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:

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

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

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

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.

In this exercise we will use Hanning window, using the built-in 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.

Exercise 7.4.2

Visualize the window function and consider the effect. Try and replace the window function using a rectangular window.

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 *magnitude* spectrum.

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

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.