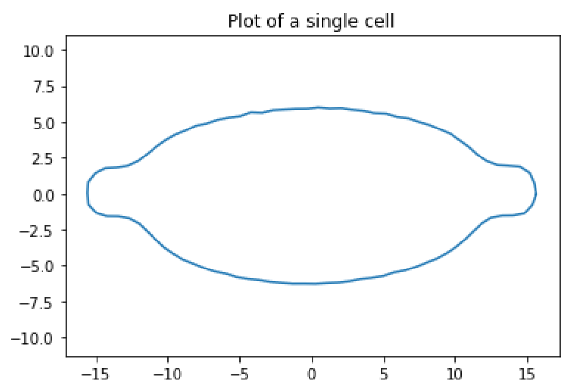
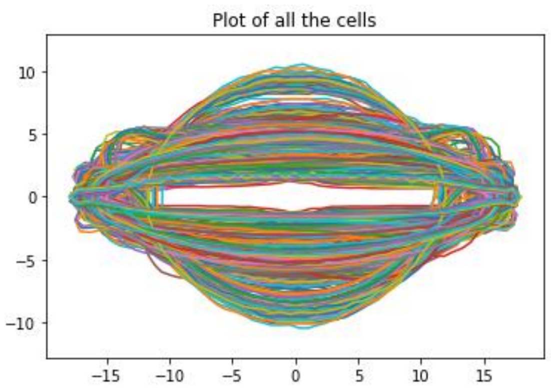
Introduction to Data Science

Assignment 4

Péter Pölös

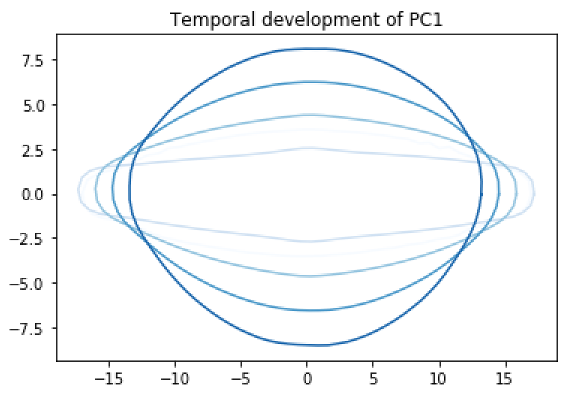
Exercise 1

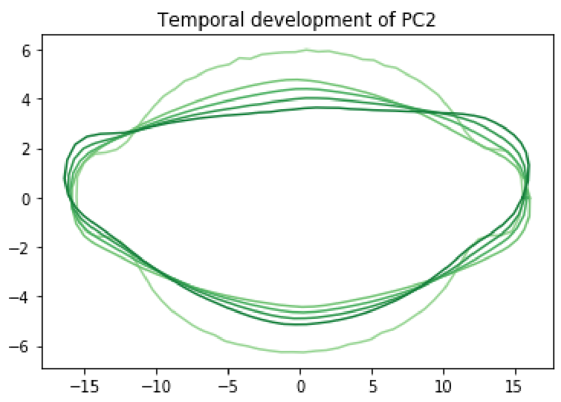


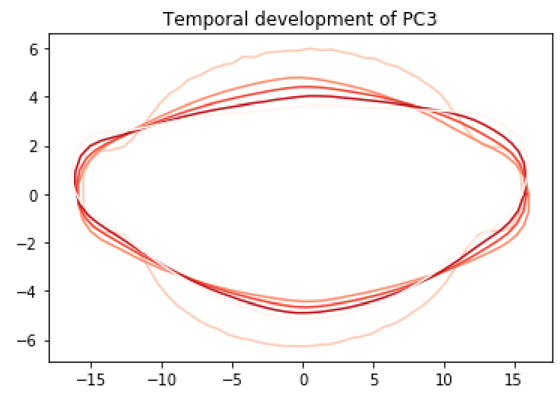


Plotting all the cells on top of each other we can see a trend in the data, that the shapes of these cells tend to take forms from elliptical to a more circular look.

Exercise 2







We can see the temporal development of the cells, as we know that the variance captured by the first principal component is by far the biggest, it is also shown on the plots that the sequence of cells have the biggest changes in case of the first principal components.

The variance captured by the 3 three different Principal Components:

```
[206.42706203  41.67416028   6.5308586 ]
```

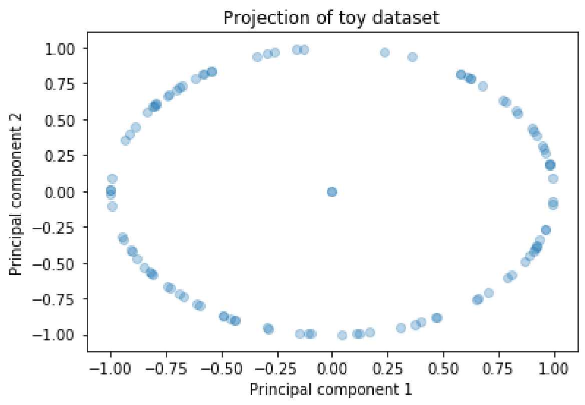
Exercise 3

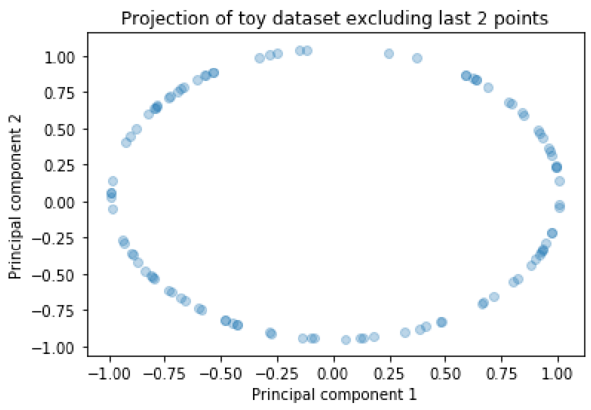
i) When we are doing PCA for computing covariance matrix we implicitly perform centering: variance, by definition, is the average squared deviation *from the mean*. Centered and non-centered data will have identical covariance matrices, so centering does not make any difference.

ii) Standardization is important doing before performing PCA, since PCA is a variance maximizing exercise. Principal component analysis will tend to give more emphasis to those variables that have higher variances than to those variables that have lower variances, so if you want to give equal weight to all of your data points then you need to standardize them.

iii) Whitening makes the data points uncorrelated. It is a step recommended to do after performing PCA not prior.

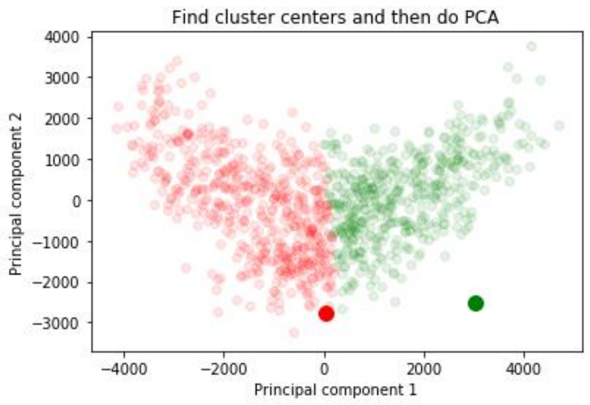
b)

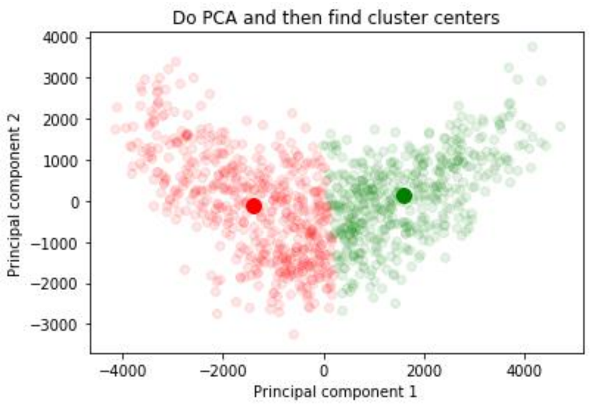




The obvious change in the plots, is that by excluding that two datapoints, the dot from the center(origin) disappears. (which is not surprising since those two datapoints only had zero coordinates)

Exercise 4





The cluster centers in 2D: (for the left plot)

```
[[ 3039.49368092 -2517.66768068]
 [   37.98660506 -2750.28531206]]
```

As it can be seen from the plots it is quite clear that we do not get meaningful clusters, since our centers are not really close to the data points at all. This is due to the fact that we did first the clustering and then the dimensional reduction, but in high dimensional spaces the distance measures do not work very well (as it is pointed out on these websites as well: https://stats.stackexchange.com/questions/99171/why-is-euclidean-distance-not-a-good-metric-in-high-dimensions)On the plot on the right hand side, I did first the PCA, the dimension reduction of the datapoints and then I found the cluster centers, and as it can be seen from the plot, I have found much more reasonable solutions this way.

For the description of how I did the clustering I can only repeat myself from the previous Assignment:

I have attached my code here just to make the explanation easier:

Step 1

- first step of my function is to choose the first n data points as the initial cluster centers. (choosing them randomly could have led to multiple solutions cause we can not know for sure if our function has converged to a global or local optimum)

Step 2

- then I computed the distance of each data point from each cluster center and assigned the label of the closest center to each, using this pairwise_distances_argmin function
- calculate the new cluster centers by taking the mean of all data points assigned to that centroid's cluster.

Step 3

- then I was repeating **Step 2**, by a while a loop until the centers were converging, meaning that the new centers were the same as the ones in the previous step

```
from sklearn.metrics import pairwise_distances_argmin

# MY implementation
def findclusters(X, n_clusters):
    #initial centers with the first two data points
    centers = X[0:n_clusters]

    while True:
        # calculate distance of all data points from all centers
        labels = pairwise_distances_argmin(X, centers)

        # centroid recalculation: taking the mean of all data points again
        new_centers = np.array([X[labels == i].mean(axis=0)
                                for i in range(n_clusters)])
        # do it until convergence
        if np.all(centers == new_centers):
            break
        centers = new_centers

    return centers, labels

findclusters(x_train, 2)
```

And the two cluster centers I have ended up are: