PCA Project

April 24, 2019

1 Principal Component Analysis

In this project, I have implemented PCA to reduce a 30 dimensions dataset into 2 Principal components and then checked how logistic regression would work with 2 variables and compared it to a 30 features model.

Importing the required libraries

```
In [1]: import matplotlib.pyplot as plt
        import pandas as pd
        import numpy as np
        import seaborn as sns
        %matplotlib inline
```

I have used Cancer data set that is available online for doing this.

```
In [17]: df=pd.read_csv('data.csv')
```

```
In [18]: df.head()
```

```
Out[18]:          id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
         0    842302         M        17.99         10.38          122.80     1001.0
         1    842517         M        20.57         17.77          132.90     1326.0
         2  84300903         M        19.69         21.25          130.00     1203.0
         3  84348301         M        11.42         20.38           77.58      386.1
         4  84358402         M        20.29         14.34          135.10     1297.0

            smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
         0          0.11840           0.27760          0.3001              0.14710
         1          0.08474           0.07864          0.0869              0.07017
         2          0.10960           0.15990          0.1974              0.12790
         3          0.14250           0.28390          0.2414              0.10520
         4          0.10030           0.13280          0.1980              0.10430

               ...         texture_worst  perimeter_worst  area_worst  smoothness_worst  \
         0     ...                 17.33           184.60      2019.0            0.1622
         1     ...                 23.41           158.80      1956.0            0.1238
         2     ...                 25.53           152.50      1709.0            0.1444
```

```
3     ...               26.50            98.87       567.7              0.2098
4     ...               16.67           152.20      1575.0              0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

In [19]:
```
df.pop('id')
df.head()
```

Out[19]:
```
  diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0         M        17.99         10.38          122.80     1001.0
1         M        20.57         17.77          132.90     1326.0
2         M        19.69         21.25          130.00     1203.0
3         M        11.42         20.38           77.58      386.1
4         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   symmetry_mean     ...       texture_worst  perimeter_worst  area_worst  \
0         0.2419     ...               17.33           184.60      2019.0
1         0.1812     ...               23.41           158.80      1956.0
2         0.2069     ...               25.53           152.50      1709.0
3         0.2597     ...               26.50            98.87       567.7
4         0.1809     ...               16.67           152.20      1575.0

   smoothness_worst  compactness_worst  concavity_worst  concave points_worst  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
```

```
4           0.1374              0.2050            0.4000                  0.1625

   symmetry_worst  fractal_dimension_worst  Unnamed: 32
0          0.4601                  0.11890          NaN
1          0.2750                  0.08902          NaN
2          0.3613                  0.08758          NaN
3          0.6638                  0.17300          NaN
4          0.2364                  0.07678          NaN

[5 rows x 32 columns]
```

In [21]:
```
df['diagnosis'].replace('M',1,inplace=True)
df['diagnosis'].replace('B',0,inplace=True)
```

In [22]:
```
df.head()
```

Out[22]:
```
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1        17.99         10.38          122.80     1001.0
1          1        20.57         17.77          132.90     1326.0
2          1        19.69         21.25          130.00     1203.0
3          1        11.42         20.38           77.58      386.1
4          1        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   symmetry_mean     ...       texture_worst  perimeter_worst  area_worst  \
0         0.2419     ...               17.33           184.60      2019.0
1         0.1812     ...               23.41           158.80      1956.0
2         0.2069     ...               25.53           152.50      1709.0
3         0.2597     ...               26.50            98.87       567.7
4         0.1809     ...               16.67           152.20      1575.0

   smoothness_worst  compactness_worst  concavity_worst  concave points_worst  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   symmetry_worst  fractal_dimension_worst  Unnamed: 32
0          0.4601                  0.11890          NaN
1          0.2750                  0.08902          NaN
2          0.3613                  0.08758          NaN
```

```
3          0.6638                   0.17300          NaN
4          0.2364                   0.07678          NaN

[5 rows x 32 columns]
```

```
In [23]: df_x = df.drop('diagnosis',axis=1)
```

```
In [33]: df_x.head()
```

```
Out[33]:    radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  \
         0        17.99         10.38          122.80     1001.0          0.11840
         1        20.57         17.77          132.90     1326.0          0.08474
         2        19.69         21.25          130.00     1203.0          0.10960
         3        11.42         20.38           77.58      386.1          0.14250
         4        20.29         14.34          135.10     1297.0          0.10030

            compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
         0           0.27760          0.3001              0.14710         0.2419
         1           0.07864          0.0869              0.07017         0.1812
         2           0.15990          0.1974              0.12790         0.2069
         3           0.28390          0.2414              0.10520         0.2597
         4           0.13280          0.1980              0.10430         0.1809

            fractal_dimension_mean     ...       texture_worst  perimeter_worst  \
         0                 0.07871     ...               17.33           184.60
         1                 0.05667     ...               23.41           158.80
         2                 0.05999     ...               25.53           152.50
         3                 0.09744     ...               26.50            98.87
         4                 0.05883     ...               16.67           152.20

            area_worst  smoothness_worst  compactness_worst  concavity_worst  \
         0      2019.0            0.1622             0.6656           0.7119
         1      1956.0            0.1238             0.1866           0.2416
         2      1709.0            0.1444             0.4245           0.4504
         3       567.7            0.2098             0.8663           0.6869
         4      1575.0            0.1374             0.2050           0.4000

            concave points_worst  symmetry_worst  fractal_dimension_worst  Unnamed: 32
         0                0.2654          0.4601                  0.11890          NaN
         1                0.1860          0.2750                  0.08902          NaN
         2                0.2430          0.3613                  0.08758          NaN
         3                0.2575          0.6638                  0.17300          NaN
         4                0.1625          0.2364                  0.07678          NaN

         [5 rows x 31 columns]
```

```
In [25]: df_y = df['diagnosis']
```

Below steps are performed to standardize the independent variables before using PCA.

In [26]:
```python
from sklearn.preprocessing import StandardScaler
```

In [27]:
```python
scaler = StandardScaler()
scaler.fit(df_x)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\extmath.py:776: RuntimeWarning: invali
  updated_mean = (last_sum + new_sum) / updated_sample_count
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\extmath.py:781: RuntimeWarning: Degree
  new_unnormalized_variance = np.nanvar(X, axis=0) * new_sample_count
```

Out[27]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [28]:
```python
scaled_data = scaler.transform(df_x)
```

In [31]:
```python
df_xnew = pd.DataFrame(data = scaled_data[0:,0:],columns=df_x.columns)
```

In [34]:
```python
df_xnew.pop('Unnamed: 32')
df_xnew.head()
```

Out[34]:
```
   radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  \
0     1.097064     -2.073335        1.269934   0.984375         1.568466
1     1.829821     -0.353632        1.685955   1.908708        -0.826962
2     1.579888      0.456187        1.566503   1.558884         0.942210
3    -0.768909      0.253732       -0.592687  -0.764464         3.283553
4     1.750297     -1.151816        1.776573   1.826229         0.280372

   compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
0          3.283515        2.652874             2.532475       2.217515
1         -0.487072       -0.023846             0.548144       0.001392
2          1.052926        1.363478             2.037231       0.939685
3          3.402909        1.915897             1.451707       2.867383
4          0.539340        1.371011             1.428493      -0.009560

   fractal_dimension_mean          ...             radius_worst  \
0                2.255747          ...                 1.886690
1               -0.868652          ...                 1.805927
2               -0.398008          ...                 1.511870
3                4.910919          ...                -0.281464
4               -0.562450          ...                 1.298575

   texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0      -1.359293         2.303601    2.001237          1.307686
1      -0.369203         1.535126    1.890489         -0.375612
2      -0.023974         1.347475    1.456285          0.527407
3       0.133984        -0.249939   -0.550021          3.394275
4      -1.466770         1.338539    1.220724          0.220556

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
```

```
0          2.616665         2.109526               2.296076         2.750622
1         -0.430444        -0.146749               1.087084        -0.243890
2          1.082932         0.854974               1.955000         1.152255
3          3.893397         1.989588               2.175786         6.046041
4         -0.313395         0.613179               0.729259        -0.868353

   fractal_dimension_worst
0                 1.937015
1                 0.281190
2                 0.201391
3                 4.935010
4                -0.397100

[5 rows x 30 columns]
```

I have used PCA from scikit learn and specified no of components to be 2 and created 2 Principal components.

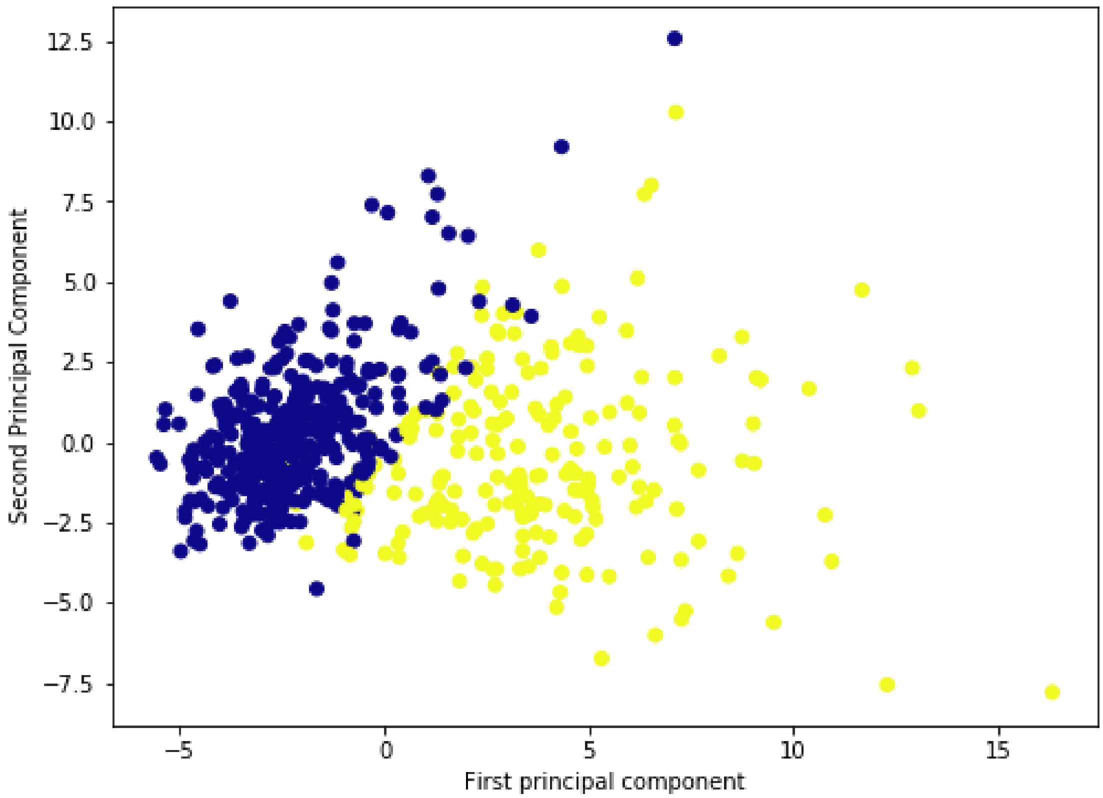
```
In [35]: from sklearn.decomposition import PCA

In [36]: pca = PCA(n_components=2)

In [37]: pca.fit(df_xnew)

Out[37]: PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
           svd_solver='auto', tol=0.0, whiten=False)
```

Now we can transform this data to its first 2 principal components.

```
In [38]: x_pca = pca.transform(df_xnew)

In [39]: df_xnew.shape

Out[39]: (569, 30)

In [40]: x_pca.shape

Out[40]: (569, 2)
```

Plotting the 2 Principal components

```
In [42]: plt.figure(figsize=(8,6))
         plt.scatter(x_pca[:,0],x_pca[:,1],c=df_y,cmap='plasma')
         plt.xlabel('First principal component')
         plt.ylabel('Second Principal Component')

Out[42]: Text(0, 0.5, 'Second Principal Component')
```



Clearly by using these two components we can easily separate these two classes.

The components correspond to combinations of the original features, the components themselves are stored as an attribute of the fitted PCA object:

```
In [43]: pca.components_
```

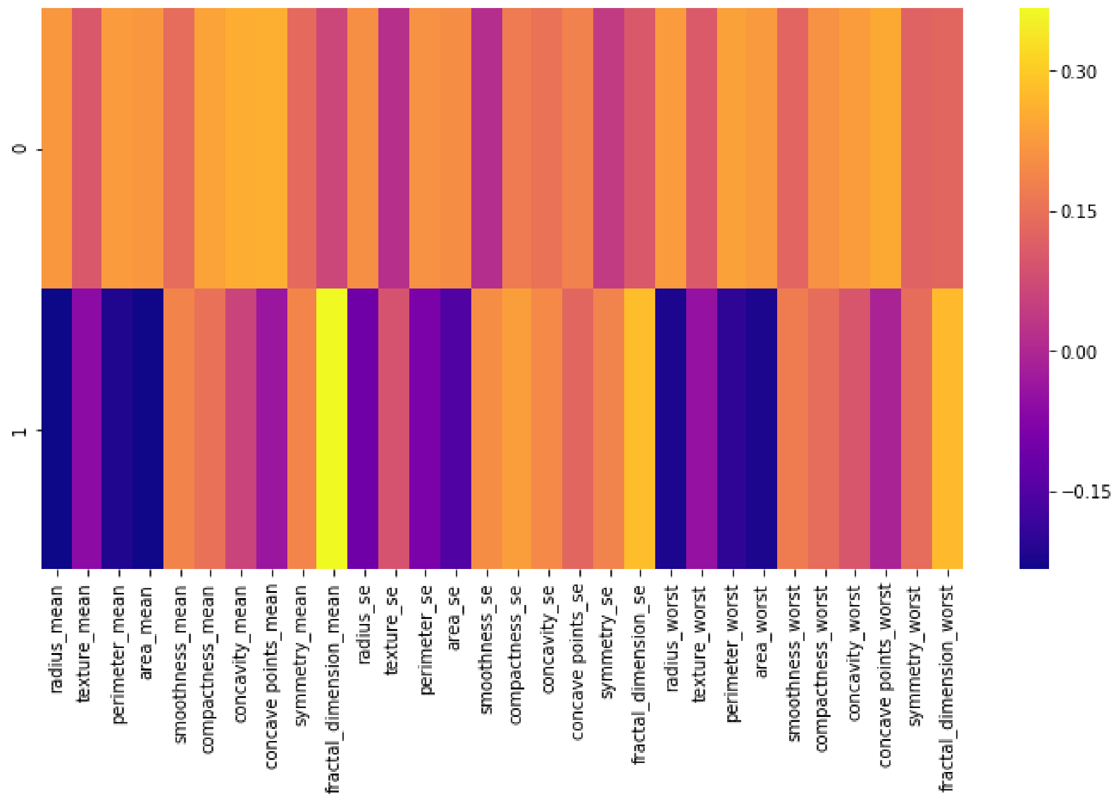
```
Out[43]: array([[ 0.21890244,  0.10372458,  0.22753729,  0.22099499,  0.14258969,
                  0.23928535,  0.25840048,  0.26085376,  0.13816696,  0.06436335,
                  0.20597878,  0.01742803,  0.21132592,  0.20286964,  0.01453145,
                  0.17039345,  0.15358979,  0.1834174 ,  0.04249842,  0.10256832,
                  0.22799663,  0.10446933,  0.23663968,  0.22487053,  0.12795256,
                  0.21009588,  0.22876753,  0.25088597,  0.12290456,  0.13178394],
                [-0.23385713, -0.05970609, -0.21518136, -0.23107671,  0.18611302,
                  0.15189161,  0.06016536, -0.0347675 ,  0.19034877,  0.36657547,
                 -0.10555215,  0.08997968, -0.08945723, -0.15229263,  0.20443045,
                  0.2327159 ,  0.19720728,  0.13032156,  0.183848  ,  0.28009203,
                 -0.21986638, -0.0454673 , -0.19987843, -0.21935186,  0.17230435,
                  0.14359317,  0.09796411, -0.00825724,  0.14188335,  0.27533947]])
```

In this numpy matrix array, each row represents a principal component, and each column relates back to the original features. we can visualize this relationship with a heatmap:

```
In [44]: df_comp = pd.DataFrame(pca.components_,columns=df_xnew.columns)
```

```
In [45]: plt.figure(figsize=(12,6))
         sns.heatmap(df_comp,cmap='plasma',)
```

```
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x2f01c2e3630>
```



This heatmap shows correlation between each component and actual variables

```
In [48]: df_pca = pd.DataFrame(data=x_pca[0:,0:],columns=['Component_1','Component_2'])
```

```
In [49]: df_pca.head()
```

Out[49]:

	Component_1	Component_2
0	9.192837	1.948583
1	2.387802	-3.768172
2	5.733896	-1.075174
3	7.122953	10.275589
4	3.935302	-1.948072

Doing Logistic Regression using Actual Variables

```
In [50]: from sklearn.model_selection import train_test_split
```

```
In [51]: x1_train,x1_test,y1_train,y1_test = train_test_split(df_xnew,df_y,test_size=0.3,randon
```

In [52]: from sklearn.linear_model import LogisticRegression

In [53]: logmodel=LogisticRegression()

In [54]: logmodel.fit(x1_train,y1_train)

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning
  FutureWarning)
```

Out[54]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [55]: pred_y1 = logmodel.predict(x1_test)

In [56]: from sklearn.metrics import classification_report,confusion_matrix

In [58]:
```
print('Before PCA:')
print('\n')
print(confusion_matrix(y1_test,pred_y1))
print('\n')
print(classification_report(y1_test,pred_y1))
```

```
Before PCA:


[[104   0]
 [  3  64]]


              precision    recall  f1-score   support

           0       0.97      1.00      0.99       104
           1       1.00      0.96      0.98        67

   micro avg       0.98      0.98      0.98       171
   macro avg       0.99      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

In [60]: x1_train.shape

Out[60]: (398, 30)

Doing Logistic Regression Using PCA components

In [61]: x2_train,x2_test,y2_train,y2_test = train_test_split(df_pca,df_y,test_size=0.3,random_

In [62]:
```
from sklearn.linear_model import LogisticRegression
```

In [63]:
```
logmodel2=LogisticRegression()
```

In [64]:
```
logmodel2.fit(x2_train,y2_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning
  FutureWarning)
```

Out[64]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='warn',
                   n_jobs=None, penalty='l2', random_state=None, solver='warn',
                   tol=0.0001, verbose=0, warm_start=False)
```

In [65]:
```
pred_y2 = logmodel2.predict(x2_test)
```

In [66]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [69]:
```
print('After PCA:')
print('\n')
print(confusion_matrix(y2_test,pred_y2))
print('\n')
print(classification_report(y2_test,pred_y2))
```

```
After PCA:


[[103   1]
 [  9  58]]


              precision    recall  f1-score   support

           0       0.92      0.99      0.95       104
           1       0.98      0.87      0.92        67

   micro avg       0.94      0.94      0.94       171
   macro avg       0.95      0.93      0.94       171
weighted avg       0.94      0.94      0.94       171
```

From the results of logistic regression. we can see that we had an accuracy of 0.98 with 30 dimensions, and when reduced to 2 dimensions using PCA, we still had 0.94 dimensions. This shows the power of PCA in rapidly reducing the dimensions.

In [ ]: