Lab 9 - Dimensionality Reduction I [CAC 3A - Lab 1]

Submitted By
Name: Harsha KG
Register Number: 19112005
Class: **5 BSc Data Science**

Lab Overview

Part A. Perform PCA and LDA on Breast Cancer Dataset, write down your obsevations. While loading, use the toy dataset available in SKLearn (load_breast_cancer)

Part B. Illustrate the effect of changing various method parameters of PCA and LDA. Compare the accuracies, and provide visualizations and interpretations for the evaluation metrices.

Part C. Illustrate the usage of make_classification methods and make_multilabel_classification in Sklearn and perform Dimensionality Reduction on it.

Part D. "PCA could be used in applications such as Image Processing, to reduce the complexity of data and improve performance or to compress images". Justify this statement with your own findings.

Objectives

Understand the dataset and perform PCA and LDA and give valid reason for using it in dataset.

Problem Definition

Understand the Dataset & Features and then perform preprocessing technique and statistical analysis to get insights and then perform PCA and LD and understand the usage of it in this dataset.

Approach

Imported the Dataset using Sklearn Library to notebook .Did some pre-processing technique and then build the PCA and LDA and after that did a accuracy checking by changing paramter values.Understood the usage of make_classification methods and make_multilabel_classification in Sklearn and perform Dimensionality Reduction on it and function of PCA in image compression.

Sections

References

Datasets: https://scikit-learn.org/stable/datasets/toy_dataset.html (https://scikit-learn.org/stable/datasets/toy_dataset.html) http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html (http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html) http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_multilabel_classification.html (http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_multilabel_classification.html)

PCA https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html) https://towardsdatascience.com/principal-component-analysis-for-breast-cancer-data-with-r-and-python-b312d28e911f (https://towardsdatascience.com/principal-component-analysis-for-breast-cancer-data-with-r-and-python-b312d28e911f) https://www.kaggle.com/jahirmorenoa/pca-to-the-breast-cancer-data-set (https://www.kaggle.com/jahirmorenoa/pca-to-the-breast-cancer-data-set) https://www.youtube.com/watch?v=e2sM7ccaA9c&ab_channel=DigitalSreeni (https://www.youtube.com/watch?v=e2sM7ccaA9c&ab_channel=DigitalSreeni) https://www.datacamp.com/community/tutorials/principal-component-analysis-in-python (https://www.datacamp.com/community/tutorials/principal-component-analysis-in-python) https://towardsdatascience.com/dimensionality-reduction-of-a-color-photo-splitting-into-rgb-channels-using-pca-algorithm-in-python-ba01580a1118 (https://towardsdatascience.com/dimensionality-reduction-of-a-color-photo-splitting-into-rgb-channels-using-pca-algorithm-in-python-ba01580a1118) https://www.kaggle.com/mirzarahim/introduction-to-pca-image-compression-example (https://www.kaggle.com/mirzarahim/introduction-to-pca-image-compression-example) https://github.com/gtraskas/breast_cancer_prediction/blob/master/breast_cancer.ipynb (https://github.com/gtraskas/breast_cancer_prediction/blob/master/breast_cancer.ipynb)

LDA http://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html (http://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html) https://machinelearningmastery.com/linear-discriminant-analysis-with-python/ (https://machinelearningmastery.com/linear-discriminant-analysis-with-python/) https://towardsdatascience.com/linear-discriminant-analysis-in-python-76b8b17817c2 (https://towardsdatascience.com/linear-discriminant-analysis-in-python-76b8b17817c2) https://www.mygreatlearning.com/blog/linear-discriminant-analysis-or-lda/ (https://www.mygreatlearning.com/blog/linear-discriminant-analysis-or-lda/) https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/ (https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/)

PCA and LDA

PCA is an unsupervised pre-processing task that is carried out before applying any ML algorithm. PCA is based on “orthogonal linear transformation” which is a mathematical technique to project the attributes of a data set onto a new coordinate system. The attribute which describes the most variance is called the first principal component and is placed at the first coordinate. Similarly, the attribute which stands second in describing variance is called a second principal component and so on. In short, the complete dataset can be expressed in terms of principal components. Usually, more than 90% of the variance is explained by two/three principal components. Principal component analysis, or PCA, thus converts data from high dimensional space to low dimensional space by selecting the most important attributes that capture maximum information about the dataset.

Linear Discriminant Analysis or Normal Discriminant Analysis or Discriminant Function Analysis is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modelling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space. For example, we have two classes and we need to separate them efficiently. Classes can have multiple features. It works by calculating summary statistics for the input features by class label, such as the mean and standard deviation. These statistics represent the model learned from the training data. In practice, linear algebra operations are used to calculate the required quantities efficiently via matrix decomposition. LDA assumes that the input variables are numeric and normally distributed and that they have the same variance (spread). If this is not the case, it may be desirable to transform the data to have a Gaussian distribution and standardize or normalize the data prior to modeling.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Part A

Perform PCA and LDA on Breast Cancer Dataset, write down your obsevations. While loading, use the toy dataset available in SKLearn (load_breast_cancer)

In [2]:
```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
data
```

Out[2]:
```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'frame': None,
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) dataset\n
--------------------------------------------\n\n**Data Set Characteristics:**\n\n    :N
umber of Instances: 569\n\n    :Number of Attributes: 30 numeric, predictive attributes
and the class\n\n    :Attribute Information:\n        - radius (mean of distances from
center to points on the perimeter)\n        - texture (standard deviation of gray-scale
values)\n        - perimeter\n        - area\n        - smoothness (local variation in
radius lengths)\n        - compactness (perimeter^2 / area - 1.0)\n        - concavity
(severity of concave portions of the contour)\n        - concave points (number of conc
ave portions of the contour)\n        - symmetry\n        - fractal dimension ("coastli
ne approximation" - 1)\n\n        The mean, standard error, and "worst" or largest (mea
n of the three\n        worst/largest values) of these features were computed for each
image,\n        resulting in 30 features.  For instance, field 0 is Mean Radius, field
\n        10 is Radius SE, field 20 is Worst Radius.\n\n        - class:\n
- WDBC-Malignant\n                - WDBC-Benign\n\n    :Summary Statistics:\n\n    ====
================================= ====== ======\n
Min    Max\n    ===================================== ====== ======\n    radius (mean):
6.981  28.11\n    texture (mean):                       9.71   39.28\n    perimeter (me
an):                 43.79  188.5\n    area (mean):                          143.5
2501.0\n    smoothness (mean):                    0.053  0.163\n    compactness (mean):
0.019  0.345\n    concavity (mean):                     0.0    0.427\n    concave point
s (mean):              0.0    0.201\n    symmetry (mean):                      0.106
0.304\n    fractal dimension (mean):             0.05   0.097\n    radius (standard err
or):              0.112  2.873\n    texture (standard error):             0.36   4.885
\n    perimeter (standard error):           0.757  21.98\n    area (standard error):
6.802  542.2\n    smoothness (standard error):          0.002  0.031\n    compactness
(standard error):          0.002  0.135\n    concavity (standard error):           0.0
0.396\n    concave points (standard error):      0.0    0.053\n    symmetry (standard e
rror):              0.008  0.079\n    fractal dimension (standard error):   0.001  0.03\n
```

```
radius (worst):                       7.93   36.04\n    texture (worst):
12.02  49.54\n    perimeter (worst):                    50.41  251.2\n    area (worst):
185.2  4254.0\n    smoothness (worst):                    0.071  0.223\n    compactness
(worst):                 0.027  1.058\n    concavity (worst):                   0.0
1.252\n    concave points (worst):              0.0    0.291\n    symmetry (worst):
0.156  0.664\n    fractal dimension (worst):           0.055  0.208\n    =============
======================= ====== ======\n\n    :Missing Attribute Values: None\n\n    :C
lass Distribution: 212 - Malignant, 357 - Benign\n\n    :Creator:  Dr. William H. Wolbe
rg, W. Nick Street, Olvi L. Mangasarian\n\n    :Donor: Nick Street\n\n    :Date: Novemb
er, 1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.\nht
tps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of a fine needle\na
spirate (FNA) of a breast mass.  They describe\ncharacteristics of the cell nuclei pres
ent in the image.\n\nSeparating plane described above was obtained using\nMultisurface
Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nConstruction Via Linear Programmin
g." Proceedings of the 4th\nMidwest Artificial Intelligence and Cognitive Science Socie
ty,\npp. 97-101, 1992], a classification method which uses linear\nprogramming to const
ruct a decision tree.  Relevant features\nwere selected using an exhaustive search in t
he space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual linear program used
to obtain the separating plane\nin the 3-dimensional space is that described in:\n[K.
P. Bennett and O. L. Mangasarian: "Robust Linear\nProgramming Discrimination of Two Lin
early Inseparable Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis da
tabase is also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-
prog/cpo-dataset/machine-learn/WDBC/\n\n.. topic:: References\n\n   - W.N. Street, W.H.
Wolberg and O.L. Mangasarian. Nuclear feature extraction \n     for breast tumor diagno
sis. IS&T/SPIE 1993 International Symposium on \n     Electronic Imaging: Science and T
echnology, volume 1905, pages 861-870,\n     San Jose, CA, 1993.\n   - O.L. Mangasaria
n, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and \n     prognosis via linea
r programming. Operations Research, 43(4), pages 570-577, \n     July-August 1995.\n
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques\n     to
diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) \n     163-
171.',
 'feature_names': array(['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'], dtype='<U23'),
 'filename': 'C:\\Users\\HP\\anaconda3\\anacondaorginal\\lib\\site-packages\\sklearn\\d
atasets\\data\\breast_cancer.csv'}
```

In [3]:
```
df = pd.DataFrame(data.data, columns = data.feature_names)
df['Target'] = pd.Series(data.target)
```

In [4]:
```
df.shape
```

Out[4]:
```
(569, 31)
```

In [5]:
```
df.columns
```

Out[5]:
```
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')
```

In [6]:
```
df.head()
```

Out[6]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883

5 rows × 31 columns

In [7]:
```
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    int32  
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

In [8]:
```
df.isna().sum()
df.isnull().sum()
```

Out[8]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
Target                     0
dtype: int64
```

In [9]:
```
df.describe()
```

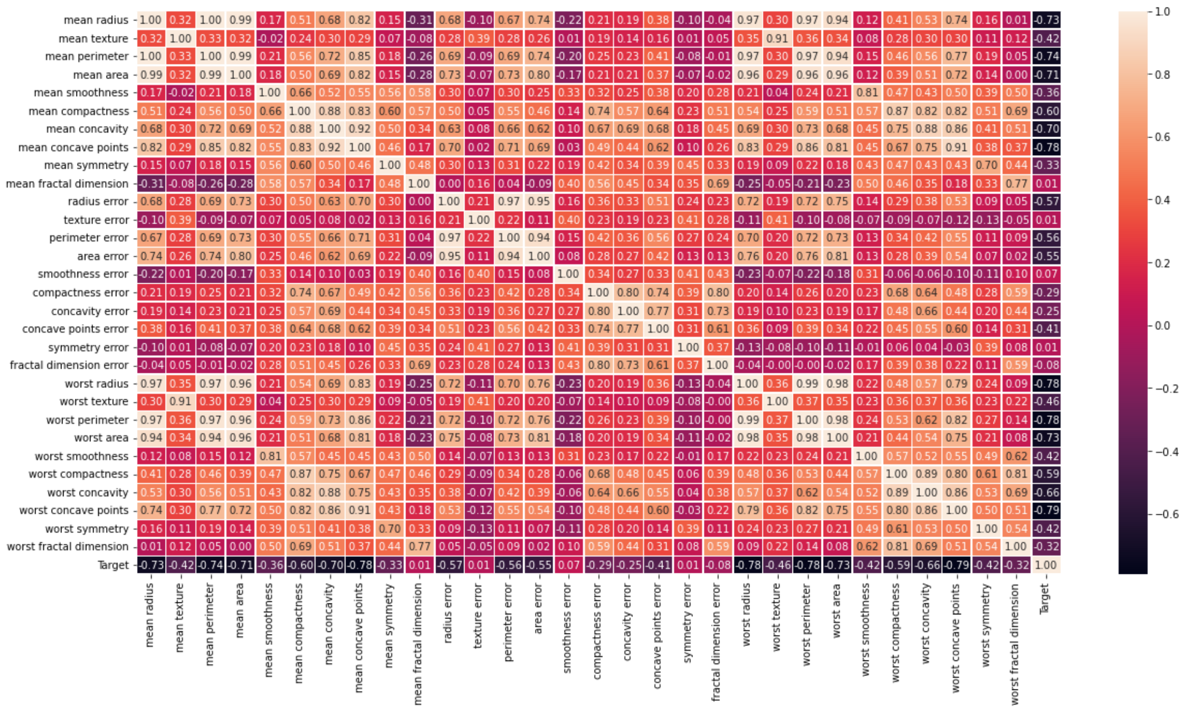
Out[9]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200

8 rows × 31 columns

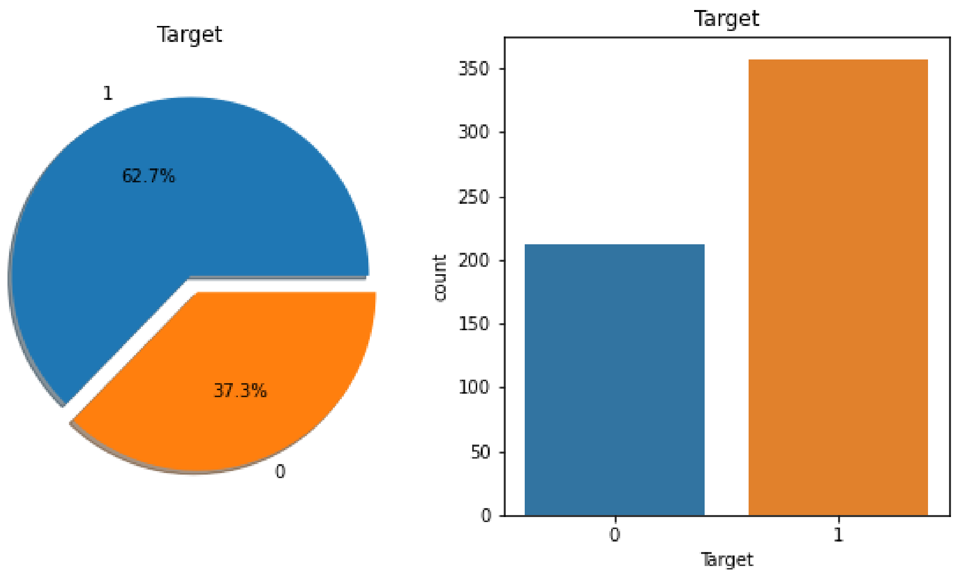
In [10]:
```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(),annot=True, fmt=".2f",annot_kws={"size":10},linewidths=.7)
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x29f1df7bf10>



In [11]:
```
BC1=df.copy()
```

In [12]:
```
f,ax=plt.subplots(1,2,figsize=(10,5))
df['Target'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('Target')
ax[0].set_ylabel('')
sns.countplot('Target',data=df,ax=ax[1])
ax[1].set_title('Target')
plt.show()
```



In [13]:
```
df['Target'].replace(0, 'Malignant',inplace = True)
df['Target'].replace(1, 'Benign',inplace = True)
```

In [14]:
```
df
```

Out[14]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mea fracta dimensio
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.0787
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.0566
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.0599
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.0974
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.0588
...	...	...	...	...	...	...	...	...	...	.
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.0562
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.0553
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.0564
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.0701
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.0588

569 rows × 31 columns

In [15]:
```
df_features = df.drop(['Target'], axis=1)
```

In [16]:
```
df_label = BC1['Target']
```

In [17]:
```
from sklearn.preprocessing import StandardScaler
```

In [18]:
```
standardized = StandardScaler()
```

In [19]:
```
standardized.fit(df_features)
```

Out[19]:
```
StandardScaler()
```

In [20]:
```
scaled_data = standardized.transform(df_features)
```

In [21]:
```
scaled_data
```

Out[21]:
```
array([[ 1.09706398, -2.07333501,  1.26993369, ...,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, ...,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, ...,  1.95500035,
         1.152255  ,  0.20139121],
       ...,
       [ 0.70228425,  2.0455738 ,  0.67267578, ...,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, ...,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, ..., -1.74506282,
        -0.04813821, -0.75120669]])
```

In [22]:
```
scaled_data.shape
```

Out[22]:
```
(569, 30)
```

In [23]:
```
from sklearn.decomposition import PCA
```

In [24]:
```
pca = PCA(n_components=3)
```

In [25]:
```
pca.fit(scaled_data)
```

Out[25]:
```
PCA(n_components=3)
```

In [26]:
```
x_pca = pca.transform(scaled_data)
```

In [27]:
```
scaled_data.shape
```

Out[27]:
```
(569, 30)
```

In [28]:
```
x_pca.shape
```

Out[28]:
```
(569, 3)
```

In [29]:
```
def diag(x):
    if x =='Malignant':
        return 1
    else:
        return 0
df_diag= df['Target'].apply(diag)
```

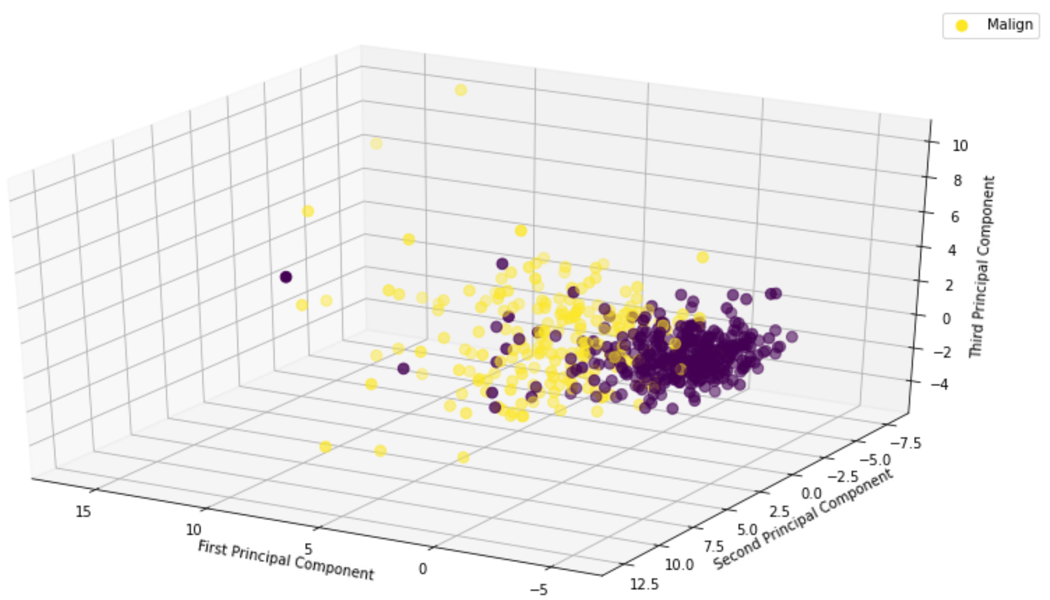
In [30]:
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [31]:
```
x_pca[:1]
```

Out[31]:
```
array([[ 9.19283683,  1.9485831 , -1.12316545]])
```

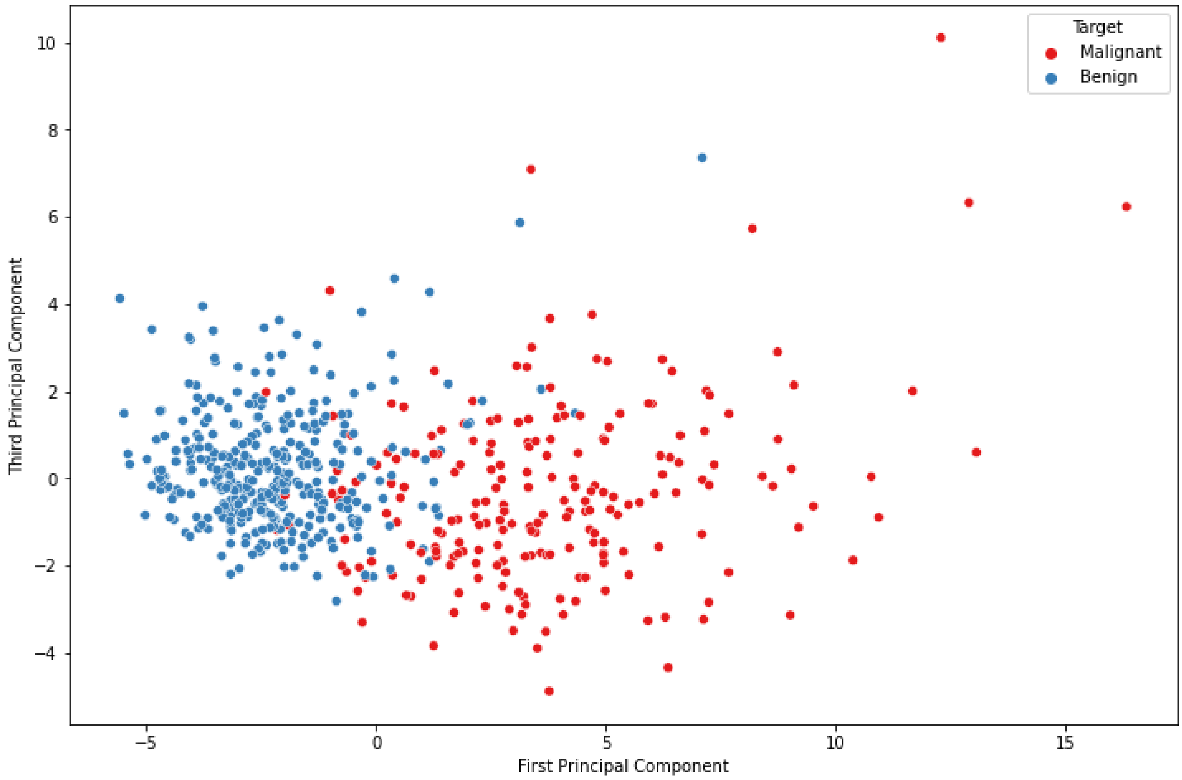
In [32]:
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [33]:
```
fig = plt.figure(figsize=(15, 8))
ax = fig.add_subplot(111, projection = '3d')
ax.scatter(x_pca[:,0], x_pca[:,1], x_pca[:,2], c = df_diag, s = 60)
ax.legend(['Malign'])
ax.set_xlabel('First Principal Component')
ax.set_ylabel('Second Principal Component')
ax.set_zlabel('Third Principal Component')
ax.view_init(30, 120)
```



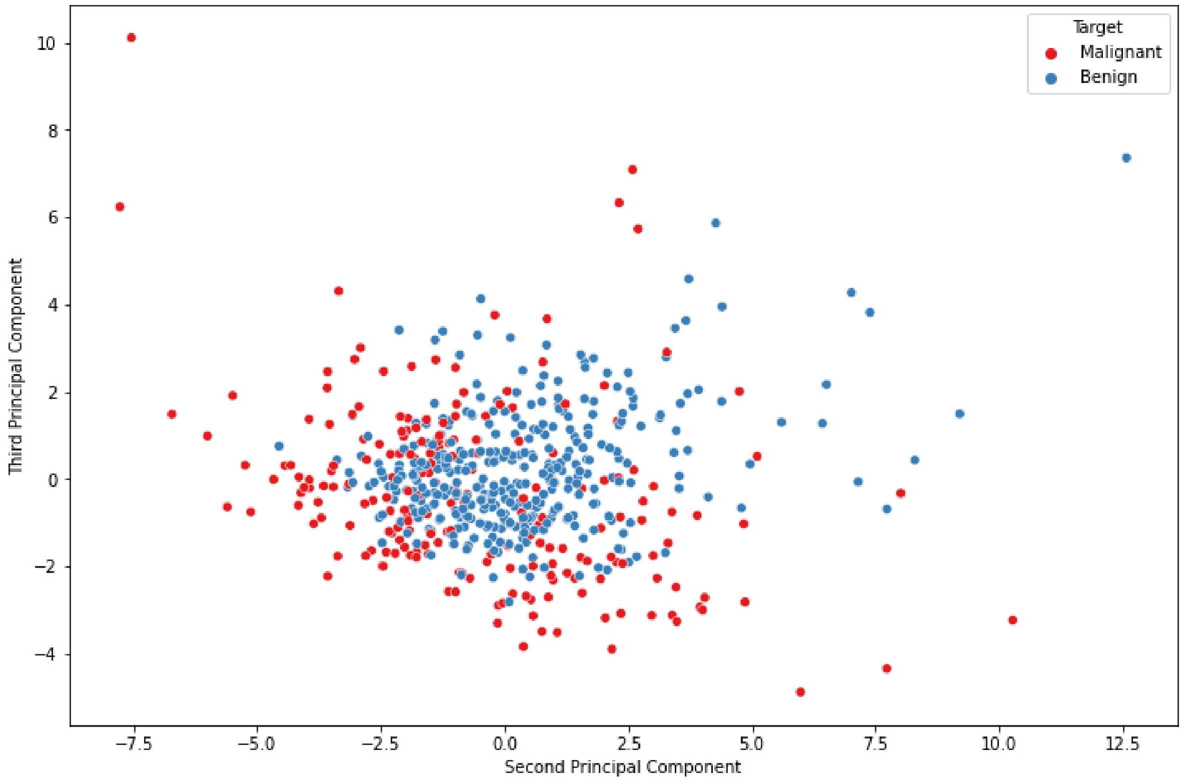
In [34]:
```
ax = plt.figure(figsize=(12,8))
sns.scatterplot(x_pca[:,0], x_pca[:,2],hue=df['Target'], palette ='Set1' )
plt.xlabel('First Principal Component')
plt.ylabel('Third Principal Component')
```

Out[34]: Text(0, 0.5, 'Third Principal Component')



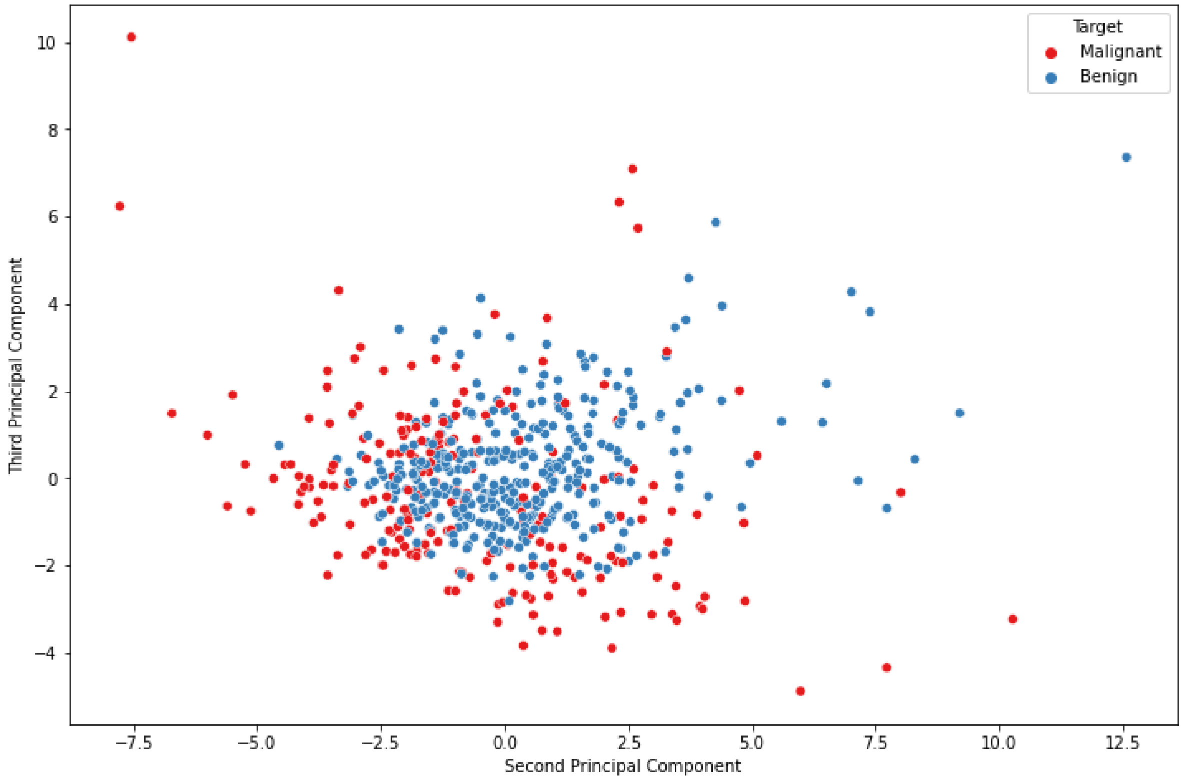
In [35]:
```
ax = plt.figure(figsize=(12,8))
sns.scatterplot(x_pca[:,1], x_pca[:,2],hue=df['Target'], palette ='Set1' )
plt.xlabel('Second Principal Component')
plt.ylabel('Third Principal Component')
```

Out[35]: Text(0, 0.5, 'Third Principal Component')



In [36]:
```
ax = plt.figure(figsize=(12,8))
sns.scatterplot(x_pca[:,1], x_pca[:,2],hue=df['Target'], palette ='Set1' )
plt.xlabel('Second Principal Component')
plt.ylabel('Third Principal Component')
```

Out[36]: Text(0, 0.5, 'Third Principal Component')



In [37]:
```
df_pc = pd.DataFrame(pca.components_, columns = df_features.columns)
```

In [38]:
```
df_pc
```

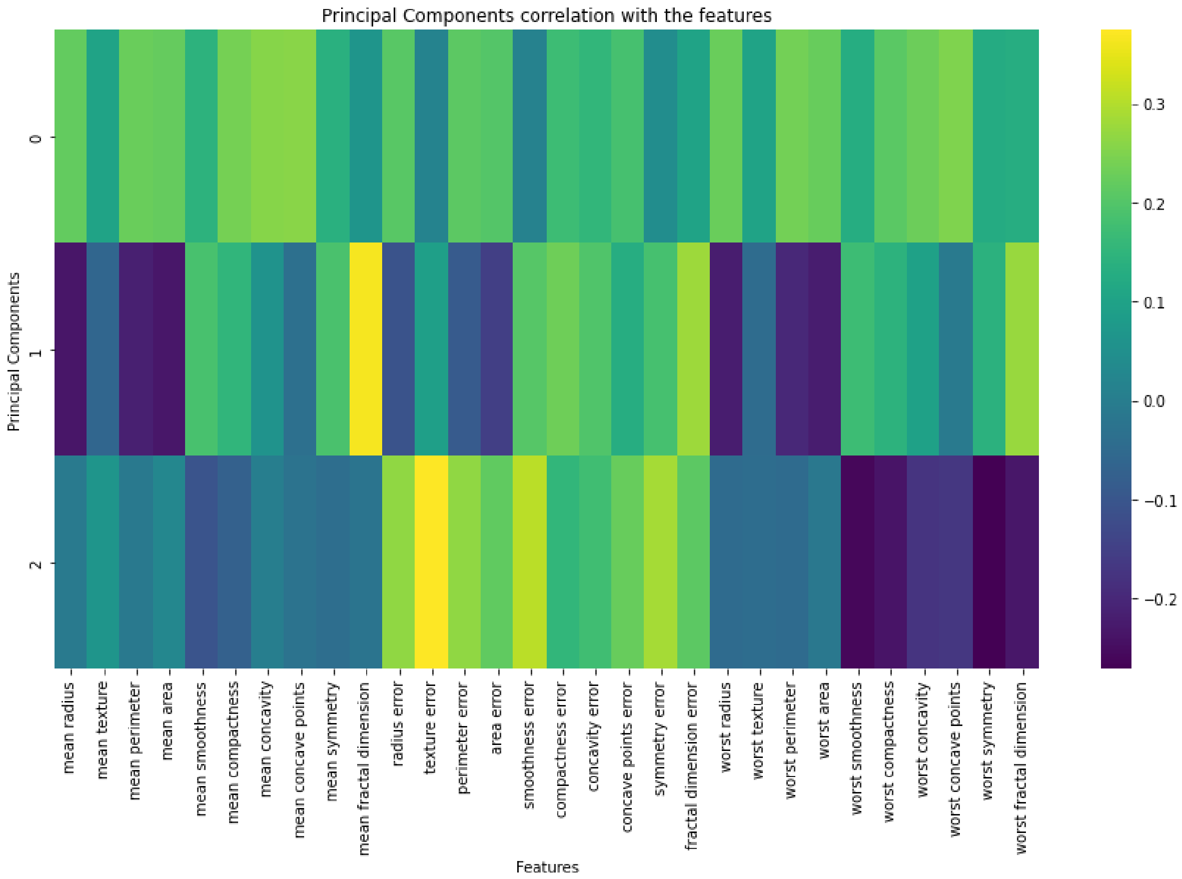
Out[38]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	di
0	0.218902	0.103725	0.227537	0.220995	0.142590	0.239285	0.258400	0.260854	0.138167	
1	-0.233857	-0.059706	-0.215181	-0.231077	0.186113	0.151892	0.060165	-0.034768	0.190349	
2	-0.008531	0.064549	-0.009314	0.028699	-0.104292	-0.074092	0.002733	-0.025564	-0.040240	-

3 rows × 30 columns

In [39]:
```
plt.figure(figsize=(15, 8))
sns.heatmap(df_pc, cmap='viridis')
plt.title('Principal Components correlation with the features')
plt.xlabel('Features')
plt.ylabel('Principal Components')
```

Out[39]: `Text(114.0, 0.5, 'Principal Components')`



In [40]:
```
#####
```

In [41]:
```
X = BC1.iloc[:, 1:].values
y = BC1['Target'].values
```

In [42]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state=
0)
```

In [43]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [44]:
```
pca = PCA(n_components = 3)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

In [45]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(max_depth = 2, random_state = 0)
clf.fit(X_train, y_train)

# Predicting the Test set results
y_pred = clf.predict(X_test)
```

In [46]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)

print('Accuracy -> '+ str(accuracy_score(y_test, y_pred)))
```

```
[[43  4]
 [ 6 61]]
Accuracy -> 0.9122807017543859
```

In [47]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.91      0.90        47
           1       0.94      0.91      0.92        67

    accuracy                           0.91       114
   macro avg       0.91      0.91      0.91       114
weighted avg       0.91      0.91      0.91       114
```

In [48]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [49]:
```
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
y_pred=knn.predict(X_test)
```

In [50]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)

print('Accuracy -> '+ str(accuracy_score(y_test, y_pred)))
```

```
[[43  4]
 [ 2 65]]
Accuracy -> 0.9473684210526315
```

In [51]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.91      0.93        47
           1       0.94      0.97      0.96        67

    accuracy                           0.95       114
   macro avg       0.95      0.94      0.95       114
weighted avg       0.95      0.95      0.95       114
```

In [52]:
```
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

In [53]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(max_depth = 2, random_state = 0)
clf.fit(X_train, y_train)

# Predicting the Test set results
y_pred = clf.predict(X_test)
```

In [54]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)

print('Accuracy -> '+ str(accuracy_score(y_test, y_pred)))
```

```
[[42  5]
 [ 6 61]]
Accuracy -> 0.9035087719298246
```

In [55]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.89      0.88        47
           1       0.92      0.91      0.92        67

    accuracy                           0.90       114
   macro avg       0.90      0.90      0.90       114
weighted avg       0.90      0.90      0.90       114
```

In [56]:
```
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
y_pred=knn.predict(X_test)
```

In [57]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)

print('Accuracy -> '+ str(accuracy_score(y_test, y_pred)))
```

```
[[46  1]
 [ 4 63]]
Accuracy -> 0.956140350877193
```

In [58]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95        47
           1       0.98      0.94      0.96        67

    accuracy                           0.96       114
   macro avg       0.95      0.96      0.96       114
weighted avg       0.96      0.96      0.96       114
```

In [59]:
```
pca = PCA(n_components = 1)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

In [60]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(max_depth = 2, random_state = 0)
clf.fit(X_train, y_train)

# Predicting the Test set results
y_pred = clf.predict(X_test)
```

In [61]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)

print('Accuracy -> '+ str(accuracy_score(y_test, y_pred)))
```

```
[[43  4]
 [10 57]]
Accuracy -> 0.8771929824561403
```

In [62]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.91      0.86        47
           1       0.93      0.85      0.89        67

    accuracy                           0.88       114
   macro avg       0.87      0.88      0.88       114
weighted avg       0.88      0.88      0.88       114
```

In [63]:
```
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
y_pred=knn.predict(X_test)
```

In [64]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)

print('Accuracy -> '+ str(accuracy_score(y_test, y_pred)))
```

```
[[43  4]
 [ 9 58]]
Accuracy -> 0.8859649122807017
```

In [65]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.91      0.87        47
           1       0.94      0.87      0.90        67

    accuracy                           0.89       114
   macro avg       0.88      0.89      0.88       114
weighted avg       0.89      0.89      0.89       114
```

LDA

In [66]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state=
40)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [67]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components = 1)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
```

In [68]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(max_depth = 2, random_state = 0)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

In [69]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)

print('Accuracy -> ' + str(accuracy_score(y_test, y_pred)))
```

```
[[39  0]
 [ 0 75]]
Accuracy -> 1.0
```

In [103]:
```
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
y_pred=knn.predict(X_test)
```

In [104]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)

print('Accuracy -> ' + str(accuracy_score(y_test, y_pred)))
```

```
[[39  0]
 [ 0 75]]
Accuracy -> 1.0
```

Part B

llustrate the effect of changing various method parameters of PCA and LDA. Compare the accuracies, and provide visualizations and interpretations for the evaluation metrices.

PCA

In [70]:
```
def doKNeighborsClassifier(X, y, randomstate = None,compo=8,featu='auto'):
    X_train, X_test, Y_train, Y_test = train_test_split(X, y)
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
    pca = PCA(n_components = compo,random_state=randomstate,svd_solver=featu)
    X_train = pca.fit_transform(X_train)
    X_test = pca.transform(X_test)
    cls1 = KNeighborsClassifier()
    cls1.fit(X_train,Y_train)
    predA = cls1.predict(X_test)
    acc_score = accuracy_score(predA, Y_test)
    return acc_score
```

In [71]:
```
df1 = pd.DataFrame(columns = ['Random_States','KNN_Accuracy'])
n_comp= [2,8,17,20]
random_states = [45, 21, 42, 22]
svd_sol=['auto', 'full', 'arpack', 'randomized']
```

In [72]:
```
for nc in n_comp:
    for r_state in random_states:
        for rt in svd_sol:
            a = doKNeighborsClassifier(X, y, r_state,nc,rt)
            I = {}
            I['Random_States'] = r_state
            I['number of components'] = nc
            I['solver'] = rt
            I['KNN_Accuracy'] = a
            df1 = df1.append(I, ignore_index = True)
```

In [106]:
```
df1
```

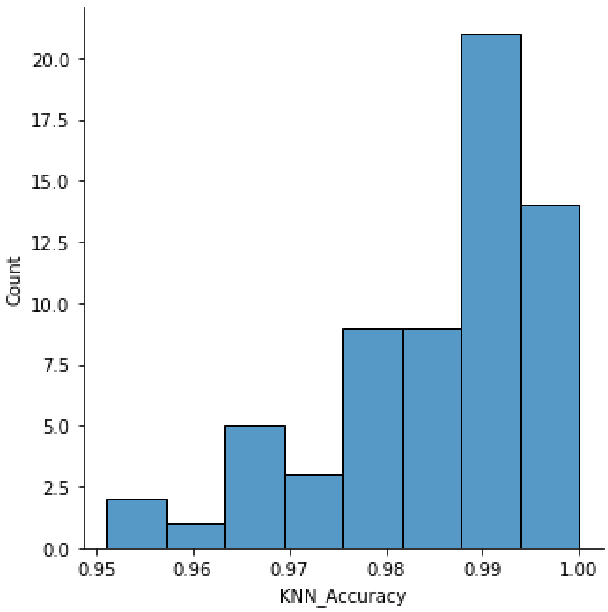
Out[106]:

	Random_States	KNN_Accuracy	number of components	solver
0	45	0.979021	2.0	auto
1	45	0.965035	2.0	full
2	45	0.965035	2.0	arpack
3	45	0.958042	2.0	randomized
4	21	0.979021	2.0	auto
...	...	...	...	...
59	42	0.986014	20.0	randomized
60	22	0.986014	20.0	auto
61	22	0.993007	20.0	full
62	22	0.993007	20.0	arpack
63	22	1.000000	20.0	randomized

64 rows × 4 columns

In [74]:
```
sns.displot(x = 'KNN_Accuracy', data = df1)
```

Out[74]: <seaborn.axisgrid.FacetGrid at 0x29f212c6c70>



In [121]:
```
from sklearn.model_selection import train_test_split
# Create a train/test split using 30% test size.
features_train, features_test, labels_train, labels_test = train_test_split(df_features,
 df_label,
test_size=0.3,
 random_state=42)
# Check the split printing the shape of each set.
print(features_train.shape, labels_train.shape)
print(features_test.shape, labels_test.shape)
```

```
(398, 30) (398,)
(171, 30) (171,)
```

In [123]:
```
# Evalution of KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from time import time
def print_ml_results():
    t0 = time()
 # Create classifier.
    clf = KNeighborsClassifier()
 # Fit the classifier on the training features and labels.
    t0 = time()
    clf.fit(features_train, labels_train)
    print("Training time:", round(time()-t0, 3), "s")
 # Make predictions.
    t1 = time()
    predictions = clf.predict(features_test)
    print("Prediction time:", round(time()-t1, 3), "s")
    # Evaluate the model.
    accuracy = clf.score(features_test, labels_test)
    report = classification_report(labels_test, predictions)
 # Print the reports.
    print("\nReport:\n")
    print("Accuracy: {}".format(accuracy))
    print("\n", report)
    print(confusion_matrix(labels_test, predictions))
print_ml_results()
```

```
Training time: 0.011 s
Prediction time: 0.018 s

Report:

Accuracy: 0.9590643274853801

              precision    recall  f1-score   support

           0       0.98      0.90      0.94        63
           1       0.95      0.99      0.97       108

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171

[[ 57   6]
 [  1 107]]
```

LDA

In [ ]:

In [101]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components = 1,solver='eigen',shrinkage=0.5)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
```

In [107]:
```
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
y_pred=knn.predict(X_test)
```

In [108]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)

print('Accuracy -> ' + str(accuracy_score(y_test, y_pred)))
```

```
[[39  0]
 [ 0 75]]
Accuracy -> 1.0
```

In [ ]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components = 1,solver='eigen',shrinkage=0.5)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
```

In [109]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components = 1,solver='eigen',shrinkage=0.25)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
```

In [110]:
```
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
y_pred=knn.predict(X_test)
```

In [111]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)

print('Accuracy -> ' + str(accuracy_score(y_test, y_pred)))
```

```
[[39  0]
 [ 0 75]]
Accuracy -> 1.0
```

In [75]:
```
# grid search solver for lda
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_redundant=
0, random_state=1)
# define model
model = LinearDiscriminantAnalysis()
# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['solver'] = ['svd', 'lsqr', 'eigen']
# define search
search = GridSearchCV(model, grid, scoring='accuracy', cv=cv, n_jobs=-1)
# perform the search
results = search.fit(X, y)
# summarize
print('Mean Accuracy: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
```

```
Mean Accuracy: 0.893
Config: {'solver': 'svd'}
```

In [76]:
```
# grid search shrinkage for lda
from numpy import arange
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_redundant=
0, random_state=1)
# define model
model = LinearDiscriminantAnalysis(solver='lsqr')
# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['shrinkage'] = arange(0, 1, 0.01)
# define search
search = GridSearchCV(model, grid, scoring='accuracy', cv=cv, n_jobs=-1)
# perform the search
results = search.fit(X, y)
# summarize
print('Mean Accuracy: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
```

```
Mean Accuracy: 0.894
Config: {'shrinkage': 0.02}
```

Part C

llustrate the usage of make_classification methods and make_multilabel_classification in Sklearn and perform Dimensionality Reduction on it.

make_classification methods

In [77]:
```
# create the lda model
model = LDA()
```

In [78]:
```
from sklearn.datasets import make_classification
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_redundant=
0, random_state=1)
# summarize the dataset
print(X.shape, y.shape)
```

```
(1000, 10) (1000,)
```

In [79]:
```
# evaluate a lda model on the dataset
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_redundant=
0, random_state=1)
# define model
model = LinearDiscriminantAnalysis()
# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# summarize result
print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Mean Accuracy: 0.893 (0.033)
```

In [80]:
```
from sklearn.datasets import make_classification
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_redundant=
0, random_state=1)
# define model
model = LinearDiscriminantAnalysis()
# fit model
model.fit(X, y)
# define new data
row = [0.12777556,-3.64400522,-2.23268854,-1.82114386,1.75466361,0.1243966,1.03397657,2.
35822076,1.01001752,0.56768485]
# make a prediction
yhat = model.predict([row])
# summarize prediction
print('Predicted Class: %d' % yhat)
```

```
Predicted Class: 1
```

In [81]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_redundant=
0, random_state=1)
# define model
model = LinearDiscriminantAnalysis()
# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['solver'] = ['svd', 'lsqr', 'eigen']
# define search
search = GridSearchCV(model, grid, scoring='accuracy', cv=cv, n_jobs=-1)
# perform the search
results = search.fit(X, y)
# summarize
print('Mean Accuracy: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
```

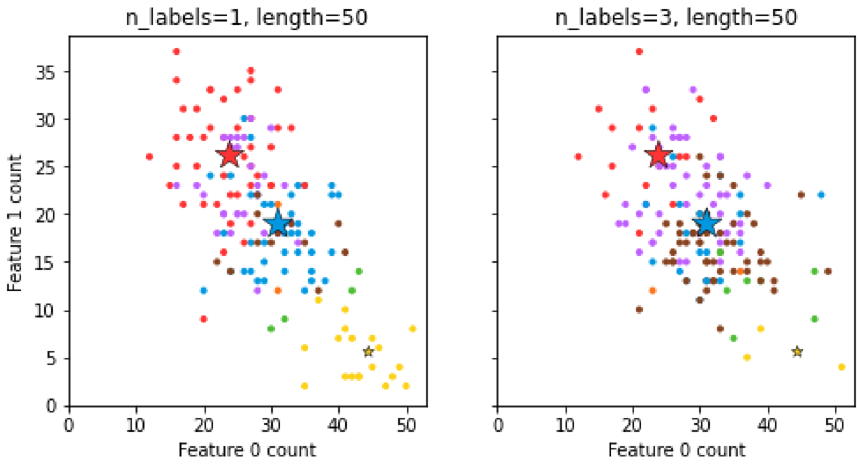
```
Mean Accuracy: 0.893
Config: {'solver': 'svd'}
```

In [82]:
```
# grid search shrinkage for lda
from numpy import arange
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_redundant=
0, random_state=1)
# define model
model = LinearDiscriminantAnalysis(solver='lsqr')
# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['shrinkage'] = arange(0, 1, 0.01)
# define search
search = GridSearchCV(model, grid, scoring='accuracy', cv=cv, n_jobs=-1)
# perform the search
results = search.fit(X, y)
# summarize
print('Mean Accuracy: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
```

```
Mean Accuracy: 0.894
Config: {'shrinkage': 0.02}
```

make_multilabel_classification

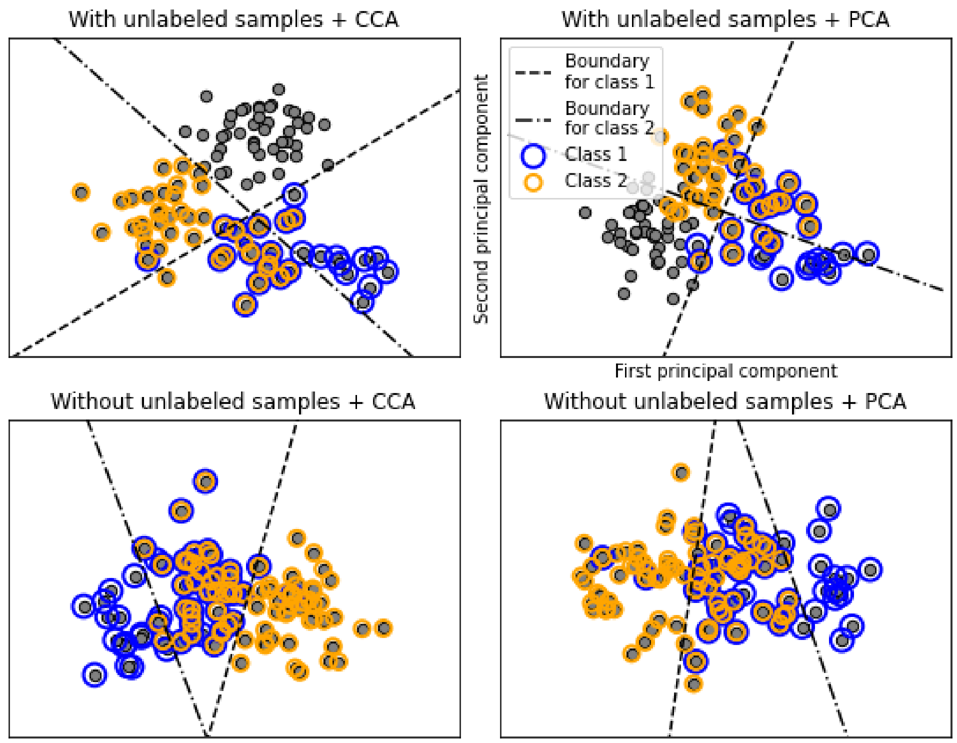
In [83]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_multilabel_classification as make_ml_clf

COLORS = np.array(
    [
        "!",
        "#FF3333",  # red
        "#0198E1",  # blue
        "#BF5FFF",  # purple
        "#FCD116",  # yellow
        "#FF7216",  # orange
        "#4DBD33",  # green
        "#87421F",  # brown
    ]
)

# Use same random seed for multiple calls to make_multilabel_classification to
# ensure same distributions
RANDOM_SEED = np.random.randint(2 ** 10)


def plot_2d(ax, n_labels=1, n_classes=3, length=50):
    X, Y, p_c, p_w_c = make_ml_clf(
        n_samples=150,
        n_features=2,
        n_classes=n_classes,
        n_labels=n_labels,
        length=length,
        allow_unlabeled=False,
        return_distributions=True,
        random_state=RANDOM_SEED,
    )

    ax.scatter(
        X[:, 0], X[:, 1], color=COLORS.take((Y * [1, 2, 4]).sum(axis=1)), marker="."
    )
    ax.scatter(
        p_w_c[0] * length,
        p_w_c[1] * length,
        marker="*",
        linewidth=0.5,
        edgecolor="black",
        s=20 + 1500 * p_c ** 2,
        color=COLORS.take([1, 2, 4]),
    )
    ax.set_xlabel("Feature 0 count")
    return p_c, p_w_c


_, (ax1, ax2) = plt.subplots(1, 2, sharex="row", sharey="row", figsize=(8, 4))
plt.subplots_adjust(bottom=0.15)

p_c, p_w_c = plot_2d(ax1, n_labels=1)
ax1.set_title("n_labels=1, length=50")
ax1.set_ylabel("Feature 1 count")

plot_2d(ax2, n_labels=3)
ax2.set_title("n_labels=3, length=50")
ax2.set_xlim(left=0, auto=True)
ax2.set_ylim(bottom=0, auto=True)

plt.show()

print("The data was generated from (random_state=%d):" % RANDOM_SEED)
print("Class", "P(C)", "P(w0|C)", "P(w1|C)", sep="\t")
for k, p, p_w in zip(["red", "blue", "yellow"], p_c, p_w_c.T):
    print("%s\t%0.2f\t%0.2f\t%0.2f" % (k, p, p_w[0], p_w[1]))
```




```
The data was generated from (random_state=989):
Class   P(C)    P(w0|C) P(w1|C)
red     0.42    0.48    0.52
blue    0.45    0.62    0.38
yellow  0.13    0.89    0.11
```

In [84]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_multilabel_classification
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import CCA


def plot_hyperplane(clf, min_x, max_x, linestyle, label):
    # get the separating hyperplane
    w = clf.coef_[0]
    a = -w[0] / w[1]
    xx = np.linspace(min_x - 5, max_x + 5)  # make sure the line is long enough
    yy = a * xx - (clf.intercept_[0]) / w[1]
    plt.plot(xx, yy, linestyle, label=label)


def plot_subfigure(X, Y, subplot, title, transform):
    if transform == "pca":
        X = PCA(n_components=2).fit_transform(X)
    elif transform == "cca":
        X = CCA(n_components=2).fit(X, Y).transform(X)
    else:
        raise ValueError

    min_x = np.min(X[:, 0])
    max_x = np.max(X[:, 0])

    min_y = np.min(X[:, 1])
    max_y = np.max(X[:, 1])

    classif = OneVsRestClassifier(SVC(kernel="linear"))
    classif.fit(X, Y)

    plt.subplot(2, 2, subplot)
    plt.title(title)

    zero_class = np.where(Y[:, 0])
    one_class = np.where(Y[:, 1])
    plt.scatter(X[:, 0], X[:, 1], s=40, c="gray", edgecolors=(0, 0, 0))
    plt.scatter(
        X[zero_class, 0],
        X[zero_class, 1],
        s=160,
        edgecolors="b",
        facecolors="none",
        linewidths=2,
        label="Class 1",
    )
    plt.scatter(
        X[one_class, 0],
        X[one_class, 1],
        s=80,
        edgecolors="orange",
        facecolors="none",
        linewidths=2,
        label="Class 2",
    )

    plot_hyperplane(
        classif.estimators_[0], min_x, max_x, "k--", "Boundary\nfor class 1"
    )
    plot_hyperplane(
        classif.estimators_[1], min_x, max_x, "k-.", "Boundary\nfor class 2"
    )
    plt.xticks(())
    plt.yticks(())
```

```
    plt.xlim(min_x - 0.5 * max_x, max_x + 0.5 * max_x)
    plt.ylim(min_y - 0.5 * max_y, max_y + 0.5 * max_y)
    if subplot == 2:
        plt.xlabel("First principal component")
        plt.ylabel("Second principal component")
        plt.legend(loc="upper left")


plt.figure(figsize=(8, 6))

X, Y = make_multilabel_classification(
    n_classes=2, n_labels=1, allow_unlabeled=True, random_state=1
)

plot_subfigure(X, Y, 1, "With unlabeled samples + CCA", "cca")
plot_subfigure(X, Y, 2, "With unlabeled samples + PCA", "pca")

X, Y = make_multilabel_classification(
    n_classes=2, n_labels=1, allow_unlabeled=False, random_state=1
)

plot_subfigure(X, Y, 3, "Without unlabeled samples + CCA", "cca")
plot_subfigure(X, Y, 4, "Without unlabeled samples + PCA", "pca")

plt.subplots_adjust(0.04, 0.02, 0.97, 0.94, 0.09, 0.2)
plt.show()
```

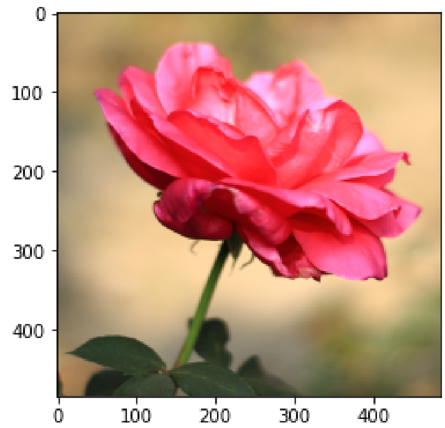


Part D

PCA could be used in applications such as Image Processing, to reduce the complexity of data and improve performance or to compress images". Justify this statement with your own findings.

In [85]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import cv2
from scipy.stats import stats
import matplotlib.image as mpimg
```

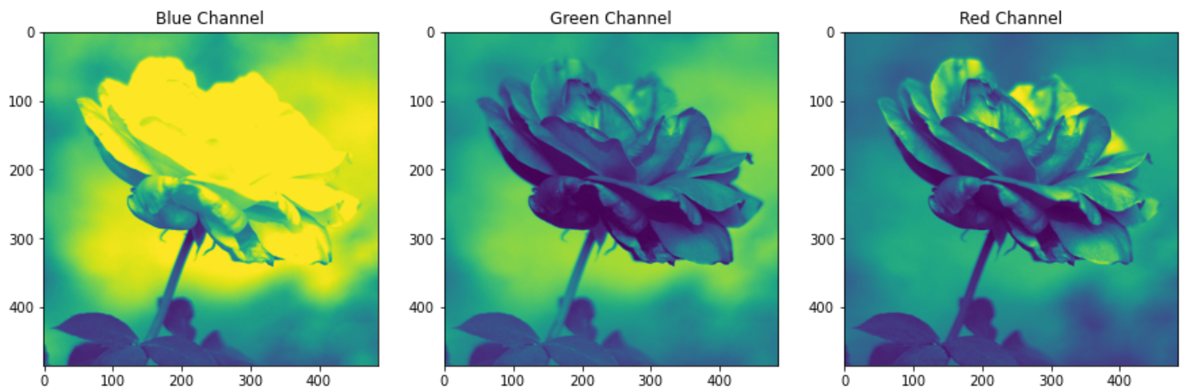
In [86]:
```
img = cv2.cvtColor(cv2.imread('rose1.png'), cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.show()
```



In [87]:
```
img.shape
```

Out[87]: (485, 485, 3)

In [88]:
```
#Splitting into channels
blue,green,red = cv2.split(img)
# Plotting the images
fig = plt.figure(figsize = (15, 7.2))
fig.add_subplot(131)
plt.title("Blue Channel")
plt.imshow(blue)
fig.add_subplot(132)
plt.title("Green Channel")
plt.imshow(green)
fig.add_subplot(133)
plt.title("Red Channel")
plt.imshow(red)
plt.show()
```



In [89]:
```
blue_temp_df = pd.DataFrame(data = blue)
blue_temp_df
```

Out[89]:

	0	1	2	3	4	5	6	7	8	9	...	475	476	477	478	479	480	481	482	483
0	181	177	175	180	177	183	183	183	185	181	...	218	216	217	217	215	215	209	210	210
1	173	180	178	180	180	177	183	175	174	181	...	218	218	219	212	213	213	219	216	213
2	182	176	181	178	181	181	179	186	185	184	...	218	212	215	217	211	212	210	212	213
3	181	176	178	181	181	181	178	182	182	177	...	214	218	218	215	217	218	215	209	210
4	180	181	176	177	180	175	180	179	181	186	...	219	216	218	218	209	214	214	209	210
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
480	37	43	59	70	82	89	104	136	163	170	...	131	133	135	131	128	134	129	125	123
481	39	36	49	58	78	88	104	118	148	170	...	142	133	130	134	129	127	126	127	122
482	65	56	42	55	73	81	95	114	133	155	...	134	134	136	131	132	131	126	124	123
483	96	81	71	54	58	73	83	95	114	129	...	136	137	134	131	129	128	128	127	121
484	131	132	118	98	77	67	82	90	89	99	...	138	138	131	134	131	130	126	128	130

485 rows × 485 columns

In [90]:
```
df_blue = blue/255
df_green = green/255
df_red = red/255
```

Fit and transform the data in PCA

We already have seen that each channel has 485 dimensions, and we will now consider only 50 dimensions for PCA and fit and transform the data and check how much variance is explained after reducing data to 50 dimensions.

In [91]:
```
pca_b = PCA(n_components=50)
pca_b.fit(df_blue)
trans_pca_b = pca_b.transform(df_blue)
pca_g = PCA(n_components=50)
pca_g.fit(df_green)
trans_pca_g = pca_g.transform(df_green)
pca_r = PCA(n_components=50)
pca_r.fit(df_red)
trans_pca_r = pca_r.transform(df_red)
```

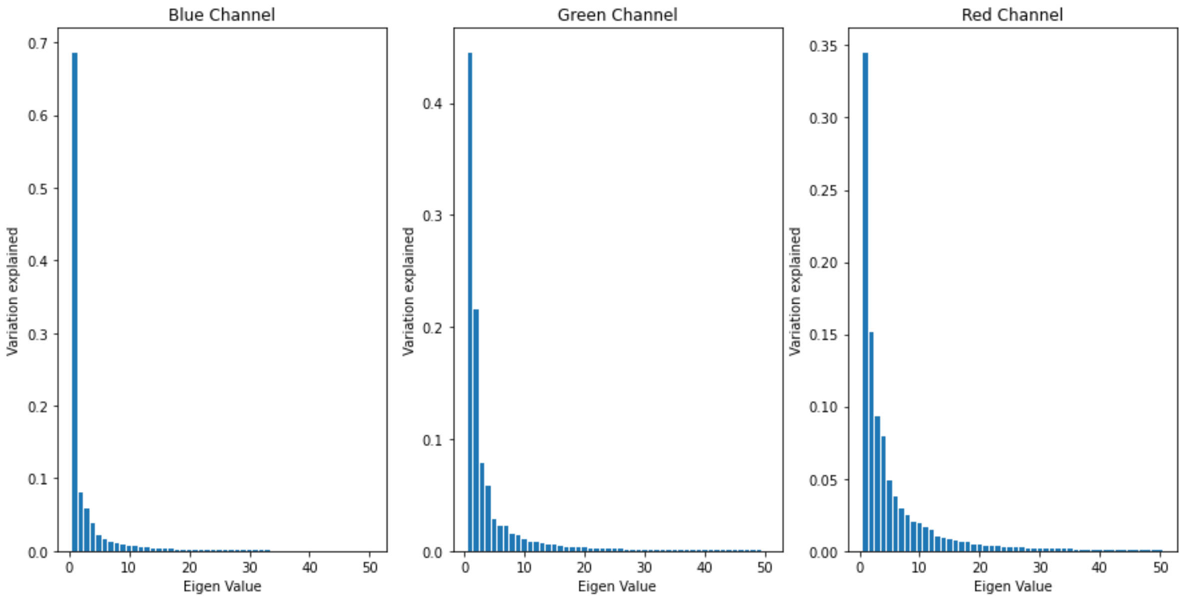
In [92]:
```
print(trans_pca_b.shape)
print(trans_pca_r.shape)
print(trans_pca_g.shape)
```

```
(485, 50)
(485, 50)
(485, 50)
```

In [93]:
```
print(f"Blue Channel : {sum(pca_b.explained_variance_ratio_)}")
print(f"Green Channel: {sum(pca_g.explained_variance_ratio_)}")
print(f"Red Channel  : {sum(pca_r.explained_variance_ratio_)}")
```

```
Blue Channel : 0.9933957242176127
Green Channel: 0.990605421269907
Red Channel  : 0.9840899526429719
```

In [94]:
```
fig = plt.figure(figsize = (15, 7.2))
fig.add_subplot(131)
plt.title("Blue Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,51)),pca_b.explained_variance_ratio_)
fig.add_subplot(132)
plt.title("Green Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,51)),pca_g.explained_variance_ratio_)
fig.add_subplot(133)
plt.title("Red Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,51)),pca_r.explained_variance_ratio_)
plt.show()
```



In [95]:
```
b_arr = pca_b.inverse_transform(trans_pca_b)
g_arr = pca_g.inverse_transform(trans_pca_g)
r_arr = pca_r.inverse_transform(trans_pca_r)
print(b_arr.shape, g_arr.shape, r_arr.shape)
```

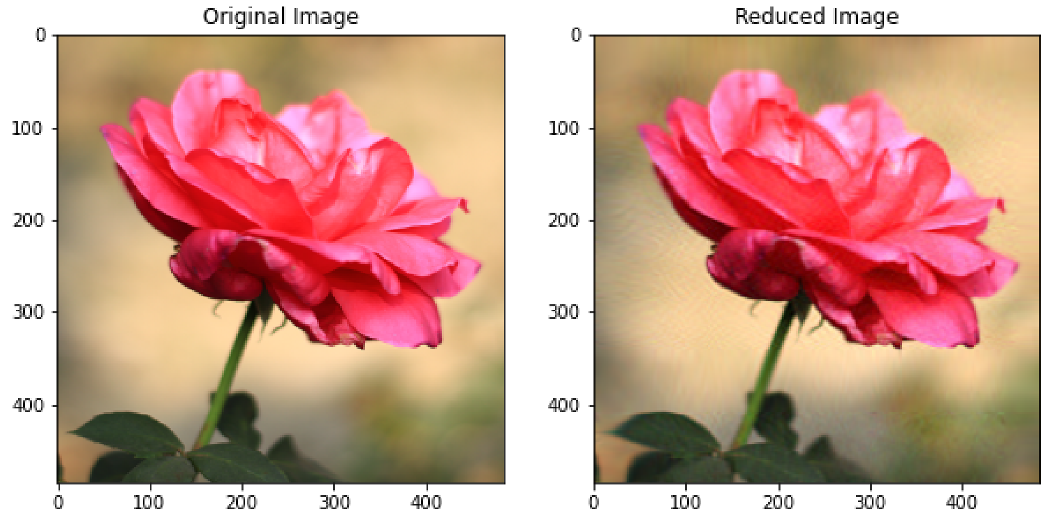
```
(485, 485) (485, 485) (485, 485)
```

In [96]:
```
img_reduced= (cv2.merge((b_arr, g_arr, r_arr)))
print(img_reduced.shape)
```

```
(485, 485, 3)
```

In [97]:
```
fig = plt.figure(figsize = (10, 7.2))
fig.add_subplot(121)
plt.title("Original Image")
plt.imshow(img)
fig.add_subplot(122)
plt.title("Reduced Image")
plt.imshow(img_reduced)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



Observation

Although we have decreased the dimension individually for each channel to only 50 from 485, the compressed image is fairly comparable (at least we can still recognise it as a rose) to that of the original one. However, we have accomplished our objective. Without a doubt, the computer will now process the smaller image considerably faster.

Final Observation

PCA=2 have higher value for both KNN and Random forest. Out of that KNN have the highest accuracy. For LDA, Mean Accuracy is 0.893 for {'solver': 'svd'} and Mean Accuracy is 0.894 for {'shrinkage': 0.02}. These are the best paramter for PCA in breast cancer. Out of the two LDA have higher accuracy than PCA where the n_component value of LDA is 1 One of the best application of Dimensionality reduction is Image Compression

Conclusion

we built and analysed multiple classification models in order to achieve high accuracy in terms of predicting breast cancer, as well as dimensionality reduction, which can aid clinicians in predicting breast cancer in patients. and later on we understood the usage of make_multilabel_classification and make classification. At the end, did an application of dimensionality reduction(image compression)

In [ ]: