A2_template_2022

May 17, 2022

1 Assignment 2: Classification

2 Using Machine Learning Tools CS3317

2.1 Overview

In this assignment, you will apply some popular machine learning techniques to the problem of classifying data from histological cell images for the diagnosis of malignant breast cancer. This will be presented as a practical scenario where you are approached by a client to solve a problem.

The main aims of this assignment are:

- to use the best practice machine learning workflow for producing a solution to a client's problem;
- to visualise data and determine the best pre-processing;
- to create the necessary datasets for training and testing purposes;
- to train and optimise a selection of models, then choose the best;
- to obtain an unbiased measurement of the final model's performance;
- to interpret results clearly and concisely.

This assignment relates to the following ACS CBOK areas: abstraction, design, hardware and software, data and information, HCI and programming.

2.2 General instructions

This assignment is divided into several tasks. Use the spaces provided in this notebook to answer the questions posed in each task. Note that some questions require writing a small amount of code, some require graphical results, and some require comments or analysis as text. It is your responsibility to make sure your responses are clearly labelled and your code has been fully executed (**with the correct results displayed**) before submission!

Do not manually edit the data set file we have provided! For marking purposes, it's important that your code runs correctly on the original data file.

Some of the parts of this assignment build on the workflow from the first assignment and that part of the course, and so less detailed instructions are provided for this, as you should be able to implement this workflow now without low-level guidance. A substantial portion of the marks for this assignment are associated with making the right choices and executing this workflow correctly and efficiently. Make sure you have clean, readable code as well as producing outputs, since your coding will also count towards the marks (however, excessive commenting is discouraged and will lose marks, so aim for a modest, well-chosen amount of comments and text in outputs).

This assignment can be solved using methods from sklearn, pandas, and matplotlib as presented in the workshops. Other libraries should not be used (even though they might have nice functionality) and certain restrictions on sklearn functions will be made clear in the instruction text. You are expected to search and carefully read the documentation for functions that you use, to ensure you are using them correctly.

3 Scenario

A client approaches you to solve a machine learning problem for them. They run a pathology lab that processes histological images for healthcare providers and they have created a product that measures the same features as in the *Wisconsin breast cancer data set* though using different acquisitions and processing methods. This makes their method much faster than existing ones, but it is also slightly noisier. They want to be able to diagnose *malignant* cancer (and distinguish them from *benign* growths) by employing machine learning techniques, and they have asked you to implement this for them.

Their requirements are: 1) have at least a 95% probability of detecting malignant cancer when it is present; 2) have no more than 1 in 10 healthy cases (those with benign tumours) labelled as positive (malignant).

They have hand-labelled 300 samples for you, which is all they have at the moment.

Please follow the instructions below, which will vary in level of detail, as appropriate to the marks given.

3.1 1. Investigate Dataset (10% = 3 marks)

[1]:
```
# This code imports some libraries that you will need.
# You should not need to modify it, though you are expected to make other␣
 ↪imports later in your code.

# Python 3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Common imports
import numpy as np
import time

# Pandas for overview
import pandas as pd

# Scikit-Learn 0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"
from sklearn import tree
from sklearn import svm
from sklearn.pipeline import Pipeline
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix

# Plot setup
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=7)
mpl.rc('xtick', labelsize=6)
mpl.rc('ytick', labelsize=6)
mpl.rc('figure', dpi=240)
plt.close('all')

import seaborn as sns
```

3.1.1 1.1 Load the dataset [0.5 marks]

Do this from the csv file, `assignment2.csv`, as done in assignment 1 and workshops 2 and 3. Extract the feature names and label names for use later on. Note that we will be treating the *malignant* case as our *positive* case, as this is the standard convention in medicine.

Print out some information (in text) about the data, to verify that the loading has worked and to get a feeling for what is present in the dataset and the range of the values.

Also, graphically show the proportions of the labels in the whole dataset.

[2]:
```
# Your code here
df = pd.read_csv("assignment2.csv")
print(df.head())
```

```
       label  mean radius  mean texture  mean perimeter   mean area  \
0  malignant    15.494654     15.902542      103.008265  776.437239
1  malignant    16.229871     18.785613      105.176755  874.712003
2  malignant    16.345671     20.114076      107.083804  872.563251
3  malignant    13.001009     19.876997       85.889775  541.281012
4  malignant    16.416060     17.397533      107.857386  891.516818

   mean smoothness  mean compactness  mean concavity  mean concave points  \
0         0.104239          0.168660        0.170572             0.085668
1         0.091843          0.092548        0.081681             0.053670
2         0.099924          0.123799        0.128788             0.078310
3         0.113423          0.173069        0.146214             0.069574
4         0.097321          0.111530        0.125971             0.068575

   mean symmetry  …  worst radius  worst texture  worst perimeter  \
0       0.205053  …     19.522957      22.427276       135.128520
1       0.180435  …     19.140235      24.905156       123.886045
2       0.189756  …     19.144816      25.601433       125.113036
```

```
3       0.212078  …     15.565911      26.145119       102.958265
4       0.179562  …     18.620376      22.306233       124.002529

    worst area  worst smoothness  worst compactness  worst concavity  \
0  1286.903131          0.142725           0.407483         0.445992
1  1234.499997          0.129135           0.223918         0.248846
2  1202.749973          0.135017           0.314402         0.332505
3   737.655082          0.161390           0.485912         0.430007
4  1139.490971          0.133950           0.230996         0.316620

   worst concave points  worst symmetry  worst fractal dimension
0              0.171662        0.353211                 0.097731
1              0.136735        0.284427                 0.085758
2              0.161497        0.313038                 0.084340
3              0.167254        0.432297                 0.117705
4              0.131715        0.269591                 0.080497

[5 rows x 31 columns]
```

[3]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300 entries, 0 to 299
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   label                    300 non-null    object
 1   mean radius              300 non-null    float64
 2   mean texture             300 non-null    float64
 3   mean perimeter           300 non-null    float64
 4   mean area                300 non-null    float64
 5   mean smoothness          300 non-null    float64
 6   mean compactness         300 non-null    float64
 7   mean concavity           300 non-null    float64
 8   mean concave points      300 non-null    float64
 9   mean symmetry            300 non-null    float64
 10  mean fractal dimension   300 non-null    float64
 11  radius error             300 non-null    float64
 12  texture error            300 non-null    float64
 13  perimeter error          300 non-null    float64
 14  area error               300 non-null    float64
 15  smoothness error         300 non-null    float64
 16  compactness error        300 non-null    float64
 17  concavity error          300 non-null    float64
 18  concave points error     300 non-null    float64
 19  symmetry error           300 non-null    float64
 20  fractal dimension error  300 non-null    float64
 21  worst radius             300 non-null    float64
```

```
 22  worst texture            300 non-null    float64
 23  worst perimeter          300 non-null    float64
 24  worst area               300 non-null    float64
 25  worst smoothness         300 non-null    float64
 26  worst compactness        300 non-null    float64
 27  worst concavity          300 non-null    float64
 28  worst concave points     300 non-null    float64
 29  worst symmetry           300 non-null    float64
 30  worst fractal dimension  300 non-null    float64
dtypes: float64(30), object(1)
memory usage: 72.8+ KB
None
```

[4]:
```
print(df['label'].describe())
'''label column has two types: malignant/benign,
This shows it has 154 benigns so it has 146 miligants'''
```

```
count        300
unique         2
top       benign
freq         154
Name: label, dtype: object
```

[4]:
```
'label column has two types: malignant/benign,\nThis shows it has 154 benigns so
it has 146 miligants'
```

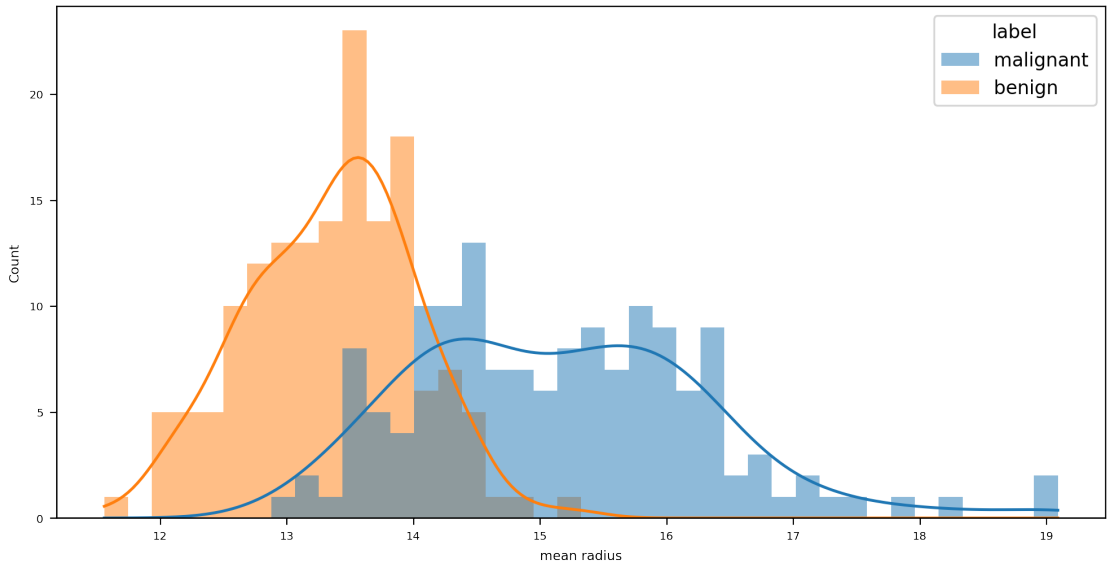
3.1.2 1.2 Visualise the dataset [1.5 marks]

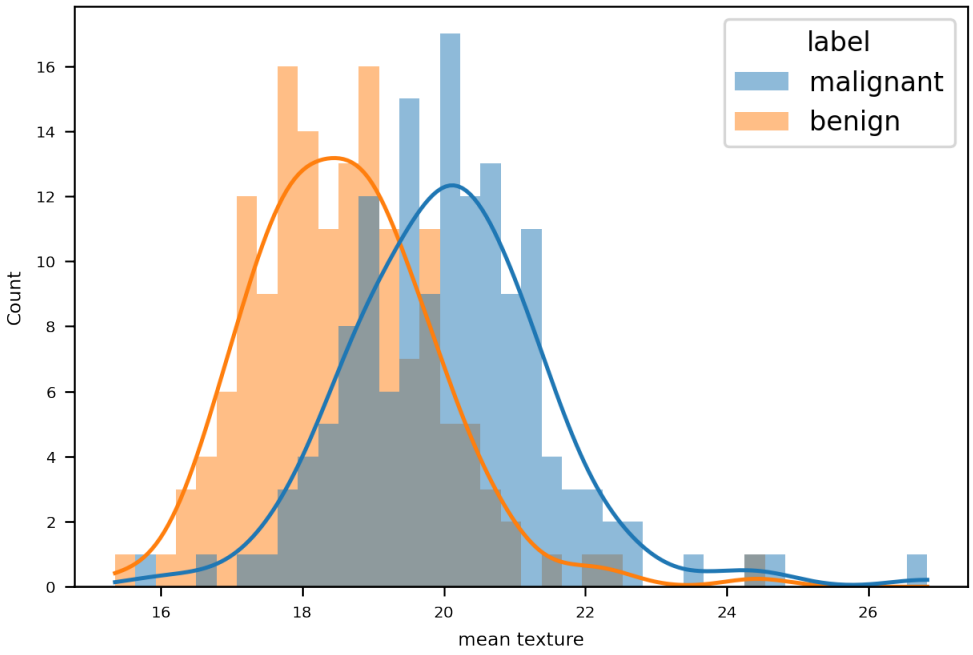
As this data is well curated by the client already, you do not need to worry about outliers, missing values or imputation in this case, but be aware that this is the exception, not the rule.

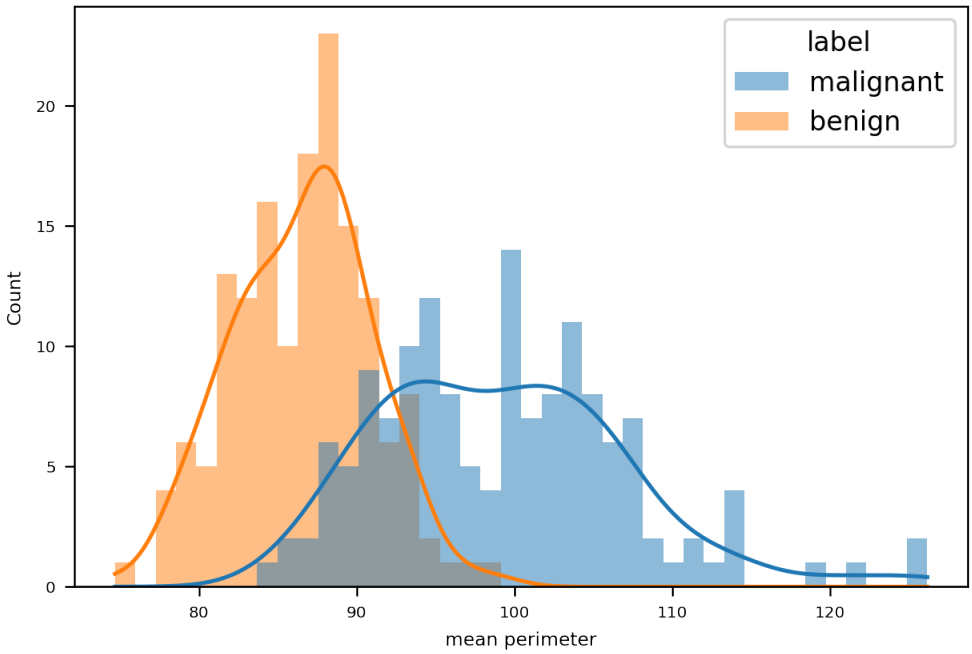
To familiarise yourself with the nature and information contained in the data, display histograms for the data according to the following instructions: - **display histograms** for each feature in the *mean* group, but on *each* histogram **have the two classes displayed together in one plot** (see example plot below and a code fragment to help you) - and note that your plot does not need to look exactly the example here; - **repeat this** for the *standard error* and *worst* groups; - make sure that in all cases you clearly label the plots and the classes in histograms.

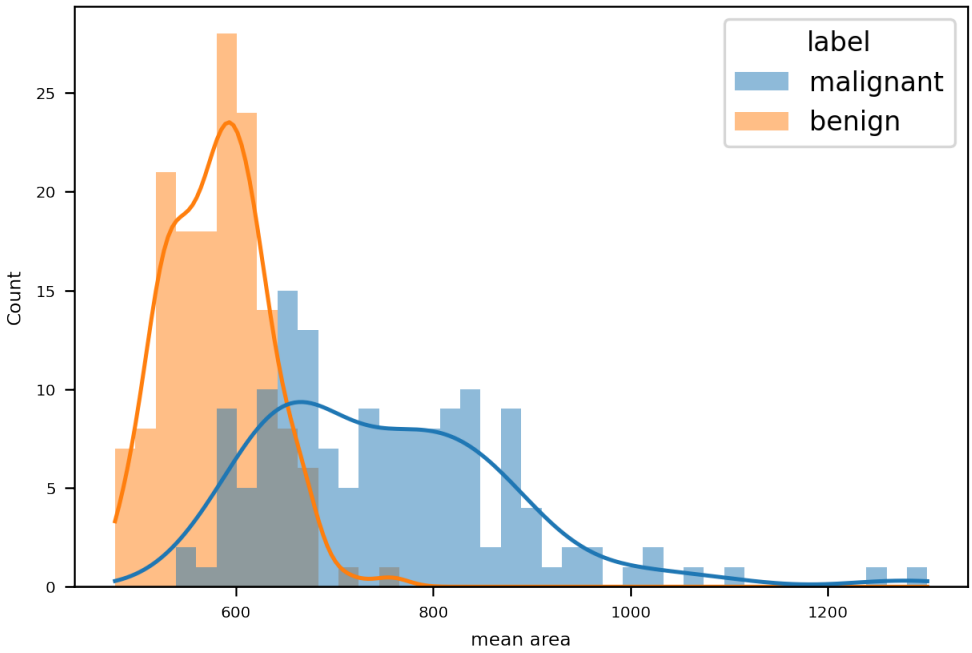
[5]:
```
# Code fragment to help with plotting histograms combining matplotlib and␣
 ↪seaborn (and pandas)

fig, axes = plt.subplots( figsize=(10,5))
# print(axes)
for col in df.columns:
#     fig, axes = plt.subplots( figsize=(10,5))
    if col != "label":
#         print(col)
```

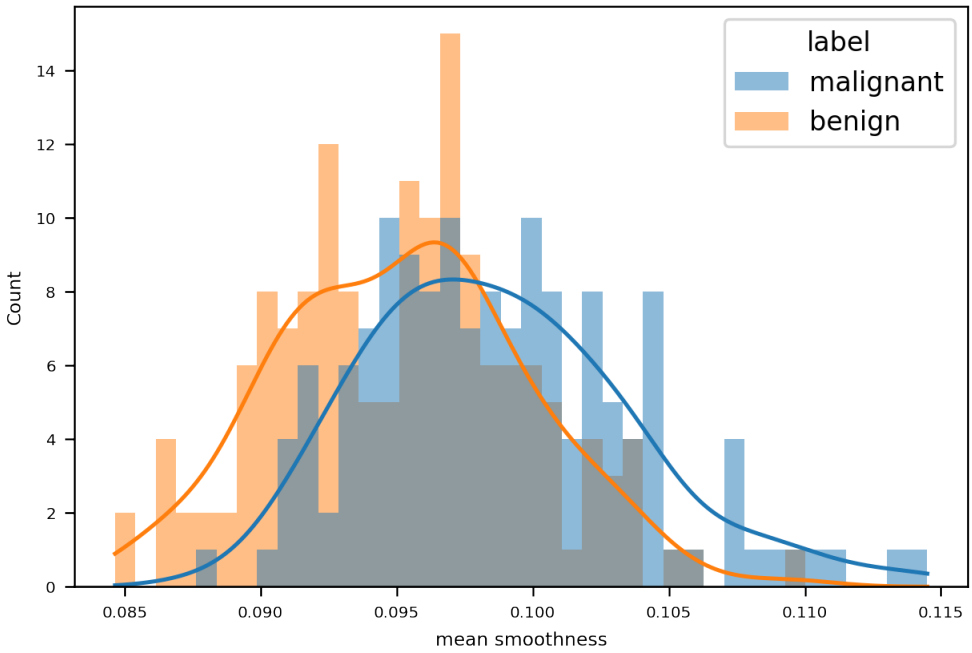
```
        n=sns.histplot(data=df, x=col,hue="label",bins=40, kde=True,␣
↪edgecolor=None)
        plt.show()
```

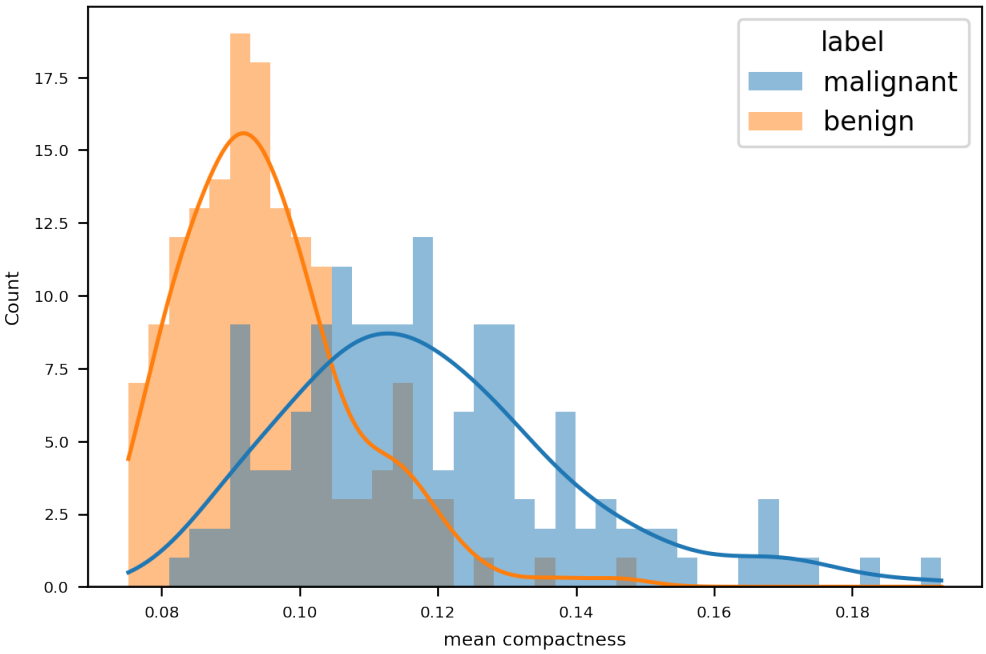


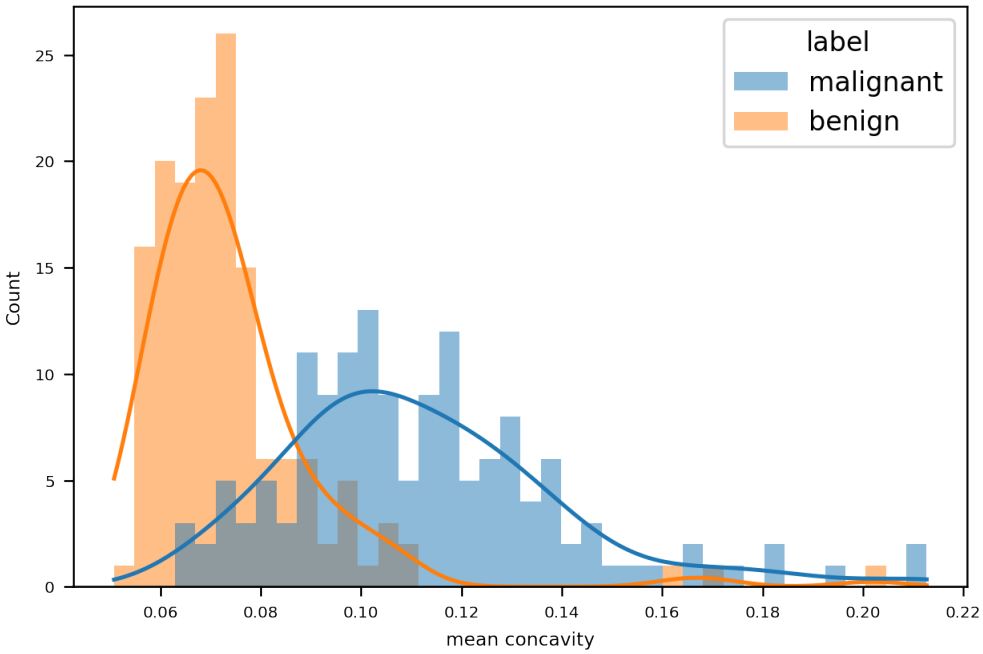


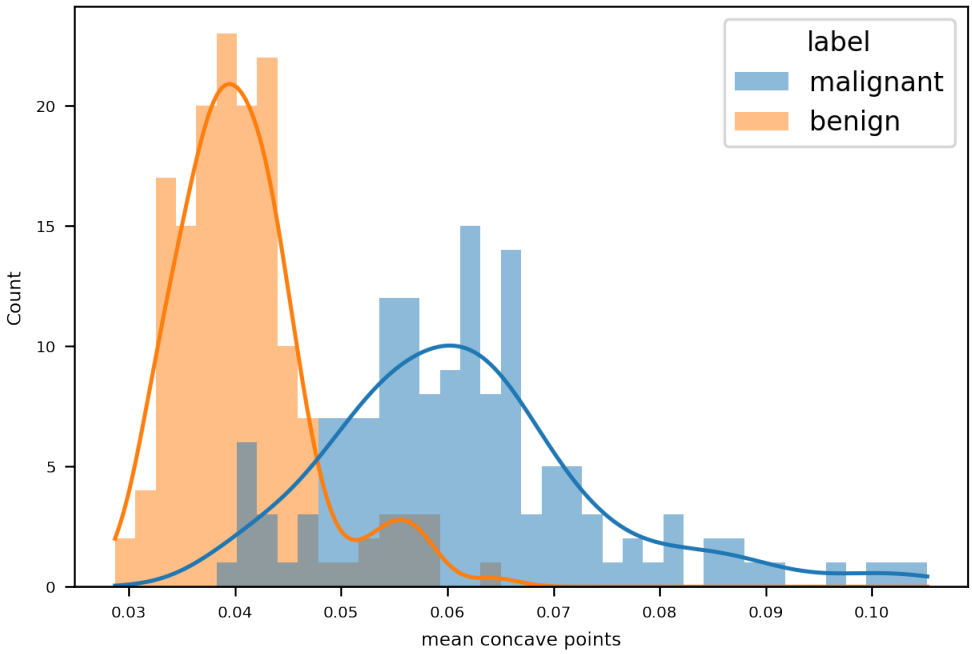

label
malignant
benign
Count
0
5
10
15
20
80
90
100
110
120
mean perimeter

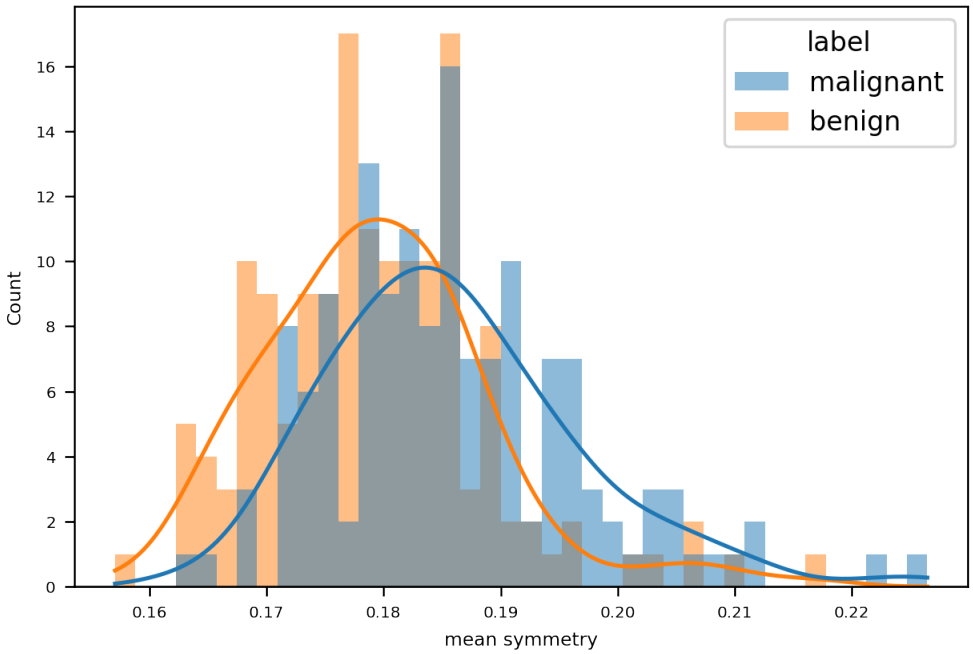

label
malignant
benign
Count
0
5
10
15
20
25
600
800
1000
1200
mean area

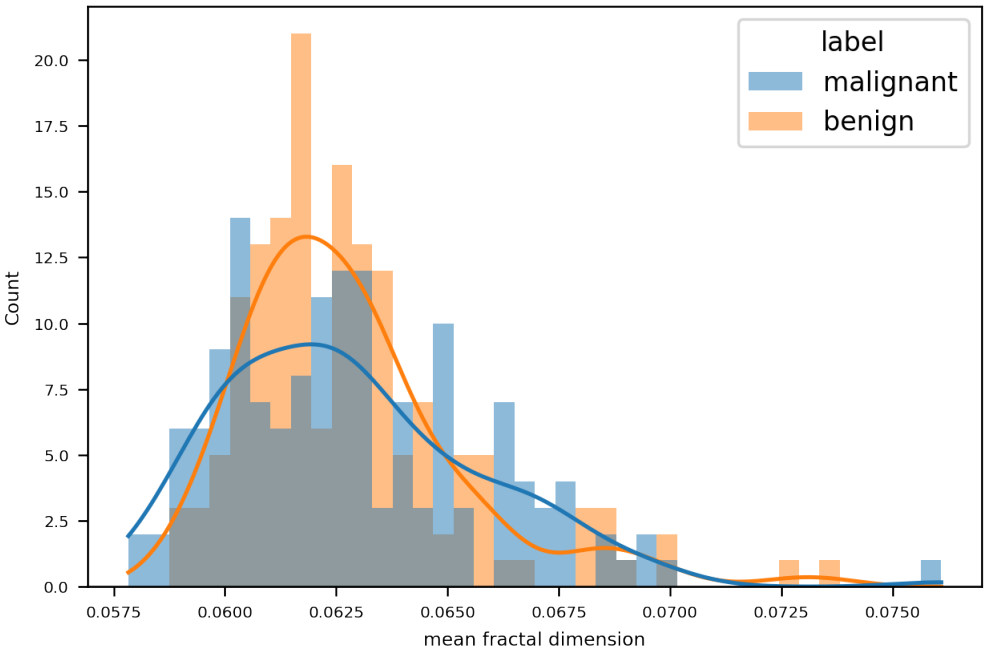

label
malignant
benign
Count
mean smoothness
0
2
4
6
8
10
12
14
0.085
0.090
0.095
0.100
0.105
0.110
0.115

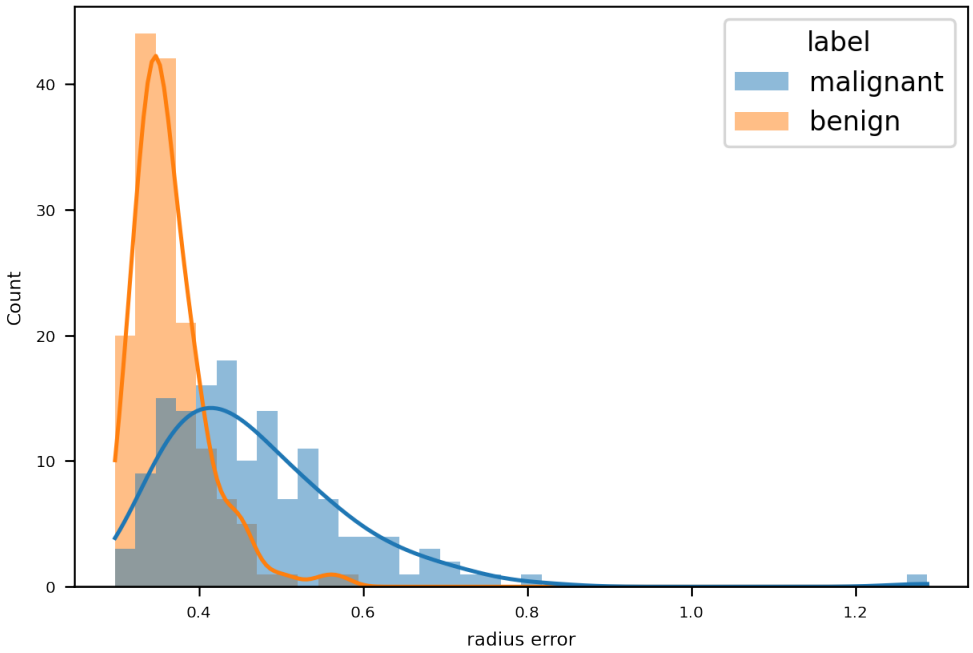

label
malignant
benign
Count
mean compactness
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.08
0.10
0.12
0.14
0.16
0.18

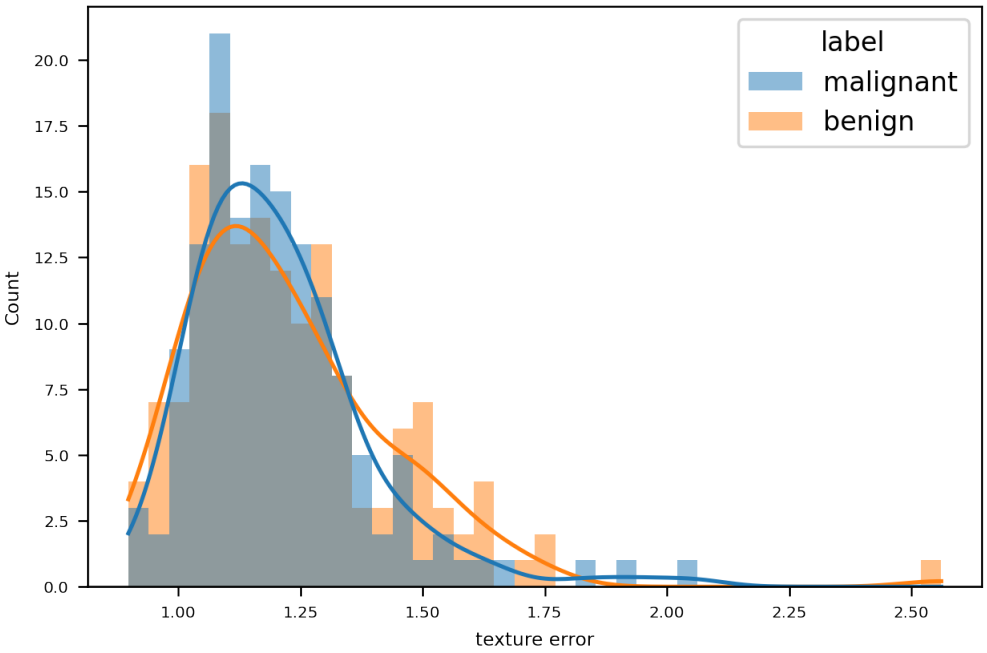

label
malignant
benign
Count
mean concavity
0
5
10
15
20
25
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20
0.22

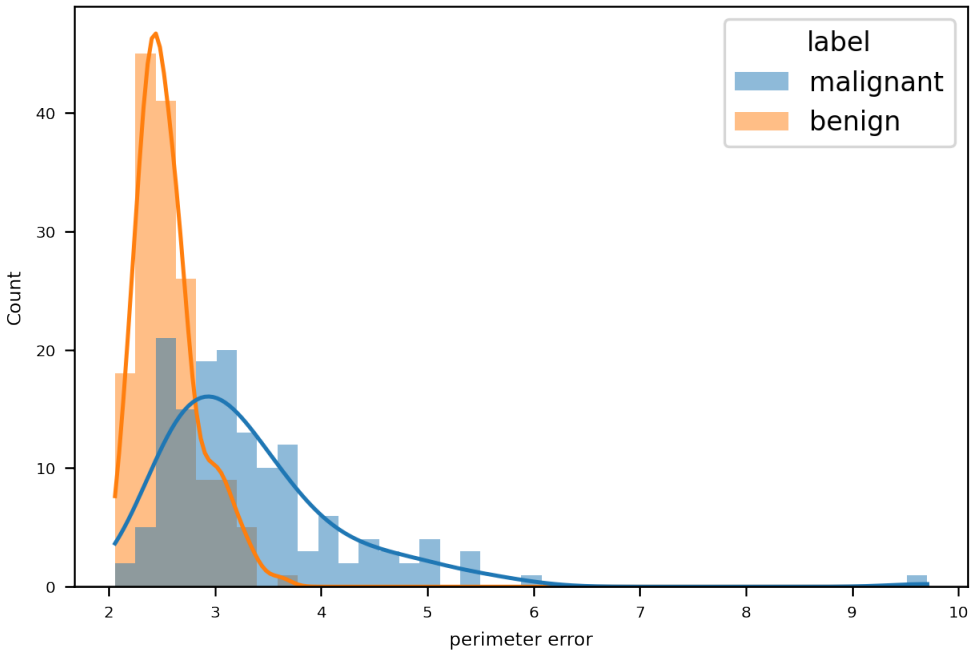

label
malignant
benign
Count
mean concave points
0
5
10
15
20
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.10

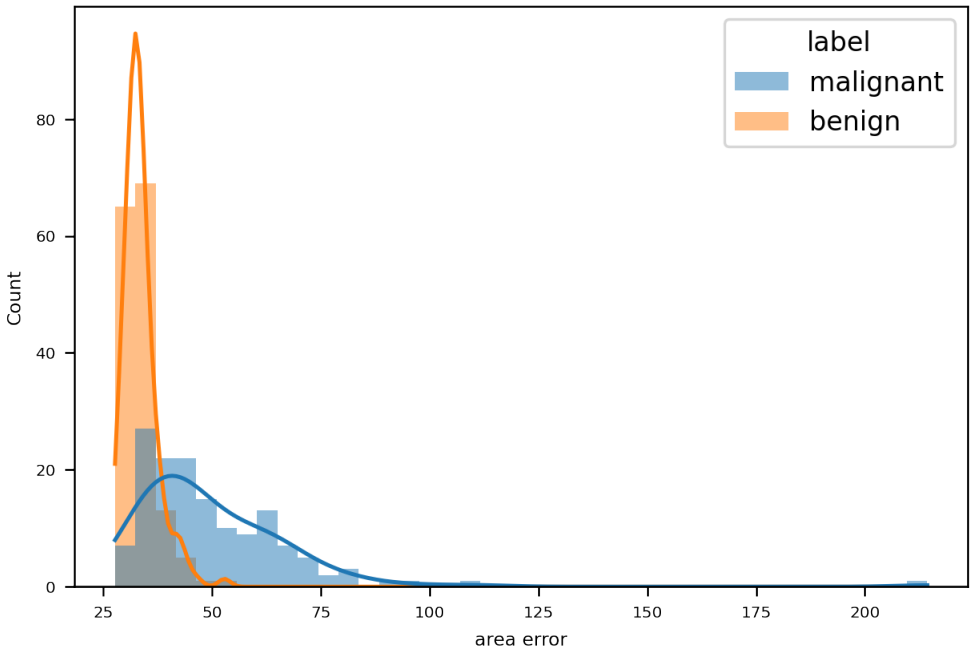

label
malignant
benign
Count
mean symmetry
0
2
4
6
8
10
12
14
16
0.16
0.17
0.18
0.19
0.20
0.21
0.22

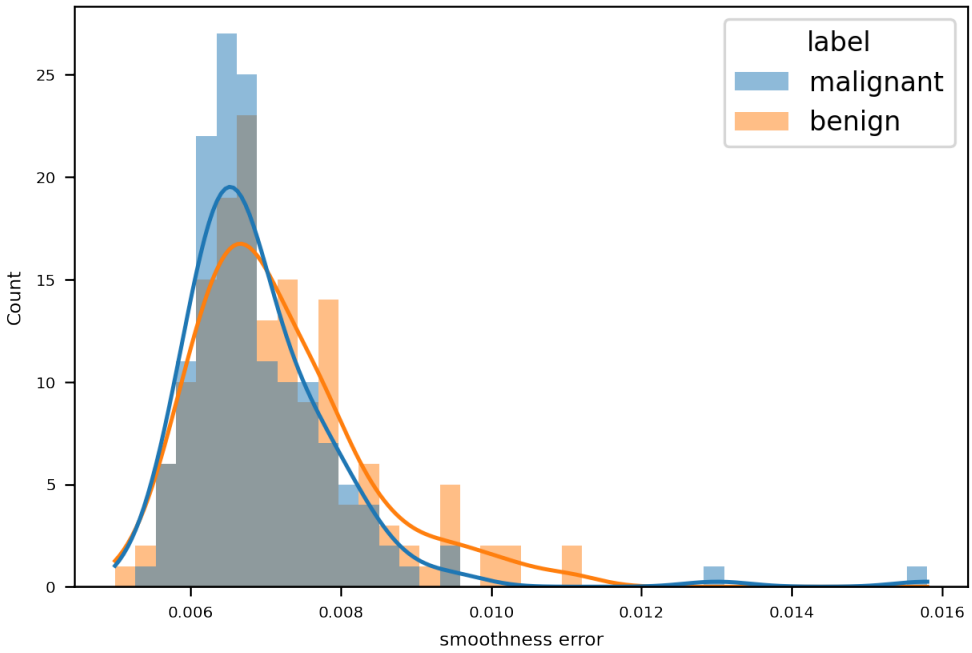

label
malignant
benign
Count
mean fractal dimension
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.0575
0.0600
0.0625
0.0650
0.0675
0.0700
0.0725
0.0750

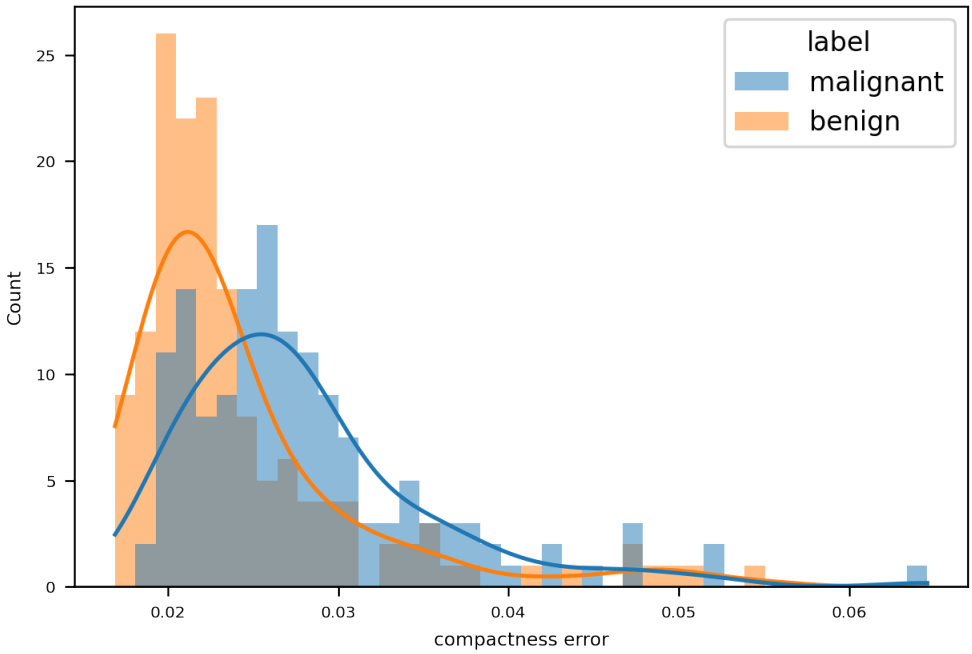

label
malignant
benign
Count
radius error
0
10
20
30
40
0.4
0.6
0.8
1.0
1.2

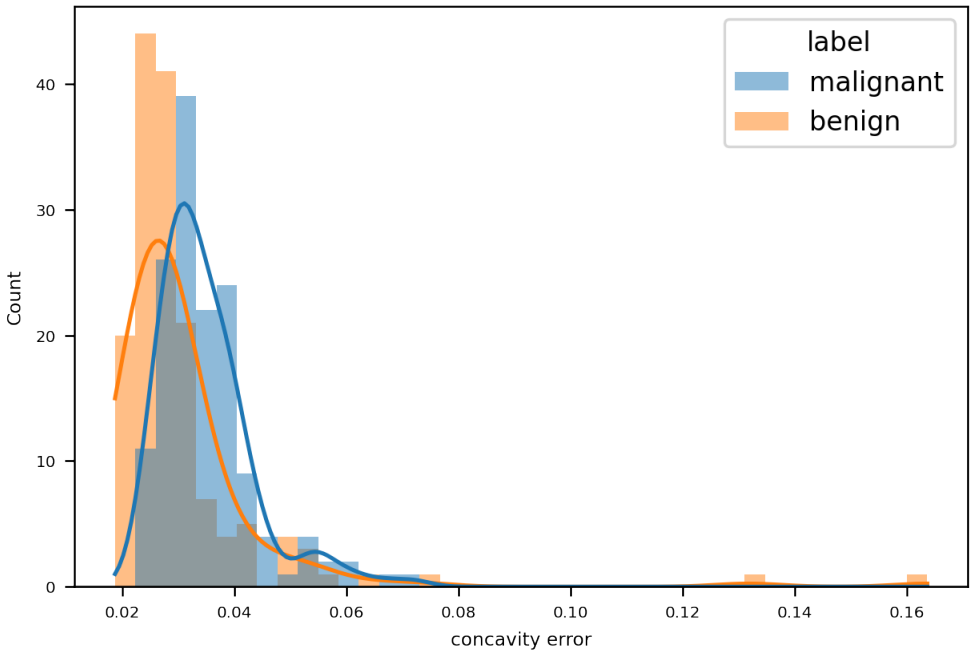

label
malignant
benign
Count
texture error
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
1.00
1.25
1.50
1.75
2.00
2.25
2.50

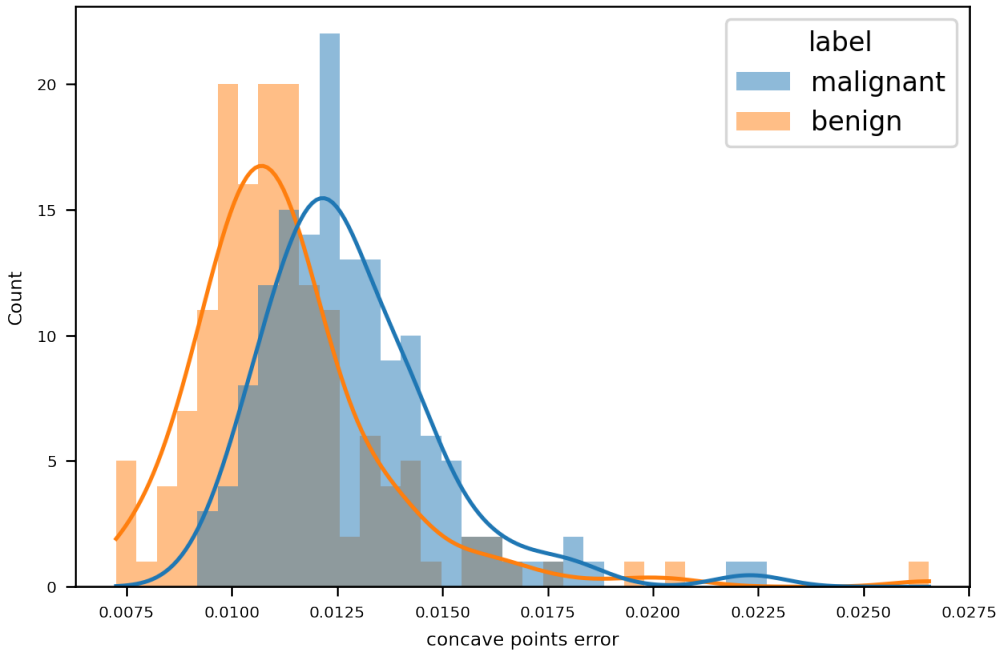

label
malignant
benign
Count
perimeter error
0
10
20
30
40
2
3
4
5
6
7
8
9
10

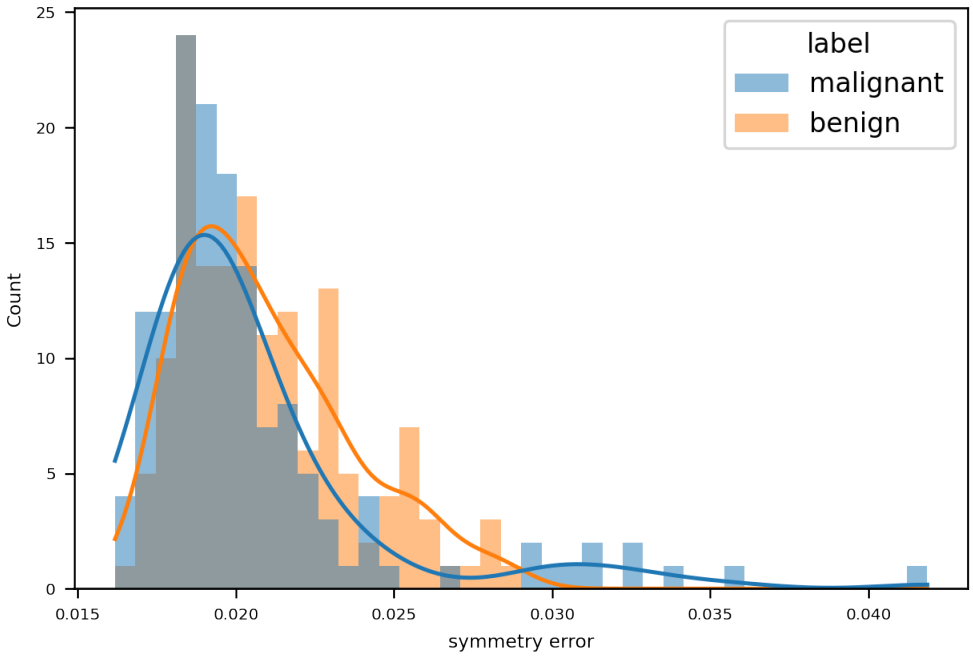

label
malignant
benign
Count
area error
0
20
40
60
80
25
50
75
100
125
150
175
200

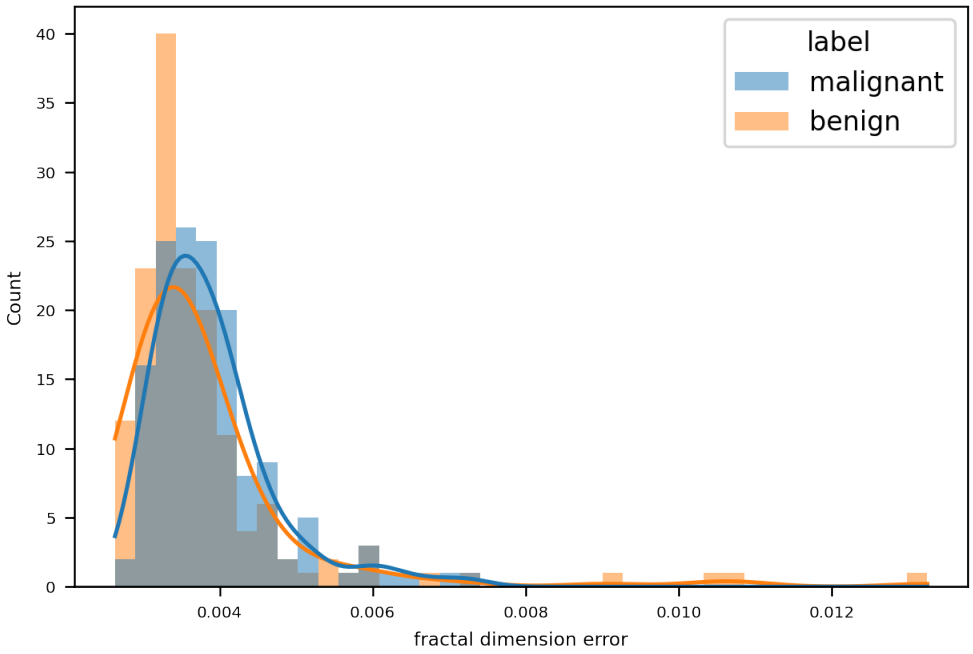

label
malignant
benign
Count
0
5
10
15
20
25
0.006
0.008
0.010
0.012
0.014
0.016
smoothness error

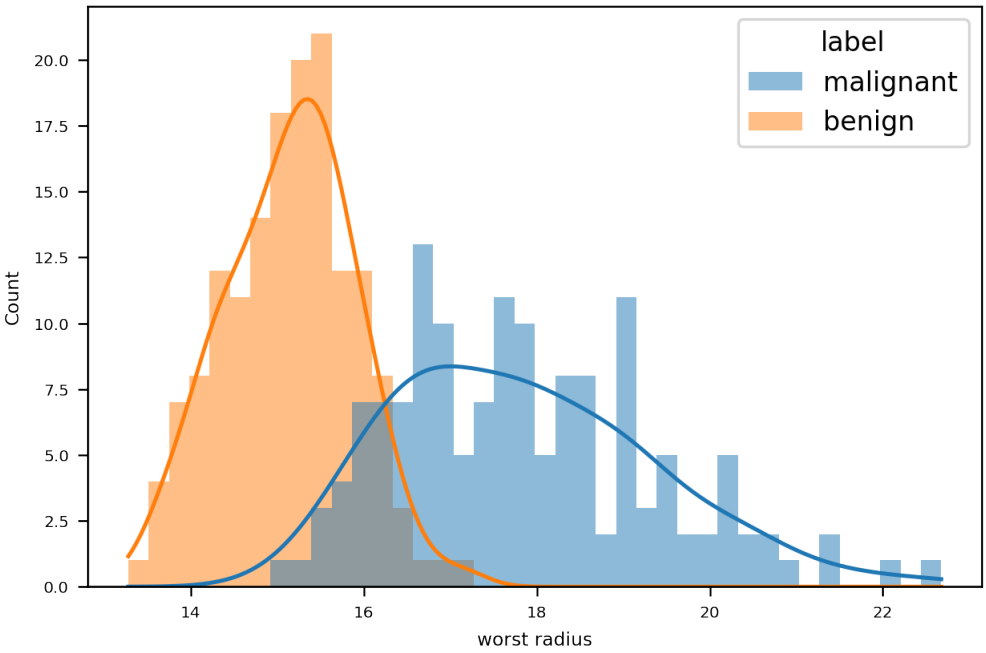

label
malignant
benign
Count
0
5
10
15
20
25
0.02
0.03
0.04
0.05
0.06
compactness error

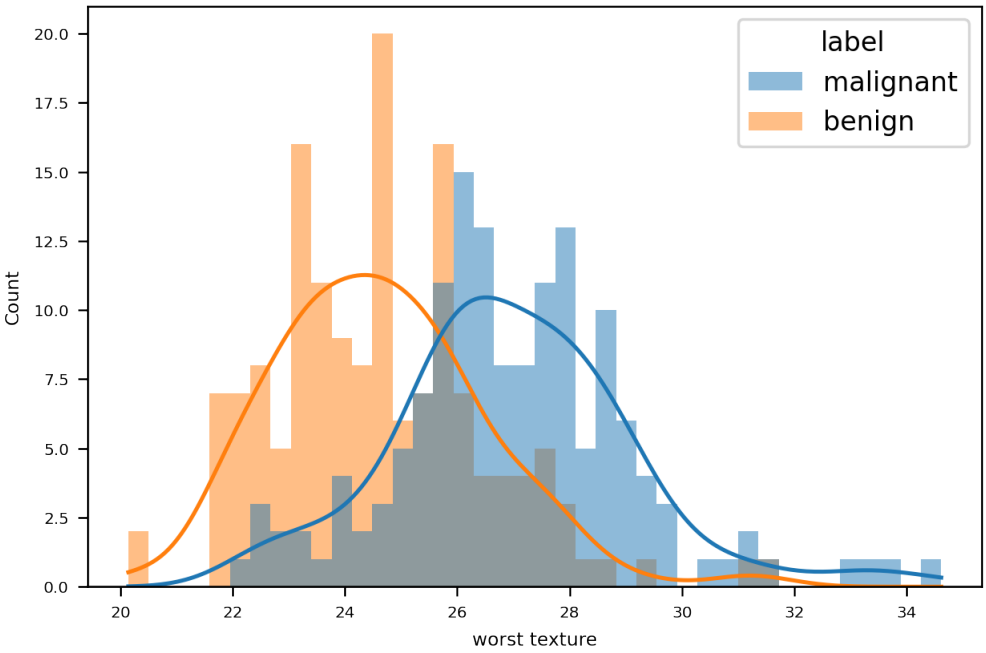

label
malignant
benign
Count
concavity error
0
10
20
30
40
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16

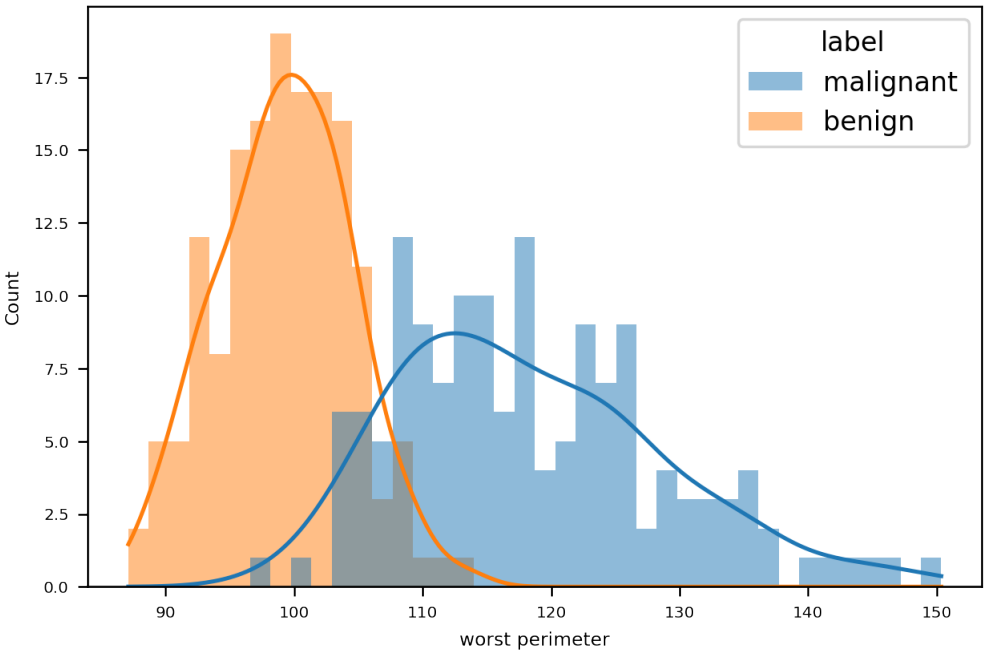

label
malignant
benign
Count
concave points error
0
5
10
15
20
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
0.0225
0.0250
0.0275

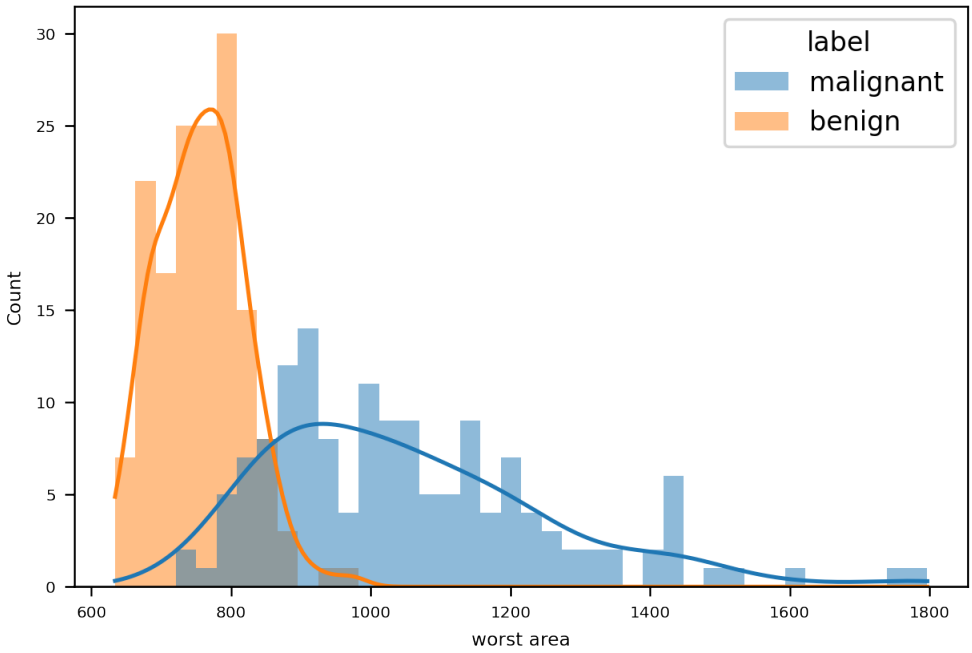

label
malignant
benign
Count
symmetry error
0
5
10
15
20
25
0.015
0.020
0.025
0.030
0.035
0.040

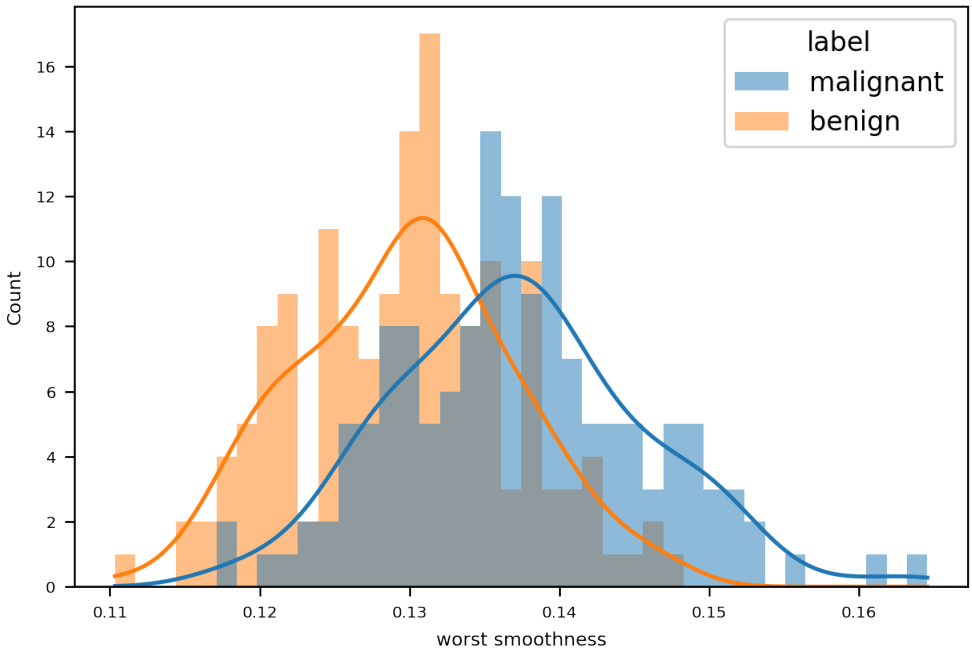

label
malignant
benign
Count
fractal dimension error
0
5
10
15
20
25
30
35
40
0.004
0.006
0.008
0.010
0.012

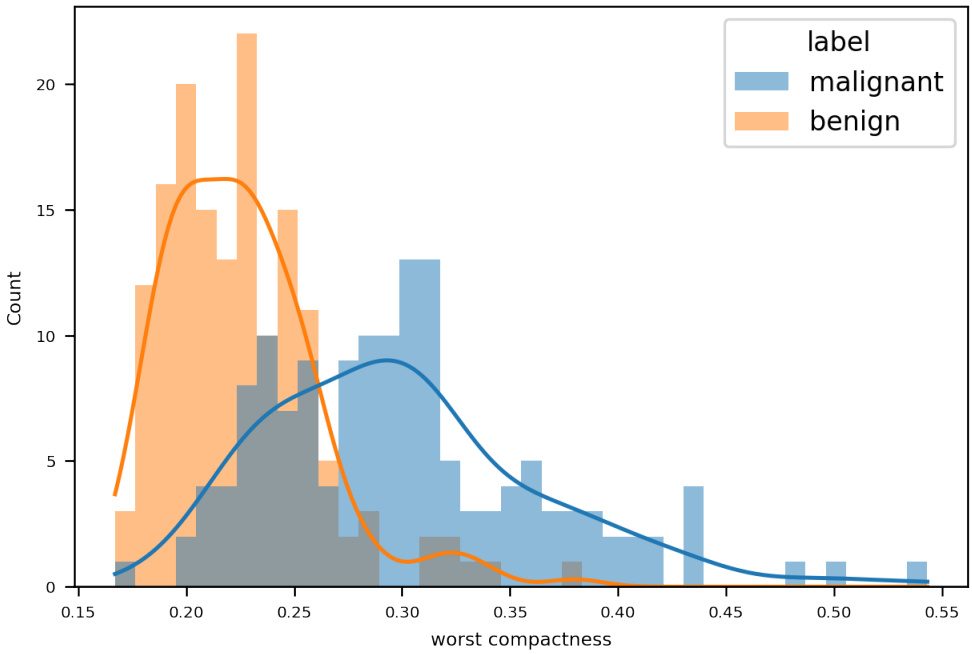

label
malignant
benign
Count
worst radius
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
14
16
18
20
22

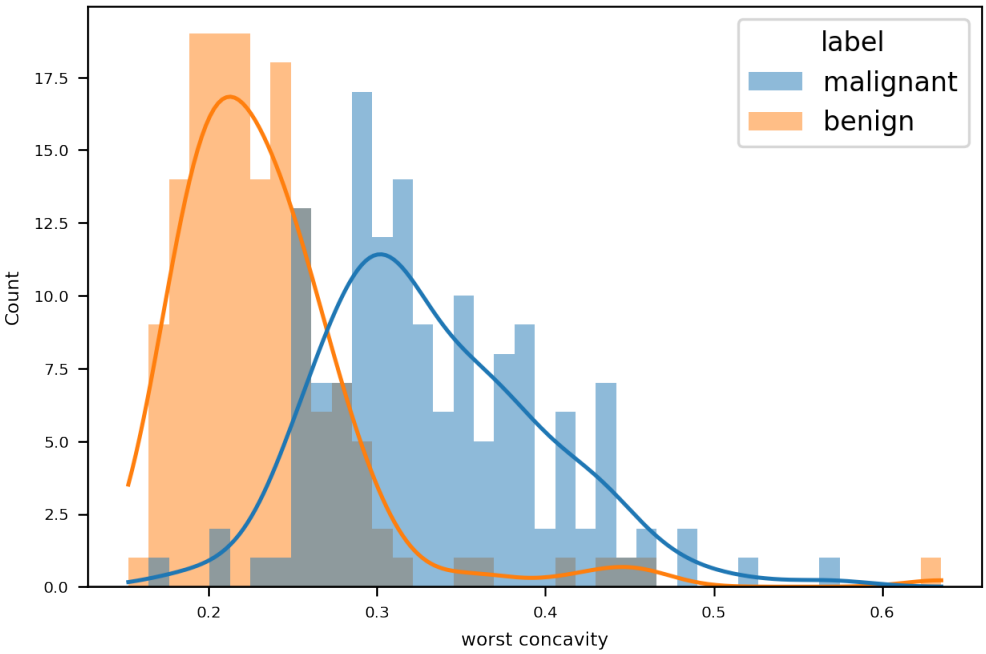

label
malignant
benign
Count
worst texture
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
20
22
24
26
28
30
32
34

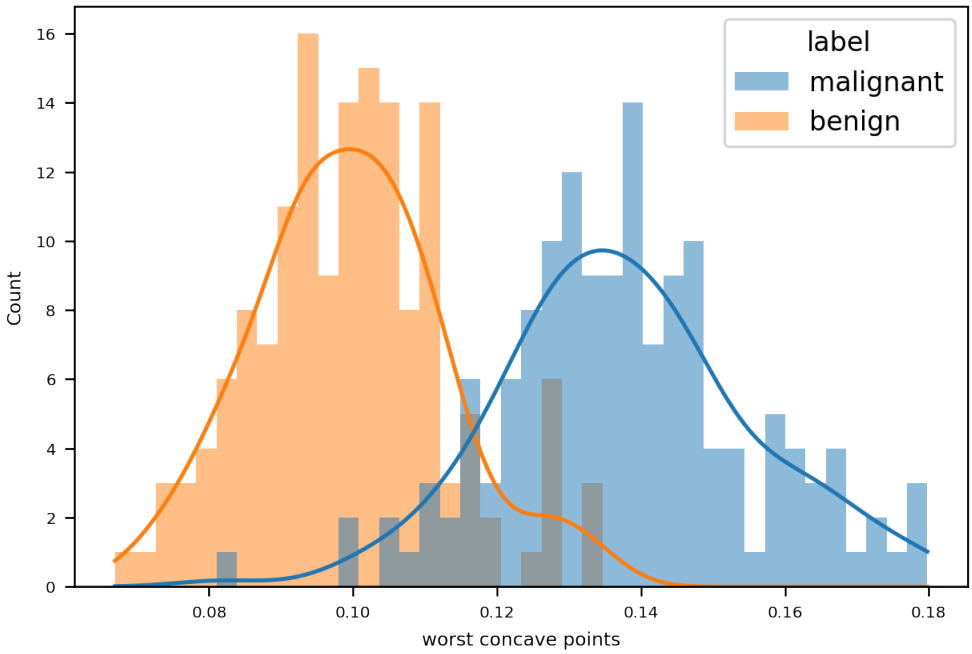

label
malignant
benign
Count
worst perimeter
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
90
100
110
120
130
140
150

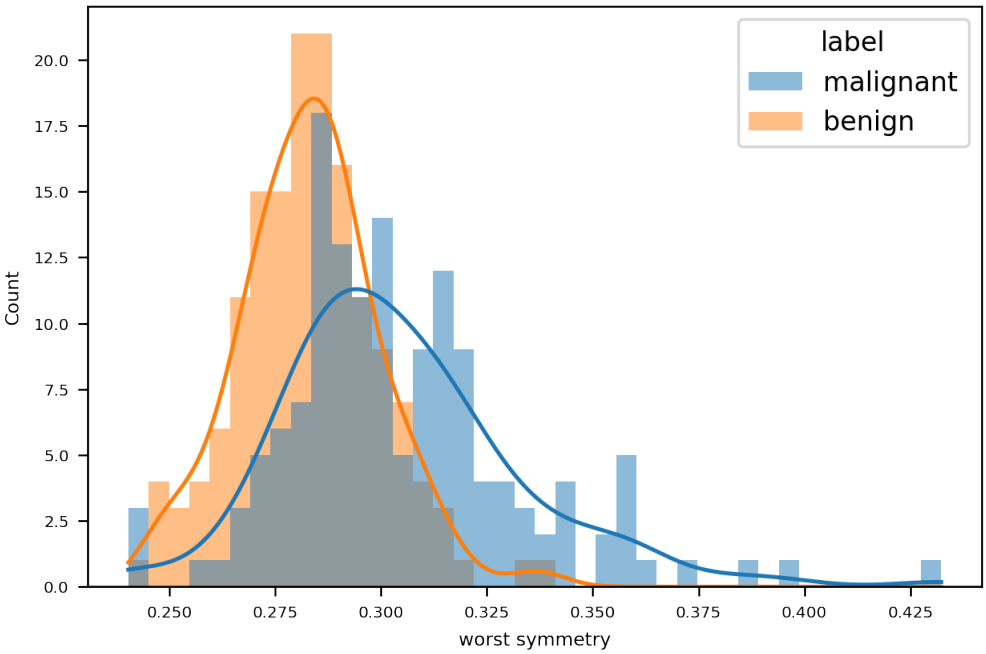

label
malignant
benign
Count
worst area
0
5
10
15
20
25
30
600
800
1000
1200
1400
1600
1800

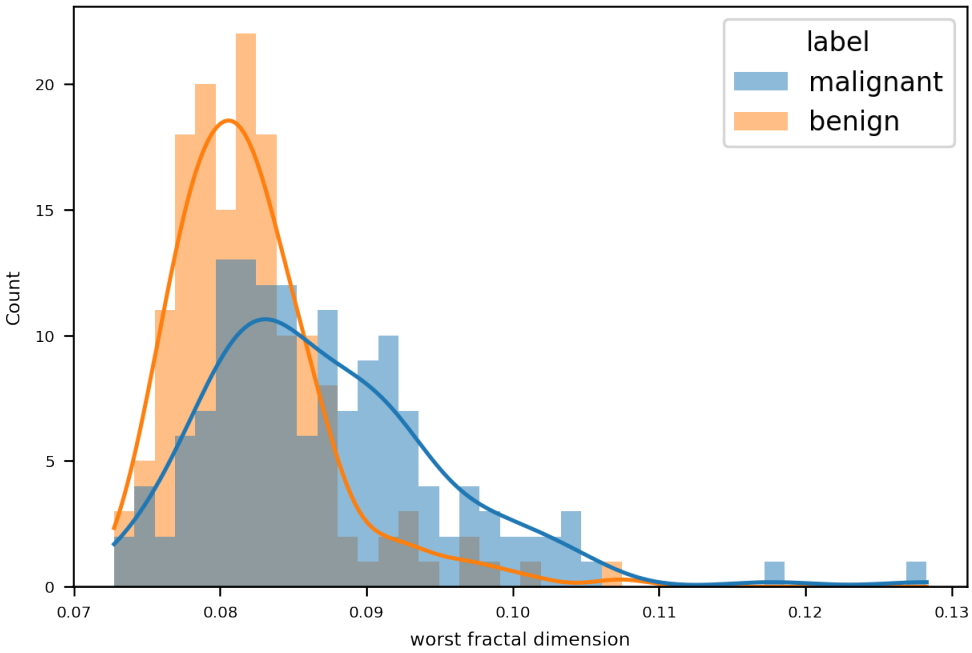

label
malignant
benign
Count
worst smoothness
0
2
4
6
8
10
12
14
16
0.11
0.12
0.13
0.14
0.15
0.16


label
malignant
benign
Count
worst compactness
0
5
10
15
20
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55


label
malignant
benign
Count
worst concavity
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.2
0.3
0.4
0.5
0.6


label
malignant
benign
Count
worst concave points
0
2
4
6
8
10
12
14
16
0.08
0.10
0.12
0.14
0.16
0.18





[6]:
```
# Your code here
```

3.1.3 1.3 Ranking the features [0.5 marks]

Based on the histograms, which do you think are the 3 strongest features for discriminating between the classes?

[ ]:

[7]:
```
# Your answer here
'''worst concave points
worst radius
worst perimeter '''
```

[7]:
```
'worst concave points \nworst radius  \nworst perimeter '
```

3.1.4 1.4 Splitting the dataset [0.5 marks]

Split the dataset into appropriate subsets. You must choose what the subsets are and how big they are. However, we want to make sure the proportion of the two classes is consistent across all datasets, so use the *stratify* option, as used in workshops 5 and 6. Verify the size and label distribution in each dataset.

[8]:
```
#we want to replace malignant with 1 and benign with 0

for i in range(len(df['label'])):
    if df['label'][i] == 'malignant':
        df['label'][i] = 1
    elif df['label'][i] == 'benign':
        df['label'][i] = 0
df['label'] = pd.to_numeric(df['label'])
print(df.describe())
```

```
C:\Users\hao\AppData\Local\Temp\ipykernel_3540\611356547.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['label'][i] = 1
C:\Users\hao\AppData\Local\Temp\ipykernel_3540\611356547.py:7:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['label'][i] = 0

            label  mean radius  mean texture  mean perimeter    mean area  \
```

```
count  300.000000   300.000000    300.000000      300.000000   300.000000
mean     0.486667    14.231808     19.312619       92.727687   664.367372
std      0.500657     1.297393      1.572224        8.949937   129.515717
min      0.000000    11.560025     15.349270       74.690886   477.371592
25%      0.000000    13.356676     18.194791       86.659535   580.383274
50%      0.000000    13.976933     19.220652       90.896982   628.004851
75%      1.000000    15.103078     20.245660       99.093762   737.444716
max      1.000000    19.090091     26.836291      126.168030  1300.788708

       mean smoothness  mean compactness  mean concavity  mean concave points  \
count       300.000000        300.000000      300.000000           300.000000
mean          0.096937          0.106615        0.092591             0.050820
std           0.005067          0.020819        0.030312             0.014350
min           0.084651          0.075184        0.050771             0.028701
25%           0.093305          0.091105        0.069071             0.039507
50%           0.096722          0.102401        0.084829             0.046744
75%           0.099995          0.117334        0.107994             0.060606
max           0.114500          0.192880        0.212704             0.105212

       mean symmetry  …  worst radius  worst texture  worst perimeter  \
count     300.000000  …    300.000000     300.000000       300.000000
mean        0.182546  …     16.460566      25.772128       108.563914
std         0.010754  …      1.798202       2.346310        12.500033
min         0.157059  …     13.279265      20.144214        87.110184
25%         0.175353  …     15.148044      24.058893        99.229249
50%         0.181685  …     16.007171      25.689861       105.540619
75%         0.187789  …     17.656889      27.333610       116.274995
max         0.226448  …     22.676185      34.614459       150.353232

        worst area  worst smoothness  worst compactness  worst concavity  \
count   300.000000        300.000000         300.000000       300.000000
mean    900.644633          0.133424           0.261732         0.282075
std     209.738842          0.008678           0.063535         0.079831
min     633.771881          0.110342           0.167098         0.152272
25%     752.124790          0.127682           0.215767         0.219671
50%     828.667704          0.133064           0.247022         0.267894
75%    1011.628413          0.138650           0.298732         0.325278
max    1796.820974          0.164583           0.543118         0.635074

       worst concave points  worst symmetry  worst fractal dimension
count            300.000000      300.000000               300.000000
mean               0.118146        0.293620                 0.084556
std                0.024552        0.025620                 0.007427
min                0.066927        0.240341                 0.072745
25%                0.098389        0.277676                 0.079636
50%                0.115679        0.288994                 0.082610
75%                0.136687        0.305227                 0.087645
max                0.179794        0.432297                 0.128288
```

```
[8 rows x 31 columns]
```

[9]:
```
#change pd into numpy
df = df.to_numpy()
print(df)
```

```
[[ 1.          15.49465383 15.90254214 …  0.17166193  0.3532114
   0.09773137]
 [ 1.          16.2298708  18.78561273 …  0.13673522  0.28442748
   0.08575834]
 [ 1.          16.34567074 20.11407563 …  0.16149678  0.31303841
   0.08434036]
 …
 [ 1.          13.12305171 18.79305672 …  0.10036103  0.25686255
   0.07966661]
 [ 0.          14.41199052 18.97067446 …  0.10535445  0.28090046
   0.08182846]
 [ 0.          12.70417392 20.89514253 …  0.08505254  0.26596321
   0.07826855]]
```

[10]:
```
# Your code here
# print(df[''])
from sklearn.model_selection import train_test_split
big_train_set, test_set = train_test_split(df, test_size=0.2, random_state=42,␣
 ↪stratify=df[:,0])
train_set, val_set = train_test_split(big_train_set, test_size=0.2,␣
 ↪random_state=42, stratify=big_train_set[:,0])
```

[11]:
```
X_train = train_set[:,1:]
y_train = train_set[:,0]
X_test = test_set[:,1:]
y_test = test_set[:,0]
X_val = val_set[:,1:]
y_val = val_set[:,0]
# print(y_train)
print(f'Shapes are {[X_train.shape,y_train.shape,X_test.shape,y_test.
 ↪shape,X_val.shape,y_val.shape]}')
```

```
Shapes are [(192, 30), (192,), (60, 30), (60,), (48, 30), (48,)]
```

3.2 2. Build, Train and Optimise Classifiers (60% = 18 marks)

3.2.1 2.1 Pipeline [0.5 marks]

Build a pre-processing pipeline that includes imputation (as even though we don't strictly need it here it is a good habit to always include it) and other appropriate pre-processing.

[12]:
```
# Your code here
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

preproc_pl = Pipeline([ ('imputer', SimpleImputer(strategy="median")),
                        ('std_scaler', StandardScaler()) ])
```

3.2.2 2.2 Baseline measurements [1.5 marks]

For our classification task we will consider **three simple baseline cases**: 1) predicting all samples to be negative (class 1) 2) predicting all samples to be positive (class 2) 3) making a random prediction for each sample with equal probability for each class

For each case measure and display the following metrics: - balanced accuracy - recall - precision - auc - f1score - fbeta_score with beta=0.1 - fbeta_score with beta=10

Code is given below for the latter metrics (all metrics are discussed in lecture 4 and many are in workshop 4).

Also **calculate and display the confusion matrix** for each baseline case, using a heatmap and numbers (as in workshop 4).

[13]:
```
from sklearn.metrics import fbeta_score, make_scorer

f10_scorer = make_scorer(fbeta_score, beta=10)
f01_scorer = make_scorer(fbeta_score, beta=0.1)

def f10_score(yt,yp):
    return fbeta_score(yt, yp, beta=10)

def f01_score(yt,yp):
    return fbeta_score(yt, yp, beta=0.1)
```

[14]:
```
# Your code here
#if every pridiction is negative
y_ng = [0.0]* y_val.shape[0]
y_ng = np.array(y_ng)
#if every pridiction is negative
y_pt = [1.0]* y_val.shape[0]
y_pt = np.array(y_pt)
#if random predicted
import random
choices = [0.0,1.0]
y_rd = random.choices(choices, k=y_val.shape[0])
y_rd = np.array(y_rd)
print(y_ng)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

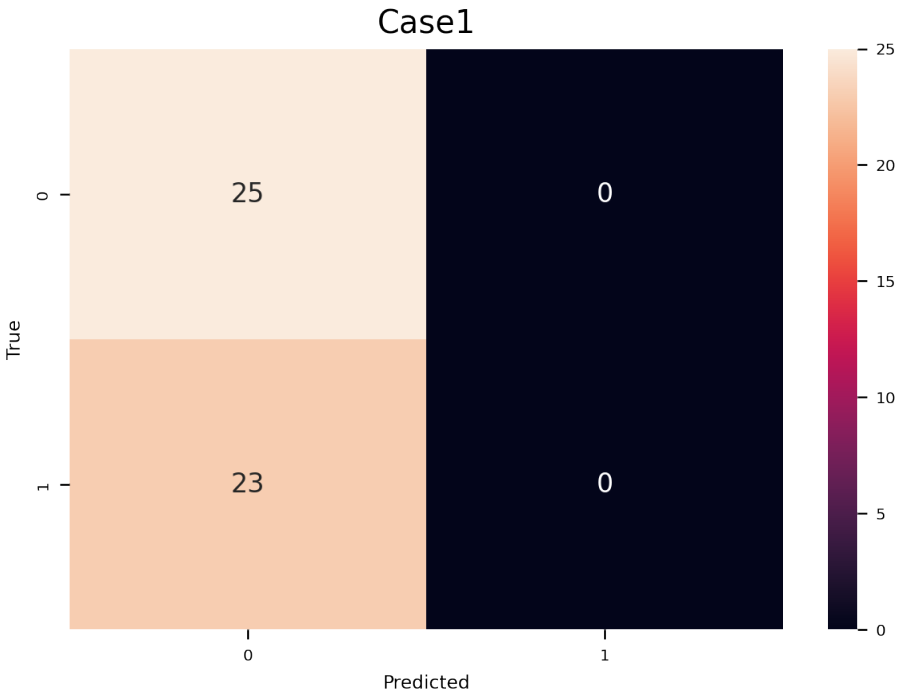
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

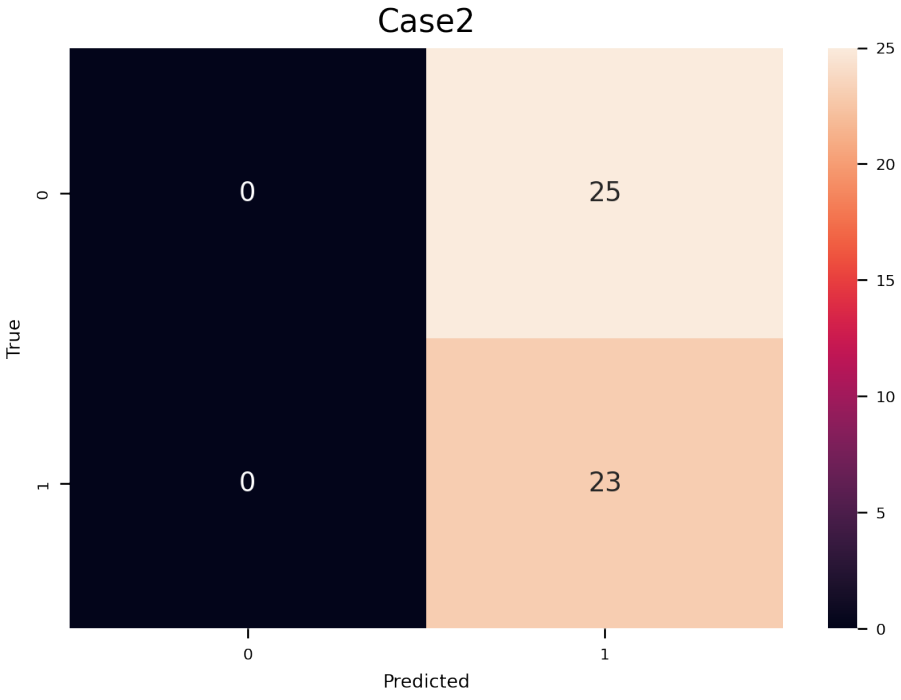
[15]:
```
from sklearn.metrics import balanced_accuracy_score,␣
↪recall_score,precision_score,roc_auc_score,f1_score,auc,roc_curve
#calculate the 3 cases
y_pre_list = [y_ng,y_pt,y_rd]
count = 1
# print(str(y_val))
# print(str(y_pre_list[0]))
for y_pre in y_pre_list:
    print("case "+ str(count))
    print("balanced accuracy: " + str(balanced_accuracy_score(y_val, y_pre)))
    print("recall: " + str(recall_score(y_val, y_pre, average='macro')))
    print("precision: " + str(precision_score(y_val, y_pre, average='macro')))
    fpr, tpr, thresholds = roc_curve(y_val, y_pre, pos_label=1)
    print("auc: " + str(auc(fpr,tpr)))
    print("f1score: " + str(f1_score(y_val, y_pre, average='macro')))
    print("fbeta_score with beta =0.1: "+ str(fbeta_score(y_val, y_pre, beta=0.
↪1)))
    print("fbeta_score with beta =10: "+ str(fbeta_score(y_val, y_pre,␣
↪beta=10)))
    print("_____________________________________________________")
    count+=1
```

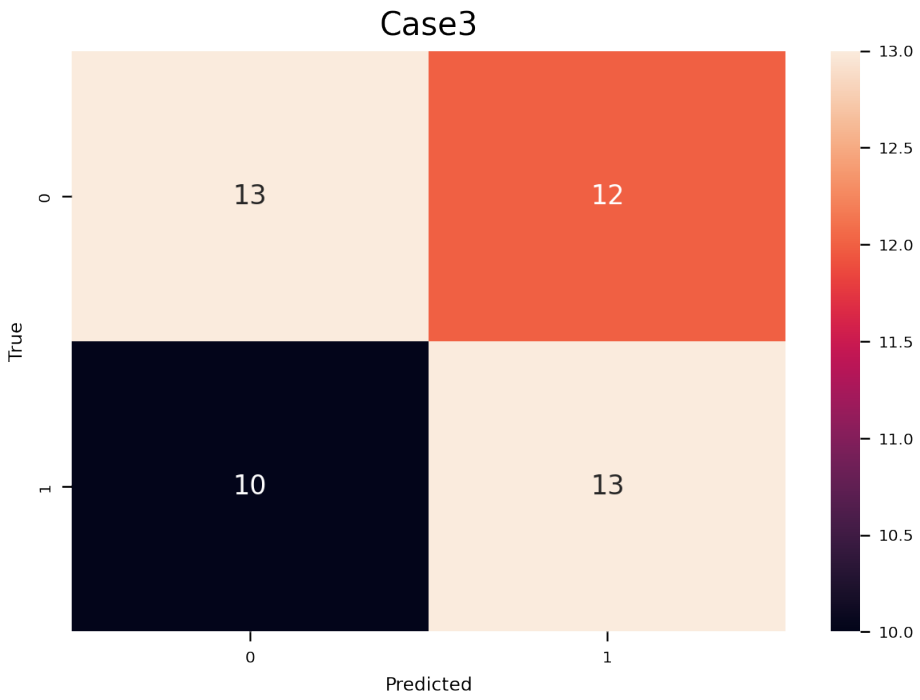
```
case 1
balanced accuracy: 0.5
recall: 0.5
precision: 0.2604166666666667
auc: 0.5
f1score: 0.3424657534246575
fbeta_score with beta =0.1: 0.0
fbeta_score with beta =10: 0.0
-----------------------------------------------------
case 2
balanced accuracy: 0.5
recall: 0.5
precision: 0.23958333333333334
auc: 0.5
f1score: 0.323943661971831
fbeta_score with beta =0.1: 0.48165042504665145
fbeta_score with beta =10: 0.9893526405451447
-----------------------------------------------------
case 3
balanced accuracy: 0.5426086956521738
recall: 0.5426086956521738
precision: 0.5426086956521738
```

```
auc: 0.5426086956521738
f1score: 0.5416666666666667
fbeta_score with beta =0.1: 0.5204122076892589
fbeta_score with beta =10: 0.5647311827956989
--------------------------------------------------
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[16]:
```
#calculate and display the confusion matrix
count=1
from sklearn.metrics import confusion_matrix
import seaborn as sn
for y_pred in y_pre_list:
    cmat = confusion_matrix(y_true=y_val, y_pred=y_pred)
    sn.heatmap(cmat,annot=True)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Case' + str(count))
    plt.show()
    count+=1
```


Case1
25
0
23
0
True
Predicted
0
1
0
1
25
20
15
10
5
0


Case2
0
25
0
23
True
Predicted
0
1
0
1
25
20
15
10
5
0



3.2.3 2.3 Choose a performance metric [0.5 marks]

Based on the above baseline tests and the client's requirements, **choose a performance metric** to use for evaluating/driving your machine learning methods. **Give a reason for your choice.**

[17]:
```
# Your answer here
'''we use f1score as the performance metric
based on performance: because based on the confusion matrix, the third (random)␣
↪case performs closer to the client request that the score of case 3
    is higher than 1 and 2. In other cases they are not obviouse
based on client request: FN and FP are both important in this project, so we␣
↪can use f1 score that:
    F1 Score = 2*(Recall * Precision) / (Recall + Precision)
    so that it will consider both the FN and FP
'''
```

[17]:
```
'we use f1score as the performance metric\nbased on performance: because based
on the confusion matrix, the third (random) case performs closer to the client
request that the score of case 3\n    is higher than 1 and 2. In other cases
they are not obviouse\nbased on client request: FN and FP are both important in
this project, so we can use f1 score that:\n    F1 Score = 2*(Recall *
Precision) / (Recall + Precision)\n    so that it will consider both the FN and
FP\n'
```

3.2.4 2.4 SGD baseline [1 mark]

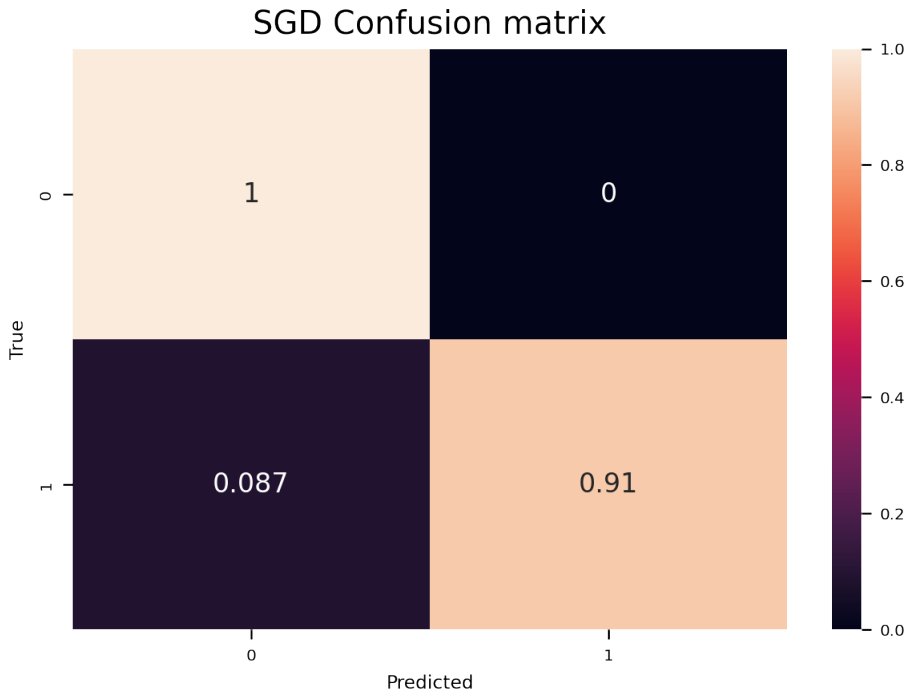
For a stronger baseline, **train and evaluate** the Stochastic Gradient Descent classifier (as seen in workshop 5). For this baseline case use the default settings for all the hyperparameters.

[18]:
```
# Your code here
from sklearn.linear_model import SGDClassifier
sgd_pl = Pipeline([ ('preproc',preproc_pl), ('sgd',SGDClassifier()) ])
sgd_pl.fit(X_train,y_train)
y_val_pred = sgd_pl.predict(X_val)
```

3.2.5 2.5 Confusion matrix [1 mark]

Calculate and display the normalized version of the confusion matrix. From this **calculate the *probability* that a sample from a person with a malignant tumour is given a result that they do not have cancer. Which of the client's two criteria does this relate to, and is this baseline satisfying this criterion or not?**

[19]:
```
# Your code here
cmat = confusion_matrix(y_true=y_val, y_pred=y_val_pred,normalize = 'true')
sn.heatmap(cmat,annot=True)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('SGD Confusion matrix')
plt.show()
```



[20]:
```
# Your answer here
'''0.13 ( based on the buttom left area in the confusion matrix)
1) have at least a 95% probability of detecting malignant cancer when it is␣
 ↪present:
    buttom right's rate: 0.87  (not satisfied)
2) have no more than 1 in 10 healthy cases (those with benign tumours) labelled␣
 ↪as positive (malignant)
    :top right's rate: 0   (satisfied)
'''
```

[20]:
```
"0.13 ( based on the buttom left area in the confusion matrix)\n1) have at least
a 95% probability of detecting malignant cancer when it is present:  \n
buttom right's rate: 0.87  (not satisfied)\n2) have no more than 1 in 10 healthy
cases (those with benign tumours) labelled as positive (malignant) \n    :top
right's rate: 0   (satisfied)\n"
```

3.2.6 2.6 Main classifier [11 marks]

Train and optimise the hyperparameters to give the best performance for **each of the following classifiers**: - KNN (K-Nearest Neighbour) classifier - Decision tree classifier - Support vector machine classifier - SGD classifier

Follow best practice as much as possible here. You must make all the choices and decisions yourself, and strike a balance between computation time and performance.

You can use any of the sci-kit learn functions in sklearn.model_selection.cross* and anything used in workshops 3, 4, 5 and 6. Other hyper-parameter optimisation functions apart from these cannot be used (even if they are good and can be part of best practice in other situations - for this assignment everyone should assume they only have very limited computation resources and limit themselves to these functions).

Display the performance of the different classifiers and the optimised hyperparameters.

Based on these results, list the best 3 classifiers and indicate if you think any perform equivalently.

[21]:
```
# Your code here
#KNN we use GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
knn_pl = Pipeline([ ('imputer', SimpleImputer(strategy="median")),
                            ('std_scaler',StandardScaler()␣
 ↪),('knn',KNeighborsClassifier())])

parameters = {'knn__n_neighbors':[3,5,10,20,30,50], 'knn__weights':
 ↪['uniform','distance']}
gridcv = GridSearchCV(knn_pl, parameters, cv=5, scoring='f1')
gridcv.fit(X_train, y_train)
```

```
print(f'Best score is {gridcv.best_score_} for best params of {gridcv.
 ↪best_params_}')
```

```
Best score is 0.947325746799431 for best params of {'knn__n_neighbors': 20,
'knn__weights': 'distance'}
```

[22]:
```
#Decision tree classifier
from sklearn.tree import DecisionTreeClassifier, plot_tree


dt_pl = Pipeline([ ('preproc',preproc_pl), ('dt',DecisionTreeClassifier()) ])
parameters = {'dt__ccp_alpha':[0.0,0.1,0.05,1,2,3,4,5,0.01],'dt__splitter':
 ↪['best','random'],'dt__criterion':['gini', 'entropy'],}
gridcv = GridSearchCV(dt_pl, parameters, cv=5, scoring = 'f1')
gridcv.fit(X_train, y_train)


print(f'Best score is {gridcv.best_score_} for best params of {gridcv.
 ↪best_params_}')
```

```
Best score is 0.9185870080606922 for best params of {'dt__ccp_alpha': 0.0,
'dt__criterion': 'entropy', 'dt__splitter': 'random'}
```

[23]:
```
#Support vector machine classifier
from sklearn.svm import SVC

svm_pl = Pipeline([('preproc',preproc_pl), ('svc',SVC())])
parameters = {'svc__C':[0.1,0.05,1,2,3,4,5,0.01],'svc__gamma':
 ↪['scale','auto'],'svc__random_state':[1,2,3,4,5,10,15,20,30,40],}
gridcv = GridSearchCV(svm_pl, parameters, cv=5, scoring='f1')
gridcv.fit(X_train, y_train)
print(f'Best score is {gridcv.best_score_} for best params of {gridcv.
 ↪best_params_}')
```

```
Best score is 0.9635419630156472 for best params of {'svc__C': 2, 'svc__gamma':
'scale', 'svc__random_state': 1}
```

[24]:
```
#SGD classifier

sgd_pl = Pipeline([('preproc',preproc_pl), ('sgd',SGDClassifier())])
parameters = {'sgd__alpha':[0.0001,0.001, 0.01, 0.1, 1, 10],
              'sgd__loss':['hinge','log','modified_huber','squared_hinge'],
              'sgd__penalty':['l2','l1','elasticnet'],
             }
gridcv = GridSearchCV(sgd_pl, parameters, cv=5, scoring='f1')
gridcv.fit(X_train, y_train)
```

```python
print(f'Best score is {gridcv.best_score_} for best params of {gridcv.
 ↪best_params_}')
```

```
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
```

```
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(

Best score is 0.9561330561330561 for best params of {'sgd__alpha': 1,
'sgd__loss': 'hinge', 'sgd__penalty': 'l2'}
```

[25]:

```
# Your answer here
'''
    we can see the svc model with parameters: {'svc__C': 2, 'svc__gamma':␣
↪'scale', 'svc__random_state': 1}
    and sgd model with parameters: performs {'sgd__alpha': 1, 'sgd__loss':␣
↪'modified_huber', 'sgd__penalty': 'elasticnet'}
    equaly the best
    the next is the KNeighborsClassifier with parameters: {'knn__n_neighbors':␣
↪20, 'knn__weights': 'distance'}
    '''
```

[25]:
```
"\n    we can see the svc model with parameters: {'svc__C': 2, 'svc__gamma':
'scale', 'svc__random_state': 1} \n    and sgd model with parameters: performs
{'sgd__alpha': 1, 'sgd__loss': 'modified_huber', 'sgd__penalty': 'elasticnet'}\n
equaly the best \n    the next is the KNeighborsClassifier with parameters:
{'knn__n_neighbors': 20, 'knn__weights': 'distance'}\n    "
```

3.2.7 2.7 Model selection [1 mark]

Choose the best classifier (as seen in workshops 3 to 6) and give details of your hyperparameter settings. **Explain the reason for your choice.**

[26]:
```
# Your answer here
'''
    SGDClassifier with parameter {'sgd__alpha': 1, 'sgd__loss':␣
↪'modified_huber', 'sgd__penalty': 'elasticnet'} because it performs
    equaly as the result as the svc, but runs faster
'''
```

[26]:
```
"\n    SGDClassifier with parameter {'sgd__alpha': 1, 'sgd__loss':
'modified_huber', 'sgd__penalty': 'elasticnet'} because it performs\n    equaly
as the result as the svc, but runs faster\n"
```

3.2.8 2.8 Final performance [1.5 marks]

Calculate and display an unbiased performance measure that you can present to the client.

Is your chosen classifier underfitting or overfitting?

Does your chosen classifier meet the client's performance criteria?

[27]:
```
# Your code here
#model
best_sgd_pl = Pipeline([('preproc',preproc_pl), ('sgd',SGDClassifier(alpha = 1,␣
↪loss = 'modified_huber', penalty = 'elasticnet'))])
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score
cv_results = cross_validate(best_sgd_pl, X_train, y_train, cv=10,␣
↪return_train_score=True, scoring='neg_root_mean_squared_error')
print(f"The mean of the validation split's rmse is : {np.
↪mean(cv_results['test_score'])}, \nthe standard deviation  of the validation␣
↪split's rmse is : {np.std(cv_results['test_score'])} ")
print(f"The mean of the training split's rmse is : {np.
↪mean(cv_results['train_score'])}, \nthe standard deviation  of the training␣
↪split's rmse is : {np.std(cv_results['train_score'])} ")
```

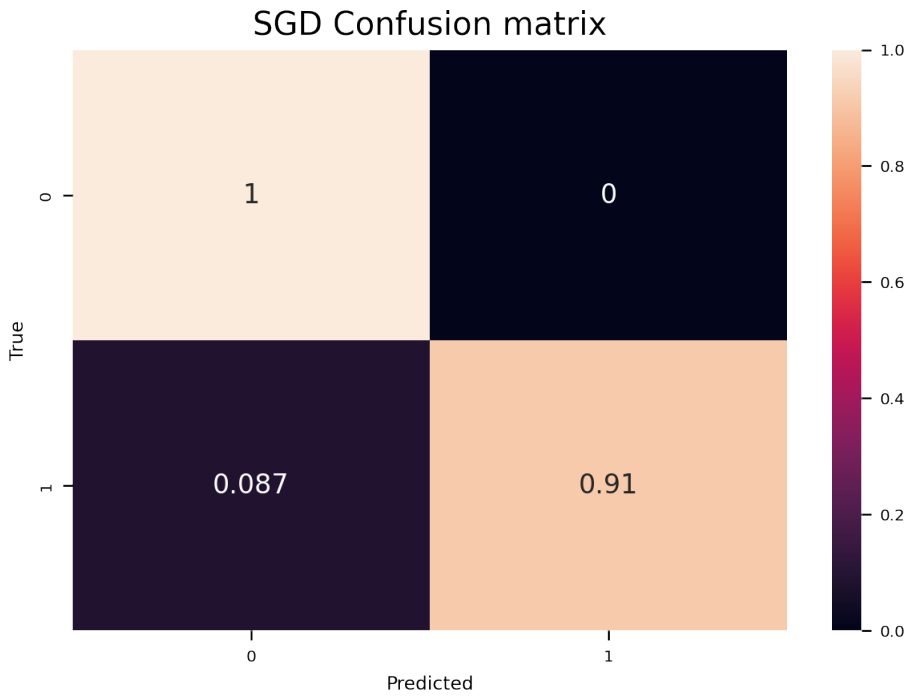
```
The mean of the validation split's rmse is : -0.1457497499249064,
the standard deviation  of the validation split's rmse is : 0.12169487243008481
```

```
The mean of the training split's rmse is : -0.18892285196029057,
the standard deviation  of the training split's rmse is : 0.013413163824032261
```

[28]:
```
# Your answers here
'''Based on the cross validate, the test error and validate error are both␣
 ↪small,
so I believe they are well fitted (not underfitting nor overfitting)'''
```

[28]:
```
'Based on the cross validate, the test error and validate error are both small,
\nso I believe they are well fitted (not underfitting nor overfitting)'
```

[29]:
```
#make confusion matrix
sgd_pl = Pipeline([ ('preproc',preproc_pl), ('sgd',SGDClassifier(alpha = 1,␣
 ↪loss = 'modified_huber', penalty = 'elasticnet')) ])
sgd_pl.fit(X_train,y_train)
y_val_pred = sgd_pl.predict(X_val)


cmat = confusion_matrix(y_true=y_val, y_pred=y_val_pred,normalize = 'true')
sn.heatmap(cmat,annot=True)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('SGD Confusion matrix')
plt.show()
```



[30]:
```
'''
1) have at least a 95% probability of detecting malignant cancer when it is␣
 ↪present:
    buttom right's rate: 0.91  (not satisfied)
2) have no more than 1 in 10 healthy cases (those with benign tumours) labelled␣
 ↪as positive (malignant)
    :top right's rate: 0   (satisfied)
    '''
```

[30]:
```
"\n1) have at least a 95% probability of detecting malignant cancer when it is
present:  \n    buttom right's rate: 0.91  (not satisfied)\n2) have no more than
1 in 10 healthy cases (those with benign tumours) labelled as positive
(malignant) \n    :top right's rate: 0   (satisfied)\n    "
```

3.3 3. Decision Boundaries (15% = 4.5 marks)

3.3.1 3.1 Rank features [1 mark]

Although it is only possible to know the true usefulness of a feature when you've combined it with others in a machine learning method, it is still helpful to have some measure for how discriminative each feature is on its own. One common method for doing this is to calculate a T-score (often used in statistics, and in the LDA machine learning method) for each feature.

The formula for the T-score is (mean(x2) - mean(x1))/(0.5*(stddev(x2) + stddev(x1))), where x1 and x2 are the datasets corresponding to the two classes. Large values for the T-score (either positive or negative) indicate discriminative ability.

Calculate the T-score for each feature and print out the best 4 features according to this score.

[31]:
```
# Your code here
# print(df[:,0]==1)
def T_score(col):
    positive = (df[:,0]==1)
    negative = (df[:,0]==0)
    x1 = col[positive]
    x2 = col[negative]
    TScore = (np.mean(x2) - np.mean(x1))/(0.5*(np.std(x2) + np.std(x1)))
    return TScore
# print(df.shape)
T_dict = {}
for i in range(1,df.shape[1]):
    TScore = T_score(df[:,i])
    T_dict[i] = abs(TScore)
T_dict = sorted(T_dict.items(), key=lambda x: x[1], reverse=True)
print(T_dict)
```

```
[(28, 2.495528853810555), (23, 2.4814395467315733), (21, 2.418995524549576), (8,
```

```
2.2369776759406075), (24, 2.2344434981169043), (3, 2.0893390613317733), (1,
2.0099485745945516), (4, 1.9290508959212673), (7, 1.6142051887880295), (27,
1.5911984303710183), (26, 1.503649841836972), (6, 1.4303842054768625), (14,
1.410309074824269), (13, 1.3030191610338049), (11, 1.280761713716786), (22,
1.189235329203478), (2, 1.0894469142315608), (25, 0.9698666062380196), (29,
0.9549747572295687), (30, 0.7620267894247168), (5, 0.7449476389951291), (18,
0.6701013526730872), (9, 0.6437206939768972), (16, 0.5090403693491431), (17,
0.3372121419150813), (15, 0.18979536323296795), (19, 0.12350146054177355), (12,
0.092680458567879), (20, 0.062487765503678516), (10, 0.01136536590369952)]
```

[32]:
```
'''The best 4 features are
worst concave points : 2.495528853810555
worst perimeter: 2.4814395467315733
worst radium: 2.418995524549576
mean concave points: 2.2369776759406075'''
```

[32]:
```
'The best 4 features are \nworst concave points : 2.495528853810555\nworst
perimeter: 2.4814395467315733\nworst radium: 2.418995524549576\nmean concave
points: 2.2369776759406075'
```

3.3.2 3.2 Visualise decision boundaries [2.5 marks]

Display the decision boundaries for each pair of features from the best 4 chosen above. You can use the functions below to help (taken from workshop 6).

Instead of using the simple mean as the input for `xmean` in `plot_contours`, use the following: 0.5*(mean(x1) + mean(x2)), where x1 and x2 are the data associated with the two classes. This way of calculating a "mean" point takes into account any imbalance between the classes.

[33]:
```
def make_meshgrid(x, y, ns=100):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on (only min and max used)
    y: data to base y-axis meshgrid on (only min and max used)
    ns: number of steps in grid, optional

    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min(), x.max()
    y_min, y_max = y.min(), y.max()
    hx = (x_max - x_min)/ns
    hy = (y_max - y_min)/ns
    xx, yy = np.meshgrid(np.arange(x_min, x_max + hx, hx), np.arange(y_min,␣
↪y_max + hy, hy))
```

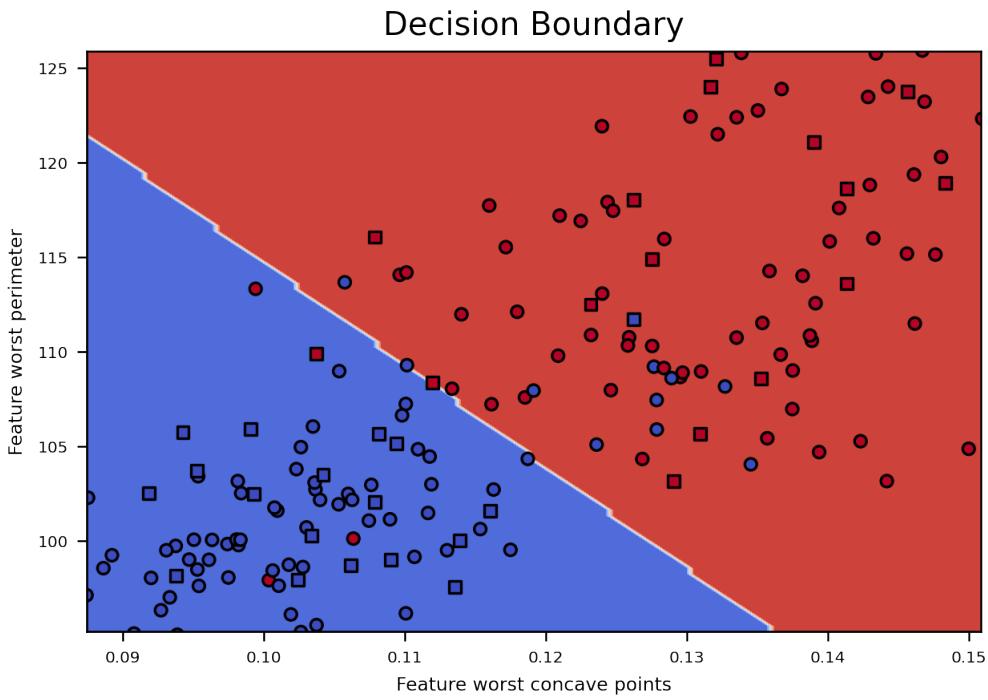
```
    return xx, yy
```

[34]:
```
def plot_contours(clf, xx, yy, xmean, n1, n2, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    xmean : 1d array of mean values (to populate constant features with)
    n1, n2: index numbers of features that change (for xx and yy)
    params: dictionary of params to pass to contourf, optional
    """
    # The following lines makes an MxN matrix to pass to the classifier (# samples x # features)
    # It does this by multiplying Mx1 and 1xN matrices, where the former is filled with 1's
    #  where M is the number of grid points in xx and N is the number of features in xmean
    #  It is done in such a way that the xmean vector is replaced in each row
    fullx = np.ones((xx.ravel().shape[0],1)) * np.reshape(xmean,(1,-1))
    fullx[:,n1] = xx.ravel()
    fullx[:,n2] = yy.ravel()
    print(xmean)
    Z = clf.predict(fullx)
#     print(Z)
    Z = Z.reshape(xx.shape)
#     print(yy.shape)
    out = plt.contourf(xx, yy, Z, **params)
    return out
```

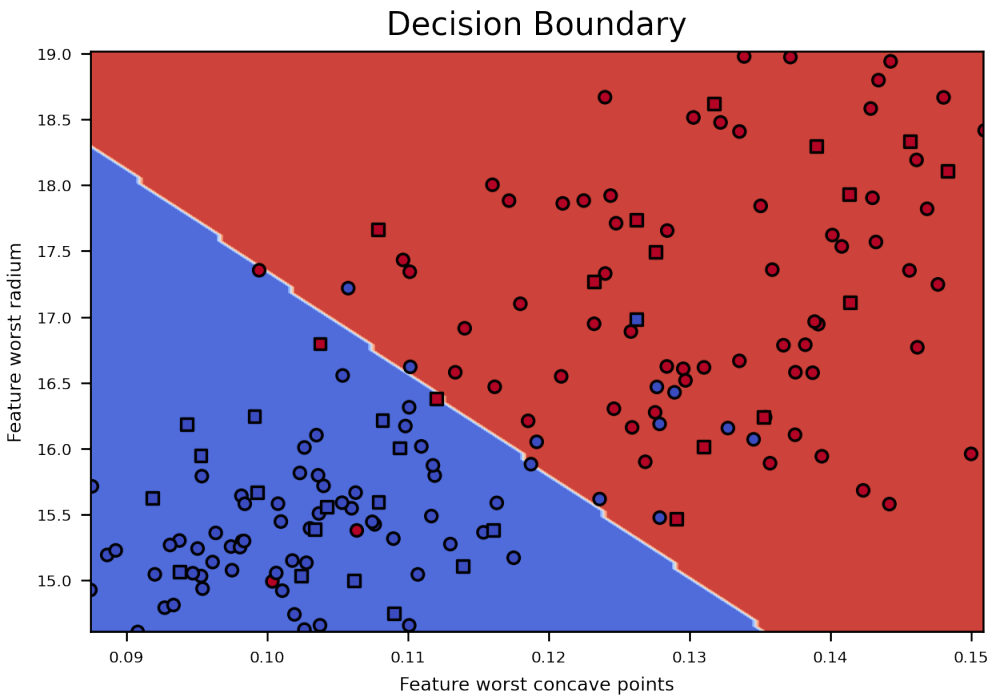
[35]:
```
# Your code here

def xmean(col):
    positive = (df[:,0]==1)
    negative = (df[:,0]==0)
    x1 = col[positive]
    x2 = col[negative]
#     print(x1)
    X_mean = 0.5*(np.mean(x1) + np.mean(x2))
    return X_mean
sgd_pl.fit(X_train,y_train)
feature_titles = ["worst concave points","worst perimeter","worst radium","mean concave points"]
feature_indexes = [28,23,21,8]
X_mean =[]
```

```
for i in range(1,31):
    X_mean.append( xmean(df[:,i]))
X_mean = np.array(X_mean)

# print(X_mean.shape)
for i in range(0,4):
    for j in range(i+1,4):
        x_index = feature_indexes[i] -1
        y_index = feature_indexes[j] -1
        x_title = feature_titles[i]
        y_title = feature_titles[j]
        x = X_train[:,x_index]
        y = X_train[:,y_index]
        x10, x90 = np.percentile(x,[10,90])
        y10, y90 = np.percentile(y,[10,90])
        xx, yy = make_meshgrid(np.array([x10, x90]), np.array([y10, y90]))
#         print(X_train[:,x_index])
        plot_contours(sgd_pl, xx, yy, X_mean, x_index, y_index, cmap=plt.cm.
 ↪coolwarm)
        plt.scatter(X_train[:,x_index], X_train[:,y_index], c=y_train, cmap=plt.
 ↪cm.coolwarm, s=20, edgecolors="k")
        plt.scatter(X_val[:,x_index], X_val[:,y_index], c=y_val, cmap=plt.cm.
 ↪coolwarm, s=20, edgecolors="k", marker='s')

        plt.xlim(xx.min(), xx.max())
        plt.ylim(yy.min(), yy.max())
        plt.xlabel(f"Feature {x_title}")
        plt.ylabel(f"Feature {y_title}")
        plt.title("Decision Boundary")

        plt.show()
```

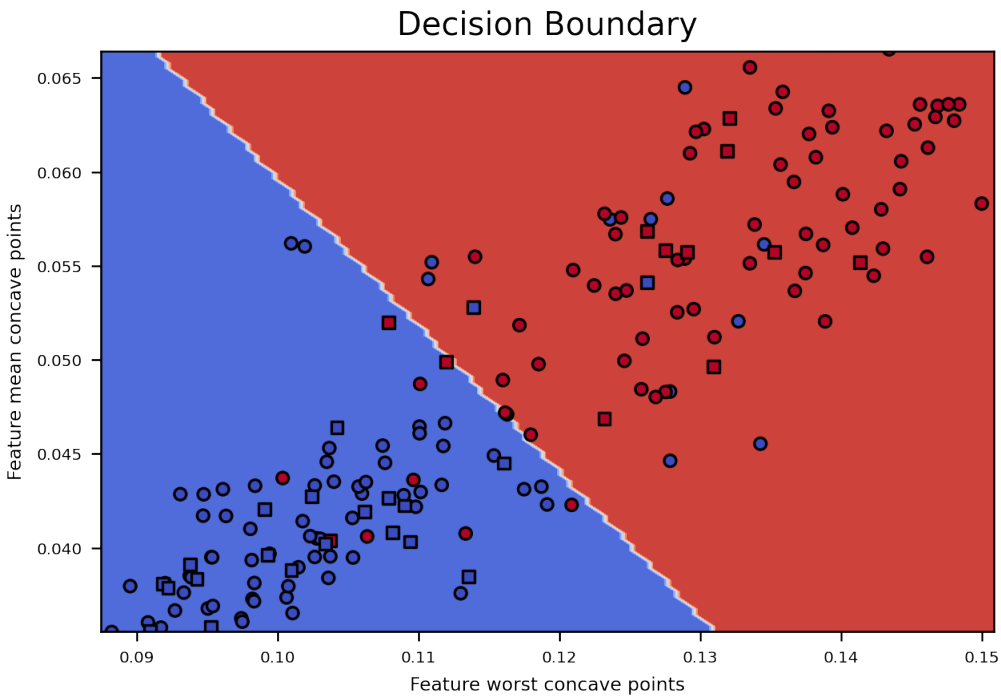
```
[1.42559363e+01 1.93326393e+01 9.28971467e+01 6.66660133e+02
 9.69838127e-02 1.06931855e-01 9.30947386e-02 5.10984432e-02
 1.82634111e-01 6.28411497e-02 4.17809311e-01 1.21667852e+00
 2.94933618e+00 4.18915346e+01 7.05696326e-03 2.61593156e-02
 3.28580597e-02 1.20667572e-02 2.07442821e-02 3.86096286e-03
 1.64967564e+01 2.58040461e+01 1.08818258e+02 9.04588057e+02
 1.33524568e-01 2.62727045e-01 2.83398986e-01 1.18655516e-01
 2.93907426e-01 8.46249012e-02]
```



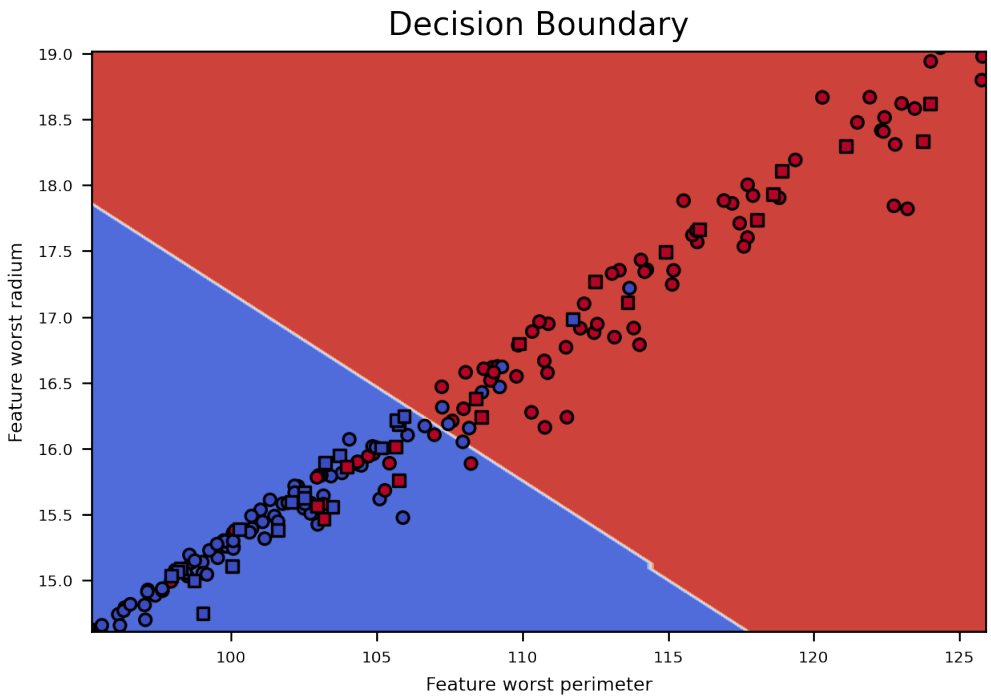

```
[1.42559363e+01 1.93326393e+01 9.28971467e+01 6.66660133e+02
 9.69838127e-02 1.06931855e-01 9.30947386e-02 5.10984432e-02
 1.82634111e-01 6.28411497e-02 4.17809311e-01 1.21667852e+00
 2.94933618e+00 4.18915346e+01 7.05696326e-03 2.61593156e-02
 3.28580597e-02 1.20667572e-02 2.07442821e-02 3.86096286e-03
 1.64967564e+01 2.58040461e+01 1.08818258e+02 9.04588057e+02
 1.33524568e-01 2.62727045e-01 2.83398986e-01 1.18655516e-01
 2.93907426e-01 8.46249012e-02]
```



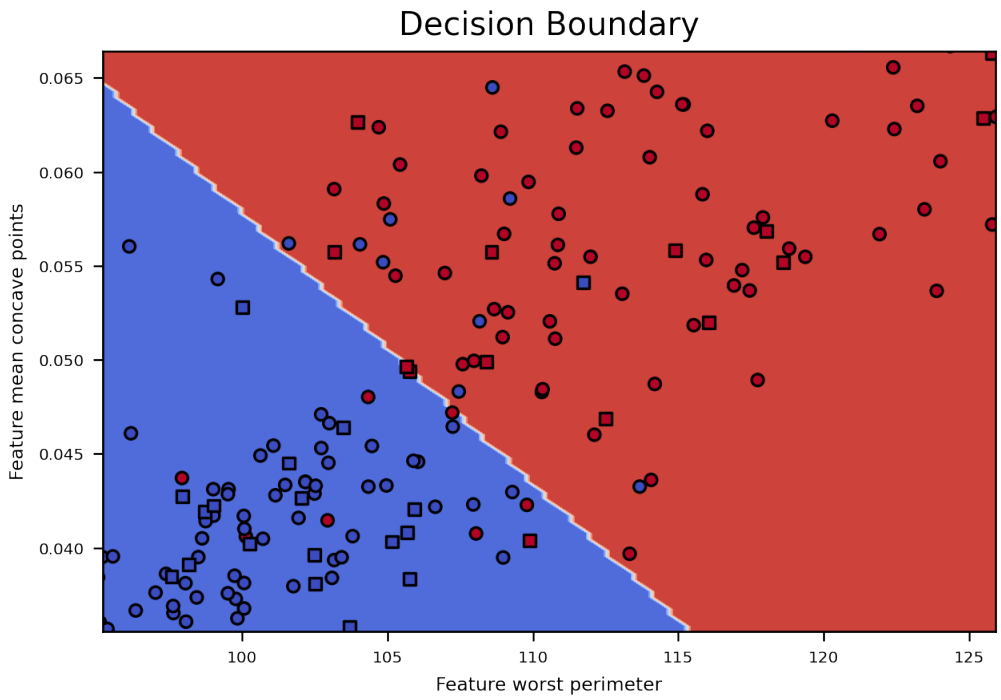

```
[1.42559363e+01 1.93326393e+01 9.28971467e+01 6.66660133e+02
 9.69838127e-02 1.06931855e-01 9.30947386e-02 5.10984432e-02
 1.82634111e-01 6.28411497e-02 4.17809311e-01 1.21667852e+00
 2.94933618e+00 4.18915346e+01 7.05696326e-03 2.61593156e-02
 3.28580597e-02 1.20667572e-02 2.07442821e-02 3.86096286e-03
 1.64967564e+01 2.58040461e+01 1.08818258e+02 9.04588057e+02
 1.33524568e-01 2.62727045e-01 2.83398986e-01 1.18655516e-01
 2.93907426e-01 8.46249012e-02]
```



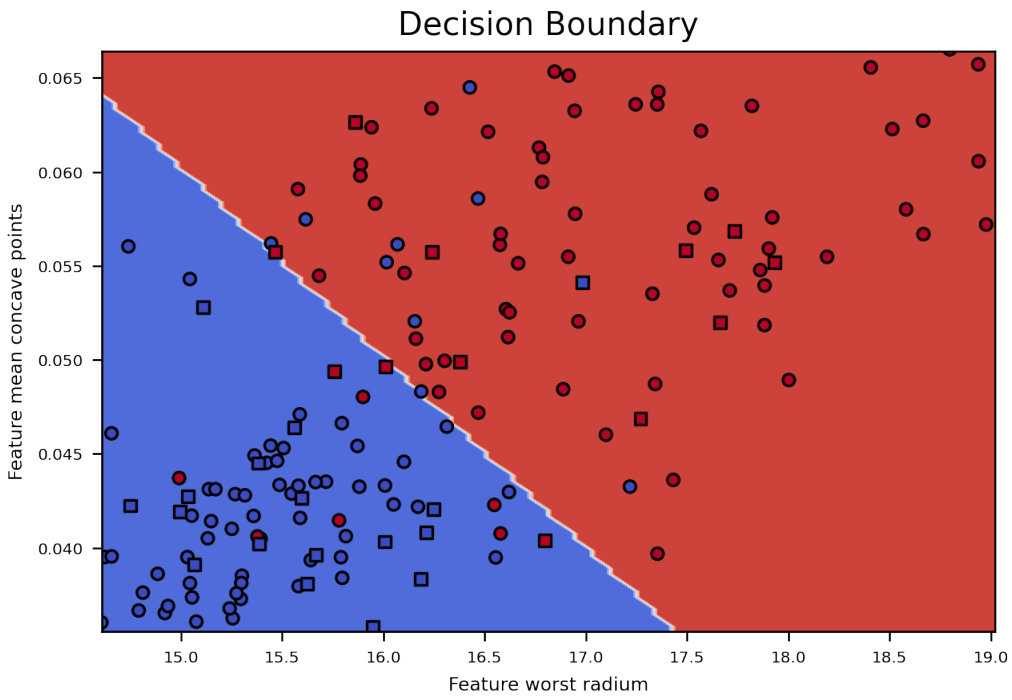

```
[1.42559363e+01 1.93326393e+01 9.28971467e+01 6.66660133e+02
 9.69838127e-02 1.06931855e-01 9.30947386e-02 5.10984432e-02
 1.82634111e-01 6.28411497e-02 4.17809311e-01 1.21667852e+00
 2.94933618e+00 4.18915346e+01 7.05696326e-03 2.61593156e-02
 3.28580597e-02 1.20667572e-02 2.07442821e-02 3.86096286e-03
 1.64967564e+01 2.58040461e+01 1.08818258e+02 9.04588057e+02
 1.33524568e-01 2.62727045e-01 2.83398986e-01 1.18655516e-01
 2.93907426e-01 8.46249012e-02]
```




```
[1.42559363e+01 1.93326393e+01 9.28971467e+01 6.66660133e+02
 9.69838127e-02 1.06931855e-01 9.30947386e-02 5.10984432e-02
 1.82634111e-01 6.28411497e-02 4.17809311e-01 1.21667852e+00
 2.94933618e+00 4.18915346e+01 7.05696326e-03 2.61593156e-02
 3.28580597e-02 1.20667572e-02 2.07442821e-02 3.86096286e-03
 1.64967564e+01 2.58040461e+01 1.08818258e+02 9.04588057e+02
 1.33524568e-01 2.62727045e-01 2.83398986e-01 1.18655516e-01
 2.93907426e-01 8.46249012e-02]
```




```
[1.42559363e+01 1.93326393e+01 9.28971467e+01 6.66660133e+02
 9.69838127e-02 1.06931855e-01 9.30947386e-02 5.10984432e-02
 1.82634111e-01 6.28411497e-02 4.17809311e-01 1.21667852e+00
 2.94933618e+00 4.18915346e+01 7.05696326e-03 2.61593156e-02
 3.28580597e-02 1.20667572e-02 2.07442821e-02 3.86096286e-03
 1.64967564e+01 2.58040461e+01 1.08818258e+02 9.04588057e+02
 1.33524568e-01 2.62727045e-01 2.83398986e-01 1.18655516e-01
 2.93907426e-01 8.46249012e-02]
```



3.3.3 3.3 Interpretation [1 mark]

From the decision boundaries displayed above, **would you expect the method to extrapolate well or not**? Give reasons for your answer.

[36]:
```
# Your answer here
'''Yes, it fits well with the scatters,
but it is may not be able to satisfy the client, still need improvements'''
```

[36]:
```
'Yes, it fits well with the scatters,\nbut it is may not be able to satisfy the
client, still need improvements'
```

3.4 4. Second Round (15% = 4.5 marks)

After presenting your initial results to the client they come back to you and say that they have done some financial analysis and it would save them a lot of time and money if they did not have to analyse every cell, which is needed to get the "worst" features. Instead, they can quickly get accurate estimates for the "mean" and "standard error" features from a much smaller, randomly selected set of cells.

They ask you to **give them a performance estimate for the same problem, but without using any of the "worst" features.**

3.4.1 4.1 New estimate [3.5 marks]

Calculate an unbiased performance estimate for this new problem, as requested by the client.

[37]:
```
# Your code here
'''The worst features are the col 21-30  in the data frame'''
df2 = df[:,0:21]
big_train_set2, test_set2 = train_test_split(df2, test_size=0.2,␣
↪random_state=42, stratify=df2[:,0])
train_set2, val_set2 = train_test_split(big_train_set2, test_size=0.2,␣
↪random_state=42, stratify=big_train_set2[:,0])
X_train2 = train_set2[:,1:]
y_train2 = train_set2[:,0]
X_val2 = val_set2[:,1:]
y_val2 = val_set2[:,0]
print(X_train2.shape)

sgd_pl2 = Pipeline([('preproc',preproc_pl), ('sgd',SGDClassifier())])
parameters = {'sgd__alpha':[0.0001,0.001, 0.01, 0.1, 1, 10],
              'sgd__loss':['hinge','log','modified_huber','squared_hinge'],
              'sgd__penalty':['l2','l1','elasticnet'],
             }
gridcv = GridSearchCV(sgd_pl, parameters, cv=5, scoring='f1')
gridcv.fit(X_train2, y_train2)
print(f'Best score is {-gridcv.best_score_} for best params of {gridcv.
↪best_params_}')
```

```
(192, 20)

D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
```

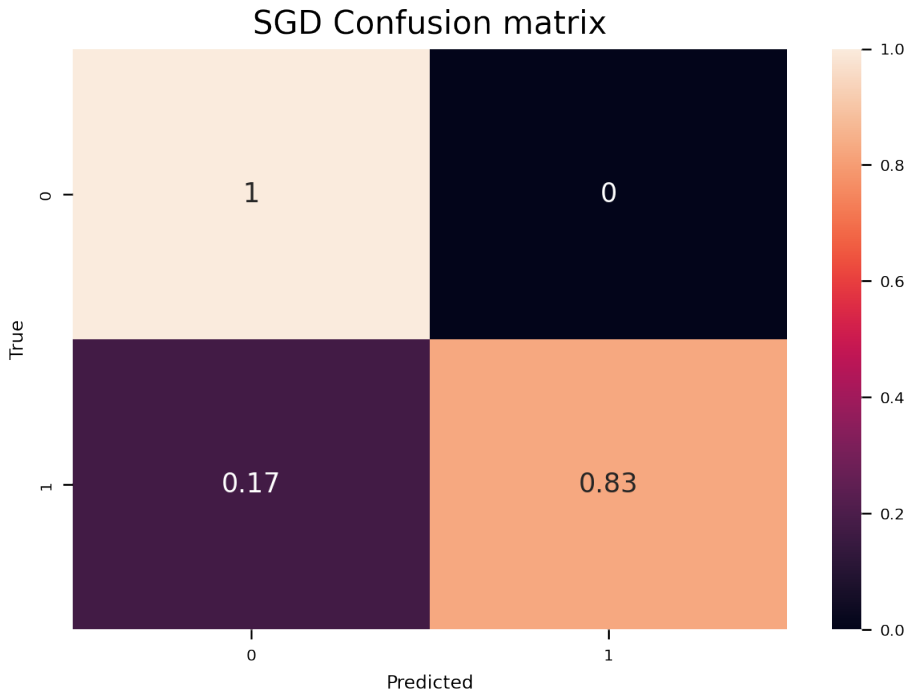
```
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
```

```
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
D:\anaconda\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:696:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit.
  warnings.warn(
```

```
Best score is -0.9347130647130648 for best params of {'sgd__alpha': 0.01,
'sgd__loss': 'log', 'sgd__penalty': 'l1'}
```

[38]:
```
sgd_pl_best2 = Pipeline([ ('preproc',preproc_pl), ('sgd',SGDClassifier(alpha =␣
↪1, loss = 'modified_huber', penalty = 'elasticnet')) ])
sgd_pl_best2.fit(X_train2,y_train2)
y_val_pred2 = sgd_pl_best2.predict(X_val2)



cmat = confusion_matrix(y_true=y_val2, y_pred=y_val_pred2,normalize = 'true')
sn.heatmap(cmat,annot=True)
```

```
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('SGD Confusion matrix')
plt.show()
```



3.4.2 4.2 Performance difference [1 mark]

Do you think the new classifier, that does not use the "worst" features, is: - **as good as the previous classifier** (that uses all the features) - **better than the previous classifier** - **worse than the previous classifier**

Give reasons for your answer.

[39]:
```
# Your answer here
'''worse than the previous classifier:
1) have at least a 95% probability of detecting malignant cancer when it is␣
 ↪present;
    become further from this requirement
2) have no more than 1 in 10 healthy cases (those with benign tumours) labelled␣
 ↪as positive (malignant).
    no change
'''
```

[39]:
```
'worse than the previous classifier:\n1) have at least a 95% probability of
detecting malignant cancer when it is present; \n    become further from this
requirement\n2) have no more than 1 in 10 healthy cases (those with benign
tumours) labelled as positive (malignant).\n    no change\n'
```

[ ]: