byrldh2f8

February 6, 2023

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
# from sklearn.neighbors import KNeighborsRegressor (if u want to use it in␣
 ↪regression )
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold , cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.feature_selection import SelectKBest , f_classif
from time import time
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
data = pd.read_csv('breast cancer.csv', index_col = 0)
data.head()
```

[3]:
```
         diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
id
842302           M        17.99         10.38          122.80     1001.0
842517           M        20.57         17.77          132.90     1326.0
84300903         M        19.69         21.25          130.00     1203.0
84348301         M        11.42         20.38           77.58      386.1
84358402         M        20.29         14.34          135.10     1297.0

          smoothness_mean  compactness_mean  concavity_mean  \
id
842302            0.11840           0.27760          0.3001
842517            0.08474           0.07864          0.0869
84300903          0.10960           0.15990          0.1974
84348301          0.14250           0.28390          0.2414
84358402          0.10030           0.13280          0.1980

          concave points_mean  symmetry_mean  …  texture_worst  \
```

```
id                                      …
842302                 0.14710  0.2419  …          17.33
842517                 0.07017  0.1812  …          23.41
84300903               0.12790  0.2069  …          25.53
84348301               0.10520  0.2597  …          26.50
84358402               0.10430  0.1809  …          16.67

          perimeter_worst  area_worst  smoothness_worst  compactness_worst  \
id
842302             184.60      2019.0            0.1622             0.6656
842517             158.80      1956.0            0.1238             0.1866
84300903           152.50      1709.0            0.1444             0.4245
84348301            98.87       567.7            0.2098             0.8663
84358402           152.20      1575.0            0.1374             0.2050

          concavity_worst  concave points_worst  symmetry_worst  \
id
842302             0.7119                0.2654          0.4601
842517             0.2416                0.1860          0.2750
84300903           0.4504                0.2430          0.3613
84348301           0.6869                0.2575          0.6638
84358402           0.4000                0.1625          0.2364

          fractal_dimension_worst  Unnamed: 32
id
842302                    0.11890          NaN
842517                    0.08902          NaN
84300903                  0.08758          NaN
84348301                  0.17300          NaN
84358402                  0.07678          NaN

[5 rows x 32 columns]
```

[4]:
```
# Print Summary
print('Shape ------------->',data.shape)
print('Each Column and data type and its count','\n')
print(data.info())
```

```
Shape -------------> (569, 32)
Each Column and data type and its count

<class 'pandas.core.frame.DataFrame'>
Int64Index: 569 entries, 842302 to 92751
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   diagnosis                569 non-null    object
```

```
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
 31  Unnamed: 32              0 non-null      float64
dtypes: float64(31), object(1)
memory usage: 146.7+ KB
None
```

[5]:
```
data = data.drop(['Unnamed: 32'], axis =1 )
```

[6]:
```
data.describe()
```

[6]:

	radius_mean	texture_mean	perimeter_mean	area_mean	\
count	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	
std	3.524049	4.301036	24.298981	351.914129	
min	6.981000	9.710000	43.790000	143.500000	
25%	11.700000	16.170000	75.170000	420.300000	
50%	13.370000	18.840000	86.240000	551.100000	
75%	15.780000	21.800000	104.100000	782.700000	

```
max      28.110000     39.280000      188.500000  2501.000000

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200

       symmetry_mean  fractal_dimension_mean  …  radius_worst  \
count     569.000000              569.000000  …    569.000000
mean        0.181162                0.062798  …     16.269190
std         0.027414                0.007060  …      4.833242
min         0.106000                0.049960  …      7.930000
25%         0.161900                0.057700  …     13.010000
50%         0.179200                0.061540  …     14.970000
75%         0.195700                0.066120  …     18.790000
max         0.304000                0.097440  …     36.040000

       texture_worst  perimeter_worst   area_worst  smoothness_worst  \
count     569.000000       569.000000   569.000000        569.000000
mean       25.677223       107.261213   880.583128          0.132369
std         6.146258        33.602542   569.356993          0.022832
min        12.020000        50.410000   185.200000          0.071170
25%        21.080000        84.110000   515.300000          0.116600
50%        25.410000        97.660000   686.500000          0.131300
75%        29.720000       125.400000  1084.000000          0.146000
max        49.540000       251.200000  4254.000000          0.222600

       compactness_worst  concavity_worst  concave points_worst  \
count         569.000000       569.000000            569.000000
mean            0.254265         0.272188              0.114606
std             0.157336         0.208624              0.065732
min             0.027290         0.000000              0.000000
25%             0.147200         0.114500              0.064930
50%             0.211900         0.226700              0.099930
75%             0.339100         0.382900              0.161400
max             1.058000         1.252000              0.291000

       symmetry_worst  fractal_dimension_worst
count      569.000000               569.000000
mean         0.290076                 0.083946
std          0.061867                 0.018061
min          0.156500                 0.055040
```

```
25%          0.250400                0.071460
50%          0.282200                0.080040
75%          0.317900                0.092080
max          0.663800                0.207500

[8 rows x 30 columns]
```

[7]:
```
data.isna().sum()
```

[7]:
```
diagnosis                0
radius_mean              0
texture_mean             0
perimeter_mean           0
area_mean                0
smoothness_mean          0
compactness_mean         0
concavity_mean           0
concave points_mean      0
symmetry_mean            0
fractal_dimension_mean   0
radius_se                0
texture_se               0
perimeter_se             0
area_se                  0
smoothness_se            0
compactness_se           0
concavity_se             0
concave points_se        0
symmetry_se              0
fractal_dimension_se     0
radius_worst             0
texture_worst            0
perimeter_worst          0
area_worst               0
smoothness_worst         0
compactness_worst        0
concavity_worst          0
concave points_worst     0
symmetry_worst           0
fractal_dimension_worst  0
dtype: int64
```

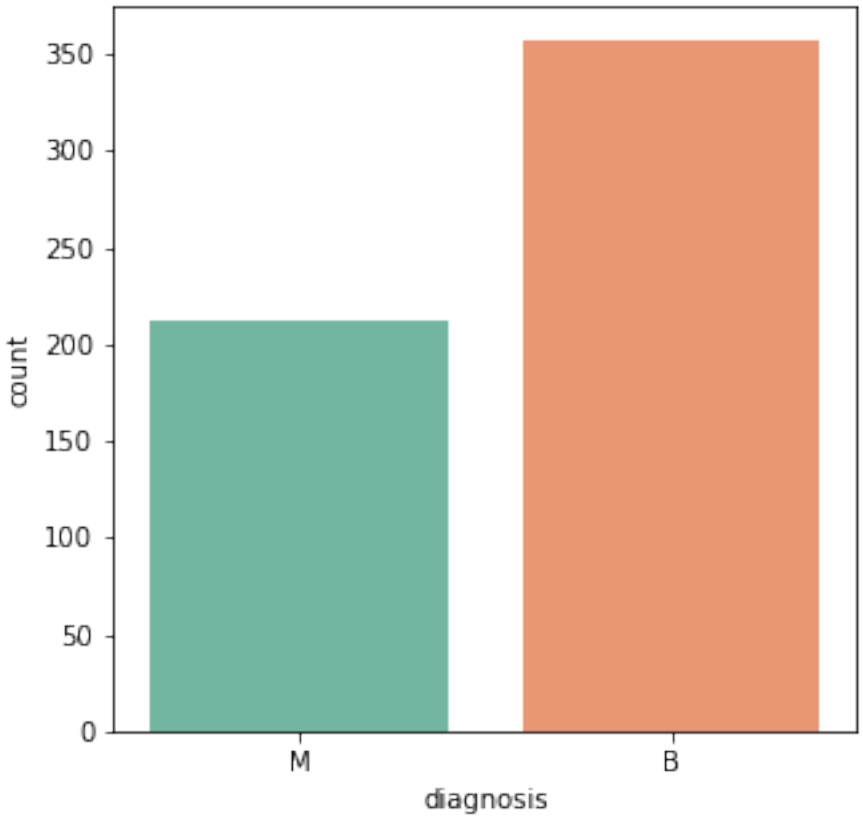
Seems non of the columns have nul values in it. It's safe to proceed

[11]:
```
data.diagnosis.value_counts()
```

[11]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

Data is not imbalanced,we are good to proceed

[14]:
```
plt.figure(figsize=(5,5))
sns.countplot(x='diagnosis', data = data, palette ='Set2')
plt.show()
```



[16]:
```
data.shape
```

[16]:
```
(569, 31)
```

1 Using Select KBest feature Selection method

1.0.1 Select KBest use f_classif function to find best features, where f_classif uses ANOVA test.

[ ]:
```
from sklearn.feature_selection import SelectKBest , f_classif
```

[17]:
```
# Replace Label column (diagnosis) into binary codes
data['diagnosis'] = data['diagnosis'].replace({'M':1,'B':0})
```

[18]:
```
x = data.drop('diagnosis', axis =1)
y = data.diagnosis
```

[20]:
```
best_features = SelectKBest(score_func = f_classif , k = 15) # k=15 means how many features do i want

fit = best_features.fit(x,y)

data_score = pd.DataFrame(fit.scores_)

data_columns = pd.DataFrame(x.columns)
```

[31]:
```
# Concatenate dataframes

feature_scores = pd.concat([data_columns,data_score], axis=1)

feature_scores.columns = ['Feature_Name','Score'] # name output columns

print(feature_scores.nlargest(15,'Score'))  # print 15 best features (nlargest means starting from the largest number)

# Export selected features to .csv for later use
# data_backup = feature_scores.nlargest(15,'Score')
# data_backup.to_csv('Selected_features.csv', index = False)
```

```
            Feature_Name       Score
27  concave points_worst  964.385393
22       perimeter_worst  897.944219
7     concave points_mean  861.676020
20          radius_worst  860.781707
2         perimeter_mean  697.235272
23            area_worst  661.600206
0           radius_mean  646.981021
3             area_mean  573.060747
6         concavity_mean  533.793126
26       concavity_worst  436.691939
5       compactness_mean  313.233079
25     compactness_worst  304.341063
```

```
10              radius_se  268.840327
12           perimeter_se  253.897392
13                area_se  243.651586
```

[33]:
```
new_x = data[['concave points_worst','perimeter_worst','concave␣
↪points_mean','radius_worst','perimeter_mean','area_worst','radius_mean','area_mean','concav
```

[34]:
```
new_x
```

[34]:

id	concave points_worst	perimeter_worst	concave points_mean	\
842302	0.2654	184.60	0.14710	
842517	0.1860	158.80	0.07017	
84300903	0.2430	152.50	0.12790	
84348301	0.2575	98.87	0.10520	
84358402	0.1625	152.20	0.10430	
…	…	…	…	
926424	0.2216	166.10	0.13890	
926682	0.1628	155.00	0.09791	
926954	0.1418	126.70	0.05302	
927241	0.2650	184.60	0.15200	
92751	0.0000	59.16	0.00000	

id	radius_worst	perimeter_mean	area_worst	radius_mean	area_mean	\
842302	25.380	122.80	2019.0	17.99	1001.0	
842517	24.990	132.90	1956.0	20.57	1326.0	
84300903	23.570	130.00	1709.0	19.69	1203.0	
84348301	14.910	77.58	567.7	11.42	386.1	
84358402	22.540	135.10	1575.0	20.29	1297.0	
…	…	…	…	…	…	
926424	25.450	142.00	2027.0	21.56	1479.0	
926682	23.690	131.20	1731.0	20.13	1261.0	
926954	18.980	108.30	1124.0	16.60	858.1	
927241	25.740	140.10	1821.0	20.60	1265.0	
92751	9.456	47.92	268.6	7.76	181.0	

id	concavity_mean	concavity_worst	compactness_mean	\
842302	0.30010	0.7119	0.27760	
842517	0.08690	0.2416	0.07864	
84300903	0.19740	0.4504	0.15990	
84348301	0.24140	0.6869	0.28390	
84358402	0.19800	0.4000	0.13280	
…	…	…	…	
926424	0.24390	0.4107	0.11590	
926682	0.14400	0.3215	0.10340	

```
926954          0.09251          0.3403          0.10230
927241          0.35140          0.9387          0.27700
92751           0.00000          0.0000          0.04362

          compactness_worst  radius_se  perimeter_se  area_se
id
842302              0.66560     1.0950         8.589   153.40
842517              0.18660     0.5435         3.398    74.08
84300903            0.42450     0.7456         4.585    94.03
84348301            0.86630     0.4956         3.445    27.23
84358402            0.20500     0.7572         5.438    94.44
…                     …         …              …        …
926424              0.21130     1.1760         7.673   158.70
926682              0.19220     0.7655         5.203    99.04
926954              0.30940     0.4564         3.425    48.55
927241              0.86810     0.7260         5.772    86.22
92751               0.06444     0.3857         2.548    19.15

[569 rows x 15 columns]
```

[43]:
```
scaler = StandardScaler()
x_scalar = scaler.fit_transform(new_x)
```

[44]:
```
from time import time

# Buidling model to test unexposed data
x_train,x_test,y_train,y_test = train_test_split(x_scalar,y , test_size = 0.25␣
 ↪, random_state = 55)
```

[45]:
```
knn =KNeighborsClassifier()
```

[46]:
```
knn =KNeighborsClassifier()

# Checking training and testing time (Lazy Learner)
start = time()
knn.fit(x_train,y_train)
print("Knn training time: ",(time() - start))

start = time()
y_pred = knn.predict(x_test)
print("Knn testing time: ",(time() - start))
```

```
Knn training time:  0.01599597930908203
Knn testing time:  0.0400080680847168
```

[48]:
```
cm = confusion_matrix(y_test,y_pred)
cm
```

[48]:
```
array([[79,  8],
       [ 3, 53]], dtype=int64)
```

[51]:
```
print(classification_report(y_test,y_pred,digits = 2))
```

```
              precision    recall  f1-score   support

           0       0.96      0.91      0.93        87
           1       0.87      0.95      0.91        56

    accuracy                           0.92       143
   macro avg       0.92      0.93      0.92       143
weighted avg       0.93      0.92      0.92       143
```

2 Cross Validation score to check if the model is overfitting

[52]:
```
cross_val_score(knn,x_scalar,y , cv =10).mean()
# cv =10 means how many timed do i want to cross check the model u can change␣
 ↪it aswell
# .mean() so it will give us the average mean of all the accuracies (score)
```

[52]:
```
0.9350250626566415
```

3 Hyperparameter Tuning

3.1 Let's use GridSearchCV for the best parameter to improve the accuracy

[54]:
```
param_grid = {'algorithm' : ['kd_tree','brute'],
              'leaf_size' : [3,5,6,7,9],
              'n_neighbors' :[3,5,7,9,11]}
```

[55]:
```
gridsearch = GridSearchCV(estimator = knn , param_grid = param_grid)
```

[57]:
```
# Training the model
gridsearch.fit(x_train,y_train)
```

[57]:
```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['kd_tree', 'brute'],
                         'leaf_size': [3, 5, 6, 7, 9],
                         'n_neighbors': [3, 5, 7, 9, 11]})
```

[58]:
```
# Printing the best parameters
gridsearch.best_params_
```

[58]:
```
{'algorithm': 'kd_tree', 'leaf_size': 3, 'n_neighbors': 5}
```

[61]:
```
# we will use the best parameter in our k-NN algorithm and check if accuracy is␣
↪increasing.
knn = KNeighborsClassifier(algorithm = 'kd_tree', leaf_size= 3, n_neighbors= 5)
```

[62]:
```
knn.fit(x_train,y_train)
```

[62]:
```
KNeighborsClassifier(algorithm='kd_tree', leaf_size=3)
```

[63]:
```
y_pred = knn.predict(x_test)
```

[64]:
```
cm = confusion_matrix(y_test,y_pred)
```

[65]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.91      0.93        87
           1       0.87      0.95      0.91        56

    accuracy                           0.92       143
   macro avg       0.92      0.93      0.92       143
weighted avg       0.93      0.92      0.92       143
```

[ ]:

[ ]: