Machine Learning Homework 2 Report

R04921039 彭俊人

1. **Logistic regression function.**

Logistic regression: $w_i \leftarrow w_i - \eta \sum -\left(\hat{y}^n - f_{w,b}(x^n)\right) x_i^n$, where $f_{w,b}(x) = \sigma\left(\sum w_i x_i + b\right)$

Therefore, it is very similar to linear regression, except for the sigmoid function.

We need to be careful with sigmoid function when dealing with zero values, so I use the “logaddexp” function in numpy for numerically-stability.

```
def sigmoid(z):
    # Numerically-stable sigmoid function
    return np.exp(-np.logaddexp(0,-z))
```

```
def logistic_regression(theta, learning_rate, X, Y):
    num_points, num_features = X.shape
    Z = sigmoid(X.dot(theta))
    gradient = np.dot((Z - Y),X)
    theta = theta - learning_rate * gradient
```

We use cross entropy for cost function $L(f) = \sum C(f(x^n), \hat{y}^n) = \sum -[\hat{y}^n ln f(x^n) + (1-\hat{y}^n) ln(1-f(x^n))]$

Here, we also need to be careful with zero values in log, so I used the numpy.ma module to filter the zero values, and filled in with 0 later.

```
def cross_entropy(x, y):
    # y is binary classification
    return -( y*(ma.log(x).filled(0)) + (1-y)*(ma.log(1-x).filled(0)))
```

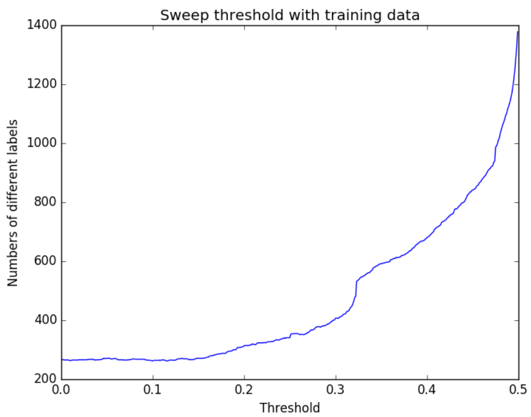
```
def compute_cost(theta, X, Y):
    num_points, num_features = X.shape
    Z = sigmoid(X.dot(theta))
    loss = cross_entropy(Z, Y)
    return sum(loss)/num_points
```

Then we use gradient descent with learning rate 0.0005 to optimize our function.

2. **Describe your another method, and which one is best.**

At first, I use 0.5 as round up threshold for the probability. However, later I found that threshold plays a huge role in deciding class.

Here, I did an experiment by using the model I trained to classify the training data using different value of threshold. We can see that the lowest point of threshold falls around 0.1, i.e. when threshold is at 0.4, we’ll get the best accuracy with only about 260 samples been misclassified. I also found that this method gives a more precise sense of when overfitting starts. So instead of using validation cost for regularization, I calculate the lowest number of misclassified sample with different threshold every 1000 iterations to decide whether the model is overfitting the training data or not. Following figure is part of the result, where we can see misclassified samples are increasing while validation cost continues to drop.



```
Iteration      23000 | Train Cost: 0.1937131697 | Validation Cost: 0.2412417918 | thres: 0.029 | diff:  260
Iteration      24000 | Train Cost: 0.1936297983 | Validation Cost: 0.2410891462 | thres: 0.057 | diff:  259
Iteration      25000 | Train Cost: 0.1935500343 | Validation Cost: 0.2409395164 | thres: 0.029 | diff:  260
Iteration      26000 | Train Cost: 0.1934735744 | Validation Cost: 0.2407928839 | thres: 0.057 | diff:  259
Iteration      27000 | Train Cost: 0.1934001536 | Validation Cost: 0.2406492118 | thres: 0.058 | diff:  258
Iteration      28000 | Train Cost: 0.1933295391 | Validation Cost: 0.2405084502 | thres: 0.058 | diff:  258
Iteration      29000 | Train Cost: 0.1932615255 | Validation Cost: 0.2403705408 | thres: 0.058 | diff:  258
Iteration      30000 | Train Cost: 0.1931959306 | Validation Cost: 0.2402354190 | thres: 0.059 | diff:  258
Iteration      31000 | Train Cost: 0.1931325921 | Validation Cost: 0.2401030165 | thres: 0.059 | diff:  258
Iteration      32000 | Train Cost: 0.1930713648 | Validation Cost: 0.2399732624 | thres: 0.059 | diff:  258
Iteration      33000 | Train Cost: 0.1930121184 | Validation Cost: 0.2398460845 | thres: 0.059 | diff:  258
Iteration      34000 | Train Cost: 0.1929547352 | Validation Cost: 0.2397214099 | thres: 0.059 | diff:  258
Iteration      35000 | Train Cost: 0.1928991087 | Validation Cost: 0.2395991654 | thres: 0.059 | diff:  256
Iteration      36000 | Train Cost: 0.1928451422 | Validation Cost: 0.2394792783 | thres: 0.059 | diff:  256
Iteration      37000 | Train Cost: 0.1927927474 | Validation Cost: 0.2393616761 | thres: 0.059 | diff:  257
Iteration      38000 | Train Cost: 0.1927418434 | Validation Cost: 0.2392462873 | thres: 0.031 | diff:  259
Iteration      39000 | Train Cost: 0.1926923561 | Validation Cost: 0.2391330409 | thres: 0.031 | diff:  259
Iteration      40000 | Train Cost: 0.1926442170 | Validation Cost: 0.2390218669 | thres: 0.031 | diff:  259
Iteration      41000 | Train Cost: 0.1925973630 | Validation Cost: 0.2389126963 | thres: 0.032 | diff:  259
Iteration      42000 | Train Cost: 0.1925517353 | Validation Cost: 0.2388054609 | thres: 0.032 | diff:  259
Iteration      43000 | Train Cost: 0.1925072795 | Validation Cost: 0.2387000933 | thres: 0.032 | diff:  259
Iteration      44000 | Train Cost: 0.1924639448 | Validation Cost: 0.2385965272 | thres: 0.031 | diff:  260
Iteration      45000 | Train Cost: 0.1924216836 | Validation Cost: 0.2384946970 | thres: 0.033 | diff:  260
Iteration      46000 | Train Cost: 0.1923804512 | Validation Cost: 0.2383945382 | thres: 0.033 | diff:  260
Iteration      47000 | Train Cost: 0.1923402059 | Validation Cost: 0.2382959866 | thres: 0.033 | diff:  260
Iteration      48000 | Train Cost: 0.1923009080 | Validation Cost: 0.2381989792 | thres: 0.033 | diff:  260
Iteration      49000 | Train Cost: 0.1922625204 | Validation Cost: 0.2381034533 | thres: 0.011 | diff:  260
```