CS4100/CS5100 CW Assignment 1

Updated on October 19, 2023

This assignment must be submitted by **Friday, 3 November 2023, at 10 am**. Feedback will be possibly provided by Thursday, 23 November 2023, EOB.

How to submit

You will submit a single pdf file, called `dataAnalysisCW1.pdf`, using this link. All submissions will be graded anonymously. Your name should not appear anywhere in your files.

> **Note:** All the work you submit should be solely your work. Coursework submissions are routinely checked for this.

You can overwrite your submission as often as you like. Only the last version submitted will be marked. Submissions uploaded after the deadline may be accepted but will be automatically recorded as *late*. You *cannot* apply to College Extensions for this assignment.

Your report

`dataAnalysisCW1.pdf` should be a polished and well-structured document. Divide the content into four sections, one per each of the tasks described in Section 2. Each section will contain

- a brief description of your programs,
- all required plots and numerical results,
- your answers to the questions, and
- possible additional comments.

Create an appendix for copying the Python implementation of the auxiliary functions. [1] You must follow the templates provided in Section 3. You **cannot use any existing implementations** of logistic regression or gradient descent. The only Python libraries you are allowed to import are[2],

- `numpy` and
- `matplotlib.pyplot`.

[1] Do not include your entire program in the report.

[2] `scikit-learn` is strictly forbidden.

Assessed learning outcomes

- Develop, validate, and evaluate a simple machine-learning algorithm.
- Implement a gradient descent algorithm and run it on a given data set to train a machine learning model.
- Use the trained models to perform data analysis tasks.
- Write a detailed data analysis report based on numerical simulations and draw statistical conclusions from the results.

Marking criteria

To be awarded full marks, you should obtain good numerical results, explain what you did in detail, and correctly answer all theoretical questions. Also, you need to justify all your answers and comments and show you understand your numerical results, even if they differ from expected. We will evaluate each task separately and may award a fraction of the indicated marks if your answers are incomplete. Not attempted tasks will be awarded 0 marks.

1 Introduction

This assignment is about training a logistic regression model to complete binary classification tasks. Read this introduction and Sections 4.1 - 4.3 of An Introduction To Statistical Learning (Hastie et al.) before starting.

1.1 Logistic regression

Assume you have a data set

$$D = \{(X_n, Y_n) \in \mathbb{R}^d \otimes \{\texttt{yes}, \texttt{no}\}\}_{n=1}^N \tag{1}$$

$d \in \mathbb{N}_+$. Interpret X_n and Y_n as the attribute and label of the n-th object. In this case, a logistic regression model,

$$f(X_n, \lambda) = \sigma\left(\lambda_0 + \sum_{i=1}^{d} \lambda_i X_{ni}\right), \quad \sigma(t) = \frac{1}{1+e^{-t}} \tag{2}$$

$X_n \in \mathbb{R}^d$ and $\lambda \in \mathbb{R}^{d+1}$, approximates the *conditional probability* of observing $Y_n = \texttt{yes}$, i.e.

$$f(X_n, \lambda) \approx \text{Prob}(Y_n = \texttt{yes}|X_n), \quad Y_n \in \{\texttt{yes}, \texttt{no}\} \tag{3}$$

In a computer implementation, it is convenient to use 0 and 1 as class labels, e.g. to represent `yes` with 1 and `no` with 0.

1.2 Gradient Descent (GD)

To estimate the model free parameter, λ, you can *maximize* the probability of observing the labels of D,

$$\mathcal{L}(\lambda) = \text{Prob}(Y_1, \dots Y_N | X_1, \dots, X_N, \lambda)$$

or *minimize* its negative logarithm,

$$\ell(\lambda) = -\log \mathcal{L}(\lambda) \tag{4}$$

If D is a collection of i.i.d. random variables, $\mathcal{L}(\lambda)$ factorizes and

$$\ell(\lambda) = -\sum_{n=1}^{N} \log \text{Prob}(Y_n | X_n, \lambda) \tag{5}$$

$$= -\sum_{n=1}^{N} \log \left(\mathbf{1}(Y_n = \texttt{yes}) f(X_n, \lambda) + \mathbf{1}(Y_n = \texttt{no})(1 - f(X_n, \lambda)) \right) \tag{6}$$

In this assignment, you will focus on minimizing $\ell(\lambda)$ with GD. Given an *arbitrary* initialization, $\lambda^{(0)} \in \mathbb{R}^{d+1}$, let

$$\lambda^{(t+1)} = \lambda^{(t)} - \eta \nabla \ell(\lambda^{(t)}), \quad t = 0, \dots, T_{epochs} \tag{7}$$

$\eta \in \mathbb{R}_+$ and T_{epochs} are the algorithm learning rate and total number of iterations. To compute the t-th gradient descent update (7), you need to evaluate (in the order)

1. the model, $f(X_n, \lambda)$, for $\lambda = \lambda^{(t)}$ and $n = 1, \dots, N$, (*forward pass*) and
2. the gradient of the objective function,

$$\nabla \ell(\lambda) = -\sum_{n=1}^{N} \frac{(\mathbf{1}(Y_n = \texttt{yes}) - \mathbf{1}(Y_n = \texttt{no})) \nabla f(X_n, \lambda)}{\mathbf{1}(Y_n = \texttt{yes}) f(X_n, \lambda) + \mathbf{1}(Y_n = \texttt{no})(1 - f(X_n, \lambda))} \tag{8}$$

$$\nabla f(X_n, \lambda) = \sigma' \left(\lambda_0 + \sum_{i=1}^{d} \lambda_i X_{ni} \right) [1, X_{n1}, \dots, X_{nd}]^T \tag{9}$$

$$\sigma'(t) = \frac{d}{dt} \sigma(t) = \sigma(t)(1 - \sigma(t)) \tag{10}$$

at $\lambda = \lambda^{(t)}$.

You may refer to these steps as the *forward* and *backward* passes. In practice, you need to store the model evaluations obtained in Step 1 to compute the gradient of the objective in Step 2.

2 Tasks

2.1 Q1 - Synthetic data (40 marks)

Set $d = 10$, $N = 100$, and use `generateData` defined in 3.1 to create a binary classificationdata set, $D = \{(X_n, Y_n)\}_{n=1}^{N}$. Split D into a training set and a test set, $D_{train} = \{(X_n, Y_n)\}_{n=1}^{N_{train}}$ and $D_{test} = \{(X_n, Y_n)\}_{n=N_{train}+1}^{N}$, $N_{train} = 50$. Complete the templates of `train` and all auxiliary functions given in Section 3.2. To verify your implementation, set $\lambda = [0.01, 0.01, \dots]^T$ and $\eta = 0.001$, run `train`, and check that `obj` contains decreasing values. If everything works correctly, create and store five distinct Gaussian initialization[3],

$$\lambda_m^{(0)} \sim \mathcal{N}(0,1)^{d+1}, \quad m = 1, \dots, 5$$

Run your algorithm five times using the obtained random initializations, λ_m, $m = 1, \dots, 5$, $T_{epochs} = 100$, and $\eta = 0.01$. Let $\lambda_m = \lambda_m^{T_{epochs}}$ be the parameter estimate you obtain from $\lambda_m^{(0)}$, $m = 1, \dots 5$. For each λ_m, $m = 1, \dots, 5$, compute the Error Rate (ER) of $f(X_n, \lambda)$ on D_{test},

$$\mathrm{ER}(\lambda, D_{test}) = (N - N_{train})^{-1} \sum_{n=N_{train}+1}^{N} (Y_n - \hat{Y}_n)^2, \quad \hat{Y}_n = \mathbf{1}(f(X_n, \lambda) > \frac{1}{2}) \tag{11}$$

To evaluate the model $\mathrm{ER}(\lambda, D_{test})$, complete and use `ER` given in Section 3.3.

In the report. Plot the objective function, $\ell(\lambda_m^{(t)})$, $t = 1, \dots, T_{epochs}$, for all $m = 1, \dots, 5$ on a single figure. Write the initialization indexes, $m = 1, \dots, 5$, and the model ERs in the legend. Figure 1 is an example of what your plot should look like. What conclusions can you draw from the obtained results? Does the model performance depend on the initialization? Can you say anything about the global or local optimality of your estimates? Copy your implementation of `objective`, `gradient`, and `train` in the report appendix.

2.2 Q2 - Cars data set (20 marks)

Download CarsDataSetForCW1.csv, move it to your working directory, and load the data to your program[4]. Check that each data object, Z_n, has the expected attributes, `mpg`, `cylinders`, `displacement`, `horsepower`, `weight`, `acceleration`, `year`, `origin`, and `name`, and that the data set contains $N = 397$ objects. Before proceeding, you need to preprocess the data. First of all, make the data compatible with the functions you implemented in Section 2.1 by transforming the `Pandas` Data Frame into a `Numpy` Array[5]. Then, create a binary-classification

[3] Python command: `tryInit = [np.random.randn(len(X[0]))forminrange(5)]`.
[4] Python command: `data = pd.read_csv("CarsDataSetForCW1.csv", sep =',')`.
[5] Python command: `Z = data[['mpg','cylinders',...,'year']].to_numpy()`.

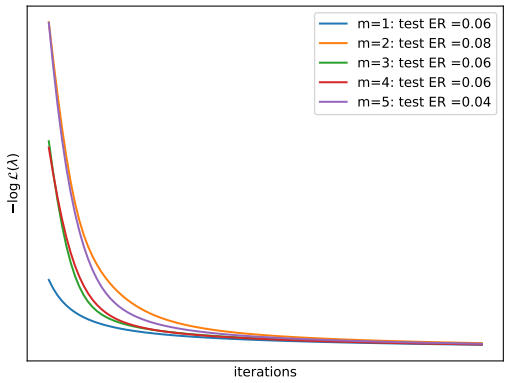

Figure 1: Negative log-likelihood minimization for five random initializations. All models are trained on the same synthetic data set with $\eta = 0.01$ and $T_{epoch} = 100$.

task by defining N new binary variables,[6],

$$Y_{n\texttt{high}} = \begin{cases} 1 & \text{if } Z_{n\texttt{mpg}} \geq M \\ 0 & \text{otherwise} \end{cases}, \quad M = N^{-1} \sum_{n=1}^{N} Z_{n\texttt{mpg}}$$

Let $Y_{n\texttt{high}}$ and

$$X_n = [Z_{n\texttt{cylinders}}, Z_{n\texttt{displacement}}, Z_{n\texttt{horsepower}}, Z_{n\texttt{weight}}, Z_{n\texttt{acceleration}}, Z_{n\texttt{year}}]^T$$

be the label and the attribute of the n-th object[7] and split the data into a training set and a test set of the same size, $D_{train} = \{(X_n, Y_n)\}_{n=1}^{N_{train}}$ and $D_{test} = \{(X_n, Y_n)\}_{N_{train}+1}^{N}$, $N_{train} \approx \frac{N}{2}$. Finally, normalize the attributes in D_{train} and D_{test}. To fairly test your models, use the training set normalizing constants,

$$\mu_i = N_{train}^{-1} \sum_{n=1}^{N_{train}} X_{ni}, \quad \sigma_i = \sqrt{N_{train}^{-1} \sum_{n=1}^{N_{train}} (X_{ni} - \mu_i)^2} \tag{12}$$

to rescale both the training and test attributes,

$$X_{ni} = \frac{X_{ni} - \mu_i}{\sigma_i}, \quad n = 1, \ldots, N, \quad i = 1, \ldots, 6 \tag{13}$$

After preprocessing the data, run the learning algorithm you implemented in Section 2.1 on D_{train}, with $T_{epocs} = 100$, a fixed random initialization, $\lambda^{(0)} \sim$

[6]Python command: `Y = 1. * (Z[:, : 1] >= np.mean(Z[:, : 1]))`.
[7]Python command: `X, Y = Z[:, 1 :], Y`.

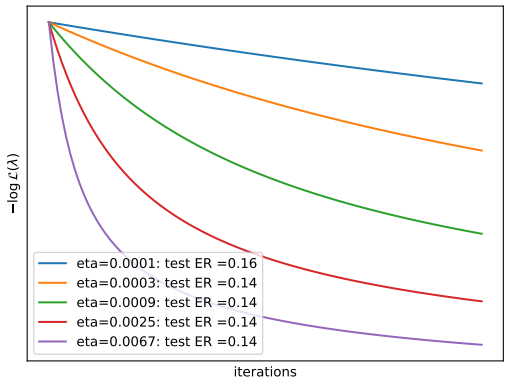

Figure 2: Negative log-likelihood minimization for five different learning rates. All models are trained from the same initialization on half of the Cars data set, with $\eta_m = e^{m-10}$, $m = 1, \ldots, 5$ and $T_{epoch} = 100$.

$\mathcal{N}(0, 0.01)$[8], and

$$\eta_m = e^{m-10}, \quad m = 1, \ldots, 5 \tag{14}$$

Let λ_m be the parameter estimate you obtain using η_m and compute the ER of $f(X, \lambda_m)$ on D_{test}.

In the report. Plot the objective function, $\ell(\lambda_m^{(t)})$, $t = 1, \ldots, T_{epochs}$, for all $m = 1, \ldots, 5$ on a single figure. Write η_m and the model ERs in the legend. Figure 2 is an example of what your plot should look like. What is the best learning rate for this task? What happens if you consider higher values of η, e.g. $\eta = 1$? Briefly explain why, in general, you expect $\ell(\lambda^{(t+1)}) \leq \ell(\lambda^{(t)})$ for all $t = 1, \ldots, T_{epochs}$ if η is small enough and $\lambda^{(t+1)} = \lambda^{(t)} - \eta \nabla \ell(\lambda^{(t)})$.

2.3 Q3 - Nominal variables (20 marks)

Extend f to account for the nominal variables $Z_{n\texttt{origin}}$. The new task is to predict $Y_{n\texttt{high}}$, given the numerical attributes, X_n defined in Section2.2), and $Z_{n\texttt{origin}}$. You need to create an appropriate set of *dummy* variables even if, in `CarsDataSetForCW1.csv`, the values of $Z_{n\texttt{origin}}$ are integers. After creating the new variables, append them to X_n. As above, split the data into a training set and a test set of the same size. Re-renormalize the attributes as in Section 2.2. Finally, run your algorithm with five different learning rates as you did in Section 2.2.

[8] Python command: `lambda0 = 0.01 * np.random.randn(len(X[0]))`.

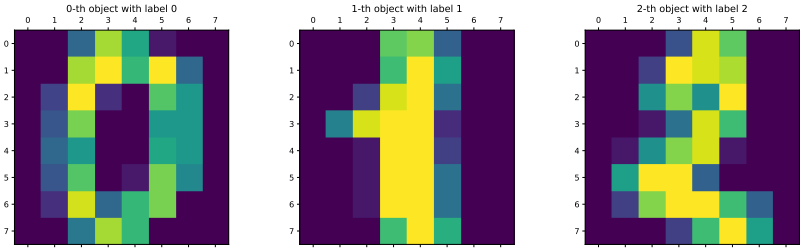

Figure 3: Attributes and labels of the first three objects in the Digit data set.

In the report. Produce a plot similar to the one you obtained in Section 2.2. Write the model learning rates and the test ERs in the legend. Compare the results with Section 2.2. Does including $Z_{n\texttt{origin}}$ improve the model performance? More generally, do better performances on the training set imply better performances on the test set? Copy the Python code you wrote to create the dummy variables in the report appendix.

2.4 Q4 - Digits data set (20 marks)

Download DigitsDataSetForCW1.csv, move it to your working directory, and load it into a `Numpy` Array[9]. Use the first 64 entries of each row as the object attributes, X_n, and the last entry as the object label[10]. Check that the loaded data set consists of 1797 objects divided into 10 classes[11]. Figure 3 shows the attribute and label of the first 3 objects[12]. Create four binary-classification data sets, D_{01}, D_{12}, D_{23}, and D_{30}, by selecting the objects with labels 0 and 1, 1 and 2, 2 or 3, and 3 and 0. After creating the data sets, split them into a training set and a test set of equal size, $D_{train} = \{(X_n, Y_n)\}_{n=1}^{N_{train}}$ and $D_{test} = \{(X_n, Y_n)\}_{N_{train}+1}^{N}$, $N_{train} \approx \frac{N}{2}$. Note that N may vary depending on the selected digits. Normalize the attributes of D_{train} and D_{test} as explained in Section 2.2. Use the data sets to train and test four logistic regression models, $f(X_n, \lambda_{ij})$, $(i,j) \in \{(0,1),(1,2),(2,3),(3,0)\}$. As in Section 2.2, initialize the learning algorithm using a fixed random estimate $\lambda_0 \sim \mathcal{N}(0,\sigma)$[65], with $\sigma = 0.0001$[13], i.e. set $\lambda_{ij}^{(0)} = \lambda_0$ for all $(i,j) \in \{(0,1),(1,2),(2,3),(3,0)\}$, and $\eta = 0.0001$ and $T_{epochs} = 50$ for all models. After training the models, compute their ER on the corresponding test sets.

In the report. Plot the objective function, $\ell(\lambda_{ij}^{(t)})$, $t = 1,\ldots,T_{epochs}$, for all $(i,j) \in \{(0,1),(1,2),(2,3),(3,0)\}$ on a single figure. Write the digit pairs,

[9] Python command: `dataset` $=$ `pd.read_csv(file, sep =',').to_numpy()`.
[10] Python command: `[X, Y]` $=$ `dataset[:, : -1], dataset[:, -1 :]`.
[11] The number of objects per class should be 178, 182, 177, 183, 181, 182, 181, 179, 174, and 180.
[12] Python command: `plt.matshow(np.reshape(X[k], [8, 8]))`.
[13] Python command: `lambda0` $=$ `0.0001` $*$ `np.random.randn(65)`

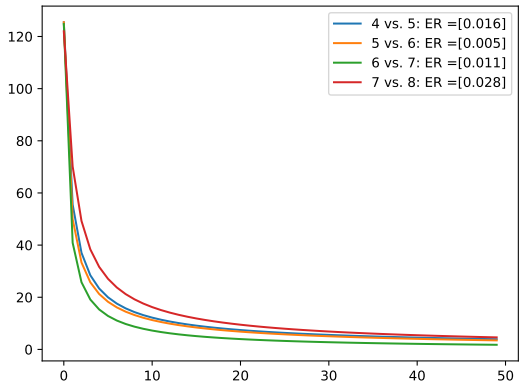

Figure 4: Log-likelihood minimization for four logistic regression models. The models are trained on subsets of the DigitsDataSetForCW1.csvdata sets to discriminate between the specified digits.

(i, j), and the model test ERs in the legend. Figure 4 is an example of what your plot should look like. Can you draw any conclusion from this experiment? What are the hardest and the simplest discrimination tasks? Are you surprised a logistic regression model obtains such good results? Do you think the models are overfitting the training data? Copy the commands you used to create the binary classification data sets in the appendix.

3 Supporting Material

3.1 Data generating function

```
def generateData(d, N, noise = 0.1):
    np.random.seed(12345)
    X = np.concatenate([np.ones([N, 1]), np.random.randn(N, d)
        ], axis=1)
    lambdaTrue = - 1 + 2 * np.tan(np.random.randn(len(X[0])))
    Y = 1 * ((X @ lambdaTrue + noise * np.random.randn(N)) > 0)
    D = X, Y
    return D
```

3.2 GD templates

```
def model(x, par):
    # evaluate the logistic regression model, see (2)
    s = x @ ...
```

```
    return 1/(1 + ...)

def dModel(x, par):
    # compute the gradient of f at x, see (9)
    f = ...
    return f * (1 - f) * ...

def objective(par, data):
    # compute the objective function, see (5) and Algorithm 3
    ell = 0
    X, Y = data
    for n in range(...):
        x, y = X[n], Y[n]
        f = ...
        s = - np.log(...)
        ell = ell + ...
    return ell

def gradient(par, data):
    # compute the gradient of the objective function, see (8)
        and Algorithm 3
    grad = np.zeros(len(par))
    X, Y = data
    for n in range(...):
        x, y = X[n], Y[n]
        f = ...
        s =  - (...)/(y * f + (1 - y) * (1 - f))
        grad  = grad + s * ...
    return grad

def train(par0, eta, T, data):
    # training, see Algorithm 1
    par = par0
    obj = []
    for t in ...:
        ell = ...
        obj.append(ell)
        grad = ...
        par = par - ...
    return par, obj
```

3.3 ER template

```
def ER(par, data):
    # compute the error rate, see (11)
    ER = 0
    X, Y = data
    for n in range(...):
        x, y = X[n], Y[n]
```

```
7           yHat = ...
8           s = 1 * (y != yHat)
9           ER = ER + ...
10      return ER/len(data)
```

3.4 Pseudo code of the GD functions

Algorithm 1 `train`

input: initial parameter, $\lambda^{(0)} \in \mathbb{R}^d$, learning rate, $\eta \in \mathbb{R}_+$, number of epochs, $T_{epochs} \in \mathbb{N}_+$, training data set $D = \{(X_n, Y_n)\}_{n=1}^N$

let $\lambda = \lambda^{(0)}$
let `obj` be an empty list
for $t = 1, \ldots, T_{epochs}$ **do**
  let $\ell = \texttt{objective}(\lambda, D)$
  let $\nabla\ell = \texttt{gradient}(\lambda, D)$
  append ℓ to `obj`
  let $\lambda = \lambda - \eta\nabla\ell$
end for

output: trained parameter, $\lambda^{(T)} = \lambda$, list of objective values for $t = 1, \ldots, T_{epochs}$, `obj`

Algorithm 2 `objective`

input: parameter, λ, training data set $D = \{(X_n, Y_n) \in \mathbb{R}^d \times \mathcal{Y}\}_{n=1}^N$

let $\ell = 0$
for $n \in \{1, \ldots, N\}$ **do**
  let $f = \sigma(\lambda_0 + \sum_{i=1}^d \lambda_i X_{ni}) = \texttt{model}(X_n, \lambda)$
  let $\ell = \ell - \log(\mathbf{1}(Y_n = \texttt{yes})f + \mathbf{1}(Y_n = \texttt{no})(1 - f))$
end for

output: negative log-likelihood, ℓ.

Algorithm 3 `gradient`

input: parameter, λ, training data set, $D = \{(X_n, Y_n) \in \mathbb{R}^d \times \mathcal{Y}\}_{n=1}^N$

let $\nabla\ell = [0, \ldots, 0]^T \in \mathbb{R}^{d+1}$

for $n \in \{1, \ldots, N\}$ **do**

let $f = \sigma(\lambda_0 + \sum_{i=1}^d \lambda_i X_{ni}) = \texttt{model}(X_n, \lambda)$

let $\nabla f = \texttt{dModel}(X_n, \lambda)$

let $g = -\frac{(\mathbf{1}(Y_n=\texttt{yes}) - \mathbf{1}(Y_n=\texttt{no}))\nabla f}{\mathbf{1}(Y_n=\texttt{yes})f + \mathbf{1}(Y_n=\texttt{no}(1-f)}$

let $\nabla\ell = \nabla\ell + g$

end for

output: gradient of the negative log-likelihood, $\nabla\ell$.