Uppsala University

Introduction to Machine Learning, Big Data, and AI

Assignment 6

General information

- The recommended tool in this course is R (with the IDE R-Studio). You can download R **here** and R-Studio **here**. You can use Python and Jupyter Notebooks, although the assignments may use data available only through the R package, a problem you would need to solve yourself.
- Report all results in a single, `*.pdf`-file. *Other formats, such as Word, Rmd, Jupyter Notebook, or similar, will automatically be failed.* Although, you are allowed first to knit your document to Word or HTML and then print the assignment as a PDF from Word or HTML if you find it difficult to get TeX to work.
- The report should be written in English.
- If a question is unclear in the assignment. Write down how you interpret the question and formulate your answer.
- You should submit the report to **Studium**. Deadlines for all assignments are **Sunday 23.59**. See **Studium** for dates. Assignments will be graded within 10 working days from the assignment deadline.
- To pass the assignments, *you should answer all questions not marked with **, and get at least 75% correct.
- To get VG on the assignment, *all questions should be answered, including questions marked with a **, although minor errors are ok. VG will only be awarded on the first deadline of the assignment.
- A report that does not contain the general information (see the **template**), will be automatically rejected.
- When working with R, we recommend writing the reports using R markdown and the provided **R markdown template**. The template includes the formatting instructions and how to include code and figures.
- Instead of R markdown, you can use other software to make the pdf report, but you should use the same instructions for formatting. These instructions are also available in **the PDF produced from the R markdown template**.
- The course has its own R package `uuml` with data and functionality to simplify coding. To install the package just run the following:

  1. `install.packages("remotes")`
  2. ```
     remotes::install_github("MansMeg/IntroML",
     subdir = "rpackage")
     ```

- We collect common questions regarding installation and technical problems in a course Frequently Asked Questions (FAQ). This can be found **here**.
- You are not allowed to show your assignments (text or code) to anyone. Only discuss the assignments with your fellow students. The student that show their assignment to anyone else could also be considered to cheat. Similarly, on zoom labs, only screen share when you are in a separate zoom room with teaching assistants.
     ```

- The computer labs are for asking all types of questions. Do not hesitate to ask! The purpose of the computer labs are to improve your learning. We will hence focus on more computer labs and less on assignment feedback. *Warning!* There might be bugs in the assignments! Hence, it is important to ask questions early on so you dont waste time of unintentional bugs.
- If you have any suggestions or improvements to the course material, please post in the course chat feedback channel, create an issue **here**, or submit a pull request to the public repository.

Contents

Introduction to Unsupervised Learning

This computer assignment will look into two to three widespread algorithms for unsupervised learning.

The dataset `iris` contain data on petal and sepal length and width for three different species of iris flowers. In addition, we will also use the `faithful` data on the Old Faithful Geyser in Yellowstone. The data consists of the eruption time (in minutes) and the waiting time (in minutes) until the next eruption. Finally we will also use the `mixture_data` from Table 8.1 in Hastie et al. (2009).

To access the data, just use:

```
data("iris")
data("faithful")
library(uuml)
data("mixture_data")
```

For a more indepth description of the datasets, just run `?iris`, `?faithful`, or `?mixture_data` in R.

1 The k-means Algorithm

We are now going to build a k-means algorithm from scratch.

1. In the `iris` data we will use the `Speices` variable as one way to cluster the data. Visualize (for yourself) the `faithful` data and manually allocate each data point to two different clusters of your choosing.

2. Visualize the `iris` as a scatterplot based on `Petal.length` and `Petal.width`. Show the different species using different colors. *Hint!* Use `ggplot2` and geom_point.

3. Similarily, visualize the `faithful` dataset as a scatterplot by `eruptions` and `waiting`. Show your manually assigned clusters in the Figure using different colors.

4. Now, we are going to implement the different parts of the k-means algorithm. This will follow Algorithm 14.1 in Hastie et al. (2009). First, implement Step 1 in this algorithm as a function called `compute_cluster_means(X, C)` that takes a $n \times p$ matrix `X` and a $p \times 1$ vector of cluster assignments called `C`, where n is the number of rows (observations) and p is the number of variables (columns). The algorithm should output the cluster means of the algorithm as a $K \times p$ matrix, where K is the total number of clusters in `C`.

```
set.seed(4711)
X <- as.matrix(faithful)
C <- sample(1:3, nrow(X), replace = TRUE)
m <- compute_cluster_means(X, C)
m

##      C eruptions  waiting
## [1,] 1  3.416354 70.57576
## [2,] 2  3.571106 71.44706
## [3,] 3  3.487659 70.72727
```

5. Next, implement the second step in Algorithm 14.1 of Hastie et al. (2009) and call the function `compute_cluster_encoding(X, m)`. The function should take a $n \times p$ (design) matrix `X` and a $K \times p$ matrix `m` with one cluster mean per row. The function should return a $n \times 1$ vector of cluster assignments.

```
C <- compute_cluster_encoding(X, m)
C[1:10]
## [1] 2 1 2 1 2 1 2 2 1 2
```

6. Now, use `compute_cluster_means(X, C)` and `compute_cluster_encoding(X, m)` to implement Algorithm 14.1 in Hastie et al. (2009) as `k_means(X,K)`. The `k_means(X,K)` should take an $n \times p$ matrix `X` and the number of clusters K. The function should return a $n \times 1$ vector of cluster assignments `C` and a $K \times p$ mean matrix `m` after the algorithm has run until no cluster assignments change. Randomly initialize the cluster assignments as a part of the algorithm.

7. Implement a function called `k_means_W(X,C)` that computes the k-means within-point scatter (WPS) of Eq. 14.31 in Hastie et al. (2009).

```
set.seed(4711)
X <- as.matrix(faithful)
C <- sample(1:3, nrow(X), replace = TRUE)
k_means_W(X, C)
## [1] 4601439
```

8. Now run your `k_means(X,K)` a couple of times for both `faithful` with $K = 2$ and `iris` with $K = 3$ using the two variables used above. Use different seeds every time. Compute the WPS for the final cluster assignments. Below is an example how this could be done.

```
set.seed(4711)
X <- as.matrix(faithful)
result1 <- k_means(X, K = 2)
wps1 <- k_means_W(X, result1$C)
result2 <- k_means(X, K = 2)
wps2 <- k_means_W(X, result2$C)
```

9. Visualize the clustering for the worst and the best clustering according to the WPS. *Note!* There might be multiple runs with the same (best) WPS.

10. Describe and discuss the results you get.

2 Probabilistic PCA

As a last part of the assignment, we will look into the probabilistic Principal Component analysis model. Here, Ch. 13 - 13.1 in Goodfellow et al. (2016) is helpful.

1. Implement a function `pPCA(W, b, sigma2)` that simulate data $\mathbf{x} \in \mathbb{R}^D$ from the probabilistic PCA model with $\mathbf{h} \sim N(0, I_K)$ where $\mathbf{h} \in \mathbb{R}^{K \times 1}$, $\mathbf{W} \in \mathbb{R}^{D \times K}$, $\mathbf{b} \in \mathbb{R}^{D \times 1}$ and $\sigma^2 = 1$.

2. Simulate 300 observation PCA model with $\mathbf{W} = (-1, 3)^T$, $\mathbf{b} = (0.5, 2)^T$, and $\sigma^2 = 1$. The resulting distribution should look something similar as in Figure 1.

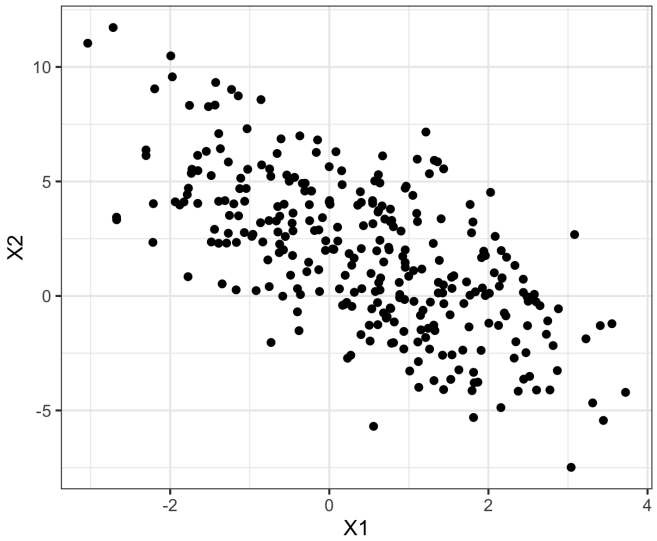

Figure 1: Data from a pPCA model with $\mathbf{W} = (-1, 3)^T$, $\mathbf{b} = (0.5, 2)^T$ and $\sigma^2 = 1$

3. Describe the different parts of the model. What are the parameters? What are the latent variables?

4. Try some other parameter values for W and b and visualize the result in a similar scatter plot as in Fig. 1.

5. Now run principal component analysis on your simulated data (with $\mathbf{W} = (-1, 3)^T$, $\mathbf{b} = (0.5, 2)^T$) using

```
pr <- prcomp(X, center = TRUE, scale. = FALSE)
```

   Multiply the first `stdev` with the first principal component (`rotation`).

6. What parameters in your pPCA model do these results correspond to. *Note!* You might have to multiply your result/vector with -1.

7. Finally, simulate values $\mathbf{x}$ of dimension five from a probabilistic PCA model for a given choice of parameters. The latent variables $\mathbf{h}$ should be of dimension two, i.e. $\mathbf{h} \in \mathbb{R}^{2 \times 1}$ for each simulated observation. You are free to choose the parameter

values $\mathbf{W}$, $\mathbf{b}$ and σ^2 yourself, but the resulting distribution should have at least one negative and one positive correlation between the Xs. Visualize the variables using a pair-wise scatterplot using `pairs(X)` in R.

3 * The EM-algorithm for soft clustering: Univariate Two-Component Normal Mixture

This task is only necessary to get one point toward a *pass with distinction* (VG) grade. Hence, if you do not want to get a *pass with distinction* (VG) point, you can ignore this part of the assignment.

In the next step, we will implement the EM algorithm conceptually similar to k-means clustering. We are going to implement the EM by testing at the mixture data of Hastie et al. (2009). This dataset has been added to the course R package.

```
library(uuml)
data("mixture_data")
theta0 <- list(mu_1 = 4.12, mu_2 = 0.94, sigma_1 = 2, sigma_2 = 2, pi = 0.5)
```

1. Simulate data from a univariate mixture model by implementing a mixture model as in Eq. 8.36 in Hastie et al. (2009). That is, create a hiearchical simulation that first sample δ for each observation, and then sample Y_i from the relevant component. The function should be called `r_uni_two_comp(n, theta)` take have the arguments `n` (the number of observations to simulate) (the number of simulated observations) and `theta`, an R list with with values `mu1`, `mu2`, `sigma1`, `sigma2`, and `pi`.

2. Simulate 200 observations using $\mu_1 = -2$, $\mu_2 = 1.5$, $\sigma_1 = 2$, $\sigma_2 = 1$ and $\pi = 0.3$. Visualize observations drawn in a histogram.

3. Implement a function `d_uni_two_comp(x, theta)` that computes the density value for a given set of parameters `theta` for values of `x`. *Hint!* The function `dnorm` might be handy.

4. Visualize the density for the mixture model above over the interval -4 to 4 (by steps of 0.01).

5. Visualize the `eruptions` variable in the `faithful` dataset using a histogram.

6. Manually choose values for μ_1, μ_2, σ_1, σ_2 and π that gives a density you think is a good fit for the `eruptions` variable.

7. Implement a function called `e_uni_two_comp(X, theta)` that returns a vector of gamma values for each row in `X` (for component 2). This function should implement the expectation step in algorithm 8.1 in Hastie et al. (2009).

   ```
   gamma <- e_uni_two_comp(mixture_data, theta0)
   head(gamma)
   ## [1]  0.9106339 0.8716861 0.7797225 0.6645640 0.6484311 0.5178799
   ```

   Note! Gamma values for component 1 is just 1 - `gamma`.

8. How can we interpret γ here?

9. Implement a function called `max_uni_two_comp(X, gamma)` that returns a list with parameters `mu1`, `mu2`, `sigma1`, `sigma2` and `pi`. This function should implement the maximization step in algorithm 8.1 in Hastie et al. (2009).

```
theta <- max_uni_two_comp(mixture_data, gamma)
theta
## $mu_1
## [1] 3.842941
##
## $mu_2
## [1] 1.450413
##
## $sigma_1
## [1] 1.700666
##
## $sigma_2
## [1] 1.47168
##
## $pi
## [1] 0.4883709
##
```

10. Implement the log-likelihood of the model as `ll_uni_two_comp(x, theta)`. *Hint!* See Eq. 8.39 in Hastie et al. (2009).

```
ll_uni_two_comp(mixture_data, theta0)
## -43.1055
```

11. Now combine the implemented functions to an EM algorithm `em_uni_two_comp(X, theta_0, iter)` that takes in a $n \times p$ data matrix `X` and an initialization value for as `theta_0`. Then the algorithm should be run `iter` number of iterations. For each iteration print out the log-likelihood value for that current iteration and store the value of θ for each iteration.

```
theta_and_gamma_em3 <- em_uni_two_comp(mixture_data, theta0, iter = 3)
## Log Lik:  -41.53247
## Log Lik:  -41.11211
## Log Lik:  -40.48348
```

12. Test your algorithm on the `mixture_data` for 20 iterations. Do you get the same results for $\hat{\pi}$ as in Hastie et al. (2009, p. 275)? Note that you might get slightly different results (at the second decimal place). Also, compare your log-likelihoods with Figure 8.6 in Hastie et al. (2009). Do you have a similar result? *Hint!* There can be a slight different order depending on if you compute the log-likelihood value before or after the update of θ.

```
theta_and_gamma_em20 <- em_uni_two_comp(mixture_data, theta0, iter = 20)
## Log Lik:  -41.53247
## ...
```

13. Run your EM algorithm on the `eruptions` variable of the `faithful` data and on the `Petal.Length` variable of the `iris` data. What are the estimated parameters for the two datasets?

14. Visualize the density for the two datasets in two different figures using the parameters estimated with your EM algorithm. Also show histograms for the real data in the same Figures (i.e. so you show the data and the estimated density). *Hint!* You should be able to use `d_uni_two_comp(x, theta)`.

References

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.