Homework: Linear Regression without Correspondences

This homework invites students to implement several algorithms, including

- brute force,
- RANSAC,
- alternating minimization,
- robust regression,
- the algebraic method,

and explore their limitations in solving the problem of Linear Regression without Correspondences. We provide MATLAB starting code for this homework.

Submission. Your submission (`pinyinname_number.zip` or `.rar`) should include all code files, and a PDF report (`pinyinname_number.pdf`), and be sent via email to `penglz@shanghaitech.edu.cn`. Your code implementation can be in any programming language you like (except in §6 where we use some MATLAB functions), not necessarily in MATLAB, but the file name should be in agreement with the aforementioned ones (e.g., it can be `SLR_2_brute_force.py` for Python implementation). The deadline is **December 3rd, 2019** before the class.

Grading. This homework is graded by not only checking the PDF report, we will also investigate and run the code. Please provide instructions when you feel necessary for running your code.

1 Appetizer (10 points)

We assume the following model:

$$y = \Pi A x + \epsilon, y \in \mathbb{R}^m, x \in \mathbb{R}^n, \tag{1.1}$$

where Π belongs to the set $\mathcal{P}$ of $m \times m$ permutation matrices and ϵ is noise. The evaluation metric for performance of algorithms is the *relative estimation error*

$$100\% \times \frac{\|x - \hat{x}\|_2}{\|x\|_2}, \tag{1.2}$$

where x is the groundtruth and $\hat{x}$ the estimate given by some algorithm.

Exercise 1.1 (Synthetic Data Generation, 5 points)**.** Without data, we can not play. Synthetic data are generated in the following way. The $m \times n$ matrix A and the parameter $x \in \mathbb{R}^n$ are randomly sampled from the standard normal distribution. The vector y is then obtained by 1) shuffling the entries of Ax via an arbitrary permutation $\Pi \in \mathcal{P}$ and 2) contaminating the shuffled entries by additive noise sampled from the normal distribution $\mathcal{N}(0, \sigma^2 I_m)$.

you are required to implement the function `SLR_1_gen_data`. This function takes four parameters as inputs:

- m, n: the dimensions of the data.
- `sigma`: the standard deviation of noise.
- `shuffled_ratio`: how many percentages of the rows of A, or equivalently the entries of y, should be shuffled.

Exercise 1.2 (5 points)**.** For given $x \in \mathbb{R}^n$, design an algorithm, ideally of complexity $\mathcal{O}(m \times \log m)$, to find the optimal permutation Π_x for the problem:

$$\Pi_x \in \operatorname*{argmin}_{\Pi \in \mathcal{P}} \|y - \Pi A x\|_2 \,. \tag{1.3}$$

Prove the correctness of your algorithm, and explain when the solution of (1.3) is unique. Your implementation should go into the file `SLR_1_Pi_given_x.m`.

2 Brute Force (5 points)

Exercise 2.1 (least-squares, 2 points)**.** generate data A, y without shuffling, i.e., Π is the identity matrix. Solve the problem (code it in `SLR_2_brute_force_run.m`)

$$\min_{x \in \mathbb{R}^n} \|y - Ax\|_2 \tag{2.1}$$

by least-squares. Now the question:

- How large `sigma` can be for least-squares to have small errors ($\leq 1\%$)?

Exercise 2.2 (brute force, 3 points)**.** Put your implementation in `SLR_2_brute_force.m` and experiments in `SLR_2_brute_force_run.m`:

- Implement the brute force algorithm as described in the slide.
- How large m, n can go for this algorithm to be efficient (terminates in less than half an hour)?
- How large `sigma` can be for this algorithm to have small errors ($\leq 1\%$)?
- How many percentages of shuffled data can the algorithm tolerate?

You are suggested to answer those questions also for the following algorithms.

3 RANSAC (10-15 points)

Exercise 3.1 (RANSAC?)**.** Put the algorithm implementation in `SLR_3_RANSAC.m` and the experiments in `SLR_3_RANSAC_run.m` respectively.

- Implement the RANSAC algorithm. With correct implementation you will have nearly zero error for the noiseless case (3 points).
- Describe this algorithm in the language of RANSAC that you learned from one of the previous lectures.[1] (2 points)
- What is the time complexity of this RANSAC algorithm? (1 point)
- Produce a table containing two rows. The first row consists of the values of m with $m = \{20, 40, \dots, 180, 200\}$, and the second row is the maximal values of n corresponding to the above m, so that the algorithm remains efficient (terminates in less than half an hour). (2 points)
- set m and n to be within the region so that Brute Force and RANSAC are fast, with `sigma` $= 0.01$ fixed. Run the algorithms and compare their errors averaged over 100 trials. Which one produces larger errors? if the errors are not of the same order, explain (not prove) why. (2 points)

Exercise 3.2 (Randomization (*bonus*, 5 points))**.** Assume that there is no noise, and we run the RANSAC algorithm not for all possible $n \times n$ submatrices of A, but only for T iterations ($T \leq$ the number K of all possible $n \times n$ submatrices of A), by randomly choosing the rows of A. If we want the algorithm to find the groundtruth parameter x with probability γ, then how many iterations should we at least run? (if we run 0 iteration, the probability is 0; if we run K iteration, the probability is 1) Support your argument with a proof.

[1]or check: https://en.wikipedia.org/wiki/Random_sample_consensus

4 Alternating Minimization (10-20 points)

Exercise 4.1 (Alternating Minimization)**.** Put respectively the algorithm implementation in `SLR_4_AltMin.m` and the experiments in `SLR_4_AltMin_run.m`.

- Implement the alternating minimization algorithm. With a correct implementation you should observe the value of the objective function

$$\min_{x \in \mathbb{R}^n, \Pi \in \mathcal{P}} \|y - \Pi A x\|_2 \tag{4.1}$$

  decreases in each iteration. Plot a figure to present this phenomenon: a decreasing curve as the iteration goes. (3 points)

- Prove that for random initialization the objective function (4.1) decreases in each iteration. (2 points)

- Instead of running fixed number of iterations before termination, which is ad-hoc, or terminating until it converges, which is equivocal, propose and implement a meaningful termination criterion for this algorithm. (2 points)

- Using as initialization the least-square solution of (2.1), run the algorithms, and plot the errors $\frac{\|\hat{x}-x\|_2}{\|x\|_2}$ as a function of `shuffled_ratio` (100 averages). The `shuffled_ratio` is $\{0\%, 10\%, 20\%, \cdots, 100\%\}$. This curve should be increasing. (3 points)

Exercise 4.2 (Expectation Maximization (*bonus*, 10 points))**.** Describe this alternating minimization algorithm in the language of Expectation Maximization (EM). This is a good opportunity to learn the algorithmic idea behind EM, which I found is usually poorly. Moreover, it is also current research topics to prove the (global) convergence of the EM algorithm in certain problems, e.g., [1]. For this bonus you may find [2] and this post[2] helpful, and you may also consult some machine learning books. A full-points answer should include a clear description of the general EM algorithm in the probabilistic language, followed by a specialization to the problem of linear regression without correspondences.

[2] https://abidlabs.github.io/EM-Algorithm/

5 Robust Regression

Exercise 5.1 (robust regression)**.**

- For the noiseless case and less than 50% percentage of shuffled data, describe a brute force solution to the ℓ_0 norm minimization problem

$$\min_{x\in\mathbb{R}^n} \|y - Ax\|_0 \,, \tag{5.1}$$

  and explain (not prove) why it makes sense to minimize (5.1) for partially shuffled data.[3] (3 points)

- Using CVX[4], solve the problems

$$\min_{x\in\mathbb{R}^n} \|y - Ax\|_1 \,, \text{ and} \tag{5.2}$$

$$\min_{x\in\mathbb{R}^n, e\in\mathbb{R}^m} \frac{1}{m} \left\|y - Ax - \sqrt{m}e\right\|_2^2 + \lambda \|e\|_1 \,, \lambda > 0. \tag{5.3}$$

  For (5.3) you will need to choose the value of λ. Is the solution sensitive to the choice of λ? Verify the correctness of your implementation. (3 points)

- Describe the relation between these two formulations (5.2) and (5.3). (1 point)

- Experimentally verify that the algorithms are scalable by setting m and/or n large enough, e.g., can they run for $m = 10000$? 50000? (1 point)

- Set m, n appropriately and `sigma` $= 0.01$ fixed, and produce error curves as a function of `shuffled_ratio` as in §4. This figure should contain three methods ((5.2), (5.3), alternating minimization) and should have distinguishable appearance (e.g., corret legends). (2 points)

[3] A related justification in the context of compressed sensing: https://arxiv.org/pdf/1211.5231.pdf

[4] http://cvxr.com/cvx/. We assume you can learn to use CVX very quickly.

6 The Algebraic Method (10 points)

Exercise 6.1.

- Recall that for $k \in \{1, 2, ...\}$ the symmetric polynomial p_k is defined as:

$$p_k(z) = z_1^k + z_2^k + \cdots + z_m^k. \tag{6.1}$$

  Write down the formula for $p_2(Ax)$, where x is the variable and

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}. \tag{6.2}$$

  Do you want to compute by hand $p_3(Ax)$? (2 point)

- Implement the required part in `SLR_5_algebraic.m` so that it produces the correct output when noiseless and running `SLR_5_algebraic_run.m` gives nearly zero errors. (4 points)

- (a) For $n = 3$, how long it takes to run for $m = 100, m = 10000$ and $m = 50000$? (b) Check the code `solver_SLR_n2.m` and `solver_SLR_n5.m`. Does it surprise you?[5] (1 point)

- Perform a noiseless experiment with $m = 100$ and produce three figures (for $n = 3, 4, 5$ respectively) as follows: (3 points)

  1. take four intervals $I_1 = [10^{-4}, \infty], I_2 = [10^{-8}, 10^{-4}], I_3 = [10^{-12}, 10^{-8}], I_4 = [-\infty, 10^{-12}]$, and their corresponding counters $c_1 = c_2 = c_3 = c_4 = 0$.
  2. run the algorithm, and compute the error e, if e falls into the interval I_i, then increase the value of c_i by 1.
  3. repeat the above step as many times as you can, e.g., 100000 times.
  4. plot c_1, c_2, c_3, c_4 with x-axis being the intervals and y-axis denotes the value of the counters.
  5. use eight intervals instead of four:)
  6. say something about the results. Do you observe something unusual for $n = 5$?

[5]those codes are generated by something called *automatic solver generation* technique.

References

[1] Jeongyeol Kwon, Wei Qian, Constantine Caramanis, Yudong Chen, and Damek Davis. Global convergence of the em algorithm for mixtures of two component linear regression. In Alina Beygelzimer and Daniel Hsu, editors, *Proceedings of the Thirty-Second Conference on Learning Theory*, volume 99 of *Proceedings of Machine Learning Research*, pages 2055–2110, Phoenix, USA, 25–28 Jun 2019. PMLR.

[2] A. Abid and J. Zou. Stochastic em for shuffled linear regression. *arXiv preprint arXiv:1804.00681*, 2018.