select-k-best

August 21, 2024

[1]:
```
#These all are function, it's in single cell
import pandas as pd
from sklearn.model_selection import train_test_split
import time
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
import pickle
import matplotlib.pyplot as plt

#All the required libraries loaded above

def selectkbest(indep_X,dep_Y,n):
        test = SelectKBest(score_func=chi2, k=n)
        #formula load,Select K ALGORITHM we are using here, based on␣
 ↪chiSquare(chi=2 or Rsquare value) and k=n (which number we will give and try␣
 ↪for k parameter)
        fit1= test.fit(indep_X,dep_Y)
        #once formula loaded full process come to test'. so test.fit loading␣
 ↪input varaiable(indep_X-27 COLUMNS) AND oUtput varaiable(dep_Y- 1 COLUMN)␣
 ↪LOADED and Module ceate.
        #once fit1 method executed it'll create model.for example if we give␣
 ↪k=4 means it'll load best 4 value into selectk_features varaiable.
        selectk_features = fit1.transform(indep_X)
        #for example if we give k=4 means it'll load best 4 value into␣
 ↪selectk_features varaiable.so that using fit1.transform(indep_X)-(4.
 ↪transform(indep_X)) Giving 4 value to input as indep_X.
        return selectk_features
        #selectk_features RETURN the result.
        #SelectK best main rule, Suppose if we have 14 input columns in dataset␣
 ↪means, we give only 4 columns as input. this 4 column(n) we will get as␣
 ↪Select_feature using ChiSquare module.
        #ChiSquare same Rsquare evaluation matrix.
```

```
        #We are going using IT foR many algorithms like Logistic,svm,RF, SO␣
↪THAT we have integrated traningset and testset pirkarathu StandardScaler()␣
↪integrate panitom.


def split_scalar(indep_X,dep_Y):
    #Suppose if we input and output here indep_X,dep_Y as input, finally(tain␣
↪and test pricitu finally these 4 varaiables will execute) these X_train,␣
↪X_test, y_train, y_test values we will get.
        X_train, X_test, y_train, y_test = train_test_split(indep_X, dep_Y,␣
↪test_size = 0.25, random_state = 0)
        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.transform(X_test)
        return X_train, X_test, y_train, y_test

def cm_prediction(classifier,X_test):
    #its confusion matrix, if we give testset and classfier, all the above code␣
↪we given as seperate file, here created as functions.

     y_pred = classifier.predict(X_test)

        # Making the Confusion Matrix
     from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test, y_pred)

     from sklearn.metrics import accuracy_score
     from sklearn.metrics import classification_report

     Accuracy=accuracy_score(y_test, y_pred )

     report=classification_report(y_test, y_pred)
     return  classifier,Accuracy,report,X_test,y_test,cm
    #once above function executed above we will get result as␣
↪classifier,Accuracy,report,X_test,y_test,cm

def logistic(X_train,y_train,X_test):
        # Fitting K-NN to the Training set
        from sklearn.linear_model import LogisticRegression
        #Model load pandrom
        classifier = LogisticRegression(random_state = 0)
        classifier.fit(X_train, y_train)
        #fit panduvom
      ␣
↪classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        #training set and testset pirikarathu
```

```
        return  classifier,Accuracy,report,X_test,y_test,cm
    #here we called cm-prediction model.

def svm_linear(X_train,y_train,X_test):

        from sklearn.svm import SVC
        classifier = SVC(kernel = 'linear', random_state = 0)
        classifier.fit(X_train, y_train)
       ␣
 ↪classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm

def svm_NL(X_train,y_train,X_test):

        from sklearn.svm import SVC
        classifier = SVC(kernel = 'rbf', random_state = 0)
        classifier.fit(X_train, y_train)
       ␣
 ↪classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm

def Navie(X_train,y_train,X_test):
        # Fitting K-NN to the Training set
        from sklearn.naive_bayes import GaussianNB
        classifier = GaussianNB()
        classifier.fit(X_train, y_train)
       ␣
 ↪classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm


def knn(X_train,y_train,X_test):

        # Fitting K-NN to the Training set
        from sklearn.neighbors import KNeighborsClassifier
        classifier = KNeighborsClassifier(n_neighbors = 5, metric =␣
 ↪'minkowski', p = 2)
        classifier.fit(X_train, y_train)
       ␣
 ↪classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm
def Decision(X_train,y_train,X_test):

        # Fitting K-NN to the Training set
        from sklearn.tree import DecisionTreeClassifier
        classifier = DecisionTreeClassifier(criterion = 'entropy', random_state␣
 ↪= 0)
```

```
        classifier.fit(X_train, y_train)
       ␣
 ↪classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm


def random(X_train,y_train,X_test):

        # Fitting K-NN to the Training set
        from sklearn.ensemble import RandomForestClassifier
        classifier = RandomForestClassifier(n_estimators = 10, criterion =␣
 ↪'entropy', random_state = 0)
        classifier.fit(X_train, y_train)
       ␣
 ↪classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm

def selectk_Classification(acclog,accsvml,accsvmnl,accknn,accnav,accdes,accrf):

    dataframe=pd.
 ↪DataFrame(index=['ChiSquare'],columns=['Logistic','SVMl','SVMnl','KNN','Navie','Decision','
    for number,idex in enumerate(dataframe.index):
        dataframe['Logistic'][idex]=acclog[number]
        dataframe['SVMl'][idex]=accsvml[number]
        dataframe['SVMnl'][idex]=accsvmnl[number]
        dataframe['KNN'][idex]=accknn[number]
        dataframe['Navie'][idex]=accnav[number]
        dataframe['Decision'][idex]=accdes[number]
        dataframe['Random'][idex]=accrf[number]
    return dataframe
#Here we are creating table.enumerate like range, it'll come with index(1,2,3).␣
 ↪index helful for us the positoning.stilabove function load.
```

[2]:
```
dataset1=pd.read_csv("prep.csv",index_col=None)
#Loading preprocessed csv file.

df2=dataset1
#Hrer dataset1 created as backup, we are going to use df2(dataset2) furtherly.
df2 = pd.get_dummies(df2, drop_first=True)
#null value check and doing preprocessing

indep_X=df2.drop('classification_yes', 1)
#classification_yes is last column so delete classification_yes values, if we 1␣
 ↪means column wise delete.
#Now independent dependent varaiable pirkarathu.it's kidney cronic dataset.here␣
 ↪classification_yes is output varaibale that WE DROPPED here
```

```
dep_Y=df2['classification_yes']
#Again classification_yes is our output parameter.
#
```

[3]:
```
df2
#there is no categorical data here
```

[3]:
```
           age         bp   al   su         bgr          bu        sc  \
0     2.000000  76.459948  3.0  0.0  148.112676   57.482105  3.077356
1     3.000000  76.459948  2.0  0.0  148.112676   22.000000  0.700000
2     4.000000  76.459948  1.0  0.0   99.000000   23.000000  0.600000
3     5.000000  76.459948  1.0  0.0  148.112676   16.000000  0.700000
4     5.000000  50.000000  0.0  0.0  148.112676   25.000000  0.600000
..         …         …   …   …          …          …         …
394  51.492308  70.000000  0.0  0.0  219.000000   36.000000  1.300000
395  51.492308  70.000000  0.0  2.0  220.000000   68.000000  2.800000
396  51.492308  70.000000  3.0  0.0  110.000000  115.000000  6.000000
397  51.492308  90.000000  0.0  0.0  207.000000   80.000000  6.800000
398  51.492308  80.000000  0.0  0.0  100.000000   49.000000  1.000000

             sod       pot       hrmo  …  pc_normal  pcc_present  ba_present  \
0     137.528754  4.627244  12.518156  …          0            0           0
1     137.528754  4.627244  10.700000  …          1            0           0
2     138.000000  4.400000  12.000000  …          1            0           0
3     138.000000  3.200000   8.100000  …          1            0           0
4     137.528754  4.627244  11.800000  …          1            0           0
..           …         …         …  …  …          …          …           …
394   139.000000  3.700000  12.500000  …          1            0           0
395   137.528754  4.627244   8.700000  …          1            0           0
396   134.000000  2.700000   9.100000  …          1            0           0
397   142.000000  5.500000   8.500000  …          1            0           0
398   140.000000  5.000000  16.300000  …          1            0           0

     htn_yes  dm_yes  cad_yes  appet_yes  pe_yes  ane_yes  classification_yes
0          0       0        0          1       1        0                   1
1          0       0        0          1       0        0                   1
2          0       0        0          1       0        0                   1
3          0       0        0          1       0        1                   1
4          0       0        0          1       0        0                   1
..       …       …        …          …       …        …                   …
394        0       0        0          1       0        0                   1
395        1       1        0          1       0        1                   1
396        1       1        0          0       0        0                   1
397        1       1        0          1       0        1                   1
398        0       0        0          1       0        0                   0

[399 rows x 28 columns]
```

[17]:
```
kbest=selectkbest(indep_X,dep_Y,4)
#Here we are passing indep_X=df2.drop('classification_yes', 1) and␣
 ↪dep_Y=df2['classification_yes'] values  input passing as parameter.

acclog=[]
accsvml=[]
accsvmnl=[]
accknn=[]
accnav=[]
accdes=[]
accrf=[]
#Now we got accuracy of log,svn,knn like that as list.
#Now we are finding best 5 values.
#feature selection using select best 5 or 4 columns.
```

[18]:
```
kbest
```

[18]:
```
array([[1.48112676e+02, 5.74821053e+01, 3.07735602e+00, 8.40819113e+03],
       [1.48112676e+02, 2.20000000e+01, 7.00000000e-01, 1.23000000e+04],
       [9.90000000e+01, 2.30000000e+01, 6.00000000e-01, 8.40819113e+03],
       …,
       [1.10000000e+02, 1.15000000e+02, 6.00000000e+00, 9.20000000e+03],
       [2.07000000e+02, 8.00000000e+01, 6.80000000e+00, 8.40819113e+03],
       [1.00000000e+02, 4.90000000e+01, 1.00000000e+00, 8.50000000e+03]])
```

[19]:
```
#Program calling count, itll create table.
X_train, X_test, y_train, y_test=split_scalar(kbest,dep_Y)
#ontetime,here we are calling split_scalar(kbest,dep_Y) function and getting␣
 ↪

classifier,Accuracy,report,X_test,y_test,cm=logistic(X_train,y_train,X_test)
acclog.append(Accuracy)
#while running above it'll create logistic model and show that's␣
 ↪classifier,Accuracy,report,X_test,y_test,cm, but here we need only Accuracy.
#we are appending Accuracy with acclog.
classifier,Accuracy,report,X_test,y_test,cm=svm_linear(X_train,y_train,X_test)
accsvml.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=svm_NL(X_train,y_train,X_test)
accsvmnl.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=knn(X_train,y_train,X_test)
accknn.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=Navie(X_train,y_train,X_test)
accnav.append(Accuracy)
```

```
classifier,Accuracy,report,X_test,y_test,cm=Decision(X_train,y_train,X_test)
accdes.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=random(X_train,y_train,X_test)
accrf.append(Accuracy)

result=selectk_Classification(acclog,accsvml,accsvmnl,accknn,accnav,accdes,accrf)
#Finally in this classification if we give all the above result and run means␣
 ↪it'll create one table and show as result.
```

```
C:\Users\Kathirvel\Anaconda3\envs\aiml\lib\site-
packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\Kathirvel\Anaconda3\envs\aiml\lib\site-
packages\sklearn\neighbors\base.py:441: DeprecationWarning: distutils Version
classes are deprecated. Use packaging.version instead.
  old_joblib = LooseVersion(joblib_version) < LooseVersion('0.12')
C:\Users\Kathirvel\Anaconda3\envs\aiml\lib\site-
packages\sklearn\neighbors\base.py:441: DeprecationWarning: distutils Version
classes are deprecated. Use packaging.version instead.
  old_joblib = LooseVersion(joblib_version) < LooseVersion('0.12')
C:\Users\Kathirvel\Anaconda3\envs\aiml\lib\site-
packages\sklearn\utils\fixes.py:230: DeprecationWarning: distutils Version
classes are deprecated. Use packaging.version instead.
  if _joblib.__version__ >= LooseVersion('0.12'):
C:\Users\Kathirvel\Anaconda3\envs\aiml\lib\site-
packages\sklearn\utils\fixes.py:230: DeprecationWarning: distutils Version
classes are deprecated. Use packaging.version instead.
  if _joblib.__version__ >= LooseVersion('0.12'):
```

[7]:
```
result
#5
```

[7]:

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.96	0.96	0.96	0.93	0.89	0.97	0.97

[8]:
```
result
#3
#above good
```

[8]:

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.96	0.96	0.96	0.93	0.89	0.97	0.97

[20]:
```
result
#4
```

[20]:

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.85	0.82	0.83	0.86	0.79	0.89	0.89

[16]:
```
result
#6
#It's best
```

[16]:

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.96	0.96	0.96	0.93	0.89	0.97	0.97

[ ]:
```
#using 6 we can choose as best algorithm. we can take mode and get repeated␣
 ↪value as our final answer.
#we are putting input 5 or 6 into all algorithms and getting predictions. and␣
 ↪choosing which algorithm prediction is good.Finally if we mode we can use␣
 ↪repeated answer.
#fLOW DIAGRAM IS need to PREPARE.
```