Statistical Machine Learning 2016

Assignment 4
Deadline: Thursday 12 January 2017

Instructions:

- **IMPORTANT: Write the full name of all team members on the first page of the report.**
- Weights of the exercises in this assignment:
  - Exercise 1: 8 (+1 bonus)
  - Exercise 2: 12 (+1 bonus)
  - Exercise 3: 10
  - Exercise 4: 10
- Working together in **pairs** (that is, at most two persons) and handing in a single set of solutions per couple is recommended.
- Write a **self-contained report** with the answers to each question, **including** comments, derivations, explanations, graphs, etc.
- Note: Answers like 'No', or 'x=27.2' by themselves are not sufficient; this hold also for results that are only available by running your code.
- Note: All figures should have axis labels and a caption or title that states to which exercise (and part) they belong.
- If an exercise requires coding, put **relevant code snippets** in your answer to the question in the report, and describe what it does. E.g. for a plot show how you compute the function.
- Upload reports to **Blackboard** as a **single pdf** file: '`SML_A4_<Namestudent(s)>.pdf`', in combination with **one zip-file** with the executable source/data files (e.g. matlab m-files).
- The grading will solely be based on the report pdf file. The source files are considered as supplementary material.
- Email addresses: `tomc@cs.ru.nl` and `b.kappen@science.ru.nl`
- For problems or questions: use the BB discussion board, email, or just ask.

Exercise 1 – Gaussian processes for regression

We would like to apply Gaussian process models to the problem of regression (Bishop 6.4.2). We consider a noisy model of the form:

$$t_n = y_n + \epsilon_n, \tag{1}$$

where $y_n = y(\mathbf{x}_n)$ and ϵ_n are i.i.d. samples from a random noise variable on the observed target values. Furthermore, we assume that the noise process has a Gaussian distribution given by:

$$p(t_n|y_n) = \mathcal{N}(t_n|y_n, \beta^{-1}) \tag{2}$$

One widely used kernel function for Gaussian process regression is given by the exponential of a quadratic form, with the addition of constant and linear terms (eq. 6.63 Bishop):

$$k(\mathbf{x}, \mathbf{x}') = \theta_0 \exp\left(-\frac{\theta_1}{2}||\mathbf{x} - \mathbf{x}'||^2\right) + \theta_2 + \theta_3 \mathbf{x}^T \mathbf{x}' \tag{3}$$

We denote by $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \theta_3)$ the hyperparameter vector governing the kernel function k.

1. Implement the kernel given by Equation (3) in MATLAB as a function of $\mathbf{x}$, $\mathbf{x}'$ and $\boldsymbol{\theta}$.

2. For the parameter values $\boldsymbol{\theta} = (1, 1, 1, 1)$ and $N = 101$ equally spaced points $\mathbf{X}$ in the interval $[-1, 1]$, compute the Gram matrix $\mathbf{K}(\mathbf{X}, \mathbf{X})$ (eq. 6.54 Bishop). (*Hint:* $\mathbf{x}$ and $\mathbf{x}'$ are univariate in this case; use `linspace` to determine the points $\mathbf{X}$)

3. What is the dimension of $\mathbf{K}$? How can we show that $\mathbf{K}$ is semipositive definite?

4. We will now use the previously computed matrix $\mathbf{K}(\mathbf{X}, \mathbf{X})$ to produce samples from the Gaussian process prior $\mathbf{y}(\mathbf{X}) \sim \mathcal{N}(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X}))$, with $\mathbf{X}$ being the previously determined N equally spaced points. Generate five functions $\mathbf{y}(\mathbf{X})$ with MATLAB and plot them against the N input values $\mathbf{X}$.

5. Repeat 2. and 4. for the hyperparameter configurations from Bishop, Figure 6.5:

   $$\boldsymbol{\theta} \in \{(1, 4, 0, 0), (9, 4, 0, 0), (1, 64, 0, 0), (1, 0.25, 0, 0), (1, 4, 10, 0), (1, 4, 0, 5)\}.$$

   Bonus (1p): Briefly describe the differences between the plots and try to explain why / how changing the kernel parameters leads to these differences.

We now consider the following training data consisting of four data points:

$$\mathcal{D} = \{(x_1 = -0.5, t_1 = 0.5), (x_2 = 0.2, t_2 = -1), (x_3 = 0.3, t_3 = 3), (x_4 = -0.1, t_4 = -2.5)\} \tag{4}$$

6. Compute the Gram matrix of the training data for $\boldsymbol{\theta} = (1, 1, 1, 1)$. Then, taking $\beta = 1$ in Equation (2), compute the covariance matrix $\mathbf{C}$ corresponding to the marginal distribution of the training target values: $p(\mathbf{t}) = \mathcal{N}(\mathbf{t}|\mathbf{0}, \mathbf{C})$.

7. Using the previous results, compute the mean and the covariance of the conditional distribution $p(t|\mathbf{t})$ of a new target value t corresponding to the input $x = 0$. [Which equations from Bishop do you need?]

8. Does the mean of the conditional distribution $p(t|\mathbf{t})$ go to zero in the limit $x \to \pm\infty$? If so, explain why this happens. If not, how would you set the parameters $\boldsymbol{\theta}$ of the kernel function to make it happen?

Exercise 2 – Neural network regression

We train a neural network using backpropagation, to learn to mimic a 2D multimodal probability density. First, we implement the network and test its regression capabilities on a standard Gaussian; then we train it on the real data set. Visualization of the network output plays an important role in monitoring the progress.

1. Create a plot of an isotropic 2D Gaussian $y = 3 \cdot \mathcal{N}(\mathbf{x}|\mathbf{0}, \frac{2}{5}\mathbf{I}_2)$ centered at the origin using the `meshgrid()`, `mvnpdf()` and `surf()` functions. Sample the density at 0.1 intervals over the range $[-2, 2] \times [-2, 2]$ and store the data in column vector variables $\mathbf{X}$ (2D) and $\mathbf{Y}$ (1D).

2. Implement a 2-layer neural network with $D = 2$ input nodes, $K = 1$ output nodes and M hidden nodes in the intermediate layer that can be trained using a sequential error backpropagation procedure, as described in Bishop §5.3. Use $\tanh(\cdot)$ activation functions for the hidden nodes and a linear activation function (regression) for the output node. Introduce appropriate weights and biases, and set the learning rate parameter to $\eta = 0.1$. Initialize the weights to random values in the interval $[-0.5, 0.5]$. Plot a 2D graph of the initial output of the network over the same $[-2, 2] \times [-2, 2]$ grid as the Gaussian (again using `surf()`).

3. Train the network for $M = 8$ hidden nodes on the previously stored $\mathbf{X}$ and $\mathbf{Y}$ values (the $\{x_1, x_2\}$ input coordinates and corresponding output probability density y), by repeatedly looping over all datapoints and updating the weights in the network after each point. Repeat for at least 500 complete training cycles and monitor the progress of the training by plotting the output of the network over the $\mathbf{X}$ grid after each full cycle. Verify the output starts to resemble the Gaussian density after some 200 cycles (all be it with lots of 'wobbles').

4. Permute the $\mathbf{X}$ and $\mathbf{Y}$ arrays to a random order using the `randperm()` function, keeping corresponding $\mathbf{x}$ and y together. Repeat the network training session using this randomized data set. Verify that convergence is now much quicker. Can you understand why?
   Try out the effect of different numbers of hidden nodes, different initial weights and different learning rates on speed and quality of the network training. Explain your results.

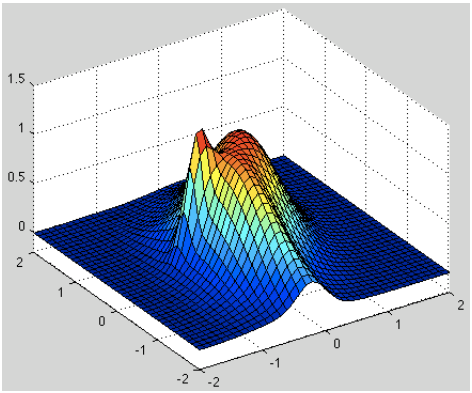

Figure 2 - Multi-modal probability density.

After these preliminaries we are now going to train the network on the real data set.
Load the data using

```
data = load('a017_NNpdfGaussMix.txt', '-ASCII');
X = data(:,1:2); Y = data(:,3);
```

5. Create a 2D-plot of the target probability density function. Notice that the data is in the correct sequence to use in `surf()`.

6. Train the network on this data set. Use at least 40 hidden nodes and a learning rate parameter of no higher than $\eta = 0.01$. Make sure the input data is properly randomized. Run the training phase for at least 2000 complete cycles and follow the progress by plotting the updated network output after every 20 full cycles.
   How does the final output of the network compare to the target distribution in the data? Explain. Try to give suggestions how you might improve the neural network in terms of speed of convergence and/or quality of the approximation?

7. **Bonus (1p)**: Download and install the *Netlab* toolbox for MATLAB via the website of the book. Use the standard `mlp()` and `netopt()` functions to create and train a neural network on our data set (using scaled conjugate gradients, option 'scg'). Compare the final output of the trained network, `mlpfwd()`, with our results.

Exercise 3 – EM and doping

In a certain hypothetical sport, banned substance 'X' has become popular as a performance enhancing drug, as its presence is hard to establish in blood samples directly. Recently, it has been discovered that users of the drug tend to show a strong positive correlation between concentrations of two other quantities, x_1 and x_2, present in the blood. In contrast, 'clean' athletes tend to fall in one of two or three groups, that either show no or a negative correlation between x_1 and x_2. Unfortunately, as each sample contains only a single, instantaneous, measurement for each variable, it is not possible to establish this correlation from the sample. However, in many cases it is possible to distinguish to which *class* a certain sample belongs by also looking at the values of two other measured variables, x_3 and x_4: certain combinations of measured values are often typical for one class but highly unusual for others.
After a high profile event, a large scale test has resulted in 2000 samples. Rumours suggest the number of positives could be as high as 20%. However, the exact relationship between different classes and typical $\mathbf{x}$ values is still not clear. This is where the EM-algorithm comes in ...

The blood sample measurements are modelled as a mixture of K Gaussians, one for each class

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{5}$$

where $\mathbf{x} = [x_1, x_2, x_3, x_4]$ represents the values for the measured quantities in the blood sample, $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K\}$ and $\boldsymbol{\Sigma} = \{\Sigma_1, \ldots, \Sigma_K\}$ are the means and covariance matrices of the Gaussians for each class, and $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_K\}$ are the mixing coefficients in the overall data set.

Load the data using

```
X = load('a011_mixdata.txt', '-ASCII');
```

and set N to the number of datapoints and D to the number of variables in the data set X.

1. Try to give an estimate of the number, size and shape of the classes in the data, by plotting the distribution of the variables, e.g using `hist()`, `scatter()` or `scatter3()`.

2. Implement an EM-algorithm using the description and formulas given in Bishop, §9.2.2. Use variable K for the number of classes and choose a priori equal mixing coefficients π_k. Initialize the means $\boldsymbol{\mu}_k$ to random values around the sample mean of each variable, e.g. set $\mu_{k,1}$ to $\bar{x}_1 + [-1 \leq \epsilon \leq +1]$. Initialize the $\boldsymbol{\Sigma}_k$ to diagonal matrices with reasonably high variances, e.g. `4*rand()+2`, to avoid very small responsibilities in the first step. Make sure the EM-loop runs over at least 100 iterations. Display relevant quantities, at least the log likelihood (9.28), after each step so you can monitor progress and convergence. Write a plot routine that plots the x_1, x_2 coordinates of the data points, and color each data point according to the most probable component according to the mixture model.

3. Set $K = 2$, initialize your random generator and run the EM-algorithm on the data. Describe what happens. Try different random initializations and compare results.
   (Should converge within 50 steps to two clusters, accounting for ±1/3 resp. 2/3 of the data).
   Plot the x_1, x_2 coordinates colored according to the most probable component. Compute the correlation coefficients
   $$\rho_{12} = \frac{\mathrm{cov}[x_1, x_2]}{\sqrt{\mathrm{var}[x_1]\mathrm{var}[x_2]}} \tag{6}$$
   of each of the components (i.e., use their covariance matrices to compute variances and covariances in (6), see also (Bishop, eq. (2.93)). Does either class show the characteristic strong[1] positive correlation for $\{x_1, x_2\}$?

4. Increase the number of classes to $K = 3$ and rerun your algorithm on the data, again trying different random initializations. Plot the x_1, x_2 coordinates colored according to the most probable component and compute the correlation coefficients of each of the components. Check both your plot and your coefficients if one of the clusters now displays the strong positive $\{x_1, x_2\}$ correlation we are looking for.
   Increase to $K = 4$, do the same, and see if this improves your result (in terms of detection of the doping-cluster). Based on your findings, is the rumoured 1-in-5 estimate for users of X credible?

Having found the offending cluster in the data using the EM-algorithm, we are now presented with four samples $\{A, B, C, D\}$, with values for $[x_1, x_2, x_3, x_4]$ given as:

$$\begin{aligned} \mathrm{A} &= [11.85, 2.2, 0.5, 4.0] \\ \mathrm{B} &= [11.95, 3.1, 0.0, 1.0] \\ \mathrm{C} &= [12.00, 2.5, 0.0, 2.0] \\ \mathrm{D} &= [12.00, 3.0, 1.0, 6.3] \end{aligned}$$

One of these is from a subject who took drug X, and one is from a subject who tried to tamper with the test by artificially altering one or more of the x_i levels in his/her blood sample.

5. Identify which sample belongs to the suspected user and which one belongs to the 'fraud'.

Exercise 4 – Handwritten digit recognition

We use the EM-algorithm to tackle the problem of recognizing real, handwritten digits. The dataset contains 800 digital images of handwritten numbers '2', '3' and '4', derived from the MNIST dataset (Bishop, app.A). Each image consists of 28x28 binary pixels that are either black (= 1) or white (= 0). The labels belonging to the images are (initially) unknown.
The bitmapped images are modelled with a Bernoulli mixture model (Bishop §9.3.3):

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{k=1}^{K} \pi_k \prod_{i=1}^{D} \mu_{ki}^{x_i} (1 - \mu_{ki})^{(1-x_i)} \tag{7}$$

where x_i is the value of pixel i in an image, μ_{ki} represents the probability that pixel i in class k is black, and $\{\pi_1, \ldots, \pi_K\}$ are the mixing coefficients of classes in the data. We want to use this data set to classify new images of handwritten numbers '2', '3' and '4'.

Load the binary image data using the following commands:

[1] According to Wikipedia, the correlation is none if $|\rho| < 0.1$, small if $0.1 < |\rho| < 0.3$, medium if $0.3 < |\rho| < 0.5$ and strong if $|\rho| > 0.5$

```
N = 800; D = 28*28; X = uint8(zeros(N,D));
fid = fopen ('a012_images.dat', 'r');
for i = 1:N
 X(i,:)  = fread(fid, [D], 'uint8');
end;
status = fclose(fid);
```

1. Use `image();` to display some of the handwritten digits in the dataset (reshape vector `X(i,:)` into a square matrix and use '`BW_map=[1,1,1; 0,0,0]; colormap(BW_map);`'). Do you think it will be possible to classify them correctly with the right class prototypes?

2. Implement an EM-algorithm for the Bernoulli mixture model, as described in Bishop, §9.3.3 (or better: copy and modify the one from exercise 2). Again, use variable $K = 3$ for the number of classes and choose a priori equal mixing coefficients π_k. Initialize the μ_{ki}, the pixel means for each of the class prototypes, to random values in the interval $[0.25, 0.75]$. Let the algorithm run for at least 40 steps. To monitor progress and convergence, display the class images $\boldsymbol{\mu}$ after each step. Do this by mapping the pixel means to integer values in the range [0,255] and display as a greyscale image using `colormap(gray(255));`

   Note: The likelihood terms per class in eq.9.56 can become extremely small. To counter this problem you can work with the logarithms, or you can process an image pixel-by-pixel for all classes and renormalize after every few steps.

3. Run the EM-algorithm on the image data set. Describe process and result. Compare the effect of different random initializations.

4. If you have the algorithm running, then try to

   - start with more/less classes and see what happens (= describe & explain)
   - use the image labels in 'a012_labels.dat' to identify some misclassified images and see if you understand why
   - initialize the three classes with the true values and see what happens

   How could you improve the overall performance for this dataset?

   Note: The image labels can be loaded, using

   ```
   fid = fopen ('a012_labels.dat', 'r');
   Z = fread(fid, N, 'uint8');
   ```

5. Create a handwritten digit image of your own and see if it is classified correctly by the class prototypes $\boldsymbol{\mu}$ from your EM-algorithm.