Feature Selection, Feature Extraction, Cross-Validation, Pipeline and Grid Search (Predicting Breast Cancer Using KNN)

In this project some of the basic components of machine learning methods are presented and examined. These components include feature selection based on correlation, feature extraction using principal component analysis (PCA) and hyperparameter tuning through cross-validation using packages available in `scikit-learn`. KNN is used in a k-fold cross-validation process to predict new cases of breast cancer. The breast cancer Wisconsin dataset is used for this purpose which is a classic and binary dataset available as one of the `scikit-learn` datasets. This project has five parts:

- **Part 1: Exploratory Data Analysis**: A dataframe is created. Data is split into training and test sets and standardized in a way that there is no leakage from the test set. Then, only the training set is used for visualization.
- **Part 2: Using All Features**: KNN with all the dataset features is used for predicting breast cancer. Each of part 2, 3 and 4 of this project has three sections. In the first section, for different number of neighbors, KNN is applied to the trainiing set without cross-validation, and the test score is reported for each number of neighbors. In the next section hyperparameter tuning is done to find the best number of neighbors in KNN using cross-validation for the training data. A loop over number of neighbors and `cross_val_score` is used in this section. The last step of part 2, 3 and 4 includes using `GridSearchCV` for cross-validation and hyperparameter tuning.
- **Part 3: Feature Selection**: Based on the correlation between features and the target, and the correlation between features themselves, a function is designed to drop some of the features. This function accepts training dataset and order the features based on correlation with the target, then from each two highly correlated features the one which has a weaker correlation with the target is dropped. Then similar to part 2, KNN is used for predicting breast cancer without and with cross-validation based on the new set of features.
- **Part 4: Feature Extraction**: Principal component analysis (PCA) is applied to the training data to extract the most important components (eigenvectors) using singular value decomposition (SVD) of training data or eigendecomposition of the covariance matrix. Then similar to part 2 and 3, KNN is used for predicting breast cancer without and with cross-validation based on the new set of extracted features (which are not the same as the

original features).

- **Part 5: Standardization, Feature Extraction, Cross-Validation and Parameter Tuning, All Together Using Pipeline and GridSearchCV (NO DATA LEAKAGE)**: The proper manner of performing feature extraction is getting it done during cross-validation. In fact there should be no leakage from validation set (the test set inside the training data used in cross-validation) and from the test data. This is done by using a `Pipeline` for cross-validation. Here, a `Pipeline` includes standardization, PCA and KNN. Then combinations of different number of PCA components and KNN neighbors are used for hyperparameter tuning by cross-validation using `GridSearchCV`. This way the optimum parameters are found while making sure there is no data leakage.

```
!pip install -U scikit-learn  # Install the latest version of scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-package
Collecting scikit-learn
  Downloading scikit_learn-1.4.1.post1-cp310-cp310-manylinux_2_17_x86_64.manylinux201
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.1/12.1 MB 42.4 MB/s eta 0:00:00
Requirement already satisfied: numpy<2.0,>=1.19.5 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.2.2
    Uninstalling scikit-learn-1.2.2:
      Successfully uninstalled scikit-learn-1.2.2
Successfully installed scikit-learn-1.4.1.post1
```

Before doing exloratory data analysis let's import all the necessary libraries and packages which will be used in this project.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm  # Built-in colormaps and colormap handling utilities
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split  # For splitting the data
from sklearn.preprocessing import StandardScaler  # For standaridizing the data
from sklearn.neighbors import KNeighborsClassifier  # Import k-nearest neighbors classifi
from sklearn.metrics import accuracy_score  # Import accuracy_score for checking accuracy
from sklearn.model_selection import cross_val_score  # For k-fold cross-validation
from sklearn.pipeline import make_pipeline  # For making a pipeline
from sklearn.decomposition import PCA  # For feature extraction using PCA
from sklearn.model_selection import GridSearchCV  # For performing cross-validation using
```

Part 1: Exploratory Data Analysis

First, the breast cancer Wisconsin dataset is loaded. As is seen the output is a Bunch object which is a special type of dictionary supporing attribute-style access. In version 0.23 of `scikit-learn` it has become possible to export the data part and the target part of this dataset as pandas objects which makes creating the final dataframe easier. We will show three methods for creating the dataframe but before that we extract some information from our Bunch object. Later the data is split into training set and testing set, and standardized in a way making sure there is no leakage from the test set. In the last step of exploratory data analysis, the training data is used to visualize some of the features and it is demonstrated that there is strong correlation between some features which makes it necessary to perform feature selection or feature extraction later on.

```
dataset = load_breast_cancer()
```

```
type(dataset)
```

```
sklearn.utils._bunch.Bunch
def __init__(**kwargs)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/_bunch.py
Container object exposing keys as attributes.

Bunch objects are sometimes used as an output for functions and methods.
They extend dictionaries by enabling values to be accessed by key,
`bunch["value_key"]`, or by an attribute, `bunch.value_key`.
```

```
dataset.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module'])
```

The full description of the dataset is given here. It is seen that the number of instances (participants) is `569` where `357` cases are `benign` and `212` cases are `malignant`. There are `30` features (attributes) in the dataset.

```
## The full description of the dataset ##

print(dataset['DESCR'])  # print(dataset.DESCR)
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------
```

```
    **Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 0 is Mean Radius, field
        10 is Radius SE, field 20 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
```

Information regarding the features, including their numbers and names can be obtained using the following lines.

```
# print(type(dataset['feature_names']))
print(dataset['feature_names'].shape)  # print(dataset.feature_names.shape)
print('Features: {}'.format(dataset['feature_names']))  # print('Feature names: {}'.forma
```

```
(30,)
Features: ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

Target names and the corresponing numbers are found as follows.

```
# print(type(dataset.target_names))
print('Target names: {}'.format(dataset['target_names']))  # print('Target names: {}'.for
```

```
Target names: ['malignant' 'benign']
```

```
# print(type(dataset['target']))  # print(type(dataset.target))
print(dataset['target'].shape)  # print(dataset.target.shape)
print('Targets: {}'.format(set(dataset['target'])))  # print('Targets: {}'.format(set(dat
```

```
(569,)
Targets: {0, 1}
```

As sum of the column target is `357`, `1` represnets benign cases for this dataset.

```
print('Sum of column target = {}'.format(sum(dataset['target'])))  # print('Sum of column
print('Therefore 1 represents benign.')
```

```
Sum of column target = 357
Therefore 1 represents benign.
```

As expected the data part of this dataset has 30 columns (attributes, features) and 569 rows (instances, samples).

```
print(type(dataset['data']))  # print(type(dataset.data))
print(dataset['data'].shape)  # print(dataset.data.shape)
print(dataset['data'].size)  # print(dataset.data.size)
```

```
<class 'numpy.ndarray'>
(569, 30)
17070
```

Part 1.1: Creating the DataFrame

Here three methods are presented for creating the dataframe. In the first method a more recent feature of `scikit-learn` is used which lets us import the data and target directly as pandas objects. In the second and third method the dataframe is created using the information available in the previously imported Bunch object.

Method 1

Question 1: Create the breast cancer dataframe using the most recent features of scikit-learn which lets you import and create the dataframe in one line. For more information visit https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html.

```
df = pd. concat (load_breast_cancer(return_X_y=True, as_frame=True), axis=1)
```

Method 2

Question 2: Create the breast cancer dataframe using a one-line command from the Bunch object `dataset` created before. Use a one-line command.

```
df = pd.DataFrame(np.c_[dataset['data'], dataset[ 'target']], columns=np. append (dataset
```

Method 3

Question 3: Use a third method for creating the dataframe from the Bunch object `dataset`, not necessarily in one line.

```
df = pd. DataFrame(dataset['data'], columns=dataset ['feature_names'])
df[ 'target'] = dataset. target
```

By taking a look at the head of the dataframe and its statistical information it is clear that some features have totally different scales and the data is not standardized or normalized.

Question 4: Display the first five rows of the dataframe.

```
df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	sy
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	

5 rows × 31 columns

Question 5: Display some basic statistical information of the dataframe including mean and standard deviation of each column.

```
df.describe()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	conca
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.00
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.08
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.07
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.00
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.02
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.06
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.13
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.42

8 rows × 31 columns

Here we check data types for the features and the target. All the data types are numeric which makes working with this dataframe easier.

Question 6: Use a command to display the data type for all the features and the target.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

Once more as a practice some information of the dataset is generated using the dataframe itself not the Bunch object we previously explored.

Question 7: Display features of the dataframe.

```
df.columns
```

```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension',
       'target'],
      dtype='object')
```

Question 8: Display the target values and their frequencies.

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

```
1    357
0    212
Name: target, dtype: int64
```

Question 9: Extract and display the number of features, samples, malignant cases, and benign cases.

```
N_total = len(df)
N_malignant = len(df[df['target']==0])
N_benign = len(df[df['target']==1])

print('Number of features: {}'.format(df.shape[1]-1))
print('Target classes: {}'.format(set(df['target'])))
print('Number of participants: {}'.format(N_total))
print('Number of participants tested malignant: {}'.format(N_malignant))
print('Number of participants tested benign: {}'.format(N_benign))
```

```
Number of features: 30
Target classes: {0, 1}
Number of participants: 569
Number of participants tested malignant: 212
Number of participants tested benign: 357
```

Part 1.2: Splitting and standardizing the data

A function is defined to split the dataset into training and testing sets, standardize the dataset using training set making sure there is no leakage from the test set. The outputs `X_train`, `X_test` are in the form of pandas dataframe and `y_train` and `y_test` are in the form of pandas series.

Question 10: Define a function which splits the dataset into training and testing sets, and standardizes the data. Check out the functions provided by `scikit-learn` to complete this task:

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

```
## Function to split and standardize the dataset ##

def split_standard(df, target, test_size, random_state):
  '''
  target: name of the column target
  '''
  ## Split the data ##
  X_train, X_test, Y_train, Y_test = train_test_split(df.drop(target, axis=1), df[target],

  ## Applying StandardScaler to the DataFrame ##
  SS = StandardScaler()
  X_train = SS.fit_transform(X_train)
  X_test = SS.transform(X_test)

  ## Converting X_train and X_test to DataFrame which might be useful later on ##
  X_train = pd.DataFrame(X_train, columns=dataset.feature_names).set_index(y_train.index)
  X_test = pd.DataFrame(X_test, columns=dataset.feature_names).set_index(y_test.index)  #

  return X_train, X_test, y_train, y_test
```

```
## Split and standardize data ##

X_train, X_test, y_train, y_test = split_standard(df, 'target', test_size=0.2, random_sta
```

Part 1.3: Visualizing the standardized training data

Here some features and the correlations between them are visualized. Training data (after standardization) is used for visualization. It is seen that there is a strong correlation between some of the feaures. It is also clear that the distributions of highly correlated features are similar.

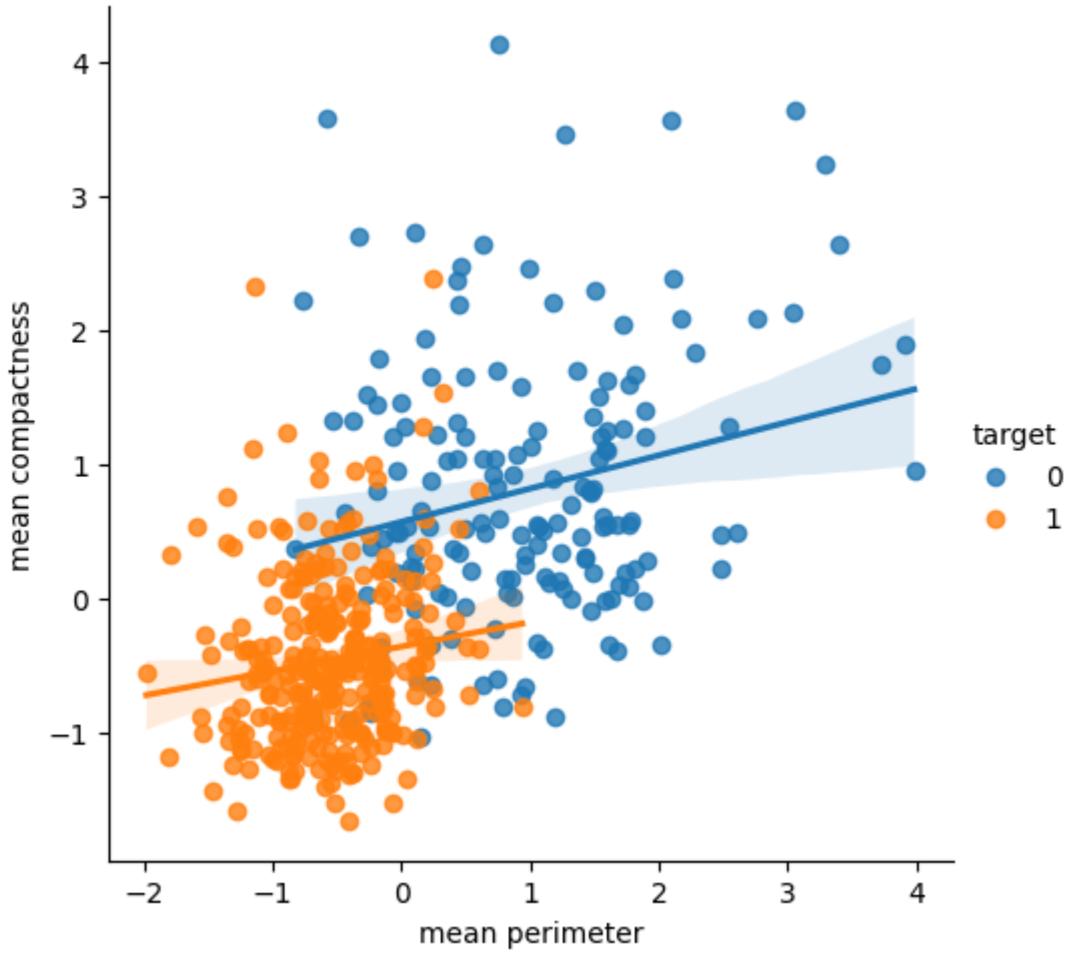
```
## Creating a dataframe including X_train and y_train for visualization ##

df_train = pd.concat([X_train, y_train], axis=1)  # Here the index for X_train and y_trai
```

Question 11: Seaborn is a powerful Python data visualization library based on matplotlib. Use this library to visualize `mean compactness` in terms of `mean perimeter` for different target classes. Make sure that the regression lines are plotted as well. For more information check https://seaborn.pydata.org/generated/seaborn.lmplot.html https://seaborn.pydata.org/generated/seaborn.lmplot.html.

```
## Visualizing two features of the standardized data ##

ax = sns.lmplot(x='mean perimeter', y='mean compactness', hue='target', data=df_train)
plt.show()
```

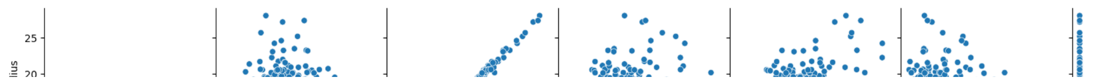


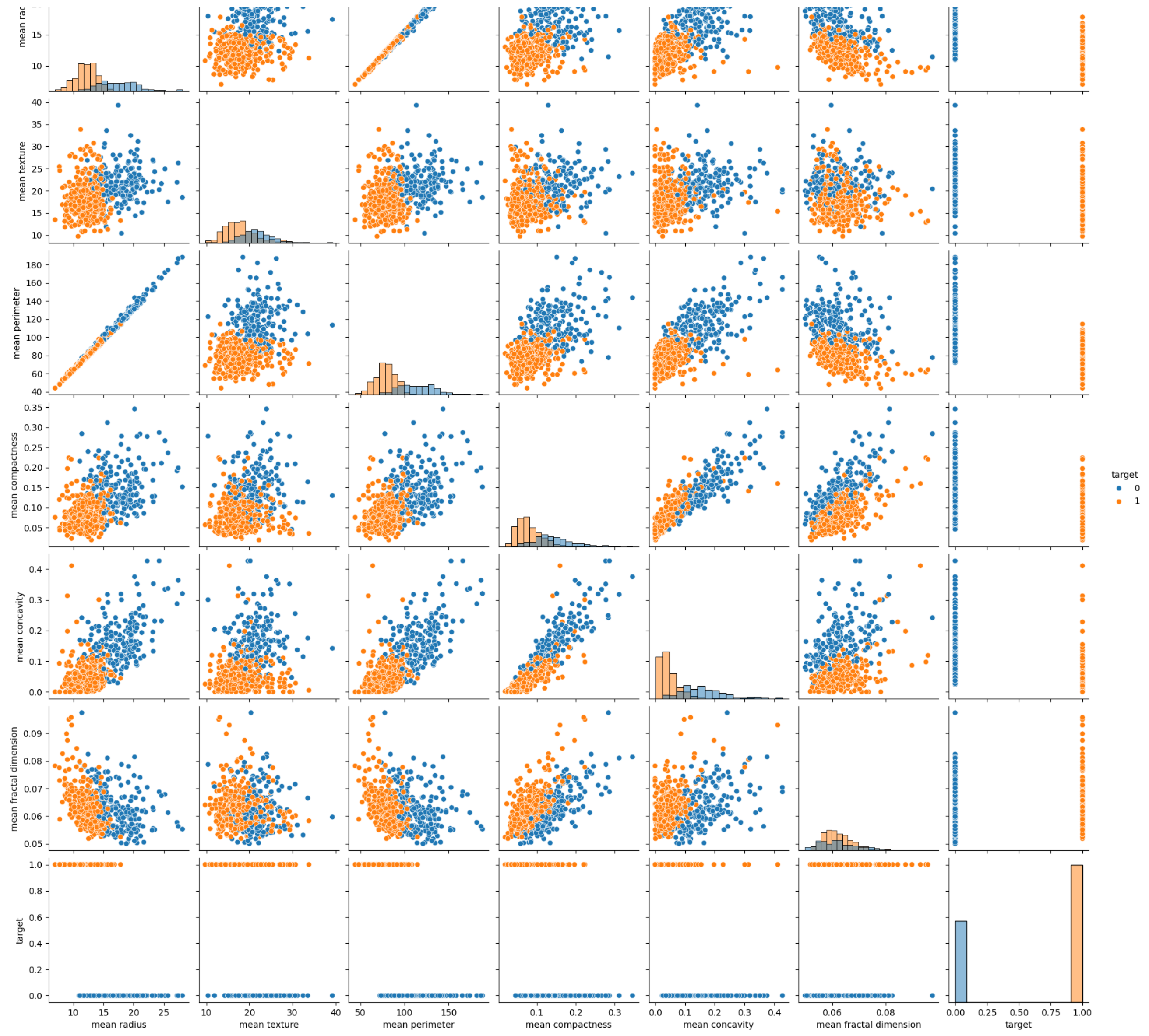
Question 12: In one figure plot all the pairs available in this set of features for different target classes: `mean radius`, `mean texture`, `mean perimeter`, `mean compactness`, `mean concavity`, `mean fractal dimension`, `target`. Make sure that the regression lines are plotted as well. Use Seaborn library for this purpose and have a look at this: https://seaborn.pydata.org/generated/seaborn.pairplot.html.

```
features = ['mean radius', 'mean texture', 'mean perimeter', 'mean compactness', 'mean conc

sns.pairplot(df, vars=features, hue='target', diag_kind='hist')

plt.show()
```




mean radius
mean texture
mean perimeter
mean compactness
mean concavity
mean fractal dimension
target
target
● 0
● 1