Assignment 2

Student name: Ya Zou
Date : 30,October,2023

Linear Models for Regression and Classification

1 Written Exercises: Linear Regression and Precision/Recall [5pts]

I'll take any opportunity to sneak in another probability question. It's a small one.

▶ Q1 Linear Model with Laplace Error [2pts]. Assuming the model described in Eq.3, show that the MLE for this model also minimizes the sum of absolute errors (SAE):

$$SAE(\mathbf{w}) = \sum_{i=1}^{N} |y_i - \mathbf{w}^T \mathbf{x}_i| \tag{4}$$

Note that you do *not* need to solve for an expression for the actual MLE expression for $\mathbf{w}$ to do this problem. Simply show / argue that the $\mathbf{w}$ that minimizes SAE would maximize the likelihood.

The given equation are: $y_i \sim \text{Laplace}(\mu = \mathbf{w}^T \mathbf{x}_i, b) \longrightarrow P(y_i|\mathbf{x}_i, \mathbf{w}) = \frac{1}{2b} e^{-\frac{|y_i - \mathbf{w}^T \mathbf{x}_i|}{b}}$

First we can show the laplace distribution has the same minimum that SAE(w) as following:

$$P(D|\theta) = \prod_{i=1}^{N} P(x_i|\theta) = \prod_{i=1}^{N} \frac{1}{2b} e^{-\frac{|y_i - w^T x_i|}{b}}$$

Apply natrual log to minimize the function as below:

$$\ln(P(D|\theta)) = \ln\left(\prod_{i=1}^{N} P(x_i|\theta)\right) = \ln\left(\prod_{i=1}^{N} \frac{1}{2b} e^{-\frac{|y_i - w^T x_i|}{b}}\right)$$

$$\Rightarrow \ln(P(D|\theta) = \ln\left(\sum_{i=1}^{n} \frac{1}{2b} e^{-\frac{|y_i - w^T x_i|}{b}}\right)$$ Note(log break multiplications to summations)

$$\Rightarrow \ln(P(D|\theta) = \sum_{i=1}^{n} -\frac{|y_i - w^T x_i|}{b} \left(\ln\left(\frac{1}{2b}\right) + \ln(e)\right)$$

$$\Rightarrow \ln(P(D|\theta) = \sum_{i=1}^{n} -\frac{|y_i - w^T x_i|}{b}\left(\ln\left(\frac{1}{2b}\right)+1\right)$$

$$\Rightarrow \ln(P(D|\theta) = \sum_{i=1}^{n} -\frac{|y_i - w^T x_i|}{b}\left(\ln(1)-\ln(2b)+1\right)$$

$$\Rightarrow \ln(P(D|\theta) = \sum_{i=1}^{n} -\frac{|y_i - w^T x_i|}{b}\left(-\ln(2b)+1\right)$$

we can see $(-\ln(2b+1)$ and b is independent from n function, thus we can know equation below:

$$\Rightarrow \ln(P(D|\theta) \propto \sum_{i=1}^{n} -\frac{|y_i - w^T x_i|}{b}$$

$$\Rightarrow \ln(P(D|\theta) \propto \sum_{i=1}^{n} -|y_i - w^T x_i|$$

Therefore, The $SAE(w) = \sum_{i=1}^{N} |y_i - w^T x_i|$ is we need to minimize and the likihood function

$\ln(P(D|\theta)) \propto \sum_{i=1}^{n} -|y_i - w^T x_i|$ is we need to maximize, we can see the inside term is the same $y_i - w^T x_i$, The value will be minimum when $y_i = w^T x_i$ and Because the negative sing in the likihood function, this will maximize when $y_i = w^T x_i$ and minimize the SAE function, Thus, we showed that w that minimize SAE would maximize the likelihood function.

1.2 Recall and Precision

y	P(y|x)	y	P(y|x)
0	0.1	0	0.55
0	0.1	1	0.7
0	0.25	1	0.8
1	0.25	0	0.85
0	0.3	1	0.9
0	0.33	1	0.9
1	0.4	1	0.95
0	0.52	1	1.0

Beyond just calculating accuracy, we discussed recall and precision as two other measures of a classifier's abilities. Remember that we defined recall and precision as in terms of true positives, false positives, true negatives, and false negatives:

$$\texttt{Recall} = \frac{\#\texttt{TruePositives}}{\#\texttt{TruePositives} + \#\texttt{FalseNegatives}} \tag{5}$$

and

$$\texttt{Precision} = \frac{\#\texttt{TruePositives}}{\#\texttt{TruePositives} + \#\texttt{FalsePositives}} \tag{6}$$

▶ Q2 Computing Recall and Precision [3pts]. To get a feeling for recall and precision, consider the set of true labels (y) and model predictions $P(y|x)$ shown in the tables above. We compute Recall and Precision at a specific threshold t – considering any point with $P(y|x) > t$ as being predicted to be the positive class (1) and $\leq t$ to be the negative class (0). Compute and report the recall and precision for thresholds t = 0, 0.2, 0.4, 0.6, 0.8, and 1.

For this question I wrote a python program to compute when t= 0, 0.2, 0.4, 0.6, 0.8 and 1. the following screen shot is my python code and the results:

Code:

logreg_hw2.py U precison_recall.py U

precison_recall.py > ...

```python
Py_x = [0.1,0.1,0.25,0.25,0.3,0.33,0.4,0.52,0.55,0.7,0.8,0.85,0.9,0.9,0.95,1.0]
y = [0,0,0,1,0,0,1,0,0,1,1,0,1,1,1,1]

for t in [0,0.2,0.4,0.6,0.8,1]:
  print("==========================================")
  print(f"When t is: {t}")

  numTruePositives = 0.000
  numFalsenegatives = 0.000
  numFlasePositives = 0.000
  for i in range(len(Py_x)):
    classifyPoint = 0
    if Py_x[i] <=  t:
      classifyPoint = 0
    else:
      classifyPoint = 1

    if y[i]==1 and classifyPoint==1:
      numTruePositives += 1
    elif y[i] == 1 and classifyPoint == 0:
      numFalsenegatives += 1
    elif y[i] == 0 and classifyPoint == 1:
      numFlasePositives += 1
  print(f"#TruePositives:{numTruePositives},#FalseNegative: {numFalsenegatives},#TruePositives:{numTruePositives}")
  if numTruePositives + numFalsenegatives == 0 or numTruePositives+numFlasePositives == 0:
    recall = precision = 0
    print(f" recall is: {recall:.2f}\n precision is: {precision:.2f}")
    break
  recall = numTruePositives/(numTruePositives+numFalsenegatives)
  precision =numTruePositives/(numTruePositives+numFlasePositives)
  # check for denominator equal to )

  print(f" recall is: {recall:.2f}\n precision is: {precision:.2f}")
```

results:

```
❯ python3 precison_recall.py
================================================
When t is: 0
#TruePositives:8.0,#FalseNegative: 0.0,#TruePositives:8.0
 recall is: 1.00
 precision is: 0.50
================================================
When t is: 0.2
#TruePositives:8.0,#FalseNegative: 0.0,#TruePositives:8.0
 recall is: 1.00
 precision is: 0.57
================================================
When t is: 0.4
#TruePositives:6.0,#FalseNegative: 2.0,#TruePositives:6.0
 recall is: 0.75
 precision is: 0.67
================================================
When t is: 0.6
#TruePositives:6.0,#FalseNegative: 2.0,#TruePositives:6.0
 recall is: 0.75
 precision is: 0.86
================================================
When t is: 0.8
#TruePositives:4.0,#FalseNegative: 4.0,#TruePositives:4.0
 recall is: 0.50
 precision is: 0.80
================================================
When t is: 1
#TruePositives:0.0,#FalseNegative: 8.0,#TruePositives:0.0
 recall is: 0.00
 precision is: 0.00
```

2.2 Playing with Logistic Regression on This Dataset

▶ Q5 Adding A Dummy Variable [1pt]. Implement the `dummyAugment` function in `logreg.py` to add a column of 1's to the left side of an input matrix and return the new matrix.

Once you've done this, running the code should produce the training accuracy for both the no-bias and this updated model. Report the new weight vector and accuracy. Did it make a meaningful difference?

my new weight vector and accuracy is screenshot below:

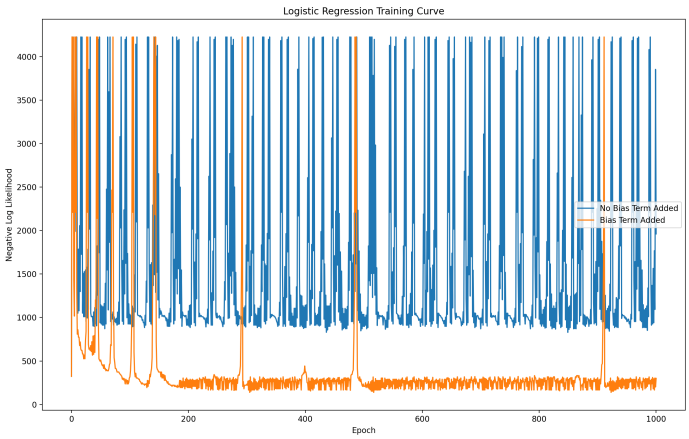
```
2023-10-30 22:28:17 INFO     Training logistic regression model (Added Bias Term)
2023-10-30 22:28:17 INFO     Learned weight vector: [-3.4144, 0.08, 0.4192, 0.2177, 0.2745, -0.2522, 0.056
1, 0.2724, -0.0528]
2023-10-30 22:28:17 INFO     Train accuracy: 96.35%
```

this definitely makes a meaningful difference from 86.27% to 96.35%, Because add bias, this will allow the algorithm to learn the best offset term for each dimensions. If there

▶ Q6 Learning Rates / Step Sizes. [2pt] Gradient descent is sensitive to the learning rate (or step size) hyperparameter and the number of iterations. Does it look like the gradient descent algorithm has converged or does it look like the negative log-likelihood could continue to drop if `max_iters` was set higher?

Different values of the step size will change the nature of the curves in the training curve plot. In the skeleton code, this is originally set to 0.0001. Change the step size to 1, 0.1, 0.01, and 0.00001. Provide the resulting training curve plots and training accuracy. Discuss any trends you observe.

stepSize = 1, max_iters = 1000:



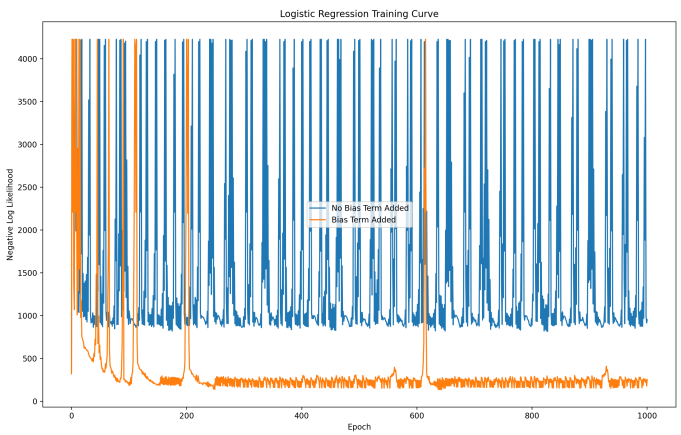
No Bais Term:

train accuracy: 69.53%

Added Bias Term:

train accuracy: 95.06%

stepsize = 0.1, max-iters = 1000:



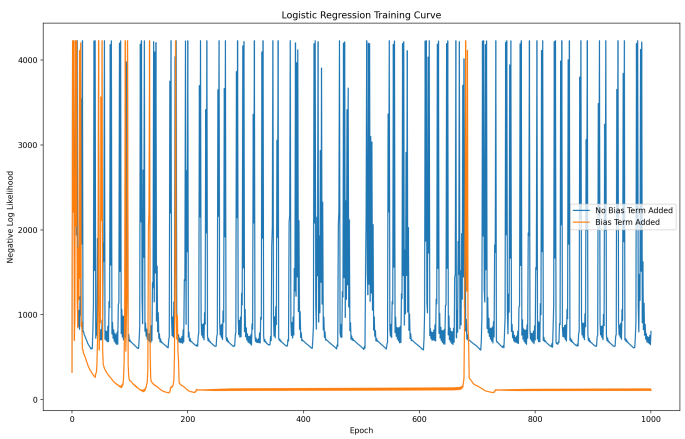
No Bais Term:

train accuracy: 84.12%

Added Bias Term:

train accuracy: 95.49%

step size = 0.01, max-iters = 1000:



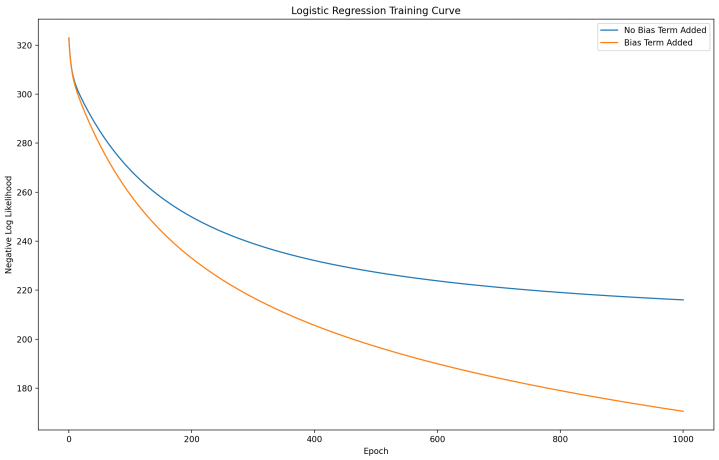
No Bais Term:

train accuracy: 81.55%

Added Bias Term:

train accuracy: 95.49%

stepsize = 0.00001, max-iters = 1000:



No Bais Term:

train accuracy: 85.62%

Added Bias Term:

train accuray: 90.77%.

If we set higher max_iters and continue to drop, it looks like the negative log-liklihood

Trends I observe:

when step size is greater, the graph is more fluctuated and unstable, when stepsize = 0.00001 it more look like negtive loglikelihood function, also when the value of step size is bigger, the train accuracy rate of No Bias term is increasing (69.53% < 84.12% < 81.55% < 85.62%). However, the training accuracy rate of added bias term didnit change that much.

▶ Q7 Evaluating Cross Validation [2pt] *Come back to this after making your Kaggle submission.*

The point of cross-validation is to help us make good choices for model hyperparameters. For different values of K in K-fold cross validation, we got different estimates of the mean and standard deviation of our accuracy. How well did these means and standard deviations capture your actual performance on the leaderboard? Discuss any trends you observe.

Higher standar deviation with a large K, will indicate that my model's performance is sensitive to the daten split, that's why my 50-fold cross-validation has the highest standerd deviation of 5.281%, when K=5 my mean = 96.56 and my standard deviation is the smallest (0.839%). Well, these means and standard deviations has higher validation accuracy rate than my Kaggle submition, accordig to these means and standard deviations my accuracy rate should be around 96.00%, but on my kaggle submision It only has 93.00%. It dropped about 3%.

In Summay, I tlink theese means and std 95% captured my accuracy rate on leaderboard.

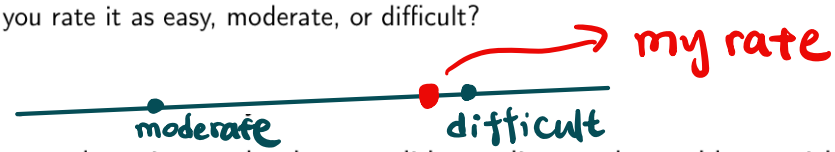
```
2023-10-30 22:59:11 INFO     Running cross-fold validation for bias case:
2023-10-30 22:59:15 INFO     2-fold Cross Val Accuracy -- Mean (stdev): 95.92% (1.502%)
2023-10-30 22:59:22 INFO     3-fold Cross Val Accuracy -- Mean (stdev): 96.34% (1.546%)
2023-10-30 22:59:32 INFO     4-fold Cross Val Accuracy -- Mean (stdev): 95.5% (1.936%)
2023-10-30 22:59:44 INFO     5-fold Cross Val Accuracy -- Mean (stdev): 96.56% (0.839%)
2023-10-30 23:00:10 INFO     10-fold Cross Val Accuracy -- Mean (stdev): 96.34% (2.354%)
2023-10-30 23:01:04 INFO     20-fold Cross Val Accuracy -- Mean (stdev): 96.46% (4.013%)
2023-10-30 23:03:11 INFO     50-fold Cross Val Accuracy -- Mean (stdev): 96.17% (5.281%)
```

3 Debriefing (required in your report)

1. Approximately how many hours did you spend on this assignment?

18 hous

2. Would you rate it as easy, moderate, or difficult?



3. Did you work on it mostly alone or did you discuss the problems with others?

I discussed with TA, Proffessor and strangers

4. How deeply do you feel you understand the material it covers (0%–100%)?

84%.

5. Any other comments?

OMG, Finally I Finished