Machine Learning

Submitted by,

- Name:Jagannath v v

- Reg.No.-21122024

- class:1st MSc DataScience

Overview

Objectives

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. "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.

Problem Definition

For above mentioned objectives, detailed analysis are being performed in this lab. And the purpose of the lab is to have proper understanding of the algorithms PCA and LDA.

Approach

Here i reffered internet to find some more about the PCA and LDA

In [14]:
```
import pandas as pd
import numpy as np
import matplotlib as mlt
import seaborn as sns
import matplotlib.pyplot as plt
```

In [15]:
```
from sklearn.datasets import load_breast_cancer
```

In [16]:
```
df=load_breast_cancer()
```

In [17]:
```
df_new=df.data
df_new
```

Out[17]:
```
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]])
```

In [20]:
```
df_labels=df.target
```

In [21]:
```
df_labels
```

Out[21]:
```
array([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, 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, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 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, 1, 0, 1, 1, 0, 1, 0, 1, 1, 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, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 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, 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])
```

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

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

In [23]:
```
labels = np.reshape(df_labels,(569,1))
```

In [25]:

```
final_df = np.concatenate([df_new,labels],axis=1)
```

In [26]:
```
final_df.shape
```

Out[26]:
```
(569, 31)
```

In [28]:
```
dataset = pd.DataFrame(final_df)
```

In [30]:
```
features = df.feature_names
```

In [31]:
```
features
```

Out[31]:
```
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')
```

In [32]:
```
features_labels = np.append(features,'label')
```

In [33]:
```
dataset.columns = features_labels
```

In [34]:
```
dataset.head()
```

Out[34]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	m fra dimen
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05

5 rows × 31 columns

In [35]:
```
dataset.tail()
```

Out[35]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	dim
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	C
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	C
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	C
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	C
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	C

5 rows × 31 columns

In [36]:
```
dataset['label'].replace(1, 'Benign',inplace=True)
dataset['label'].replace(0, 'Malignant',inplace=True)
```

In [37]:
```
dataset.tail()
```

Out[37]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	dim
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	C
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	C
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	C
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	C
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	C

5 rows × 31 columns

In [38]:
```
# na values
dataset.isna().sum()
```

Out[38]:
```
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
label                     0
dtype: int64
```

In [39]:

```
dataset.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  label                    569 non-null    object
```

```
dtypes: float64(30), object(1)
memory usage: 137.9+ KB
```

In [40]:
```
dataset.describe()
```

Out[40]:

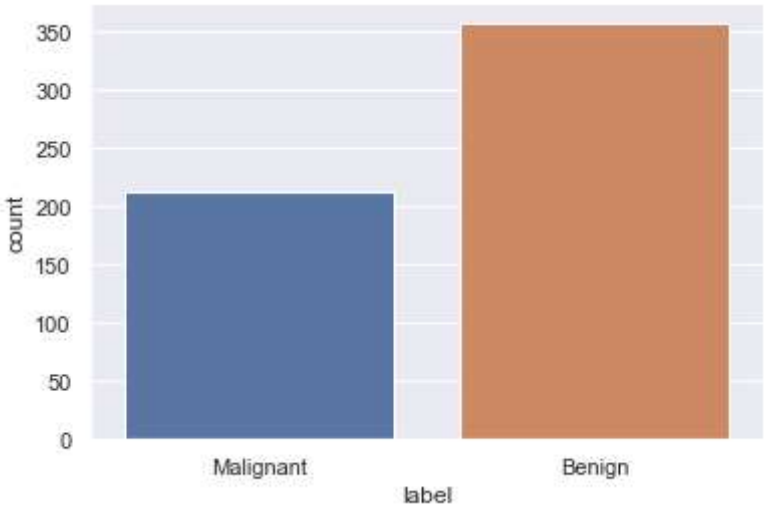
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	me conca poi
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000(
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048(
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038(
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000(
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020(
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033(
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074(
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201(

8 rows × 30 columns

In [41]:
```
import seaborn as sns
from matplotlib import pyplot as plt
sns.set()
```

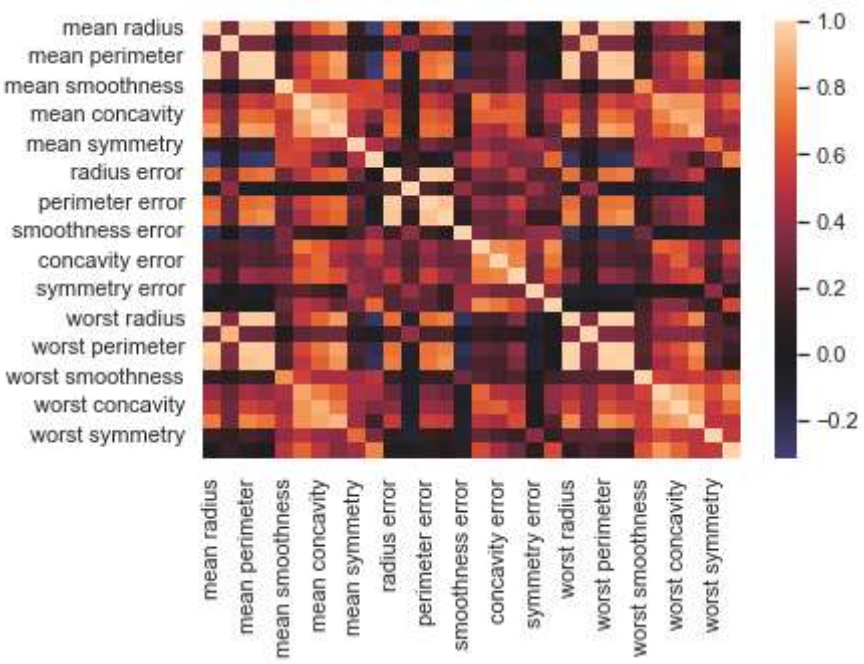
In [42]:
```
sns.countplot(dataset['label'])
plt.show()
```

```
C:\Users\jagan\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid position
al argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```



In [43]:
```
sns.heatmap(dataset.corr(),center = 0,linewidths=0)
```

Out[43]: <AxesSubplot:>



Feature Scaling

In [44]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
dataset['label'] = labelencoder.fit_transform(dataset['label'].values)
```

In [45]:
```
X=dataset.iloc[:,:30].values
y=dataset['label'].values
```

In [46]:
```
sc = StandardScaler()
X= sc.fit_transform(X)
```

In [47]:
```
np.mean(X),np.std(X)
```

Out[47]: (-6.826538293184326e-17, 1.0)

In [48]:
```
feat_cols = ['feature'+str(i) for i in range(X.shape[1])]
```

In [49]:
```
normalised_dataset= pd.DataFrame(X,columns=feat_cols)
normalised_dataset.tail()
```

Out[49]:

	feature0	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	fe
564	2.110995	0.721473	2.060786	2.343856	1.041842	0.219060	1.947285	2.320965	-0.312589	-0.

	feature0	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	fe
565	1.704854	2.085134	1.615931	1.723842	0.102458	-0.017833	0.693043	1.263669	-0.217664	-1.
566	0.702284	2.045574	0.672676	0.577953	-0.840484	-0.038680	0.046588	0.105777	-0.809117	-0.
567	1.838341	2.336457	1.982524	1.735218	1.525767	3.272144	3.296944	2.658866	2.137194	1.
568	-1.808401	1.221792	-1.814389	-1.347789	-3.112085	-1.150752	-1.114873	-1.261820	-0.820070	-0.

5 rows × 30 columns

In [50]:
```
X.shape
```

Out[50]:
```
(569, 30)
```

In [51]:
```
y.shape
```

Out[51]:
```
(569,)
```

PCA

IMPORTING SOME LIBRARY

In [52]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
```

In [53]:
```
for n in range(1,31):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

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

    pca = PCA()
    X_train = pca.fit_transform(X_train)
    X_test = pca.transform(X_test)

    explained_variance = pca.explained_variance_ratio_

    pca = PCA(n_components=n)
    X_train = pca.fit_transform(X_train)
    X_test = pca.transform(X_test)
    classifier = RandomForestClassifier(max_depth=2, random_state=0)
    classifier.fit(X_train, y_train)

    y_pred = classifier.predict(X_test)
```

```
    cm = confusion_matrix(y_test, y_pred)

    print('Accuracy for n_components',n,':',accuracy_score(y_test, y_pred))
```

```
Accuracy for n_components 1 : 0.9035087719298246
Accuracy for n_components 2 : 0.9035087719298246
Accuracy for n_components 3 : 0.9035087719298246
Accuracy for n_components 4 : 0.9035087719298246
Accuracy for n_components 5 : 0.9035087719298246
Accuracy for n_components 6 : 0.9035087719298246
Accuracy for n_components 7 : 0.9035087719298246
Accuracy for n_components 8 : 0.9473684210526315
Accuracy for n_components 9 : 0.9298245614035088
Accuracy for n_components 10 : 0.9122807017543859
Accuracy for n_components 11 : 0.9122807017543859
Accuracy for n_components 12 : 0.9210526315789473
Accuracy for n_components 13 : 0.9210526315789473
Accuracy for n_components 14 : 0.9298245614035088
Accuracy for n_components 15 : 0.9298245614035088
Accuracy for n_components 16 : 0.9035087719298246
Accuracy for n_components 17 : 0.9210526315789473
Accuracy for n_components 18 : 0.9122807017543859
Accuracy for n_components 19 : 0.9035087719298246
Accuracy for n_components 20 : 0.9210526315789473
Accuracy for n_components 21 : 0.8859649122807017
Accuracy for n_components 22 : 0.8947368421052632
Accuracy for n_components 23 : 0.8859649122807017
Accuracy for n_components 24 : 0.9210526315789473
Accuracy for n_components 25 : 0.9210526315789473
Accuracy for n_components 26 : 0.8771929824561403
Accuracy for n_components 27 : 0.8771929824561403
Accuracy for n_components 28 : 0.868421052631579
Accuracy for n_components 29 : 0.8771929824561403
Accuracy for n_components 30 : 0.8771929824561403
```

great accuracy with principal component value n_components = 8 (94.73)

LDA

In [54]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis



sc = StandardScaler()
X = sc.fit_transform(X)
le = LabelEncoder()
y = le.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)


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


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

```
print('Accuracy : ' + str(accuracy_score(y_test, y_pred)))
conf_m = confusion_matrix(y_test, y_pred)
print(conf_m)
```

```
Accuracy : 0.9649122807017544
[[71  0]
 [ 4 39]]
```

Conclusion

The accuracy for the LDA is 94.73 which is equal of PCA model.

PART C: "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 [55]:
```
#importing the dataset
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

digits = load_digits()
data = digits.data
data.shape
```

Out[55]:
```
(1797, 64)
```

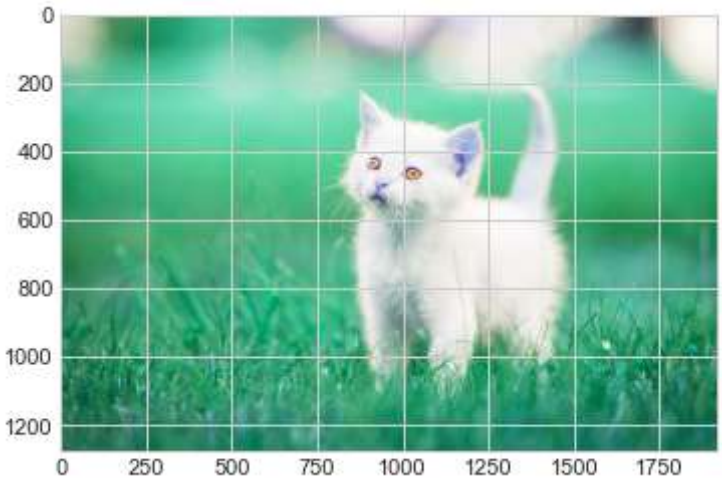
In [57]:
```
from sklearn.decomposition import PCA

pca = PCA(2) #we need 2 principal components.
converted_data = pca.fit_transform(digits.data)

converted_data.shape
```

Out[57]:
```
(1797, 2)
```

In [59]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

In [60]:
```
img=cv2.imread(r"C:\Users\jagan\OneDrive\Desktop\cat.jpg")
plt.imshow(img)
```

Out[60]:
```
<matplotlib.image.AxesImage at 0x1e4835a2070>
```



In [61]:
```
# Splitting the image in R,G,B arrays.

blue,green,red = cv2.split(img)
```

Compression Image Function

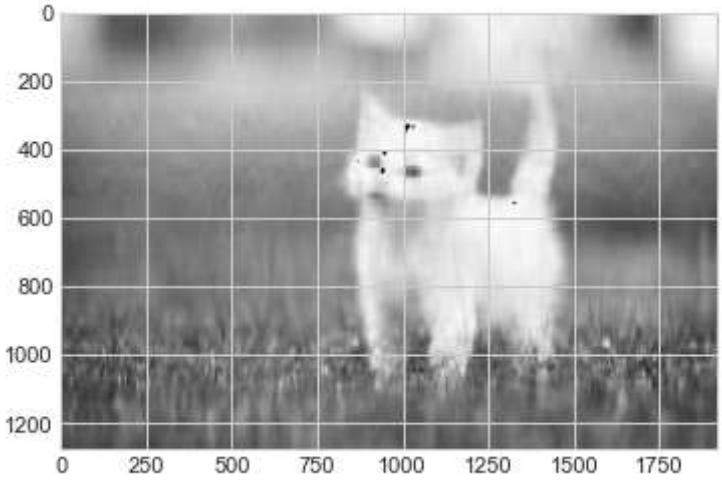
In [62]:
```
def compression_img(n):
    pca=PCA(n)
    #Applying to red channel and then applying inverse transform to transformed array.
    red_transformed = pca.fit_transform(red)
    red_inverted = pca.inverse_transform(red_transformed)

    #Applying to Green channel and then applying inverse transform to transformed array
    green_transformed = pca.fit_transform(green)
    green_inverted = pca.inverse_transform(green_transformed)

    #Applying to Blue channel and then applying inverse transform to transformed array.
    blue_transformed = pca.fit_transform(blue)
    blue_inverted = pca.inverse_transform(blue_transformed)

    img_compressed = (np.dstack((red_inverted, red_inverted, red_inverted))).astype(np.
    return img_compressed
```

Image Compression with 20 PCA components

In [63]:
```
img_compressed1=compression_img(20)
```

In [64]:
```
#viewing the compressed image
plt.imshow(img_compressed1)
```

Out[64]:
```
<matplotlib.image.AxesImage at 0x1e4832ca5e0>
```

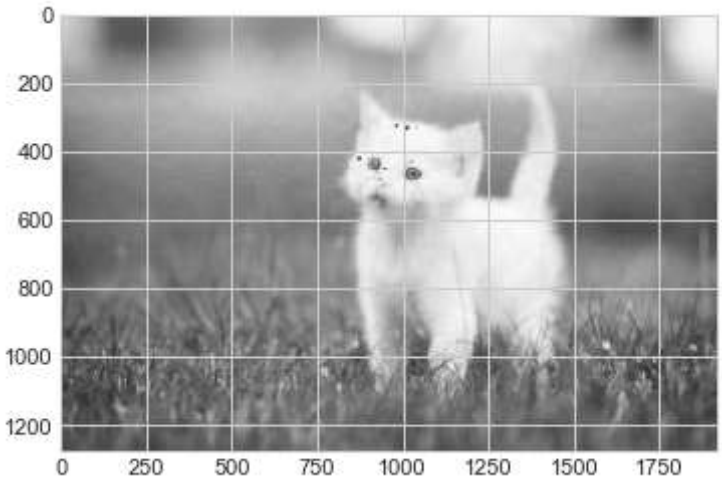


In [65]:
```
#### PCA with 50 components
```

In [66]:
```
img_compressed2=compression_img(50)
```

In [67]:
```
#viewing the compressed image
plt.imshow(img_compressed2)
```

Out[67]:
```
<matplotlib.image.AxesImage at 0x1e483409940>
```

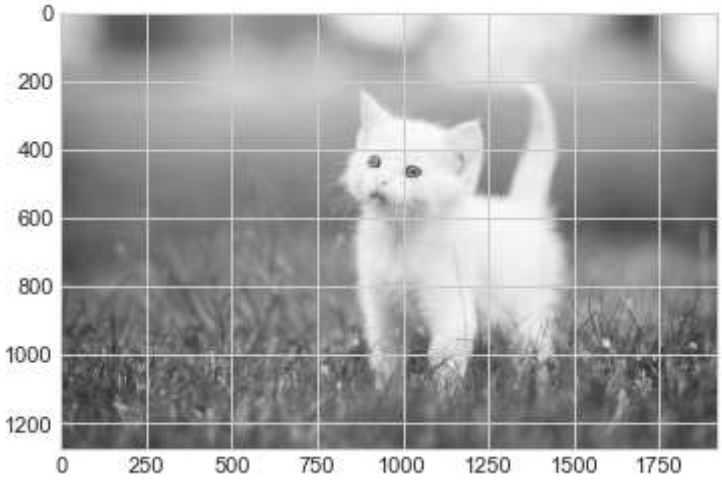


PCA with 100 components

In [68]:
```
img_compressed3=compression_img(100)
```

In [69]:
```
plt.imshow(img_compressed3)
```

Out[69]:
```
<matplotlib.image.AxesImage at 0x1e48338cf70>
```

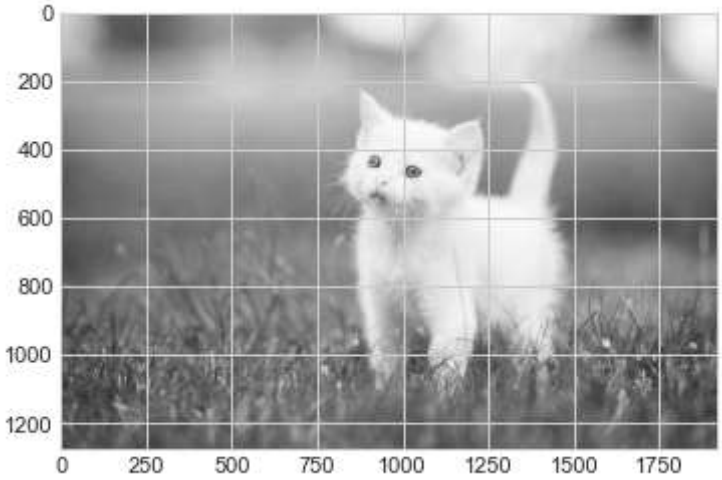


PCA with 200 components

In [70]:
```
img_compressed4=compression_img(200)
```

In [71]:
```
#viewing the compressed image
plt.imshow(img_compressed4)
```

Out[71]:
```
<matplotlib.image.AxesImage at 0x1e4832f14c0>
```



200 principal components we were able to create a sharp image just like the original one

conclusion

here i tried to explore more about PCA AND LDA .Principal component analysis is an unsupervised Dimensionality reduction technique, it ignores the class label. PCA focuses on capturing the direction of maximum variation in the data set.And LDA focuses on finding a feature subspace that maximizes the separability between the groups.

REFERENCE

Reference Links:

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://www.kaggle.com/jahirmorenoa/pca-to-the-breast-cancer-data-set

https://www.youtube.com/watch?v=e2sM7ccaA9c&ab_channel=DigitalSreeni

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://www.kaggle.com/mirzarahim/introduction-to-pca-image-compression-example

https://github.com/gtraskas/breast_cancer_prediction/blob/master/breast_cancer.ipynb

http://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

https://machinelearningmastery.com/linear-discriminant-analysis-with-python/

https://towardsdatascience.com/linear-discriminant-analysis-in-python-76b8b17817c2

https://www.mygreatlearning.com/blog/linear-discriminant-analysis-or-lda/

https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/

In [ ]: