_features.fillna(0))

print('Accuracy on training_data: {}'.format(roc_auc_score(train_labels ,train_predict[

test_predict = clf_before.predict_proba(test_features.fillna(0))
print('accuracy on test_set:{}'.format(roc_auc_score(test_labels,test_predict[:,1])))
```

```
Accuracy on training_data: 0.7099047641193875
accuracy on test_set:0.7096477998274374
```

Implementing Step Forward Feature Selection in Python

To select the most optimal features, we will be using SequentialFeatureSelector function from the mlxtend library.

We will use the Random Forest Classifier to find the most optimal parameters. The evaluation criteria used will be ROC-AUC

In [24]:

```
from sklearn.ensemble import RandomForestRegressor,RandomForestClassifier
from sklearn.metrics import roc_auc_score
from mlxtend.feature_selection import SequentialFeatureSelector
```

In [25]:

```
feature_selector = SequentialFeatureSelector(RandomForestClassifier(n_jobs = 1),
                                            k_features = 15,
                                            forward = True,
                                            verbose = 2,
                                            scoring = 'roc_auc',
                                            cv = 4)
```

In the script above we pass the RandomForestClassifieras the estimator to the SequentialFeatureSelector function. The k_features specifies the number of features to select. You can set any number of features here. The forward parameter, if set to True, performs step forward feature selection. The verbose parameter is used for logging the progress of the feature selector, the scoring parameter defines the performance evaluation criteria and finally, cv refers to cross-validation folds.

In [27]:

```
features = feature_selector.fit(train_features.fillna(0),train_labels)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
workers.
c:\users\dell\appdata\local\programs\python\python36\lib\site-packages
\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_
estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
c:\users\dell\appdata\local\programs\python\python36\lib\site-packages
\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_
estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
c:\users\dell\appdata\local\programs\python\python36\lib\site-packages
\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_
estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
c:\users\dell\appdata\local\programs\python\python36\lib\site-packages
\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_
estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.0s remaining:
```

In [33]:

```
type(features.k_feature_idx_)
```

Out[33]:

```
tuple
```

In [39]:

```
(list(features.k_feature_idx_))
# return the indices of the features selcted
```

Out[39]:

```
[2, 8, 11, 13, 15, 18, 20, 21, 23, 24, 27, 31, 40, 47, 55]
```

In [38]:

```
filtered_features = train_features.columns[list(features.k_feature_idx_)]
```

In [40]:

```
len(filtered_features)
```

Out[40]:

```
15
```

In [43]:

```
features.k_score_
```

Out[43]:

```
0.6560006400208643
```

In [53]:

```
clf = RandomForestClassifier(n_estimators = 100 , random_state = 41 ,
                             max_depth = 3
                            )
clf.fit(train_features[filtered_features].fillna(0),train_labels )

train_pred = clf.predict_proba(train_features[filtered_features].fillna(0))

print('Accuracy on training_data: {}'.format(roc_auc_score(train_labels ,train_pred[:,1

test_pred = clf.predict_proba(test_features[filtered_features].fillna(0))
print('accuracy on test_set:{}'.format(roc_auc_score(test_labels,test_pred[:,1])))
```

```
Accuracy on training_data: 0.7076619404708223
accuracy on test_set:0.7091114754098361
```

NOTE: From the outcome accuracy of cell 53(on 15 features) and 57(on entire features) we can sese that the accuracy is maintained on same scale

#### links for the methods() used

##### *SequentialFeatureSelector(): https://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/#sequential-feature-selector*

##### *RandomForestClassifier(): https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html*