index

April 20, 2022

1 Performing Principal Component Analysis (PCA)

1.1 Introduction

In this lesson, we'll write the PCA algorithm from the ground up using NumPy. This should provide you with a deeper understanding of the algorithm and help you practice your linear algebra skills.

1.2 Objectives

You will be able to:

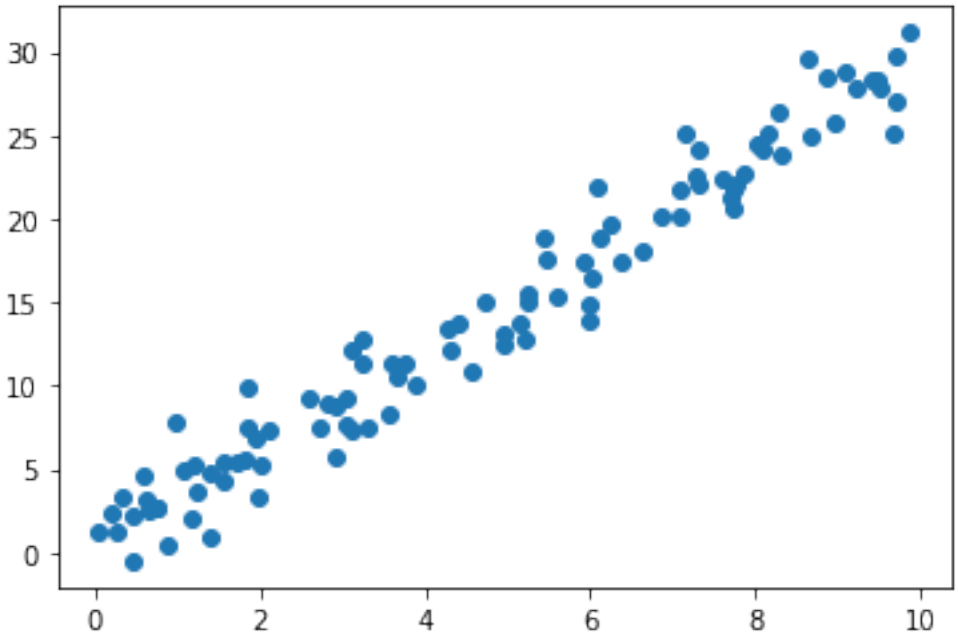
- List the steps required to perform PCA on a given dataset
- Decompose and reconstruct a matrix using eigendecomposition
- Perform a covariance matrix calculation with NumPy

1.3 Step 1: Get some data

To start, generate some data for PCA!

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
np.random.seed(42)

x1 = np.random.uniform(low=0, high=10, size=100)
x2 = [(xi*3)+np.random.normal(scale=2) for xi in x1]
plt.scatter(x1, x2);
```



1.4 Step 2: Subtract the mean

Next, you have to subtract the mean from each dimension of the data.

[2]:
```
import pandas as pd

data = pd.DataFrame([x1,x2]).transpose()
data.columns = ['x1', 'x2']
data.head()
```

[2]:

	x1	x2
0	3.745401	11.410298
1	9.507143	27.923414
2	7.319939	22.143340
3	5.986585	13.984617
4	1.560186	4.241215

[3]:
```
data.mean()
```

[3]:
```
x1     4.701807
x2    14.103262
dtype: float64
```

[4]:
```
mean_centered = data - data.mean()
mean_centered.head()
```

[4]:
```
          x1         x2
0 -0.956406  -2.692964
1  4.805336  13.820153
2  2.618132   8.040078
3  1.284777  -0.118645
4 -3.141621  -9.862046
```

[31]:
```
## My Code

XX = np.matrix(list(zip(x1, x2)))
XX_mean = XX.mean(axis = 0)
XX_diff = XX - XX_mean

## To check if my answer is correct
XX_df = pd.DataFrame(XX_diff, columns = ["x1", "x2"])
(XX_df == mean_centered).sum()
```

[31]:
```
x1    100
x2    100
dtype: int64
```

1.5 Step 3: Calculate the covariance matrix

Now that you have normalized your data, you can calculate the covariance matrix.

[35]:
```
cov = np.cov([mean_centered['x1'], mean_centered['x2']])
cov
```

[35]:
```
array([[ 8.84999497, 25.73618675],
       [25.73618675, 78.10092586]])
```

[47]:
```
### My Code
cov_matrix = XX_diff.T.dot(XX_diff)/(XX_diff.shape[0]-1)
print(type(cov_matrix))
print(type(cov))

cov_matrix_array = np.array(cov_matrix)
np.round(cov_matrix - cov, 10)
```

```
<class 'numpy.matrix'>
<class 'numpy.ndarray'>
```

[47]:
```
array([[-0., -0.],
       [-0., -0.]])
```

1.6 Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix

It's time to compute the associated eigenvectors. These will form the new axes when it's time to reproject the dataset on the new basis.

[48]:
```
eigen_value, eigen_vector = np.linalg.eig(cov)
eigen_vector
```

[48]:
```
array([[-0.94936397, -0.31417837],
       [ 0.31417837, -0.94936397]])
```

[49]:
```
eigen_value
```

[49]:
```
array([ 0.33297363, 86.61794719])
```

[55]:
```
eigval, eigvec = np.linalg.eig(cov_matrix)
print(eigval)
print()
print(eigvec)
```

```
[ 0.33297363 86.61794719]

[[-0.94936397 -0.31417837]
 [ 0.31417837 -0.94936397]]
```

1.7 Step 5: Choosing components and forming a feature vector

If you look at the eigenvectors and eigenvalues above, you can see that the eigenvalues have very different values. In fact, it turns out that **the eigenvector with the highest eigenvalue is the principal component of the dataset.**

In general, once eigenvectors are found from the covariance matrix, the next step is to order them by eigenvalue in descending order. This gives us the components in order of significance. Typically, PCA will be used to reduce the dimensionality of the dataset and as such, some of these eigenvectors will be subsequently discarded. In general, the smaller the eigenvalue relative to others, the less information encoded within said feature.

Finally, you need to form a **feature vector**, which is just a fancy name for a matrix of vectors. This is constructed by taking the eigenvectors that you want to keep from the list of eigenvectors, and forming a matrix with these eigenvectors in the columns as shown below:

[65]:
```
# Get the index values of the sorted eigenvalues
e_indices = np.argsort(eigen_value)[::-1]

# Sort
eigenvectors_sorted = eigen_vector[:, e_indices]
eigenvectors_sorted
```

[65]:
```
array([[-0.31417837, -0.94936397],
       [-0.94936397,  0.31417837]])
```

[66]:
```
np.argsort(eigen_value)[::-1]
```

[66]:
```
array([1, 0])
```

1.8 Step 5: Deriving the new dataset

This the final step in PCA, and is also the easiest. Once you have chosen the components (eigenvectors) that you wish to keep in our data and formed a feature vector, you simply take the transpose of the vector and multiply it on the left of the original dataset, transposed.

[9]:
```
transformed = eigenvectors_sorted.dot(mean_centered.T).T
transformed[:5]
```

[9]:
```
array([[  2.85708504,   0.06190663],
       [-14.63008778,  -0.2200194 ],
       [ -8.45552104,   0.04045849],
       [ -0.29101209,  -1.25699704],
       [ 10.34970068,  -0.11589976]])
```

[ ]:
```

```

1.9 Summary

That's it! We just implemented PCA on our own using NumPy! In the next lab, you'll continue to practice this on your own!