1 Logistic regression and the loss formulation

Logistic regression (LR) has the following form:

$$f(x) = \frac{1}{\exp(-(W^T x + b)) + 1} \tag{1}$$

Where x is the feature for a sample (shape $1 \times D$), W is the weight (shape $1 \times D$), and b is the bias term.

The log loss for a single sample has the following form:

$$loss = -(y \log(1-p) + (1-y) \log(1-p)) \tag{2}$$

where y is the ground truth label and p is the prediction by LR.

For a batch of N samples, the loss is averaged over the batch:

$$loss = -\frac{1}{N} \sum_{i=1}^{N} y \log(p) + (1-y) \log(1-p) \tag{3}$$

If we add L2 regularization for the weight vector, the final loss we are going to minimize is:

$$L = -\frac{1}{N} \sum_{i=1}^{N} y \log(p) + (1-y) \log(1-p) + \alpha \frac{1}{2} \|W\|^2 \tag{4}$$

α is the term used to control the regularization effect.

2 Derivative of the loss

Now let's derive the first part of the loss w.r.t to the weight W for a single sample.

Let

$$\begin{aligned} p &= \frac{1}{e^{-z} + 1} \\ z &= W^T x + b \end{aligned} \tag{5}$$

Based on chain rule, the derivative of loss in eq. (2) w.r.t W can be written as:

$$-\left(y \cdot \frac{1}{p} \cdot \frac{\partial p}{\partial z} \cdot \frac{\partial z}{\partial W} + (1-y) \cdot \frac{-1}{1-p} \cdot \frac{\partial p}{\partial z} \cdot \frac{\partial z}{\partial W} \right) \tag{6}$$

The partial derivative is:

$$\begin{aligned} \frac{\partial p}{\partial z} &= \frac{e^{-z}}{(e^{-z}+1)^2} = p(1-p) \\ \frac{\partial z}{\partial W} &= x \end{aligned} \tag{7}$$

Now we substitute eq. (7) into eq. (6) and simply it a bit, we get the following:

$$(p-y)x \tag{8}$$

Now we deal with the regularization part in the loss, the derivative w.r.t to W is easy to get:

$$\alpha W \tag{9}$$

With eq. (8) and eq. (9), now it is easy to get the derivative of overall L w.r.t W:

$$\frac{\partial L}{\partial W} = \frac{1}{N}\sum_{i=1}^{N}(p^{(i)} - y^{(i)})x^{(i)} + \alpha W \tag{10}$$

Actually, we can write the above formulation as matrix form to make it more succinct:

$$\frac{\partial L}{\partial W} = \frac{1}{N}(\mathbf{P} - \mathbf{Y})^T\mathbf{X} + \alpha W \tag{11}$$

In the above equation, both $\mathbf{P}$ and $\mathbf{Y}$ are of shape $N \times 1$, and $\mathbf{X}$ is of shape $N \times D$.
Similarly, we can derive the derivative of L w.r.t bias b:

$$\frac{\partial L}{\partial b} = \frac{1}{N}(\mathbf{P} - \mathbf{Y})^T\mathbf{I} \tag{12}$$

where I is of shape $N \times 1$ and has value of all 1.

3 Parameter update

The parameter update rule is:

$$\begin{aligned} W^{t+1} &= W^t - \eta\frac{\partial L}{\partial W} \\ b^{t+1} &= b^t - \eta\frac{\partial L}{\partial b} \end{aligned} \tag{13}$$

In the above equation, η is the learning rate.

4 Implementation and discussions

Overall structure of the code:

1. Load and clean the data
2. Split the data into train and val, here I use 0.8 for train and 0.2 for val
3. Augment the train data to make the two classes have balanced samples
4. Train the model (checkpoint the best model)
5. Predict on the validation set

The SGDClassifier has been updated to support training the logistic regression and for predicting the result. See the code for the details.

4.1 data cleaning

I used t-SNE for data visualization and find that there is indeed a few samples that can not be clustered well. The t-SNE code can be found in the jupyter notebook data-vis.ipynb.

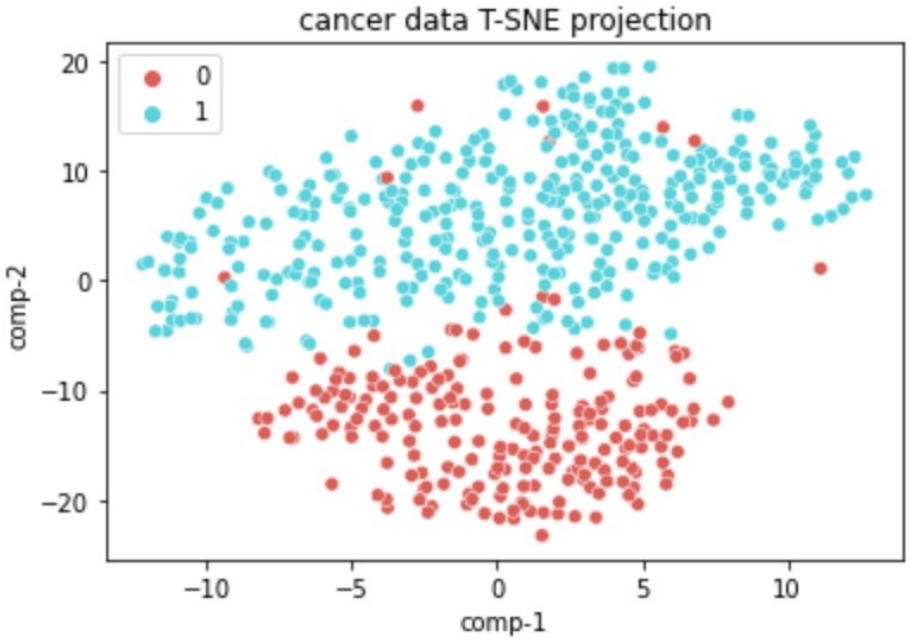

Figure 1: data t-SNE visualization

Here I only used simple heuristics to find possible noise. Since the data is relatively clean, to find noise, I simply train the model on the whole dataset. This will give use a relatively good model for predictions. Then we use the trained model to predict the probability and class for each sample. If the class prediction is wrong, but the model has strong confidence (0.8 if the ground truth is 0, 0.2 if the ground truth is 1), then this sample is likely to be an outlier.

I have also tried more sophisticated method like LOF or Isolation forest, but find their result less interpretable and do not match with each other. The code can be found in script det-noise.py. So I decided to use this simpler way to find outliers.

4.2 Early stopping

For early stopping, we usually use the accuracy on validation set as a indicator. If the accuracy on validation set stops improving for a certain period, then further training may lead to overfitting of the model.

4.3 Model performance on unseen dataset

It depends on the data distribution of unseen data. If the data distribution on the unseen dataset is similar to that of training set, then the model will perform reasonably well on unseen data. Also with the use of weight regularization and dropout, the model should be better at unseen dataset.

4.4 Impact of lr and batchsize

If we use a small learning rate, the model may converge a lower val accuracy or converges slower. For example, fix batch size to 32, if we use 0.001 as lr, the best val acc is 96.46%. If we use 0.01 as lr, the best val acc is 98.23%, at about epoch 42. If we use 0.05 as lr, the beat val acc is 99.11%, at about epoch 22.

If we fix learning rate and change batch size, large batch size may lead to slower convergence, since the model weight update frequency is lower compared to smaller batch size. For example, if we use batch size 64, the model gets 99.11% accuracy at epoch 50. If we use batch size 128, the model gets the same accuracy at epoch 60. If we use batch size 300, the model may need more than 200 epochs to reach the same accuracy.

4.5 Bonus questions

1. Is it possible to modify the training data and learn just the weight vector?

   Yes, we can add a feature with value 1 for each sample, then we can drop the bias term. Now the new weight we are going to learn is $D + 1$ dimension, where D is the original feature dimension.

2. Add a function __**dropout**, which randomly sets some of feature values to zero during training. How will you incorporate it during fit / predict?

   The dropout function has been added to the SGDClassifier. During training, each sample is masked based on the probability p. We multiply the original feature with the mask, and then divide it by $1 - p$. Because if we mask the element of each sample with probability p, then the remained value is roughly only $1 - p$ of the original value. During test, we just use the unmasked features and predict as is, not modification is needed.

3. Does __dropout help in convergence / overfitting ?

   Generally, dropout will help regularize the model and make it do better on testing. However, this dataset is so small, even without dropout, we can get high accuracy on val set.