

# COMP9417 - Machine Learning

## Homework 1: Regularized Regression & Statistical Estimators

**Introduction** In this homework we will explore *gradient* based optimization and explain in detail how gradient descent can be implemented. Gradient based algorithms have been crucial to the development of machine learning in the last few decades. The most famous example is the backpropagation algorithm used in deep learning, which is in fact just an application of a simple algorithm known as (stochastic) gradient descent. We will first implement gradient descent from scratch on a deterministic problem (no data), and then extend our implementation to solve a real world regression problem. We then also look at the notions of bias, variance and MSE of statistical estimators.

**Points Allocation** There are a total of 28 marks.

- Question 1 a): 3 marks
- Question 1 b): 1 mark
- Question 1 c): 2 marks
- Question 1 d): 2 marks
- Question 1 e): 5 marks
- Question 1 f): 4 marks
- Question 1 g): 1 mark
- Question 1 h): 2 marks
- Question 1 i): 4 marks
- Question 2 a): 1 mark
- Question 2 b): 1 mark
- Question 2 c): 1 mark
- Question 2 d): 1 mark

### What to Submit

- A **single PDF** file which contains solutions to each question. For each question, provide your solution in the form of text and requested plots. For some questions you will be requested to provide screen shots of code used to generate your answer — only include these when they are explicitly asked for.

- **.py file(s) containing all code you used for the project, which should be provided in a separate .zip file.** This code must match the code provided in the report.
- You may be deducted points for not following these instructions.
- You may be deducted points for poorly presented/formatted work. Please be neat and make your solutions clear. Start each question on a new page if necessary.
- You **cannot** submit a Jupyter notebook; this will receive a mark of zero. This does not stop you from developing your code in a notebook and then copying it into a .py file though, or using a tool such as **nbconvert** or similar.
- We will set up a Moodle forum for questions about this homework. Please read the existing questions before posting new questions. Please do some basic research online before posting questions. Please only post clarification questions. Any questions deemed to be *fishing* for answers will be ignored and/or deleted.
- Please check Moodle announcements for updates to this spec. It is your responsibility to check for announcements about the spec.
- Please complete your homework on your own, do not discuss your solution with other people in the course. General discussion of the problems is fine, but you must write out your own solution and acknowledge if you discussed any of the problems in your submission (including their name(s) and zID).
- As usual, we monitor all online forums such as Chegg, StackExchange, etc. Posting homework questions on these site is equivalent to plagiarism and will result in a case of academic misconduct.
- You may **not** use SymPy or any other symbolic programming toolkits to answer the derivation questions. This will result in an automatic grade of Zero for the relevant question. You must do the derivations manually.

#### When and Where to Submit

- **Due date:** Week 4, Monday March 6th, 2023 by 5pm. Please note that the forum will not be actively monitored on weekends.
- Late submissions will incur a penalty of 5% per day **from the maximum achievable grade**. For example, if you achieve a grade of 80/100 but you submitted 3 days late, then your final grade will be  $80 - 3 \times 5 = 65$ . Submissions that are more than 5 days late will receive a mark of zero.
- Submission must be made on **Moodle**, **no exceptions**.

### Question 1. Gradient Based Optimization

The general framework for a gradient method for finding a minimizer of a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is defined by

$$x^{(k+1)} = x^{(k)} - \alpha_k \nabla f(x_k), \quad k = 0, 1, 2, \dots, \quad (1)$$

where  $\alpha_k > 0$  is known as the step size, or learning rate. Consider the following simple example of minimizing the function  $g(x) = 2\sqrt{x^3 + 1}$ . We first note that  $g'(x) = 3x^2(x^3 + 1)^{-1/2}$ . We then need to choose a starting value of  $x$ , say  $x^{(0)} = 1$ . Let's also take the step size to be constant,  $\alpha_k = \alpha = 0.1$ . Then we have the following iterations:

$$x^{(1)} = x^{(0)} - 0.1 \times 3(x^{(0)})^2((x^{(0)})^3 + 1)^{-1/2} = 0.7878679656440357$$

$$x^{(2)} = x^{(1)} - 0.1 \times 3(x^{(1)})^2((x^{(1)})^3 + 1)^{-1/2} = 0.6352617090300827$$

$$x^{(3)} = 0.5272505146487477$$

$\vdots$

and this continues until we terminate the algorithm (as a quick exercise for your own benefit, code this up and compare it to the true minimum of the function which is  $x_* = -1$ ). This idea works for functions that have vector valued inputs, which is often the case in machine learning. For example, when we minimize a loss function we do so with respect to a weight vector,  $\beta$ . When we take the step-size to be constant at each iteration, this algorithm is known as gradient descent. For the entirety of this question, do not use any existing implementations of gradient methods, doing so will result in an automatic mark of zero for the entire question.

(a) Consider the following optimisation problem:

$$\min_{x \in \mathbb{R}^n} f(x)$$

where

$$f(x) = \frac{1}{2} \|Ax - b\|_2^2 + \frac{\gamma}{2} \|x\|_2^2,$$

and where  $A \in \mathbb{R}^{m \times n}$ ,  $b \in \mathbb{R}^m$  are defined as

$$A = \begin{bmatrix} 1 & 2 & 1 & -1 \\ -1 & 1 & 0 & 2 \\ 0 & -1 & -2 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 2 \\ -2 \end{bmatrix},$$

and  $\gamma$  is a positive constant. Run gradient descent on  $f$  using a step size of  $\alpha = 0.1$  and  $\gamma = 0.2$  and starting point of  $x^{(0)} = (1, 1, 1, 1)$ . You will need to terminate the algorithm when the following condition is met:  $\|\nabla f(x^{(k)})\|_2 < 0.001$ . In your answer, clearly write down the version of the gradient steps (1) for this problem. Also, print out the first 5 and last 5 values of  $x^{(k)}$ , clearly indicating the value of  $k$ , in the form:

$$k = 0, \quad x^{(k)} = [1, 1, 1, 1]$$

$$k = 1, \quad x^{(k)} = \dots$$

$$k = 2, \quad x^{(k)} = \dots$$

$\vdots$

*What to submit: an equation outlining the explicit gradient update, a print out of the first 5 ( $k = 5$  inclusive) and last 5 rows of your iterations. Use the round function to round your numbers to 4 decimal places. Include a screen shot of any code used for this section and a copy of your python code in solutions.py.*

In the next few parts, we will use gradient methods explored above to solve a real machine learning problem. Consider the CarSeats data provided in `CarSeats.csv`. It contains 400 observations with each observation describing child car seats for sale at one of 400 stores. The features in the data set are outlined below:

- Sales: Unit sales (in thousands) at each location
- CompPrice: Price charged by competitor at each location
- Income: Local income level (in thousands of dollars)
- Advertising: advertising budget (in thousands of dollars)
- Population: local population size (in thousands)
- Price: price charged by store at each site
- ShelfLoc: A categorical variable with Bad, Good and Medium describing the quality of the shelf location of the car seat
- Age: Average age of the local population
- Education: Education level at each location
- Urban: A categorical variable with levels No and Yes to describe whether the store is in an urban location or in a rural one
- US: A categorical variable with levels No and Yes to describe whether the store is in the US or not.

The target variable is Sales. The goal is to learn to predict the amount of Sales as a function of a subset of the above features. We will do so by running Ridge Regression (Ridge) which is defined as follows

$$\hat{\beta}_{\text{Ridge}} = \arg \min_{\beta} \frac{1}{n} \|y - X\beta\|_2^2 + \phi \|\beta\|_2^2,$$

where  $\beta \in \mathbb{R}^p$ ,  $X \in \mathbb{R}^{n \times p}$ ,  $y \in \mathbb{R}^n$  and  $\phi > 0$ .

- (b) We first need to preprocess the data. Remove all categorical features. Then use `sklearn.preprocessing.StandardScaler` to standardize the remaining features. Print out the mean and variance of each of the standardized features. Next, center the target variable (subtract its mean). Finally, create a training set from the first half of the resulting dataset, and a test set from the remaining half and call these objects `X_train`, `X_test`, `Y_train` and `Y_test`. Print out the first and last rows of each of these.

*What to submit: a print out of the means and variances of features, a print out of the first and last rows of the 4 requested objects, and some commentary. Include a screen shot of any code used for this section and a copy of your python code in solutions.py.*

- (c) It should be obvious that a closed form expression for  $\hat{\beta}_{\text{Ridge}}$  exists. Write down the closed form expression, and compute the exact numerical value on the training dataset with  $\phi = 0.5$ .

*What to submit: Your working, and a print out of the value of the ridge solution based on  $(X_{\text{train}}, Y_{\text{train}})$ . Include a screen shot of any code used for this section and a copy of your python code in solutions.py.*

We will now solve the ridge problem but using numerical techniques. As noted in the lectures, there are a few variants of gradient descent that we will briefly outline here. Recall that in gradient descent our update rule is

$$\beta^{(k+1)} = \beta^{(k)} - \alpha_k \nabla L(\beta^{(k)}), \quad k = 0, 1, 2, \dots,$$

where  $L(\beta)$  is the loss function that we are trying to minimize. In machine learning, it is often the case that the loss function takes the form

$$L(\beta) = \frac{1}{n} \sum_{i=1}^n L_i(\beta),$$

i.e. the loss is an average of  $n$  functions that we have labelled  $L_i$ , and each  $L_i$  depends on the data only through  $(x_i, y_i)$ . It then follows that the gradient is also an average of the form

$$\nabla L(\beta) = \frac{1}{n} \sum_{i=1}^n \nabla L_i(\beta).$$

We can now define some popular variants of gradient descent.

- (i) Gradient Descent (GD) (also referred to as batch gradient descent): here we use the full gradient, as in we take the average over all  $n$  terms, so our update rule is:

$$\beta^{(k+1)} = \beta^{(k)} - \frac{\alpha_k}{n} \sum_{i=1}^n \nabla L_i(\beta^{(k)}), \quad k = 0, 1, 2, \dots$$

- (ii) Stochastic Gradient Descent (SGD): instead of considering all  $n$  terms, at the  $k$ -th step we choose an index  $i_k$  randomly from  $\{1, \dots, n\}$ , and update

$$\beta^{(k+1)} = \beta^{(k)} - \alpha_k \nabla L_{i_k}(\beta^{(k)}), \quad k = 0, 1, 2, \dots$$

Here, we are approximating the full gradient  $\nabla L(\beta)$  using  $\nabla L_{i_k}(\beta)$ .

- (iii) Mini-Batch Gradient Descent: GD (using all terms) and SGD (using a single term) represents the two possible extremes. In mini-batch GD we choose batches of size  $1 < B < n$  randomly at each step, call their indices  $\{i_{k_1}, i_{k_2}, \dots, i_{k_B}\}$ , and then we update

$$\beta^{(k+1)} = \beta^{(k)} - \frac{\alpha_k}{B} \sum_{j=1}^B \nabla L_{i_j}(\beta^{(k)}), \quad k = 0, 1, 2, \dots,$$

so we are still approximating the full gradient but using more than a single element as is done in SGD.

- (d) The ridge regression loss is

$$L(\beta) = \frac{1}{n} \|y - X\beta\|_2^2 + \phi \|\beta\|_2^2.$$

Show that we can write

$$L(\beta) = \frac{1}{n} \sum_{i=1}^n L_i(\beta),$$

and identify the functions  $L_1(\beta), \dots, L_n(\beta)$ . Further, compute the gradients  $\nabla L_1(\beta), \dots, \nabla L_n(\beta)$

*What to submit: your working, either typed or handwritten.*

- (e) In this question, you will implement (batch) GD from scratch to solve the ridge regression problem. Use an initial estimate  $\beta^{(0)} = 1_p$  (the vector of ones), and  $\phi = 0.5$  and run the algorithm for 1000 epochs (an epoch is one pass over the entire data, so a single GD step). Repeat this for the following step sizes:

$$\alpha \in \{0.000001, 0.000005, 0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01\}$$

To monitor the performance of the algorithm, we will plot the value

$$\Delta^{(k)} = L(\beta^{(k)}) - L(\hat{\beta}),$$

where  $\hat{\beta}$  is the true (closed form) ridge solution derived earlier. Present your results in a  $3 \times 3$  grid plot, with each subplot showing the progression of  $\Delta^{(k)}$  when running GD with a specific step-size. State which step-size you think is best and let  $\beta^{(K)}$  denote the estimator achieved when running GD with that choice of step size. Report the following:

- The train MSE:  $\frac{1}{n} \|y_{\text{train}} - X_{\text{train}}\beta^{(K)}\|_2^2$
- The test MSE:  $\frac{1}{n} \|y_{\text{test}} - X_{\text{test}}\beta^{(K)}\|_2^2$

*What to submit: a single plot, the train and test MSE requested. Include a screen shot of any code used for this section and a copy of your python code in solutions.py.*

- (f) We will now implement SGD from scratch to solve the ridge regression problem. Use an initial estimate  $\beta^{(0)} = 1_p$  (the vector of ones) and  $\phi = 0.5$  and run the algorithm for 5 epochs (this means a total of  $5n$  updates of  $\beta$ , where  $n$  is the size of the training set). Repeat this for the following step sizes:

$$\alpha \in \{0.000001, 0.000005, 0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.006, 0.02\}$$

Present an analogous  $3 \times 3$  grid plot as in the previous question. Instead of choosing an index randomly at each step of SGD, we will cycle through the observations in the order they are stored in  $X_{\text{train}}$  to ensure consistent results. Report the best step-size choice and the corresponding train and test MSEs. In some cases you might observe that the value of  $\Delta^{(k)}$  jumps up and down, and this is not something you would have seen using batch GD. Why do you think this might be happening?

*What to submit: a single plot, the train and test MSE requested and some commentary. Include a screen shot of any code used for this section and a copy of your python code in solutions.py.*

- (g) Based on your GD and SGD results, which algorithm do you prefer? When is it a better idea to use GD? When is it a better idea to use SGD? *What to submit: Some commentary.*

(h) Consider now the Lasso problem:

$$\hat{\beta}_{\text{Lasso}} = \arg \min_{\beta} \frac{1}{2n} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1.$$

Using sklearn, what are the solutions for ridge with  $\phi = 0.5$  and Lasso for  $\lambda = .15$ . Note, you need to be careful here about how sklearn parameterizes the ridge/lasso formulas, they are slightly different to how we have defined them here. Can you explain the differences between the estimate of  $\beta_4$  under each of the two models? *What to submit: some commentary. Include a screen shot of any code used for this section and a copy of your python code in solutions.py.*

- (i) Run ridge regression with  $\phi = \{0.01, 0.1, 0.5, 1, 1.5, 2, 5, 10, 20, 30, 50, 100, 200, 300\}$ . Create a plot with  $x$ -axis representing  $\log(\phi)$ , and  $y$ -axis representing the value of the coefficient for each feature in each of the fitted ridge models. In other words, the plot should describe what happens to each of the coefficients in your model for the different choices of  $\phi$ . For this problem you are permitted to use the sklearn implementation of Ridge regression to run the models and extract the coefficients, and base matplotlib/numpy to create the plots but no other packages are to be used to generate this plot. Repeat the analysis for Lasso with  $\lambda = \{0.01, 0.1, 0.5, 1, 1.5, 2, 5, 10, 20, 30, 50, 100, 200, 300\}$ . Comment on the differences between the plots, and what happens as the regularization parameters  $\phi, \lambda$  increase. *What to submit: Two plots, some commentary. Include a screen shot of any code used for this section and a copy of your python code in solutions.py*

## Question 2

Let  $X_1, \dots, X_n$  be i.i.d. random variables with mean  $\mu$  and variance  $\sigma^2$ . Define the estimator  $T = \sum_{i=1}^n a_i X_i$  for some constants  $a_1, \dots, a_n$ .

- (a) What condition must the  $a_i$ 's satisfy to ensure that  $T$  is an unbiased estimator of  $\mu$ ? Unbiased means that  $\mathbb{E}T = \mu$ .

*What to submit: your working out, either typed or handwritten, and some commentary.*

- (b) Under the condition identified in the previous part, which choice of the  $a_i$ 's will minimize the MSE of  $T$ ? Does this choice of  $a_i$ 's surprise you? Provide some brief discussion. **Hint: this is a constrained minimization problem. If you need a refresher on how to approach these kinds of problems, see the following short note.**

*What to submit: your working out, either typed or handwritten, and some commentary.*

- (c) Suppose that instead, we let  $a_i = \frac{1+b}{n}$  for  $i = 1, \dots, n$ , where  $b$  is some constant. Find the value of  $b$  (in terms of  $\mu$  and  $\sigma^2$ ) which minimizes the MSE of  $T$ . How does the answer here compare to the estimator found in the previous part? How do the two compare as the sample size  $n$  increases? Are there any obvious issues with using the result of this question in practice?

*What to submit: your working out, either typed or handwritten, and some commentary.*

- (d) Suppose now that you are told that  $\sigma^2 = \mu^2$ . For the choice of  $b$  identified in the previous part, find the MSE of the estimator in this setting. How does the MSE compare to the MSE of the sample average  $\bar{X}$ ? (Recall that  $\text{mse}(\bar{X}) = \text{var}(\bar{X}) = \sigma^2/n = \mu^2/n$ ). Further, explain whether or not you can use this choice of  $b$  in practice.

*What to submit: your working out, either typed or handwritten, and some commentary.*