Principal Component Analysis

Magnus Chr. Hvidtfeldt

Technical University of Denmark, Lyngby, DK,
s255792@dtu.dk

1 Loading dataset

We are given a dataset of 10.000 people, each giving their gender, heigh and weight. The goal with this dataset is to use it, and perform Principal Component Analysis (PCA) on it. We are only interested in the weight and height, not the gender. Therefore we want to take our 2D data, and turn it into a 1D data keeping the most essential information, but in a dimension that is easier to see and do calcs on.

One might imagine that this analysis is useful for data in dimensions greater than 3, where we can't see it, and can thus scale them down into smaller dimensions, but still get the most relevant information.

We can load the dataset in python as such.

```python
from sympy import Matrix, init_printing
import numpy as np
import matplotlib.pyplot as plt
init_printing()
```

```python
# Load the CSV file into a NumPy array
data = np.genfromtxt('weight-height.csv', delimiter=',',
dtype=[('Gender', 'U10'), ('Height', float), ('Weight', float)],
names=True)

# Access the columns by their names
gender = data['Gender']
height = data['Height']
weight = data['Weight']

X = np.array([height, weight]).T
X.shape
```

```
(10000, 2)
```

The amount of people are 10.000, and we extract 2 columns, weight and height. Now, in the first step of PCA, we want to standardize it. This means that we want to center the data around $(0, 0)$, and make the average of each row 0. This is done in case we have varying information, then its to make it more accurate.

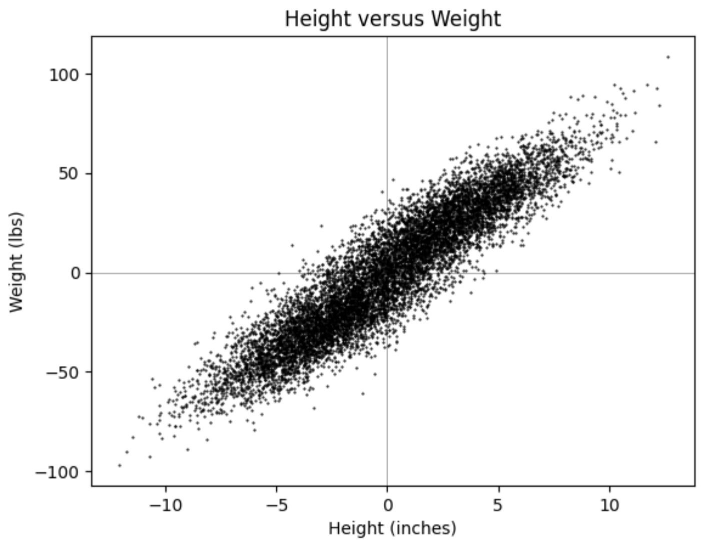
We take the mean of each column, and subtract the mean from our X columns as such.

```
me = X.mean(axis=0)
X_st = X - me
display(X_st)
```

```
array([[  7.47945726,  80.45320635],
       [  2.41434429,   0.87011569],
       [  7.74254564,  51.30049872],
       …,
       [ -2.49956754, -32.96503805],
       [  2.66668338,   2.41210451],
       [ -4.42331388, -47.79125416]], shape=(10000, 2))
```

What we get is X_{st}, which shows the difference in the mean for each row. Let's plot the data centered around $(0, 0)$ here.

```
plt.axhline(0, color='gray',linewidth=0.5)
plt.axvline(0, color='gray',linewidth=0.5)
plt.plot(X_st[:,0], X_st[:,1], '.k', markersize=1)
plt.title("Height versus Weight")
plt.xlabel("Height (inches)")
plt.ylabel("Weight (lbs)")
```



Here we can see that we successfully plotted all the data points around 0.

Now we want to get into the second step of PCA, which is to calculate the covariance matrix.

2 Covariance matrix

We can calculate the covariance matrix through

$$C := \frac{1}{n-1}(X_{st})^T X_{st}$$

Where n is the size of the array, in this case, we have $n = 10000$ people.

```
C = 1/(X_st.shape[0]-1) * X_st.T @ X_st
display(C)
```

```
array([[  14.80347264,  114.24265645],
       [ 114.24265645, 1030.95185544]])
```

Now that we have C, the next step is to calculate the eigenstuff of the C matrix. The eigenvalues will tell us which dimensions of data is most important to keep, and the eigenvectors helps us in seeing the position of the data we want to see. Let's calculate eigenstuff.

```
lamda, Q = np.linalg.eig(C)
display(lamda, Q)
```

```
array([   2.11786479, 1043.63746329])
```

```
array([[-0.99389139, -0.1103626 ],
       [ 0.1103626 , -0.99389139]])
```

Here we have our eigenvalues and Q matrix of eigenvectors. We clearly see that q_2 $\lambda_2 = 1043.63$ is most important, as that is the one with the highest eigenvalue. The amount of eigenvalues here also represent the dimension of our data, which is 2D in this case, but it could be any k.

3 Data analysis

Now, note what dimension you want to scale your data down to. In our case, we want to scale our 2D data down to 1D, so therefore, we will pick only one of the most important (highest) eigenvalue and its corresponding eigenvector.

We use the corresponding eigenvector(s) to project the data down to 1D through a projection matrix found through $P = uu^* = uu^T$, where $u = q_2/\|q_2\|_2$ ie, the normalized version of q_2 vector we chose. We essentially pick an orthonormal basis to project it down to the 1D line.

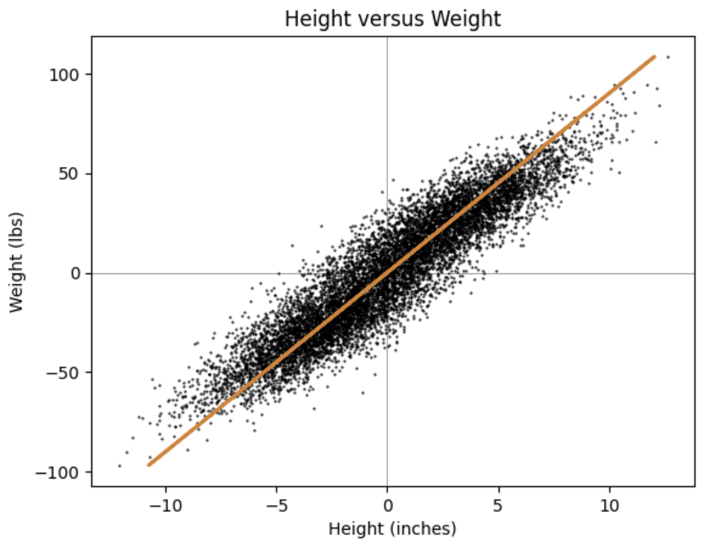
```
u_np = Q[:,1] / np.linalg.norm(Q[:,1])
P = u_np.reshape(2, 1) @ u_np.reshape(1, 2)
display(P)
```

```
array([[0.0121799 , 0.10968844],
       [0.10968844, 0.9878201 ]])
```

Then, we do matrix multiplication X_st on the transposed projection as $X_{st}P^T$ to get the projection. Let us also plot it.

```
proj = X_st @ P.T
plt.axhline(0, color='gray',linewidth=0.5)
plt.axvline(0, color='gray',linewidth=0.5)

plt.plot(X_st[:,0], X_st[:,1], '.k', markersize=1)
plt.plot(X_st @ P[:,0].T, X_st @ P[:,1].T, color="peru", linewidth=2)
plt.title("Height versus Weight")
plt.xlabel("Height (inches)")
plt.ylabel("Weight (lbs)")
```



Here we have our result. We have plotted both the standardized data plotted in 2D, as well as the projection of the data onto the q_2 basis (orange line), giving us a 1D line which is our PCA of 2D turned to 1D data, representing the most important data in a lower dimension.

This is useful for higher dimensions, and scaling it down to see something we otherwise could not. Here is an example.

4 Iris dataset

Here is an example of how we can scale 4D data (which we cannot see), into 3D data which we can see, using PCA. We use the Iris dataset.

The dataset contains 150 observations of three different species of iris flowers (Setosa, Versicolor and Virginica). Each flower is described by 4 physical measurements (features): sepal length/width and petal length/width.

The purpose of this part is to investigate whether the four physical measurements (length and width of sepal and petals) are sufficient to distinguish between the three different species of Iris flowers. Since we cannot visualize four dimensions at once, we use PCA as a “mathematical projector” to transform the data.

We want to see whether the flowers naturally occur as three distinct groups (clusters). If the species cluster into their own groups, it proves that the physical differences are systematic enough for us to classify the flowers mathematically.

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import numpy as np
```

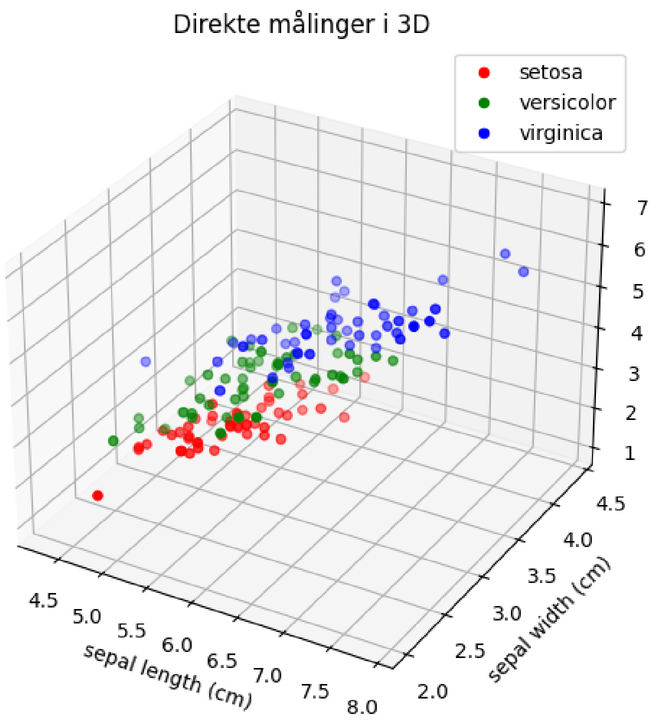
```
iris = load_iris()
X_iris = iris.data
y_iris = iris.target
names = iris.target_names
```

We first plot it using only 3 columns of the 4 we have.

```
# %matplotlib qt
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

# Vi vælger de første 3 ud af 4 søjler
colors = ['r', 'g', 'b']
for i in range(3):
    ax.scatter(X_iris[y_iris == i, 0],
               X_iris[y_iris == i, 1],
               X_iris[y_iris == i, 2],
               c=colors[i], label=names[i])

ax.set_xlabel(iris.feature_names[0])
ax.set_ylabel(iris.feature_names[1])
ax.set_zlabel(iris.feature_names[2])
ax.set_title("Direkte målinger i 3D")
ax.legend()
plt.show()
```



We can't see too much difference and where the clusters are, so we will do PCA analysis here.

```
# step. 1
X_st = X_iris - X_iris.mean(axis=0)
# step. 2
C = 1/(150-1) * X_st.T @ X_st
# step. 3
l, Q = np.linalg.eig(C)
display(l, Q)
```

```
array([4.22824171, 0.24267075, 0.0782095 , 0.02383509])

array([[ 0.36138659, -0.65658877, -0.58202985,  0.31548719],
       [-0.08452251, -0.73016143,  0.59791083, -0.3197231 ],
       [ 0.85667061,  0.17337266,  0.07623608, -0.47983899],
       [ 0.3582892 ,  0.07548102,  0.54583143,  0.75365743]])
```

Since we are scaling 4D data down to 3D, we pick the 3 most important eigenvalues and corresponding eigenvectors as such.

```
# step. 4
q1 = Q[:,0]
q2 = Q[:,1]
q3 = Q[:,2]
```

We will now project the data down into 3D using the three most important principal components (the PCA directions $\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3$, corresponding to the three largest eigenvalues). We form the matrix $Q_{reduced} = [\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3]$ of size 4×3 .

We use a different method than projection, instead of our projection matrix P, we use the eigenvectors to tell us our eigenbasis and thus position of the vectors, which we use to project into the smaller 3D plane.

The new 3D coordinates (also called *PCA scores*) are calculated as:

$$Z = X_{st} Q_{reduced}$$

Let us plot our PCA now.

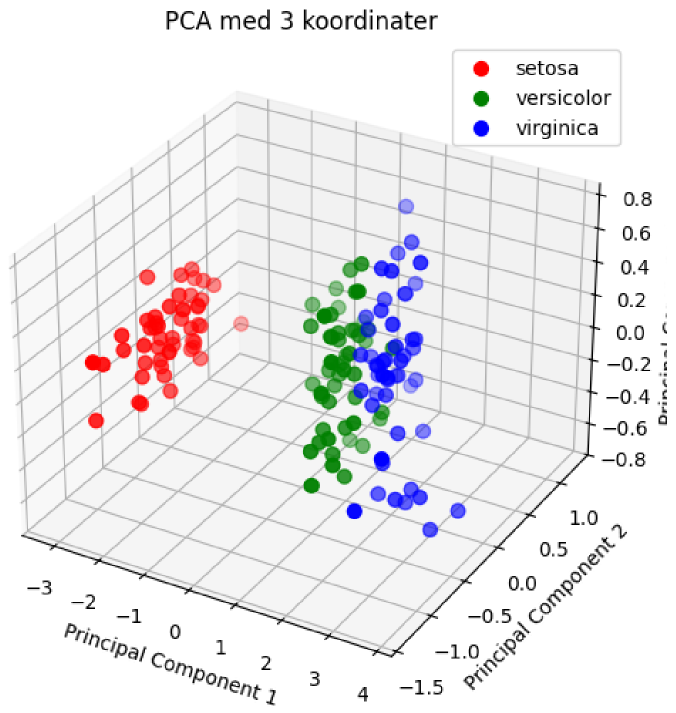
```
# step. 5
Q_red = np.array([q1, q2, q3]).T
Z = X_st @ Q_red
display(Q_red)
```

```
array([[ 0.36138659, -0.65658877, -0.58202985],
       [-0.08452251, -0.73016143,  0.59791083],
       [ 0.85667061,  0.17337266,  0.07623608],
       [ 0.3582892 ,  0.07548102,  0.54583143]])
```

```
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

colors = ['r', 'g', 'b']
for i, target_name in enumerate(names):
    # Vi bruger de tre søjler i Z som hhv. x, y og z koordinater
    ax.scatter(Z[y_iris == i, 0],
               Z[y_iris == i, 1],
                Z[y_iris == i, 2],
                c=colors[i],
                label=target_name,
                s=50)

ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.set_title('PCA med 3 koordinater')
```



After doing PCA and turning our 4D data into 3D, we can easily see the clusters compared to when we just removed a column. This shows how effective PCA can be for turning higher dimension data into something visible, and something we can work with.