Feature Selection

What's the Purpose of Feature Selection

Many learning algorithms perform poorly on high-dimensional data. This is known as the **curse of dimensionality**

There are other reasons we may wish to reduce the number of features including:

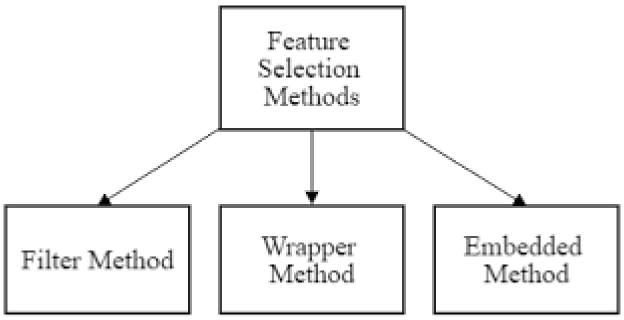
1. Reducing computational cost

2. Reducing the cost associated with data collection

3. Improving Interpretability

Reference for entire topic-

https://www.youtube.com/watch?v=EqLBAmtKMnQ (https://www.youtube.com/watch?v=EqLBAmtKMnQ)



In [181]:

```
#import these libraries as we are going to use them.
# Note: just have a look what all libraries you have imported
import pandas as pd
import numpy as np

from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_predict
from sklearn.linear_model import LinearRegression
from math import sqrt
```

1.Filter Methods:

Filter method applies a statistical measure to assign a scoring to each feature.Then we can decide to keep or remove those features based on those scores. The methods are often univariate and consider the feature independently, or with regard to the dependent variable.

In this section we will cover below approaches:

1. Missing Value Ratio Threshold
2. Variance Threshold
3. Chi^2 Test
4. Anova Test

Download the dataset from https://github.com/arupbhunia/Data-Pre-processing/blob/master/datasets/diabetes.csv (https://github.com/arupbhunia/Data-Pre-processing/blob/master/datasets/diabetes.csv)

Dataset used - diabetes.csv

Download instruction: go to the given link--->click raw button on top right corner---->Press Ctrl+S -->save it as .csv file.

(a) Missing Value Ratio Threshold

20 points

In [183]:

```
# create a data frame named diabetes and load the csv file

#print the head
```

Out[183]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

We know that some features can not be zero(e.g. a person's blood pressure can not be 0) hence we will impute zeros with nan value in these features.

Reference to impute: https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.replace.html (https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.replace.html)

In [184]:

```
#Glucose BloodPressure, SkinThickness, Insulin, and BMI features cannot be zero ,we will im
```

In [185]:

```
#display the no of null values in each feature
```

Out[185]:

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

Now let's see for each feature what is the percentage of having missing values.

In [186]:

```
#percentage of missing values for Glucose
```

Out[186]:

```
0.6510416666666667
```

In [187]:

```
# calculate the percentage for Bloodpressure
```

Out[187]:

```
4.557291666666666
```

In [188]:

```
# calculate the percentage for SkinThickness
```

Out[188]:

```
29.557291666666668
```

In [189]:

```
# calculate the percentage for Insulin
```

Out[189]:

```
48.69791666666667
```

In [190]:

```
# calculate the percentage for BMI
```

Out[190]:

```
1.4322916666666665
```

Hey can you see that a large number of data missing in SkinThickness and Insulin.

Here we will keep only those features which are having missing data less than 10% as our threshold.

Reference to check methods for dropping nan in pandas- https://www.youtube.com/watch?v=57vFbsiZYHg (https://www.youtube.com/watch?v=57vFbsiZYHg)

You can also check its document official: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html)

In [191]:

```
#we are keep only those features which are having missing data less than 10%
diabetes_missing_value_threshold=#code

# print diabetes_missing_value_threshold
```

Out[191]:

	Pregnancies	Glucose	BloodPressure	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	33.6	0.627	50	1
1	1	85.0	66.0	26.6	0.351	31	0
2	8	183.0	64.0	23.3	0.672	32	1
3	1	89.0	66.0	28.1	0.167	21	0
4	0	137.0	40.0	43.1	2.288	33	1
...	...	...	...	...	...	...	...
763	10	101.0	76.0	32.9	0.171	63	0
764	2	122.0	70.0	36.8	0.340	27	0
765	5	121.0	72.0	26.2	0.245	30	0
766	1	126.0	60.0	30.1	0.349	47	1
767	1	93.0	70.0	30.4	0.315	23	0

768 rows × 7 columns

Let's now Seperate the data diabetes_missing_value_threshold into features and labels

Hey buddy! label is something which is dependent on other features for its outcome. You can also called it as our Target variable which we predict using ML algorithms.

Can you think which column would be considered as label.

In [192]:

```
diabetes_missing_value_threshold_features = #code

diabetes_missing_value_threshold_label= #code
```

In [194]:

```
#print diabetes_missing_value_threshold_features
```

Out[194]:

	Pregnancies	Glucose	BloodPressure	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	33.6	0.627	50
1	1	85.0	66.0	26.6	0.351	31
2	8	183.0	64.0	23.3	0.672	32
3	1	89.0	66.0	28.1	0.167	21
4	0	137.0	40.0	43.1	2.288	33
...	...	...	...	...	...	...
763	10	101.0	76.0	32.9	0.171	63
764	2	122.0	70.0	36.8	0.340	27
765	5	121.0	72.0	26.2	0.245	30
766	1	126.0	60.0	30.1	0.349	47
767	1	93.0	70.0	30.4	0.315	23

768 rows × 6 columns

In [195]:

```
#print diabetes_missing_value_threshold_label
```

Out[195]:

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

(b) Variance Threshold

20 points

If the variance is low or close to zero, then a feature is approximately constant and will not improve the performance of the model. In that case, it should be removed.

Variance will also be very low for a feature if only a handful of observations of that feature differ from a constant value.

Reference- https://www.youtube.com/watch?v=uMlU2JaiOd8 (https://www.youtube.com/watch?v=uMlU2JaiOd8)

Are you ready to implement feature selection using variance threshold? Smile and download the dataset diabetes_cleaned.csv from https://github.com/arupbhunia/Data-Pre-processing/blob/master/datasets/diabetes_cleaned.csv (https://github.com/arupbhunia/Data-Pre-processing/blob/master/datasets/diabetes_cleaned.csv)

Dataset used - diabetes_cleaned.csv

In [197]:

```
# load the csv to dataframe name "diabetes" and print the head values


# display diabetes.head()
```

Out[197]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFun
0	6.0	148.0	72.0	35.0	218.937760	33.6	
1	1.0	85.0	66.0	29.0	70.189298	26.6	
2	8.0	183.0	64.0	29.0	269.968908	23.3	
3	1.0	89.0	66.0	23.0	94.000000	28.1	
4	0.0	137.0	40.0	35.0	168.000000	43.1	

In [198]:

```
# seperate the features and the target as x and y
```

If you have seen the video then Krish must have told you to use sklearn library to calculate variance threshold. But we will use var function to calculate our variance so that you understand the concept from base.

In [199]:

```
# Return  the variance for X along the specified axis=0.
```

Out[199]:

```
Pregnancies                   11.354056
Glucose                      929.680350
BloodPressure                146.321591
SkinThickness                 77.285567
Insulin                     9484.259268
BMI                           48.813618
DiabetesPedigreeFunction       0.109779
Age                          138.303046
dtype: float64
```

```
    Hey smarty! did you see that DiabetesPedigreeFunction variance is less so it bring
    s almost no information because it is (almost) constant , this can be a justificat
    ion to remove DiabetesPedigreeFunction column but before considering this we shoul
    d scale these features because they are of different scales.
```

Reference for minmax scaling: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html)

Lets use sklearn minmax scaler here.

In [1]:

```
# import minmax_scale

# use minmax scale with feature_range=(0,10) and columns=X.columns,to scale the features of
```

Wait a minute! whats minmax scaling?

It is the simplest method and consists in rescaling the range of features to scale the range in [0, 1] or [−1, 1]

hey hey heyieeee! Fun fact time:

There is another scaling method called StandardScaler which follows Standard Normal Distribution (SND). Therefore, it makes mean = 0 and scales the data to unit variance.

Cool right? :)

In [202]:

```
# return X_scaled_df
```

Out[202]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
0	3.529412	6.709677	4.897959	3.043478	2.740295	3.149284	
1	0.588235	2.645161	4.285714	2.391304	1.018185	1.717791	
2	4.705882	8.967742	4.081633	2.391304	3.331099	1.042945	
3	0.588235	2.903226	4.285714	1.739130	1.293850	2.024540	
4	0.000000	6.000000	1.632653	3.043478	2.150572	5.092025	
...	...	...	...	...	...	...	
763	5.882353	3.677419	5.306122	4.456522	2.289500	3.006135	
764	1.176471	5.032258	4.693878	2.173913	2.044244	3.803681	
765	2.941176	4.967742	4.897959	1.739130	1.502241	1.635992	
766	0.588235	5.290323	3.673469	2.391304	2.217956	2.433538	
767	0.588235	3.161290	4.693878	2.608696	1.215086	2.494888	

768 rows × 8 columns

In [203]:

```
# Again return  the variance for X along the specified axis=0 to check the scales after usi
```

Out[203]:

```
Pregnancies                 3.928739
Glucose                     3.869637
BloodPressure               1.523548
SkinThickness               0.913109
Insulin                     1.271218
BMI                         2.041377
DiabetesPedigreeFunction    2.001447
Age                         3.841751
dtype: float64
```

Now again you can check the previous video:https://www.youtube.com/watch?v=uMlU2JaiOd8 (https://www.youtube.com/watch?v=uMlU2JaiOd8)

In [205]:

```
# import variancethreshold

# set threshold=1 and define it to variable select_features
```

Impliment fit_transform on select_features passing X_scaled_df into it and save this result in variable X_variance_threshold_df

In [206]:

```
X_variance_threshold_df=#code
```

Were you thinking of fit_transform? We are here to help you understand

fit_transform() is used on the training data so that we can scale the training data and also learn the scaling parameters of that data. Here, the model built by us will learn the mean and variance of the features of the training set. These learned parameters are then used to scale our test data

Don't worry you will get lot of challenges to use these things in our other assignments

In [207]:

```
#print X_variance_threshold_df
```

Out[207]:

```
array([[3.52941176, 6.70967742, 4.89795918, ..., 3.14928425, 2.3441503 ,
        4.83333333],
       [0.58823529, 2.64516129, 4.28571429, ..., 1.71779141, 1.16567037,
        1.66666667],
       [4.70588235, 8.96774194, 4.08163265, ..., 1.04294479, 2.53629377,
        1.83333333],
       ...,
       [2.94117647, 4.96774194, 4.89795918, ..., 1.63599182, 0.71306576,
        1.5       ],
       [0.58823529, 5.29032258, 3.67346939, ..., 2.43353783, 1.15713066,
        4.33333333],
       [0.58823529, 3.16129032, 4.69387755, ..., 2.49488753, 1.01195559,
        0.33333333]])
```

In [208]:

```
#Convert X_variance_threshold_df into dataframe
```

In [209]:

```
# print of head values of X_variance_threshold_df
```

Out[209]:

	0	1	2	3	4	5	6
0	3.529412	6.709677	4.897959	2.740295	3.149284	2.344150	4.833333
1	0.588235	2.645161	4.285714	1.018185	1.717791	1.165670	1.666667
2	4.705882	8.967742	4.081633	3.331099	1.042945	2.536294	1.833333
3	0.588235	2.903226	4.285714	1.293850	2.024540	0.380017	0.000000
4	0.000000	6.000000	1.632653	2.150572	5.092025	9.436379	2.000000

Below mentioned is the function get_selected_features for returning selected_features to be used further.

Warning ;) If we have provided you a readymade function, don't just use it but understand it too.

In [210]:

```
def get_selected_features(raw_df,processed_df):
    selected_features=[]
    for i in range(len(processed_df.columns)):
        for j in range(len(raw_df.columns)):
            if (processed_df.iloc[:,i].equals(raw_df.iloc[:,j])):
                selected_features.append(raw_df.columns[j])
    return selected_features
```

In [211]:

```
# pass the X_scaled_df as raw_df and X_variance_threshold_df as processed_df inside get_sel
selected_features= # code

# print selected_features
```

Out[211]:

```
['Pregnancies',
 'Glucose',
 'BloodPressure',
 'Insulin',
 'BMI',
 'DiabetesPedigreeFunction',
 'Age']
```

Super! you can see SkinThickness feature is not selected as its variance is less.

Lets give column names to our X_variance_threshold_df

In [212]:

```
# define selected_features as columns and save it in variabe named X_variance_threshold_df

#print X_variance_threshold_df
```

Out[212]:

	Pregnancies	Glucose	BloodPressure	Insulin	BMI	DiabetesPedigreeFunction	
0	3.529412	6.709677	4.897959	2.740295	3.149284	2.344150	4.83
1	0.588235	2.645161	4.285714	1.018185	1.717791	1.165670	1.66
2	4.705882	8.967742	4.081633	3.331099	1.042945	2.536294	1.83
3	0.588235	2.903226	4.285714	1.293850	2.024540	0.380017	0.00
4	0.000000	6.000000	1.632653	2.150572	5.092025	9.436379	2.00
...	...	...	...	...	...	...	
763	5.882353	3.677419	5.306122	2.289500	3.006135	0.397096	7.00
764	1.176471	5.032258	4.693878	2.044244	3.803681	1.118702	1.00
765	2.941176	4.967742	4.897959	1.502241	1.635992	0.713066	1.50
766	0.588235	5.290323	3.673469	2.217956	2.433538	1.157131	4.33
767	0.588235	3.161290	4.693878	1.215086	2.494888	1.011956	0.33

768 rows × 7 columns

(c) Chi-Squared statistical test (SelectKBest)

20 points

Chi2 is a measure of dependency between two variables. It gives us a goodness of fit measure because it measures how well an observed distribution of a particular feature fits with the distribution that is expected if two features are independent.

Scikit-Learn offers a feature selection estimator named SelectKBest which select K numbers of features based on the statistical analysis.

Reference link: https://chrisalbon.com/machine_learning/feature_selection/chi-squared_for_feature_selection/ (https://chrisalbon.com/machine_learning/feature_selection/chi-squared_for_feature_selection/)

The below mentioned function generate_feature_scores_df is used to get feature score for using it in Chi-Squared statistical test explained below

In [213]:

```
def generate_feature_scores_df(X,Score):
    feature_score=pd.DataFrame()
    for i in range(X.shape[1]):
        new =pd.DataFrame({"Features":X.columns[i],"Score":Score[i]},index=[i])
        feature_score=pd.concat([feature_score,new])
    return feature_score
```

Hey coder! lets use dataset - diabetes.csv from below link

https://github.com/arupbhunia/Data-Pre-processing/blob/master/datasets/diabetes.csv (https://github.com/arupbhunia/Data-Pre-processing/blob/master/datasets/diabetes.csv)

In [214]:

```
# create a data frame named diabetes and load the csv file again
```

In [215]:

```
# assign features to X variable and 'outcome' to y variable from the dataframe diabetes
```

In [216]:

```
#import chi2 and SelectKBest
```

In [217]:

```
# converting data cast to a float type.
```

Lets use SelectKBest to calculate the best feature score. Use Chi2 as Score Function and no.of feature i.e. k as 4

In [218]:

```
# Initialise SelectKBest with above parameters
chi2_test=#

# fit it with X and Y
chi2_model=#
```

In [219]:

```
#print the scores of chi2_model
```

Out[219]:

```
array([ 111.51969064, 1411.88704064,   17.60537322,   53.10803984,
       2175.56527292,  127.66934333,    5.39268155,  181.30368904])
```

In [220]:

```
# use generate_feature_scores_df function to get features and their respective scores passi
feature_score_df=#


# return feature_score_df
```

Out[220]:

	Features	Score
0	Pregnancies	111.519691
1	Glucose	1411.887041
2	BloodPressure	17.605373
3	SkinThickness	53.108040
4	Insulin	2175.565273
5	BMI	127.669343
6	DiabetesPedigreeFunction	5.392682
7	Age	181.303689

Did you see the features and corresponding chi square scores? This is so easy right, higher the score better the feature. Just like higher the marks in assignment better the student of ours.

In [222]:

```
#Lets get X with selected features of chi2_model using tranform function so we will have X_
X_new=#
```

whaaaat! tranform()? , how is it different from fit_transform.You know buddy fit() can also confuse you. Hey you inquisitive learner we will tell you the difference.

The fit() function calculates the values of these parameters. The transform function applies the values of the parameters on the actual data and gives the normalized value. The fit_transform() function performs both in the same step. Note that the same value is got whether we perform in 2 steps or in a single step.

for more info on this refer: https://www.youtube.com/watch?v=BotYLBQfd5M (https://www.youtube.com/watch?v=BotYLBQfd5M)

In [223]:

```
# Convert X_new into a dataframe

X_new=#
```

In [224]:

```
#repeat the previous steps of calling get_selected_features function( pass X and X_new as s
selected_features=#

# return selected_features
```

Out[224]:

```
['Glucose', 'Insulin', 'BMI', 'Age']
```

Let have X with all features given in list selected_features and save this dataframe in variable chi2_best_features

In [225]:

```


# print chi2_best_features.head()
```

Out[225]:

	Glucose	Insulin	BMI	Age
0	148.0	0.0	33.6	50.0
1	85.0	0.0	26.6	31.0
2	183.0	0.0	23.3	32.0
3	89.0	94.0	28.1	21.0
4	137.0	168.0	43.1	33.0

As you can see chi-squared test helps us to select important independent features out of the original features that have the strongest relationship with the target feature.

You did it well!

(d) Anova-F Test

20 points

The F-value scores examine the varaiance by grouping the numerical feature by the target vector, the means for each group are significantly different.

References:

1. https://www.youtube.com/watch?v=9zrQ_c5RZkI (https://www.youtube.com/watch?v=9zrQ_c5RZkI)

In [226]:

```
#import libraries
from sklearn.feature_selection import f_classif,SelectPercentile


# Initialise SelectPercentile function with parameters f_classif and percentile as 80
Anova_test=#



#Fit the above object to the features and target i.e X and Y
Anova_model=#
```

here you have used f_classif for Anova-F test. To know more about this test you can check this artical.

https://towardsdatascience.com/anova-for-feature-selection-in-machine-learning-d9305e228476 (https://towardsdatascience.com/anova-for-feature-selection-in-machine-learning-d9305e228476)

In [227]:

```
# return scores of anova model
```

Out[227]:

```
array([ 39.67022739, 213.16175218,   3.2569504 ,   4.30438091,
        13.28110753,  71.7720721 ,  23.8713002 ,  46.14061124])
```

In [228]:

```
# use generate_feature_scores_df function to get features and their respective scores by pa

feature_scores_df=#code

# print feature_scores_df
```

Out[228]:

	Features	Score
0	Pregnancies	39.670227
1	Glucose	213.161752
2	BloodPressure	3.256950
3	SkinThickness	4.304381
4	Insulin	13.281108
5	BMI	71.772072
6	DiabetesPedigreeFunction	23.871300
7	Age	46.140611

In [229]:

```
# Get all supported columns values in Anova_model with indices=True
cols = #
# Reduce X to the selected features of anova model using tranform

X_new = #
```

In [232]:

```
#print X_new.head()
```

Out[232]:

	Pregnancies	Glucose	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6.0	148.0	0.0	33.6	0.627	50.0
1	1.0	85.0	0.0	26.6	0.351	31.0
2	8.0	183.0	0.0	23.3	0.672	32.0
3	1.0	89.0	94.0	28.1	0.167	21.0
4	0.0	137.0	168.0	43.1	2.288	33.0

Hey brighty! Hope you learned to implement Anova F-test method for feature selection. It has selected 6 best features as you can see in above output

2. Wrapper Methods

30 points

Wrapper methods are used to select a set of features by preparing where different combinations of features, then each combination is evaluated and compared to other combinations.Next a predictive model is used to assign a score based on model accuracy and to evaluate the combinations of these features.

In [234]:

```
# load and read the diabetes.csv using pandas and print the head values
```

Out[234]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

In [235]:

```
# assign features to X and target 'outcome' to Y variable(Think why the 'outcome' column is
X=#
Y=#

#return X,Y
```

Out[235]:

```
(     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
 0              6      148             72             35        0  33.6
 1              1       85             66             29        0  26.6
 2              8      183             64              0        0  23.3
 3              1       89             66             23       94  28.1
 4              0      137             40             35      168  43.1
 ..           ...      ...            ...            ...      ...   ...
 763           10      101             76             48      180  32.9
 764            2      122             70             27        0  36.8
 765            5      121             72             23      112  26.2
 766            1      126             60              0        0  30.1
 767            1       93             70             31        0  30.4

      DiabetesPedigreeFunction  Age
 0                       0.627   50
 1                       0.351   31
 2                       0.672   32
 3                       0.167   21
 4                       2.288   33
 ..                        ...  ...
 763                     0.171   63
 764                     0.340   27
 765                     0.245   30
 766                     0.349   47
 767                     0.315   23

 [768 rows x 8 columns],
 0      1
 1      0
 2      1
 3      0
 4      1
       ..
 763    0
 764    0
 765    0
 766    1
 767    0
 Name: Outcome, Length: 768, dtype: int64)
```

(a) Recursive Feature Elemination

25 points

Recursive Feature Elimination selects features by recursively considering smaller subsets of features by pruning the least important feature at each step. Here models are created iteartively and in each iteration it determines the best and worst performing features and this process continues until all the features are explored.Next ranking is given on eah feature based on their elimination orde. In the worst case, if a dataset contains N number of features RFE will do a greedy search for N^2 combinations of features.

reference video: [https://www.youtube.com/watch?v=MYnxxRoPiwl (https://www.youtube.com/watch?v=MYnxxRoPiwl)](https://www.youtube.com/watch?v=MYnxxRoPiwl)

Note: the video is using random forest classifier, but we are going to use logistic regression as our model.

In [236]:

```
# import RFE and LogisticRegression
```

In [242]:

```
# Initialise model variable with LogisticRegression function with solver = 'liblinear'
model=#

# rfe variable has RFE instance with should have model and n_features_to_select=4 as parame

rfe=#
```

In [243]:

```
# fit rfe with X and Y

fit=#
```

In [239]:

```
print('Number of selected features',fit.n_features_)
print('Selected Features',fit.support_)
print('Feature rankings',fit.ranking_)
```

```
Number of selected features 4
Selected Features [ True  True False False False  True  True False]
Feature rankings [1 1 2 4 5 1 1 3]
```

In [241]:

```
# use below function to get ranks of all the features
def feature_ranks(X,Rank,Support):
    feature_rank=pd.DataFrame()
    for i in range(X.shape[1]):
        new =pd.DataFrame({"Features":X.columns[i],"Rank":Rank[i],'Selected':Support[i]},in
        feature_rank=pd.concat([feature_rank,new])
    return feature_rank
```

In [244]:

```
#Get all feature's ranks using feature_ranks function with suitable parameters. Sotre it in
feature_rank_df=#

# print feature_rank_df
```

Out[244]:

	Features	Rank	Selected
0	Pregnancies	1	True
1	Glucose	1	True
2	BloodPressure	2	False
3	SkinThickness	4	False
4	Insulin	5	False
5	BMI	1	True
6	DiabetesPedigreeFunction	1	True
7	Age	3	False

We can see there are four features with rank 1 ,RFE states that these are the most significant features.

In [247]:

```
# filter feature_rank_df  with selected column values as True and save result in variable c
recursive_feature_names=#

# print recursive_feature_names
```

Out[247]:

	Features	Rank	Selected
0	Pregnancies	1	True
1	Glucose	1	True
5	BMI	1	True
6	DiabetesPedigreeFunction	1	True

In [248]:

```
# finally get dataframe X with all the features selected by RFE and store this result in va
RFE_selected_features=#

# print RFE head()
```

Out[248]:

	Pregnancies	Glucose	BMI	DiabetesPedigreeFunction
0	6	148	33.6	0.627
1	1	85	26.6	0.351
2	8	183	23.3	0.672
3	1	89	28.1	0.167
4	0	137	43.1	2.288

We recommend you to watch this video to know about working of RFE: https://www.youtube.com/watch?v=Yo1vYRdJ95k (https://www.youtube.com/watch?v=Yo1vYRdJ95k)

3. Embedded Method using random forest

25 points

Reference: https://www.youtube.com/watch?v=em4OFr-4C34 (https://www.youtube.com/watch?v=em4OFr-4C34)

Feature selection using Random forest comes under the category of Embedded methods. Embedded methods combine the qualities of filter and wrapper methods. They are implemented by algorithms that have their own built-in feature selection methods. Some of the benefits of embedded methods are :

1. They are highly accurate.
2. They generalize better.
3. They are interpretable

In [249]:

```
#Importing libraries RandomForestClassifier and SelectFromModel
```

In [250]:

```
# load the csv file using pandas and print the head values

# print diabetes.head()
```

Out[250]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

In all feature selection procedures, it is a good practice to select the features by examining only the training set. This is to avoid overfitting. So considering we have a train and a test dataset. We select the features from the train set and then transfer the changes to the test set later

In [251]:

```
# assign features to X and target 'outcome' to Y(Think why the 'outcome' column is taken as
```

In [ ]:

```
# import test_train_split module

# splitting of dataset(test_size=0.3)
```

Here We will do the model fitting and feature selection altogether in one line of code.

Firstly, specify the random forest instance, indicating the number of trees.

Then use selectFromModel object from sklearn to automatically select the features. Simple right?. Don't worry trust your code. It helps.

Reference link to use selectFromModel: [https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html)](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html)

In [253]:

```
#create an instance of Select from Model. Pass an object of Random Forest Classifier with n
sel = #code


# fit sel on X and y

```

Out[253]:

```
SelectFromModel(estimator=RandomForestClassifier())
```

SelectFromModel will select those features which importance is greater than the mean importance of all the features by default, but we can alter this threshold if we want.

To see which features are important we can use get_support method on the fitted model.

In [254]:

```
# Using sel.get_support() print the boolean values for the features selected.
```

Out[254]:

```
array([False,  True, False, False, False,  True, False,  True])
```

In [255]:

```
#make a list named selected_feat with all columns which are True
selected_feat= #code

# print length of selected_feat
```

Out[255]:

```
3
```

In [256]:

```
# Print selected_feat
```

```
Index(['Glucose', 'BMI', 'Age'], dtype='object')
```

Well done Champ!. Let us impliment SelectFromModel using LinearSVC model also

Feature selection using SelectFromModel

25 points

SelectFromModel is a meta-transformer that can be used along with any estimator that has a coef_ or featureimportances attribute after fitting. The features are considered unimportant and removed, if the corresponding coef_ or featureimportances values are below the provided threshold parameter. Apart from specifying the threshold numerically, there are built-in heuristics for finding a threshold using a string argument. Available heuristics are “mean”, “median” and float multiples of these like “0.1*mean”.

Lets use selectfrommodel again with LinearSVC

In [259]:

```
# import LinearSVC
```

In [260]:

```
#Use SelectFromModel with LinearSVC() as its parameter and save it in variable 'm'

m = #code

#fit m with X and Y
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages/sklearn/sv
m/_base.py:986: ConvergenceWarning: Liblinear failed to converge, increase t
he number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[260]:

```
SelectFromModel(estimator=LinearSVC())
```

In [262]:

```
#make a list named selected_feat with all columns which are supported

selected_feat=#

print(selected_feat)
```

```
Index(['Pregnancies', 'DiabetesPedigreeFunction'], dtype='object')
```

4. Handling Multicollinearity with VIF

15 points

Multicollinearity refers to a situation in which more than two explanatory variables in a multiple regression model are highly linearly related. We have perfect multicollinearity if, for example as in the equation above, the correlation between two independent variables is equal to 1 or −1.

Variance inflation factor measures how much the behavior (variance) of an independent variable is influenced, or inflated, by its interaction/correlation with the other independent variables.

VIF has big defination but for now understand that:- Variance inflation factor (VIF) is a measure of the amount of multicollinearity in a set of multiple regression variables

Reference: https://www.youtube.com/watch?v=6JpmgzCAusI (https://www.youtube.com/watch?v=6JpmgzCAusI)

In [263]:

```
#load and read the diabetes_cleaned.csv file using pandas and print the head values

dia_df=
```

Out[263]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFun
0	6.0	148.0	72.0	35.0	218.937760	33.6	
1	1.0	85.0	66.0	29.0	70.189298	26.6	
2	8.0	183.0	64.0	29.0	269.968908	23.3	
3	1.0	89.0	66.0	23.0	94.000000	28.1	
4	0.0	137.0	40.0	35.0	168.000000	43.1	

In [264]:

```
# describe the dataframe using .describe()
```

Out[264]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabet
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	121.539062	72.405184	29.108073	152.222767	32.307682	
std	3.369578	30.490660	12.096346	8.791221	97.387162	6.986674	
min	0.000000	44.000000	24.000000	7.000000	-17.757186	18.200000	
25%	1.000000	99.000000	64.000000	25.000000	89.647494	27.300000	
50%	3.000000	117.000000	72.202592	29.000000	130.000000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	188.448695	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

As we can see range of these features are very different that means they all are in different scales so lets standardize the features using sklearn's preprocessing scale function.

Reference doc: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.scale.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.scale.html)

In [266]:

```
from sklearn import preprocessing

#iterate over all features in dia_df and scale
```

In [267]:

```
# describe dataframe using .describe()
```

Out[267]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	B
count	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+
mean	5.898060e-17	-7.372575e-18	1.163710e-17	-2.822540e-17	-4.192248e-18	2.276825e-
std	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+
min	-1.141852e+00	-2.544700e+00	-4.004245e+00	-2.516429e+00	-1.746542e+00	-2.020543e+
25%	-8.448851e-01	-7.396938e-01	-6.953060e-01	-4.675972e-01	-6.429600e-01	-7.172147e-
50%	-2.509521e-01	-1.489643e-01	-1.675912e-02	-1.230129e-02	-2.283386e-01	-4.406715e-
75%	6.399473e-01	6.140612e-01	6.282695e-01	3.291706e-01	3.722209e-01	6.147581e-
max	3.906578e+00	2.542136e+00	4.102655e+00	7.955377e+00	7.128551e+00	4.983056e+

In [278]:

```
#import variance inflation factor
```

In [279]:

```
# assign features to X and target to Y



# split the data to test and train with test_size=0.2
```

In [280]:

```
#assign an empty dataframe to variable vif
vif=#

# make a new column 'VIF Factor' in vif dataframe and calculate the variance_inflation_fact
vif['VIF Factor']=#
```

In [281]:

```
# define vif['Features'] with columns names in X

vif['Features']=
```

In [282]:

```
#  round off all the decimal values in the dataframe to 2 decimal places for VIF dataframe
```

Out[282]:

	VIF Factor	Features
0	1.43	Pregnancies
1	2.07	Glucose
2	1.24	BloodPressure
3	1.43	SkinThickness
4	2.04	Insulin
5	1.58	BMI
6	1.05	DiabetesPedigreeFunction
7	1.62	Age

- VIF = 1: Not correlated
- VIF =1-5: Moderately correlated
- VIF >5: Highly correlated

Glucose, Insulin, and Age are having large VIF scores, so lets drop it.

In [283]:

```
# according to above observation , drop  'Glucose', 'Insulin' and 'Age' from X

X=
```

Now again we calculate the VIF for the rest of the features

Again repeat the previous steps to assign an empty dataframe() to vif and make a new column 'VIF Factor' and calculate the variance_inflation_factorfor each X

In [284]:

```
#code here
```

In [285]:

```
#define vif['Features'] as columns of X and return vif with round off to 2 decimal places
```

Out[285]:

	VIF Factor	Features
0	1.05	Pregnancies
1	1.13	BloodPressure
2	1.42	SkinThickness
3	1.50	BMI
4	1.03	DiabetesPedigreeFunction

So now colinearity of features has been reduced using VIF.

The need to fix multicollinearity depends primarily on the below reasons:

1. When you care more about how much each individual feature rather than a group of features affects the target variable, then removing multicollinearity may be a good option
2. If multicollinearity is not present in the features you are interested in, then multicollinearity may not be a problem.

Hip Hip Hurray! Congratulations you have completed the 7th assignment too! Very well done.

Its Feedback Time!

We hope you've enjoyed this course so far. We're committed to help you use "AI for All" course to its full potential, so that you have a great learning experience. And that's why we need your help in form of a feedback here.

Please fill this feedback form https://zfrmz.in/MtRG5oWXBdesm6rmSM7N (https://zfrmz.in/MtRG5oWXBdesm6rmSM7N)
