2.4 Boruta:

One of the best ways for implementing feature selection with wrapper methods is to use Boruta package that finds the importance of a feature by creating shadow features.

In contrary to the previous algorithms, Boruta tries to find all relevant features useful for prediction, instead of defining a subset of features with minimal error.

It works in the following steps:

1. Firstly, it adds randomness to the given data set by creating shuffled copies of all features (which are called shadow features).
2. Then, it trains a random forest classifier on the extended data set and applies a feature importance measure (the default is Mean Decrease Accuracy) to evaluate the importance of each feature where higher means more important.
3. At every iteration, it checks whether a real feature has a higher importance than the best of its shadow features (i.e. whether the feature has a higher Z-score than the maximum Z-score of its shadow features) and constantly removes features which are deemed highly unimportant.
4. Finally, the algorithm stops either when all features get confirmed or rejected or it reaches a specified limit of random forest runs.

In [2]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

In [3]:

```
# Load dataset
data = pd.read_csv('paribas.csv',nrows =20000)
```

In [5]:

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

numerical_columns = list(data.select_dtypes(include = num_cols).columns)
data = data[numerical_columns]
```

In [6]:

```
#we can seet.add() method to add column names
correlated_features = set()
correlation_matrix = data.corr()
```

In [7]:

```
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 [8]:

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

In [9]:

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

Using Boruta

link for documentation on BorutaPy:

https://pypi.org/project/Boruta/ (https://pypi.org/project/Boruta/)

In [10]:

```
# Installation
#!pip install boruta
```

In [12]:

```
from boruta import BorutaPy
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier as rfc
```

In [15]:

```
boruta_selector = BorutaPy(rfc(n_jobs = 1),
                 n_estimators = 'auto',
                 verbose = 2)
```

In [17]:

```
features = boruta_selector.fit(np.array(train_features.fillna(0)) , train_labels)
```

```
Iteration: 	1 / 100
Confirmed: 	0
Tentative: 	57
Rejected: 	0
Iteration: 	2 / 100
Confirmed: 	0
Tentative: 	57
Rejected: 	0
Iteration: 	3 / 100
Confirmed: 	0
Tentative: 	57
Rejected: 	0
Iteration: 	4 / 100
Confirmed: 	0
Tentative: 	57
Rejected: 	0
Iteration: 	5 / 100
Confirmed: 	0
Tentative: 	57
Rejected: 	0
Iteration: 	6 / 100
Confirmed: 	0
Tentative: 	57
Rejected: 	0
Iteration: 	7 / 100
Confirmed: 	0
Tentative: 	57
Rejected: 	0
Iteration: 	8 / 100
Confirmed: 	0
Tentative: 	4
Rejected: 	53
Iteration: 	9 / 100
Confirmed: 	4
Tentative: 	0
Rejected: 	53


BorutaPy finished running.

Iteration: 	10 / 100
Confirmed: 	4
Tentative: 	0
Rejected: 	53
```

In [38]:

```
features_selected = train_features.columns[features.support_]
train_features[features_selected].shape
```

Out[38]:

```
(16000, 4)
```

In [35]:

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

train_pred = clf.predict_proba(train_features[features_selected].fillna(0))
print('accuracy on training data: {}'.format(roc_auc_score(train_labels , train_pred[:,

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

```
accuracy on training data: 0.7035399696485769
accuracy on test data  :0.7112540120793788
```

conclusion :

In Both forward and backward feature_Selection we have to manually give no:of features to be selected by our algorithm ,where as in RFECV and Boruta it is automatic ¶

Both forward and backward feature selection are time consuming methods and computationally very expensive methods , where as RFECV and Boruta methods are very fast

Among all the wrapper methods Boruta method is the best method since it is automatic and computationally less expensive

In the above section 2.1,2.2,2.3,2.4 we use same dataset,and the no: features selcted are as follows:

1. forward :15
2. backward :15
3. RFECV : 13
4. Boruta :4

on observing the no:features we can say , boruta is the best method since the accuracy achived by all the methods are same