Name:

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:

   Assume data is i.i.d.

   $$P(\mathcal{D}) = \prod_{(x^i, y^i) \in \mathcal{D}} P(y|x)$$

   $$\arg\max_w \prod_{(x^i,y^i)\in\mathcal{D}} P(y|x) = \arg\min_w \left(\prod_{(x^i,y^i)\in\mathcal{D}} -P(y|x) \right)$$

   $$= \arg\min_w \left(\sum_{(x^i,y^i)\in\mathcal{D}} -\log P(y^i|x^i) \right) = \arg\min_w \sum_{(x^i,y^i)\in\mathcal{D}} \log\left(1 + \exp\left(-y^{(i)} w^\top \begin{bmatrix} x^{(i)} \\ 1 \end{bmatrix}\right)\right)$$

   (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:

   $$g = \sum_{(x^{(i)}, y^{(i)})\in\mathcal{D}} \frac{-y_i \exp\left(-y^{(i)} w^\top \begin{bmatrix} x^{(i)} \\ 1 \end{bmatrix}\right)}{1 + \exp\left(-y^{(i)} w^\top \begin{bmatrix} x^{(i)} \\ 1 \end{bmatrix}\right)} \begin{bmatrix} x^{(i)} \\ 1 \end{bmatrix}.$$

   Pseudo Code:

   ```
   initialize w, α, num_steps.
   for (i in range (num_steps):
       calculate g
       w := w − αg
   ```

Name:

(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:

CODE ADDED:

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

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

(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:

This will not change it by much in the case of logistic regression. It changed from 0.048 to 0.0104. Had we used linear regression, we would have seen a big difference because linear regression has a quadratic loss instead of a log loss. Instead of rewarding the correct classification for 10, it would penalize it for being so far from the other points. Therefore, it penalizes data that is easier to classify. Log loss simply rewards or penalizes based on if the data is correctly classified.

(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:

CODE ADDED:

```
loss = torch.mean(torch.log(1+tmp))
loss.backward()
optimizer.step()
optimizer.zero_grad()
```

You get the same w* as part c. This is because we are implementing SGD with the same batch size (ie updating our weights based on the whole data set at each step) as we do in c and since the program is the same (ie the math under the hood is the same), we get the same value.

Name:

(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 modify the data by making all the -1 y values 0, so it becomes: D = {(-1,0), (1,1), (2,1)}.

$P(y=1|x) = \frac{1}{1+e^{-w^T[x]}}$

$P(y=0|x) = 1 - P(y=1|x) = \frac{1}{1+e^{w^T[x]}}$

Sanity check for when $y=1$: $P(y|x) = \frac{1}{1+e^{-w^Tx}}$

when $y=0$: $P(y|x) = \frac{1}{1+e^{w^Tx}}$

π (from pytorch doc)

$loss = [y \log(\sigma(w^Tx)) + (1-y)\log(1-\sigma(w^Tx))]$

$= \left(\frac{1}{\frac{e^{w^Tx}}{e^{w^Tx}}(1+\frac{1}{e^{-w^Tx}})}\right)^y \cdot \left(\frac{1}{1+e^{w^Tx}}\right)^{(1-y)}$

$= \left(\frac{1}{1+\frac{1}{e^{w^Tx}}}\right)^y \left(\frac{1}{1+e^{w^Tx}}\right)^{(1-y)} = \frac{e^{yw^Tx}}{(e^{w^Tx}+1)^y} \cdot \frac{1}{(1+e^{w^Tx})^{1-y}}$

$P(y|x) = \frac{1}{e^{-w^Txy}(e^{w^Tx}+1)}$

(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:

CODE ADDED:

```
y = torch.Tensor([0,1,1]).
loss = criterion(netOutput, y)
```

The result is the exact same as we got in both part c and part e. Again, the program is the same and we are performing SGD with the same batch size (3, ie all our data). Therefore it makes sense that we get the same answer. PyTorch just provides us convenient functions we can use instead of writing code from scratch.