Recovering features names of explained_variance_ratio_ in PCA with sklearn

I'm trying to recover from a PCA done with scikit-learn, **which** features are selected as *relevant*.

A classic example with IRIS dataset.

```
import pandas as pd
import pylab as pl
from sklearn import datasets
from sklearn.decomposition import PCA

# load dataset
iris = datasets.load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# normalize data
df_norm = (df - df.mean()) / df.std()

# PCA
pca = PCA(n_components=2)
pca.fit_transform(df_norm.values)
print pca.explained_variance_ratio_
```

This returns

```
In [42]: pca.explained_variance_ratio_
```

this features in iris.feature_names ?

```
In [47]: print iris.feature_names
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

Thanks in advance for your help.

python machine-learning scikit-learn pca

edited Sep 20 '16 at 13:09

asked Apr 10 '14 at 9:43

mazieres

607 1 9 19

1 `pca.components_` is what you are looking for. – Sangram Jun 2 '16 at 11:24

4 Answers

Edit: as others have commented, you may get same values from `.components_` attribute.

Each principal component is a linear combination of the original variables:

$$\mathrm{PC}^j = \beta_1^j X_1 + \beta_2^j X_2 + \cdots + \beta_n^j X_n$$

where `X_i` s are the original variables, and `Beta_i` s are the corresponding weights or so called coefficients.

To obtain the weights, you may simply pass identity matrix to the `transform` method:

```
>>> i = np.identity(df.shape[1])  # identity matrix
>>> i
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])

>>> coef = pca.transform(i)
```

```
       [-0.2634, -0.9256],
       [ 0.5813, -0.0211],
       [ 0.5656, -0.0654]])
```

Each column of the `coef` matrix above shows the weights in the linear combination which obtains corresponding principal component:

```
>>> pd.DataFrame(coef, columns=['PC-1', 'PC-2'], index=df.columns)
                    PC-1   PC-2
sepal length (cm)  0.522 -0.372
sepal width (cm)  -0.263 -0.926
petal length (cm)  0.581 -0.021
petal width (cm)   0.566 -0.065

[4 rows x 2 columns]
```

For example, above shows that the second principal component (`PC-2`) is mostly aligned with `sepal width` , which has the highest weight of `0.926` in absolute value;

Since the data were normalized, you can confirm that the principal components have variance `1.0` which is equivalent to each coefficient vector having norm `1.0` :

```
>>> np.linalg.norm(coef,axis=0)
array([ 1.,  1.])
```

One may also confirm that the principal components can be calculated as the dot product of the above coefficients and the original variables:

```
>>> np.allclose(df_norm.values.dot(coef), pca.fit_transform(df_norm.values))
True
```

Note that we need to use `numpy.allclose` instead of regular equality operator, because of floating point precision error.

edited Jul 16 '17 at 11:50

answered Apr 10 '14 at 10:58

behzad.nouri

30.8k 9 70 79

2 Awesome and exhaustive answer, thank you very much ! – mazieres Apr 10 '14 at 11:56

3 There's no need for that identity matrix: your `coef` is the same as `pca.components_.T` . scikit-learn

3 insightful answer for people like me trying to learn PCA – goh Nov 8 '14 at 13:00

3 Why not directly use `pca.components_` ? – Sangram Jun 2 '16 at 11:24

1 Using the identity matrix doesn't work as the inverse transform function adds the empirical mean of each feature. The result gives equal weight (coefficients) to all original variables. (See this answer). By using `pca.components_` , you get the right answer. – Rahul Murmuria Jun 17 '16 at 3:21

This information is included in the `pca` attribute: `components_` . As described in the documentation, `pca.components_` outputs an array of `[n_components, n_features]` , so to get how components are linearly related with the different features you have to:

Note: each coefficient represents the correlation between a particular pair of component and feature

```
import pandas as pd
import pylab as pl
from sklearn import datasets
from sklearn.decomposition import PCA

# load dataset
iris = datasets.load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# normalize data
from sklearn import preprocessing
data_scaled = pd.DataFrame(preprocessing.scale(df),columns = df.columns)

# PCA
pca = PCA(n_components=2)
pca.fit_transform(data_scaled)

# Dump components relations with features:
print pd.DataFrame(pca.components_,columns=data_scaled.columns,index = ['PC-1','PC-
2'])
```

```
      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
PC-1           0.522372         -0.263355           0.581254          0.565611
PC-2          -0.372318         -0.925556          -0.021095         -0.065416
```

the sign does not affect the variance contained in each component. Check this post for further reference.

edited Oct 23 '17 at 8:04

answered Jan 9 '16 at 10:47

Rafa
794 9 21

1 This should be the accepted answer. – Rahul Murmuria Jun 17 '16 at 3:23

What happen when two component show one feature ? – Daniel.V Dec 1 '16 at 17:57

Components are actually combinations of features, so any particular feature is (at certain degree) correlated with different components.... – Rafa Dec 5 '16 at 12:09

1 So say you want to know which original feature was most important, should you just take the absolute values and sum them? What I mean is, starting from the last line from the answer: pd.DataFrame(pca.components_,columns=data_scaled.columns,index = ['PC-1','PC-2']).abs().sum(axis=0), which results in there values: 0.894690 1.188911 0.602349 0.631027. Could we hereby say that sepal width was most important, followed by sepal length? – Guido Feb 23 '17 at 15:17

2 To understand which features are important you need to pay attention to the correlations. For example, sepal width and PC-2 are strongly correlated (inversely) since the correlation coef is -0.92. In the other hand, petal length and PC-2 are not correlated at all since corr coef is -0.02. So, PC-2 grows as sepal width decreases and PC-2 is independent of changes in petal length. That is, for PC-2 sepal width is important while petal length is not. Same analysis you can conduct for the other variables considering correlation coef is in the interval [-1, 1] – Rafa Feb 24 '17 at 6:49

The way this question is phrased reminds me of a misunderstanding of Principle Component Analysis when I was first trying to figure it out. I'd like to go through it here in the hope that others won't spend as much time on a road-to-nowhere as I did before the penny finally dropped.

The notion of "recovering" feature names suggests that PCA identifies those features that are most important in a dataset. That's not strictly true.

PCA, as I understand it, identifies the features with the greatest variance in a dataset, and can then use this quality of the dataset to create a smaller dataset with a minimal loss of descriptive power. The advantages of a smaller dataset is that it requires less processing power and should have less noise in the data. But the features of greatest variance are not the

at all.

To bring that theory into the practicalities of @Rafa's sample code above:

```
# load dataset
iris = datasets.load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# normalize data
from sklearn import preprocessing
data_scaled = pd.DataFrame(preprocessing.scale(df),columns = df.columns)

# PCA
pca = PCA(n_components=2)
pca.fit_transform(data_scaled)
```

consider the following:

```
post_pca_array = pca.fit_transform(data_scaled)

print data_scaled.shape
(150, 4)

print post_pca_array.shape
(150, 2)
```

In this case, `post_pca_array` has the same 150 rows of data as `data_scaled`, but `data_scaled`'s four columns have been reduced from four to two.

The critical point here is that the two columns – or components, to be terminologically consistent – of `post_pca_array` are not the two "best" columns of `data_scaled`. They are two new columns, determined by the algorithm behind `sklearn.decomposition`'s `PCA` module. The second column, `PC-2` in @Rafa's example, is informed by `sepal_width` more than any other column, but the values in `PC-2` and `data_scaled['sepal_width']` are not the same.

As such, while it's interesting to find out how much each column in original data contributed to the components of a post-PCA dataset, the notion of "recovering" column names is a little misleading, and certainly misled me for a long time. The only situation where there would be a match between post-PCA and original columns would be if the number of principle components were set at the same number as columns in the original. However, there would be no point in using the same number of columns because the data would not have changed. You would only have gone there to come back again, as it were.

amunnelly
141 2 4

This helped. Thanks man – Aziz Javed Nov 28 '17 at 18:47

Thanks Aziz. I'm glad you were able to get something from it. – amunnelly Nov 29 '17 at 21:07

Given your fitted estimator `pca`, the components are to be found in `pca.components_`, which represent the directions of highest variance in the dataset.

answered Apr 10 '14 at 19:01

eickenberg
9,129 1 18 33