swetha-lda

May 8, 2023

[1]:
```
#Importing required libraries
import numpy as np
import pandas as pd
```

[2]:
```
df=pd.read_csv('ctg_data1.csv')
df
```

[2]:

	b	e	AC	FM	UC	DL	DS	DP	DR	LB	…	C	D	E	AD	DE	LD	\
0	240	357	0	0	0	0	0	0	0	120	…	-1	-1	-1	-1	-1	-1	
1	5	632	4	0	4	2	0	0	0	132	…	-1	-1	-1	1	-1	-1	
2	177	779	2	0	5	2	0	0	0	133	…	-1	-1	-1	1	-1	-1	
3	411	1192	2	0	6	2	0	0	0	134	…	-1	-1	-1	1	-1	-1	
4	533	1147	4	0	5	0	0	0	0	132	…	-1	-1	-1	-1	-1	-1	
…	…	…	..	..	..	..	..	..	..	…	…	..	..	..	..	..	..	
2121	2059	2867	0	0	6	0	0	0	0	140	…	-1	-1	1	-1	-1	-1	
2122	1576	2867	1	0	9	0	0	0	0	140	…	-1	-1	1	-1	-1	-1	
2123	1576	2596	1	0	7	0	0	0	0	140	…	-1	-1	1	-1	-1	-1	
2124	1576	3049	1	0	9	0	0	0	0	140	…	-1	-1	1	-1	-1	-1	
2125	2796	3415	1	1	5	0	0	0	0	142	…	-1	-1	-1	-1	-1	-1	

	FS	SUSP	CLASS	NSP
0	1	-1	9	2
1	-1	-1	6	1
2	-1	-1	6	1
3	-1	-1	6	1
4	-1	-1	2	1
…	..	…	…	…
2121	-1	-1	5	2
2122	-1	-1	5	2
2123	-1	-1	5	2
2124	-1	-1	5	2
2125	-1	-1	1	1

[2126 rows x 42 columns]

[3]:
```
df.dtypes
```

[3]:
```
b             int64
e             int64
AC            int64
FM            int64
UC            int64
DL            int64
DS            int64
DP            int64
DR            int64
LB            int64
AC.1        float64
FM.1        float64
UC.1        float64
DL.1        float64
DS.1        float64
DP.1        float64
ASTV          int64
MSTV        float64
ALTV          int64
MLTV        float64
Width         int64
Min           int64
Max           int64
Nmax          int64
Nzeros        int64
Mode          int64
Mean          int64
Median        int64
Variance      int64
Tendency      int64
A             int64
B             int64
C             int64
D             int64
E             int64
AD            int64
DE            int64
LD            int64
FS            int64
SUSP          int64
CLASS         int64
NSP           int64
dtype: object
```

[4]:
```
df.isna().sum()
```

[4]:
```
b           0
e           0
AC          0
FM          0
UC          0
DL          0
DS          0
DP          0
DR          0
LB          0
AC.1        0
FM.1        0
UC.1        0
DL.1        0
DS.1        0
DP.1        0
ASTV        0
MSTV        0
ALTV        0
MLTV        0
Width       0
Min         0
Max         0
Nmax        0
Nzeros      0
Mode        0
Mean        0
Median      0
Variance    0
Tendency    0
A           0
B           0
C           0
D           0
E           0
AD          0
DE          0
LD          0
FS          0
SUSP        0
CLASS       0
NSP         0
dtype: int64
```

[5]:
```
df.dropna()
```

[5]:
```
          b     e  AC  FM  UC  DL  DS  DP  DR   LB  …  C  D  E  AD  DE  LD  \
0       240   357   0   0   0   0   0   0   0  120  … -1 -1 -1  -1  -1  -1
1         5   632   4   0   4   2   0   0   0  132  … -1 -1 -1   1  -1  -1
2       177   779   2   0   5   2   0   0   0  133  … -1 -1 -1   1  -1  -1
3       411  1192   2   0   6   2   0   0   0  134  … -1 -1 -1   1  -1  -1
4       533  1147   4   0   5   0   0   0   0  132  … -1 -1 -1  -1  -1  -1
…     …   …  ..  ..  ..  ..  ..  ..  ..  …  … .. .. ..  ..  ..  ..
2121  2059  2867   0   0   6   0   0   0   0  140  … -1 -1  1  -1  -1  -1
2122  1576  2867   1   0   9   0   0   0   0  140  … -1 -1  1  -1  -1  -1
2123  1576  2596   1   0   7   0   0   0   0  140  … -1 -1  1  -1  -1  -1
2124  1576  3049   1   0   9   0   0   0   0  140  … -1 -1  1  -1  -1  -1
2125  2796  3415   1   1   5   0   0   0   0  142  … -1 -1 -1  -1  -1  -1

      FS  SUSP  CLASS  NSP
0      1    -1      9    2
1     -1    -1      6    1
2     -1    -1      6    1
3     -1    -1      6    1
4     -1    -1      2    1
…    ..   …    …  …
2121  -1    -1      5    2
2122  -1    -1      5    2
2123  -1    -1      5    2
2124  -1    -1      5    2
2125  -1    -1      1    1

[2126 rows x 42 columns]
```

[6]:
```
df.isna().sum()
```

[6]:
```
b           0
e           0
AC          0
FM          0
UC          0
DL          0
DS          0
DP          0
DR          0
LB          0
AC.1        0
FM.1        0
UC.1        0
DL.1        0
DS.1        0
DP.1        0
ASTV        0
```

```
MSTV        0
ALTV        0
MLTV        0
Width       0
Min         0
Max         0
Nmax        0
Nzeros      0
Mode        0
Mean        0
Median      0
Variance    0
Tendency    0
A           0
B           0
C           0
D           0
E           0
AD          0
DE          0
LD          0
FS          0
SUSP        0
CLASS       0
NSP         0
dtype: int64
```

[7]:
```
Features=df.drop('NSP', axis=1)
Label=df['NSP']
```

[8]:
```
Features.shape
```

[8]:
```
(2126, 41)
```

[9]:
```
Features_T=Features.T
#Features_T.columns= Features_T.iloc[0]
#Features_T.columns
```

[10]:
```
height, width = Features.shape
unique_classes = np.unique(Label)
unique_classes
```

[10]:
```
array([1, 2, 3])
```

[11]:
```
num_classes = len(unique_classes)

scatter_train = np.cov(Features_T)*(height - 1)
```

```
scatter_within = 0
```

[12]:
```
for i in range(num_classes):
  class_items = np.flatnonzero(Label == unique_classes[i])
  scatter_within = scatter_within + np.cov(Features_T[class_items]) *␣
 ↪(len(class_items)-1)

scatter_between = scatter_train - scatter_within
```

[13]:
```
#Calculating Eigenvalues and Eigenvectors of the covariance matrix

eigen_values, eigen_vectors = np.linalg.eigh(np.linalg.pinv(scatter_within).
 ↪dot(scatter_between))
        #print(eig_vectors.shape)
        #pc = Features.dot(eig_vectors[:,::-1][:,:self.n_components])
```

[14]:
```
#sort the eigenvalues in descending order
sorted_index = np.argsort(eigen_values)[::-1]

sorted_eigenvalue = eigen_values[sorted_index]
#similarly sort the eigenvectors
sorted_eigenvectors = eigen_vectors[:,sorted_index]
sorted_eigenvectors
```

[14]:
```
array([[ 0.4732847 ,  0.24254814,  0.42273079, …,  0.41870789,
        -0.24948021, -0.47328548],
       [ 0.52515063, -0.21592378, -0.37658088, …, -0.37297375,
         0.22209553, -0.52515194],
       [ 0.00600605, -0.0299112 , -0.05207817, …, -0.05190328,
         0.03087553, -0.00590249],
       …,
       [-0.0053102 ,  0.34093265, -0.1084806 , …,  0.11113142,
         0.33409319, -0.00530764],
       [ 0.00305067, -0.31402563,  0.08722879, …, -0.09110959,
        -0.31046069,  0.00304864],
       [-0.00265968,  0.0893928 , -0.03727638, …,  0.03699019,
         0.08649929, -0.00265855]])
```

[15]:
```
# select the first n eigenvectors, n is desired dimension
# of our final reduced data.

n_components = 30 #you can select any number of components.
eigenvector_subset = sorted_eigenvectors[:,0:n_components]
eigenvector_subset
```

[15]:
```
array([[ 4.73284698e-01,  2.42548135e-01,  4.22730790e-01, …,
         7.63720733e-10,  1.75092483e-04,  1.37573284e-08],
```

```
        [ 5.25150629e-01, -2.15923776e-01, -3.76580882e-01, …,
         -6.85421174e-10, -1.59906800e-04, -1.25703354e-08],
        [ 6.00604869e-03, -2.99111997e-02, -5.20781663e-02, …,
         -1.05205744e-09, -7.82502928e-05, -5.56153603e-09],
        …,
        [-5.31020232e-03,  3.40932652e-01, -1.08480604e-01, …,
         -1.00785630e-04,  2.07782694e-08, -5.76979344e-04],
        [ 3.05067081e-03, -3.14025633e-01,  8.72287917e-02, …,
         -1.01899986e-04,  2.00804668e-08, -5.39667020e-04],
        [-2.65968191e-03,  8.93927987e-02, -3.72763803e-02, …,
          3.94605752e-05, -7.25295933e-09,  1.90991045e-04]])
```

[16]:
```
#Transform the data
Features_reduced = np.dot(eigenvector_subset.transpose(),Features.transpose()).
 ↪transpose()
Features_reduced
```

[16]:
```
array([[ 3.00219816e+02, -6.59347499e+01, -1.70109936e+01, …,
         9.66501783e+01,  9.93731877e-02, -1.28168547e+02],
        [ 3.33399785e+02, -2.29947009e+02, -1.93619923e+02, …,
         8.90024644e+01,  2.92614281e-02, -1.12946527e+02],
        [ 4.91985411e+02, -2.19317383e+02, -1.76595992e+02, …,
         8.71069126e+01,  3.98596106e-02, -1.11783888e+02],
        …,
        [ 2.10824774e+03, -2.76186450e+02, -2.68519034e+02, …,
         8.52819164e+01, -2.73678564e-03, -1.14438456e+02],
        [ 2.34614499e+03, -3.72528190e+02, -4.39990901e+02, …,
         8.63382791e+01, -7.52897025e-02, -1.13345872e+02],
        [ 3.11573744e+03, -1.55130641e+02, -6.22744758e+01, …,
         7.48223254e+01,  7.44609241e-02, -1.07311230e+02]])
```

[17]:
```
LDA_df = pd.DataFrame(Features_reduced)
LDA_df
```

[17]:

	0	1	2	3	4	5	\
0	300.219816	-65.934750	-17.010994	59.124660	5.844690	83.327318	
1	333.399785	-229.947009	-193.619923	4.830999	5.698256	63.120979	
2	491.985411	-219.317383	-176.595992	4.263896	5.729716	67.714538	
3	819.616540	-249.799360	-234.419524	3.914594	5.726278	63.298216	
4	853.750886	-211.771827	-165.231601	4.768994	1.620399	71.980959	
…	…	…	…	…	…	…	
2121	2479.152505	-223.564029	-163.022571	92.693472	0.325785	92.150053	
2122	2250.567891	-334.604751	-370.648326	70.348146	1.251712	60.715813	
2123	2108.247735	-276.186450	-268.519034	70.427735	1.285141	74.090073	
2124	2346.144988	-372.528190	-439.990901	71.613250	1.165710	51.774940	
2125	3115.737442	-155.130641	-62.274476	86.923337	-3.104452	114.572317	

```
            6         7          8          9   …         20         21  \
0    -0.203357  1.663515  35.688892 -92.227559  … -0.002469  6.223204
1    -1.465685  2.318493   7.015414 -80.783311  … -0.003556  9.602217
2    -1.468023  2.360565   8.122100 -81.004249  … -0.003581  9.519549
3    -1.438225  2.373349   7.973535 -78.399752  … -0.003149  7.671355
4    -0.417396  2.008112   7.561642 -78.252255  … -0.003221  7.794299
…          …         …          …         …  …         …         …
2121 -1.527871  0.195028  17.044123 -92.712355  … -0.003341  8.982612
2122 -1.489909  0.360775   5.286524 -95.852363  … -0.003301  8.396834
2123 -1.496576  0.375939   4.856340 -97.232114  … -0.003310  8.492253
2124 -1.438468  0.291062   5.286853 -94.067070  … -0.003290  8.392110
2125  0.505112 -0.218215  12.720148 -83.709241  … -0.003137  8.078005

              22         23        24        25          26         27  \
0     120.825410  -1.982918 -2.631853 -1.082220   22.913832  96.650178
1     186.113122 -10.672797 -0.208109 -0.797236  116.502231  89.002464
2     184.581013 -10.984091 -0.249276 -0.806527  113.477588  87.106913
3     150.432435 -13.923588 -1.373662 -0.902953  148.155457  89.211810
4     152.384273 -12.007955 -1.343551 -0.969307  114.884013  89.290539
…            …         …         …         …          …         …
2121  173.529240 -14.094041 -0.321187 -0.796896  158.297727  70.721189
2122  162.741917 -23.273412 -1.308703 -0.859766  248.717641  84.805254
2123  164.381248 -18.131520 -1.336325 -0.866378  197.032727  85.281916
2124  162.712550 -25.742869 -1.321778 -0.860555  283.661612  86.338279
2125  156.015743 -10.251931 -0.368898 -0.834270  123.420448  74.822325

            28          29
0     0.099373 -128.168547
1     0.029261 -112.946527
2     0.039860 -111.783888
3     0.016817 -111.603704
4     0.039574 -111.900873
…          …          …
2121  0.032935 -113.128042
2122 -0.045385 -113.577918
2123 -0.002737 -114.438456
2124 -0.075290 -113.345872
2125  0.074461 -107.311230

[2126 rows x 30 columns]
```

[18]:
```
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

[19]:
```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(LDA_df, Label, test_size=0.
 ↪3, random_state=1)
```

Decision Tree

[20]:
```
from sklearn.tree import DecisionTreeClassifier
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred_train = clf.predict(X_train)
```

[21]:
```
print("Training-set accuracy (in %):",metrics.accuracy_score(y_train,␣
 ↪y_pred_train)*100)
```

```
Training-set accuracy (in %): 99.93279569892472
```

[22]:
```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=8)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy (in %):",metrics.accuracy_score(y_test, y_pred)*100)
```

```
Accuracy (in %): 95.92476489028213
```

Naive Bayes

[23]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)

y_pred_train = gnb.predict(X_train)

print('Training-set accuracy (in %):', metrics.accuracy_score(y_train,␣
 ↪y_pred_train)*100)
```

```
Training-set accuracy (in %): 84.13978494623656
```

[24]:
```
# making predictions on the testing set
y_pred = gnb.predict(X_test)

print("Gaussian Naive Bayes model accuracy (in %):", metrics.
 ↪accuracy_score(y_test, y_pred)*100)
```

```
Gaussian Naive Bayes model accuracy (in %): 81.50470219435736
```

Random Forest

[25]:
```
# importing random forest classifier from assemble module
from sklearn.ensemble import RandomForestClassifier
```

[26]:
```
# creating a RF classifier
rfcl = RandomForestClassifier(n_estimators = 100)

# Training the model on the training dataset
rfcl.fit(X_train, y_train)

y_pred_train = rfcl.predict(X_train)

print('Training-set accuracy (in %):', metrics.accuracy_score(y_train,␣
 ↪y_pred_train)*100)
```

```
Training-set accuracy (in %): 99.93279569892472
```

[27]:
```
# performing predictions on the test dataset
y_pred = rfcl.predict(X_test)

print('Training-set accuracy (in %):', metrics.accuracy_score(y_test,␣
 ↪y_pred)*100)
```

```
Training-set accuracy (in %): 96.23824451410658
```

SVM

[28]:
```
#Import svm model
from sklearn import svm

#Create a svm Classifier
svmclf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
svmclf.fit(X_train, y_train)

y_pred_train = svmclf.predict(X_train)

print('Training-set accuracy (in %):', metrics.accuracy_score(y_train,␣
 ↪y_pred_train)*100)
```

```
Training-set accuracy (in %): 98.0510752688172
```

[29]:
```
#Predict the response for test dataset
y_pred = svmclf.predict(X_test)
```

```
# using metrics module for accuracy calculation
print("SVM model accuracy (in %): ", metrics.accuracy_score(y_test, y_pred)*100)
```

```
SVM model accuracy (in %):  96.86520376175548
```

[ ]: