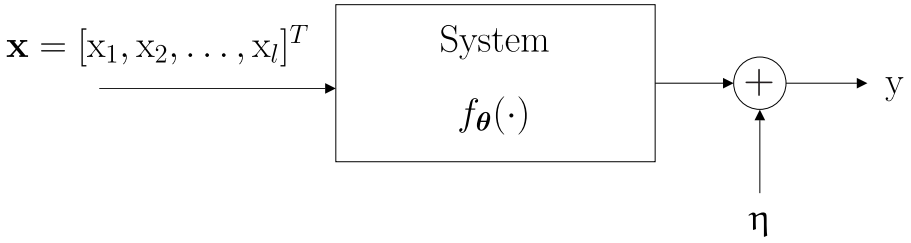
02471 Machine Learning for Signal Processing

Week 2: Parameter Estimation

This exercise is based on S. Theodoridis: Machine Learning, A Bayesian and Optimization Perspective 2nd edition, section(s) 3.1–3.3, 3.5, 3.8–3.11.

The objective of the exercise is to get acquainted with parameter estimation, the bias–variance dilemma and implications with key results.

We consider the following system $\mathrm{y} = f_{\boldsymbol{\theta}}(\mathbf{x}) + \eta$ (see figure).



For this particular exercise, we ignore the choice of $f_{\boldsymbol{\theta}}(\mathbf{x})$ and simply use linear regression models, and put focus on the estimation task, i.e. how to determine the parameters $\boldsymbol{\theta}$.

Overview

The exercise have the following structure:

2.1 is an exercise for linear regression, where we get acquainted with matrix manipulations and the bias-variance tradeoff.

2.2 will derive central results from ML section 3.5 and 3.9. This exercise will train you to get more comfortable with manipulating matrices and expectations, as well as show how some of the key results are derived.

2.3 is an exercise for ridge regression which is used to explore the bias-variance trade-off further.

Exercises can be carried out in any order, however it is beneficial to complete 2.1 first.

Notation

- Random variables are denoted with roman font, such as x, or θ, and their corresponding values (i.e. an observation) are denoted in italic, such as x or θ.
- Random vectors are denotes in bold roman font, such as $\mathbf{x}$ or $\boldsymbol{\theta}$, and their corresponding values are denoted in bold italic, such as $\boldsymbol{x}$ and $\boldsymbol{\theta}$.
- $J(\boldsymbol{\theta})$ is a cost function that we are seeking to minimize with respect to $\boldsymbol{\theta}$.
- θ denotes the parameters of a model. $\hat{\theta}$ denotes a point estimate of theta and θ_o denotes the optimal value for θ.

Code

The code can be found in the .m and .py files named in the same way as exercises, ie. the code for exercise 2.x.y is in the file 2_x_y.m (or .py).

For coding exercises that requires implementation we will usually write `complete this line` where the implementing should be done.

Solutions

The solution is provided for all derivation exercises, and often hints are provided at the end of the document. If you get stuck, take a look at the hints, and if you are still stuck, take a look in the solution to see the approach being taken. Then try to do it on your own.

Solutions are also provided for some coding exercises. If you get stuck, take a look at the solution, and then try to implement it on your own.

2.1 Linear Models

To keep focus on the estimation task, we will use the simple linear regression model. The linear regression model, can for the n'th data point be written as

$$y_n = \theta_0 + \theta_1 x_{n,1} + \cdots + \theta_l n, l + \eta = \theta_0 + \boldsymbol{\theta}^T \mathbf{x} + \eta$$

Exercise 2.1.1

If you are unfamiliar with Linear Regression and Least Squares, work through example 3.1, page 74 in the ML book.

Exercise 2.1.2

The parameters of the linear regression model is estimated by minimizing the cost function

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} (y_n - \boldsymbol{\theta}_b^T \boldsymbol{x}_n - \theta_0)^2$$

Let $\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta}_b^T & \theta_0 \end{bmatrix}^T$ ($\boldsymbol{\theta}$ is a column vector). Define a matrix X and vector $\boldsymbol{y}$ such that the cost function can be rewritten to

$$J(\boldsymbol{\theta}) = (\boldsymbol{y} - X\boldsymbol{\theta})^T (\boldsymbol{y} - X\boldsymbol{\theta})$$

Validate the expressions are identical e.g. by using a 3-dimensional $\boldsymbol{\theta}$ vector and 2 datapoints. Make sure to familiarize yourself with these types of rewritings.

Exercise 2.1.3

Show that

$$\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) = -2\mathrm{X}^T \boldsymbol{y} + 2\mathrm{X}^T X\boldsymbol{\theta}$$

We can now find the optimal set of parameters $\boldsymbol{\theta}$ by setting the derivative to zero, which then yields

$$(X^T X)\boldsymbol{\theta} = X^T \boldsymbol{y}$$

A few remarks to this solution.

- this is the prefered form for numerical solvers since these typically takes input on the form $A\boldsymbol{x} = \boldsymbol{b}$ where we solve for $\boldsymbol{x}$. We will use this later in the coding exercises.
- the cost function $J(\boldsymbol{\theta})$ is actually the squared error, and not the *mean* squared error (MSE), which is MSE $= \frac{1}{N}(\boldsymbol{y} - X\boldsymbol{\theta})^T(\boldsymbol{y} - X\boldsymbol{\theta})$. Note however, the factor $\frac{1}{N}$ vanishes when the derivative is set to zero, so the two functions have the same optimal solution.

Exercise 2.1.4

Inspect the code associated with this exercise (`ex2_1_4`). Run the code to create a training-set with a 2-dimensional input variable and a 1-dimensional output variable.

Identify the different parameters for the linear regression, and identify the line where the model is applied.

Exercise 2.1.5

Inspect the code associated with this exercise (`ex2_1_5`) and complete the missing lines.

Use the script to evaluate the training and test errors on independent sets. The sets are generated by the same true weight vector and the same noise variance, for two models.

- Identify the number of parameters for the two models
- Discuss the reason why the models perform best for different training set sizes.

Exercise 2.1.6

Compare the training and test errors *per example* as function of the size of the training set for the two models.

Exercise 2.1.7

Compare the values of the training and test errors for large training sets with the value of the noise variance.

2.2 Estimation

This exercise is purely hand–derivation, where we will show three results. It is important to reflect on the results that you derive, don't let yourself be bogged down in pure algebraic manipulations.

Exercise 2.2.1

We will work with the Mean-squared error estimation. Make sure to read sec. 3.9.1 in the book before carrying out this exercise.

We assume the general form of regression

$$\mathrm{y} = g(\mathbf{x}) + \eta$$

where η is zero–mean Gaussian noise with variance σ_η^2. If we have found an optimal estimate in the mean squared sense using data:

$$\hat{g}(\boldsymbol{x}) = \mathbb{E}[\mathrm{y}|\boldsymbol{x}]$$

Then,

$$\mathrm{MSE} = \mathbb{E}\big[(\mathrm{y} - \mathbb{E}[\mathrm{y}|\boldsymbol{x}])^2\big] = \sigma_\eta^2$$

Show the MSE relation. Consider what is the implications of the result?

Exercise 2.2.2 (This is a difficult exercise)

As a first step towards understanding the parameter estimation task, we will analyze the variance of an unbiased estimator. This analysis does not impose any constraints on the model as such, only on the estimator.

From sec 3.5.1 we have the following result that we want to derive.

Let $\hat{\boldsymbol{\theta}}_i, i = 1, 2, \ldots, m$, be unbiased estimators of a parameter vector $\boldsymbol{\theta}$, so that $\mathbb{E}\Big[\hat{\boldsymbol{\theta}}_i\Big] = \boldsymbol{\theta}, i = 1, \ldots, m$. Moreover, assume that the respective estimators are uncorrelated to each other and that all have the same (total) variance, $\sigma^2 = \mathbb{E}\Big[\Big(\hat{\boldsymbol{\theta}}_i - \boldsymbol{\theta}\Big)^T\Big(\hat{\boldsymbol{\theta}}_i - \boldsymbol{\theta}\Big)\Big]$. Then by averaging the estimates,

$$\hat{\boldsymbol{\theta}} = \frac{1}{m}\sum_{i=1}^{m}\hat{\boldsymbol{\theta}}_i$$

the new estimator has total variance:

$$\sigma_c^2 = \mathbb{E}\Big[(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta})^T(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta})\Big] = \frac{1}{m}\sigma^2$$

Show that, by averaging the estimators $\hat{\boldsymbol{\theta}}_i$, we obtain $\sigma_c^2 = \frac{1}{m}\sigma^2$.

Exercise 2.2.3 (This is a difficult exercise)

In this exercise we will provide theoretic proof that a biased estimator can obtain a lower MSE compared to an unbiased estimator. To simplify the derivations, we assume that the estimator is a real scalar instead of a vector.

We define the biased estimator as a scaled estimate of the unbiased estimator,

$$\hat{\theta}_b = (1 + \alpha)\hat{\theta}_u$$

where $\hat{\theta}_u$ is the unbiased estimator. We will show that $\hat{\theta}_b$ has a lower MSE than the corresponding unbiased estimator for certain choices of α, ie. $\mathrm{MSE}(\hat{\theta}_b) < \mathrm{MSE}(\hat{\theta}_u)$.

In addition, we also show that the norm of the biased estimator is lower than the corresponding unbiased estimator, ie. $|\hat{\theta}_b| < |\hat{\theta}_u|$.

The task is the following:

1. Compute $\mathrm{MSE}(\hat{\theta}_b)$ as a function of α, $\mathrm{MSE}(\hat{\theta}_u)$ and θ_0, where θ_0 are the true optimal parameter (ignore the value for α in this step).
2. Show that there exists an α such that $\mathrm{MSE}(\hat{\theta}_b) \leq \mathrm{MSE}(\hat{\theta}_u)$ by placing bounds on α using the expression derived in step 1 . What can you for instance say about the sign of α?
3. Conclude that $\mathrm{MSE}(\hat{\theta}_b) \leq \mathrm{MSE}(\hat{\theta}_u)$ implies that $|1 + \alpha| < 1$ and relate that to $|\hat{\theta}_b|$.

2.3 Bias-variance trade-off

Now we will consider a regularized version of the linear model (Ridge regression), where the parameters $\boldsymbol{\theta}$ are learned by minimization for a fixed value of λ:

$$L(\boldsymbol{\theta}, \lambda) = \sum_{n=1}^{N} (y_n - \boldsymbol{\theta}^T \boldsymbol{x}_n)^2 + \lambda \|\boldsymbol{\theta}\|^2$$

where it is assumed that the bias θ_0 is encoded as part of the $\boldsymbol{x}_n$ vector as in exercise 2.1.2.

Exercise 2.3.1

Use the results from exercise 2.1 to write up the solution for Ridge regression on the form $A\boldsymbol{x} = \boldsymbol{b}$.

Exercise 2.3.2

Inspect the code associated with this exercise (`ex2_3_2`) and complete the requested lines.

Consider why the function is not simultaneously minimized for $(\boldsymbol{\theta}, \lambda)$.

The training set averages generalization error in the point $\mathbf{x}$ can be written as

$$\mathbb{E}_{\mathcal{D}}\left[(f(\mathbf{x}; \mathcal{D}) - \mathbb{E}[\mathrm{y}|\mathbf{x}])^2\right] = \mathbb{E}_{\mathcal{D}}\left[(f(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[f(\mathbf{x}; \mathcal{D})])^2\right] + (\mathbb{E}_{\mathcal{D}}[f(\mathbf{x}; \mathcal{D})] - \mathbb{E}[\mathrm{y}|\mathbf{x}])^2$$

where $\mathbb{E}_{\mathcal{D}}[\cdot]$ is the expectation with respect to training sets.

Hence the average error is split into a variance part, quantifying the variation among solutions for different training sets and a bias part quantifying the performance of the average model with respect to best possible model.

The corresponding code snippet plots the relative amount of variance and bias for a linear model.

Exercise 2.3.3

Plot the average generalization error, the bias error, and the variance error for a large range of weight decay values. Comment on the two regimes where the generalization error stems from variance and bias respectively.

Exercise 2.3.4

Consider the following questions;

- What is the role of the weight decay in these two regimes?
- Which weight decay value would you recommend?

Exercise 2.3.5 (Optional)

In this exercise we will analyze bias-variance trade-off on the example of a simple physical experiment which is tossing the ball up in the air. We consider an experiment in which a ball has been tossed into the air (from the height of an averaged human, 168cm) and its position is

measured as a function of time. The data obtained from this hypothetical experiment is stored in 'h' array.

Inspect the code associated with this exercise (`ex2_3_5`), and answer a following questions:

1. From mechanics we know that the height should vary as the square of the time, so instead of a linear fit to the data we should use a quadratic one:

$$h(t) = h_0 + v_0 t + \frac{1}{2} a t^2$$

   Which degree of plotted polynomials would you choose? Explain why.

2. How does the bias and variance change with the degree of polynomials?

3. In which case do we see underfitting and in which overfitting. Explain why.

4. How does the model complexity change with the degree of polynomials?

HINTS

For exercise 2.2.1

Relate the result to your observations from 2.1.7

For exercise 2.2.2

To solve this problem, first carry out substitution of $\hat{\theta}$ in the expression $\sigma_c^2 = \mathbb{E}[(\hat{\theta}-\theta_o)^T(\hat{\theta}-\theta_o)]$, and then use the result that the estimators are uncorrelated, formally $\mathbb{E}[(\hat{\theta}_i - \theta_o)^T(\hat{\theta}_j - \theta_o)] = \sigma^2\delta_{ij}$, where $\delta_{ij} = 1$ when $i{=}j$ and zero otherwise.

For exercise 2.2.3

To show this, we first need an expression for the MSE of the biased estimator

$$\mathrm{MSE}(\hat{\theta}_b) = \mathbb{E}[(\hat{\theta}_b - \theta_o)^2]$$

Insert by substitution the expression for $\hat{\theta}_b$, and then add $\alpha\theta_o - \alpha\theta_o$ to obtain

$$\mathrm{MSE}(\hat{\theta}_b) = (1+\alpha)^2\mathrm{MSE}(\hat{\theta}_u) + \alpha^2\theta_o^2$$

Next we seek the solution for α so that

$$\mathrm{MSE}(\hat{\theta}_b) < \mathrm{MSE}(\hat{\theta}_u)$$

We solve this inequality for α first by substituting the expression for $\mathrm{MSE}(\hat{\theta}_b)$ and then divide both sides with $\theta_o^2 + \mathrm{MSE}(\hat{\theta}_u)$ and rearranging. Note you can assume $\alpha \neq 0$, since $\hat{\theta}_b = \hat{\theta}_u$ for $\alpha = 0$. This will lead to the solution

$$-\frac{2\mathrm{MSE}(\hat{\theta}_u)}{\mathrm{MSE}(\hat{\theta}_u) + \theta_o^2} < \alpha < 0$$

The final step is to verify that the norm of $\hat{\theta}_b < \hat{\theta}_u$. This result is obtained by bounding the lower bound to get the result $|1+\alpha| < 1$.

For exercise 2.3.1

The solution is written using sums in eq. (3.42) in the book.

02471 Machine Learning for Signal Processing

Week 3: Stochastic Processes and Linear Filtering

This exercise is based on S. Theodoridis: Machine Learning, A Bayesian and Optimization Perspective 2nd edition, section(s) 4.1–4.3, 4.5–4.7.

The objective of the exercise is to get acquainted with stochastic processes, linear filtering and the interplay between linear regression from last week and classical DSP filtering.

Overview

The exercise have the following structure

3.1 will derive the correlation functions for an auto-regressive process (AR). We will use this result to compare a theoretical expression to one that is estimated from data and inspect how they differ.

3.2 will now consider a signal that is modeled by an AR process, contaminated with noise, and use a linear filter to suppress the noise. We will use the Wiener filter solution (which is the linear filter that minimizes the mean squared error), and compare the theoretical optimal filter to a filter estimated from real samples.

3.3 will consider a real-world example of filtering on an audio signal, which can be approximated as an AR process. The audio is contaminated with noise, and a Wiener filter is then used to suppress the noise.

The exercises should be completed in order, but, if you would like to see the Wiener filter in action before deriving the results, consider starting with 3.3.

Notation

- There are two traditions of notation w.r.t. signals. In DSP literature, a signal is often denoted as $x(n)$ with n being the time index. In ML literature, the signal is often denoted x_n if it is an observed entity, and x_n or X_n if it is a random variable. In this exercise, we will use the ML notation.
- $r(k)$ refers to a correlation between two signal (called cross-correlation), or the correlation between the signal itself (called auto-correlation). The function $\hat{r}(k)$ denotes the estimate of $r(k)$ from data.
- $r_{xy}(k) = \mathbb{E}[x(n)y(n-k)] = \mathbb{E}[\mathrm{x}_n \mathrm{y}_{n-k}]$ refers to the cross-correlation between the two stochastic processes x_n and y_n. This is often in DSP literature denoted as $\gamma_{xy}(m)$.
- $r_x(k) = \mathbb{E}[x(n)x(n-k)] = \mathbb{E}[\mathrm{x}_n \mathrm{x}_{n-k}]$ refers to the autocorrelation of x_n. This function is also often written as $r_{xx}(k)$ or $\gamma_{xx}(m)$.

Code

The code can be found in the .m and .py files named in the same way as exercises, ie. the code for exercise 3.x.y is in the file 3_x_y.m (or .py).

For coding exercises that requires implementation we will usually write `complete this line` where the implementing should be done.

Solutions

The solution is provided for all derivation exercises, and often hints are provided at the end of the document. If you get stuck, take a look at the hints, and if you are still stuck, take a look in the solution to see the approach being taken. Then try to do it on your own.

Solutions are also provided for some coding exercises. If you get stuck, take a look at the solution, and then try to implement it on your own.

3.1 Correlation functions

The correlation between two signals measures the similarity between the signals and is widely used in signal processing applications. This exercise will introduce you to deriving correlation functions for stochastic signals. The correlation functions are required for the autocorrelation matrix and cross-correlation vector for the linear filter. We will also demonstrate correlation functions experimentally.

Make sure to read section 2.4.2–2.4.4 in the book before carrying out this exercise.

Exercise 3.1.1

Consider the signal $\mathrm{u}_n = c_1\mathrm{x}_n + c_2\mathrm{y}_{n-d}$, where c_1, c_2 are constants and $d > 0$ is a delay.

By using the definition of autocorrelation $r_u(k) = \mathbb{E}[\mathrm{u}_n\mathrm{u}_{n-k}]$ and cross-correlation $r_{uv}(k) = \mathbb{E}[\mathrm{u}_n\mathrm{v}_{n-k}]$, express $r_u(k)$ in terms of $r_x(k)$, $r_y(k)$, $r_{xy}(k)$, and the constants c_1, c_2 and d.

Exercise 3.1.2

Consider now two wide-sense stationary (WSS) random signals, x_n, y_n for which N_x and N_y samples have been recorded.

Complete the code in function `crosscor(x,y,k)` which estimates the cross-correlation function $r_{xy}(k)$ for $k = -K, -K+1, \cdots, 0, \cdots, K-1, K$, where $0 \leq K \leq \min(N_x, N_y) - 1$ using the biased, but asymptotically unbiased (which means the estimate is unbiased as $N \to \infty$) estimate:

$$\hat{r}_{xy}(k) = \frac{1}{N}\sum_{n=k}^{N-1} x_n y_{n-k}, \quad \text{for } k = 0, 1, \cdots, N-1$$

That the estimator is biased can be shown by evaluating $\mathbb{E}[\hat{r}_{xy}(k)] = \big(1 - \frac{k}{N}\big)r_{xy}(k)$ (not part of the curriculum).

Exercise 3.1.3

To verify the found analytical expressions, we generate some signals.

Inspect and run the code for this exercise (`ex3_1_3`). The code generates u_n, x_n and y_n according to the previous exercise, and plots the estimated autocorrelation function $r_u(k)$. Compare with a plot generated by using the expression found with estimates for the correlation functions in exercise 3.1.1.

Exercise 3.1.4 (This is a difficult exercise)

A great deal of signals can be adequately modeled as autoregressive (AR) processes, such as audio signals, and general time-series forecasting. We will consider the simplest case, a 1st order AR process, often denoted as AR(1), where the number refers to the amount of previous outputs remembered. There are two ways to define the AR(1) process, and it simply depends on whether the noise term (η_n) is isolated on the r.h.s, or the signal value u_n is isolated on the l.h.s. The two ways to define the AR process are:

$$\begin{aligned} u_n + a' u_{n-1} = \eta_n \qquad &\Leftrightarrow \\ u_n = a u_{n-1} + \eta_n \end{aligned}$$

where a is a real constant ($a \in \mathbb{R}, |a| < 1, a = -a'$), and η_n is a white noise sequence with variance σ_v^2. In this exercise, we consider the form $u_n = a u_{n-1} + \eta_n$.

According to the ML book, the analytical expression for $r_u(k)$ is

$$r_u(k) = \frac{a^{|k|}}{1 - a^2} r_\eta(0)$$

Prove the relation by inserting in the expression for u_n into $r_u(k) = \mathbb{E}[u_n u_{n-k}]$.

Exercise 3.1.5

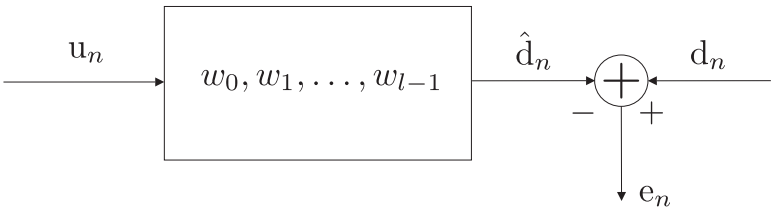
The corresponding code (`ex3_1_5`) generates signals u_n of lengths $N = 30$ and $N = 1000$, $k = N - 1$, with parameters $a = 0.1$ and $a = 0.9$. Compute and plot the estimated autocorrelation function in all 4 cases and compare with the theoretical autocorrelation function.

Exercise 3.1.6

As a final step, we will demonstrate the application of cross correlation functions on real data. We use the daily new confirmed the cases of COVID-19 in Denmark and cross-correlate with the cases on newly hospitalized patients. Use the associated code (`ex3_1_6`) to load the data, and modify the script to determine the most significant lags. What does this lag imply?

3.2 Wiener filter

Before completing this exercise, be sure to read section 4.5 in the book. Here the linear filter is depicted as



We will derive the expressions required to apply a Wiener filter and establish the theoretical performance of the filter. To this end, we will consider the example signal

$$u_n = s_n + \epsilon_n$$

where s_n is the source signal of interest and is modeled as an AR(1) process, $\mathrm{s}_n = a\mathrm{s}_{n-1} + \eta_n$, and ϵ_n is Gaussian white noise. The goal is to retrieve the clean signal s_n from the noisy signal u_n. Thus the desired signal is $\mathrm{d}_n = \mathrm{s}_n$. We will design a filter of length l with impulse response $\boldsymbol{w} = [w_0\, w_1\, \cdots\, w_{l-1}]^T$, that estimates s_n. To apply the linear filter, we require the two expressions: $r_u(k)$ and $r_{du}(k)$.

Exercise 3.2.1

Determine the analytical expression for $r_u(k)$.

Exercise 3.2.2

Determine the analytical expression for $r_{du}(k)$.

Exercise 3.2.3

Write the FIR Wiener filter solution, with filter length $l = 3$, on the form $(\Sigma_s + \Sigma_\epsilon)\boldsymbol{w} = \boldsymbol{r}_{du}$.

Exercise 3.2.4

Consider the case where $\sigma_\epsilon^2 \to 0$. What kind of filtering task are we then solving? (consider the filter figure). Determine the solution for $\boldsymbol{w}$ for $\sigma_\epsilon^2 \to 0$. Why is this a reasonable solution, i.e explain why the filter "chooses" to do what it does from the solution.

Exercise 3.2.5

Consider the case where $\sigma_\epsilon^2 \gg \sigma_\eta^2$. What kind of filtering task are we then solving? (consider the filter figure). Determine the solution for $\boldsymbol{w}$ when $\sigma_\epsilon^2 \gg \sigma_\eta^2$, and explain why this solution is a reasonable filtering solution.

Exercise 3.2.6

Now we consider a practical example with following coefficients: $l = 3$, $a = 0.6$, $\sigma_\eta^2 = 0.64$, and $\sigma_\epsilon^2 = 1$. The code snippets for this exercise (`ex3_2_6`) carries out the calculations for the filter coefficients. Inspect the code and verify the code aligns with your theoretical derived expressions.

Exercise 3.2.7

The error of the wiener filter, when using the optimal coefficients, is denoted as the *minimum mean squared error*, and is, for filter length l denoted MMSE_l. The expression is written as

$$\mathrm{MMSE}_l = \sigma_d^2 - \boldsymbol{r}_{du}^T \boldsymbol{\theta}_*$$

where $\boldsymbol{r}_{dx}$ is the cross-correlation vector between d_n and u_n, and $\boldsymbol{\theta}_*$ is the optimal solution, i.e. $\boldsymbol{\theta}_* = \boldsymbol{w}$. Implement the calculation of MMSE_l in the code (`ex3_2_7`).

Exercise 3.2.8

We will now estimate correlation functions from data which enables us to create the "plug-in" filter solution (when the correlation functions are estimated from data, we call the solution "plug-in"). Inspect and use the code (`ex3_2_8`) to generate data and obtain $r_u(k)$ and $r_{du}(k)$ for $k = 0, \pm 1, \pm 2$, and obtain the plug-in filter solution $\boldsymbol{w} = \hat{\Sigma}_u^{-1}\hat{\boldsymbol{r}}_{du}$ by substituting the true correlations functions with estimates. $\hat{\Sigma}_u$ is the estimated input correlation matrix and $\hat{\boldsymbol{r}}_{du}$ is the estimated cross-correlation vector (ref p. 136 in the book).

Exercise 3.2.9

What is the difference between the optimal (theoretical) filter solution and the plug-in filter. Compare the filters e.g., by evaluating the mean square error.

3.3 Noise removal using the Wiener filter

This exercise will demonstrate the application of a Wiener filter on real data, where there is an audio piece contaminated by noise, and then the Wiener filter is used to suppress the noise.

Exercise 3.3.1

Load the clean speech and noisy speech and listen to the sounds.

Exercise 3.3.2

Use the Wiener filter in the noise reduction setup, where the clean speech signal is used as desired signal, and conduct a rudimentary time-frequency analysis by inspecting the spectrograms of the signals. Try and listen to the result.

Exercise 3.3.3

Experiment with different filter lengths (the filter order). Is the result satisfactory? Why/Why not?

HINTS

Exercise 3.1.1

We have the definition, $r_u(k) = E[\mathrm{u}_n \mathrm{u}_{n-k}]$, solve by substitution, and take advantage of wide-sense stationarity (WSS). That means e.g. when $r_u(k)$ is a WSS process, the correlation function is a function only of the time-differences, so time indices can be shifted ie. $r_u(k) = E[\mathrm{u}_n \mathrm{u}_{n-k}] = E[\mathrm{u}_{n-m} \mathrm{u}_{n-k-m}]$. The final result is $r_u(k) = c_1^2 r_x(k) + c_1 c_2 (r_{xy}(d+k) + r_{xy}(d-k)) + c_2^2 r_y(k)$.

Exercise 3.1.4

Consider only $k > 0$, and insert the signal in the expression: $r_u(k) = \mathbb{E}[\mathrm{u}_n \mathrm{u}_{n-k}] = \mathbb{E}[(a\mathrm{u}_{n-1} + \eta_n)\mathrm{u}_{n-k}]$. Isolate $r_u(k)$ to generate a recursive formula. Once the formula is created, consider $k = 0$ to create the final expression. The final result is $r_u(k) = \frac{a^{|k|}}{1-a^2} r_\eta(0)$.

Exercise 3.2.1

Insert into the expression for $r_u(k) = \mathbb{E}[\mathrm{u}_n \mathrm{u}_{n-k}]$ (note, $\mathrm{u}_n = \mathrm{s}_n + \eta_n$) to obtain the final result, $r_u(k) = r_s(k) + r_\epsilon(k)$.

Exercise 3.2.2

Insert into the expression for $r_{du}(k) = \mathbb{E}[\mathrm{d}_n \mathrm{u}_{n-k}]$ (note, $\mathrm{d}_n = \mathrm{s}_n$) to obtain the final result, $r_{du}(k) = r_s(k)$.

Exercise 3.2.3

First use the fact that $r_\epsilon(0) = \sigma_\epsilon^2$ and otherwise $r_\epsilon(k) = 0$, $k \neq 0$, since it is a white noise signal. Then write out the input covariance matrix fully, and plug in the solutions from exercise 3.2.1 and 3.2.2 to obtain the solution

$$\left(\begin{bmatrix} r_s(0) & r_s(1) & r_s(2) \\ r_s(1) & r_s(0) & r_s(1) \\ r_s(2) & r_s(1) & r_s(0) \end{bmatrix} + \begin{bmatrix} \sigma_\epsilon^2 & 0 & 0 \\ 0 & \sigma_\epsilon^2 & 0 \\ 0 & 0 & \sigma_\epsilon^2 \end{bmatrix} \right) \mathbf{h} = \begin{bmatrix} r_s(0) \\ r_s(1) \\ r_s(2) \end{bmatrix}$$

02471 Machine Learning for Signal Processing

Week 4: Adaptive Linear Filtering with LMS

This exercise is based on S. Theodoridis: Machine Learning, A Bayesian and Optimization Perspective 2nd edition, section(s) 2.6, 5.1–5.4 (excluding 5.3.1), 5.6.

The objective of the exercise is to expand on the linear filtering exercise from week 3 and get acquainted with adaptive (online learning) filtering.

Overview

Adaptive filters are used in a wide range of applications where the characteristics of the signals or the system is not known a priori, or is known to be time varying.

In this exercise we will consider the adaptive finite impulse response (FIR) filter. The adaptation strategy will be both the Least-Mean-Square (LMS) and Normalized Least-Mean-Square (NLMS) algorithms.

The exercise have the following structure:

4.1 will review the wiener filter setup and find the solution given values for the correlation functions.

4.2 will consider the system identification setup, first applying a Wiener filter, and then investigate how the adaptive algorithms LMS and NLMS performs.

4.3 will consider an audio signal (from last week) contaminated with noise and use adaptive filtering to suppress the noise.

The exercises should be completed in order, but, if you would like to see the adaptive filter in action before deriving the results, consider starting with 4.3.

Notation

- $r(k)$ refers to a correlation between two signal (called cross-correlation), or the correlation between the signal itself (called auto-correlation). The function $\hat{r}(k)$ denotes the estimate of $r(k)$ from data.
- $r_{xy}(k) = \mathbb{E}[x(n)y(n-k)] = \mathbb{E}[\mathrm{x}_n \mathrm{y}_{n-k}]$ refers to the cross-correlation between the two stochastic processes $x(n)$ and $y(n)$. This is often in DSP literature denoted as $\gamma_{xy}(m)$.
- $r_x(k) = \mathbb{E}[x(n)x(n-k)] = \mathbb{E}[\mathrm{x}_n \mathrm{x}_{n-k}]$ refers to the autocorrelation of x_n. This function is also often written as $r_{xx}(k)$ or $\gamma_{xx}(m)$.

Code

The code can be found in the .m and .py files named in the same way as exercises, ie. the code for exercise 4.x.y is in the file 4_x_y.m (or .py).

For coding exercises that requires implementation we will usually write `complete this line` where the implementing should be done.

Solutions

The solution is provided for all derivation exercises, and often hints are provided at the end of the document. If you get stuck, take a look at the hints, and if you are still stuck, take a look in the solution to see the approach being taken. Then try to do it on your own.

Solutions are also provided for some coding exercises. If you get stuck, take a look at the solution, and then try to implement it on your own.

4.1 Wiener Filter review

In order to solve the subsequent exercises we will briefly review the Wiener filter. Make sure to read section 4.2 and 4.5 in the book before completing this exercise.

This problem considers a digital system for measurement and analysis of the analog signal $s_a(t)$. The measurement and A/D conversion is carried out with appropriate sampling frequency, so it can be assumed that the required spectral limitation of the input signal is appropriate.

During the conversion process, the signal is contaminated by a zero–mean noise sequence which is assumed to be uncorrelated with the source signal $s_a(t)$. The available digital signal is then given by

$$\mathrm{u}_n = \mathrm{s}_n + \mathrm{v}_n$$

where s_n is the sampled source signal $s_a(nT)$ with an appropriate sampling period $T > 0$, and v_n is the uncorrelated noise signal. We assume that all signals are stochastic and wide-sense stationary.

For entire exercise 4.1, we assume a FIR Wiener filter of length 2.

Exercise 4.1.1

Write up the normal equations for the Wiener filter using correlations functions for the input signal u_n and desired signal d_n on the form

$$\begin{bmatrix} r_?(?) & \cdots & r_?(?) \\ \vdots & \ddots & \vdots \\ r_?(?) & \cdots & r_?(?) \end{bmatrix} \begin{bmatrix} w_? \\ \vdots \\ w_? \end{bmatrix} = \begin{bmatrix} r_?(?) \\ \vdots \\ r_?(?) \end{bmatrix}$$

$$\Sigma_? \boldsymbol{w} = \boldsymbol{p}$$

Exercise 4.1.2

Determine expressions for the correlation functions $r_u(k)$ and $r_{du}(k)$ in terms of $r_s(k)$ and $r_v(k)$.

Exercise 4.1.3

Now assume that the follow correlations are estimated

k	0	1	2
$r_v(k)$	0.6	-0.1	-0.1
$r_s(k)$	0.6	-0.3	0.2

Determine the values of the filter coefficients.

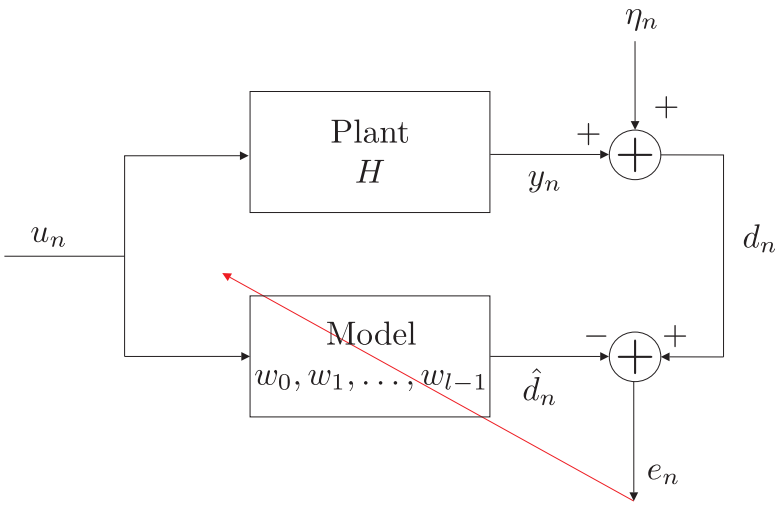
Exercise 4.1.4

We are now given the information that the variance (i.e. the power) of the desired signal is $\sigma_d^2 = 0.9$. Compute the minimum mean squared error for the filter.

4.2 System identification using a Wiener filter

In this exercise the system identification setup is considered. Make sure to read sec 4.7.2 and 5.5–5.5.1 in the book before completing this exercise.

The system identification setup is depicted on the following figure (note that the variables are written as realizations of signals);



The figure contains the following signals: the input signal to the systems, impulse response for the unknown system, impulse response for the filter to be trained, measurement noise (assumed to be white noise), measured output from the unknown system, output from the filter, residual noise from the filter, and a signal that is used as target for the filter.

The goal of this exercise is to derive a solution for the Wiener filter setup and an expression for the expected error for the Wiener filter setup. In other words, we want to apply the Wiener filter setup from last week to a system identification setup.

Exercise 4.2.1

Identify all the signals on the figure, i.e. determine what the different symbols signify (use the terminology used to describe Wiener Filter signals). What is the input sequence, desired sequence and error sequence? What is the filter length?

Exercise 4.2.2

Now we let our input signal (also sometimes called the excitation signal), be a white noise sequence with variance σ_u^2. Additionally, we assume that the η_n is uncorrelated with u_n. If we organize the input sequence in a vector, $\mathbf{u}_n = [\mathrm{u}_n, \cdots, \mathrm{u}_{n-1}, \mathrm{u}_{n-l+1}]^T$, we have $\mathrm{y}_n = \mathbf{u}_n * H = \mathbf{u}^T H$.

Show that $\mathbb{E}[(\mathbf{u}_n * H)\mathrm{u}_{n-k}] = h_k \sigma_{\mathrm{u}}^2$, where $H = [h_0, h_1, \cdots, h_{p-1}]^T$, assuming $p = l$. This result will be used in the next exercise.

Exercise 4.2.3

By solving the normal equations, as was done in exercise 4.1, we obtain the solution for the filter weights $\boldsymbol{w} = [w_0, w_1, \cdots, w_{l-1}]^T$.

Derive expressions for the input correlation matrix Σ_u and cross-correlation vector $\boldsymbol{p}$ by using the expressions for correlation functions, and show that $\boldsymbol{w}_* = H$ (assuming $p = l$).

Exercise 4.2.4 (This is a difficult exercise)

Derive a result for the mean squared error, $\mathbb{E}[\mathrm{e}_n^2]$, and show that the mean squared error can be expressed as $\mathbb{E}[\mathrm{e}_n^2] = \sigma_u^2 \|H - \boldsymbol{w}\|_2^2 + \sigma_\eta^2$ (assuming $p = l$). What is your interpretation of this result?

Exercise 4.2.5

Now we investigate the results obtains so far with data (realizations of the stochastic processes). We consider the setup where $H = \{1, 0.8, 0.2\}$, and let the excitation signal, u_n, be a realization of a white noise sequence of length 200. Furthermore, we assume $\eta_n = 0$ (no measurement noise).

Inspect and modify the code for this exercise (`ex4_2_5`) and use it to compute the plug-in Wiener filter solution. Consider the estimated weights compared to the true weights, e.g. by creating a curve with the size of the signal on the x-axis and error on y-axis.

Exercise 4.2.6

Now replicate the previous exercise, but compute the weights using the LMS and NLMS algorithm instead, by using the code for `ex4_2_6` as template. Consider the residuals and the convergence of the adaptive filter, e.g by plotting the error and the weights over time.

Experiment with different learning rates. Try to add noise to the output sequence, e.g. consider $\eta_n \neq 0$. Comment on the results.

Exercise 4.2.7

Now increase the excitation signal to a length of 500 and use the NLMS algorithm in a system identification setup. Load the impulse responses lpir.mat, hpir.mat, and bpir.mat and use these impulse responses as H and identify the impulse responses. Use the code `ex4_2_7` as template. Consider the residuals and the convergence of the adaptive filter, both in time domain and frequency domain. Comment on the results.

4.3 Adaptive filtering on audio

This exercise will demonstrate the application of an adaptive Wiener filter on real data, where there is piece of audio contaminated by noise, and then the filter is used to suppress the noise. It is a repetition from the audio exercise last week, but now with an adaptive filter.

Exercise 4.3.1

Inspect and use the code for `ex4_3_1`. Use the LMS algorithm in the noise reduction setup and experiment with different filter lengths (the filter order) and step sizes. Is the result satisfactory (Listen to the sounds)? Why/why not?

HINTS

Exercise 4.1.1

The formulas (4.5)–(4.6) and (4.43) from the ML book can be of help.

Exercise 4.1.3

The formula (4.9) is the formula to use.

Exercise 4.2.2

First replace the convolution operation with a sum. Secondly, the h_i can be taken out of expectation term, since the linear filter weights are considered as deterministic. Also remember u_n is a white noise sequence.

Exercise 4.2.3

The input sequence is u_n, and the desired signal is $\mathrm{d}_n = \mathrm{y}_n + \eta_n$, where y_n is the output of the unknown system (the plant), which is $\mathrm{y}_n = H * \mathbf{u}_n$.

To compute $\boldsymbol{p}$, we write out $r_{du}(k)$ completely, and solve using the normal equations.

Exercise 4.2.4

Solve by substitution and squaring e_n. Use the fact that convolution is distributive, i.e.

$$f * (g + h) = f * g + f * h$$

02471 Machine Learning for Signal Processing

Week 5: Adaptive Linear Filtering with RLS

This exercise is based on S. Theodoridis: Machine Learning, A Bayesian and Optimization Perspective 2nd edition, section(s) 5.12, 6.6–6.8.

The objective of this exercise is to expand on the linear filtering exercises from week 3 and 4, and get better acquainted with adaptive linear filtering and the RLS algorithm.

Overview

Adaptive filters are used in a wide range of applications where the characteristics of the signals or the system is not known a priori, or is known to be time varying.

In this exercise we will consider the adaptive finite impulse response (FIR) filter. The adaptation strategies will be Least-Mean-Square (LMS), Normalized Least-Mean-Square (NLMS) and Recursive Least Squares (RLS), with the primary focus on RLS.

The exercise have the following structure:

5.1 will derive the RLS algorithm and relate the equations to implementation.

5.2 will study the convergence behavior of the derived RLS algorithm in a stationary environment, and compare to NLMS.

5.3 will study the behavior of NLMS and RLS in a non-stationary environment, and get hands-on experience with tuning, and see under certain circumstances the NLMS performs better than RLS.

5.4 will continue to work on audio signals in the same manner as the previous two weeks. The audio is contaminated with noise, both stationary and non–stationary, and a FIR Wiener filter is then used to suppress the noise. Adaptive algorithms are used to train the Wiener filter.

Notation

- $J(\boldsymbol{\theta})$ is a cost function that we are seeking to minimize with respect to $\boldsymbol{\theta}$.
- In this exercise, the vector $\boldsymbol{p}$ is used, which is NOT related to the cross-correlation vector (that was in previous exercises also denoted $\boldsymbol{p}$).
- n indicates the time step as we collect data $(y_n, \boldsymbol{x}_n)$, where y_n is our target, or desired signal, and $\boldsymbol{x}_n$ is our filter input.

Code

The code can be found in the .m and .py files named in the same way as exercises, ie. the code for exercise 5.x.y is in the file 5_x_y.m (or .py).

For coding exercises that requires implementation we will usually write `complete this line` where the implementing should be done.

Solutions

The solution is provided for all derivation exercises, and often hints are provided at the end of the document. If you get stuck, take a look at the hints, and if you are still stuck, take a look in the solution to see the approach being taken. Then try to do it on your own.

Solutions are also provided for some coding exercises. If you get stuck, take a look at the solution, and then try to implement it on your own.

5.1 Derivation of the RLS algorithm

In this exercise we will derive the RLS algorithm step–by step. Be sure to read section 6.6 in the book before carrying out this exercise.

The starting point is the cost function defined as

$$J(\boldsymbol{\theta}, \beta, \lambda) = \sum_{i=0}^{n} \beta^{n-i}(y_i - \boldsymbol{\theta}^T \boldsymbol{x}_i)^2 + \lambda \beta^{n+1} \|\boldsymbol{\theta}\|^2$$

which is related to Ridge regression, hence the RLS algorithm minimizes a Ridge regression–like cost in a sequential manner as data points are collected.

Exercise 5.1.1

For which value of β will this correspond to Ridge regression?

Exercise 5.1.2

What happens with the regularization term (the last term) as we observe more data ?

Exercise 5.1.3

Find an expression for the derivative of $J(\boldsymbol{\theta}, \beta, \lambda)$ with respect to $\boldsymbol{\theta}$ and set the derivative to zero. This will readily give the expressions:

$$\begin{aligned}
\Phi_n &= \sum_{i=0}^{n} \beta^{n-i} \boldsymbol{x}_i \boldsymbol{x}_i^T + \lambda \beta^{n+1} I \\
\boldsymbol{p}_n &= \sum_{i=0}^{n} \beta^{n-i} \boldsymbol{x}_i y_i \\
\boldsymbol{\theta}_n &= \Phi_n^{-1} \boldsymbol{p}_n
\end{aligned}$$

Exercise 5.1.4

The goal is to find an iterative scheme for $\boldsymbol{\theta}_n$. Next step to achieve this goal is to create the time–iterative updates for Φ_n and $\boldsymbol{p}_n$. First write out the expressions for Φ_{n-1} and $\boldsymbol{p}_{n-1}$ by replacing n with $n-1$ in the above formulas. Next, manipulate the original expressions for Φ_n and $\boldsymbol{p}_n$ to get

$$\begin{aligned}
\Phi_n &= \beta \Phi_{n-1} + \boldsymbol{x}_n \boldsymbol{x}_n^T \\
\boldsymbol{p}_n &= \beta \boldsymbol{p}_{n-1} + \boldsymbol{x}_n y_n
\end{aligned}$$

Exercise 5.1.5

To arrive at the primary result (eq 6.44 and 6.45 in the book), we use Woodburry's inversion formula

$$(A + BD^{-1}C)^{-1} = A^{-1} - A^{-1}B(D + CA^{-1}B)^{-1}CA^{-1}$$

Use $A = \beta\Phi_{n-1}$, $B = \boldsymbol{x}_n$, $C = \boldsymbol{x}_n^T$, $D = 1$, to get

$$\Phi_n^{-1} = \beta^{-1}\Phi_{n-1}^{-1} - \beta^{-1}\boldsymbol{k}_n\boldsymbol{x}_n^T\Phi_{n-1}^{-1}$$
$$\boldsymbol{k}_n = \frac{\beta^{-1}\Phi_{n-1}^{-1}\boldsymbol{x}_n}{1 + \beta^{-1}\boldsymbol{x}_n^T\Phi_{n-1}^{-1}\boldsymbol{x}_n}$$

Exercise 5.1.6 (This is a difficult exercise)

Now we have time–iterative expressions for Φ_n and $\boldsymbol{p}_n$. Show by substituting the expressions into $\boldsymbol{\theta}_n = \Phi_n^{-1}\boldsymbol{p}_n$, and defining $e_n = y_n - \boldsymbol{\theta}_{n-1}^T\boldsymbol{x}_n$, that

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \boldsymbol{k}_n e_n$$

which concludes the derivation of the RLS algorithm.

Exercise 5.1.7

Inspect the code for RLS and relate the code and derived equations to algorithm 6.1 in the book. Try and implement the update formulas on your own (i.e. after inspection, erase the RLS code, and re-implement it).

5.2 Adaptive filtering simulations in stationary environment

The total error for a filter trained with an adaptive algorithm can be divided into three parts:

$$J = J_{\text{min}} + J_{\text{excess}} + J_{\text{lag}}$$

where J_{min} is the minimum achievable error (the error if the weights are optimal), J_{excess} is the error due to fluctuations around the true weights, and J_{lag} is the error induced by the filter lagging behind the correct solution. The expressions for $J_{\text{exc}} = J_{\text{excess}} + J_{\text{lag}}$ is summarized in table 6.1 in the book.

In this exercise we will simulate a stationary environment (hence $J_{\text{lag}} = 0$) and create convergence curves. This exercise corresponds to example 6.1, and problem 6.20 in the book, ignoring the APA algorithm. Make sure to read both the example 6.1 and problem 6.20 before completing this exercise.

Exercise 5.2.1

We consider the following model which result in a stationary environment:

$$y_n = \boldsymbol{\theta}_o^T\boldsymbol{x}_n + \eta_n$$

Explain why this results in a stationary environment.

Exercise 5.2.2

Run the code associated with this exercise. The code will reproduce figure 6.8 in the book. Inspect the code and identify the elements that generate the data and runs the algorithms.

Exercise 5.2.3

Investigate the different parameters of the algorithms (filter size, step size, forget factor, regularization), and study their effects on convergence speed and error floor. Discuss the results.

5.3 Adaptive filtering simulations in non–stationary environment

In this exercise we will simulate a non–stationary environment and create convergence curves. This exercise corresponds to example 6.2, and problem 6.21 in the book. Make sure to read both the example 6.2 and problem 6.21 before completing this exercise.

Exercise 5.3.1

We consider the following model which result in a non-stationary environment:

$$y_n = \boldsymbol{\theta}_{o,n}^T \boldsymbol{x}_n + \eta_n$$
$$\boldsymbol{\theta}_{o,n} = \alpha \boldsymbol{\theta}_{o,n-1} + \boldsymbol{\omega}_n$$

Explain why this results in a non–stationary environment.

Exercise 5.3.2

Run the code associated with this exercise. The code will reproduce figure 6.9 in the book. Inspect the code and identify the elements that generate the data and runs the algorithms.

Exercise 5.3.3

Investigate the different parameters of the algorithms (filter size, step size, forget factor, regularization), and study their effects on convergence speed and error floor. Make sure to reproduce figure 6.10 in the book. Discuss the results.

5.4 Adaptive filtering on audio

This exercise will demonstrate the application of an adaptive Wiener filter on real data, where there is piece of audio contaminated by noise, and then the adaptive Wiener filter is used to suppress the noise. It is a repetition and expansion of the audio exercise from last week, but now with RLS.

Exercise 5.4.1

Use and inspect the code for exercise 5.4.1. Determine the input signal, the output signal, and desired signal. Is the clean audio the filter output?

Try out RLS and experiment with the parameters of RLS. Listen to the sound, and experiment with the parameters for LMS, NLMS and RLS and different filter lengths. Is the results satisfactory (Listen to the sounds)? Discuss pros and cons of RLS compared to LMS and NLMS. Which algorithm performs best in terms of MSE? Which in terms of audible quality?

Exercise 5.4.2

Use and inspect the code for exercise 5.4.2. The filter will now handle a non-stationary noise source. Go through the same steps as the previous exercise and consider the same questions.

HINTS

For exercise 5.1.1

Ridge regression is treated in sec 3.8 and 6.6 in the book.

For exercise 5.1.6

You need to establish that $\Phi_n^{-1}\boldsymbol{x}_n = \boldsymbol{k}_n$, and use $e_n = y_n - \boldsymbol{x}^T\boldsymbol{\theta}_{n-1}$ in order to get the final update rule.

02471 Machine Learning for Signal Processing

Week 6: Sparsity-aware learning with ℓ_1

This exercise is based on S. Theodoridis: Machine Learning, A Bayesian and Optimization Perspective 2nd edition, section(s) 9.1–9.5, 9.9.

The objective of this exercise is to get better acquainted with learning sparse solutions using ℓ_1 regularization and touch upon compressed sensing.

Overview

The exercise have the following structure:

6.1 will prove important results we need for norms.

6.2 will manually calculate the solution for least squares using different norms as regularization.

6.3 will use the ℓ_1 norm to carry out compressed sensing and signal recovery for a toy example.

6.4 will demonstrate a use-case where we use compressed sensing to recover a multi-tone signal using fewer samples than usually required.

Notation

- $J(\boldsymbol{\theta})$ is a cost function that we are seeking to minimize with respect to $\boldsymbol{\theta}$.
- n indicates the time step as we collect data $(y_n, \boldsymbol{x}_n)$, where y_n is our target, or desired signal, and $\boldsymbol{x}_n$ is our filter input.
- $\mathcal{N}(\mu, \sigma^2)$ denotes a normal (Gaussian) distribution with mean value μ and variance σ^2.

Code

The code can be found in the .m and .py files named in the same way as exercises, ie. the code for exercise 6.x.y is in the file 6_x_y.m (or .py).

For coding exercises that requires implementation we will usually write `complete this line` where the implementing should be done.

Solutions

The solution is provided for all derivation exercises, and often hints are provided at the end of the document. If you get stuck, take a look at the hints, and if you are still stuck, take a look in the solution to see the approach being taken. Then try to do it on your own.

Solutions are also provided for some coding exercises. If you get stuck, take a look at the solution, and then try to implement it on your own.

6.1 Norms

In this exercise we will work with norms. Be sure to read 9.2 in the book before carrying out this exercise.

A function, $\|\cdot\|_p : \mathbb{R}^l \longmapsto [0,\infty)$ is a valid norm if the following properties hold:

1. $\|\boldsymbol{\theta}\|_p \geq 0, \quad \|\boldsymbol{\theta}\|_p = 0 \Leftrightarrow \boldsymbol{\theta} = \mathbf{0}$.
2. $\|\alpha\boldsymbol{\theta}\|_p = |\alpha|\|\boldsymbol{\theta}\|_p, \forall\alpha \in \mathbb{R}$.
3. $\|\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2\|_p \leq \|\boldsymbol{\theta}_1\|_p + \|\boldsymbol{\theta}_2\|_p$ (triangle inequality).

We will work with the ℓ_p norm, which for $p =]0;\infty[$ is defined as

$$\|\theta\|_p := \left(\sum_{i=1}^{l} |\theta_i|^p\right)^{1/p}$$

For $p \to 0$ we have

$$\|\theta\|_0 := \sum_{i=1}^{l} \chi_{(0,\infty)}(|\theta_i|)$$

$$\chi_{\mathcal{A}}(\tau) := \begin{cases} 1, & \text{if } \tau \in \mathcal{A} \\ 0, & \text{if } \tau \notin \mathcal{A} \end{cases}$$

Exercise 6.1.1

In the search for the sparsest solution, we want to use the $\|\cdot\|_0$ norm. However, this is not a true norm unfortunately. Show that $\|\cdot\|_p$ for $0 \leq p < 1$ is not a true norm.

6.2 The regularized least-squares solution

In this exercise we will work with the norm shrinkage solution. Be sure to skim sec. 9.3 before carrying out this exercise.

We will work with the regression task

$$\boldsymbol{y} = X\boldsymbol{\theta} + \boldsymbol{\eta}, \quad \boldsymbol{y} \in \mathbb{R}^N,\ X \in \mathbb{R}^{N\times l},\ \boldsymbol{\theta} \in \mathbb{R}^l,\ \boldsymbol{\eta} \in \mathbb{R}^N,$$

where we assume $\boldsymbol{\eta}$ is generated using a white noise sequence.

We will estimate the solution using the cost function

$$J(\boldsymbol{\theta}) = \|\boldsymbol{y} - X\boldsymbol{\theta}\|^2 + \lambda\|\boldsymbol{\theta}\|_\ell^2$$

where, for $\lambda = 0$, we obtain the least squares estimate, for $\lambda > 0$ and $\ell = 2$ we obtain Ridge regression, and for $\lambda > 0$ and $\ell = 1$ we obtain LASSO regression. If we assume the regressors have the property $X^TX = I$, the solutions to our regression problem are:

$$\hat{\boldsymbol{\theta}}_{LS} = X^T\boldsymbol{y}$$

$$\hat{\boldsymbol{\theta}}_R = \frac{1}{1+\lambda}\hat{\boldsymbol{\theta}}_{LS}$$

$$\hat{\theta}_{1,i} = \operatorname{sgn}\left(\hat{\theta}_{LS,i}\right)\left(|\hat{\theta}_{LS,i}| - \frac{\lambda}{2}\right)_+ \qquad i = 1, 2, \cdots, l$$

Exercise 6.2.1

The solutions for least squares regression and Ridge regression are

$$\hat{\boldsymbol{\theta}}_{\text{LS}} = \left(X^TX\right)^{-1}X^T\boldsymbol{y} = X^T\boldsymbol{y}$$

$$\hat{\boldsymbol{\theta}}_R = \left(X^TX + \lambda I\right)^{-1}X^T\boldsymbol{y}$$

Prove the result $\hat{\boldsymbol{\theta}}_R = \frac{1}{1+\lambda}\hat{\boldsymbol{\theta}}_{LS}$.

Exercise 6.2.2

For LASSO regression, taking the gradient of the cost function and equating to zero now yields a set (there are multiple solutions)

$$\mathbf{0} \in -2X^T\boldsymbol{y} + 2X^TX\boldsymbol{\theta} + \lambda\partial\|\boldsymbol{\theta}\|_1$$

If $X^TX = I$, we get

$$0 \in -\hat{\theta}_{\mathrm{LS},i} + \hat{\theta}_{1,i} + \frac{\lambda}{2}\partial\left|\hat{\theta}_{1,i}\right|, \quad \forall i$$

$$\partial|\theta| = \begin{cases} \{1\}, & \text{if } \theta > 0 \\ \{-1\}, & \text{if } \theta < 0 \\ [-1,1], & \text{if } \theta = 0 \end{cases}$$

Based on the information presented to you, show that

$$\hat{\theta}_{1,i} = \mathrm{sgn}\left(\hat{\theta}_{LS,i}\right)\left(|\hat{\theta}_{LS,i}| - \frac{\lambda}{2}\right)_+ \qquad i = 1, 2, \cdots, l$$

Exercise 6.2.3

Assume now we have a least-squares solution vector $\hat{\boldsymbol{\theta}}_{LS} = [0.7, -0.3, 0.1, -2]^T$. Compute the corresponding Ridge and LASSO solution for $\lambda = 1$.

Exercise 6.2.4

For what value of λ will Ridge regression result in the null vector? For what value of λ will LASSO regression result in the null vector?

6.3 Reconstruct a sparse vector using compressed sensing

In this exercise we will replicate example 9.5 and reconstruct a sparse vector $\boldsymbol{\theta} \in \mathbb{R}^l$, with its first components taking random values drawn from a normal distribution, $\mathcal{N}(0,1)$ and the rest being equal to zero.

To recover $\boldsymbol{\theta}$, we build a sensing matrix X with N rows having samples normally distributed as $\mathcal{N}(0, 1/N)$. Then we solve the regression problem using LASSO and Ridge regression and compare the results and recovery rate.

Exercise 6.3.1

Inspect and complete the code associated with this exercise. What is the parameters λ, N, and how many non-zeros does $\boldsymbol{\theta}$ have? What is the figure showing? Experiment with different parameters – can you make the ridge regression have equal performance to the LASSO?

Exercise 6.3.2

Inspect and complete the code associated with this exercise. The code repeats the previous experiment and counts the number of successful recoveries. How is this calculated? Use the number to estimate the recovery probability. What is the figure showing? Experiment with the different parameters, e.g. how many observations do you need to have a high recovery probability?

6.4 Signal recovery of a multi-tone signal

In this exercise we will build upon the previous exercise and construct a multi-tone signal and recover the signal in a compressed sensing situation. This could be e.g that the signal is transmitted through a communication channel, and now we need to estimate the tones that was transmitted.

The multi-tone signal is sampled as

$$x_n = \sum_{j=1}^{3} a_j \cos\Big(\frac{\pi}{2l}(2m_j - 1)n\Big), \quad n = 0, ..., l-1,$$

where $N = 30$, $l = 2^8$, $\boldsymbol{a} = [0.3, 1, 0.75]^T$ and $\boldsymbol{m} = [4, 10, 30]^T$.

Exercise 6.4.1

Inspect and complete the code associated with this exercise. What is the figure showing?

Exercise 6.4.2

To recover $\boldsymbol{x}$, we build a sensing matrix $X \in \mathbb{R}^{30\times 2^8}$ with entries drawn from a normal distribution $\mathcal{N}(0, 1/N)$ and use LASSO regression to recover the signal. This is not at all a realistic scenario in practice, and only used to demonstrate the theory works

Inspect and complete the code associated with this exercise.

What is the figure showing? Experiment with the different parameters.

Exercise 6.4.3

To make the situation more realistic, imagine now that we are doing random sampling of the signal (as opposed to uniform sampling with a sampling frequency abiding the nyquest rate.)

We will now use a sparse sensing matrix instead, where each row of X is a 1–sparse vector with the non-zero element being set to 1. Moreover, each column has at most one nonzero component. What operation will this sensing matrix carry out?

Inspect and complete the code associated with this exercise.

Empirically show (using the code) that $\boldsymbol{x}$ can be recovered exactly using such a sparse sensing matrix. Observe that the unknown $\boldsymbol{x}$ is sparse in the frequency domain and give an explanation why the recovery is successful with the specific sparse sensing matrix.

You may get different number of estimated components on each run. Complete the code to have a selection criteria for λ.

HINTS

Exercise 6.1.1

This is easiest proven by falsification, i.e. providing an example of vector(s) where the norm conditions does not hold.

Consider the cases $p = 0$ and $0 < p < 1$ separately.

Exercise 6.1.2

Property 1 and 2 are readily verified by substitution.

Exercise 6.2.1

Remember we assume $X^T X = I$.

Exercise 6.2.2

First try to arrive at the interim result by considering the different sign cases:

$$\hat{\theta}_{1,i} = \begin{cases} \hat{\theta}_{\mathrm{LS},i} - \frac{\lambda}{2}, & \text{if } \hat{\theta}_{1,i} > 0 \\ \hat{\theta}_{\mathrm{LS},i} + \frac{\lambda}{2}, & \text{if } \hat{\theta}_{1,i} < 0 \end{cases}$$

Exercise 6.2.3

Consider figure 9.3 in the book.

02471 Machine Learning for Signal Processing

Week 7: Sparse analysis models and time-frequency analysis

This exercise is based on S. Theodoridis: Machine Learning, A Bayesian and Optimization Perspective 2nd edition, section(s) 10.1–10.2, 10.5–10.6.

The exercise is divided into two major parts; part one (exercise 7.1–7.2) builds upon last week where we used built-in libraries to estimate the sparse solution to learn a parameter vector. This week we will implement two algorithms on our own, one that estimates the ℓ_0 solution and one that estimates the LASSO solution.

Part two (exercise 7.3–7.5) is concerned with time-frequency analysis, where we introduce signal representations using pre-determined dictionaries and conduct time-frequency analysis using the short-time Fourier transform. This serves as a warp-up to data-driven dictionary learning that will be treated next week.

Overview

7.1 implements the OMP algorithm and runs the algorithm on the data example from last week.

7.2 implements the IST algorithm and runs the algorithm on the data example from last week.

7.3 implements the Discrete Fourier Transform (DFT) as a matrix projection.

7.4 implements the Short-Time Fourier Transform (STFT).

7.5 applies a time-frequency analysis, and uses the extracted features to carry out musical genre classification.

Notation

There is nothing special with respect to notation in this exercise.

Code

The code can be found in the .m and .py files named in the same way as exercises, ie. the code for exercise 7.x.y is in the file 7_x_y.m (or .py).

For coding exercises that requires implementation we will usually write `complete this line` where the implementing should be done.

Solutions

The solution is provided for all derivation exercises, and often hints are provided at the end of the document. If you get stuck, take a look at the hints, and if you are still stuck, take a look in the solution to see the approach being taken. Then try to do it on your own.

Solutions are also provided for some coding exercises. If you get stuck, take a look at the solution, and then try to implement it on your own.

7.1 Orthogonal Matching Pursuit (OMP)

The Orthogonal Matching Pursuit (OMP) algorithm is a greedy algorithm and one of the simplest approaches on can take to identify a k–sparse solution. The algorithm can be terminated either when a k-sparse vector has been identified, or when the solution error is below a certain threshold.

In general, it is not guarantied that the OMP identifies the optimal k–sparse solution, just a k–sparse solution. There exists theory under which conditions the algorithm identifies such solution (see page 477), but we will not touch upon those.

Exercise 7.1.1

Make sure to read page 474–475 before carrying out this exercise.

We will implement OMP on our own. The algorithm is defined as Algorithm 10.1, p. 475 in the book. Inspect and complete the code that is associated with this exercise (`ex7_1_1`), and implement the missing pieces for the OMP algorithm.

The code runs the same problem as last week (ex 6.4.3). If your implementation is correct, the OMP algorithm should be able to identify weights that a similar to the built-in implementation.

Can you specify a way to estimate a good choice of k, now that OMP does not have a regularization parameter to use?

7.2 Iterative Shrinkage/thresholding (IST)

There are various approaches to identify the LASSO solution. One approach is to modify the OMP algorithm slightly, and obtain the so-called LARS-LASSO algorithm, which is also a greedy algorithm (since it is just modified OMP).

Another approach is to derive an iterative update rules, that closer resembles the type of algorithms we have already worked with.

Exercise 7.2.1

This exercise in essence replicates the steps in the book from equation (10.3)–(10.7). Make sure to read page 481–482 before carrying out this exercise.

First show that, if

$$J(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{y} - X\boldsymbol{\theta}\|_2^2$$

then the gradient descent update, $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu X^T \boldsymbol{e}^{(i-1)}$ is equivalent to minimizing the following problem

$$\boldsymbol{\theta}^{(i)} = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^l} \left\{ J\big(\boldsymbol{\theta}^{(i-1)}\big) + \big(\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)}\big)^T J'\big(\boldsymbol{\theta}^{(i-1)}\big) + \frac{1}{2\mu}\big\|\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)}\big\|_2^2 \right\}$$

Where $J'\big(\boldsymbol{\theta}^{(i-1)}\big)$ is the derivative of $J(\boldsymbol{\theta})$ evaluated at $\boldsymbol{\theta}^{(i-1)}$. Use this result to define the new minimization problem

$$\boldsymbol{\theta}^{(i)} = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^l} \left\{ J\big(\boldsymbol{\theta}^{(i-1)}\big) + \big(\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)}\big)^T J'\big(\boldsymbol{\theta}^{(i-1)}\big) + \frac{1}{2\mu}\big\|\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)}\big\|_2^2 + \lambda\|\boldsymbol{\theta}\|_1 \right\}$$

which, once minimized, arrives at the following update formulas:

$$\begin{aligned}\boldsymbol{e}^{(i-1)} &= \boldsymbol{y} - X\boldsymbol{\theta}^{(i-1)} \\ \tilde{\boldsymbol{\theta}} &= \boldsymbol{\theta}^{(i-1)} + \mu X^T \boldsymbol{e}^{(i-1)} \\ \boldsymbol{\theta}^{(i)} &= \operatorname{sign}\Big(\tilde{\boldsymbol{\theta}}\Big) \max\Big(\Big|\tilde{\boldsymbol{\theta}}\Big| - \lambda\mu, 0\Big)\end{aligned}$$

Exercise 7.2.2

Implement the derived formulas in the code associated with this exercise (`ex7_2_2`). The code runs the same problem as the previous exercise. If your implementation is correct, the IST algorithm should be able to identify weights that a similar to the built-in implementation.

Note: this naive approach is quite slow, and numerous optimizations has been suggested in the literature. However, this will give you a basic idea on how to derive an algorithm that can find sparse solutions.

7.3 Signal representation using the Discrete Fourier Transform (DFT)

As noted in the book section 9.4 and 10.5, as linear signal representation model can be thought of as

$$\begin{aligned}\tilde{\boldsymbol{s}} &= \Phi^H \boldsymbol{s} \qquad \text{analysis} \\ \boldsymbol{s} &= \Phi \tilde{\boldsymbol{s}} \qquad \text{synthetis}\end{aligned}$$

where $\boldsymbol{s}$ is the vector of raw samples, and $\tilde{\boldsymbol{s}}$ is the transformed vector. E.g. if the matrix Φ is chosen as the DFT matrix, $\tilde{\boldsymbol{s}}$ will contain the fourier coefficients (the signal in frequency domain).

The DFT matrix is just one choice of matrix, we can also choose others, e.g. based on the discrete cosine transform (DCT), discrete wavelets and so on.

In this exercise, we will derive the DFT matrix. The Discrete Fourier Transform (DFT) is described by the following equation

$$\tilde{x}_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}$$

Exercise 7.3.1

Determine how the matrix Φ^H should look like, so that the DFT can be computed like $\tilde{\boldsymbol{x}} = \Phi^H \boldsymbol{x}$.

Exercise 7.3.2

Inspect and run the code associated with this exercise (`ex7_3_2`). Identify the sample rate, the frequency of the signal?

Correct the code so that a DFT is computed by implementing the matrix you found in the previous exercise.

Note that, you can verify if your implementation is correct without looking in the solution, but instead identify if the figure plots an expected spectrum.

7.4 Short Time Fourier Transform

In this section we will implement the STFT and will explore a set of examples to show the differences between frequency analysis and time-frequency analysis.

We will start by implementing the STFT, which is going to use the DFT function which you just have implemented.

Please follow the instructions given in the comments and code the marked places (section 5). In this exercise we will use Hanning window, using the built-in MATLAB function hanning. The STFT function should return the complex STFT spectrogram (positive frequencies only). It should take three inputs:

- x_n - Time domain signal
- B - Block size (number of samples in each STFT block) must be even for perfect reconstruction.
- Bs - Block skip (number of samples between STFT blocks)

Your task will be to prepare a block of signal for DFT in each iteration in order to implement the STFT, where the signal needs to be multiplied by the window function:

$$X(n,k) = \sum_{m=-\infty}^{\infty} x_m w_{m-n} e^{-i2\pi km/N}$$

Exercise 7.4.1

Identify the main differences in the STFT formula compared to the DFT. Explain the differences in your own words, and

Exercise 7.4.2

What window function is being used in the code? Try and visualise the window function.

Exercise 7.4.3

Correct the code to implement the STFT. To see why STFT can be more useful than DFT we will create a signal and visualize its spectrum.

What is important to note here is that the two frequency spectra contains exactly the same two components, so it can only provide information about frequency even though both signals are significantly different.

When performing time-frequency analysis with the STFT, by windowing the signal at different local subsections of the original signal we have time localization of the spectrum computed at each block.

Exercise 7.4.4

Change the block size and block skip numbers and explore the effect of the parameters.

At what block size will you loose either time resolution or frequency resolution?

7.5 Audio signal processing demonstration

In this exercise we will extract information from audio using the STFT, compute features from the spectrogram, and the apply the features to a classifier.

The exercise serves as an example on how to carry out such task, so the primary learning from this tasks come from reading and understanding the code as well as to understand how the STFT can be used to extract features of time series data.

As a first step, we will check the STFT implementation by comparing with built-in methods. This will serve as a sanity check for your implementation as well as to get familiar with built-in methods.

The corresponding code block loads the audio data that we will use to perform music genre classification.

Exercise 7.5.1

Listen to the audio and see if you can couple what you see on the spectrogram with what you listen to.

Exercise 7.5.2

Next step is to extract some features based on audio spectrograms, which will enable us to carry out the classification. For each audio file we compute the Spectral Centroid feature.

The spectral centroid indicates at which frequency the energy of a spectrum is centered upon. This can be applied at each spectrum block as follows:

$$f_c = \frac{\sum_k S(k) f(k)}{\sum_k S(k)}$$

where $f(k)$ is the center frequency at each block (or simply the bin number), and $S(k)$ is the spectrum present at such bin. In order to combine all the bins in a single feature, we compute the mean spectral centroid over all the bins. We will add a second feature, called spectral rolloff which is the frequency threshold for which a percentage of the total spectral energy lies. The corresponding code block performs this feature extraction for each audio file.

IMPORTANT: You should substitute the DFT operation for the built-in FFT in the STFT. This will significantly speed-up the computation of features.

Next, we will build a k-nearest neighbor classifier that determines the musical genre based on the computed features. Here will carry out a simple approach, we choose k=5, and then classify each file using the remainder of the files to determine the classification.

Exercise 7.5.3

Which genres are more likely to be confused with each other using this approach? What is the classification accuracy?

HINTS

For exercise 7.3.2

The plot of the spectrum should indicate the primary frequency equal to 10Hz.

02471 Machine Learning for Signal Processing

Week 8: Dictionary learning and source separation

This exercise is based on S. Theodoridis: Machine Learning, A Bayesian and Optimization Perspective 2nd edition, section(s) 2,2, 2.5, 19.5.

The objective of this exercise is to understand the derivation behind ICA, and apply ICA as a source separation technique.

NOTE: in this exercise we are going to derive and implement ICA based on mutual information. The resulting algorithm is primarily for education purposes. If you use ICA for "production", use a library e.g the FastICA library, available for both Matlab and Python.

Overview

The exercise have the following structure:

8.1 will investigate why ICA cannot carry out separation for Gaussian signals, and additionally, provide some mathematical tools we need to derive the ICA update formula.

8.2 will derive the ICA update formula eq (19.59) in the book.

8.3 will implement the ICA update and run ICA on simulated data in order to get better acquainted with the performance of ICA.

8.4 are generating a simplified artificial signal resembling the recordings obtained in EEG and apply ICA to separate the sources.

8.5 is using ICA to perform source separation on audio.

You can in principle start with exercise 3, and just use the update formula eq. (19.60) from the book to implement ICA. Exercise 2 assumes that exercise 1 has been completed, and exercise 4 and 5 requires a working ICA algorithm (implemented in exercise 3).

Notation

- $J(\boldsymbol{\theta})$ is a cost function that we are seeking to minimize with respect to $\boldsymbol{\theta}$.
- n indicates the time step as we collect data $(y_n, \boldsymbol{x}_n)$, where y_n is our target, or desired signal, and $\boldsymbol{x}_n$ is our filter input.
- $\mathcal{N}(\mu, \sigma^2)$ denotes a normal (Gaussian) distribution with mean value μ and variance σ^2.
- $U(a, b)$ denotes a uniform distribution with a as lower limit and b as upper limit.
- $\mathbf{x} \sim$ distribution$(\cdot)$ means that $\mathbf{x}$ is a random variable that follows the associated distribution.

Code

The code can be found in the .m and .py files named in the same way as exercises, ie. the code for exercise 8.x.y is in the file 8_x_y.m (or .py).

For coding exercises that requires implementation we will usually write `complete this line` where the implementing should be done.

Solutions

The solution is provided for all derivation exercises, and often hints are provided at the end of the document. If you get stuck, take a look at the hints, and if you are still stuck, take a look in the solution to see the approach being taken. Then try to do it on your own.

Solutions are also provided for some coding exercises. If you get stuck, take a look at the solution, and then try to implement it on your own.

8.1 ICA and Gaussian signals

In this exercise we will provide proof of the claim that ICA cannot identify Gaussian distributed sources. Before completing this exercise, be sure to read section 2.2.5 and 19.5.1.

The ICA model is written as

$$\mathbf{x} = A\mathbf{s}$$

where we have observed a realization of $\mathbf{x}$ ($\mathbf{x}$ is a random vector), and we want to provide an estimate for the sources $\mathbf{s}$, denoted as $\mathbf{z}$, where $\mathbf{z} := \hat{\mathbf{s}}$.

Assuming that A is a square invertible matrix, we can write

$$\mathbf{z} = W\mathbf{x} = WA\mathbf{s}$$

where, if $W = A^{-1}$, we get perfect recovery of $\mathbf{s}$. Hence, ICA consists of estimating W. To do that, we assume that the underlying sources are statistically independent (that is, $p(\mathbf{s}) = \prod_{i=1}^{l} p(\mathrm{s}_i)$) and use that as our optimization criteria.

Now, consider the case where we have l Gaussian distributed sources with a mean of zero and diagonal covariance matrix $\Sigma_s = I$:

$$p_{\mathbf{s}}(\mathbf{s}) = \frac{1}{(2\pi)^{l/2}} \exp\left(-\frac{\|\mathbf{s}\|^2}{2}\right)$$

We will now investigate how that affects the observed signals $\mathbf{x}$.

In essence, the ICA model assumes that $\mathbf{x}$ is a linear transformation of $\mathbf{s}$. That can be considered as a change random variables in order to obtain $p_x(\mathbf{x})$ from $p_s(\mathbf{x})$. We need the theorem of change of random variables (eq. 2.45), defined as (where $\mathbf{x}$, and $\mathbf{y}$ is the random variables):

$$\begin{aligned} p_{\mathbf{y}}(\mathbf{y}) &= \frac{p_{\mathbf{x}}(\mathbf{x})}{|\det(J(\mathbf{y}, \mathbf{x}))|} \\ \mathbf{y} &= f(\mathbf{x}) \\ \mathbf{x} &= f^{-1}(\mathbf{y}) \end{aligned}$$

$|\det(\cdot)|$ is the determinant of a matrix, and $J(\mathbf{y}, \mathbf{x})$ is the Jacobian matrix defined as

$$J(\mathbf{y}, \mathbf{x}) = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_l} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_l} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_l}{\partial x_1} & \frac{\partial y_l}{\partial x_2} & \cdots & \frac{\partial y_l}{\partial x_l} \end{bmatrix}.$$

Exercise 8.1.1

Assume for simplicity that A is orthogonal (that implies $A^T A = I$, and $\det(A) = \pm 1$). Apply the change of variable theorem, we need to identify $f(\cdot)$, $f^{-1}(\cdot)$ and derive $J(\mathbf{x}, \mathbf{s})$. Identify these functions given the ICA model $\mathbf{x} = A\mathbf{s}$, and show that $J(\mathbf{x}, \mathbf{s}) = A$.

Exercise 8.1.2

Use the obtained results and plug into the change of random variables theorem to show that the distribution for $p_{\mathbf{x}}(\mathbf{x})$ is

$$p_{\mathbf{x}}(\mathbf{x}) = \frac{1}{(2\pi)^{l/2}} \exp\left(-\frac{\|\mathbf{x}\|^2}{2}\right)$$

Now we can se that $p_x(\mathbf{x})$ has the exact same distribution as $p_s(\mathbf{s})$.

The implication is that the observation variables and the source variables have the same distribution under the ICA model, hence, the ICA model is not able to separate Gaussian sources.

8.2 Derivation of ICA based on mutual information

In this exercise, we will derive the ICA update formula:

$$W^{(i)} = W^{(i-1)} + \mu_i \big(I - \mathbb{E}\big[\boldsymbol{\phi}(\mathbf{z})\mathbf{z}^T\big]\big)(W^{(i-1)})^{-T}$$

Before completing this exercise, be sure to read section 2.5 and 19.5.4. We are going through the exact steps from eq. (19.51)–(19.59) in the book. I feel the book is too brief here, so this exercise is a step-by-step guide on how we go from mutual information to an update formula.

This can be a rather difficult and time-consuming exercise depending on your mathematical background, so consider using this as a tutorial on the ICA derivation (especially if hand-deriving is meaningless for your learning).

The starting point of ICA based on mutual information is the minimization of the expression

$$\hat{W} = \arg\min_{W} I(\mathbf{z})$$

Where $\mathbf{z}$ is the estimate of the sources $\mathbf{s}$ we want to discover.

The function $I(\mathbf{z})$ is called the mutual information (since it measure how much random variables have in common), and $H(\mathbf{z})$ is called the differential entropy. The entropy is inspired from physics, and measures the amount of randomness of the distribution $p(\mathbf{z})$.

Our goal is to minimize $I(\mathbf{z})$ w.r.t. W, since if e.g. $I(\mathrm{z}_1, \mathrm{z}_2) = 0$ the variables z_1 and z_2 have no mutual information and are statistically independent (observing z_1 reveals no information about z_2 and vice versa).

We want to apply the ICA model to $\mathbf{z}$, compute the gradients of $I(\mathbf{z})$ w.r.t. W (the matrix we want to estimate). Then we can apply gradient descent estimate W. In order do that, we need to rewrite $I(\mathbf{z})$ as we cannot optimize it in the current form.

Exercise 8.2.1

First we rewrite $I(\mathbf{z})$ to a more convenient form. The mutual information in the two-dimensional case is (eq. 2.158):

$$I(\mathrm{x};\mathrm{y}) := \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x,y)\ln\frac{p(x,y)}{p(x)p(y)}dxdy$$

Show that this expression can be rewritten to $I(\mathrm{x};\mathrm{y}) = -H(\mathrm{x};\mathrm{y}) + H(\mathrm{x}) + H(\mathrm{y})$ which, for the l–dimensional case leads to

$$\begin{aligned} I(\mathbf{z}) &= -H(\mathbf{z}) + \sum_{i=1}^{l} H(\mathrm{z}_i) \\ H(\mathbf{z}) &= -\int_{-\infty}^{\infty} p(\mathbf{z})\ln p(\mathbf{z})d\mathbf{z} \end{aligned}$$

Exercise 8.2.2 (This is a difficult exercise)

We need to have W in our expression as well in order to optimize w.r.t that, so we need to perform additional rewrites. Using the result from the change of variables in the previous exercise (8.1), we have $p_{\mathbf{z}}(\mathbf{z}) = p_{\mathbf{x}}(\mathbf{x})/|\det(W)|$. Show that, by applying these rewrites, the mutual information can be written as

$$I(\mathbf{z}) = -H(\mathbf{x}) - \ln|\det(W)| + \sum_{i=1}^{l} H(\mathrm{z}_i)$$

Exercise 8.2.3

Using the previously found results, argue or show that our original minimization problem is equivalent to the following maximization problem

$$\begin{aligned} \hat{W} &= \arg\min_{W} I(\mathbf{z}) \qquad \Rightarrow \\ \hat{W} &= \arg\max_{W}\ J(W) \\ J(W) &:= \ln|\det(W)| + \mathbb{E}\left[\sum_{i=1}^{l}\ln p_i(\mathrm{z}_i)\right] \end{aligned}$$

We now have the cost function in a form we need, so now the focus can turn to deriving the gradient.

Exercise 8.2.4 (Optional)

We will take the derivative w.r.t $J(W)$ term by term. Use the following formula:

$$\frac{d}{dW}\det(W) = W^{-T}\det(W), \quad W^{-T} := (W^{-1})^T$$

and the chain rule to show that

$$\frac{d}{dW}\ln|\det(W)| = W^{-T}$$

Exercise 8.2.5 (Optional)

Now focusing on the second term of the cost function $J(W)$, we can, since $\ln x$ is integrable for $x > 0$, interchange the expectations with the derivative[1], that is

$$\frac{d}{dx}\mathbb{E}[\ln x] = \mathbb{E}\left[\frac{d}{dx}\ln x\right]$$

Argue that, in our case, we have $x > 0$. Then, compute $\frac{d}{dW}\ln p_i(\mathrm{z}_i)$ and show (by using the chain rule) that we obtain

$$\frac{d}{dW}\ln p_i(\mathrm{z}_i) = \frac{1}{p(\mathrm{z}_i)}\frac{\partial p_i(\mathrm{z}_i)}{\partial \mathrm{z}_i}\frac{d\mathrm{z}_i}{dW}$$

Exercise 8.2.6 (Optional - this is a difficult exercise)

We need to compute two gradients now, $\frac{\partial p_i(\mathrm{z}_i)}{\partial \mathrm{z}_i}$ and $\frac{d\mathrm{z}_i}{dW_{k,m}}$, which we will do seperately. To compute $\frac{d\mathrm{z}_i}{dW_{k,m}}$, we need an expression for z_i that includes $W_{k,m}$. In exercise 8.1, we showed that, under the ICA model, $\mathrm{z}_i = W_{i,:}^T\mathbf{x}$ where $W_{i,:}$ denotes the i'th row in W. Use this result to obtain

$$\frac{d\mathrm{z}_i}{dW_{k,m}} = \mathrm{x}_m, \quad i = k, \text{ otherwise zero}$$

If we define a short-hand notation:

$$\boldsymbol{\phi}(\mathbf{z}) := \begin{bmatrix}\frac{p_1'(\mathrm{z}_1)}{p(\mathrm{z}_1)} & \frac{p_2'(\mathrm{z}_2)}{p(\mathrm{z}_2)} & \cdots & \frac{p_l'(\mathrm{z}_l)}{p(\mathrm{z}_l)}\end{bmatrix}^T, \qquad p_i'(\mathrm{z}_i) := \frac{\partial p_i(\mathrm{z}_i)}{\partial \mathrm{z}_i}$$

show that result can be rewritten in matrix notation as

$$\frac{d}{dW}\sum_{i=1}^{l}\ln p_i(\mathrm{z}_i) = \boldsymbol{\phi}(\mathbf{z})\mathbf{x}^T$$

Exercise 8.2.7

By substiting the derivatives we have found in the previous two exercises, we obtain the final result for the gradient

$$\frac{d}{dW}J(W) = W^{-T} - \mathbb{E}\big[\boldsymbol{\phi}(\mathbf{z})\mathbf{x}^T\big]$$

Use the result and apply gradient descent to obtain the following update rule

$$W^{(i)} = W^{(i-1)} + \mu_i\big(I - \mathbb{E}\big[\boldsymbol{\phi}(\mathbf{z})\mathbf{z}^T\big]\big)(W^{(i-1)})^{-T}$$

As is noted in the book, the use of the natural gradient descent leads to replacing $(W^{(i-1)})^{-T}$ with $W^{(i-1)}$, and this is the update formula we will use in the subsequent exercise, i.e

$$W^{(i)} = W^{(i-1)} + \mu_i\big(I - \mathbb{E}\big[\boldsymbol{\phi}(\mathbf{z})\mathbf{z}^T\big]\big)W^{(i-1)}$$

A missing piece in applying the algorithm is that we need to choose an expression for $\boldsymbol{\phi}(\mathbf{z})$. From the book "Independent component analysis" by Erkki Oja, Juha Karhunen and Aapo Hyvarinen, two choices are suggested; for super-Gaussian distributed $\mathbf{z}$, use $\boldsymbol{\phi}(\mathbf{z}) = 2\tanh(\mathbf{z})$ and for sub-Gaussian distributed $\mathbf{z}$, use $\boldsymbol{\phi}(\mathbf{z}) = \mathbf{z} - \tanh(\mathbf{z})$.

[1] https://en.wikipedia.org/wiki/Leibniz_integral_rule#Measure_theory_statement

8.3 ICA for simulated data

In this exercise we are going to test ICA on a simple dataset and compare to PCA.

Consider the following generative model

$$\begin{aligned} \mathrm{s}_1 &\sim U(0,1) \\ \mathrm{s}_2 &\sim U(0,1) \\ \mathbf{x} &= A \cdot [\mathrm{s}_1\ \mathrm{s}_2]^T \end{aligned}$$

where A is a mixing matrix, $A \in \mathbb{R}^{2\times 2}$.

Exercise 8.3.1

Use the code associated with this exercise. Modify the code to use a mixing matrix as $A = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}$, generate a X matrix of size $2 \times N$, and visualize the results. Apply PCA using a built-in/library function on X, and create a scatter plot with principal component 1 and 2 as the axes.

Exercise 8.3.2

Use the code associated with this exercise (`ICA.m` for matlab). There is a template implementation for ICA. Relate the code to the optimization algorithm we have previously derived and complete the implementation.

Exercise 8.3.3

Use ICA to unmix X, and recover the original source. Try and use both a sub Gaussian and a super Gaussian activation function.The algorithm can only recover the source signal for one of the activation functions. Was the "good" activation function the one you expected? What happens if you don't zero-mean the data first?

Exercise 8.3.4

Consider different mixing matrices A and/or different distributions for s. E.g. you found in the first exercise that ICA cannot recover multiple Gaussian sources, try to produce that result. Also check for one Gaussian distributed source.

8.4 ICA as source separation on simulated EEG

Electroencephalography (EEG) is an electrophysiological monitoring method to record electrical activity of the brain. Generally the data collection is a non-invasive procedure that takes over a period of time readings from electrodes placed at specific locations on the scalp. These electrodes measure voltage fluctuations that are caused by brain activity. There are multiple applications based on EEG data such as epilepsia diagnosis or brain diseases or disorders, among others.

In this exercise we are generating a simplified artificial signal resembling the recordings obtained in EEG. In EEG, multiple electrodes are placed at different locations on the scalp and thus capturing a combination of waves generated by brain activity.

Run the code associated with this exercise. A sinusoidal signal is generated alongside with a spike signal similar to the ones present when a subject blinks. The EEG reading is obtained from 3 locations (parietal, central and frontal), modeled as having different amplitudes. Use ICA to remove the blink signal. In order to perform the blink removal, identify and remove the component obtained after applying ICA and use the mixing matrix to project back the signals.

Try to add a third source (beta) that is highest at central location. What is the mixing matrix being used? Play with different signals to get comfortable with ICA and how the time series are unmixed.

8.5 ICA as source separation on audio

In this exercise we will work on a set of mixed audio recordings which have been created using instantaneous mixing. Inspect and run the code associated with this exercise.

The audio file contains two channels. Listen to the two channels individually, mixed and after separation. Consider to create your own mixture of audio to run ICA on. Can you create a mixture where ICA is not able to unmix the signals?

Load the `mixture_convolutive.wav` and listen to it. Run ICA on this sound file? Does it still work well? Why/Why not?

Note: Convolutive blind source seperation has been developed to handle the above case, see e.g. "A survey of convolutive blind source separation methods", by Michael Syskind Pedersen, Jan Larsen, Ulrik Kjems, and Lucas C. Parra.

HINTS

Exercise 8.1.1

Since the model is $\mathbf{x} = A\mathbf{s}$, we get $\mathbf{s} = A^T\mathbf{x}$ (since we assumed $A^{-1} = A^T$), hence $\mathbf{x} = f(\mathbf{s}) = A\mathbf{s}$, and $\mathbf{s} = f^{-1}(\mathbf{x}) = A^T\mathbf{x}$.

Exercise 8.1.2

By using the change of random variable theorem you should get the intermediate result

$$p_{\mathbf{x}}(\mathbf{x}) = \frac{1}{(2\pi)^{l/2}} \exp\left(-\frac{\|A^T\mathbf{x}\|^2}{2}\right) |\det(A^T)|$$

You will also need the rule (Appendix A.1) $\det(A^{-1}) = \frac{1}{\det(A)}$ to obtain the result.

Exercise 8.2.2

The entropy can be written as $H(\mathbf{z}) = -\mathbb{E}[\ln p(\mathbf{z})]$ (show this by the definition of the expectation).

By applying the logarithm to both side and then the expectation we get

$$\mathbb{E}[\ln p_{\mathbf{z}}(\mathbf{z})] = \mathbb{E}[\ln p_{\mathbf{x}}(\mathbf{x})] - \ln|\det(W)|$$

and then this leads to

$$-H(\mathbf{z}) = -H(\mathbf{x}) - \ln|\det(W)|$$

02471 Machine Learning for Signal Processing

Week 9: Bayesian inference and the EM algorithm

This exercise is based on S. Theodoridis: Machine Learning, A Bayesian and Optimization Perspective 2nd edition, section(s) 3.10–3.11, 12.1–12.2, 12.4–12.5.

The objective of this exercise is to establish the connection between the way we have performed parameter estimation so far and the probabilistic framework. Previously we have designed different cost functions based on what "made sense", but it turns out we can arrive to the same cost functions in a probabilistic framework, thus establishing a unified and more interpretable way to make modeling decisions.

Overview

The exercise have the following structure:

9.1 will establish the connection between the cost functions we have used so far, maximum likelihood and Bayes inference. This exercise is mostly for students who haven't had the Bayesian machine learning course. If you had this course, you should probably skip this exercise.

9.2 will derive updates for EM using Bayes linear regression as the example model.

9.3 will experiment with the derived formulas, and get acquainted with how Bayes linear regression performs.

You can choose to start or end with exercise 9.3.

Notation

- $J(\boldsymbol{\theta})$ is a cost function that we are seeking to minimize with respect to $\boldsymbol{\theta}$.
- $\mathcal{N}(\mu, \sigma^2)$ denotes a normal (Gaussian) distribution with mean value μ and variance σ^2.
- $U(a, b)$ denotes a uniform distribution with a as lower limit and b as upper limit.
- $\mathbf{x} \sim$ distribution$(\cdot)$ means that $\mathbf{x}$ is a random variable that follows the associated distribution.
- $\|\cdot\|_\ell$ denotes the ℓ-norm. If ℓ is absent, we are implicitly referring to the $\ell = 2$ norm, i.e. $\|\cdot\| = \|\cdot\|_2$.

Code

The code can be found in the .m and .py files named in the same way as exercises, ie. the code for exercise 9.x.y is in the file 9_x_y.m (or .py).

For coding exercises that requires implementation we will usually write `complete this line` where the implementing should be done.

Solutions

The solution is provided for all derivation exercises, and often hints are provided at the end of the document. If you get stuck, take a look at the hints, and if you are still stuck, take a look in the solution to see the approach being taken. Then try to do it on your own.

Solutions are also provided for some coding exercises. If you get stuck, take a look at the solution, and then try to implement it on your own.

9.1 Cost functions, Maximum Likelihood and Bayesian Inference

Consider the following model for the supervised learning problem

$$\mathrm{y} = f(\mathbf{x}, \boldsymbol{\theta}) + \eta$$

where, when we have observations we have $(y_n, \boldsymbol{x}_n)$, $n = 1, \cdots, N$, and $\hat{y}_n = f(\boldsymbol{x}_n, \boldsymbol{\theta})$ is the model that maps input data to output data (or, in machine learning terms, maps features to the target variable). In that case, we can write the model as

$$y = f(X, \boldsymbol{\theta}) + \eta$$

To shed light on the relationships between cost functions used in previous weeks, maximum likelihood and Bayes' inference, we consider Bayes' formula, where we condition on an observed dataset X:

$$p(\boldsymbol{\theta}|\boldsymbol{y}, X) = \frac{p(\boldsymbol{y}, \boldsymbol{\theta}|X)}{p(\boldsymbol{y}|X)}$$

To avoid notational clutter, we drop the condition on X and rewrite the joint probability $p(\boldsymbol{y}, \boldsymbol{\theta}|X)$ using the product rule (eq. 2.12):

$$p(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{y})}$$

The numerator is often called the complete likelihood. Taking the log of the above expression we get

$$\ln p(\boldsymbol{\theta}|\boldsymbol{y}) = \ln p(\boldsymbol{y}|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta}) - \ln p(\boldsymbol{y})$$

To apply Bayes' formula for machine learning problems, we need to specify two distributions (models); $p(\boldsymbol{y}|\boldsymbol{\theta})$, which maps parameters and data together, and $p(\boldsymbol{\theta})$ which encodes our prior beliefs. If we consider the above as an optimization problem and only consider the terms that include $\boldsymbol{\theta}$, we get

$$\ln p(\boldsymbol{\theta}|\boldsymbol{y}) \propto \ln p(\boldsymbol{y}|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta}), \quad \text{also called:}$$
$$\log \text{posterior} \propto \log \text{likelihood} + \log \text{prior}$$

In this exercise we will derive three results; 1) the connection between the log likelihood using a Gaussian distribution and the mean-squared error cost function $J(\boldsymbol{\theta})$, 2) the connection between the MAP estimate when using a Gaussian prior and Ridge regression, and 3) the connection between the MAP estimate when using a Laplacian prior and LASSO.

Exercise 9.1.1

First we will show that the mean squared error cost function corresponds to having a improper flat prior (then $\ln p(\boldsymbol{\theta})$ is constant) and a normal likelihood. The multivariate normal

distribution (or multivariate Gaussian distribution) is

$$p(\boldsymbol{y}|\boldsymbol{\theta};\boldsymbol{\mu}_y,\Sigma_y) = \frac{1}{(2\pi)^{N/2}|\Sigma_y|^{1/2}}\exp\left(-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu}_y)^T\Sigma_y^{-1}(\boldsymbol{y}-\boldsymbol{\mu}_y)\right)$$

As can be seen, the two terms $\boldsymbol{\mu}_y$ and Σ_y enters the equation as parameters. To make the formula applicable we need expressions for these two terms.

Show that $\boldsymbol{\mu}_y = f(X,\boldsymbol{\theta}) + \mathbb{E}[\eta]$. In the sequel we assume we have zero-mean noise ($\mathbb{E}[\eta] = 0$). Show that $\Sigma_y = \Sigma_\eta$. Finally, by substituting the found expressions into the multivariate Gaussian distribution, show, using the rules $\ln ab = \ln a + \ln b$ and $\ln a^b = b\ln a$, that

$$\ln p(\boldsymbol{y}|\boldsymbol{\theta};\boldsymbol{\mu}_y,\Sigma_y) = -\frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\Sigma_\eta| - \frac{1}{2}(\boldsymbol{y}-f(X,\boldsymbol{\theta}))^T\Sigma_\eta^{-1}(\boldsymbol{y}-f(X,\boldsymbol{\theta}))$$

Exercise 9.1.2

Now we assume that we have collected N samples, and that the noise is statistically independent from sample to sample (e.g white noise), so we write $\Sigma_\eta = \sigma^2 I$. In that case, we have $|\Sigma_\eta| = |\sigma^2 I| = \sigma^{2N}$, and $\Sigma_\eta^{-1} = (\sigma^2 I)^{-1} = \frac{1}{\sigma^2}I$. Under this assumption, show that

$$\ln p(\boldsymbol{y}|\boldsymbol{\theta};\boldsymbol{\mu}_y,\Sigma_y) = -\frac{N}{2}\ln(2\pi) - \frac{N}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}\|\boldsymbol{y}-f(X,\boldsymbol{\theta})\|^2$$

Exercise 9.1.3

Next, we consider the MAP optimization problem (maximize the posterior) with this particular likelihood w.r.t. $\boldsymbol{\theta}$, hence we can disregard all terms that is not dependent on $\boldsymbol{\theta}$. Show that this optimization problem can be written as

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} -\frac{1}{2\sigma^2}\|\boldsymbol{y}-f(X,\boldsymbol{\theta})\|^2 + \ln p(\boldsymbol{\theta})$$

and then demonstrate that, if the flat improper prior ($p(\boldsymbol{\theta})$ is constant), we get

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{y}-f(X,\boldsymbol{\theta})\|^2$$

Thus we can see that maximizing the derived log-likelihood corresponds to optimizing the mean-squared error, and hence we have readily established the link and now we can understand that the likelihood is essentially the model for our noise. We now have a framework in place, where, if we desire another noise model than e..g white noise, we simply go through the previous steps to derive the cost function.

Additionally, we can see that log-prior that added to the cost function, based on what we have learned so far, could serve as a regularization term.

Exercise 9.1.4

Let us now consider the case where we use a normal prior, with $\boldsymbol{\theta} \in \mathbb{R}^K$, and the weights are statistically independent ($\Sigma_\theta = \sigma_\theta^2 I$) and zero-mean. Formally, $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta};0,\sigma^2 I)$. Reuse the expression from the previous exercise to get

$$\ln p(\boldsymbol{\theta};\mathbf{0},\sigma_\theta^2 I) = -\frac{K}{2}\ln(2\pi) - \frac{K}{2}\ln\sigma_\theta^2 - \frac{1}{2\sigma_\theta^2}\|\boldsymbol{\theta}\|^2$$

Combine this result with the log-likelihood we derived in the last exercise to obtain

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2\sigma^2}\|\boldsymbol{y} - f(X, \boldsymbol{\theta})\|^2 + \frac{1}{2\sigma_\theta^2}\|\boldsymbol{\theta}\|^2$$

Finally, reparameterize with $\sigma_\theta^2 = \frac{\sigma^2}{\lambda} \Leftrightarrow \lambda = \frac{\sigma^2}{\sigma_\theta}$, to get

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{y} - f(X, \boldsymbol{\theta})\|^2 + \lambda\|\boldsymbol{\theta}\|^2$$

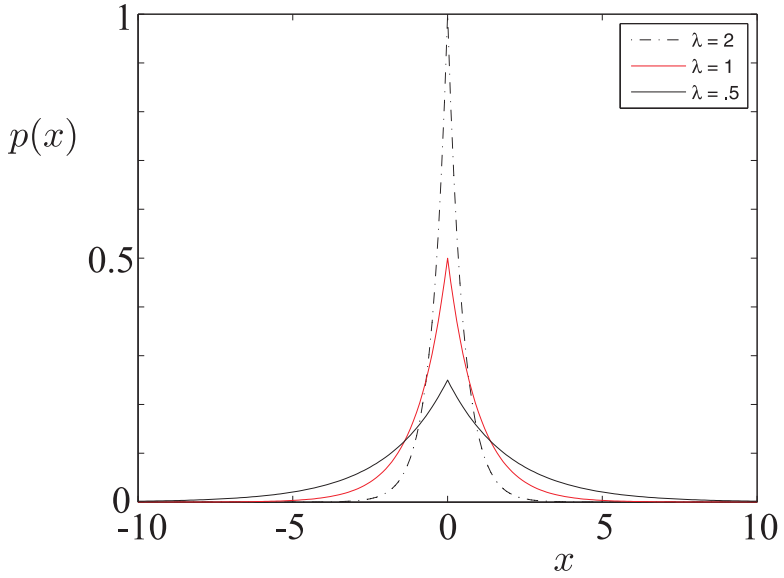
We recognize this expression as the Ridge regression if we use a linear model for $f(X, \boldsymbol{\theta})$, hence we have developed this cost function from a probabilistic perspective and shown that Ridge regression is indeed the same as maximizing the log-posterior if we use a normal log-likelihood and a normal log-prior.

Exercise 9.1.5

Let us now consider the log to the univariate Laplacian distribution, defined as

$$p(x|\mu, b) = \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right)$$

And looks like (on the figure λ corresponds to b)



Show that the log to the Laplace distribution is

$$\ln p(x|\mu, b) = -\ln 2 - \ln b - \frac{1}{b}|x - \mu|$$

Exercise 9.1.6

Let us now consider a weight vector $\boldsymbol{\theta}$ of length l. If we assume each θ_i follows a zero-mean Laplacian distribution, and the individual weights are statistical independent, show that we get

$$\ln p(\boldsymbol{\theta}|0, b) = -l\ln 2 - l\ln b - \frac{1}{b}\|\boldsymbol{\theta}\|_1$$

Exercise 9.1.7

Combine your finding with the previous results, and obtain the compete log-likelihood $\boldsymbol{\theta}$

$$\ln p(\boldsymbol{\theta}, \boldsymbol{y}|X) = -\frac{N}{2}\ln 2\pi - \frac{N}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}\|(\boldsymbol{y} - f(X,\boldsymbol{\theta}))\|^2 - l\ln 2 - l\ln b - \frac{1}{b}\|\boldsymbol{\theta}\|_1$$

From Bayes formula, we know, given a dataset X, optimizing $\ln p(\boldsymbol{\theta}, \boldsymbol{y}|X)$ is the same as optimizing $\ln p(\boldsymbol{\theta}|\boldsymbol{y}, X)$. Disregarding all terms not related to $\boldsymbol{\theta}$ we get

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \arg\max_{\boldsymbol{\theta}} -\frac{1}{2\sigma^2}\|(\boldsymbol{y} - f(X,\boldsymbol{\theta}))\|^2 - \frac{1}{b}\|\boldsymbol{\theta}\|_1 \\ &= \arg\min_{\boldsymbol{\theta}} \frac{1}{2\sigma^2}\|(\boldsymbol{y} - f(X,\boldsymbol{\theta}))\|^2 + \frac{1}{b}\|\boldsymbol{\theta}\|_1\end{aligned}$$

If we reparameterize with $b = 2\sigma^2/\lambda$ we get the Lasso cost function and we can see that LASSO corresponds to having normal likelihood with i.id.d samples and univariate Laplace prior on $\boldsymbol{\theta}$.

9.2 Derive EM updates for Bayesian linear regression

Before carrying out this exercise, be sure to read ML section 12.4–12.5. In this exercise, we are going to derive then updates necessary for using EM for the general linear regression model.

Using the EM algorithm requires the following:

1. Specification of the complete log likelihood, $\ln p(\boldsymbol{y}, \boldsymbol{\theta})$ (The model)
2. Derive $\mathbb{E}\big[\ln p(\boldsymbol{y}, \boldsymbol{\theta}; \boldsymbol{\xi}^{(j)})\big]$ and denote this term $\mathcal{Q}(\boldsymbol{\xi}, \boldsymbol{\xi}^{(j)})$.
3. Maximize $\mathcal{Q}(\boldsymbol{\xi}, \boldsymbol{\xi}^{(j)})$ in order to get $\boldsymbol{\xi}^{(j+1)}$

We consider a normal likelihood and a normal prior. These expressions (or the logs of them) have already been derived in the earlier exercise. We assume we have N observations and our parameter vector $\boldsymbol{\theta}$ is of length K. Additionally, we choose to simplify the covariance matrices and assume white noise, i.e. $\Sigma_\eta = \beta^{-1}I$, and that the parameters $\boldsymbol{\theta}$ are independent in the prior $\Sigma_{\boldsymbol{\theta}} = \alpha^{-1}I$, where α and β are the precision parameters, i.e $\alpha = \frac{1}{\sigma_{\boldsymbol{\theta}}^2}$ and $\beta = \frac{1}{\sigma_\eta^2}$.

We have already derived expressions for these in the previous exercise. Using these results we get:

$$\begin{aligned}\ln p(\boldsymbol{y}, \boldsymbol{\theta}|\alpha, \beta) &= \ln p(\boldsymbol{y}|\boldsymbol{\theta}; \Phi\boldsymbol{\theta}, \beta) + \ln p(\boldsymbol{\theta}; \mathbf{0}, \alpha) \\ &= -\frac{N}{2}\ln(2\pi) + \frac{N}{2}\ln\beta - \frac{\beta}{2}\|\boldsymbol{y} - \Phi\boldsymbol{\theta}\|^2 - \frac{K}{2}\ln(2\pi) + \frac{K}{2}\ln\alpha - \frac{\alpha}{2}\|\boldsymbol{\theta}\|^2 \\ &= -\frac{1}{2}(N+K)\ln(2\pi) + \frac{N}{2}\ln\beta - \frac{\beta}{2}\|\boldsymbol{y} - \Phi\boldsymbol{\theta}\|^2 + \frac{K}{2}\ln\alpha - \frac{\alpha}{2}\|\boldsymbol{\theta}\|^2\end{aligned}$$

To employ EM, we need to define $\boldsymbol{\xi}$, the parameter vector that is learned by EM, and derive $\mathbb{E}\big[\ln p(\boldsymbol{y}, \boldsymbol{\theta}; \boldsymbol{\xi}^{(j)})\big]$. We can readily see that $\boldsymbol{\xi} = [\alpha\ \beta]^T$. With respect to the expectation, terms that does not contain $\boldsymbol{\theta}$ are considered constants w.r.t. the expectation, we have two terms for which we need to evaluate, namely $\|\boldsymbol{y} - \Phi\boldsymbol{\theta}\|^2$ and $\|\boldsymbol{\theta}\|^2$.

Exercise 9.2.1

We will start deriving $A := \mathbb{E}[\|\theta\|^2]$.

To compute the expectation we use the following rule $\theta^T\theta = \text{trace}(\theta\theta^T)$ (since θ is a vector), and use that trace is a linear operator i.e. $\text{trace}(A+B) = \text{trace}(A) + \text{trace}(B)$, and $\mathbb{E}[\text{trace}(A)] = \text{trace}(\mathbb{E}[A])$. Then show that

$$A := \mathbb{E}\big[\|\theta\|^2\big] = \text{trace}\big(\mathbb{E}\big[\theta\theta^T\big]\big)$$

We recognize $\mathbb{E}\big[\theta\theta^T\big]$ as the structure of the correlation matrix eq (2.33), hence we have, at step j in the EM

$$\begin{aligned}\mathbb{E}\big[\theta\theta^T\big] &= \text{Cov}(\theta) + \mathbb{E}[\theta]\mathbb{E}\big[\theta^T\big] \\ &= \Sigma^{(j)}_{\theta|y} + \boldsymbol{\mu}^{(j)}_{\theta|y}\boldsymbol{\mu}^{(j)T}_{\theta|y}\end{aligned}$$

Verify this step by yourself. Finally, show that by substitution we obtain

$$A = \text{trace}\Big(\Sigma^{(j)}_{\theta|y}\Big) + \|\boldsymbol{\mu}^{(j)}_{\theta|y}\|^2$$

Hence, we have now found an expression for one of the terms needed. To compute this term, we need expressions for $\boldsymbol{\mu}^{(j)}_{\theta|y}$ and $\Sigma^{(j)}_{\theta|y}$ which we will compute in a moment. First we will derive the expectation of the second term.

Exercise 9.2.2

The other term we need to evaluate is $\|\boldsymbol{y} - \Phi\boldsymbol{\theta}\|^2$. To evaluate, we again use the properties of the trace operator, show that by doing so we can perform the following rewrite

$$\|\boldsymbol{y} - \Phi\boldsymbol{\theta}\|^2 = \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T\Phi\boldsymbol{\theta} + \text{trace}\big(\Phi\boldsymbol{\theta}\boldsymbol{\theta}^T\Phi^T\big)$$

Exercise 9.2.3

To proceed we now take the expectation to $\|\boldsymbol{y} - \Phi\boldsymbol{\theta}\|^2$, where $\boldsymbol{\theta}$ is the only random variable, and again using that $\text{trace}(\cdot)$ is a linear operator show that we get

$$\begin{aligned}B :&= \mathbb{E}\big[\|\boldsymbol{y} - \Phi\theta\|^2\big] \\ &= \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T\Phi\mathbb{E}[\theta] + \text{trace}\big(\Phi\mathbb{E}\big[\theta\theta^T\big]\Phi^T\big)\end{aligned}$$

We have already found the expressions for $\mathbb{E}[\theta]$ and $\mathbb{E}\big[\theta\theta^T\big]$ earlier, so show that by substitution, and again using that $\text{trace}(\cdot)$ is a linear operator we get

$$B = \|\boldsymbol{y} - \Phi\boldsymbol{\mu}^{(j)}_{\theta|y}\|^2 + \text{trace}\Big(\Phi\Sigma^{(j)}_{\theta|y}\Phi^T\Big)$$

We have now arrived at the solution for B, so we can now put it all together and obtain

$$\mathcal{Q}\big(\alpha,\beta;\alpha^{(j)},\beta^{(j)}\big) = \mathbb{E}[\ln p(\boldsymbol{y},\boldsymbol{\theta}|\alpha,\beta)] = -\frac{1}{2}(N+K)\ln(2\pi) + \frac{N}{2}\ln\beta - \frac{\beta}{2}B + \frac{K}{2}\ln\alpha - \frac{\alpha}{2}A$$

Exercise 9.2.4

The missing piece for the expectation step is to be able to evaluate A and B. To get those, we evaluate the posterior. Since we are considering a normal likelihood and a normal prior, we can use sec 12.10.4 in the book. Here we have expressions for how to specify the sufficient statistics fo the posterior. From eq. (12.138) and eq. (12.139) we have[1], if

$$\begin{aligned} p(\boldsymbol{z}) &= \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_z, \Sigma_z) \\ p(\boldsymbol{t}|\boldsymbol{z}) &= \mathcal{N}(\boldsymbol{t}|\boldsymbol{z}; A\boldsymbol{z}, \Sigma_{t|z}) \end{aligned}$$

then the posterior is

$$\begin{aligned} p(\boldsymbol{z}|\boldsymbol{t}) &= \mathcal{N}(\boldsymbol{z}|\boldsymbol{t}; \boldsymbol{\mu}_{z|t}, \Sigma_{z|t}) \\ \boldsymbol{\mu}_{z|t} &= \boldsymbol{\mu}_z + \Sigma_{z|t} A^T \Sigma_{t|z}^{-1} (\boldsymbol{t} - A\boldsymbol{\mu}_z) \\ \Sigma_{z|t} &= (\Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A)^{-1} \end{aligned}$$

Then we get the following expressions

$$\begin{aligned} \boldsymbol{\mu}_{\theta|y} &= \beta \Sigma_{\theta|y} \Phi^T \boldsymbol{y} \\ \Sigma_{\theta|y} &= (\alpha I + \beta \Phi^T \Phi)^{-1} \end{aligned}$$

Since we can now completely compute $\mathcal{Q}\big(\alpha, \beta; \alpha^{(j)}, \beta^{(j)}\big)$, we have completed the derivation of the E step.

Exercise 9.2.5

The next step is the maximization step where we maximize $\mathcal{Q}\big(\alpha, \beta; \alpha^{(j)}, \beta^{(j)}\big)$ w.r.t. α and β. We can derive closed form updates for these by taking the derivative to $\mathcal{Q}\big(\alpha, \beta; \alpha^{(j)}, \beta^{(j)}\big)$ w.r.t. α and β respectively and set those to zero. Show that the closed form solutions are

$$\begin{aligned} \alpha &= \frac{K}{A} \\ \beta &= \frac{N}{B} \end{aligned}$$

Hence, the update equations will be $\alpha^{j+1} = K/A$ and $\beta^{j+1} = N/B$.

We have now completed the derivation of the update formulas for the EM algorithm for Bayesian linear regression.

9.3 Bayesian linear regression on real data

In this exercise we will try out the formulas in two settings: 1) we know all the parameters (which will not be the case in real-life), and see how that affects the regressions, 2) learn the parameters from data using the EM.

Note, since our model corresponds to Ridge regression an from the first exercise we had $\lambda = \frac{\sigma_\eta}{\sigma_\theta}$, EM is effectively learning the regularization parameter without performing cross-validation!

[1] The book is inconsistent and write $\boldsymbol{u}_{z|t}$ in two different ways in sec 12.5 and 12.10. Both are correct though!. We use the form used in 12.5

Exercise 9.3.1

This example implements example 12.1 from the book, be sure to read that example before carrying out this exercise. Inspect and run the code corresponding to this exercise. Relate the code to the formulas you have derived, and try and reproduce all three plots on Figure 12.1.

Exercise 9.3.2

This example implements example 12.2 from the book, be sure to read that example before carrying out this exercise. Inspect and run the code corresponding to this exercise. Relate the code to the formulas you have derived. Play with different parameters for the noise terms and data set. You can also try to carry out ridge regression using cross-validation, and see to what extent there is correspondence with the estimate of λ by EM and cross-validation.

HINTS

Exercise 9.1.4

In our case, we have $\boldsymbol{z} := \boldsymbol{\theta}$, $\boldsymbol{\mu}_z := \mathbf{0}$, $\boldsymbol{t} := \boldsymbol{y}$, $\Sigma_z^{-1} := \alpha I$, $\boldsymbol{t} := \boldsymbol{y}$, $A := \Phi$, and $\Sigma_{t|z}^{-1} := \beta I$.

02471 Machine Learning for Signal Processing

Week 10: State-space models – the Hidden Markov Model

This exercise is based on S. Theodoridis: Machine Learning, A Bayesian and Optimization Perspective 2nd edition, section(s) 16.4–16.5. The objective of this exercise is to get better understanding of the Hidden Markov Model (HMM), both how to conduct inference and how to learn parameters. It will also use the EM algorithm from last week.

A complete derivation and implementation of the learning algorithm is beyond one exercise, so we will carry out the general principles and then use library functions for the training of the HMM. It should be stressed that the library functions we are using are learning parameters using the *Baum–Welch* algorithm, and we will derive the first parts of that algorithm as well.

Overview

The exercise have the following structure:

10.1 will demonstrate how to compute probabilities for the HMM where the model parameters are known. We will derive the first part (the forward pass) of the efficient *forward-backward* algorithm (also called the Sum-Product algorithm for the general case).

10.2 will formulate the likelihood for HMM and derive the EM update rules.

10.3 will train a simple HMM model on simulated data and investigate the stability of the EM parameter estimation.

10.4 will present a naive and simplified example using COVID data, where we use HMM to estimate the epidemic state based on the number of observed hospitalizations. *Disclaimer: these results should in no way be seen as a realistic epidemic estimation, but only serve as a toy example.*

The exercises can be carried out in any order.

Notation

- The book is unfortunately not consistent with the notation of random variables in the section describing HMMs (section 16.5). E.g, in sec 16.5, the book writes $P(x = 1)$, but in section 2.3, the (correct) notation $P(\mathrm{x} = 1)$ is used. The exercise uses the books notation. For the purpose of this exercise, do not be too concerned about this unfortunate notation ambiguity, the important aspect is the understanding of the model and derivations.

Code

The code can be found in the .m and .py files named in the same way as exercises, ie. the code for exercise 10.x.y is in the file 10_x_y.m (or .py).

For coding exercises that requires implementation we will usually write `complete this line` where the implementing should be done.

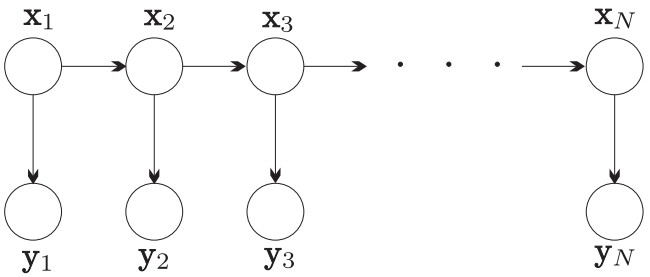
Solutions

The solution is provided for all derivation exercises, and often hints are provided at the end of the document. If you get stuck, take a look at the hints, and if you are still stuck, take a look in the solution to see the approach being taken. Then try to do it on your own.

Solutions are also provided for some coding exercises. If you get stuck, take a look at the solution, and then try to implement it on your own.

The Hidden Markov model

It is very important, for this exercise, to keep the graphical model for the HMM in mind, which is illustrated as:



Here, $\mathbf{x}_i$ is a discrete one-dimensional state variable with K possible values. There is, in general, no assumption on distribution of $\mathbf{y}_i$, but for this exercise we assume that $\mathbf{y}_i$ is a one-dimensional discrete variable.

10.1 Probabilities in HMM

This exercise is based on section 16.5.1, so be sure to read that before carrying out the exercise.

We will work with a 2-state model, with 4 different conditions. This scenario can mimic a number of real-life situations, e.g. gene sequencing (is a gene expressed or not), bio-markers, (is there a condition present or not) etc, but to simplify the calculations and the model, we assume that the observation variable y_n is one-dimensional, and only use a few states and conditions.

For discrete random variables, the distributions can simply be described as a matrix of appropriate dimension (see sec 2.2.2), so e.g $P(x_1 = \text{state } k) = P_k$, and then $P_k \in [0,1]^K$, $\sum_{k=1}^{K} P_k = 1$.

Assume the following distributions for the entirety of this exercise.

$$P_k = \begin{bmatrix} 0.7 \\ 0.3 \end{bmatrix}$$

$$P_{ij} = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$$

$$P(y|k) = \begin{bmatrix} 0.2 & 0.6 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.5 & 0.2 \end{bmatrix}$$

Exercise 10.1.1

Use the sum-rule to find an expression for $P(y_1)$ and then calculate $P(y_1 = 2)$.

Exercise 10.1.2

Inspect the code for this exercise and complete the code that implements the calculation of $P(y_1)$

Exercise 10.1.3

Use the sum-rule to show that an expression for $P(y_1, y_2)$ is

$$P(y_1, y_2) = \sum_{i=1}^{K} \sum_{j=1}^{K} P(y_2|x_2 = j)P(y_1|x_1 = i)P(x_2 = j|x_1 = i)P(x_1 = i)$$

and then calculate $P(y_1 = 2, y_2 = 3)$.

Exercise 10.1.4

Inspect the code for this exercise and complete the code that implements the calculation of $P(y_1, y_2)$. What is the number of arithmetic operations (multiplications and additions) taken?

Exercise 10.1.5

The number of operations that is needed in the naive implementation is infeasible, so we will derive a more efficient expression. We'll introduce the more compact notation

$$P(y_1, y_2, \cdots, y_n, x_n) := P(y_{[1:n]}, x_n)$$
$$\sum_{i=1}^{K} P(y_1|x_1 = i)P(x_1 = i) := \sum_{x_i} P(y_1|x_1)P(x_1)$$

Using this notation, we can write

$$P(y_2, y_1) = \sum_{x_2} P(y_1, y_2, x_2)$$

Show that

$$P(y_1, y_2, x_2) = P(y_2|x_2) \sum_{x_1} P(x_2|x_1)P(y_1, x_1)$$

Exercise 10.1.6

Define $\alpha(x_n) := P(y_{[1:n]}, x_n)$ that is, the joint distribution over observations up until time n and the latent variable at time n (i.e $\alpha(x_{n-1}) := P(y_{[1:n-1]}, x_{n-1})$ and so on).

Show that we get the recursive formula:

$$P(y_{[1:n]}, x_n) = P(y_n|x_n) \sum_{x_{n-1}} P(x_n|x_{n-1})\alpha(x_{n-1})$$

Exercise 10.1.7

We can exploit these result further using Bayes formula. Show that

$$P(x_n|y_{[1:n]}) = \frac{\alpha(x_n)}{P(y_{[1:n]})}$$
$$P(y_{[1:n]}) = \sum_{x_n} \alpha(x_n)$$

Additionally, identify the number of operations required to calculate $P(y_{[1:n]})$ using this scheme.

This is an important result, since this formula calculations the probability of being in a specific state, given all observations up until time n.

Exercise 10.1.8

Inspect the code for this exercise and complete the code that implements the calculation of $P(x_n|y_{[1:n]})$.

10.2 HMM model formulation and EM updates

This exercise is based on section 16.5.2, so be sure to read that before carrying out the exercise.

As a reminder, the EM algorithm consist of the following steps:

1. Specification of the complete log likelihood, $\ln p(\mathcal{X}, \mathcal{X}^l)$ (the model).
2. Derive $\mathcal{Q}(\boldsymbol{\xi}, \boldsymbol{\xi}^{(j)}) = \mathbb{E}\big[\ln p(\mathcal{X}, \mathcal{X}^l; \boldsymbol{\xi})^{(j)})\big]$.
3. Maximize $\mathcal{Q}(\boldsymbol{\xi}, \boldsymbol{\xi}^{(j)})$ in order to get $\boldsymbol{\xi}^{(j+1)}$.

where $\mathcal{X}$ denotes the set of observations, $\mathcal{X}^l$ denotes the set of latent random variables, and $\boldsymbol{\xi}$ is a vector of distribution parameters.

For HMM, we have $\mathcal{X} = Y$, $\mathcal{X}^l = X$, and to follow the same notation as the book, we set $\boldsymbol{\xi} = \Theta$. As a first step, we need to specify $p(X, Y)$.

Keep the graphical model of the HMM in mind when doing this exercise.

Exercise 10.2.1

We encode the state variable using using one-hot encoding (or one-of-K), and create a vector $\boldsymbol{x}_n \in \{0, 1\}^K$ for the n'th timestep, where $x_{n,k} = 1$ when the system is in state k and all other elements of the vector are 0.

Demonstrate, using an example, that this notation means that we can write

$$P(\boldsymbol{x}_1) = \prod_{k=1}^{K} P_k^{x_{1,k}}$$

and using the structure of the graph of the HMM model, that the likelihood becomes

$$p(Y,X) = P(\boldsymbol{x}_1)p(\boldsymbol{y}_1|\boldsymbol{x}_1)\prod_{n=2}^{N} P(\boldsymbol{x}_n|\boldsymbol{x}_{n-1})p(\boldsymbol{y}_n|\boldsymbol{x}_n)$$

$$P(\boldsymbol{x}_n|\boldsymbol{x}_{n-1}) = \prod_{i=1}^{K}\prod_{j=1}^{K} P_{ij}^{x_{n-1,j}x_{n,i}}$$

$$p(\boldsymbol{y}_n|\boldsymbol{x}_n) = \prod_{k=1}^{K}(p(\boldsymbol{y}_n|k;\boldsymbol{\theta}_k))^{x_{n,k}}$$

Exercise 10.2.2

Use the previous result to obtain the following expression for $\mathcal{Q}\big(\Theta,\Theta^{(j)}\big)$

$$\begin{aligned}\mathcal{Q}\big(\Theta,\Theta^{(j)}\big) &= \mathbb{E}[\ln p(X,Y;\Theta)] \\ &= \sum_{k=1}^{K}\mathbb{E}[\mathrm{x}_{1,k}]\ln P_k + \sum_{n=2}^{N}\sum_{i=1}^{K}\sum_{j=1}^{K}\mathbb{E}[\mathrm{x}_{n-1,j}\mathrm{x}_{n,i}]\ln P_{ij} \\ &\quad + \sum_{n=1}^{N}\sum_{k=1}^{K}\mathbb{E}[\mathrm{x}_{n,k}]\ln p(y_n \mid k;\boldsymbol{\theta}_k)\end{aligned}$$

Exercise 10.2.3

Let us define two helper functions

$$\gamma(\boldsymbol{x}_n) := P(\boldsymbol{x}_n|Y)$$
$$\xi(\boldsymbol{x}_{n-1},\boldsymbol{x}_n) := P(\boldsymbol{x}_{n-1},\boldsymbol{x}_n|Y)$$

Use these helper functions to obtain the final expression for the Expectation step:

$$\begin{aligned}\mathcal{Q}\big(\Theta,\Theta^{(j)}\big) &= \sum_{k=1}^{K}\gamma\big(x_{1,k}=1;\Theta^{(j)}\big)\ln P_k \\ &\quad + \sum_{n=2}^{N}\sum_{i=1}^{K}\sum_{j=1}^{K}\xi\big(x_{n-1,j}=1,x_{n,i}=1;\Theta^{(j)}\big)\ln P_{ij} \\ &\quad + \sum_{n=1}^{N}\sum_{k=1}^{K}\gamma\big(x_{n,k}=1;\Theta^{(j)}\big)\ln p(y_n \mid k;\boldsymbol{\theta}_k)\end{aligned}$$

Exercise 10.2.4

In the Maximization step, we maximize $\mathcal{Q}\big(\Theta,\Theta^{(j)}\big)$ with respect to the parameters, which in this case are P_k, P_{ij} and $\boldsymbol{\theta}_k$. We will only derive an update for P_{ij}, as the derivation for P_k is similar, and the derivation for $\boldsymbol{\theta}_k$ is not something that depends on the HMM as such, but the nature of $\mathbf{y}_i$ as this models our observations. Common choices are the Multinomial distribution (discrete), Gaussian distribution, or a Gaussian mixture model.

Use Lagrange multiplier to specify the Lagrangian, and then show that the update rule for P_{ij} is

$$P_{ij}^{(j+1)} = \frac{\sum_{n=2}^{N} \xi\big(x_{n-1,j} = 1, x_{n,i} = 1; \Theta^{(j)}\big)}{\sum_{n=2}^{N} \sum_{k=1}^{K} \xi(x_{n-1,j} = 1, x_{n,k} = 1; \Theta^{(j)})}$$

Remark

We are still missing some ingredients to have a fully fledged learning algorithm. We need to

1. Derive $P(\boldsymbol{x}_n|Y) := \gamma(x_n)$ instead of $P(\boldsymbol{x}_n|Y_{[1:n]})$ (which was derived in ex 10.1).
2. Derive $P(\boldsymbol{x}_{n-1}, \boldsymbol{x}_n|Y) := \xi(\boldsymbol{x}_{n-1}, \boldsymbol{x}_n)$.
3. Derive $P_k^{(j+1)}$.
4. Derive $\boldsymbol{\theta}_k^{(j+1)}$.

Step number 1–2 is solved in a similar fashion as the forward pass (and requires the backward pass). Step 3 is derived using Lagrange multipliers in the same manner as deriving $P_{ij}^{(t+1)}$. And finally, step 4 requires us to specify the emission distribution, and the derivation is similar as last week.

As mention in the introduction, to carry out *all* of these steps are quite lengthy both in terms of coding and derivations, so the practical examples in this exercise are using built-in library functions.

10.3 Convergence of parameter estimation using EM

In this exercise we will create a "teacher" model and use that to generate a sequence of length N. Then we will train M models to investigate the convergence properties of the EM algorithm. Inspect and run the code for this exercise. What steps is carried out? Make a simplified flow-chart of the operations being taken. Why does the EM fail to find good models (sometimes) for small sequences even though we initialize with correct parameters?

Modify the code to create learning curves and error bars when the EM is randomly initialized and initialized with the teacher model parameters.

10.4 HMM for COVID-19

Inspect and run the code for this exercise. What steps does the code carry out? Make a simplified flow-chart of the operations being taken. Make sure to identify the preprocessing steps, the number of hidden steps and the type of state emission distribution used. Modify the code to run multiple models and use the best. Experiment with other data representations (or data sources) to find better results.

HINTS

Exercise 10.1.1

$P(y_1 = 2) = 0.17$

Exercise 10.1.3

$P(y_1 = 2, y_2 = 3) = 0.084$

Exercise 10.2.3

Since $\mathrm{x}_{1,k}$ is a binary variable, the expectation is equal to the probability $\mathbb{E}[\mathrm{x}_{1,k}]$.

Exercise 10.2.4

The constraint we have is

$$\sum_{i=1} P_{ij} = 1 \quad \forall j$$

02471 Machine Learning for Signal Processing

Week 11: State-space models – Kalman filtering

This exercise is based on S. Theodoridis: Machine Learning, A Bayesian and Optimization Perspective 2nd edition, section(s) 4.10.

The objective of this exercise is to get a view on the usage of state-space models in the linear dynamical system setting. As application we choose the most often used approach and use the Kalman filter, where some parameters are assumed to be known. These parameters can for example be learned using the EM algorithm (just like the HMM).

Overview

The exercise have the following structure:

11.1 will derive the Kalman filter equations.

11.2 will experiment with the derived formulas, and get acquainted on how Kalman filter works in a 1-d setting of an AR(1) process.

11.3 is a demo where we use Kalman filtering and a RTS smoother to track an moving object using noisy observations.

Notation

- $J(\boldsymbol{\theta})$ is a cost function that we are seeking to minimize with respect to $\boldsymbol{\theta}$.
- $\mathcal{N}(\mu, \sigma^2)$ denotes a normal (Gaussian) distribution with mean value μ and variance σ^2.
- $\hat{a}_{n|n-1}$ denotes the estimation of signal a at time n given data observed until time $n-1$.

Code

The code can be found in the .m and .py files named in the same way as exercises, ie. the code for exercise 11.x.y is in the file 11_x_y.m (or .py).

For coding exercises that requires implementation we will usually write `complete this line` where the implementing should be done.

Solutions

The solution is provided for all derivation exercises, and often hints are provided at the end of the document. If you get stuck, take a look at the hints, and if you are still stuck, take a look in the solution to see the approach being taken. Then try to do it on your own.

Solutions are also provided for some coding exercises. If you get stuck, take a look at the solution, and then try to implement it on your own.

11.1 Derivation of the Kalman filter

We consider the space-state model

$$\mathbf{x}_n = F_n \mathbf{x}_{n-1} + \boldsymbol{\eta}_n$$
$$\mathbf{y}_n = H_n \mathbf{x}_n + \mathbf{v}_n$$

The goal is to predict $\mathbf{x}_n$ but we have only observed $\mathbf{y}_n$. If we have observed up to $n-1$ points, we call the estimator for $\mathbf{x}_n$ the prior estimator since we estimate without having observed $\mathbf{y}_n$. We denote this estimator $\hat{\mathbf{x}}_{n|n-1}$.

If we have observed up to n points, we call the estimator for $\mathbf{x}_n$ the posterior estimator since we now estimate with having observed $\mathbf{y}_n$. We denote this estimator $\hat{\mathbf{x}}_{n|n}$.

To goal of the Kalman filter is to provide expressions for these two estimators in a recursive manner. Then we can, as we acquire a new point $\mathbf{y}_n$ do a re-estimation and not solve the problem anew. This nature of this approach is very similar to LMS and RLS, and for that reason, Kalman filtering is often talked about as an RLS algorithm.

In Kalman filtering, we make the following assumptions:

- The distribution of the noise terms are known, and have the following properties
  - $\mathbb{E}\left[\boldsymbol{\eta}_n \boldsymbol{\eta}_n^T\right] := Q_n$
  - $\mathbb{E}\left[\boldsymbol{\eta}_n \boldsymbol{\eta}_m^T\right] = 0, n \neq m$
  - $\mathbb{E}[\boldsymbol{\eta}_n] = \mathbf{0}$
  - $\mathbb{E}\left[\mathbf{v}_n \mathbf{v}_n^T\right] := R_n$
  - $\mathbb{E}\left[\mathbf{v}_n \mathbf{v}_m^T\right] = 0, n \neq m$
  - $\mathbb{E}[\mathbf{v}_n] = \mathbf{0}$
  - $\mathbb{E}\left[\boldsymbol{\eta}_n \mathbf{v}_m^T\right] = 0, \forall n, \forall m$
- The matrices F_n and H_n are known.

Exercise 11.1.1

Describe in words what these properties mean. Are these assumptions reasonable?

Exercise 11.1.2

We make the following definitions

$$\hat{\mathbf{y}}_n := H_n \hat{\mathbf{x}}_{n|n-1}$$
$$\mathbf{e}_n := \mathbf{y}_n - \hat{\mathbf{y}}_n$$
$$\mathbf{e}_{n|n} := \mathbf{x}_n - \hat{\mathbf{x}}_{n|n}$$
$$\mathbf{e}_{n|n-1} := \mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}$$
$$P_{n|n} := \mathbb{E}\left[\mathbf{e}_{n|n} \mathbf{e}_{n|n}^T\right]$$
$$P_{n|n-1} := \mathbb{E}\left[\mathbf{e}_{n|n-1} \mathbf{e}_{n|n-1}^T\right]$$

Discuss these definition and describe what they mean in words. Why e.g. do we define $\hat{\mathbf{y}}_n$ with respect to $\hat{\mathbf{x}}_{n|n-1}$ and not $\hat{\mathbf{x}}_{n|n}$?

Exercise 11.1.3

We first seek expressions for the predictions $\hat{\mathbf{x}}_{n|n-1}$ and $P_{n|n-1}$. For predicting $\hat{\mathbf{x}}_{n|n-1}$ we simply use the model and have a noise-free prediction, so that

$$\hat{\mathbf{x}}_{n|n-1} = F_n \hat{\mathbf{x}}_{n-1|n-1}$$

To get an expression for $P_{n|n-1}$ we have to use substitutions of the previous definitions. Show that

$$P_{n|n-1} = F_n P_{n-1|n-1} F_n^T + Q_n$$

Exercise 11.1.4

The next step is to derive the updates.

In Kalman filtering we define the following recursion

$$\hat{\mathbf{x}}_{n|n} := \hat{\mathbf{x}}_{n|n-1} + K_n \mathbf{e}_n$$

This is simply a choice that is being made, stating that we aim to update our prediction in an adaptive manner as points become available. We have seen this update form many times before in the course, e.g. relate this to the LMS/RLS algorithm update. It simply means, at iteration n, we have to make a correction to our previous point. This correction is based on the prediction error $\mathbf{e}_n$ and some correction K_n. We aim to derive an expression for the correction factor.

To this end, we choose to minimize the mean squared error of the prediction performance of $\hat{\mathbf{x}}_{n|n}$, that is, we minimize the following with respect to K_n:

$$J(K_n) = \mathbb{E}\big[\mathbf{e}_{n|n}^T \mathbf{e}_{n|n}\big]$$

Using the previously stated definitions we have (show this)

$$J(K_n) = \text{trace}(P_{n|n})$$

In order to take the derivate of $J(K_n)$ w.r.t. K_n we need to rewrite $P_{n|n}$. By using the previous definitions, show

$$P_{n|n} = \mathbb{E}\big[\mathbf{e}_{n|n-1}\mathbf{e}_{n|n-1}^T\big] - K_n \mathbb{E}\big[\mathbf{e}_n \mathbf{e}_{n|n-1}^T\big] - \mathbb{E}\big[\mathbf{e}_{n|n-1}\mathbf{e}_n^T\big] K_n^T + K_n \mathbb{E}\big[\mathbf{e}_n \mathbf{e}_n^T\big] K_n^T$$

Next, show that these individual expectations can be written as

$$\begin{aligned}
\mathbb{E}\big[\mathbf{e}_{n|n-1}\mathbf{e}_{n|n-1}^T\big] &= P_{n|n-1} \\
\mathbb{E}\big[\mathbf{e}_n \mathbf{e}_{n|n-1}^T\big] &= H_n P_{n|n-1} \\
\mathbb{E}\big[\mathbf{e}_{n|n-1}\mathbf{e}_n^T\big] &= P_{n|n-1} H_n^T \\
\mathbb{E}\big[\mathbf{e}_n \mathbf{e}_n^T\big] &= H_n P_{n|n-1} H_n^T + R_n
\end{aligned}$$

Let $S = H_n P_{n|n-1} H_n^T + R_n$, so that $\mathbb{E}\big[\mathbf{e}_n \mathbf{e}_n^T\big] = S$, then put it all together to get

$$P_{n|n} = P_{n|n-1} - K_n H_n P_{n|n-1} - P_{n|n-1} H_n^T K_n^T + K_n S K_n^T$$

Exercise 11.1.5

We have to minimize trace($P_{n|n}$) with respect to K_n, so we take the derivate w.r.t. K_n and set to zero. Using that trace is a linear operator i.e. trace$(A+B) = $ trace$(A) +$ trace(B), and trace$(A) =$ trace(A^T) and using the rules

$$\frac{\partial \text{trace}(AB)}{\partial A} = B^T$$
$$\frac{\partial \text{trace}(ACA^T)}{\partial A} = 2AC$$

First show trace$(P_{n|n-1}H_n^T K_n^T) =$ trace$(K_n H_n P_{n|n-1}^T)$ and then use this to get

$$\text{trace}(P_{n|n}) = \text{trace}(P_{n|n-1}) - 2\text{trace}(K_n H_n P_{n|n-1}^T) + \text{trace}(K_n S K_n^T)$$

Take the derivative, set to zero to obtain

$$K_n = P_{n|n-1}H_n^T S^{-1}$$

That means we know have an expression for K_n that minimizes the MSE. We note that to calculate K_n, we need an expression for $P_{n|n-1}$ (the other terms are considered known per our assumptions), this is the next and final step.

Exercise 11.1.6

Go back to the previously derived recursion for $P_{n|n}$, show that $K_n S K_n^T = P_{n|n-1}H_n^T K_n^T$, and by substitution obtain

$$P_{n|n} = P_{n|n-1} - K_n H_n P_{n|n-1}$$

We have now derived the Kalman filtering algorithm. Relate the derived terms to algorithm 4.2, page 169 in the book.

11.2 Kalman filtering on an AR process

As we have discussed in the course, many signals can be modeled adequately as an AR process, so we can gain insight with Kalman filtering by assuming $\mathbf{x}$ follows an AR process, simulate data, and then see how reliably Kalman filter can predict $\mathbf{x}_n$ for the observations $\mathbf{y}_n$.

In this exercise we work through example 4.4 (although we use positive a_i coefficients instead of negative as the book).

Exercise 11.2.1

Given the following model for data generation

$$\mathrm{x}_n = \sum_{i=1}^{l} a_i \mathrm{x}_{n-i} + \eta_n$$
$$\mathrm{y}_n = \mathrm{x}_n + \mathrm{v}_n$$

where η_n is a white noise sequence with variance σ_η^2 and v_n is a white noise sequence with variance σ_v^2, specify the matrices F_n and H_n so that the model fits the state-space model. Are they fixed or time-varying (do we need the n subscript index?)?

Exercise 11.2.2

Inspect the code associated with the exercise and complete the code-lines. Generate some data using different parameters for Q_n, R_n, and a_i, and experiment with how well the Kalman filter performs for different noise levels and length of process.

11.3 Kalman filtering and smoothing on a moving object

This exercise serves as a demonstration on using Kalman filtering and smoothing on a moving object using noisy observations. The code only serves as a demo.

Inspect the code just to get an overview of the structure. Experiment with different noise parameters and sampling rate. Consider to change the script so that the filtering is interactive and happens on the fly. Another change to consider is the limit the smoother only to take a few points of the future into account instead of the entire trajectory, or to run the smoother as a window.

NOTE: The RTS smoother is NOT covered in the curriculum, it is only in this exercise as a demo.

HINTS

02471 Machine Learning for Signal Processing

Week 12: Kernel methods

This exercise is based on S. Theodoridis: Machine Learning, A Bayesian and Optimization Perspective 2nd edition, section(s) 11.5–11.7.

The objective of this exercise is to get familiar with kernel methods, and in particular the kernel ridge regression where we perform smoothing (de–noising) and anomaly detection on a time–series signal.

In this exercise, we assume all inner product operations are performed in Hilbert spaces, and all kernel functions are reproducing kernels such that we operate in reproducing kernel Hilbert space (RKHS).

Overview

The exercise have the following structure:

12.1 will present the kernel methods in a simplified setting, and show how kernels can make non-linear separable points separable in RKHS.

12.2 will derive the solution for the kernel ridge regression method.

12.3 will perform de–noising and anomaly detection on a time–series signal using kernel ridge regression.

Notation

- $J(\boldsymbol{\theta})$ is a cost function that we are seeking to minimize with respect to $\boldsymbol{\theta}$.
- $\langle \cdot, \cdot \rangle$ denotes the inner product, e.g. in $\mathbb{R}^N$, we have $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^T \boldsymbol{y} = \boldsymbol{x} \cdot \boldsymbol{y}$. If we have $\langle \boldsymbol{x}, \boldsymbol{x} \rangle$ (and we have a Hilbert space), the inner product denotes the norm of the space. In $\mathbb{R}^N$ we have $\langle \boldsymbol{x}, \boldsymbol{x} \rangle = \|\boldsymbol{x}\|_2$.
- $\kappa(\boldsymbol{x}, \boldsymbol{y})$ denotes a kernel function. $\kappa(\boldsymbol{x}, \boldsymbol{y})$ is symmetric, i.e. $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \kappa(\boldsymbol{y}, \boldsymbol{x})$.
- $\mathcal{K}$ denotes a kernel matrix (defined in Eq. 11.11 in the book). $\mathcal{K}$ is symmetric: $\mathcal{K} = \mathcal{K}^T$.

Code

The code can be found in the .m and .py files named in the same way as exercises, ie. the code for exercise 12.x.y is in the file 12_x_y.m (or .py).

For coding exercises that requires implementation we will usually write `complete this line` where the implementing should be done.

Solutions

The solution is provided for all derivation exercises, and often hints are provided at the end of the document. If you get stuck, take a look at the hints, and if you are still stuck, take a look in the solution to see the approach being taken. Then try to do it on your own.

Solutions are also provided for some coding exercises. If you get stuck, take a look at the solution, and then try to implement it on your own.

12.1 Obtaining linear separability using kernels

This exercise will introduce you to the basic ideas of kernels. We will work in a two–dimensional toy-example to make the processes clear. We will generate two categories of points in a circular disc, so that they are not linearly separable but are clearly non-linearly separable.

Exercise 12.1.1

Run and inspect the code for this exercise. The program generates two sets of points. Identify how the points are generated and write down the formulas.

Exercise 12.1.2

The points can be transformed using the mapping function

$$\mathbb{R}^2 \ni \boldsymbol{x} \longmapsto \boldsymbol{\phi}(\boldsymbol{x}) = \left[x_1^2, \sqrt{2}x_1x_2, x_2^2\right] \in \mathbb{R}^3$$

Implement the mapping function in the code and plot the points in $3d$ and validate they are now linearly separable.

Exercise 12.1.3

A useful connection of the mapping function can be made. If we take the inner product of the mapping function, this corresponds to the squared inner product of the vectors themselves:

$$\boldsymbol{\phi}^T(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{y}) = \left(\boldsymbol{x}^T\boldsymbol{y}\right)^2$$

Show that this relation is true.

Exercise 12.1.4

This inner product is referred to as the *homogeneous polynomial* kernel

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T\boldsymbol{y})^r$$

Implement the kernel in the code associated with this exercise.

Exercise 12.1.5

The data has now been transformed into an inner product format where the representer theorem can now be used. The theorem loosely states that, if we have the following optimization problem

$$\min_{f\in\mathbb{H}} J(f) := \sum_{n=1}^{N} \mathcal{L}(y_n, f(\boldsymbol{x}_n)) + \lambda\Omega\left(\|f\|^2\right)$$

Then each minimizer $f \in \mathbb{H}$ of the minimization task admits a representation of the form:

$$f(\cdot) = \sum_{n=1}^{N} \theta_n \kappa(\cdot, x_n)$$

For now, we ignore the loss function and focus our attention of the form $f(\cdot)$. What operation is this function effectively carrying out? Implement and apply the function using $\boldsymbol{\theta} = \mathbf{1}$ and validate that the function is able to linearly separate our training data.

Exercise 12.1.6

Repeat the exercise using the Gaussian kernel:

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2}\right)$$

Apply the representer theorem again, choosing $\boldsymbol{\theta} = \mathbf{1}$ using $\sigma^2 = 1$. Verify that this choice results in a correct separation.

Find values for σ^2 where the data is no longer separable. Explain why this happens.

12.2 Derivation of the kernel ridge regression

Make sure to skim section 11.7 in the book before carrying out this exercise.

Recall from section 3.8 that the ridge regression (without bias) minimizes the following function:

$$J_{RR}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \left(y_n - \sum_{i=1}^{l} \theta_i x_{ni} \right)^2 + \lambda \sum_{i=1}^{l} |\theta_i|^2$$

The kernel ridge regression (without bias) instead minimizes

$$J(\boldsymbol{\theta}) := \sum_{n=1}^{N} \left(y_n - \sum_{m=1}^{N} \theta_m \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) \right)^2 + C\langle f, f \rangle$$

where $C \in \mathbb{R}$ is a regularization parameter. On inspection we see two things have changed from the original ridge regression loss function; the function has now changed according to the representer theorem, and we are regularizing the norm of f instead of the norm of $\boldsymbol{\theta}$.

Exercise 12.2.1

From section 8.16 (only available online), definition 8.15, we have:

Definition 8.15 (Inner product). Let V be a linear space. The inner product is a function

$$f : V \times V \to \mathbb{C}$$

which assigns a value in $\mathbb{C}$, denoted as $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$, to every point of elements $\boldsymbol{x}, \boldsymbol{y} \in V$, with the following properties:

- $\langle \boldsymbol{x}, \boldsymbol{x} \rangle \geq 0$, and $\langle \boldsymbol{x}, \boldsymbol{x} \rangle = 0$ if and only if $\boldsymbol{x} = 0$.

- $\langle \boldsymbol{x} + \boldsymbol{y}, \boldsymbol{z} \rangle = \langle \boldsymbol{x}, \boldsymbol{z} \rangle + \langle \boldsymbol{y}, \boldsymbol{z} \rangle$.
- $\langle a\boldsymbol{x}, \boldsymbol{y} \rangle = a\langle \boldsymbol{x}, \boldsymbol{y} \rangle$.
- $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \langle \boldsymbol{y}, \boldsymbol{x} \rangle^*$.

Use the properties of an inner product to show that $\langle f, f \rangle = \boldsymbol{\theta}^T \mathcal{K}^T \boldsymbol{\theta}$.

Exercise 12.2.2

From the previous result it follows that

$$J(\boldsymbol{\theta}) = (\boldsymbol{y} - \mathcal{K}\boldsymbol{\theta})^T (\boldsymbol{y} - \mathcal{K}\boldsymbol{\theta}) + C\boldsymbol{\theta}^T \mathcal{K} \boldsymbol{\theta}$$

where the first term has been rewritten according to

$$\sum_{n=1}^{N} \left(y_n - \sum_{m=1}^{N} \theta_m \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) \right)^2 = (\boldsymbol{y} - \mathcal{K}\boldsymbol{\theta})^T (\boldsymbol{y} - \mathcal{K}\boldsymbol{\theta})$$

as has been done in previous exercises.

To solve the optimization task, show that we obtain

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -2(\boldsymbol{y}^T \mathcal{K})^T + 2\mathcal{K}^T \mathcal{K} \boldsymbol{\theta} + 2C\mathcal{K}^T \boldsymbol{\theta}$$

and that $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbf{0}$ leads to the solution

$$\hat{\boldsymbol{\theta}} = (\mathcal{K} + CI)^{-1} \boldsymbol{y}$$

12.3 Smoothing using kernel ridge regression

In this exercise we will do de-noising and anomaly detection using kernel methods. The corresponding material in the book is example 11.2, page 553 and exercise 11.14.

We will work on a piece of audio from the Blade Runner movie, and add both noise and anomalies (outliers) to the data. We will then use kernel ridge regression to reconstruct the data.

Exercise 12.3.1

Make sure to skim sec 11.7 in the book before carrying out this exercise.

Run and inspect the code associated with the exercise. Relate the equations in section 11.7 to the code and complete the missing lines in the code.

The code will load the sound, extract 100 samples, and then add white Gaussian noise and randomly "hit" 10 of the data samples with outliers (set the outlier values to 80% of the maximum value of the data samples). As kernel the rbf/Gaussian kernel is used.

The code finds the reconstructed data samples using both the biased and unbiased kernel ridge regression method, and plots the fitted curves of the reconstructed samples together with the data used for training.

Try different values for the parameters C and σ, and comment on the results. Make sure you investigate their impact on the smoothing of the regression.

HINTS

Exercise 12.2.1

For ridge regression we define $f(\boldsymbol{x}) = \sum_{n=1}^{N} \theta_n \kappa(\boldsymbol{x}, \boldsymbol{x}_n)$. Insert this expression into the inner product and reduce to obtain the result.

02471 Machine Learning for Signal Processing

Week 13: Support vector regression

This exercise is based on S. Theodoridis: Machine Learning, A Bayesian and Optimization Perspective 2nd edition, section(s) 11.8.

The objective of this exercise is to get more familiar with kernel methods and expand on the exercise from last week. In particular, we analyze the support vector regression method. We will use support vector regression to perform de–noising and anomaly detection on a time–series signal.

In this exercise, we assume all inner product operations are performed in Hilbert spaces, and all kernel functions er reproducing kernels such that we operate in reproducing kernel Hilbert space.

Overview

The exercise have the following structure:

13.1 will analyze the support vector regression method, and go from the linear ϵ–insensitive loss function to the optimization problem, and gain insights to the of the method.

13.2 will perform de–noising and anomaly detection on a time–series signal using kernel ridge regression and support vector regression

You can carry out the exercises in any order. To get full benefit of exercise 13.2, you should carry out the preceding exercises.

Notation

- $J(\boldsymbol{\theta})$ is a cost function that we are seeking to minimize with respect to $\boldsymbol{\theta}$.
- $\langle \cdot, \cdot \rangle$ denotes the inner product, e.g. in $\mathbb{R}^N$, we have $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^T \boldsymbol{y} = \boldsymbol{x} \cdot \boldsymbol{y}$. If we have $\langle \boldsymbol{x}, \boldsymbol{x} \rangle$ (and we have a Hilbert space), the inner product denotes the norm of the space. In $\mathbb{R}^N$ we have $\langle \boldsymbol{x}, \boldsymbol{x} \rangle = \|\boldsymbol{x}\|_2^2$.
- $\kappa(\boldsymbol{x}, \boldsymbol{y})$ denotes a kernel function. $\kappa(\boldsymbol{x}, \boldsymbol{y})$ is symmetric, i.e. $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \kappa(\boldsymbol{y}, \boldsymbol{x})$.
- $\mathcal{K}$ denotes a kernel matrix (defined in Eq. 11.11 in the book). $\mathcal{K}$ is symmetric: $\mathcal{K} = \mathcal{K}^T$.

Code

The code can be found in the .m and .py files named in the same way as exercises, ie. the code for exercise 13.x.y is in the file 13_x_y.m (or .py).

For coding exercises that requires implementation we will usually write `complete this line` where the implementing should be done.

Solutions

The solution is provided for all derivation exercises, and often hints are provided at the end of the document. If you get stuck, take a look at the hints, and if you are still stuck, take a look in the solution to see the approach being taken. Then try to do it on your own.

Solutions are also provided for some coding exercises. If you get stuck, take a look at the solution, and then try to implement it on your own.

13.1 Support vector regression (SVR)

In this exercise we are going to analyze the solution offered by the support vector regression method.

Make sure to skim section 11.8 in the book before carrying out this exercise.

At stated in the lecture, a regression task on the form

$$y_n = g(\boldsymbol{x}_n) + \eta_n, \quad n = 1, 2, \cdots, N$$

where η_n is i.i.d. noise, with the loss

$$\mathcal{L}(y, f(\boldsymbol{x})) = \begin{cases} |y - f(\boldsymbol{x})| - \epsilon, & \text{if} \quad |y - f(\boldsymbol{x})| > \epsilon \\ 0, & \text{if} \quad |y - f(\boldsymbol{x})| \leq \epsilon \end{cases} \tag{1}$$

can be written as the following optimization problem

$$\underset{\boldsymbol{\theta}, \boldsymbol{\xi}, \tilde{\boldsymbol{\xi}},}{\arg\min} \quad J(\boldsymbol{\theta}, \boldsymbol{\xi}, \tilde{\boldsymbol{\xi}}) := \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\left(\sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \tilde{\xi}_n\right) \tag{2}$$

$$\text{s.t.} \qquad y_n - f(\boldsymbol{x}_n) \leq \epsilon + \tilde{\xi}_n \tag{3}$$

$$-(y_n - f(\boldsymbol{x}_n)) \leq \epsilon + \xi_n \tag{4}$$

$$\tilde{\xi}_n \geq 0 \tag{5}$$

$$\xi_n \geq 0 \tag{6}$$

To give you an idea of where we are going, the final optimization problem for the support vector regression method can be written as

$$\hat{y} = \sum_{n=1}^{N} (\tilde{\lambda}_n - \lambda_n) \kappa(\boldsymbol{x}_n, \boldsymbol{x}) + \hat{\theta}_0 \tag{7}$$

$$\underset{\boldsymbol{\lambda}, \tilde{\boldsymbol{\lambda}},}{\arg\max} \quad \sum_{n=1}^{N} (\tilde{\lambda}_n - \lambda_n) y_n - \epsilon(\tilde{\lambda}_n - \lambda_n) \tag{8}$$

$$-\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} (\tilde{\lambda}_n - \lambda_n)(\tilde{\lambda}_m - \lambda_m) \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) \tag{9}$$

$$\text{s.t.} \quad 0 \leq \tilde{\lambda}_n \leq C, \quad 0 \leq \lambda_n \leq C, \quad n = 1, 2, \cdots, N \tag{10}$$

$$\sum_{n=1}^{N} \tilde{\lambda}_n = \sum_{n=1}^{N} \lambda_n \tag{11}$$

Inspection of this problem reveals 3 choices of importance, the choice of C, which bounds the value of $\tilde{\lambda}_n$ and λ_n, the choice of ϵ, and the choice of kernel function, $\kappa(\boldsymbol{x}_n, \boldsymbol{x}_m)$.

We are going to analyze the solution in order to get a better understanding of the parameter choice of ϵ and C, and their impact on the predictions.

Exercise 13.1.1 (This is a difficult exercise)

For now, we will return to the original stated problem in eq (1), and we assume a linear model $f(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x} + \theta_0$.

Argue why this loss makes sense (compared to the least squares loss) if the data is contaminated by outliers, and show the loss stated in eq (1) can be rewritten to eq. (2–6) (with an added regularization on $\boldsymbol{\theta}$).

Exercise 13.1.2

The learning problem in eq. (2–6) is stated in a version where it can readily be solved by using Lagrange multipliers.

From appendix C.2, we have the following result:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & J(\boldsymbol{\theta}) \\ \text{s.t.} \quad & f_i(\boldsymbol{\theta}) \geq 0, \quad i = 1, 2, \cdots, m \end{aligned}$$

then the Lagrangian is

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = J(\boldsymbol{\theta}) - \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{\theta})$$

And is solved by:

$$\begin{aligned} & \frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \Big|_{\theta=\theta_*} = \mathbf{0} \\ & \lambda_i \geq 0 \qquad\quad i = 1, 2, \cdots, m \\ & \lambda_i f_i(\boldsymbol{\theta}_*) = 0 \quad i = 1, 2, \cdots, m \end{aligned}$$

That tells us immediately how to convert this problem. Show that, by introducing Lagrange multipliers, that you arrive at the following problem:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}, \theta_0, \boldsymbol{\xi}, \tilde{\boldsymbol{\xi}}, \boldsymbol{\lambda}, \tilde{\boldsymbol{\lambda}}, \boldsymbol{\mu}, \tilde{\boldsymbol{\mu}}) \quad & = \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\left(\sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \tilde{\xi}_n\right) \\ & + \sum_{n=1}^{N} \tilde{\lambda}_n (y_n - \boldsymbol{\theta}^T \boldsymbol{x}_n - \theta_0 - \epsilon - \tilde{\xi}_n) \\ & + \sum_{n=1}^{N} \lambda_n (-y_n + \boldsymbol{\theta}^T \boldsymbol{x}_n + \theta_0 - \epsilon - \xi_n) \\ & - \sum_{n=1}^{N} \tilde{\mu}_n \tilde{\xi}_n - \sum_{n=1}^{N} \mu_n \xi_n \qquad (12) \\ \text{s.t.} \quad & \tilde{\lambda}_n (y_n - \boldsymbol{\theta}^T \boldsymbol{x}_n - \theta_0 - \epsilon - \tilde{\xi}_n) = 0, \quad n = 1, 2, \cdots, N \qquad (13) \\ & \lambda_n (\boldsymbol{\theta}^T \boldsymbol{x}_n + \theta_0 - y_n - \epsilon - \xi_n) = 0, \quad n = 1, 2, \cdots, N \qquad (14) \\ & \tilde{\mu}_n \tilde{\xi}_n = 0, \quad \mu_n \xi_n = 0, \quad n = 1, 2, \cdots, N \qquad (15) \\ & \tilde{\lambda}_n \geq 0, \ \lambda_n \geq 0, \ \tilde{\mu}_n \geq 0, \ \mu_n \geq 0, \quad n = 1, 2, \cdots, N \qquad (16) \end{aligned}$$

Exercise 13.1.3

The next step is to find the derivative w.r.t. the learning parameters, and set those derivatives to zero, that is

$$\frac{\partial}{\partial \boldsymbol{\theta}}\mathcal{L}(\cdot) = \mathbf{0}, \quad \frac{\partial}{\partial \theta_0}\mathcal{L}(\cdot) = 0, \quad \frac{\partial}{\partial \tilde{\xi}_n}\mathcal{L}(\cdot) = 0, \quad \frac{\partial}{\partial \xi_n}\mathcal{L}(\cdot) = 0$$

Show that solving these equations leads to the following solutions

$$\hat{\boldsymbol{\theta}} = \sum_{n=1}^{N}(\tilde{\lambda}_n - \lambda_n)\boldsymbol{x}_n \tag{17}$$

$$\sum_{n=1}^{N}\tilde{\lambda}_n = \sum_{n=1}^{N}\lambda_n \tag{18}$$

$$C = \tilde{\lambda}_n + \tilde{\mu}_n \tag{19}$$

$$C = \lambda_n + \mu_n \tag{20}$$

Exercise 13.1.4 (This is a difficult exercise)

We have a lot of equations that must be fulfilled, and this has some side-effects on the solutions.

Work through all the following statements, and show they are true by identifying and manipulating the correct constraints.

First we look at the case where $\xi_n > 0$ (that is, $f(\boldsymbol{x}_n)$ is outside the ϵ-tube), this implies

$$\xi_n > 0 \Rightarrow \mu_n = 0 \Rightarrow \lambda = C$$

For the case $\xi_n = 0$, $f(\boldsymbol{x}_n)$ is either on the margin of the ϵ-tube or inside the ϵ-tube.

We define $e_n := |f(\boldsymbol{x}_n) - y_n|$ and first look on the case where e_n is on the margin, ie. we have $e_n = \epsilon$. This implies λ_n is a free parameter, i.e. one constraint reduces to

$$e_n = \epsilon \Rightarrow \lambda_n \cdot 0 = 0$$

This implies λ_n is "free" to be any value. However due to the previous non-negativity constraints, we have $0 \leq \lambda_n \leq C$ when $e_n = \epsilon$.

In the third situation, we have the prediction inside the margin, i.e. $e_n < \epsilon$. This situation does not result in the previous reduction $\lambda_n \cdot 0 = 0$, but instead

$$e_n < \epsilon \Rightarrow \lambda_n \cdot \delta\epsilon = 0$$

where $\delta\epsilon = \epsilon - e_n > 0$. This forces $\lambda_n = 0$, so that means, for points that are within the ϵ-tube, $\lambda_n = 0$. That means they do not contribute to the prediction.

For this reason, we can distinguish between points that contribute to the prediction, and point that does not. The points that contribute, have positive λ_n values, and these points are called *support vectors*.

13.2 Smoothing using kernel methods

In this exercise we will do de-noising and anomaly detection using kernel methods. The corresponding material in the book is example 11.3, page 560 and exercise 11.15.

We will work on a piece of audio from the Blade Runner movie, and add both noise and anomalies (outliers) to the data. We will then use and Support Vector Regression (SVR) to reconstruct the data.

Exercise 13.2.1

Make sure to skim sec 11.8 in the book before carrying out this exercise.

Run and inspect the code associated with the exercise.

The code will load sound, extract 100 samples, and then add white Gaussian noise and randomly "hit" 10 of the data samples with outliers (set the outlier values to 80% of the maximum value of the data samples). As kernel the rbf/Gaussian kernel is used.

The code finds the reconstructed data samples , and plots the fitted curves of the reconstructed samples together with the data used for training.

Compare the solution to Kernel ridge regression from last week and explain why the SVR method works better in the presence of outliers. Experiment with the parameters C, σ, and ϵ and comment on the results.