Lab 7 - PCA & LDA

SUBMITTED BY:

Name : Kavitha.S
Reg no : 21122033
Class : 2MSDS

Lab Overview

1. Perform PCA and LDA on Breast Cancer Dataset, write down your obsevations. While loading, use the toy dataset available in SKLearn (load_breast_cancer)
2. Illustrate the effect of changing various method parameters of PCA and LDA. Compare the accuracies, and provide visualizations and interpretations for the evaluation metrices.
3. "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.

Sections

Load Data

Import Libraries

In [35]:
```
from sklearn.datasets import load_breast_cancer
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

In [4]:
```
data = load_breast_cancer(as_frame=True)
cancer_data = data['data']
cancer_data.head()
```

Out[4]:

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

5 rows × 30 columns

In [5]:
```
1 cancer_data.describe()
```

Out[5]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	56
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	

8 rows × 30 columns

In [7]:
```
1 cancer_data.shape
```

Out[7]:
```
(569, 30)
```

In [6]:
```
1 cancer_data.columns
```

Out[6]:
```
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'],
      dtype='object')
```

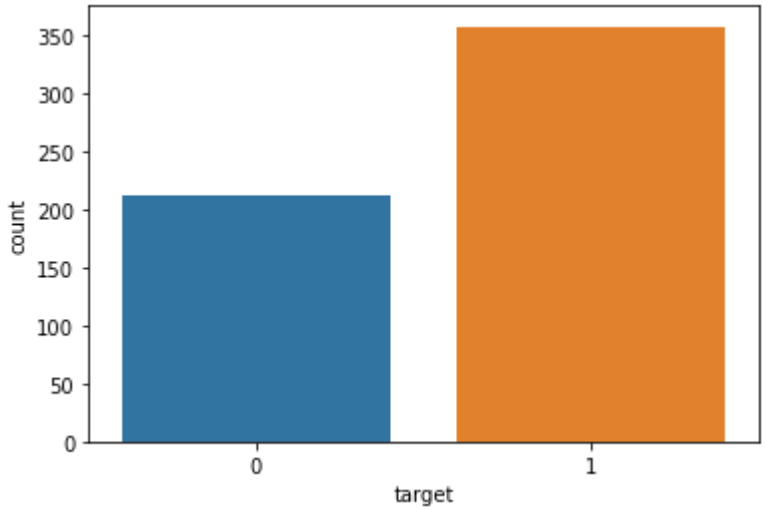
Exploratory Data Analysis

The plot above gives us an idea of the relationships between the variables, there seems to be a positive relationship and sepal length looks to be relatively normal, while petal width is positively skewed

In [9]:
```
1 sns.countplot(data['target'])
```

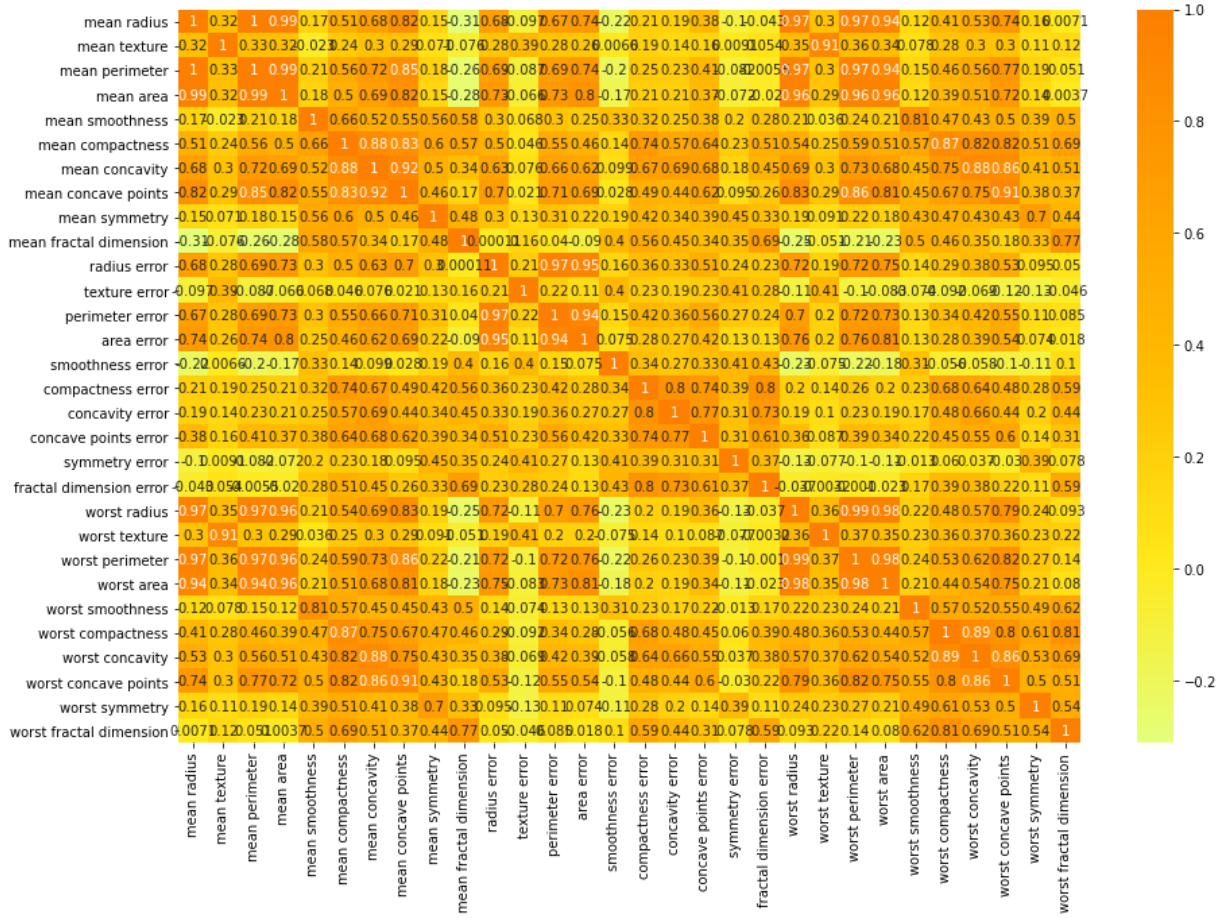
```
C:\Users\SRIDHAR\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variable as a keyword arg: x. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments with
out an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[9]:
```
<AxesSubplot:xlabel='target', ylabel='count'>
```



There are more positive cases in our dependent variable in our dataset which must be kept in mind during interpretation

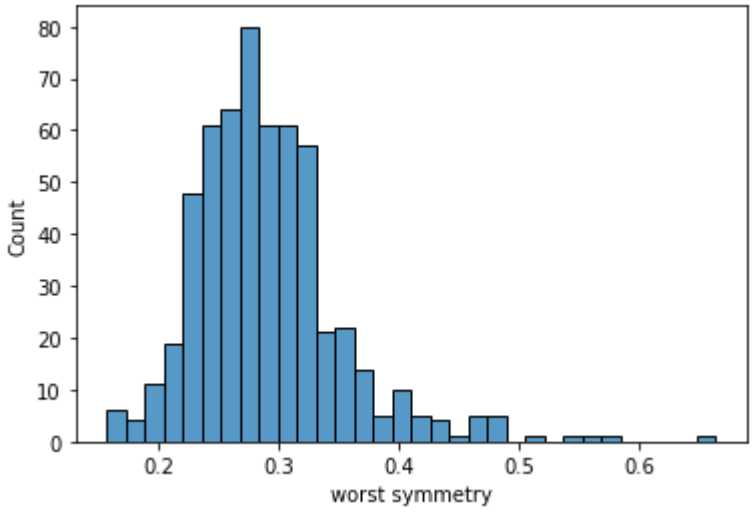
In [20]:
```
fig, ax = plt.subplots(figsize=(15,10))
corr = cancer_data.corr()
ax = sns.heatmap(corr, cmap = 'Wistia', annot= True);
```



Correlation matrix

In [157]:
```
1 sns.histplot(cancer_data['worst symmetry'])
```

Out[157]:
```
<AxesSubplot:xlabel='worst symmetry', ylabel='Count'>
```



Worst symmetry looks relatively normal here

Train Test Split

In [24]:
```
1 X_train, X_test, y_train, y_test = train_test_split(cancer_data, data['targe
```

to evaluate performance later on

Normalizing Data

In [25]:
```
1 sc = StandardScaler()
2 X_train = sc.fit_transform(X_train)
3 X_test = sc.fit_transform(X_test)
```

Normality is especially important with LDA in terms of accuracy and therefore, is just a good practice to do now

Running PCA

In [27]:
```
pca = PCA(n_components=1)
x_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

print(pca.singular_values_)
```

```
[77.05926087]
```

In [28]:
```
print(pca.explained_variance_ratio_)
```

```
[0.43502782]
```

This tells us how much variance of the training data does each component of PCA explain

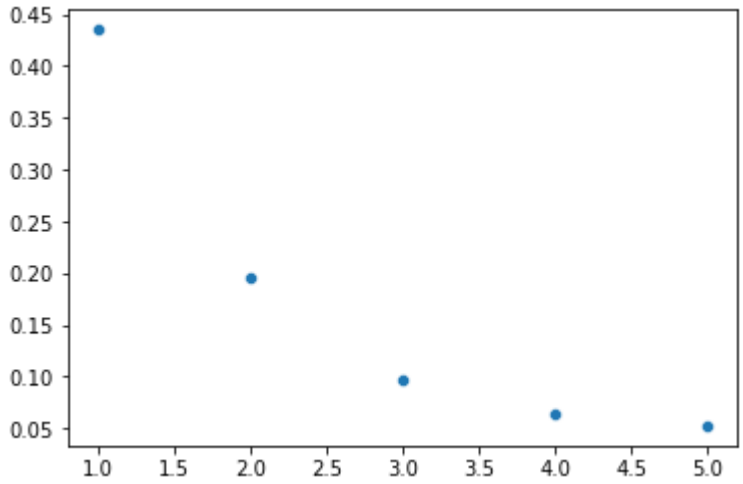
In [29]:
```
pca_5 = PCA(n_components=5)
pca_5.fit(X_train)

print(pca_5.singular_values_)
```

```
[77.05926087 51.59215967 36.54007835 29.75558584 26.77846228]
```

In [30]:
```
sns.scatterplot(range(1,6),pca_5.explained_variance_ratio_)
```

```
C:\Users\SRIDHAR\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[30]:
```
<AxesSubplot:>
```



This graph shows the drop in variance explained after every additional PCA component when we choose number of components to be 5

Running LDA

In [32]:
```
clf = LinearDiscriminantAnalysis(n_components=1)

X_lda = clf.fit_transform(X_train, y_train)
X_test_lda = clf.transform(X_test)
```

We see that LDA requires labeled data in order to fit any dimensionality reduction unlike PCA which can work on labeled or unlabeled data.

In [33]:
```
print(clf.explained_variance_ratio_)
```

```
[1.]
```

This states that all the variance can be explained by 1 LDA component which is obviously higher than PCA and therefore it is the better choice here

In [34]:
```
X_lda[0:5]
```

Out[34]:
```
array([[-1.22465232],
       [ 4.62620127],
       [-2.36259744],
       [-0.88250785],
       [-2.40526443]])
```

Evaluating performance

Let's use random forests to evaluate the datasets post-dimensionality reduction

In [63]:
```
classifier = RandomForestClassifier(max_depth=3, random_state=0)

classifier.fit(x_pca, y_train)
y_pred = classifier.predict(X_test_pca)
```

In [67]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy: ' + str(accuracy_score(y_test, y_pred)))
```

```
[[40  3]
 [ 4 67]]
Accuracy: 0.9385964912280702
```

We see with PCA we get an accuracy of around 93.8%

In [66]:
```
classifier = RandomForestClassifier(max_depth=2, random_state=0)

classifier.fit(X_lda, y_train)
y_pred = classifier.predict(X_test_lda)
```

In [64]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy: ' + str(accuracy_score(y_test, y_pred)))
```

```
[[38  5]
 [ 0 71]]
Accuracy: 0.956140350877193
```

With LDA, we get an accuracy of 95.6% which is much better and therefore the better tool to use in this scenario

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

I agree with this statement as all PCA is doing is analyzing the data provided and focusing on what it believes are the most important featuers required to recreate the image. It does this via variance analysis. Therefore, PCA could work similar to a convolution operation where we apply filters to identify important features required to identify an image. The singular values here are akin to the most important features and therefore, would help the mdoel focus on the important feeatures rather than noise which would improve performance. The same logic would be applied to compression, if we minimize the loss and reduce the size of an image by focusing on important features, PCA would work for compression.

Conclusion/Results

In conclusion, we see that LDA performed better on this dataset but has more constraints. With LDA, you are required to have some labeled target variable and the number of components you can create is 1 minus the total number of classes. While with PCA, you can use an unlabeled or labeled dataset and have as many components as total number of variables, each decreasing in explained variance per component as a component is added.

References

1. https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/ (https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/)
2. http://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.ht (http://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.ht
3. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)
4. https://stackabuse.com/implementing-lda-in-python-with-scikit-learn/ (https://stackabuse.com/implementing-lda-in-python-with-scikit-learn/)