Vijay(02-09-2023) ¶

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

In [2]:
```
from sklearn.linear_model import LogisticRegression
df=pd.read_csv(r"C:\USERS\user\Downloads\C1_ionosphere.csv")
df
```

Out[2]:

	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.1	0.03760	...	-0.511
0	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.265
1	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.402
2	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.906
3	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.651
4	1	0	0.02337	-0.00592	-0.09924	-0.11949	-0.00763	-0.11824	0.14706	0.06637	...	-0.015
...	...	...	...	...	...	...	...	...	...	...	...	
345	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	...	-0.042
346	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	...	0.013
347	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	...	0.031
348	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	...	-0.020
349	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	...	-0.151

350 rows × 35 columns

In [3]:
```
feature_matrix=df.iloc[:,0:34]
target_vector=df.iloc[:,-1]
```

In [4]:
```
feature_matrix.shape
target_vector.shape
```

Out[4]:
```
(350,)
```

In [5]:
```
from sklearn.preprocessing import StandardScaler
fs=StandardScaler().fit_transform(feature_matrix)
fs
```

Out[5]:
```
array([[ 0.34899122,  0.        ,  0.72317624, ..., -0.11824737,
        -0.93229623, -0.08614177],
       [ 0.34899122,  0.        ,  0.72317624, ..., -0.46718457,
         0.40303208, -0.85144926],
       [ 0.34899122,  0.        ,  0.72317624, ...,  1.95462525,
        -1.28905735,  2.1044496 ],
       ...,
       [ 0.34899122,  0.        ,  0.61663096, ...,  0.01300388,
         1.10438418, -0.04615616],
       [ 0.34899122,  0.        ,  0.53433434, ..., -0.06888676,
         1.00308117, -0.38113722],
       [ 0.34899122,  0.        ,  0.41574516, ..., -0.12585332,
         0.97171818, -0.16534322]])
```

In [6]:
```
logr=LogisticRegression()
logr.fit(fs,target_vector)
```

Out[6]:
```
LogisticRegression()
```

In [7]:
```
observation=[[1.4,2.3,-5.0,11,12,13,14,15,16,17,1,2,3,4,5,6,7,8,9,10,21,22
```

In [8]:
```
prediction=logr.predict(observation)
prediction
```

Out[8]:
```
array(['g'], dtype=object)
```

In [9]:
```
logr.classes_
```

Out[9]:
```
array(['b', 'g'], dtype=object)
```

In [10]:
```
logr.predict_proba(observation)[0][0]
```

Out[10]:
```
0.0
```

In [11]:
```
import re
from sklearn.datasets import load_digits
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [12]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [13]:
```
digits=load_digits()
digits
```

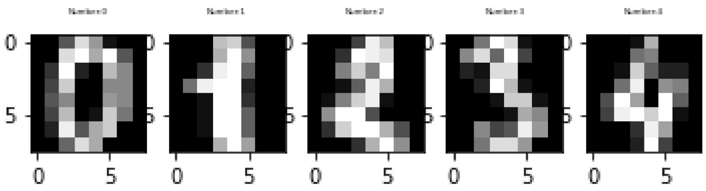
```
        [ 0.,  8., 16., ..., 16.,  8.,  0.],
        [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]),
 'DESCR': ".. _digits_dataset:\n\nOptical recognition of handwritten digits
dataset\n--------------------------------------------------\n\n**Data Set C
haracteristics:**\n\n    :Number of Instances: 1797\n    :Number of Attribu
tes: 64\n    :Attribute Information: 8x8 image of integer pixels in the ran
ge 0..16.\n    :Missing Attribute Values: None\n    :Creator: E. Alpaydin
(alpaydin '@' boun.edu.tr)\n    :Date: July; 1998\n\nThis is a copy of the
test set of the UCI ML hand-written digits datasets\nhttps://archive.ics.uc
i.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits\n\nThe data set
contains images of hand-written digits: 10 classes where\neach class refers
to a digit.\n\nPreprocessing programs made available by NIST were used to e
xtract\nnormalized bitmaps of handwritten digits from a preprinted form. Fr
om a\ntotal of 43 people, 30 contributed to the training set and different
13\nto the test set. 32x32 bitmaps are divided into nonoverlapping blocks o
f\n4x4 and the number of on pixels are counted in each block. This generate
s\nan input matrix of 8x8 where each element is an integer in the range\n
0..16. This reduces dimensionality and gives invariance to small\ndistortio
ns.\n\nFor info on NIST preprocessing routines, see M. D. Garris, J. L. Blu
```

In [14]:
```
plt.figure(figsize=(20,4))
```

Out[14]:
```
<Figure size 1440x288 with 0 Axes>

<Figure size 1440x288 with 0 Axes>
```

In [28]:
```
for index,(image,label) in enumerate(zip(digits.data[0:5],digits.target[0:
    plt.subplot(1,5,index+1)
    plt.imshow(np.reshape(image,(8
                                 ,8)),cmap=plt.cm.gray)
    plt.title('Number:%i\n'%label,fontsize=4)
```



In [16]:
```
x_train,x_test,y_train,y_test=train_test_split(digits.data,digits.target,t
```

In [17]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1257, 64)
(540, 64)
(1257,)
(540,)
```

In [18]:

```
logre=LogisticRegression(max_iter=10000)
logre.fit(x_train,y_train)

```

Out[18]:

```
LogisticRegression(max_iter=10000)
```

In [19]:

```
print(logre.predict(x_test))
```

```
[0 7 5 8 5 5 9 9 0 4 1 4 1 3 9 9 5 6 4 5 0 2 2 6 6 1 7 0 8 3 8 9 7 4 3 8 4
 8 0 0 7 7 5 9 1 8 4 3 7 1 3 8 9 0 8 3 3 4 7 8 0 0 2 5 5 2 9 3 8 7 8 4 8 1
 8 8 7 3 3 8 1 0 1 1 5 3 2 5 7 1 5 3 0 3 4 0 3 3 8 4 9 3 8 2 7 5 9 0 4 9 7
 9 6 1 1 3 8 4 1 6 4 6 4 4 9 7 9 1 0 7 0 7 9 5 9 8 5 3 8 1 5 4 6 6 0 0 6 0
 9 5 8 3 5 4 6 8 8 5 3 2 8 3 2 5 0 1 8 5 8 5 1 0 1 8 4 3 8 6 6 9 7 4 4 9 5
 2 0 6 4 9 3 2 1 7 2 7 2 5 2 8 9 2 1 1 9 1 1 9 0 8 3 5 0 9 4 9 1 8 7 1 2 8
 8 3 8 8 7 4 4 9 3 9 7 2 8 3 4 4 2 7 8 7 2 7 7 8 2 2 2 4 8 8 1 0 9 1 4 1 1
 2 0 7 7 3 3 2 8 3 2 4 9 2 2 5 4 2 4 6 7 3 3 9 0 0 0 3 3 0 9 6 4 2 5 5 9 8
 1 1 3 4 6 1 1 4 6 2 4 0 1 6 6 6 0 4 7 2 4 5 6 4 8 6 8 4 8 0 7 5 0 6 7 0 0
 9 1 6 8 5 2 1 4 3 9 6 6 8 6 0 4 5 0 3 6 3 7 4 2 1 3 1 7 4 5 9 6 7 7 3 1 4
 9 3 2 1 7 6 5 5 1 2 8 0 7 1 2 6 5 3 3 3 3 1 8 2 8 1 4 7 8 3 7 4 1 0 6 7 2
 0 2 7 5 5 0 8 1 0 8 3 2 2 6 9 1 9 4 5 5 4 9 5 0 2 7 4 4 9 2 7 6 6 4 8 7 2
 4 3 7 5 2 7 5 1 6 0 8 9 6 7 7 2 4 6 2 1 6 4 9 7 1 0 4 9 5 9 5 7 3 5 4 0 7
 6 8 7 2 3 0 5 8 4 4 2 1 6 3 6 5 7 7 5 3 1 6 6 2 7 4 3 3 7 8 3 4 7 7 5 7 4
 2 0 5 3 6 6 8 2 6 9 8 5 4 5 6 7 4 0 2 8 7 5]
```

In [23]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [30]:

```
df=pd.read_csv(r"C:\USERS\user\Downloads\C1_ionosphere.csv")
```

In [31]:

```
df['g'].value_counts()
```

Out[31]:

```
g    224
b    126
Name: g, dtype: int64
```

In [32]:

```
x=df.drop('g',axis=1)
y=df['g']
```

In [33]:
```
g1={"g":{'g':1,'b':2}}
df=df.replace(g1)
print(df)
```

```
     1  0  0.99539  -0.05889  0.85243  0.02306  0.83398  -0.37708      1.1  \
0    1  0  1.00000  -0.18829  0.93035 -0.36156 -0.10868  -0.93597  1.00000   
1    1  0  1.00000  -0.03365  1.00000  0.00485  1.00000  -0.12062  0.88965   
2    1  0  1.00000  -0.45161  1.00000  1.00000  0.71216  -1.00000  0.00000   
3    1  0  1.00000  -0.02401  0.94140  0.06531  0.92106  -0.23255  0.77152   
4    1  0  0.02337  -0.00592 -0.09924 -0.11949 -0.00763  -0.11824  0.14706   
..  .. ..      ...       ...      ...      ...      ...       ...      ...   
345  1  0  0.83508   0.08298  0.73739 -0.14706  0.84349  -0.05567  0.90441   
346  1  0  0.95113   0.00419  0.95183 -0.02723  0.93438  -0.01920  0.94590   
347  1  0  0.94701  -0.00034  0.93207 -0.03227  0.95177  -0.03431  0.95584   
348  1  0  0.90608  -0.01657  0.98122 -0.01989  0.95691  -0.03646  0.85746   
349  1  0  0.84710   0.13533  0.73638 -0.06151  0.87873   0.08260  0.88928   

     0.03760  ...  -0.51171  0.41078  -0.46168  0.21266  -0.34090  0.42267  \
0   -0.04549  ...  -0.26569 -0.20468  -0.18401 -0.19040  -0.11593 -0.16626   
1    0.01198  ...  -0.40220  0.58984  -0.22145  0.43100  -0.17365  0.60436   
2    0.00000  ...   0.90695  0.51613   1.00000  1.00000  -0.20099  0.25682   
3   -0.16399  ...  -0.65158  0.13290  -0.53206  0.02431  -0.62197 -0.05707   
4    0.06637  ...  -0.01535 -0.03240   0.09223 -0.07859   0.00732  0.00000   
..       ...  ...       ...      ...       ...      ...       ...      ...   
345 -0.04622  ...  -0.04202  0.83479   0.00123  1.00000   0.12815  0.86660   
346  0.01606  ...   0.01361  0.93522   0.04925  0.93159   0.08168  0.94066   
347  0.02446  ...   0.03193  0.92489   0.02542  0.92120   0.02242  0.92459   
348  0.00110  ...  -0.02099  0.89147  -0.07760  0.82983  -0.17238  0.96022   
349 -0.09139  ...  -0.15114  0.81147  -0.04822  0.78207  -0.00703  0.75747   

     -0.54487  0.18641  -0.45300  g  
0    -0.06288 -0.13738  -0.02447  2  
1    -0.24180  0.56045  -0.38238  1  
2     1.00000 -0.32382   1.00000  2  
3    -0.59573 -0.04608  -0.65697  1  
4     0.00000 -0.00039   0.12011  2  
..        ...      ...       ... ..  
345  -0.10714  0.90546  -0.04307  1  
346  -0.00035  0.91483   0.04712  1  
347   0.00442  0.92697  -0.00577  1  
348  -0.03757  0.87403  -0.16243  1  
349  -0.06678  0.85764  -0.06151  1  

[350 rows x 35 columns]
```

In [34]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.70)
```

In [35]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [36]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[36]:
```
RandomForestClassifier()
```

In [37]:

```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]}
```

In [38]:

```
from sklearn.model_selection import GridSearchCV
```

In [39]:

```
grid_search=GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring=
grid_search.fit(x_train,y_train)
```

Out[39]:

```
GridSearchCV(cv=2, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1, 2, 3, 4, 5],
                         'min_samples_leaf': [5, 10, 15, 20, 25],
                         'n_estimators': [10, 20, 30, 40, 50]},
             scoring='accuracy')
```

In [40]:

```
grid_search.best_score_
```

Out[40]:

```
0.9426229508196722
```

In [41]:

```
rfc_best=grid_search.best_estimator_
```

In [42]:

```
from sklearn.tree import plot_tree
```

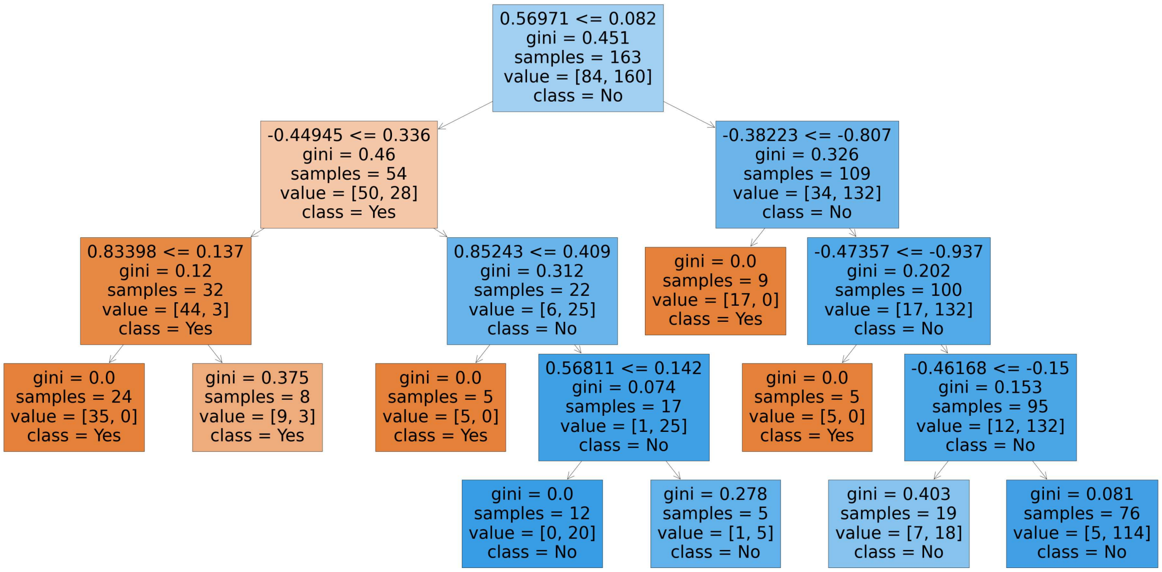
In [43]:

```
plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,class_names=['Ye

```

Out[43]:

```
[Text(2232.0, 1956.96, '0.56971 <= 0.082\ngini = 0.451\nsamples = 163\nvalue
= [84, 160]\nclass = No'),
 Text(1373.5384615384614, 1522.0800000000002, '-0.44945 <= 0.336\ngini = 0.46
\nsamples = 54\nvalue = [50, 28]\nclass = Yes'),
 Text(686.7692307692307, 1087.2, '0.83398 <= 0.137\ngini = 0.12\nsamples = 32
\nvalue = [44, 3]\nclass = Yes'),
 Text(343.38461538461536, 652.3200000000002, 'gini = 0.0\nsamples = 24\nvalue
= [35, 0]\nclass = Yes'),
 Text(1030.1538461538462, 652.3200000000002, 'gini = 0.375\nsamples = 8\nvalu
e = [9, 3]\nclass = Yes'),
 Text(2060.3076923076924, 1087.2, '0.85243 <= 0.409\ngini = 0.312\nsamples =
22\nvalue = [6, 25]\nclass = No'),
 Text(1716.9230769230767, 652.3200000000002, 'gini = 0.0\nsamples = 5\nvalue
= [5, 0]\nclass = Yes'),
 Text(2403.6923076923076, 652.3200000000002, '0.56811 <= 0.142\ngini = 0.074
\nsamples = 17\nvalue = [1, 25]\nclass = No'),
 Text(2060.3076923076924, 217.44000000000005, 'gini = 0.0\nsamples = 12\nvalu
e = [0, 20]\nclass = No'),
 Text(2747.076923076923, 217.44000000000005, 'gini = 0.278\nsamples = 5\nvalu
e = [1, 5]\nclass = No'),
 Text(3090.461538461538, 1522.0800000000002, '-0.38223 <= -0.807\ngini = 0.32
6\nsamples = 109\nvalue = [34, 132]\nclass = No'),
 Text(2747.076923076923, 1087.2, 'gini = 0.0\nsamples = 9\nvalue = [17, 0]\nc
lass = Yes'),
 Text(3433.8461538461534, 1087.2, '-0.47357 <= -0.937\ngini = 0.202\nsamples
= 100\nvalue = [17, 132]\nclass = No'),
 Text(3090.461538461538, 652.3200000000002, 'gini = 0.0\nsamples = 5\nvalue =
[5, 0]\nclass = Yes'),
 Text(3777.230769230769, 652.3200000000002, '-0.46168 <= -0.15\ngini = 0.153
\nsamples = 95\nvalue = [12, 132]\nclass = No'),
 Text(3433.8461538461534, 217.44000000000005, 'gini = 0.403\nsamples = 19\nva
lue = [7, 18]\nclass = No'),
 Text(4120.615384615385, 217.44000000000005, 'gini = 0.081\nsamples = 76\nval
ue = [5, 114]\nclass = No')]
```



In [ ]: