Name: Yuxuan Zhang (yuxuanz8)

University of Illinois Spring 2020

CS 446/ECE 449 Machine Learning
Homework 2: Binary Logistic Regression

Due on Thursday February 13 2020, noon Central Time

1. **[22 points]** Binary Logistic Regression

   We are given a dataset $\mathcal{D} = \{(-1,-1), (1,1), (2,1)\}$ containing three pairs (x, y), where each $x \in \mathbb{R}$ denotes a real-valued point and $y \in \{-1, +1\}$ is the point's class label.

   We want to train the parameters $w \in \mathbb{R}^2$ (*i.e.*, weight w_1 and bias w_2) of a logistic regression model

   $$p(y|x) = \frac{1}{1 + \exp\left(-yw^\top \begin{bmatrix} x \\ 1 \end{bmatrix}\right)} \tag{1}$$

   using maximum likelihood while assuming the samples in the dataset $\mathcal{D}$ to be i.i.d.

   (a) (1 point) Instead of maximizing the likelihood we commonly minimize the negative log-likelihood. Specify the objective for the model given in Eq. (1). Don't use any regularizer or weight-decay.

   Your answer: $\max_w \prod_{(x_i,y_i)\in D} p(y_i|x_i) = \prod_{(x_i,y_i)\in D} \frac{1}{1+\exp(-y_i w^\top \binom{x_i}{1})}$

   $\Leftrightarrow \min_w -\log \prod_{(x_i,y_i)\in D} P(y_i|x_i) = \min_w -\sum_{(x_i,y_i)\in D} \log \frac{1}{1+\exp(-y_i w^\top \binom{x_i}{1})}$

   (b) (3 points) Compute the derivative of the negative log-likelihood objective in general (the one specified in the previous question, *i.e.*, no regularizer or weight-decay). Sketch a simple gradient-descent algorithm using pseudo-code (use f for the function value, $g = \nabla_w f$ for the gradient, w for the parameters, and show the update rule).

   Your answer: $\operatorname{argmax}_w \prod_{(x_i,y_i)\in D} P(y_i|x_i) = \operatorname{argmin}_w -\log(\prod_{(x_i,y_i)\in D} P(y_i|x_i))$

   $P(y_i|x_i) = \frac{1}{1+\exp(-y_i w^\top \binom{x_i}{1})}$ $= \operatorname{argmin}_w -\sum_{(x_i,y_i)\in D} \log P(y_i|x_i)$

   $f = -\sum \log P(y_i|x_i) = \sum \log(1+\exp(-y_i w^\top \binom{x_i}{1}))$

   $\nabla_w f = \sum_{(x_i,y_i)\in D} \frac{(-y_i)\exp(-y_i w^\top \binom{x_i}{1})}{1+\exp(-y_i w^\top \binom{x_i}{1})} \binom{x_i}{1}$

   Initialize $t=0$ and stepsize α.

   Compute gradient $g_t = \nabla_w f(w_t)$

   Update $w_{t+1} = w_t - \alpha g_t$

   Update $t \leftarrow t+1$

Name: Yuxuan Zhang (yuxuanz8)

(c) (5 points) Implement the algorithm by completing `A2_LogisticRegression.py`. State the code that you implemented. What is the optimal solution w^* that your program found?

Your answer:

```
f = torch.mean(torch.log(1+tmp))
g = torch.mean((1-y)*tmp/(1+tmp)*X, 1)
g = g.view(-1,1)
W = w - alph*g.
```

$w^* = \begin{pmatrix} 4.2385 \\ 0.0408 \end{pmatrix}$

(d) (3 point) If the third datapoint $(2, 1)$ was instead $(10, 1)$, would this influence the bias w_2 much? How about if we had used linear regression to fit $\mathcal{D}$ as opposed to logistic regression? Provide a reason for your answer.

Your answer:

It will not influence the bias w_2 much. w_2 changes to a0.04. If we used linear regression to fit D as apposed to logistic regression. the dataset D will have a great influence on w. which means if we change a coordinate of a part. the w_2 will be influenced a lot. It is because the loss we used in linear regression the square of errors. But in classification. the dependent only have ± 1 value. the loss should not penglize on the scale.

(e) (3 points) Instead of manually deriving and implementing the gradient we now want to take advantage of PyTorch auto-differentiation. Investigate `A2_LogisticRegression2.py` and complete the update step using the 'optimizer' instance. What code did you add? If you compare the result of `A2_LogisticRegression.py` with that of `A2_LogisticRegression2.py` after an equal number of iterations, what do you realize?

Your answer:

```
optimizer.step()
optimizer.zero_grad()
```

The results of LR1.py and LR2.py after the equal # iterations are the same. auto-differentiation realizes the same function we explicitly get the differentiation of function of f. in (c)

Name: Yuxuan Zhang (yuxuanz28)

(f) (5 points) Instead of manually implementing the cost function we now also want to take advantage of available functions in PyTorch, specifically `torch.nn.BCEWithLogitsLoss`. Can we use the originally specified dataset $\mathcal{D}$ or do we need to modify it? How? What is the probability $p(y = 1|x)$, $p(y = 0|x)$ and $p(y|x)$ if we use `torch.nn.BCEWithLogitsLoss`, *i.e.*, how does it differ from Eq. (1)? (**Hint:** $w^\top \begin{bmatrix} x \\ 1 \end{bmatrix}$ still appears.)

Your answer:

We need to modify y. X = torch.Tensor([-1, 1, 2], [1, 1, 1]) y = torch.Tensor([0, 1, 1])

$$P(y=1|x) = \frac{1}{1+\exp(-w^T\begin{pmatrix} x \\ 1 \end{pmatrix})}$$

$$P(y=0|x) = 1 - P(y=1|x) = \frac{1}{1+\exp(w^T\begin{pmatrix} x \\ 1 \end{pmatrix})}.$$

$$P(y|x) = y\,P(y=1|x) + (1-y)\,P(y=0|x)$$

Because y use 0/1 in BCEWithLogitsLoss. therefore we use y to multiply the probability and add them together so that we can take care of both situation $y=0$ or $y=1$

(g) (2 points) Complete `A2_LogisticRegression3.py` and compare the obtained result after 100 iterations to the one obtained in previous functions. Does the result differ? Why? Why not?

Your answer:

loss = criterion(netOutput, y)

The results are the same. Because the function of A2_LR3.py is the same with A2_LogisticRegression2.py. ShallowNet is a linear layer $w^T\begin{pmatrix} x \\ 1 \end{pmatrix}$. criterion = nn.BCEWithLogitsLoss() is the same loss function as previous functions. optimizer = optim.SGD(net.parameters, lr=alpha) will track w and differentiate it. and update $t \leftarrow t+1$ $w_t = w_{t-1} - \alpha g_t$. Therefore, the results of two functions are same.