

What is Feature Selection?

Problem Statement

Let's continue solving the same problem we encountered in the regression modules.

We have with us the complete [Iowa housing dataset](#).

Each row in the dataset describes the properties of a single house as well as the amount it was sold for. The data set contains 81 features and 1300 data points.

Let's apply a simple Linear Regression model on it:

```
#Data Loaded
#Data Split
model=LinearRegression()
model.fit(X_train,y_train)
score=model.score(X_test,y_test)
print(score)
```

Output:

```
0.6665487890480106
```

We get (r2_score) as 0.66.

Knowing this score is low, we apply one Feature Selection technique and we get a subset of 30 features.

After subsetting the data to incorporate only those 30 features, we apply a Linear Regression model

```
#Data Loaded
#Feature Selection method applied to the dataset.
#Data Split

model=LinearRegression()
model.fit(X_train,y_train)
score=model.score(X_test,y_test)
print(score)
```

Output:

```
0.768239
```

The (r2_score) is now 0.76 .

That is an increase of 10% despite the removal of 50 features.

Q: Doesn't that contradict the ML assumption that more features are equivalent to more information?

Ans: All the features in a dataset might not be useful. In a dataset, some features may contribute no information at all, while some features may contribute similar information as the other features.

So selecting the important features is more important than having a high no. of features and that's what feature selection methods help us do.

Definition

Feature selection is the ML process of finding the subset of features that are most relevant for a better predictive model.

When presented data with very high dimensionality (large no. of features), models usually choke because

1. Less training time.
2. Risk of overfitting.

Feature selection methods can help identify as well as remove redundant and irrelevant attributes from data that do not contribute to the predictive power of the model.

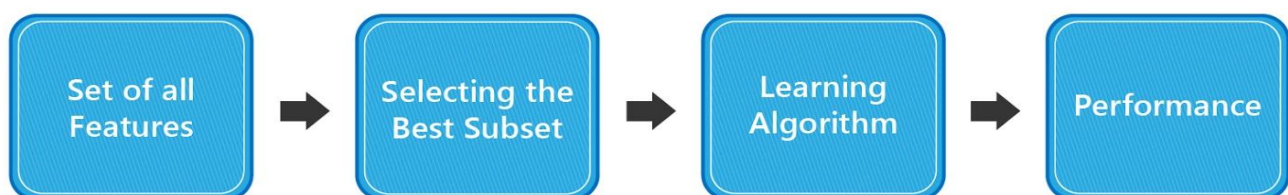
The objective of feature selection is three-fold:

1. Improving the prediction performance of the predictors.
2. Providing faster and more cost-effective predictors.
3. Providing a better understanding of the underlying process that generated the data.

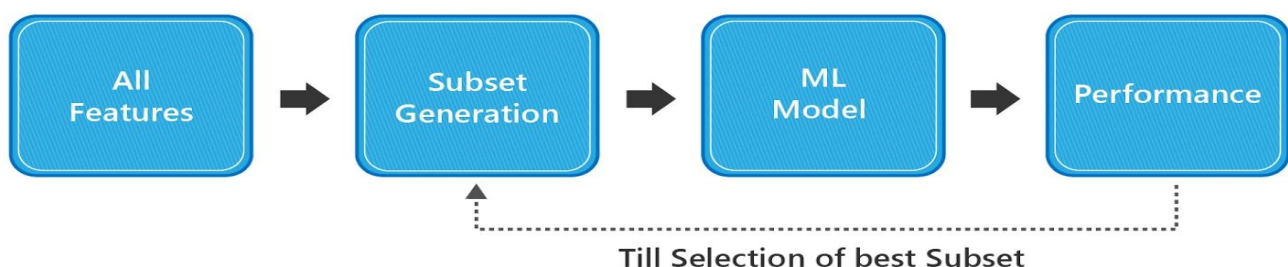
Algorithms of Feature Selection

Following are the categories, the different Feature Selection attributes are broadly divided into:

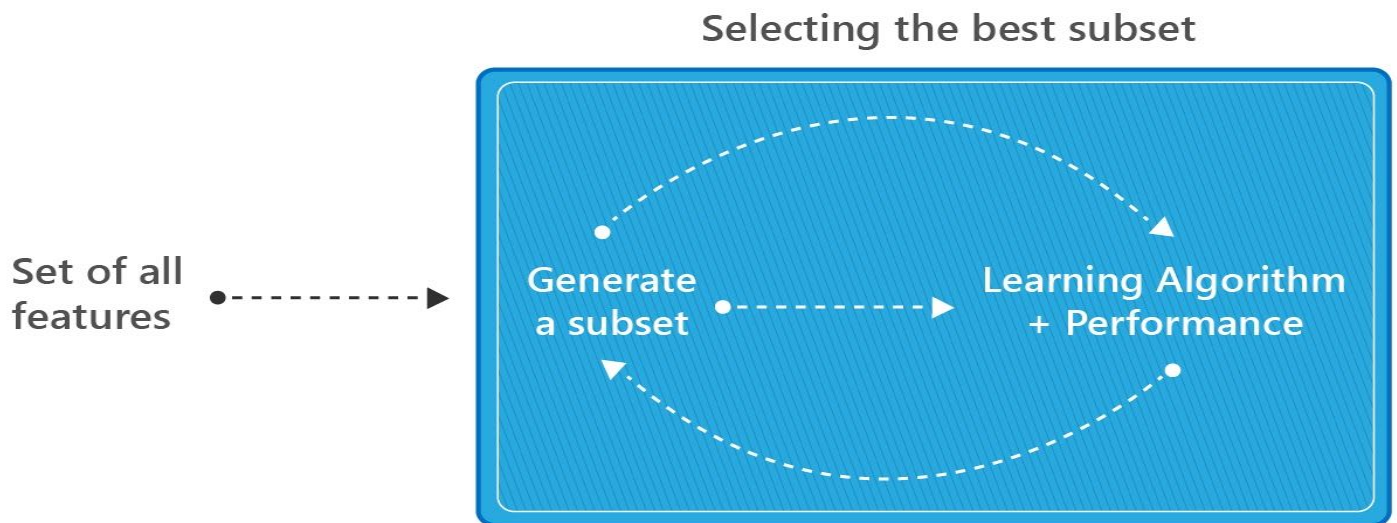
Filter Method - Filter Methods are used to find the relationship between features and the target variable. This results in computing the importance of features.



Wrapper Method - Wrapper Methods select the best subset of features by iteratively checking model performance.

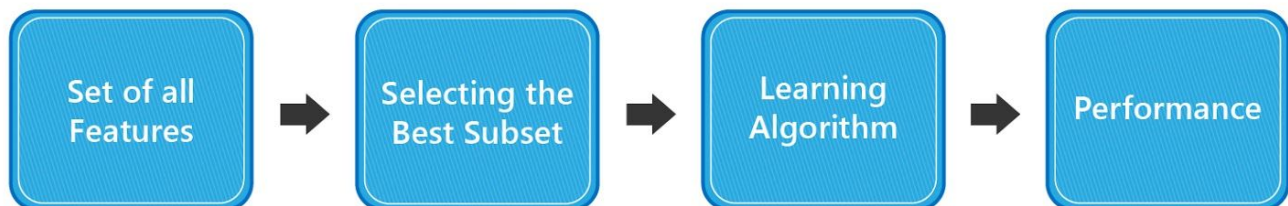


Embedded Method - Embedded methods are the methods implemented by algorithms that have a built-in feature selection 'embedded' in them. It selects the best subset of features during the building of the model itself.



Correlation Coefficient

Overview of the filter method



- Filter methods are a set of a powerful method of feature selection because selection happens independent of any machine learning algorithms.
- Features are selected on the basis of scores of statistical tests between features & target variable.

Following are some of the statistical tests:

- Correlation coefficient
- Chi-Squared test
- ANOVA(F-Score)

Correlation Coefficient

It makes intuitive sense to choose features that are highly correlated to the target variable. The more correlated the features are to the target, the easier it is for the machine to predict it

Selecting features having correlation coefficients above a certain threshold will result in the model performing better.

Additionally one can also filter out redundant features by not selecting certain features that are already strongly correlated with other features.

For eg: If x_1 and x_2 have a strong correlation to the target variable but they are also strongly correlated to each other. In that case, an ML model including both x_1 and x_2 will give almost the same results compared to the ML models where either only x_1 or x_2 was included in the dataset.

There are different methods to calculate the correlation factor, however, Pearson's correlation coefficient is most widely used.

The Pearson coefficient is a measure of the strength of association between two continuous variables.

It has already been covered in `Descriptive Statistics`.

Still, let's refresh,

Pearson's correlation coefficient is calculated by the formula:

$$corr = cov(x,y) / (\sigma_x * \sigma_y)$$

Where,

$cov(x,y)$: covariance between x and y ,

σ_x : standard deviation of x ,

σ_y : standard deviation of y

By taking x as our target variable and y as each of the features, we can easily find how much they are correlated to each other.

Setting up a threshold after that(for eg: correlation coefficient >0.5), we can identify strongly related features and drop the others.

Let's see how we can implement the same in python.

```
#Sample Dataframe
```

```
data=pd.DataFrame({'A':[1,2,3,4,5], 'B':[2,2,6,6,6], 'C':[4,4,6,10,10], 'D':[1,1,1,1,5]})
```

```
print("Dataframe:")
```

```
print(data)
```

```
#Finding the pearson's correlation among variables
```

```
data_corr=data.corr()
```

```
print("\nCorrelation Matrix:")
```

```
print(data_corr)

#Subsetting and only taking those features which are strongly correlated to 'D'

data_corr_d= data_corr[data_corr['D']>0.5]

print("\nFeatures closely related to D(Corr>0.5):")

print(data_corr_d.index.values)
```

Output:

Dataframe:

| | A | B | C | D |
|---|---|---|----|---|
| 0 | 1 | 2 | 4 | 1 |
| 1 | 2 | 2 | 4 | 1 |
| 2 | 3 | 6 | 6 | 1 |
| 3 | 4 | 6 | 10 | 1 |
| 4 | 5 | 6 | 10 | 5 |

Correlation Matrix:

| | A | B | C | D |
|---|----------|----------|----------|----------|
| A | 1.000000 | 0.866025 | 0.938315 | 0.707107 |
| B | 0.866025 | 1.000000 | 0.842701 | 0.408248 |
| C | 0.938315 | 0.842701 | 1.000000 | 0.589768 |
| D | 0.707107 | 0.408248 | 0.589768 | 1.000000 |

Features closely related to D(Corr>0.5):

```
['A' 'C' 'D']
```

If you compare values of D with A and C you can clearly see the correlation between them.

A and D have two same values.

C and D have similar pattern distribution for two values[1 map to 4]

Task:

Feature selection using correlation

In this task, after loading the housing dataset, we will filter out features based on Pearson correlation and then train the model with only the selected features.

Instructions:

- Load the dataset from path using the `"read_csv()"` method from pandas and store it in a variable called `'ames'`
- Store all the independent features of `'ames'` in a variable called `x`
- Store the target variable (SalePrice) of `'ames'` in a variable called `y`
- Split `'x'` and `'y'` into `x_train, x_test, y_train, y_test` using `train_test_split()` function. Use `test_size = 0.3` and `random_state = 0`

Finding Correlation

- To simplify the process, create a new column in `'x_train'` called `Class` which stores the value of `'y_train'`.
- Find the correlation dataframe among all the features using `"x_train.corr()"` and store it in a variable called `"t_corr"`. Since we are only interested in the correlation of features with the target variable, only extract the `Class` column of `'t_corr'` and save it back to `'t_corr'`
- From `'t_corr'`, extract only those columns whose absolute correlation score is greater than 0.5 using `index` function and store them in a variable called `'corr_columns'`. From that list drop `Class` column.
- Create a subset dataframe from `'x_train'` having only the columns stored in `'corr_columns'`. Save the new subsetted dataframe in `'x_train_new'`
- Create a subset dataframe from `'x_test'` having only the columns stored in `'corr_columns'`. Save the new subsetted dataframe in `'x_test_new'`
- Initialise a linear regression model with `LinearRegression()` and save it to a variable called `'model'`.
- Fit the model on the training data `'x_train_new'` and `'y_train'` using the `'fit()'` method.
- Find out the r^2 score between `x_test_new` and `'y_test'` using the `'score()'` method and save it in a variable called `'corr_score'`

Things to ponder

- Did the accuracy increase from the base score of 0.66?
- As a side task, see how many features were actually selected on the basis of Pearson correlation?

Code:


```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Code starts here
#Loading of data
ames = pd.read_csv(path)

X=ames.drop(['SalePrice'],1)
```

```

y=ames['SalePrice'].copy()
#Splitting of data
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3, random_state=0)
#Creating temp. dataframe
X_train['Class']=y_train
t_corr=X_train.corr()
t_corr=t_corr['Class']
#Selecting columns having correlation higher than 0.5
corr_columns=t_corr[abs(t_corr)>0.5].index
#Dropping the column `Class`
corr_columns=corr_columns.drop('Class')
print(corr_columns)
#Updating train and test dataframes
X_train_new=X_train[corr_columns]
X_test_new=X_test[corr_columns]
#Initialising the model
model=LinearRegression()
#Fitting the model
model.fit(X_train_new,y_train)
#Finding the score of the model
corr_score=model.score(X_test_new,y_test)
print(corr_score)
#Checking how many columns were selected
print(len(X_train_new.columns))


```

CODE
DATA
RESET CODE
Code Saved


```

29 #Dropping the column `Class`
30 corr_columns=corr_columns.drop('Class')
31 print(corr_columns)
32 #Updating train and test dataframes
33 X_train_new=X_train[corr_columns]
34
35 X_test_new=X_test[corr_columns]
36
37 #Initialising the model
38 model=LinearRegression()
39
40 #Fitting the model
41 model.fit(X_train_new,y_train)
42
43 #Finding the score of the model
44 corr_score=model.score(X_test_new,y_test)
45 print(corr_score)
46
47 #Checking how many columns were selected
48 print(len(X_train_new.columns))

```

Congrats! You have successfully implemented feature selection using the correlation coefficient. You can see that out of 80 columns only 13 columns are selected based on Pearson correlation. Also, we can see that accuracy has increased from 0.66 to 0.72.


> TRY IT
SUBMIT

```

Index(['OverallQual', 'YearBuilt', 'YearRemodAdd', 'ExterQual', 'BsmtQual',
      'TotalBsmtSF', '1stFlrSF', 'GrLivArea', 'FullBath', 'KitchenQual',
      'TotRmsAbvGrd', 'GarageCars', 'GarageArea'],
      dtype='object')
0.7227056628201056
13
{
  "name": "stderr",
  "text": "/opt/grevaatom/kernel-gatewav/runtime-environments/lib/pvthon3.6/site-packages/ipvkernel launcher

```

Chi Squared Test

Another way to identify relationship between features and target variable is the chi-square test statistic.

Similar to Pearson correlation, this is another preferred way to check for strongly correlated features.

Chi Squared Test has been covered in `Inferential Statistics`.

Still let's refresh,

The chi-squared test of independence is used to determine whether the two variables are inter-related to each other.

Like any statistical hypothesis test, the Chi-square test has both a null hypothesis and an alternative hypothesis.

The hypothesis for a chi-square test of independence are as follows.

H_0 : Variable A and Variable B are independent.

H_1 : Variable A and Variable B are not independent.

In this case we will be calculating a chi-square test statistic as follows

$$\chi^2 = \sum \frac{(\text{observed} - \text{expected})^2}{\text{expected}}$$

In the formula, observed is the actual observed count for each category and expected is the expected count based on the distribution of the population for the corresponding category.

In the Chi-square context, the word “expected” is equivalent to what you’d expect if the null hypothesis is true. If your observed distribution is sufficiently different than the expected distribution (no relationship), you can reject the null hypothesis and infer that the variables are related.

Since the chi-square test measures dependence between variables, using this function “weeds out” the features that are the most likely to be independent of the target variable class and therefore irrelevant for prediction.

Q: In Inferential statistics, we learned chi-square tests is an independence test between two categorical variables. How are the numerical variables dealt with?

A: Binning

Explanation: Even for categorical variables, the null hypothesis is defined as

The ‘frequency distribution’ of certain events observed in a sample is consistent with a particular theoretical distribution.

As an example look at these sets of variables:

```
a = ['dog', 'cat', 'dog', 'cat']
```

```
b = ['wild', 'trained', 'trained', 'trained']
```

The categorical variables a and b can be compared by counting the co-occurrences, and this is what happens with a chi-squared test:

| - | Dog | Cat |
|---------|-----|-----|
| wild | 1 | 0 |
| trained | 1 | 2 |

However, you can also binarise the values of 'a' and get the following variables:

```
a1 = [1, 0, 1, 0]
```

```
a2 = [0, 1, 0, 1]
```

```
b = ['wild', 'trained', 'trained', 'trained']
```

Counting the values is now equal to summing the values that correspond to the value of b.

| - | a1 | a2 |
|---------|----|----|
| wild | 1 | 0 |
| trained | 1 | 2 |

For feature selection purposes, it has a different python implementation than the one you learned in Inferential Statistics.

```
data=pd.DataFrame({'A':[1,2,3,4,5], 'B':[2,2,6,6,6], 'C':[4,4,4,2,10], 'D':[1,1,1,1,5], 'E':  
[2,2,2,1,5]})  
print('Dataframe:')  
print(data)
```

```
#Using chi square score to calculate best two features  
test = SelectKBest(score_func=chi2, k=2)
```

```
#Transforming the data based on chi square(Target variable is E)  
data_chi= test.fit_transform(data.iloc[:,4], data.iloc[:,4])
```

#In the above function, 'data.iloc[:,4]' are the features, 'data.iloc[:,4]' is the target variable

```
print("\nTwo columns having the highest chi square score with respect to 'E'")
print(data_chi)
```

Output:

Dataframe:

| | A | B | C | D | E |
|---|---|---|----|---|---|
| 0 | 1 | 2 | 4 | 1 | 2 |
| 1 | 2 | 2 | 4 | 1 | 2 |
| 2 | 3 | 6 | 4 | 1 | 2 |
| 3 | 4 | 6 | 2 | 1 | 1 |
| 4 | 5 | 6 | 10 | 5 | 5 |

Two columns having the highest chi square score **with** respect to 'E'

```
[[ 4  1]
 [ 4  1]
 [ 4  1]
 [ 2  1]
 [10  5]]
```

If you compare values of E with C and D you can clearly see why Chi-Square test score is highest for them. Both C and D have the most similar frequency distribution when compared to E.

Chi Square test

- Store all the features of 'ames' (Loaded in the previous task) in a variable called `X`
- Store the target variable (`SalePrice`) of 'ames' in a variable called `y`
- Initialise a "SelectKBest()" with the parameters `score_func=chi2` & `k=60` and save it to a variable called 'test'.
- Split 'X' and 'y' into `X_train,X_test,y_train,y_test` using `train_test_split()` function. Use `test_size = 0.3` and `random_state = 0`
- Fit 'test' on the training data 'X_train' and 'y_train' using the 'fit_transform()' method. Store the result back into 'X_train'
- Transform 'X_test' using the 'transform()' method of test. Store the result back into 'X_test'
- Initialise a linear regression model with `LinearRegression()` and save it to a variable called 'model'.
- Fit the model on the training data 'X_train' and 'y_train' using the 'fit()' method.
- Find out the `r2` score between `X_test` and 'y_test' using the 'score()' method and store it in a variable called 'chi2_score'

```
2 from sklearn.feature_selection import chi2
3 from sklearn.feature_selection import SelectKBest
4 # Code starts here
5 X=ames.drop(['SalePrice'],1)
6 y=ames['SalePrice'].copy()
7 #Splitting dataframe into test and train
8 X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3, random_state=0)
9 #Initialising the score function
10 test = SelectKBest(score_func=chi2, k=60)
11 #Fitting and transforming the model on X_train
12 X_train= test.fit_transform(X_train, y_train)
13 #Fitting and transforming the model on X_test
14 X_test= test.transform(X_test)
15 #Initialising the Linear Regression model
16 model=LinearRegression()
17 #Fitting the model
18 model.fit(X_train,y_train)
19 #Finding the model score
20 chi2_score=model.score(X_test,y_test)
21 print(chi2_score)
```

Congrats! You have successfully implemented feature selection using a chi-square test. You can see that accuracy has increased to 0.75.

CONTINUE

TRY IT

OUTPUT

RESULT

0.7526152480700989

ANOVA

Analysis of variance (ANOVA) is another method to check for a close relationship between two variables.

Just like any statistical test, it has two hypothesis

H_0 : The mean (average value) is the same for all groups. (NULL Hypothesis)

H_1 : The mean is not the same for all groups.

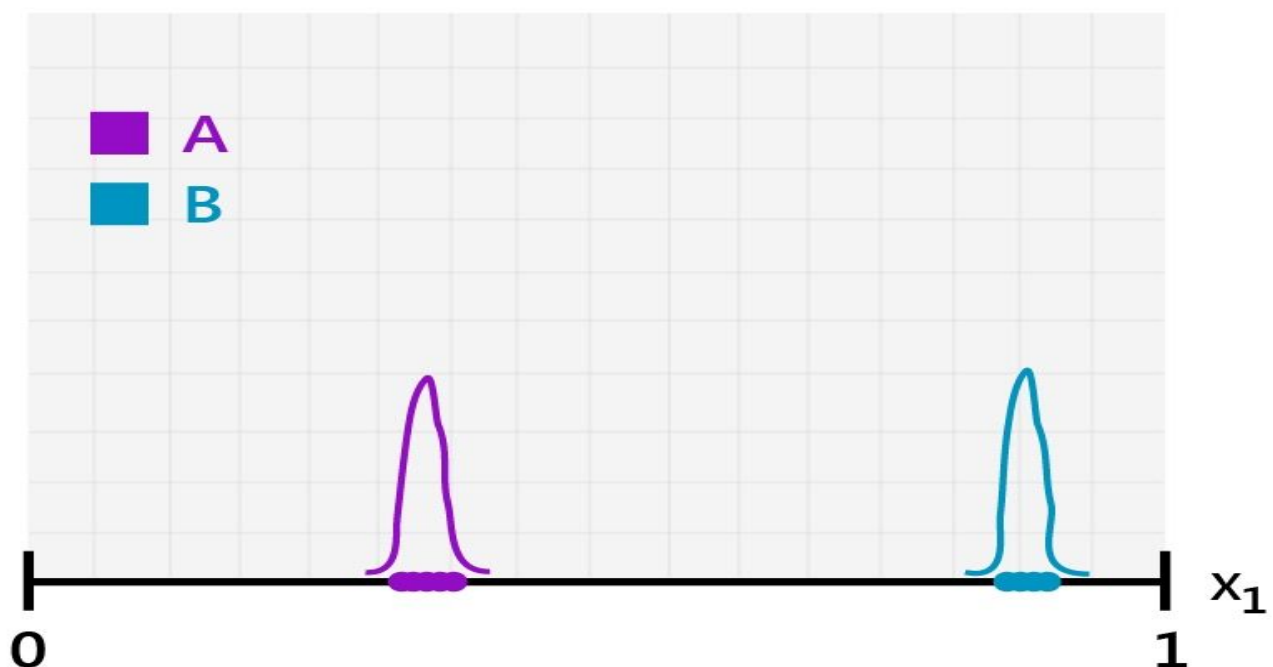
The proportion of variance (or variability) explained by the two variables is calculated using F -score to figure out if they are closely related

We won't be delving much into the actual Anova Tests and only focusing on its Feature Selection application.

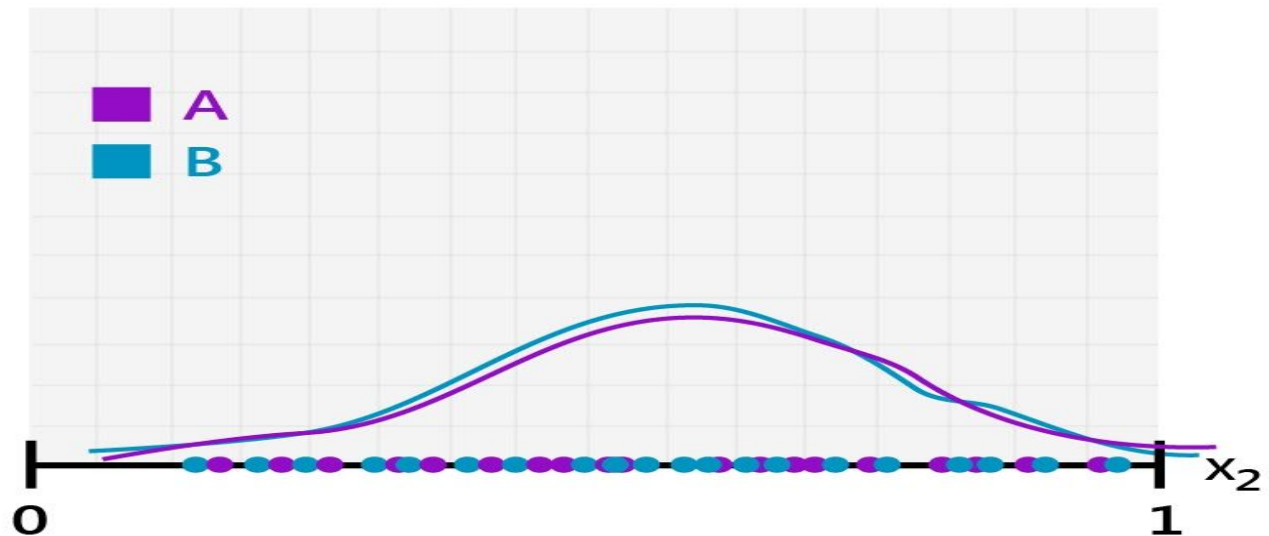
Intuition Consider a binary classification problem :

| x1 | x2 | x3 | ... | class |
|-----|-----|-----|-----|-------|
| 0.3 | 0.5 | 0.1 | ... | A |
| 0.1 | 0.7 | 0.4 | ... | B |
| 0.1 | 0.1 | 0.2 | ... | A |
| 0.2 | 0.4 | 0.2 | ... | A |
| 0.5 | 0.7 | 0.8 | ... | B |

Assume the plot for target variable with respect to x_1 looks like:



Assume the plot for target variable with respect to x_2 looks like:



Q: If one has to predict say Class A based on x_1 and x_2 , which feature do you think you can predict better?

A: x_1

Reason: There is no overlap of target variable with respect to x_1 data points.

Though x_1 and x_2 are two extreme cases, it can be seen that the overlap is reduced when

1. The means of A and B are more separated (with respect to a feature x_i)
2. The variances of A and B are small (with respect to a feature x_i)

The F-score captures these two properties, such that a high F-score reflects a small overlap.

Definition

F-score is defined as the following:

Variance between the samples (of a feature and target variable) / Variance within the samples

Variance within is found by:

$$\frac{\sum_{i=1}^m \sum_{j=1}^n (x_j - \bar{x}_i)^2}{n - m}$$

where **m** is the number of samples (groups) and **n** is the size of the m samples altogether (observations).

Variance between is found by:

$$\frac{\sum_{j=1}^n n_j (\bar{x}_j - \bar{X})^2}{m - 1}$$

(\bar{x}_j is the mean of sample j and \bar{X} is the grand mean)

With respect to Feature Selection, Anova uses F-tests to statistically assess among all features which features are more likely to correctly predict the target variable.

Working

Let's try to understand Feature Selection using ANOVA better with an example:

A large car company recently extended its working days to include Sunday from the previous schedule of Mon-Sat. The company now wants to optimise the sales on Sunday.

A large amount of important sales happens during the days of Friday & Saturday and therefore the data science team wants to know between Friday and Saturday Sales, which day will help predict the sales on Sunday.

The company collects data for the number of car sales each day for two months.

Following is the data:

Fridays: 288, 292, 310, 267, 243, 293, 255, 273

Saturdays: 276, 323, 298, 256, 277, 309, 312, 265, 311

Sundays: 243, 279, 301, 285, 274, 243, 228, 298, 255

Let's find which of the groups explains greater variance of the data:

Group 1(Sat, Sun):

$mean_{(sat)} = 291.8$

$mean_{(sun)} = 267.3$

mean of means = 279.55

Step 1: Calculating the Sum of Squares within groups:

$[(276-291.8)^2 + \dots + (311-291.8)^2 + (243-267.3)^2 + \dots + (255-267.3)^2] = 10002.4$

Step 2: Calculating degrees of freedom:

Here, degrees of freedom = No. of observation - No. of groups = 18-2 = 16

Step 3: Computing variance within:

Sum of squares within groups/degrees of freedom

$= [(276-291.8)^2 + \dots + (311-291.8)^2 + (243-267.3)^2 + \dots + (255-267.3)^2] / [18-2] = 625.185$

Step 4: Calculating the Sum of Squares between groups:

$[9(291.8-279.55)^2 + 9(267.3-279.55)^2] = 2701.125$

Step 5: Calculating degrees of freedom:

Here, degrees of freedom= No. of groups- 1 = 2-1 = 1

Step 6: Computing variance between:

$$[9(291.8-279.55)^2+9(267.3-279.55)^2]/[2-1] = 2701.125$$

Step 7: Computing the F-score:

we will get,

$$F_{\text{score}} = \{2701.125\} / \{625.18\} = 4.32$$

Group 2(Fri, Sun):

$$\text{mean}(\text{fri}) = 277.6$$

$$\text{mean}(\text{sun}) = 267.3$$

mean of means = 272.45

Step 1: Calculating the Sum of Squares within groups: $[(288-277.6)^2+\dots+(273-277.6)^2 + (243-267.3)^2+\dots+(255-267.3)^2] = 8893.8$

Step 2: Calculating degrees of freedom:

Here, degrees of freedom= No. of observation - No. of groups=17 -2= 15

Step 3: Computing variance within:

Sum of squares within groups/degrees of freedom

$$=[(288-277.6)^2+\dots+(273-277.6)^2 + (243-267.3)^2+\dots+(255-267.3)^2]/[17-2]= 592.92$$

Step 4: Calculating the Sum of Squares between groups:

$$[8(277.6-272.45)^2 + 9(267.3-272.45)^2]= 450.88$$

Step 5: Calculating degrees of freedom:

Here, degrees of freedom= No. of groups- 1 = 2-1 = 1

Step 6: Computing variance between:

$$[8(277.6-278.9)^2 + 9(291.8-279.55)^2+ 9(267.3-279.55)^2]/[2-1] = 450.88$$

Step 7: Computing the F-score:

we will get,

$$F_{\text{score}} = \{450.88\} / \{592.2\} = 0.76$$

It can be seen from the F-Score that group 1 explains more variance than group 2.

Conclusion: Saturday Sales will be a better predictor than Friday Sales

Similar to the above example, the ANOVA table tells the proportion of variance explained by the features with respect to the target variable

Obviously the features that explain the largest proportion of the variance should be retained.

It has two python implementations in the form of `f_classif` and `f_regression`

Since in our Ames Dataset problem we are dealing with a regression problem, we will learn how to implement `f_regression` score from sklearn library.

Its implementation is very similar to the implementation of 'chi-square' score

```
data=pd.DataFrame({'A':[1,2,3,4,5], 'B':[2,2,6,6,6], 'C':[4,4,6,10,10], 'D':[1,1,1,1,5],  
'E':[2,2,2,1,5]})  
print('Dataframe:')  
print(data)  
  
#Using ANOVA score to calculate best two features  
test = SelectKBest(score_func=f_regression, k=2)  
  
#Transforming the data based on ANOVA(Target variable is E)  
data_anova= test.fit_transform(data.iloc[:,4], data.iloc[:,4])  
  
print("\nTwo columns having the highest ANOVA score with respect to 'E'")  
print(data_anova)
```

Output:

```
Dataframe:  
   A  B  C  D  E  
0  1  2  4  1  2  
1  2  2  4  1  2  
2  3  6  6  1  2  
3  4  6 10  1  1  
4  5  6 10  5  5
```

```
Two columns having the highest ANOVA score with respect to 'E'  
[[1 1]  
 [2 1]  
 [3 1]  
 [4 1]  
 [5 5]]
```

If you compare values of E with A and D you can clearly see why Anova Score is highest for them.

A has the highest variance between when compared to other columns whereas D has the lowest variance within when compared to other columns.

Code:

```
# import packages

import pandas as pd

from sklearn.feature_selection import f_regression

from sklearn.feature_selection import SelectKBest

# Code starts here

X=ames.drop(['SalePrice'],1)

y=ames['SalePrice'].copy()

#Splitting dataframe into test and train

X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3, random_state=0)

#Initialising the score function

test = SelectKBest(score_func=f_regression, k=60)

#Fitting and transforming the model on X_train

X_train = test.fit_transform(X_train, y_train)

#Fitting and transforming the model on X_test

X_test = test.transform(X_test)

#Initialising the Linear Regression model

model = LinearRegression()

#Fitting the model

model.fit(X_train,y_train)

#Finding the model score

f_regress_score = model.score(X_test,y_test)

print(f_regress_score)

# Code ends here
```


Anova Score

- Store all the features of 'ames' (Loaded in the first task) in a variable called `X`
- Store the target variable (`SalePrice`) of 'ames' in a variable called `y`
- Initialise a `SelectKBest()` with the parameters `score_func=f_regression` & `k=60` and save it to a variable called `'test'`.
- Split `'X'` and `'y'` into `X_train,X_test,y_train,y_test` using `train_test_split()` function. Use `test_size = 0.3` and `random_state = 0`
- Fit `'test'` on the training data `'X_train'` and `'y_train'` using the `'fit_transform()'` method. Store the result back into `'X_train'`
- Transform `'X_test'` using the `'transform()'` method of `test`. Store the result back into `'X_test'`
- Initialise a linear regression model with `LinearRegression()` and save it to a variable called `'model'`.
- Fit the model on the training data `'X_train'` and `'y_train'` using the `'fit()'` method.
- Find out the r^2 score between `X_test` and `'y_test'` using the `'score()'` method and store it in a variable called `'f_regress_score'`

```
1 # import packages
2 import pandas as pd
3 from sklearn.feature_selection import f_regression
4
5 from sklearn.feature_selection import SelectKBest
6
7
8 # Code starts here
9 X=ames.drop(['SalePrice'],1)
10 y=ames['SalePrice'].copy()
11
12 #Splitting dataframe into test and train
13 X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3, random_
14
15 #Initialising the score function
16 test = SelectKBest(score_func=f_regression, k=60)
17
18 #Fitting and transforming the model on X_train
19 X_train = test.fit_transform(X_train, y_train)
20
```

Congrats! You have successfully implemented feature selection using Anova Method. You can see that the accuracy has increased by 1%.

CONTINUE

> TRY IT

OUTPUT

RESULT

0.7566701199447283

A researcher believes that recall of verbal material differs with the level of processing. He divided his subjects into three groups. In the low processing group, participants read each word and were instructed to count the number of letters in the word. In the medium processing group, participants were asked to read each word and think of a word that

- rhymed. In the high processing group, participants were asked to read each word and try to memorize it for later recall. Each group was allowed to read the list of 30 words three times, then they were asked to recall as many of the words on the list as possible. If the researcher wants to know whether the three groups have different amounts of recall, what type of statistical test should be used?



Two-way ANOVA

One-way ANOVA

Explanation:

The null hypothesis is that the three population means are the same, indicating that the three interventions have the same effect.

One-way ANOVA is appropriate for this situation.

An assumption for the F-test in one-way ANOVA is that the distributions of scores within each population are close to normal and that the variances of scores within each population are approximately the same. You should begin your analysis by checking to see if these assumptions are reasonably well met.

- Which among the following is not a filter method?



L1 Regularization

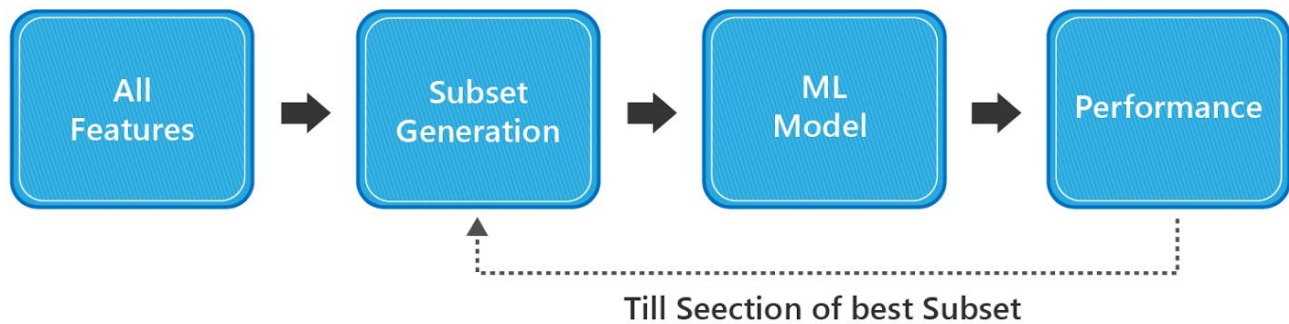
L1 Regularization

Explanation:

L1 regularization is an embedded method of feature selection

Wrapper methods

Overview of wrapper method



- Wrapper Methods generate models with different subsets of feature and gauge their model performances.
- Wrapper methods can give high classification accuracy for particular classifiers but generally, they have high computational complexity.

Following are the wrapper methods:

- Forward Selection
- Backward Selection
- RFE

Before discussing RFE, let's briefly look at Forward Selection and Backward Selection.

Forward Selection

Forward selection is an iterative technique in which we begin with having no features in the model. In every cycle, we continue including features which best enhances our model till an adding of another variable does not enhance the performance of the model. Consider the following python code:

```
#Creating a dataframe
data=pd.DataFrame({'A':[1,2,3,4,5,6,7,8,9,10], 'B':[4,4,6,10,10,4,4,6,10,10],
'C':[1,1,1,1,5,1,1,1,1,5], 'D':[2,2,2,1,5,2,2,2,1,5]})
print('Dataframe:')
print(data)
#Only selecting feature B
X=data[['B']]
y=data['D'].copy()
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3, random_state=0)
model =LinearRegression()
#Fitting the model
model.fit(X_train,y_train)
print("Score when features [B] is selected:", model.score(X_test,y_test))
```

Output

Dataframe:

| | A | B | C | D | E |
|---|----|---|----|---|---|
| 0 | 1 | 2 | 4 | 1 | 2 |
| 1 | 2 | 2 | 4 | 1 | 2 |
| 2 | 3 | 6 | 6 | 1 | 2 |
| 3 | 4 | 6 | 10 | 1 | 1 |
| 4 | 5 | 6 | 10 | 5 | 5 |
| 5 | 6 | 2 | 4 | 1 | 2 |
| 6 | 7 | 2 | 4 | 1 | 2 |
| 7 | 8 | 6 | 6 | 1 | 2 |
| 8 | 9 | 6 | 10 | 1 | 1 |
| 9 | 10 | 6 | 10 | 5 | 5 |

Score when features [B] is selected: 0.06698063840920998

Let's add c to the model as well

```
#Selecting B,C
X=data[['B','C']]
y=data['D'].copy()
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3, random_state=0)
model =LinearRegression()
#Fitting the model
model.fit(X_train,y_train)
print("Score when features [B,C] is selected:", model.score(X_test,y_test))
```

Output

Score when features [B,C] is selected: 0.9904640813731723

Let's now add A to the model.

```
#Selecting A,B,C
X=data[['A','B','C']]
y=data['D'].copy()
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3, random_state=0)
model =LinearRegression()
#Fitting the model
model.fit(X_train,y_train)
print("Score when features [A,B,C] is selected:", model.score(X_test,y_test))
```

Output

Score when features [A,B,C] is selected: 0.9812539002371345

The score increased when predicting using B & C but adding A decreased it. Therefore we stop the iteration process.

Backward Selection

In backward selection, we do the opposite of forward selection. We start with all the features and remove the least significant feature after each iteration which improves the performance of the model. We continue this until no improvement is observed on the removal of features. Consider the following python code:

```
#Creating a dataframe
data=pd.DataFrame({'A':[1,2,3,4,5,6,7,8,9,10], 'B':[4,4,6,10,10,4,4,6,10,10],
'C':[1,1,1,1,5,1,1,1,1,5], 'D':[2,2,2,1,5,2,2,2,1,5]})
print('Dataframe:')
print(data)
#Selecting A,B,C
X=data[['A','B','C']]
y=data['D'].copy()
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3, random_state=0)
model =LinearRegression()
#Fitting the model
model.fit(X_train,y_train)
print("Score when features [A,B,C] is selected:", model.score(X_test,y_test))
```

Output

Score when features [A,B,C] is selected: 0.9812539002371345

Let's remove A from the model

```
#Removing A from the dataframe
X=data[['B','C']]
y=data['D'].copy()
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3, random_state=0)
model =LinearRegression()
#Fitting the model
model.fit(X_train,y_train)
print("Score when features [B,C] is selected:", model.score(X_test,y_test))
```

Output

Score when features [B,C] is selected: 0.9904640813731723

Removing c from the model

```
#Only selecting feature B
X=data[['B']]
y=data['D'].copy()
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3, random_state=0)
model =LinearRegression()
#Fitting the model
model.fit(X_train,y_train)
print("Score when features [B] is selected:", model.score(X_test,y_test))
```

Output

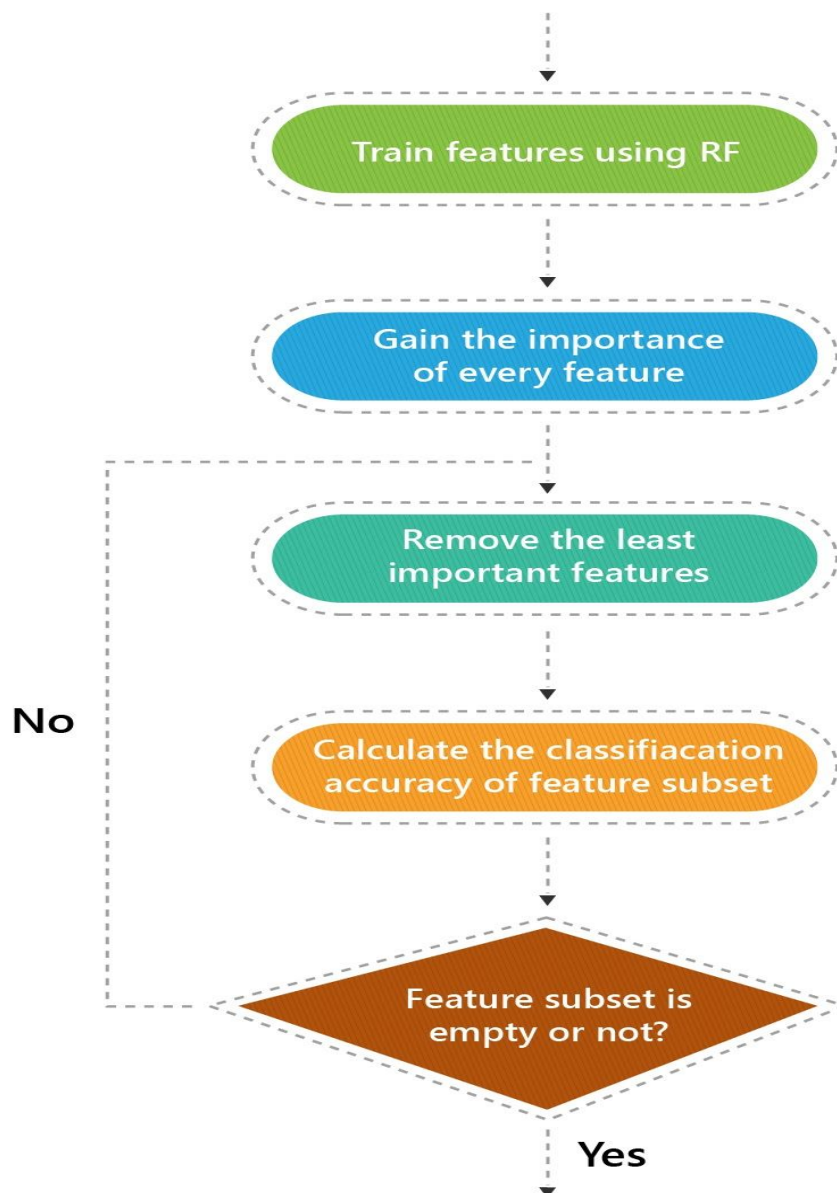
Score when features [B] is selected: 0.06698063840920998

In the above method, the score increased when we dropped **A** but the dropping of **C** resulted in a drastic decrease of a score. Therefore we stop the iteration process.

Though effective in certain cases, these two methods(Forward and Backward) can provide problems when dealing with especially large or highly-dimensional datasets. Though not popularly used, you can implement the same using [mlxtend](#) library

Recursive Feature Elimination:

RFE method involves repeatedly constructing a specific model and selecting the most impactful (or least impactful feature), setting that feature aside and then repeating the process with the rest of the features. This process is iterated until all features in the dataset are used up(or other stopping criteria are satisfied). Features are then ranked according to when they were eliminated. In this way, RFE is able to work out the best subset of features that will enhance the performance of the model.



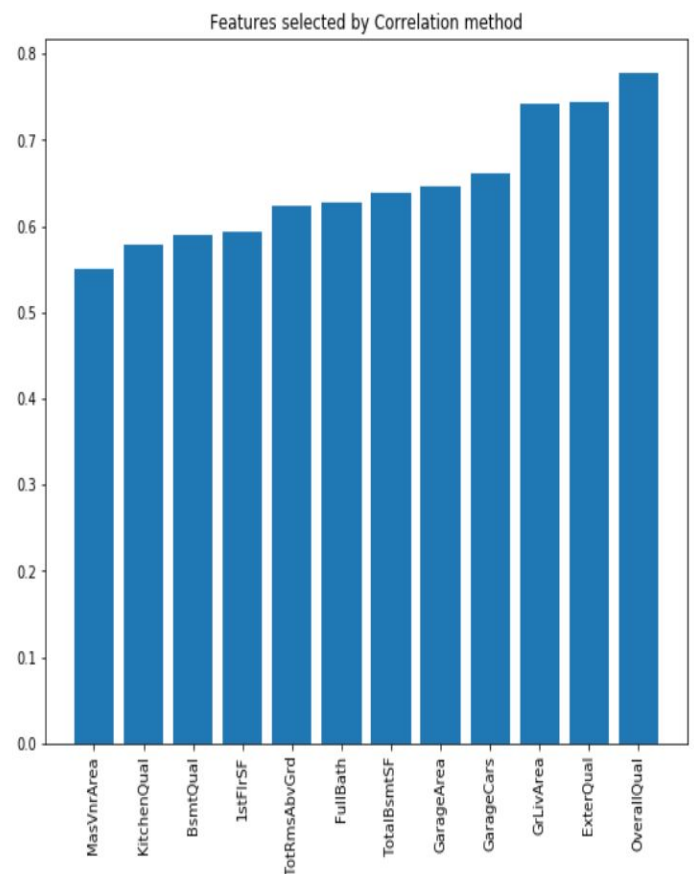
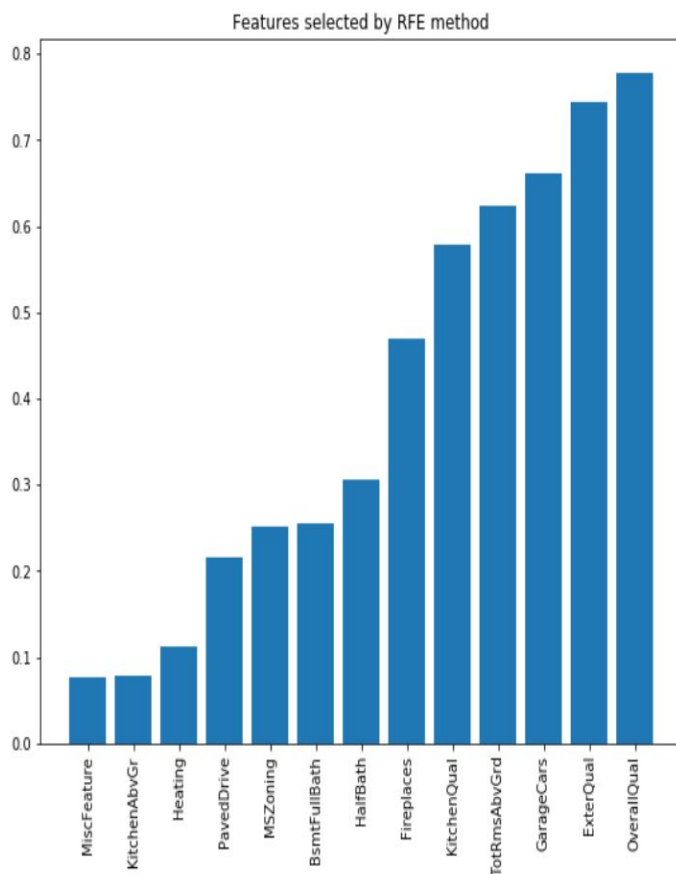
Let's see its python implementation. For this, we will use a subset of our original dataset containing only 200 data points.

```
from sklearn.feature_selection import RFE
#Creating a sample of 200 data points
df_new=ames.sample(200,random_state=49)
X = df_new.drop(['SalePrice'],1)
y=df_new['SalePrice'].copy()
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3, random_state=0)
#Selecting model as Linear Regressor
model =LinearRegression()
model.fit(X_train,y_train)
print("Score before RFE:", model.score(X_test,y_test))
#Passing the model along with no. of features you wish to select
selector = RFE(model,13)
#Fitting the data with the above conditions
X_train_rfe = selector.fit_transform(X_train, y_train)
X_test_rfe=selector.transform(X_test)
model.fit(X_train_rfe,y_train)
print("Score after RFE:",model.score(X_test_rfe,y_test))
```

Output:

```
Score before RFE: 0.599523748467163
Score after RFE: 0.8038362233621985
```

Following is a image comparing the 13 features(and their correlation score with the target variable SalePrice) selected by the RFE method and Pearson coefficient method for the above sampled dataset:



It's interesting to note that RFE isn't selecting features which have the max correlation with the target variable. It's also inherently identifying correlation between features as well.

RFE will always provide the most optimised features and almost always results in a good performance, but it has a major drawback. This whole process of going back and forth with the features is a very time-consuming process. For datasets with large feature space(which is almost always in any ML problem), this method is not a good choice.

Task:

RFE

Let's now try to implement RFE and see the score in the whole credit dataset.

In this task we will also try to identify the optimum no. of features to use

- Store all the features of 'ames' (Loaded in the first task) in a variable called `x`
- Store the target variable (SalePrice) of 'ames' in a variable called `y`
- Three variables 'nof_list', 'high_score' and 'nof' are already defined for you.
- Run a for loop `n` over every element of 'nof_list'. Inside the loop perform the following tasks:
 - Split 'x' and 'y' into `X_train, X_test, y_train, y_test` using `train_test_split()` function. Use `test_size = 0.3` and `random_state = 0`
 - Initialise a linear regression model with and save it to a variable called 'model'.
 - Initialise a `RFE()` object with parameters 'model' & and `n` store it to a variable called 'rfe'.
 - Fit 'rfe' on the training data 'X_train' and 'y_train' using the 'fit_transform()' method. Store the result into 'X_train_rfe'
 - Transform 'X_test' using the 'transform()' method of `rfe`. Store the result into 'X_test_rfe'
 - Fit the model on the training data 'X_train_rfe' and 'y_train' using the 'fit()' method.
 - Using if condition statement, check whether the `R_2` score is greater than `high_score` and store the highest `R2` score in 'high_score' and the `n` associated with it in `nof`
- Come outside the for loop and print the value of `high_score` and `nof`.

Code:

```
# Feature Extraction with RFE

from pandas import read_csv

from sklearn.feature_selection import RFE

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import ExtraTreesClassifier

#no of features list

nof_list=[20,30,40,50,60,70,80]
```



```

#Variable to store the highest score

high_score=0

#Variable to store the optimum features

nof=0

#Code begins here

X = ames.drop(['SalePrice'],1)

y=ames['SalePrice'].copy()

#Loop to select the optimum features

for n in nof_list:

    X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3,
random_state=0)

    model = LinearRegression()

    rfe = RFE(model, n)

    X_train_rfe = rfe.fit_transform(X_train, y_train)

    X_test_rfe = rfe.transform(X_test)

    model.fit(X_train_rfe,y_train)

    if model.score(X_test_rfe,y_test)>high_score:

        high_score=model.score(X_test_rfe,y_test)

        nof=n

#Printing the no. features with the highest score along with the highest score

print("No. of features=",nof, "gives the best score=",high_score)

```


Let's now try to implement RFE and see the score in the whole credit dataset.

In this task we will also try to identify the optimum no. of features to use

- Store all the features of 'ames' (Loaded in the first task) in a variable called `X`
- Store the target variable (`SalePrice`) of 'ames' in a variable called `y`
- Three variables 'nof_list', 'high_score' and 'nof' are already defined for you.
- Run a for loop `n` over every element of 'nof_list'. Inside the loop perform the following tasks:
 - Split 'X' and 'y' into `X_train,X_test,y_train,y_test` using `train_test_split()` function. Use `test_size = 0.3` and `random_state = 0`
 - Initialise a linear regression model with and save it to a variable called 'model'.
 - Initialise a `RFE()` object with parameters 'model' & `n` store it to a variable called 'rfe'.

```
1 # Feature Extraction with RFE
2 from pandas import read_csv
3 from sklearn.feature_selection import RFE
4 from sklearn.linear_model import LinearRegression
5 from sklearn.ensemble import ExtraTreesClassifier
6 #no of features list
7 nof_list=[20,30,40,50,60,70,80]
8 #Variable to store the highest score
9 high_score=0
10 #Variable to store the optimum features
11 nof=0
12 #Code begins here
13 X=ames.drop(['SalePrice'],1)
14 y=ames['SalePrice'].copy()
15 for n in nof_list:
16     X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.3, random_state = 0)
17     model = LinearRegression()
18     rfe = RFE(model,n)
19     X_train_rfe = rfe.fit_transform(X_train,y_train)
20     X_test_rfe = rfe.transform(X_test)
```

```
{
  "name": "stderr",
  "text": "/opt/greyatom/kernel-gateway/runtime-environments/lib/python3.6/site-packages/skle
}
0.7532423678317913
30
```

Correct Answers

- Which of the following methods takes into account the
1. information dependency between the evaluated features?



Filter methods

Wrapper methods

Explanation:

Wrapper method approaches the evaluation which takes into account the information dependency between the evaluated features, whereas, Filter approaches assume no dependency between the evaluated features.

Assuming the application is classification, Wrapper method approaches evaluation, use the classifier itself as evaluation criteria, whereas filter method, use general criteria such as Information Gain, Mutual Information, Chi Square, ... etc, which makes the selection of features using filter approaches not suitable for the classifier that use them.

- Which one of the following methods should be
2. preferred when dealing with large amounts of data?



Wrapper methods

Filter methods

Explanation:

We use `Filter` approaches for larger data because these approaches are rapid and for small size of data it is better to use Wrapper (KNN, SVM) approaches because they are slower than the Filter approaches. Hence `Filter methods` ans is correct.

Embedded methods

This approach is a hybrid of filter and wrapper methods and is implemented by algorithms that have their own built-in feature selection methods. These methods are thus `embedded` in the algorithm either as its normal or extended functionality.

Embedded method is a type of filter method because:

This procedure of feature selection can be understood as adding a penalty to reduce the degree of overfitting. This results in weights of features become very small(or 0), therefore filtering out unnecessary features.

Embedded method is a type of wrapper method because:

In this approach, feature selection is performed during the process of training of the model itself and therefore is specific to the learning algorithms.

Embedded methods usually achieve high accuracy characteristic to wrapper methods and high-efficiency characteristic to filter methods

Most commonly used embedded feature selection methods are regularization methods namely LASSO and RIDGE.

Both types of regularization have already been covered extensively in the `Advanced Linear Regression` module.

Just to refresh,

Lasso (L1): It stands for *Least Absolute Shrinkage and Selection Operator* and adds **absolute value of magnitude of coefficient** as penalty term to the loss function. Mathematically, the new regularized cost function becomes:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2 + \lambda \sum_{i=1}^n |\theta_i|$$



Ridge (L2): Ridge regression adds **squared magnitude of coefficient** as penalty term to the loss function. The new cost function becomes:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2 + \lambda \sum_{i=1}^n \theta_i^2$$

In both the above techniques of feature selection, the penalty term helps regularise the feature coefficients and therefore automatically eliminate the unnecessary features and select the relevant ones while building the model itself.

Implementing it is the same as implementing an ML model because feature selection happens internally.

Let's now apply the above regularization techniques in our ames dataset problem.

| QUESTIONS | YOUR ANSWER | CORRECT ANSWER |
|---|--|---|
| <p>Suppose we fit "Lasso Regression" to a data set, which has 100 features (X_1, X_2, \dots, X_{100}). Now, we rescale one of these feature by multiplying with 10</p> <p>1. (say that feature is X_1), and then refit Lasso regression with the same regularization parameter.</p> <p>Now, which of the following option will be correct?</p> <p><u>Explanation:</u> Big feature values \Rightarrow smaller coefficients \Rightarrow less lasso penalty \Rightarrow more likely to have be kept</p> | <p> Can't say</p> | <p>It is more likely for X_1 to be included in the model</p> |
| <p>2. Which of the following is true about "Ridge" or "Lasso" regression methods in case of feature selection?</p> <p><u>Explanation:</u> The coefficients of some of the features are converted to zero eliminating the feature from further use in the model, so lasso can be used for feature selection.</p> | <p> Both use subset selection of features</p> | <p>Lasso regression uses subset selection of features</p> |

PCA

Another closely related topic with Feature Selection is Feature Extraction.

Feature Extraction refers to the method of reducing the known variables of the data into a lesser number of principal variables holding the same amount of information.

Though it sounds very similar to Feature Selection(because both are just different techniques of dimensionality reduction), one thing to keep in mind is that in this process we are aiming to capture the variance of the data and nothing else.

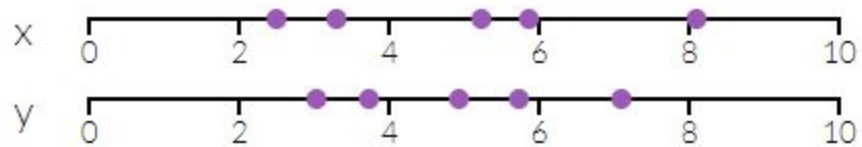
One of the popular Dimensionality Reduction techniques is Principal Component Analysis(PCA)

Intuition

Consider the following plot:

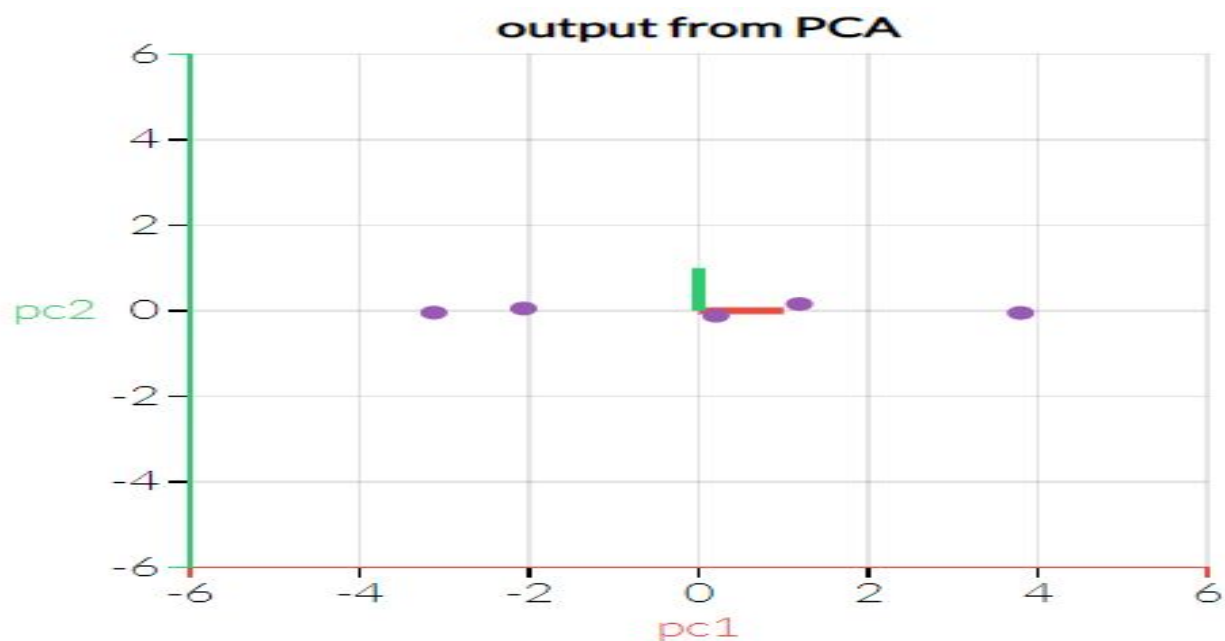


It has the following x and y axis configuration:

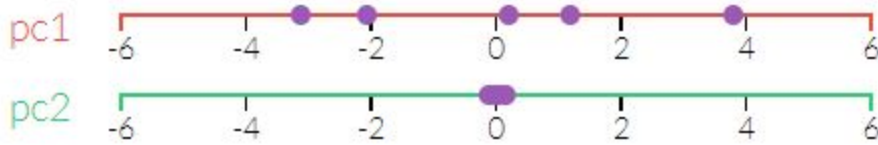


It can be intuitively seen that in the above diagram direction pointed by the 'red' line looks more important than the 'green'. (The line made in the direction pointed by 'red' is a better fit for the data points)

If we transform our x and y axes to red and green arrows, we get the following :



If we refer to red as principal component 1 and green as principal component 2, we get the following configuration:



It can be clearly seen that by dropping pc2, we don't lose many variations present in the data.

Similar to the above example, for data having multiple features, we can identify which 'directions' are important & explain the most variance and in effect drop the 'directions' which are least important.

Definition

Principal Component Analysis or PCA is the method of transforming original variables into a new set of variables such that they are orthogonal (and hence linearly independent) and then ranked according to the variance of data among them. These newly extracted variables are called Principal Components.

Principal components are extracted in such a way that the first principal component explains maximum variance in the dataset.

Second principal component(uncorrelated to the first) tries to explain the remaining variance(not explained by the first).

Third principal component explains the variance not explained by first and second and so on.

Working of PCA

To understand the working of PCA, it's important for you to have a knowledge of eigenvalues and eigenvectors.

Following are its steps:

Step 1: Calculate the covariance of matrix X (Feature matrix).

Step 2: Calculate eigenvectors and corresponding eigenvalues of X .

Step 3: Sort the eigenvectors according to their eigenvalue (in decreasing order).

Step 4: Use Eigenvectors corresponding to the (k)largest eigenvalues to reconstruct a large fraction of variance of the original data.

Let's try to understand the steps better using an example.

Consider a sample of our IOWA housing dataset:

It has 80 features and one target variable.

Before the first step, let's scale the features.

Q: Why do we need to scale it?

A: The end result of PCA is to calculate a new projection of your data set.

As in many other multivariate procedures, one needs to ensure that no extra weight is given to the “larger” variables which otherwise will lead to biased outcomes.

For eg: In the event we decide to change the scale of one of the features from metres to cm, the resulting variance of the feature would be 100 times more than it previously was. This would result in that feature playing a major role in deciding the first PC since the PCA algorithm would be biased towards this new feature, in order to maximize its variance.

Scaling the data, all variables will have the same standard deviation, thus none of the variables will have any bias and PCA can calculate the relevant axis.

After scaling, our first row of feature matrix will look something like:

```
[[-0.38780276 -0.88641057 -0.06590224 -1.73003297  5.61934547  0.07088812
  0.02132492 -1.38156245  0.34076569  0.          -1.06331739 -0.17586311
  .....
  0.09853293  0.5514855  -0.14023183 -0.10655645 -1.24153445 -0.60510586
  0.33428219 -1.43182551]
.....
.....
]]
```

Converting the above matrix into a covariance matrix, we get,

```
Covariance matrix
[[ 1.00502513  0.0461804 -0.08984672 ... -0.02916568  0.01314686
 -0.08965875]
.....
[-0.08965875  0.05498568 -0.0809293 ... -0.07523145  0.31478126
 1.00502513]]
```

Finding the eigenvalues and eigenvectors of the above matrix and sorting them in descending order we get,

```
Eigenvalues
[ 9.91860031e+00  4.92366237e+00  3.78092267e+00  3.56179086e+00
 3.39293240e+00  2.95161032e+00  2.74565913e+00  2.33694130e+00
 .....
 9.30686007e-02  9.62727861e-02  1.13943247e-01  1.30156014e-01
 1.34252254e-01 -6.77986291e-16 -7.78746845e-16  0.00000000e+00]
```

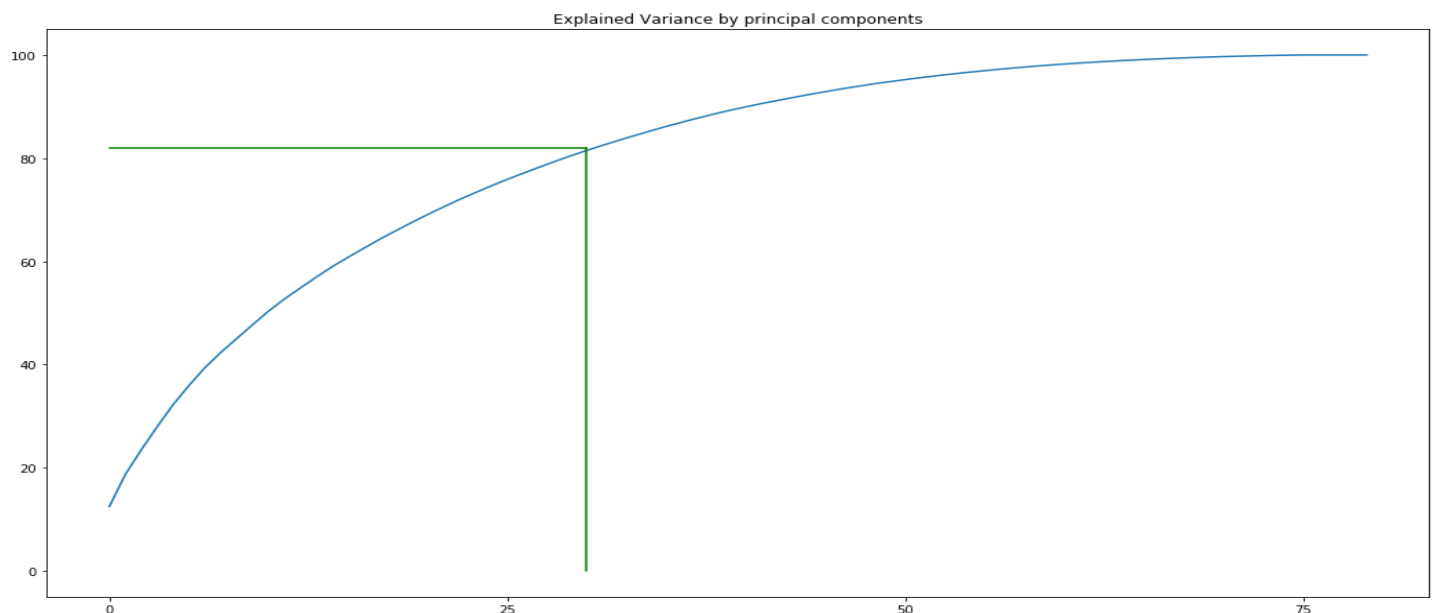
Note: PCA implementations perform a Singular Vector Decomposition instead of eigendecomposition of the covariance to improve the computational efficiency.

The V^T in the SVD formula $X = USV^T$ is nothing but the above eigenvectors matrix(Why?)

After sorting the eigenpairs, we need to decide "how many principal components are we going to choose for our new feature subspace?". We do this using "explained variance," which can be calculated from the eigenvalues.

The explained variance tells us how much information (variance) can be attributed to each of the principal components.

Following is the plot for the same:



In the plot above we can observe that most of the variance (More than 80%) can be explained by the first 30 principal components alone. While the remaining components(50) have very less information to give and therefore can be dropped.

Our projection matrix will, therefore, look like

Projection Matrix $M([80 \times 30])$:

```
[ [ 0.00448071 -0.01350817  0.12326238 ...  0.19900511  0.05865323
   0.05688455]
 [ -0.01759388  0.07024879  0.19174906 ...  0.02688533 -0.06747264
  -0.01217759]
 .....
 [ -0.01299174 -0.01049473  0.03013888 ... -0.03344006  0.00668279
  -0.12497142]
 [  0.06469133  0.10257877 -0.03185315 ... -0.12730551  0.15506105
  -0.12573368]]
```


Transforming our original matrix X using projection Matrix M, we get:

```
Y=X.M

=[ [ 2.19814018  0.34126792 -1.60125569 ...  0.78203228 -0.21044354
    -0.62360616]
  [ 0.35239026 -2.05793391  1.12301188 ...  1.53015364  0.24685173
    -0.58152253]
  .....
  [-2.51897468 -3.00342066  1.61685692 ... -0.96805527 -0.28230475
    0.19131301]
  [-0.68028629 -2.64923105  1.59040403 ... -0.38116799 -1.86049732
    -0.12103677]]
```

```
Dimensions of Y: [200 x 30]
```

Scikit implementation

Though we have learned the step by step process, python has a simpler implementation in scikit-learn with all the steps internally taken care of.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

df = pd.read_csv('../data/Cleaned_Data.csv')
df_new=df.sample(200,random_state=0)

X = df.drop(['SalePrice'],1)

print("\nFirst two rows of X matrix:")
print(X.iloc[0:2,])

scaler=StandardScaler()

X_scaled = scaler.fit_transform(X)
print("\nFirst two rows of scaled X matrix:")
print(X_scaled[0:2,])

pca = PCA(n_components=30)
X_pca = pca.fit_transform(X_scaled)
print("\nFirst two rows of pca transformed X matrix:")
print(X_pca[0:2,])
```

Output

First two rows of X matrix:

| | MSSubClass | MSZoning | LotFrontage | LotArea | Street | Alley | LotShape | \ |
|---|------------|----------|-------------|---------|--------|-------|----------|---|
| 0 | 60 | 3 | 65.0 | 8450 | 1 | 1 | | 3 |
| 1 | 20 | 3 | 80.0 | 9600 | 1 | 1 | | 3 |

| | LandContour | Utilities | ... | ScreenPorch | PoolArea | PoolQC | \ |
|--|-------------|-----------|-----|-------------|----------|--------|---|
|--|-------------|-----------|-----|-------------|----------|--------|---|

| | | | | | | |
|---|---|---|-----|---|---|---|
| 0 | 3 | 0 | ... | 0 | 0 | 3 |
| 1 | 3 | 0 | ... | 0 | 0 | 3 |

| | Fence | MiscFeature | MiscVal | MoSold | YrSold | SaleType | SaleCondition |
|---|-------|-------------|---------|--------|--------|----------|---------------|
| 0 | 4 | 1 | 0 | 2 | 2008 | 8 | 4 |
| 1 | 4 | 1 | 0 | 5 | 2007 | 8 | 4 |

[2 rows x 80 columns]

First two rows of scaled X matrix:

```
[[-1.73086488  0.07337496 -0.04553194  0.2128772  -0.20714171  0.06423821
  0.02469891  0.75073056  0.31466687 -0.02618016  0.60466978 -0.22571613
  .....
  0.06330477  0.45744736 -0.1859753  -0.08768781 -1.5991111  0.13877749
  0.31386709  0.2085023 ]
[-1.7284922  -0.87256276 -0.04553194  0.64574726 -0.09188637  0.06423821
  0.02469891  0.75073056  0.31466687 -0.02618016 -0.62831608 -0.22571613
  .....
  0.06330477  0.45744736 -0.1859753  -0.08768781 -0.48911005 -0.61443862
  0.31386709  0.2085023 ]]
```

First two rows of pca transformed X matrix:

```
[[ 2.19814018  0.34126792 -1.60125569 ...  0.78203228 -0.21044354
 -0.62360616]
 [ 0.35239026 -2.05793391  1.12301188 ...  1.53015364  0.24685173
 -0.58152253]
 .....
 [-2.51897468 -3.00342066  1.61685692 ... -0.96805527 -0.28230475
  0.19131301]
 [-0.68028629 -2.64923105  1.59040403 ... -0.38116799 -1.86049732
 -0.12103677]]
```

Though PCA does help to perform dimensionality reduction as rule of thumb consider PCA only if :

1. You want to reduce the data dimensions but aren't able to identify variables to remove.
2. You are comfortable making the independent variables less interpretable.

PCA task

In this task we will try to use PCA for dimensionality reduction on our data and then implement linear regression.

- Store all the features of 'ames' in a variable called `x`
- Store the target variable (SalePrice) of 'ames' in a variable called `y`
- Split 'x' and 'y' into `x_train, x_test, y_train, y_test` using `train_test_split()` function. Use `test_size = 0.3` and `random_state = 0`
- Initialise a "`StandardScaler()`" object and store it to a variable called '`scaler`'.
- Scale '`x_train`' using the `fit_transform()` method of '`scaler`' and store the scaled output in '`x_train_scaled`'

- Scale 'X_test' as well using the `transform()` method of 'scaler' and store the scaled output in 'X_test_scaled'
- Initialise a "PCA()" object with the parameter `n_components=35, random_state=0` and store it to a variable called 'pca'.
- Transform 'X_train_scaled' using the `fit_transform()` method of 'pca' and store the scaled output in 'X_train_pca'
- Transform 'X_test_scaled' as well using the `transform()` method of 'pca' and store the scaled output in 'X_test_pca'
- Initialise a linear regression model with `LinearRegression()` and save it to a variable called 'model'.
- Fit the model on the training data 'X_train_pca' and 'y_train' using the '`fit()`' method of 'model'.
- Find out the r^2 score between 'X_test_pca' and 'y_test' using the '`score()`' method of 'model' and store it in a variable called 'pca_score'.

Code:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Code starts here
X=ames.drop(['SalePrice'],1)
y=ames['SalePrice'].copy()

X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.3, random_state=0)
#Initialising standard scaler
scaler=StandardScaler()

#Scaling the features
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#Initialising PCA
pca = PCA(n_components=35, random_state=0)



#Transforming the features
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca=pca.transform(X_test_scaled)

#Initialising the model
model=LinearRegression()

#Fitting the model
model.fit(X_train_pca,y_train)

#Scoring the model
```

```
pca_score=model.score(X_test_pca,y_test)
print(pca_score)
```

| QUESTIONS | | YOUR ANSWER | CORRECT ANSWER |
|--|---|--|---|
| <p>1. PCA method is a feature selection method ?</p> <p><u>Explanation:</u> PCA is a dimensionality reduction method.</p> |  | True | False |
| <p>2. Which of the following are good reasons to implement PCA?</p> <p><u>Explanation:</u> PCA represents the original data in less number of dimensions speeding up the preocess as well it uses less disk space.</p> |  | <p>As a method to speed up a learning algorithm that lags with high dimensional data</p> <p>As a method to implement regularisation by reducing features and preventing overfitting.</p> | <p>To visualise high dimensional data(by choosing k=2 or k=3)</p> <p>As a method to speed up a learning algorithm that lags with high dimensional data</p> <p>To compress the data to take up less disk space</p> |

Feature Selection Checklist

Let's look at the table storing the results of the different feature selection techniques(and PCA) that we applied on our AMES dataset:

| Feature Selection Method | No. of features selected | r2 Score |
|--------------------------|--------------------------|----------|
| Linear Regression | 80 | 0.66 |
| Pearson Correlation | 13 | 0.72 |
| Chi-Square | 60 | 0.75 |
| Anova | 60 | 0.75 |
| RFE | 30 | 0.76 |
| LASSO | Internal Selection | 0.66 |

| | | |
|-------|--------------------|------|
| RIDGE | Internal Selection | 0.66 |
| PCA | 35 | 0.76 |

Does it mean that RFE or PCA are the best methods for feature selection?

The reason there exists so many methods is that there is no single feature selection method that will give the best results across all Machine Learning problems.

To make the most of the methods mentioned here, the user should know what kind will work best for him given the domain, data and the problem he is trying to solve.

If at some point, you are too confused to decide what to use, following is a checklist you can refer to, to help you ease up the feature selection process:

Feature Selection Checklist:

-
- Do you have domain knowledge? If yes, construct a better set of “ad hoc” features that will provide more information gain.
 - Are your features all of the same scales? If no, consider normalizing them.
 - Do you suspect interdependence of features? If yes, expand your feature set by constructing products of features, as much as your computer resources allow
 - Do you need to reduce the count of input variables (e.g. for cost, speed or data understanding reasons)? If no, then construct disjunctive features or weighted sums of features
 - Do you need to assess features individually (e.g. to understand their influence on the system or because their number is so large that you need to do the first filtering)? If yes, use a variable ranking method; else, do it anyway to get baseline results.
 - Do you need a predictor? If no, stop.
 - Do you suspect your data is “dirty” (has a few meaningless input patterns and/or noisy outputs or wrong class labels)? If yes, detect the outlier examples using the top ranking variables as representation; check and/or discard them.
 - Do you know what to try first? If not, use a linear predictor. Construct a sequence of predictors of same nature using increasing subsets of features. Can you match or improve performance with a smaller subset? If yes, try a non-linear predictor with that subset.
 - Do you have new ideas, time, computational resources, and enough examples? If yes, compare several feature selection methods, including your new idea, correlation coefficients, backward selection and embedded methods. Use linear and non-linear predictors. Select the best approach with model selection.
 - Do you want a stable solution (to improve performance and/or understanding)? If yes, subsample your data and redo your analysis for several “subsets”
-

The above checklist is made by Isabelle Guyon and Andre Elisseeff the authors of [“An Introduction to Variable and Feature Selection” \(PDF\)](#). You can check out the pdf link to learn more about it

- If the input features are on very different scales, PCA
1. automatically takes care of it and performs dimensionality reduction.



False

False

Explanation:

The PCA calculates a new projection of your data set. And the new axis are based on the standard deviation of your variables. So a variable with a high standard deviation will have a higher weight for the calculation of axis than a variable with a low standard deviation. If you normalize your data, all variables have the same standard deviation, thus all variables have the same weight and your PCA calculates relevant axis.

- We are building a ML model on a dataset with 5000 features and more than a million observations. Now,
2. we want to train a model on this dataset but we are facing a challenge with this big size data. What are the steps will you consider to train model efficiently?



F. All of the above

F. All of the above

Explanation:

Processing a high dimensional data on a limited memory machine is a strenuous task, Following are the methods you can use to tackle such situation:

We can randomly sample the data set. This means, we can create a smaller data set, let's say, having 1000 variables and 300000 rows and do the computations. Using online learning algorithms like those present in Vowpal Wabbit is a possible option. Also, we can use PCA and pick the components which can explain the maximum variance in the data set.

- You should implement feature selection methods
3. before the data cleaning methods because that would result in lesser features to clean and preprocess



True

False

Explanation:

Preprocessing such as handling missing values, noise reduction and data normalization should be done before to convert the data in an appropriate format for feature selection.

- A regression model is suffering with multicollinearity.
4. How will you deal with this situation without losing much information?



Instead of removing both variables, we can remove only one variable.
We can calculate VIF (variance inflation factor) to check the presence of multicollinearity and take action accordingly.
Removing correlated variables might lead to loss of information.
In order to retain those variables, we can use penalized regression models like ridge or lasso regression.

Instead of removing both variables, we can remove only one variable.
We can calculate VIF (variance inflation factor) to check the presence of multicollinearity and take action accordingly.
Removing correlated variables might lead to loss of information.
In order to retain those variables, we can use penalized regression models like ridge or lasso regression.

Explanation:

To check multicollinearity, we can create a correlation matrix to identify & remove variables having correlation above 75% (deciding a threshold is subjective). In addition, we can use calculate VIF (variance inflation factor) to check the presence of multicollinearity. VIF value ≤ 4 suggests no multicollinearity whereas a value of ≥ 10 implies serious multicollinearity. Also, we can use tolerance as an indicator of multicollinearity.

But, removing correlated variables might lead to loss of information. In order to retain those variables, we can use penalized regression models like ridge or lasso regression. Also, we can add some random noise in correlated variable so that the variables become different from each other. But, adding noise might affect the prediction accuracy, hence this approach should be carefully used.

5. F-score is defined as



Variance within the samples/
Variance between the samples

Variance between the samples/
Variance within the samples

Explanation:

F-score is the harmonic mean between precision and recall.

Which is that algorithm which begins with having no features in the model and continue including features

6. which best enhances the model till adding another model would not enhance the performance of the model.



Forward selection

Forward selection

Explanation:

Yes, forward feature selection starts with the evaluation of each individual feature, and selects that which results in the best performing selected algorithm model.

That depends entirely on the defined evaluation criteria (AUC, prediction accuracy, RMSE, etc.). Next, all possible combinations of the that selected feature and a subsequent feature are evaluated, and a second feature is selected, and so on, until the required predefined number of features is selected.

A psychologist is interested in the relationship between job satisfaction and stress. Within a large corporation, the psychologist asked a random sample of workers two questions. The first question asked workers to rate their overall satisfaction with their job on a scale from 1 to 50. The second question asked the workers to rate their stress level during the past week on a scale from 1 to 50. What type of statistical test best assesses the relationship between job satisfaction and level of stress?

7.



One-way ANOVA

Correlation

Explanation:

The null hypothesis we are testing is that there is no linear relationship between two variables in the population from which the sample is selected.

The two measures are both continuous, and the participants in our study each provide both measures. We can test for a linear relationship between these two measures with correlation or regression.

As a first step, we should plot the data to assess whether a linear relationship is an appropriate model. Correlation is sensitive to only a linear relationship, so even if the correlation is zero there could be a nonlinear relationship, as with a U shaped curve. For the test of statistical significance to be appropriate we must satisfy assumptions for the test, including a normal distribution of errors around the linear model with equal variance for one of the variables for all values of the other variable. It would be useful to compute a confidence interval, to provide information on the precision of the sample correlation as an estimate of the population correlation.

8. Which of the following techniques would perform better for reducing dimensions of a data set?



A. Removing columns which have too many missing values

A. Removing columns which have too many missing values

Explanation:

Missing values provide no useful information and the machine learning model cannot handle missing values

I have 4 variables in the dataset such as – A, B, C & D. I have performed the following actions:

Step 1: Using the above variables, I have created two more variables, namely $E = A + 3 * B$ and $F = B + 5 * C$

9. + D.



False

True

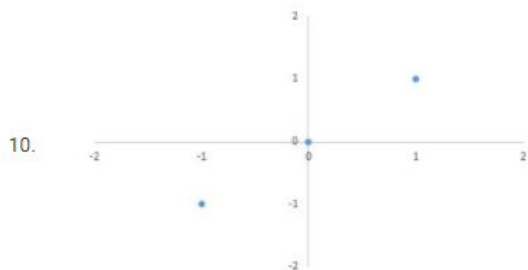
Step 2: Then using only the variables E and F I have built a Random Forest model.

Could the steps performed above represent a dimensionality reduction method?

Explanation:

Feature extraction is for creating a new, smaller set of features that stills captures most of the useful information using the existing features. So, this can also be used as dimensionality reduction method.

Consider 3 data points in the 2-d space: $(-1, -1)$, $(0,0)$, $(1,1)$.



$([-\sqrt{2}/2, \sqrt{2}/2])$

$[\sqrt{2}/2, \sqrt{2}/2]$

$([-\sqrt{2}/2, \sqrt{2}/2])$

What will be the first principal component for this data?

Exploratory Data Analysis

Let's get a little bit deeper dive into exploring the individual features via graphical methods.

Instructions

- Store all the column names in the variable called `cols`
- Store the length of a number of features (excluding target variable) in the variable `size` respectively.
- Create variable `x` containing the target column.
- Create variable `y` containing all the columns excluding target.
- Plot the violin plot for all attributes. Hint: You will have to use a loop with the `size` variable created above.

Observation

- Elevation is has a separate distribution for most classes. Highly correlated with the target and hence an important attribute
- Aspect contains a couple of normal distribution for several classes
- Horizontal distance to road and hydrology have a similar distribution
- Lots of 0s in the vertical distance to hydrology

- Wilderness_Area3 gives no class distinction. As values are not present, others give some scope to distinguish
- Soil_Type, 1,5,8,9,12,14,18-22, 25-30 and 35-40 offer class distinction as values are not present for many classes

```
# We will visualize all the attributes using Violin Plot - a combination of box and density plots
import seaborn as sns
from matplotlib import pyplot as plt

#names of all the attributes
cols = dataset.columns

#number of attributes (exclude target)
size = len(cols)-1

#x-axis has target attribute to distinguish between classes
x = cols[size]

#y-axis shows values of an attribute
y = cols[0:size]

#Plot violin for all attributes
for i in range(0,size):
    sns.violinplot(data=dataset,x=x,y=y[i])
    plt.show()
```

Closer look towards relationship between different variables

The next step comprises of looking at the relationship across different variable and further looking at variable who have a strong relationship with each other which gives us feedback as to which values to be considered for the model building.

Instructions

- 1 We can observe that the first 10 columns contain all continuous features
 - 1 Select the first 10 features and store them in the variable `subset_train`
 - 2 Calculate the Pearson correlation between these 10 features and store it in variable `data_corr`
 - 3 Plot a heatmap of `data_corr` using seaborn
 - 4 List the correlation pairs from `data_corr` using `.unstack()` and `.sort_values(kind='quicksort')` & store it in variable `correlation`
 - 5 From `correlation`, using slicing, select the values above `upper_threshold` and below `lower_threshold` but not equal to 1 and save it as `corr_var_list`.
 - 6 We neglect correlation value equal to 1 since this indicates a correlation of a feature with itself.

Skills Covered:

```
1 import numpy
2 upper_threshold = 0.5
3 lower_threshold = -0.5
4 # Code Starts Here
5 # create a subset of dataframe with only the first 10 features
6 subset_train = dataset.iloc[:, :10]
7 # Calculate the Pearson correlation
8 data_corr = subset_train.corr()
9 # Plot a heatmap
10 f, ax = plt.subplots(figsize = (10,8))
11 sns.heatmap(data_corr,vmax=0.8,square=True);
12 # List the correlation pairs
13 correlation = data_corr.unstack().sort_values(kind='quicksort')
14 # Select the highest correlation pairs using slicing
15 corr_var_list = correlation[((correlation>upper_threshold) | (correlation<lower_threshold)) & (correlation!=1)]
16 print(corr_var_list)
```

Congrats! You have successfully measured the correlation between variables. We can see that the feature `Hillshade_3pm` and `Hillshade_9am` are highly correlated with each other.

CONTINUE

TRY IT

RESULT

| | | |
|---------------------------------|---------------------------------|-----------|
| Hillshade_3pm | Hillshade_9am | -0.779965 |
| Hillshade_9am | Hillshade_3pm | -0.779965 |
| Hillshade_Noon | Slope | -0.612613 |
| Slope | Hillshade_Noon | -0.612613 |
| Aspect | Hillshade_9am | -0.593997 |
| Hillshade_9am | Aspect | -0.593997 |
| Horizontal_Distance_To_Roadways | Elevation | 0.578659 |
| Elevation | Horizontal_Distance_To_Roadways | 0.578659 |
| Hillshade_Noon | Hillshade_3pm | 0.614526 |
| Hillshade_3pm | Hillshade_Noon | 0.614526 |
| | Aspect | 0.635022 |

Data Cleaning , Data preparation ...getting towards feature selection

This step is a part of data preprocessing which involves careful observation at the data in hand and what modification needs to be done to get cleaner data. So what are waiting for, get your hands dirty to dwell deep into the process of data cleaning and preparation.

Instructions

- Split the dataset in features and target and save the same in variables `x` and `y` respectively
- Split the data into chunks of 0.2 by using `cross_validation` class on `x` and `y` and store it in `x_train`, `x_test`, `y_train`, `y_test` and set the `random_state` to 0.
- Instantiate `StandardScaler()` to a variable called `scaler`. Perform the standard scaling on the continuous data on `x_train` and `x_test` and store it in `x_train_temp` and `x_test_temp`.
- Concatenate the scaled continuous data to categorical data above (`x_train_temp` and `x_train`) and store it in `x_train1` and similarly concatenate (`x_test_temp` and `x_test`) and store it in `x_test1`.
- Create a dataframe of rescaled data `x_train1` which consists of columns and indexes set as of unscaled `x_train` and store in the variable named as `scaled_features_train_df`
- Create a dataframe of rescaled data `x_test1` which consists of columns and indexes set as of unscaled `x_test` and store in the variable named as `scaled_features_test_df`

```
#Import libraries

from sklearn import cross_validation

from sklearn.preprocessing import StandardScaler

# Identify the unnecessary columns and remove it

dataset.drop(columns=['Soil_Type7', 'Soil_Type15'], inplace=True)

r,c = dataset.shape

X = dataset.iloc[:, :-1]

Y = dataset.iloc[:, -1]

# Scales are not the same for all variables. Hence, rescaling and standardization may
be necessary for some algorithm to be applied on it.

X_train, X_test, Y_train, Y_test = cross_validation.train_test_split(X, Y,
test_size=0.2, random_state=0)

#Standardized

scaler = StandardScaler()

#Apply transform only for continuous data

X_train_temp = scaler.fit_transform(X_train.iloc[:, :10])
```

```
X_test_temp = scaler.transform(X_test.iloc[:,10:])

#Concatenate scaled continuous data and categorical

X_train1 = numpy.concatenate((X_train_temp,X_train.iloc[:,10:c-1]),axis=1)

X_test1 = numpy.concatenate((X_test_temp,X_test.iloc[:,10:c-1]),axis=1)

scaled_features_train_df = pd.DataFrame(X_train1, index=X_train.index,
columns=X_train.columns)

scaled_features_test_df = pd.DataFrame(X_test1, index=X_test.index,
columns=X_test.columns)
```

Feature Selection using Select percentile

Instructions :

- Initialise a `SelectPercentile()` with the parameters `score_func=f_classif` & `percentile=90` & store it in variable `skb`
- Fit & transform `skb` on `X_train1` and `y_train` and store the result in `predictors`.
- Access the scores generated by the above model using `.scores_` attribute and store them as a list in the variable `scores`
- Store all the feature names in a variable `Features`
- Create a dataframe named `dataframe` having `Features` and `scores` as the two columns.
- Sort `dataframe` in descending order.
- Select the features that fall under top 90 percentile and store them as a list in `top_k_predictors`
- Print `top_k_predictors`

Skills Covered:

Data Wrangling

```
1 from sklearn.feature_selection import SelectPercentile
2 from sklearn.feature_selection import f_classif
3 # Write your solution here:
4 # Code starts here
5 skb = SelectPercentile(score_func=f_classif,percentile=90)
6 predictors = skb.fit_transform(X_train1, y_train)
7 scores = list(skb.scores_)
8 Features = scaled_features_train_df.columns
9
10 dataframe = pd.DataFrame({'Features':Features
11 , 'Scores':scores})
12 dataframe=dataframe.sort_values(by='Scores',ascending=False)
13 top_k_predictors = list(dataframe['Features'][:predictors.shape[1]])
14 print(top_k_predictors)
```

Congrats! You have successfully performed the feature selection. You can see that we got 11 top predictors that fall under top 90 percentile.

CONTINUE

TRY IT

SUBMIT

OUTPUT

RESULT

```
['Elevation', 'Wilderness_Area4', 'Horizontal_Distance_To_Roadways', 'Soil_Type10', 'Soil_Type3',
'Wilderness_Area1', 'Soil_Type38', 'Horizontal_Distance_To_Fire_Points', 'Soil_Type39',
'Wilderness_Area3', 'Soil_Type40', 'Horizontal_Distance_To_Hydrology', 'Hillshade_9am', 'Soil_Type30',
'Soil_Type29', 'Slope', 'Soil_Type4', 'Soil_Type22', 'Soil_Type17', 'Soil_Type13', 'Soil_Type12',
'Soil_Type23', 'Hillshade_3pm', 'Soil_Type6', 'Wilderness_Area2', 'Soil_Type2', 'Soil_Type32',
'Hillshade_Noon', 'Soil_Type14', 'Soil_Type35', 'Soil_Type1', 'Vertical_Distance_To_Hydrology',
'Soil_Type33', 'Soil_Type24', 'Soil_Type31', 'Soil_Type11', 'Aspect', 'Soil_Type5', 'Soil_Type18',
'Soil_Type37', 'Soil_Type20', 'Soil_Type16', 'Soil_Type26', 'Soil_Type19', 'Soil_Type21', 'Soil_Type9']
{
  " " " " " "
}
```

Effect of feature selection on model prediction

We will see in this task that how feature selection affects the model prediction and performance whether it increases it as compared to the baseline model considering all attributes or it decreases.

Instructions

- define two variables as `clf` and `clf1` initializing the one vs rest classifier with logistic regression as the model as input.
- Create a variable `model_fit_all_features` which fits the classifier `clf1` onto `X_train` and `Y_train`
- Predict the values with the above model fitted on `X_test` and store it in `predictions_all_features`.
- Get the accuracy score for the model above by comparing `Y_test` and `predictions_all_features` and store it in the variable named as `score_all_features`.

- Create a variable `model_fit_top_features` which fits the classifier `clf` onto `scaled_features_train_df` which contains only `top_k_predictors` thrown out by select percentile and `Y_train`
- Predict the values with the above model fitted on `scaled_features_train_df` which contains only top predictors thrown out by select percentile and store it in `predictions_top_features`.
- Get the accuracy score for the model above by comparing `Y_test` and `predictions_top_features` and store it in `score_top_features`.

```
from sklearn.multiclass import OneVsRestClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
precision_score

clf = OneVsRestClassifier(LogisticRegression())

clf1 = OneVsRestClassifier(LogisticRegression())

model_fit_all_features = clf1.fit(X_train, Y_train)

predictions_all_features = model_fit_all_features.predict(X_test)

score_all_features = accuracy_score(Y_test, predictions_all_features)

print(score_all_features)

model_fit_top_features = clf.fit(scaled_features_train_df[top_k_predictors], Y_train)

predictions_top_features =
model_fit_top_features.predict(scaled_features_test_df[top_k_predictors])

score_top_features = accuracy_score(Y_test, predictions_top_features)

print(score_top_features)
```

We can see that we are able to achieve the same accuracy by selecting 90% of the features i.e we have been able to reduce 6 features.

