Assignment 3 - Introduction to Data Science

Ninell Oldenburg

March 8, 2022

Exercise 1

b)

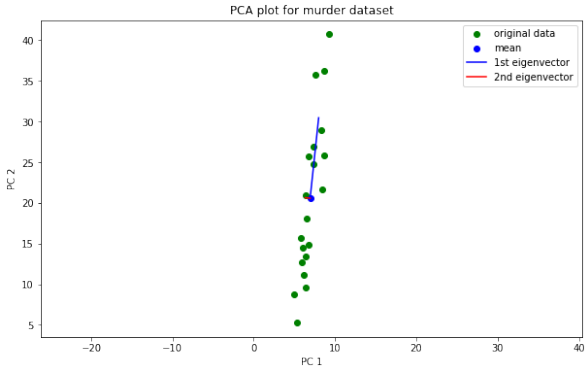

Figure 1: PCA plot for murder dataset

```
# of PCs to capture 90% of variance:  1
# of PCs to capture 95% of variance:  1
```

c)

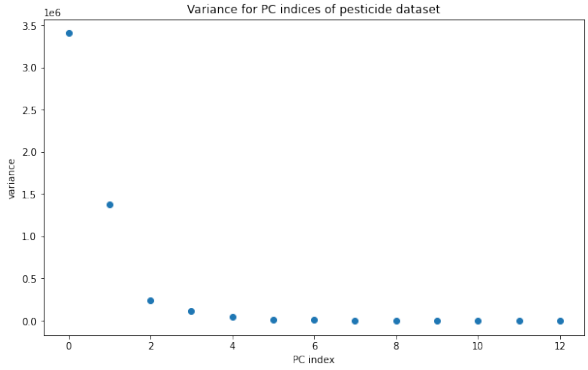

Figure 2: Variance for PC indices of pesticide dataset

```
# of PCs to capture 90% of variance:  2
# of PCs to capture 95% of variance:  3
```

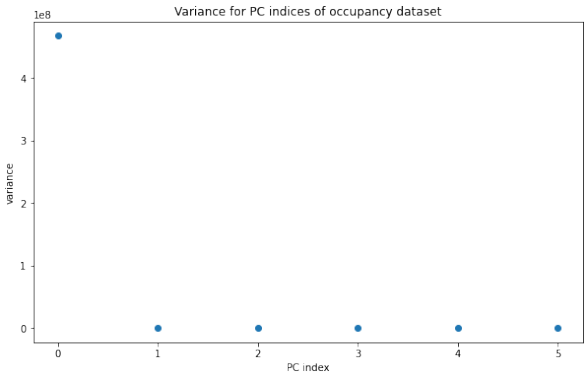

Figure 3: Variance for PC indices of occupancy dataset

d)

```
# of PCs to capture 90% of variance:  1
# of PCs to capture 95% of variance:  1
```

e)

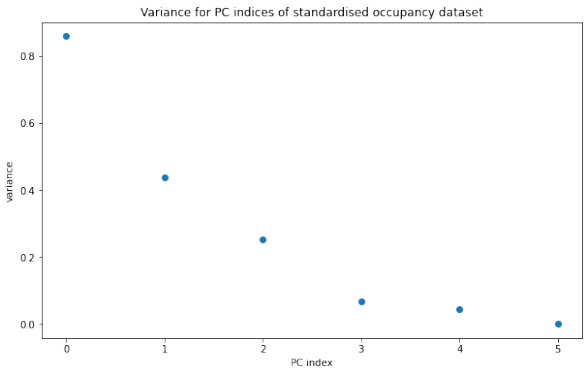

Figure 4: Variance for PC indices of standardised occupancy dataset

```
# of PCs to capture 90% of variance:  3
# of PCs to capture 95% of variance:  4
```

Exercise 2

a)

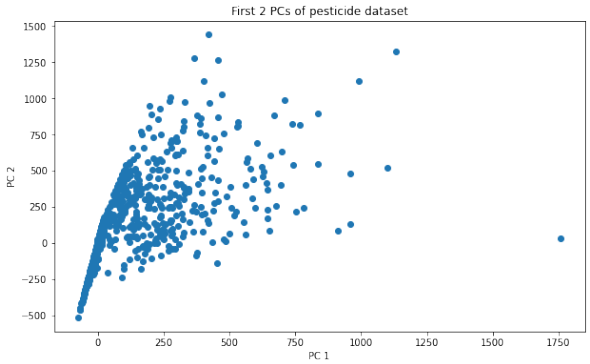

Figure 5: First 2 PCs of pesticide dataset

b)

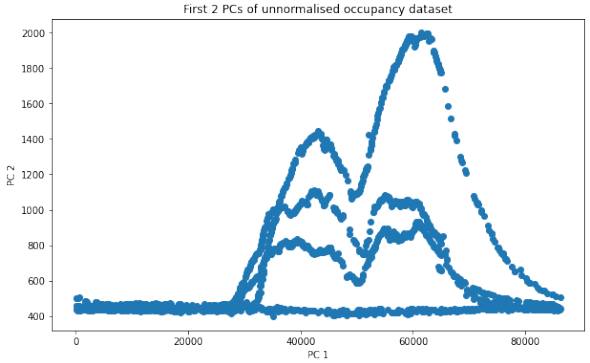

Figure 6: First 2 PCs of unnormalised occupancy dataset

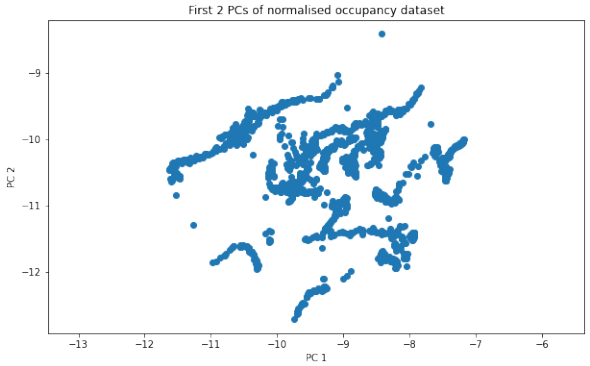

Figure 7: First 2 PCs of normalised occupancy dataset

c)

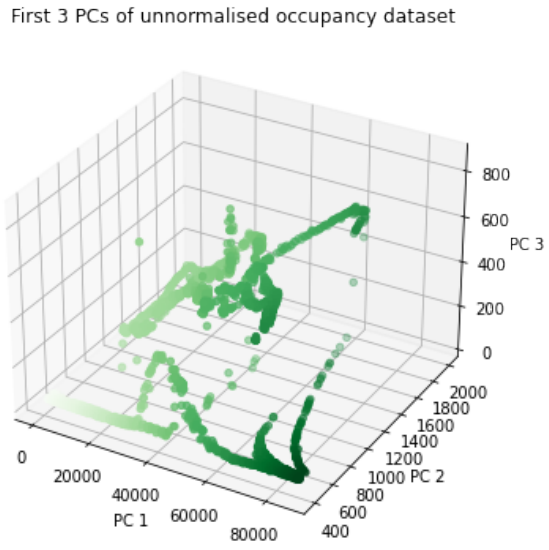

Figure 8: First 3 PCs of unnormalised occupancy dataset

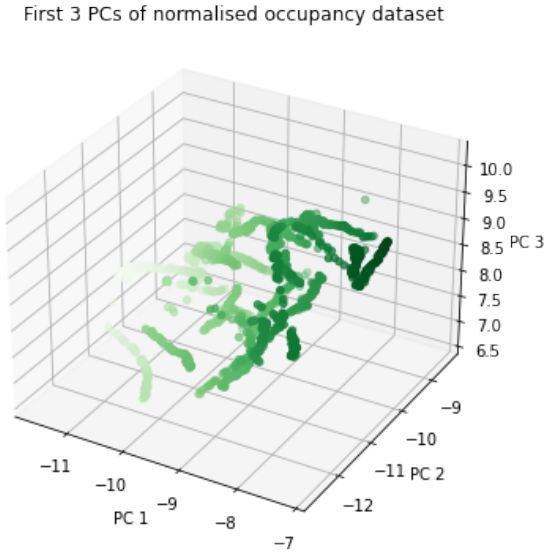

Figure 9: First 3 PCs of normalised occupancy dataset

Exercise 3

Description of the software. I've used the `KMeans` clustering method provided by the `sklearn.cluster` package. Here is how it works:

1. **Create a `KMeans` Class**. Here, we pass the parameters that we want to specialize our `KMeans` cluster with. In general, all of the parameters, e.g. the number of target clusters, the initial cluster centers, the maximum number of iterations to find cluster center, etc. have a default value, so we don't *have* to specify these. In our case, however, we tell the algorithm that we want to have 2 clusters, that it should start at `random_state=42`, we want it to use the `"full"` algorithm (so force it to not optimize on the more memory intense algorithm if the data clusters end up being well defined), that we only want to have one iteration with different center seeds (which makes sense as we are initializing the centers ourselves and increasing that number wouldn't change the output), and finally, pass the two center seeds that we've defined before as suggested by the assignment.

   The algorithm then returns a KMeans class that is tuned with these parameters and is ready to fit our data.

2. **Fit our data**. This is done in several stages.

   (a) **Choose initial cluster centers**. That can be done in several ways. 1) Centers are being initialized by the calling function (via the `init=` parameter). That's how I did it in this case as explained above and suggested/required by the assignment. The other way is the 2) the random or semi-random initialization where the centers are either being assigned completely random or semi-random by also selecting a `random_state` to make the results reproducible. The number of initial cluster centers corresponds of course always to the number of cluster centers we want to have and pass through the calling function (`n_clusters` parameter).

   (b) **Assign data points to closest centers**. Assign every data point (by which I mean vectors, not every single point!) to their closest center. So in our case every of the 13 points in our 1000 vectors is being assigned to their respective closest points at the respective index. In our case there are only two options.

   (c) **Calculate new cluster centers**. Now, that all data points are assigned to either of the clusters, we calculate the mean over all points in the cluster. That's how the initial cluster centers are being drawn more and more to the actual center of the potential clusters.

   (d) **Repeat steps (b) and (c)**. We stop when we've either 1) reached the `max_iter` parameter that is 300 per default, or 2) when the cluster center do not change anymore from the one iteration to the next one (that's how we know we've found the best possible mean for the center seeds of this iteration).

   (e) **Compare different cluster seeds**. This step is being skipped in our implementation due to us assigning the initial seeds. But usually, if they're picked at random, it makes sense to compare several seeds and see which one is having the best possible outcome. That is, which distribution of variance of the cluster centers has the most equal one over the data. If the variance over all cluster centers is comparably similar, we know that these are the best centers.

3. **Return attributes**. The `KMeans` class that is now tuned and trained on our data has now several attributes that are interesting for us. For this assignment, we're only interested in the `kmeans.cluster_centers_` that gives us, as the name suggests, the centroids of our cluster. In general, however, people like to look at the `kmeans.labels_`, that gives us an array of the size of our data, but instead of the data points/vectors, we get the labels (so the cluster index the point belongs to (in our case either 0 or 1)).

And this is our output:

```
cluster centers:

[[5.68220339e+00 4.93389831e+01 7.90923729e+02 3.84509110e+03 3.38647246e+03 1.36161229e+03
```

```
2.94523305e+02 1.31936441e+02 7.07457627e+01 3.95805085e+01 1.94152542e+01 4.22669492e+00
4.40677966e-01]
[2.19507576e+00 1.37007576e+01 1.70367424e+02 1.39206250e+03 3.18763636e+03 2.62546970e+03
1.00435985e+03 6.33471591e+02 4.96619318e+02 2.95941288e+02 1.46073864e+02 2.92537879e+01
2.84280303e+00]]
```

Exercise 4

- How is probability interpreted differently in the frequentist and Bayesian views?
  Bayesian assigns values & probabilities to every probability outcome, so it's specifying the degree to which I believe in a measure certainty. Frequentists see the probability a observed frequency.
- Cheap, efficient computers played a major role in making Bayesian methods mainstream. Why?
  Bayesian models are often not tractable with analytical methods. For this, we need to simulations that run on computers that come to a conclusion in a reasonable time show us the benefits of Bayesian models.
- What is the difference between a Bayesian credible interval and a frequentist confidence interval?
  Instead of calculating the chance that a given fixed parameter lies in an interval for a large number of observed samples (that can change), we calculate how certain we are that the value for our parameter of interest lies in a given interval.
- How does a maximum likelihood estimate approximate full Bayesian inference?
  The MLE is being plugged into the posterior distribution and then we're trying to find a value that maximizes that estimation function.
- When will point estimates be a good approximation of full Bayesian inference?
  It's a question about our desired outcome. If we rather want to have one point instead of a whole distribution, then a point estimate is a good approximation.