Feature Selection Based on Mutual Information(Entropy)Gain for Classification & Regression.

Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in. Having irrelevant features in your data can decrease the accuracy of the models and make your model learn based on irrelevant features.

Irrelevant or partially relevant features can negatively impact model performance.

What is Mutual Information ?

The elimination process aims to reduce the size of the input feature set and at the same time to retain the class Discriminatory information for classification problems.

Mutual information(MI) is a measure of the amount information between two random variables in symmetric and non-negetive. and,it could be zero if and only if variable are independed. Mutual information is calculated between two variables and measures the reduction in uncertainty for one variable given a known value of the other variable. A quantity called mutual information measures the amount of information one can obtain from one random variable given another.

Feature filtering method that is univarient method does not give ganrantee to get the better accuracy. Since in univarient method features importants are calculated indivisualy instaed of in a group. So,sometimes what happen top best performing features doesn't perform well as grouping of some other method.

Accuracy methods features importance are calculated individually instead of in a group so sometimes what happened at top 10 best-performing features doesn't perform well as grouping of some other methods so what happens in the filtering methods sometimes we enter into by selecting suboptimal features but filtering are univariate methods are quite fast and that those can be used as a screening and sometimes you might get a better accuracy but most of the time you would get a better the training time that introduced training time after selecting some particular feature set so by keeping those things in mind you should read this lessons and one more thing in for the lessons I was talking about the wrapper method as well as the imported methods which are better than the filter methods but whose methods are very costly go ahead and I'll tell you to take you through this lessons for the first will be talking about are the classification problem in which shall be working on sent Android data set then first I'll remove the constant and duplicate features and then after that I will remove all calculate the mutual information and then based on the mutual information Sal select top 10 top 10 percentage and percentile of the features which is the 23 features and then I build model and then finally I'll train it and after that I will work on regression problem which is the best and data regression problem using Boston data set and using mutual information Sal calculate the in mutual information gain on the individual features and then finally I select top 9 features which are to be stopped 9 features after selecting the staff top 9 features and I calculate the performance compare the performance of the selected features and the overall features so this is all in this lesson.

Classification Problem Using Mutual Information Gain.

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

In [67]:
```
from sklearn.feature_selection import mutual_info_classif,mutual_info_regression,SelectPercentile,SelectKBest
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier,RandomForestRegressor
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score,mean_squared_error,r2_score
```

In [5]:
```
data = pd.read_csv('titanic.csv')
data.head()
```

Out[5]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [6]:
```
data.dropna(inplace=True)
```

In [9]:
```
label = LabelEncoder()
data['sex'] = label.fit_transform(data['sex'])
data['embarked'] = label.fit_transform(data['embarked'])
data['class'] = label.fit_transform(data['embarked'])
data['who']=label.fit_transform(data['who'])
data['adult_male']=label.fit_transform(data['adult_male'])
data['embark_town']=label.fit_transform(data['embark_town'])
data['alive']=label.fit_transform(data['alive'])
data['alone'] = label.fit_transform(data['alone'])
```

In [10]:
```
del data['deck']
```

In [11]:
```
x = data.drop(labels='alive',axis=1)
y= data['alive']

x.shape,y.shape
```

Out[11]:
```
((182, 13), (182,))
```

In [12]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8,random_state=0,stratify=y)
```

In [13]:
```
x_train.shape,x_test.shape
```

Out[13]:
```
((145, 13), (37, 13))
```

Remove the constant,quaisi constant,duplicated

In [14]:
```
constant_filter = VarianceThreshold(threshold=0.01)
constant_filter.fit(x_train)
x_train_filter=constant_filter.transform(x_train)
x_test_filter= constant_filter.transform(x_test)

x_train_filter.shape,x_test_filter.shape
```

Out[14]:
```
((145, 13), (37, 13))
```

Dupllicated features Removal

In [15]:
```
x_train_T = x_train_filter.T
x_test_T  = x_test_filter.T
x_train_T = pd.DataFrame(x_train_T)
X_test_T = pd.DataFrame(x_test_T)
```

In [16]:
```
duplicated_features = x_train_T.duplicated()
```

In [17]:
```
keep_them = [not index for index in duplicated_features]
```

In [18]:
```
x_train_unique = x_train_T[keep_them].T
x_test_unique  = x_test_T[keep_them].T
```

In [19]:
```
x_train_unique.shape,x_test_unique.shape
```

Out[19]:
```
((145, 11), (37, 11))
```

calculated mutual information

In [20]:
```
mi = mutual_info_classif(x_train_unique,y_train)
len(mi)
```

Out[20]:
```
11
```

In [21]:
```
mi = pd.Series(mi)
mi.index = x_train_unique.columns
mi.sort_values(ascending= False,inplace = True)
```

If we see the graph here we can see that the few of the features are contributing the more information gain(individualy) than the others and only 60 to 70 % features are added the contribution in mutual inforamtion gain and those who are contributing the more inforation which are very important in this kind of classifiction. We are going to remove those features which minutly contributing any information individually so for that we are going to select some k best or the Selectpercentile to get the those who are contributing the more information than the others.

In [22]:
```
mi.plot.bar(figsize = (15,6))
```

Out[22]:
```
<AxesSubplot:>
```

In [36]:
```
sel = SelectPercentile(mutual_info_classif,percentile=90).fit(x_train_unique,y_train)
x_train_unique.columns[sel.get_support()]
```

Out[36]:
```
Int64Index([0, 1, 2, 3, 5, 6, 9, 10, 12], dtype='int64')
```

In [38]:
```
len(x_train_unique.columns[sel.get_support()])
```

Out[38]:
```
9
```

In [39]:
```
x_train_mi = sel.transform(x_train_unique)
x_test_mi = sel.transform(x_test_unique)

x_train_mi.shape,x_test_mi.shape
```

Out[39]:
```
((145, 9), (37, 9))
```

In [40]:
```
x_train_mi.shape,x_test_mi.shape
```

Out[40]:
```
((145, 9), (37, 9))
```

In [41]:
```
#build a classifier
```

In [42]:
```
def randomforest(x_train,x_test,y_train,y_test):
    clf = RandomForestClassifier(random_state=0,n_estimators=1000,n_jobs=-1)
    clf.fit(x_train,y_train)
    y_pred = clf.predict(x_test)
    print('Accuracy :',accuracy_score(y_test,y_pred))
```

In [43]:
```
%%time
randomforest(x_train_mi,x_test_mi,y_train,y_test)
```

```
Accuracy : 1.0
Wall time: 1.95 s
```

In [44]:
```
%%time
randomforest(x_train,x_test,y_train,y_test)
```

```
Accuracy : 1.0
Wall time: 2.03 s
```

From the above we can say that the features selection method does not make any impace of the accuracy but it impacct the compuational time from the we can say easily.

In [45]:
```
#this is it can be done with the help mutual information classifier
```

In [46]:
```
help(sel)
#this is how we can get the whole about it
```

```
Help on SelectPercentile in module sklearn.feature_selection._univariate_selection object:

class SelectPercentile(_BaseFilter)
 |  SelectPercentile(score_func=<function f_classif at 0x000002D3EA398550>, *, percentile=10)
 |  
 |  Select features according to a percentile of the highest scores.
 |  
 |  Read more in the :ref:`User Guide <univariate_feature_selection>`.
 |  
 |  Parameters
 |  ----------
 |  score_func : callable, default=f_classif
 |      Function taking two arrays X and y, and returning a pair of arrays
 |      (scores, pvalues) or a single array with scores.
 |      Default is f_classif (see below "See Also"). The default function only
 |      works with classification tasks.
 |  
 |      .. versionadded:: 0.18
 |  
 |  percentile : int, default=10
 |      Percent of features to keep.
 |  
 |  Attributes
 |  ----------
 |  scores_ : array-like of shape (n_features,)
 |      Scores of features.
 |  
 |  pvalues_ : array-like of shape (n_features,)
 |      p-values of feature scores, None if `score_func` returned only scores.
 |  
 |  Examples
 |  --------
 |  >>> from sklearn.datasets import load_digits
 |  >>> from sklearn.feature_selection import SelectPercentile, chi2
 |  >>> X, y = load_digits(return_X_y=True)
 |  >>> X.shape
 |  (1797, 64)
 |  >>> X_new = SelectPercentile(chi2, percentile=10).fit_transform(X, y)
 |  >>> X_new.shape
 |  (1797, 7)
 |  
 |  Notes
 |  -----
 |  Ties between features with equal scores will be broken in an unspecified
 |  way.
 |  
 |  See Also
 |  --------
 |  f_classif : ANOVA F-value between label/feature for classification tasks.
 |  mutual_info_classif : Mutual information for a discrete target.
 |  chi2 : Chi-squared stats of non-negative features for classification tasks.
 |  f_regression : F-value between label/feature for regression tasks.
 |  mutual_info_regression : Mutual information for a continuous target.
 |  SelectKBest : Select features based on the k highest scores.
 |  SelectFpr : Select features based on a false positive rate test.
 |  SelectFdr : Select features based on an estimated false discovery rate.
 |  SelectFwe : Select features based on family-wise error rate.
 |  GenericUnivariateSelect : Univariate feature selector with configurable
 |      mode.
 |  
 |  Method resolution order:
 |      SelectPercentile
 |      _BaseFilter
 |      sklearn.feature_selection._base.SelectorMixin
 |      sklearn.base.TransformerMixin
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |  
 |  Methods defined here:
 |  
 |  __init__(self, score_func=<function f_classif at 0x000002D3EA398550>, *, percentile=10)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |  
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |  
 |  __abstractmethods__ = frozenset()
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from _BaseFilter:
 |  
 |  fit(self, X, y)
 |      Run score function on (X, y) and get the appropriate features.
 |      
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          The training input samples.
 |      
 |      y : array-like of shape (n_samples,)
 |          The target values (class labels in classification, real numbers in
 |          regression).
 |      
 |      Returns
 |      -------
 |      self : object
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.feature_selection._base.SelectorMixin:
 |  
 |  get_support(self, indices=False)
 |      Get a mask, or integer index, of the features selected
 |      
 |      Parameters
 |      ----------
 |      indices : bool, default=False
 |          If True, the return value will be an array of integers, rather
 |          than a boolean mask.
 |      
 |      Returns
 |      -------
 |      support : array
 |          An index that selects the retained features from a feature vector.
 |          If `indices` is False, this is a boolean array of shape
 |          [# input features], in which an element is True iff its
 |          corresponding feature is selected for retention. If `indices` is
 |          True, this is an integer array of shape [# output features] whose
 |          values are indices into the input feature vector.
 |  
 |  inverse_transform(self, X)
 |      Reverse the transformation operation
 |      
 |      Parameters
 |      ----------
 |      X : array of shape [n_samples, n_selected_features]
 |          The input samples.
 |      
 |      Returns
 |      -------
 |      X_r : array of shape [n_samples, n_original_features]
 |          `X` with columns of zeros inserted where features would have
 |          been removed by :meth:`transform`.
 |  
 |  transform(self, X)
 |      Reduce X to the selected features.
 |      
 |      Parameters
 |      ----------
 |      X : array of shape [n_samples, n_features]
 |          The input samples.
 |      
 |      Returns
 |      -------
 |      X_r : array of shape [n_samples, n_selected_features]
 |          The input samples with only the selected features.
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.TransformerMixin:
 |  
 |  fit_transform(self, X, y=None, **fit_params)
 |      Fit to data, then transform it.
 |      
 |      Fits transformer to `X` and `y` with optional parameters `fit_params`
 |      and returns a transformed version of `X`.
 |      
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Input samples.
 |      
 |      y :  array-like of shape (n_samples,) or (n_samples, n_outputs),                 default=None
 |          Target values (None for unsupervised transformations).
 |      
 |      **fit_params : dict
 |          Additional fit parameters.
 |      
 |      Returns
 |      -------
 |      X_new : ndarray array of shape (n_samples, n_features_new)
 |          Transformed array.
 |  
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.TransformerMixin:
 |  
 |  __dict__
 |      dictionary for instance variables (if defined)
 |  
 |  __weakref__
 |      list of weak references to the object (if defined)
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |  
 |  __getstate__(self)
 |  
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |  
 |  __setstate__(self, state)
 |  
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |      
 |      Parameters
 |      ----------
 |      deep : bool, default=True
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |      
 |      Returns
 |      -------
 |      params : dict
 |          Parameter names mapped to their values.
 |  
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |      
 |      The method works on simple estimators as well as on nested objects
 |      (such as :class:`~sklearn.pipeline.Pipeline`). The latter have
 |      parameters of the form ``<component>__<parameter>`` so that it's
 |      possible to update each component of a nested object.
 |      
 |      Parameters
 |      ----------
 |      **params : dict
 |          Estimator parameters.
 |      
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
```

In [ ]:

Mutual Information Gain In Regression.

In [85]:
```
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score

boston = datasets.load_boston()
```

In [86]:
```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the targe
t.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity',
Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth Internat
ional Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

Here we can see that there is 13 attributes (columns)and 506 instances(rows)

In [87]:
```
x=pd.DataFrame(data=boston.data,columns=boston.feature_names)
x.head()
```

Out[87]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [88]:
```
y=boston.target
```

In [89]:
```
x_train,x_test,y_train,_y_test = train_test_split(x,y,train_size=.80,random_state=0)
```

In [90]:
```
x_train.shape,y_train.shape
```

Out[90]:
```
((404, 13), (404,))
```

In [91]:
```
y_train.shape,y_test.shape
```

Out[91]:
```
((404,), (37,))
```

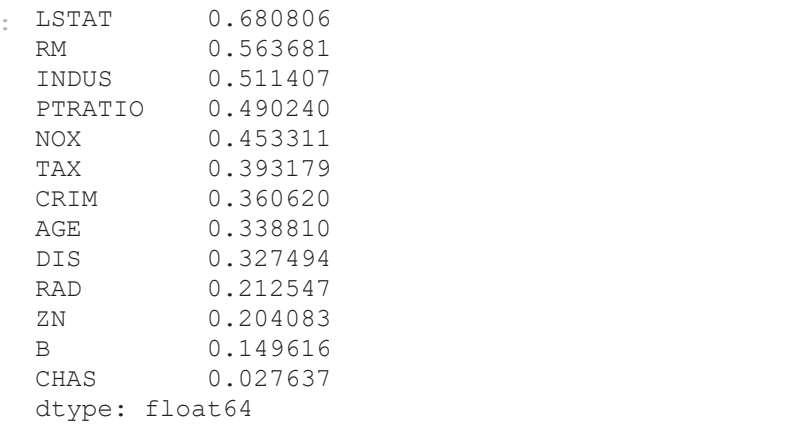
In [92]:
```
mi=mutual_info_regression(x_train,y_train)
mi =pd.Series(mi)
mi.index = x_train.columns
mi.sort_values(ascending =False,inplace = True)
```

In [93]:
```
mi
```

Out[93]:
```
LSTAT      0.680806
RM         0.563681
INDUS      0.511407
PTRATIO    0.490240
NOX        0.453311
TAX        0.393179
CRIM       0.360620
AGE        0.338810
DIS        0.327494
RAD        0.212547
ZN         0.204083
B          0.149616
CHAS       0.027637
dtype: float64
```

In [94]:
```
mi.plot.bar()
```

Out[94]:
```
<AxesSubplot:>
```



If we see the above plot in that the execpt one bar the other contributing more information and for that we can thses features are really important and the we are selecting the top 9 featues those who really contributing the imformation as expected.

In [95]:
```
sel = SelectKBest(mutual_info_regression,k=9).fit(x_train,y_train)
x_train.columns[sel.get_support()]
```

Out[95]:
```
Index(['CRIM', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS', 'TAX', 'PTRATIO', 'LSTAT'], dtype='object')
```

In [69]:
```
len(x_train.columns[sel.get_support()])
```

Out[69]:
```
9
```

In [96]:
```
x_train = np.array(x_train)
x_test  = np.array(x_test)
y_train = np.array(y_train)
y_test  = np.array(y_test)
```

In [97]:
```
model = LinearRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
y_pred
```

Out[97]:
```
array([24.88963777, 23.72141085, 29.36499868, 12.12238621, 21.44382254,
       19.2834443 , 20.49647539, 21.36099298, 18.8967118 , 19.9280658 ,
        5.12703513, 16.3867396 , 17.07776485,  5.59375659, 39.99636726,
       32.49654668, 22.45798809, 36.85192327, 30.86401089, 23.15140009,
       24.77495789, 24.67187756, 20.59543752, 30.35369168, 22.41940736,
       10.23266565, 17.64816865, 18.27419652, 35.53362541, 20.96084724,
       18.30413012, 17.79262072, 19.96561663, 24.06127231, 29.10204874,
       19.27774123, 11.15536648, 24.57560579, 17.5862644 , 15.49454112,
       26.20577527, 20.86304693, 22.31460516, 15.60710156, 23.00363104,
       25.17247952, 20.11459464, 22.90256276, 10.0380507 , 24.28515123,
       20.94127711, 17.35258791, 24.52235405, 29.95143046, 13.42695877,
       21.72673066, 20.7897053 , 15.49668805, 13.98982601, 22.18377874,
       17.73047814, 21.58869165, 32.90522136, 31.11235671, 17.73252635,
       32.76358681, 18.7124637 , 19.78693475, 19.02958927, 22.89825374,
       22.96041622, 24.02555703, 30.72859326, 28.83142691, 25.89957059,
        5.23251817, 36.72183202, 23.77267249, 27.26856352, 19.29492159,
       28.62304496, 19.17978838, 18.97185995, 37.82397662, 39.22012647,
       23.71261106, 24.93076217, 15.88545417, 26.09845751, 16.68819641,
       15.83515991, 13.10775597, 24.71583588, 31.25165267, 22.16640989,
       20.25087212,  0.59025319, 25.44217132, 15.57178328, 17.93719475,
       25.30588844, 22.3732326 ])
```

Thanks !!