Understanding PCA (Principal Component Analysis) with Python

Saptashwa Bhattacharyya Sep 15, 2018 · 7 min read ★

Getting stuck in the sea of variables to analyze your data ? Feeling lost in deciding which features to choose so that your model is safe from overfitting? Is there any way to reduce the dimension of the feature space?

Well, PCA can surely help you.

In this meditation we will go through a simple explanation of principal component analysis on cancer data-set and see examples of feature space dimension reduction to data visualization.

Without any further delay let's begin by importing the cancer data-set.

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
```

Sometimes it's better to know about the data that you're using and we can use `DESCR` to know the basic description of the data-set.

```
print cancer.DESCR
```

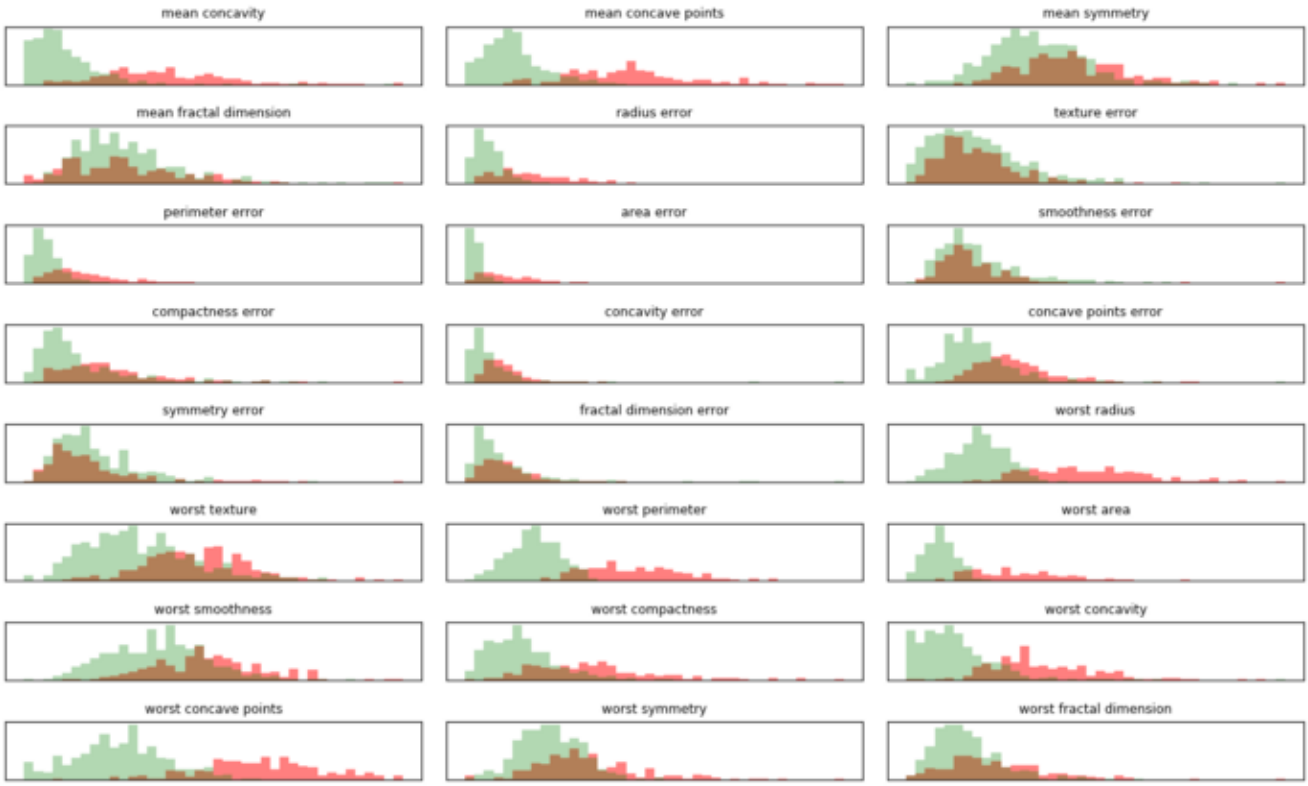
benign. The target variables are listed as 0 and 1 and just to make sure 0 represents malignant and vice-versa one can check -

```
print len(cancer.data[cancer.target==1])

>> 357
```

To know more about how the features affect the target, we can plot histograms of malignant and benign classes. If the two histograms are separated based on the feature, then we can say that the feature is important to discern the instances.

```
import numpy as np
import matplotlib.pyplot as plt
# from matplotlib.pyplot import matplotlib

fig,axes =plt.subplots(10,3, figsize=(12, 9)) # 3 columns each
containing 10 figures, total 30 features

malignant=cancer.data[cancer.target==0] # define malignant
benign=cancer.data[cancer.target==1] # define benign

ax=axes.ravel()# flat axes with numpy ravel

for i in range(30):
  _,bins=np.histogram(cancer.data[:,i],bins=40)
  ax[i].hist(malignant[:,i],bins=bins,color='r',alpha=.5)# red color
for malignant class
  ax[i].hist(benign[:,i],bins=bins,color='g',alpha=0.3)# alpha is
for transparency in the overlapped region
  ax[i].set_title(cancer.feature_names[i],fontsize=9)
  ax[i].axes.get_xaxis().set_visible(False) # the x-axis co-
ordinates are not so useful, as we just want to look how well
separated the histograms are
  ax[i].set_yticks(())

ax[0].legend(['malignant','benign'],loc='best',fontsize=8)
plt.tight_layout()# let's make good plots
plt.show()
```

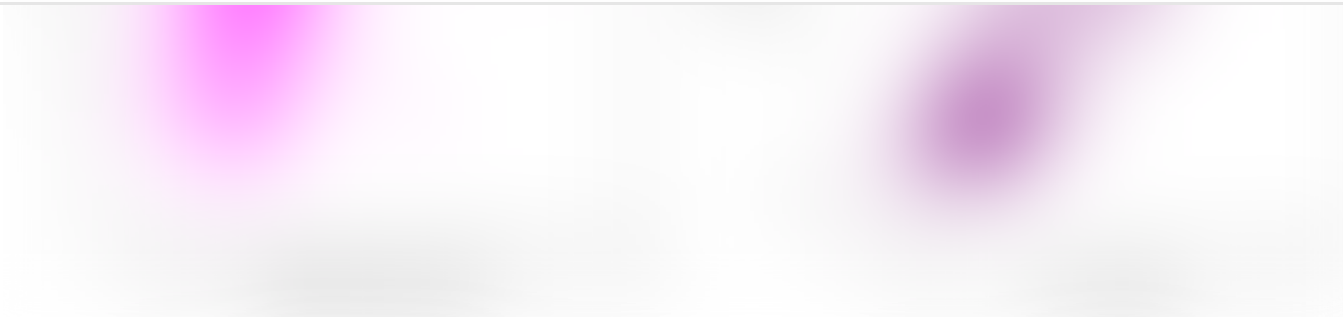
It looks like the figure below



Histogram of malignant and benign classes based on the 30 features of cancer data-set

Now from these histograms we see that features like- mean fractal dimension has very little role to play in discerning malignant from benign, but worst concave points or worst perimeter are useful features that can give us strong hint about the classes of cancer data-set. Histogram plots are essential in our astrophysics studies as they are often used to separate models. I couldn't resist the temptation to bring it up here. So if your data has only one feature e.g. worst perimeter, it can be good enough to separate malignant from benign case.

Before using PCA to these cancer data-set, let's understand very simply what PCA actually does. We know that in a data-set there are high possibilities for some features to be correlated. Let's see some example plots from cancer data set —



Scatter plots with few features of cancer data set

Now hopefully you can already understand which plot shows strong correlation between the features. Below is the code that I've used to plot these graphs.

```
imoprt pandas as pd
cancer_df=pd.DataFrame(cancer.data,columns=cancer.feature_names)#
just convert the scikit learn data-set to pandas data-frame.
plt.subplot(1,2,1)#fisrt plot
plt.scatter(cancer_df['worst symmetry'], cancer_df['worst texture'],
s=cancer_df['worst area']*0.05, color='magenta', label='check',
alpha=0.3)
plt.xlabel('Worst Symmetry',fontsize=12)
plt.ylabel('Worst Texture',fontsize=12)
plt.subplot(1,2,2)# 2nd plot
plt.scatter(cancer_df['mean radius'], cancer_df['mean concave
points'], s=cancer_df['mean area']*0.05, color='purple',
label='check', alpha=0.3)
plt.xlabel('Mean Radius',fontsize=12)
plt.ylabel('Mean Concave Points',fontsize=12)
plt.tight_layout()
plt.show()
```

PCA is essentially a method that reduces the dimension of the feature space in such a way that new variables are orthogonal to each other (i.e. they are independent or not correlated). I have put some references at the end of this post so that interested people can really delve into the mathematics of PCA.

Anyway, from the cancer data-set we see that it has 30 features, so let's reduce it to only 3 principal features and then we can visualize the scatter plot of these new independent variables.

Before applying PCA, we scale our data such that each feature has unit variance. This is necessary because fitting algorithms highly depend on the scaling of the features. Here

We first instantiate the module and then fit to the data.

```
scaler=StandardScaler()#instantiate
scaler.fit(cancer.data) # compute the mean and standard which will
be used in the next command
X_scaled=scaler.transform(cancer.data)# fit and transform can be
applied together and I leave that for simple exercise
# we can check the minimum and maximum of the scaled features which
we expect to be 0 and 1
print "after scaling minimum", X_scaled.min(axis=0)
```

Now we're ready to apply PCA on this scaled data-set. We start as before with `StandardScaler` , where we instantiate, then fit and finally transform the scaled data. While applying PCA you can mention how many principal components you want to keep.

```
pca=PCA(n_components=3)

pca.fit(X_scaled)

X_pca=pca.transform(X_scaled)

#let's check the shape of X_pca array

print "shape of X_pca", X_pca.shape
```

Now we have seen that the data have only 3 features. *Drawback of PCA is it's almost impossible to tell how the initial features (here 30 features) combined to form the principal components.* Now one important point to note is that I have chosen 3 components instead of 2, which could have reduced the dimension of the data-set even more. Can you choose `n_components=2` ? [Think about it for sometime as a mini-exercise. Can you think of some method to test this ?]

You can check by measuring the variance ratio of the principal components.

```
ex_variance=np.var(X_pca,axis=0)
ex_variance_ratio = ex_variance/np.sum(ex_variance)
```

So here you can see that the first 2 components contributes to 87% of the total variance. So it's good enough to choose only 2 components. Okay, now with these first 2 components, we can jump to one of the most important application of PCA, which is data visualization. **Now, since the PCA components are orthogonal to each other and they are not correlated, we can expect to see malignant and benign classes as distinct.** Let's plot the malignant and benign classes based on the first two principal components

```
Xax=X_pca[:,0]
Yax=X_pca[:,1]

labels=cancer.target

cdict={0:'red',1:'green'}

labl={0:'Malignant',1:'Benign'}

marker={0:'*',1:'o'}

alpha={0:.3, 1:.5}

fig,ax=plt.subplots(figsize=(7,5))
fig.patch.set_facecolor('white')

for l in np.unique(labels):
 ix=np.where(labels==l)
 ax.scatter(Xax[ix],Yax[ix],c=cdict[l],s=40,
            label=labl[l],marker=marker[l],alpha=alpha[l])

# for loop ends

plt.xlabel("First Principal Component",fontsize=14)
plt.ylabel("Second Principal Component",fontsize=14)
plt.legend()
plt.show()

# please check the scatter plot of the remaining component and you
will understand the difference
```

With the above code, the plot looks like as shown below

Plot of breast cancer classes based on the first 2 principal components of the cancer features.

Looks great, isn't it? The two classes are well separated with the first 2 principal components as new features. As good as it seems like even a linear classifier could do very well to identify a class from the test set. On a separate post, I have discussed how to apply a pipeline consisting of PCA and Support Vector Classifier to and draw the decision function for this same data-set. One important feature is how the malignant class is spread out compared to benign and take a look back to those histogram plots. Can you find some similarity?

These principal components are calculated only from features and no information from classes are considered. So PCA is unsupervised method and it's difficult to interpret the two axes as they are some complex mixture of the original features. We can make a heat-plot to see how the features mixed up to create the components.

```
plt.matshow(pca.components_,cmap='viridis')
plt.yticks([0,1,2],['1st Comp','2nd Comp','3rd Comp'],fontsize=10)
plt.colorbar()
plt.xticks(range(len(cancer.feature_names)),cancer.feature_names,rot
```

3 PCs and dependencies on original features

It's actually difficult to understand how correlated the original features are from this plot but we can always map the correlation of the features using `seaborn` heat-plot. But still, check the correlation plots before and see how 1st principal component is affected by mean concave points and worst texture. Can you tell which feature contribute more towards the 1st PC ?

Here I show the correlation plot of 'worst' values of the features.

```
feature_worst=list(cancer_df.columns[20:31]) # select the 'worst'
features

import seaborn as sns

s=sns.heatmap(cancer_df[feature_worst].corr(),cmap='coolwarm')
s.set_yticklabels(s.get_yticklabels(),rotation=30,fontsize=7)
s.set_xticklabels(s.get_xticklabels(),rotation=30,fontsize=7)
plt.show()
```

Correlation plot of the 'worst' features of cancer data-set

So to end this meditation let's summarize what we have done and learnt

1. Why PCA rather than just feature analysis ? (Answer hints: large data-set, many features, let's reduce dimension of feature space)
2. We started our example with cancer data-set and found 30 features with 2 classes.
3. To apply PCA on this data-set, first we scale all the features and then apply `fit_transform` method of PCA (with 3 principal components) on the scaled features.
4. We show that out of those 3 principal components, 2 contribute to 87% of the total variance.
5. Based on these 2 principal components we visualize the data and see very clear separation between 'Malignant' and 'Benign' classes.

Hope this will help you to grasp few concepts and guide you to effectively apply PCA on your data-set. For exercise you can try immediately on Boston house data (13 features) and see the results. Go through carefully again and remember the essential concepts about initially correlated features to finally independent principal components.

This post which is influenced from Muller's book, was my stepping stone to an attempt to separate two astrophysical scenarios (Pulsar and Dark Matter) with cosmic ray measured by CALET detector. You can find all the details in my github profile. If you are interested in the parameters and detail description please let me know. A separate

Keep an eye on the coming posts and stay strong, be happy!

Check the references below for further reading

1. In-depth analysis of how PCA works: a-one-stop-shop-for-pca
2. Machine Learning in Action; Peter Harrington: Manning Publications, pp-270–278.
3. PCA and Image Reconstruction