Uppsala University

Introduction to Machine Learning, Big Data, and AI

Assignment 7

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 are allowed to 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 to first knit your document to Word and then print the assignment as a PDF from Word if you find it difficult to get TeX to work.
- The report should be submitted to the **Studium**.
- To pass the assignments, *all questions should be answered*, although minor errors are ok.
- A report that do not contain the general information (see 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.
- If you have a problem with creating a PDF file directly from R markdown, start by creating an HTML file, and then just print the HTML to a PDF.
- Instead of R markdown, you can use other software to make the pdf report, but the same instructions for formatting should be used. 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**.
- Deadline for all assignments is **Sunday at 23.59**. See the course page for dates.
- If you have any suggestions or improvements to the course material, please post in the course chat feedback channel, create an issue, or submit a pull request to the public repository!
     ```

1 Introduction to Unsupervised Learning

In this computer assignment we will look into three very common algorithms for unsupervised learning and will study the algorithms on three different datasets.

The dataset `iris` contain data on petal and sepal length and width for three different speices 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.

1.1 Unsupervised learning

1. As a first step, describe, with your own words what unsupervised learning, and give relevant references to the course literature for this description.

1.2 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 $p \times 1$ vector of cluster assignments after the algorithm has run until no cluster assignments change. Use random initialization of cluster means.
7. Implement a function called `k_means_W(X,C)` that computes the k-means within-point scatter 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 a couple of times for both `faithful` with $K = 2$ and `iris` with $K = 3$ using the two variables used above.
9. Visualize the clustering for the worst and the best clustering according to the within-point scatter (if there are any differences between the runs).
10. Describe and discuss the results you get.

1.3 The EM-algorithm

In the next step we will implement the EM algorithm, that is conceptually very similar to k-means klustering.

1.3.1 Univariare Two Component Normal Mixture

We are going to implement the EM by testing at the data of Hastie et al. (2009). This 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)
y <- mixture_data
```

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(y, theta0)
   head(gamma)
   ## [1]  0.9106339 0.8716861 0.7797225 0.6645640 0.6484311 0.5178799
   ```

   Note! Gamma values for component 1 is essentially 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(y, 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!* You can use large parts of tour `em_uni_two_comp(x, theta)`

```
ll_uni_two_comp(y, 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 iterations.

```
theta_em <- em_uni_two_comp(y, theta0)
## Log Lik:  -41.53247
## Log Lik:  -41.11211
## Log Lik:  -40.48348
```

12. Test your algorithm on the `mixture_example` for 20 iterations. Do you get the same results 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?

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 using the parameters estimated with your EM algorithm. *Hint!* You should be able to use `d_uni_two_comp(x, theta)`.

1.4 Probabilistic PCA

As a last part of the assignment, we will look into the probabilistic Principal Component analysis model. Here, Ch. 14 - 14.1 in Goodfellow et al. is useful.

1. Implement a function `pPCA(W, b, sigma2)` and that simulated data from a probabilistic PCA model with $h \sim N(0, I)$ and $\sigma^2 = 1$

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

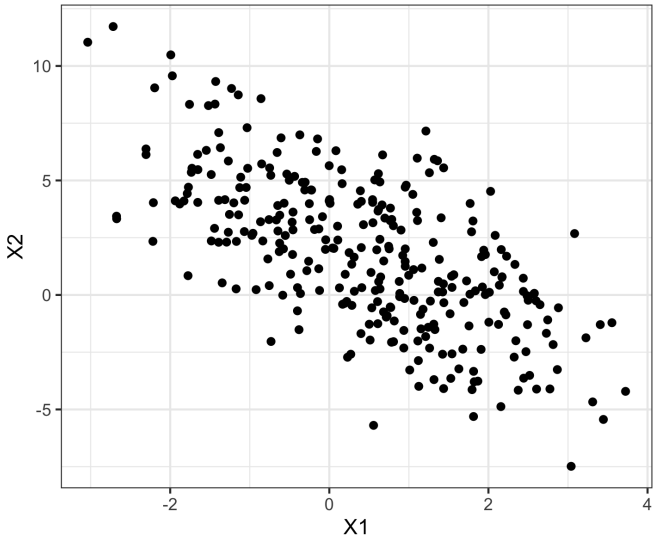

Figure 1: Data from a pPCA model with $W = (-1, 3)^T$, $b = (0.5, 2)$ and $\sigma^2 = 1$

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

4. Now run principal component analysis on your simulated data using

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

   Multiply the first `stdev` with the first principal component (`rotation`). What parameters in your pPCA does this correspond to. *Note!* You might have to multiply your result with -1.

5. Finally, simulate values $\mathbf{x}$ of dimension five from a probabilistic PCA model for a given choice of parameters. The latent variables H should be of dimension two, i.e. $H \in \mathbb{R}^{300 \times 2}$. You are free to choose the parameter values W, b and σ yourself, but the resulting distribution should have one at least one negative and one positive correlation between the Xs. Visualize the variables using a pair-wise scatterplot using `pairs(X)`.

References

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