COMP9417 - Machine Learning
Homew[illegible]ical Implementation of Logistic Regression

Introduction In homework 1, we considered Gradient Descent (and coordinate descent) for minimizing a regularized loss function. In this homework, we consider an alternative method known as Newton's algorithm. We will first run Newton's algorithm on a simple toy problem, and then implement it from scratch on a real data classification problem. We also look at the dual version of logistic regression.

Points Allocation There are a total of 30 marks.

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

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 r[illegible]o some basic research online before posting questions. Please only post clarifi[illegible] questions deemed to be *fishing* for answers will be ignored and/or deleted.

- Please check M[illegible]or updates to this spec. It is your responsibility to check for announcements[illegible]

- Please complete[illegible]ur own, do not discuss your solution with other people in the course. General[illegible]lems 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 sites is equivalent to plagiarism and will result in a case of academic misconduct.

When and Where to Submit

- Due date: Week 7, Monday **March 25th**, 2024 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 done through Moodle, no exceptions.

Question 1. Introduction to Newton's Method

Note: throughout this question do not use any existing implementations of any of the algorithms discussed unless explicitly asked to in the question. Using existing implementations can result in a grade of zero for the entire question. In homework 1 we studied gradient descent (GD), which is usually referred to as a first order method. Here, we study an alternative algorithm known as Newton's algorithm, which [illegible] as a second order method. Roughly speaking, a second order method makes us[illegible]ond derivatives. Generally, second order methods are much more accurate tha[illegible]n a twice differentiable function $g : \mathbb{R} \to \mathbb{R}$, Newton's method generates a seque[illegible]cording to the following update rule:

$$x^{(k+1)} = x^{(k)} - \frac{g'(x^{(k)})}{g''(x^{(k)})}, \qquad k = 0, 1, 2, \dots, \tag{1}$$

For example, cons[illegible] $= \frac{1}{2}x^2 - \sin(x)$ with initial guess $x^{(0)} = 0$. Then

$$g'(x) = x - \cos(x), \qquad \text{and} \qquad g''(x) = 1 + \sin(x),$$

and so we have the following iterations:

$$x^{(1)} = x^{(0)} - \frac{x^{(0)} - \cos(x^0)}{1 + \sin(x^{(0)})} = 0 - \frac{0 - \cos(0)}{1 + \sin(0)} = 1$$

$$x^{(2)} = x^{(1)} - \frac{x^{(1)} - \cos(x^1)}{1 + \sin(x^{(1)})} = 1 - \frac{1 - \cos(1)}{1 + \sin(1)} = 0.750363867840244$$

$$x^{(3)} = 0.739112890911362$$

$$\vdots$$

and this continues until we terminate the algorithm (as a quick exercise for your own benefit, code this up, plot the function and each of the iterates). We note here that in practice, we often use a different update called the *dampened* Newton method, defined by:

$$x^{(k+1)} = x^{(k)} - \alpha \frac{g'(x_k)}{g''(x_k)}, \qquad k = 0, 1, 2, \dots. \tag{2}$$

Here, as in the case of GD, the step size α has the effect of 'dampening' the update. Consider now the twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$. The Newton steps in this case are now:

$$x^{(k+1)} = x^{(k)} - (H(x^{(k)}))^{-1} \nabla f(x^{(k)}), \qquad k = 0, 1, 2, \dots, \tag{3}$$

where $H(x) = \nabla^2 f(x)$ is the Hessian of f. Heuristically, this formula generalized equation (1) to functions with vector inputs since the gradient is the analog of the first derivative, and the Hessian is the analog of the second derivative.

(a) Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ defined by

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2.$$

Create a 3D plot of the function using `mplot3d` (see lab0 for example). Use a range of $[-5, 5]$ for both x and y axes. Further, compute the gradient and Hessian of f. *what to submit: A single plot, the code used to generate the plot, the gradient and Hessian calculated along with all working. Add a copy of the code to solutions.py*

(b) Using `NumPy` only, implement the (undampened) Newton algorithm to find the minimizer of the function in the previous part, using an initial guess of $x^{(0)} = $ [illegible] 1.2 [illegible]$)^T$. Terminate the algorithm when $\|\nabla f(x^{(k)})\|_2 \leq 10^{-6}$. Report the values of $x^{(k)}$ for $k = 0, 1, \ldots, K$ where K is your final iteration. *what to submit: your iterations, and a screen shot of your code. Add a copy of the code to solutions.py*

Question 2. Solving [illegible] **numerically**

Note: throughou[illegible] **use any existing implementations of any of the algorithms discussed unless** [illegible] **so in the question. Using existing implementations can result in a grade** [illegible]**estion.** In this question we will compare gradient descent and Newton's algorit[illegible]stic regression problem. Recall that in logistic regresion, our goal is to minimiz[illegible]rred to as the cross entropy loss. Consider an intercept $\beta_0 \in \mathbb{R}$, parameter vector [illegible]^m, target $y_i \in \{0, 1\}$ and input vector $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^T$. Consider also the [illegible] $\mathbb{R}^m$ and corresponding feature vector $\phi_i = (\phi_{i1}, \phi_{i2}, \ldots, \phi_{im})^T$ where $\phi_i = \phi(x_i)$. Define the (ℓ_2-regularized) log-loss function:

$$L(\beta_0, \beta) = \frac{1}{2}\|\beta\|_2^2 + \frac{\lambda}{n}\sum_{i=1}^{n}\left[y_i \ln\left(\frac{1}{\sigma(\beta_0 + \beta^T \phi_i)}\right) + (1 - y_i)\ln\left(\frac{1}{1 - \sigma(\beta_0 + \beta^T \phi_i)}\right)\right],$$

where $\sigma(z) = (1 + e^{-z})^{-1}$ is the logistic sigmoid, and λ is a hyper-parameter that controls the amount of regularization. Note that λ here is applied to the data-fit term as opposed to the penalty term directly, but all that changes is that larger λ now means more emphasis on data-fitting and less on regularization. Note also that you are provided with an implementation of this loss in `helper.py`.

(a) Show that the gradient descent update (with step size α) for $\gamma = [\beta_0, \beta^T]^T$ takes the form

$$\gamma^{(k)} = \gamma^{(k-1)} - \alpha \times \begin{bmatrix} -\frac{\lambda}{n} 1_n^T (y - \sigma(\beta_0^{(k-1)} 1_n + \Phi\beta^{(k-1)})) \\ \beta^{(k-1)} - \frac{\lambda}{n}\Phi^T (y - \sigma(\beta_0^{(k-1)} 1_n + \Phi\beta^{(k-1)})) \end{bmatrix},$$

where the sigmoid $\sigma(\cdot)$ is applied elementwise, 1_n is the n-dimensional vector of ones and

$$\Phi = \begin{bmatrix} \phi_1^T \\ \phi_2^T \\ \vdots \\ \phi_n^T \end{bmatrix} \in \mathbb{R}^{n \times m}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^n.$$

what to submit: your working out

(b) In what follows, we refer to the version of the problem based on $L(\beta_0, \beta)$ as the *Primal* version. Consider the re-parameterization: $\beta = \sum_{j=1}^{n} \theta_j \phi(x_j)$. Show that the loss can now be written as:

$$L(\theta_0, \theta) = \frac{1}{2}\theta^T A \theta + \frac{\lambda}{n}\sum_{i=1}^{n}\left[y_i \ln\left(\frac{1}{\sigma(\theta_0 + \theta^T b_{x_i})}\right) + (1 - y_i)\ln\left(\frac{1}{1 - \sigma(\theta_0 + \theta^T b_{x_i})}\right)\right].$$

where $\theta_0 \in \mathbb{R}$, $\theta = (\theta_1, \ldots, \theta_n)^T \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times n}$ and for $i = 1, \ldots, n$, $b_{x_i} \in \mathbb{R}^n$. We refer to this version of the problem as the *Dual* version. Write down exact expressions for A and b_{x_i} in terms of $k(x_i, x_j) := \langle \phi(x_i), \phi(x_j) \rangle$ for $i, j = 1, \ldots, n$. Further, for the dual parameter $\eta = [\theta_0, \theta^T]^T$, show that the gradient descent update is given by:

$$\eta^{(k)} = \eta^{(k-1)} - \alpha \times \begin{bmatrix} -\frac{\lambda}{n} 1_n^T (y - \sigma(\theta_0^{(k-1)} 1_n + A\theta^{(k-1)})) \\ A\theta^{(k-1)} - \frac{\lambda}{n} A (y - \sigma(\theta_0^{(k-1)} 1_n + A\theta^{(k-1)})) \end{bmatrix},$$

If $m \gg n$, what is the advantage of the dual representation relative to the primal one which just makes use of the feature maps ϕ directly? *what to submit: your working along with some commentary.*

(c) We will now compare the performance of (primal/dual) GD and the Newton algorithm on a real dataset using the derived updates in the previous parts. To do this, we will work with the `songs.csv` dataset. The data contains information about various songs, and also contains a class variable outl[illegible]ong. If you are interested, you can read more about the data here, though [illegible] of each of the features will not be crucial for the purposes of this assessme[illegible]d preform the follwing preprocessing:

(I) Remove [illegible] "Artist Name", "Track Name", "key", "mode", "time_signature", "instrum[illegible]

(II) The curr[illegible]es, but logistic regression in the form we have described it here only wor[illegible]ion. We will restrict the data to classes 5 (hiphop) and 9 (pop). After removing the other classes, re-code the variables so that the target variable is $y = 1$ for hiphop and $y = 0$ for pop.

(III) Remove any remaining rows that have missing values for any of the features. Your remaining dataset should have a total of 3886 rows.

(IV) Use the `sklearn.model_selection.train_test_split` function to split your data into X_train, X_test, Y_train and Y_test. Use a `test_size` of 0.3 and a `random_state` of 23 for reproducibility.

(V) Fit the `sklearn.preprocessing.MinMaxScaler` to the resulting training data, and then use this object to scale both your train and test datasets so that the range of the data is in $(0, 0.1)$.

(VI) Print out the first and last row of X_train, X_test, y_train, y_test (but only the first 3 columns of X_train, X_test).

What to submit: the print out of the rows requested in (VI). A copy of your code in solutions.py

(d) For the primal problem, we will use the feature map that generates all polynomial features up to and including order 3, that is:

$$\phi(x) = [1, x_1, \dots, x_p, x_1^3, \dots, x_p^3, x_1 x_2 x_3, \dots, x_{p-1} x_{p-2} x_{p-1}].$$

In python, we can generate such features using `sklearn.preprocessing.PolynomialFeatures`. For example, consider the following code snippet:


```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(3)
X = np.arange(6).reshape(3, 2)
poly.fit_transform(X)

```

Transform the data appropriately, then run gradient descent with $\alpha = 0.4$ on the training dataset for 50 epochs and $\lambda = 0.5$. In your implementation, initalize $\beta_0^{(0)} = 0, \beta^{(0)} = 0_p$, where 0_p is the p-dimensional vector of zeroes. Report your final train and test losses, as well as plots of training loss at each iteration. [1] *what to submit: one plot of the train losses. Report your train and test losses, and a screen shot of any code used in this section, as well as a copy of your code in solutions.py.*

[1]if you need a sanity check here, the best thing to do is use sklearn to fit logistic regression models. This should give you an idea of what kind of loss your implementation should be achieving (if your implementation does as well or better, then you are on the right track)

(e) For the primal problem, run the dampened Newton algorithm on the training dataset for 50 epochs and $\lambda = 0.5$. Use the same initialization for β_0, β as in the previous question. Report your final train and test losses, as well as plots of your train loss for both GD and Newton algorithms for all iterations (use labels/legends to make your plot easy to read). In your implementation, you may use that the Hessian for the primal problem is given by:

[illegible] $$\beta) = \begin{bmatrix} \frac{\lambda}{n} 1_n^T D 1_n & \frac{\lambda}{n} 1_n^T D \Phi \\ \frac{\lambda}{n} \Phi^T D 1_n & I_p + \frac{\lambda}{n} \Phi^T D \Phi \end{bmatrix},$$

where D is th[illegible]x with i-th element $\sigma(d_i)(1 - \sigma(d_i))$ and $d_i = \beta_0 + \phi_i^T \beta$. *what to submit: one pl[illegible]ort your train and test losses, and a screen shot of any code used in this section, as [illegible]de in solutions.py.*

(f) For the featu[illegible]vious two questions, what is the correspongdin kernel $k(x, y)$ that can be used to give the corresponding dual problem? *what to submit: the chosen kernel.*

(g) Implement Gradient Descent for the dual problem using the kernel found in the previous part. Use the same parameter values as before (although now $\theta_0^{(0)} = 0$ and $\theta^{(0)} = 0_n$). Report your final training loss, as well as plots of your train loss for GD for all iterations. *what to submit: a plot of the train losses and report your final train loss, and a screen shot of any code used in this section, as well as a copy of your code in solutions.py.*

(h) Explain how to compute the test loss for the GD solution to the dual problem in the previous part. Implement this approach and report the test loss. *what to submit: some commentary and a screen shot of your code, and a copy of your code in solutions.py.*

(i) In general, it turns out that Newton's method is much better than GD, in fact convergence of the Newton algorithm is quadratic, whereas convergence of GD is linear (much slower than quadratic). Given this, why do you think gradient descent and its variants (e.g. SGD) are much more popular for solving machine learning problems? *what to submit: some commentary*