Weekly Report(Apr 16,2018-Apr 22,2018)

Liu Junnan
ljnsjtu@hotmail.com

Abstract

1 Work Done

This week I continued to learn machine learning course, including neural network, support vector machine, clustering algorithms and recommender system.

1.1 Neural Network Contd.

A basic neural network model is shown in Fig. 1. The course required to implement neural network algorithm, including forward and back propagation processes. Although the implementation was not quite difficult given detailed instructions, a wide-used trick – vectorization was worth mentioning.

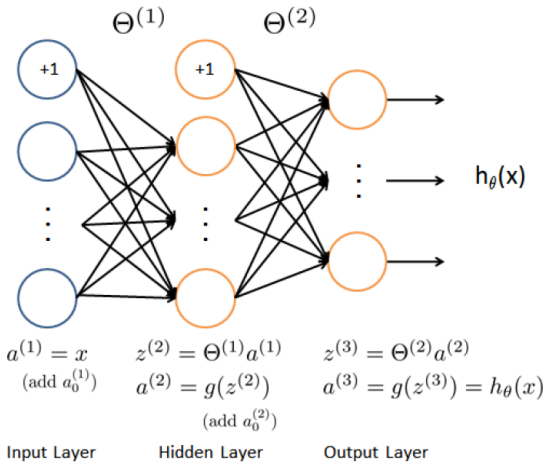

Figure 1: Neural Network Model
x denotes the input, $g(z)$ the activation function, $\Theta^{(i)}$ the weights from layer i to layer $i+1$, $a^{(i)}$ the output (activation value) of neurons in layer i, $h_\theta(x)$ the hypothesis of input x, or the prediction.

1.1.1 Vectorization

A neural network usually uses cross entropy loss function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-h_\theta(x^{(i)})_k)\right]$$

One way to implement it is using two for-loops

```
for i=1:m
   for j=1:num_labels
      J = J + -yVec(i,j)*log(a3(i,j)) - (1-yVec(i,j)) * log(1 - a3(i,j));
   end
end
J = J / m;
```

If using vectorization, the loops can be shortened as one line

```
J = 1/m * sum(sum(-yVec .* log(a3) - (1 - yVec) .* log(1 - a3)));
```

Then we can use built-in timing commands tic and toc to measure the time cost. After running these part of code for 4 times, the average time interval of for-loop is $t_l = 0.004469s$, while that of vectorization is $t_v = 0.002440s$, showing that $t_v/t_l = 0.546$, meaning almost half of the time saved.

Although it is just a simple experiment that compares the time cost between with or without vectorization, the result strongly suggests that we should apply vectorization wherever possible, since Matlab or any other programming languages' matrix packages highly optimize the matrix computations.

1.2 Bias and Variance

A machine learning algorithm sometimes suffers from over-fitting and under-fitting(shown in Fig. 2). If the number of features are large and number of training examples small, it can overfit. A very wide-used method to address overfitting problem is adding a regularization term into the cost function J(taking logistic regression as an example):

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-h_\theta(x^{(i)})_k)\right] + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

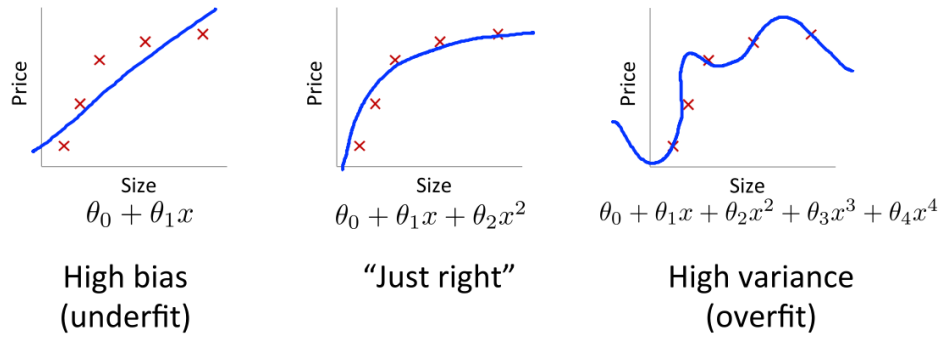

Figure 2: Under fitting and over fitting problem for linear regression

By changing the value of λ, the result will be different, maybe better or even worse. More concretely, given a set of examples with 2 features(shown in Fig. 3), we want to train a logistic classifier to predict the type of a new data. After training the model, we can draw the decision boundary for visualization. If we don't use regularization technique($\lambda = 0$), the decision boundary look like Fig. 4. The curve looks quite twisted but well separates positives from negatives. Apparently this model won't generalize to upcoming data. If we set λ as a very high value, it will cause underfitting(Fig. 5), which can't separate the training data. To avoid those problems, we should try a series of different values of λ and see how well it will be. In this case, $\lambda = 1$ works just fine(Fig. 6).

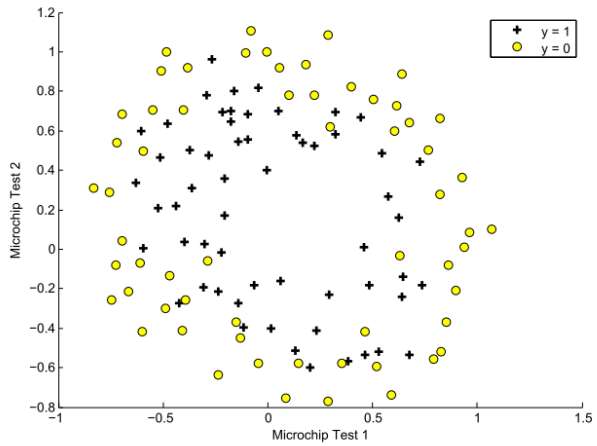

Figure 3: Plot of training data

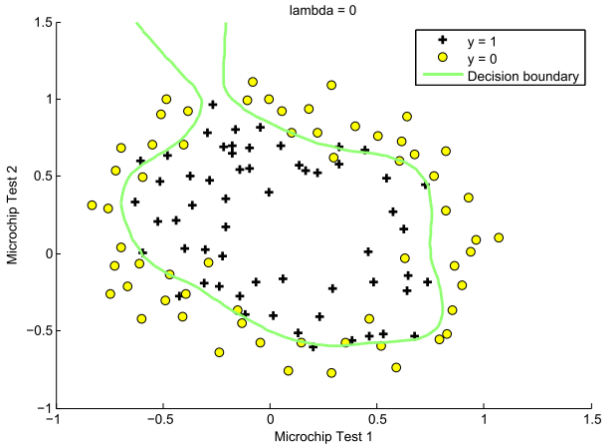

Figure 4: Overfitting($\lambda = 0$)

1.3 Error Metrics for Skewed Classes

In classification tasks, we usually use accuracy as the metric of our model. But sometimes skewed classes, or unbalanced classes problems will occur, meaning that the number of negative examples overwhelms that of positive ones. In such cases, accuracy is insufficient to measure the performance of the trained models. The confusion matrix is introduced to address this kind of problems, which is defined as follows:

Confusion Matrix		Predicted	
		0	1
Real	0	TN	FP
	1	FN	TP

Table 1: Confusion Matrix

Then accuracy is given as $\frac{TP+TN}{TP+TN+FP+FN}$. Since it is insufficient, we define precision $p = \frac{TP}{TP+FP}$, and recall $r = \frac{TP}{TP+FN}$. By changing the value of threshold, we can get pairs of precision-

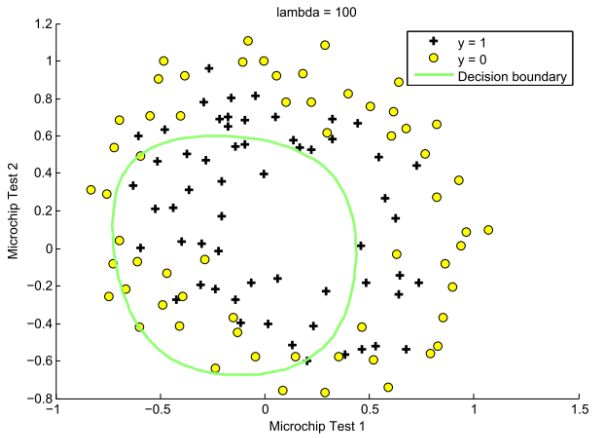

Figure 5: Underfitting($\lambda = 100$)

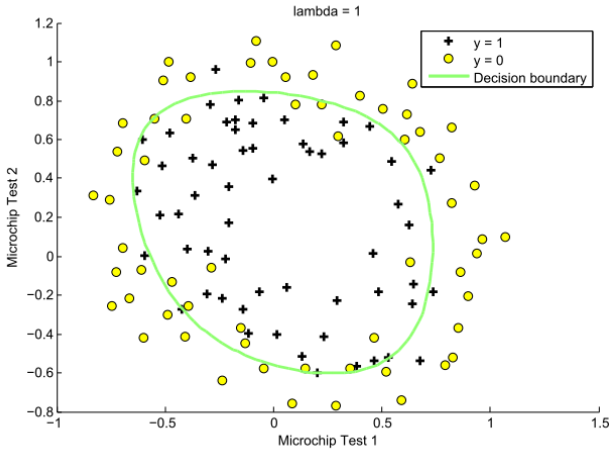

Figure 6: Good fitting($\lambda = 1$)

recall and then draw a figure. If we want a real number to measure the performance, we have

$$F_1 \text{ score} = \frac{2 * p * r}{p + r}$$

So we can compare different models with their F_1 scores accordingly, the more it is close to 1, the better performance it is.

2 Plans