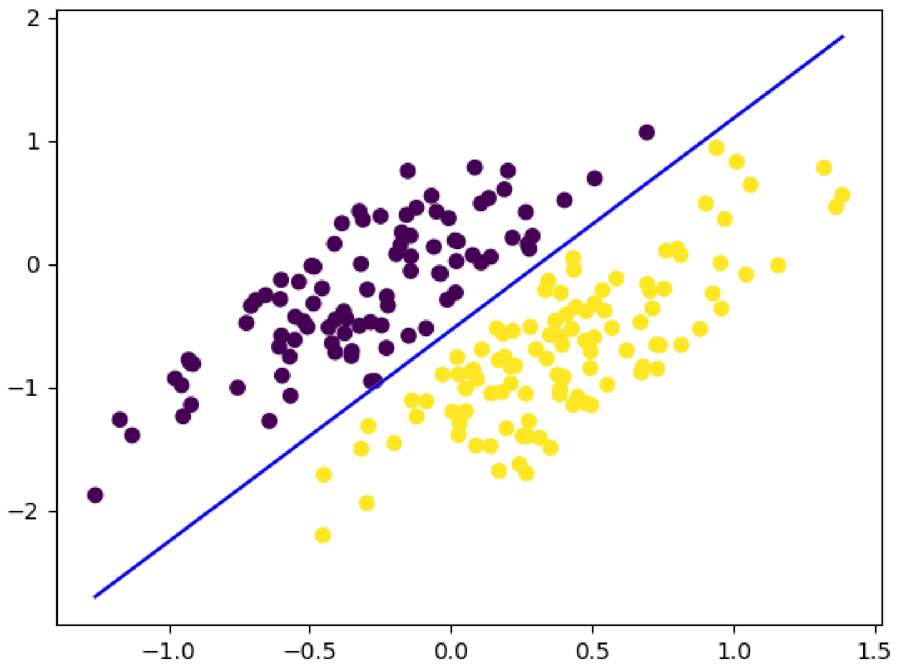
Linear Classification

Anonymity
Department of Computer Science, Fudan University
April 3, 2019

Part I

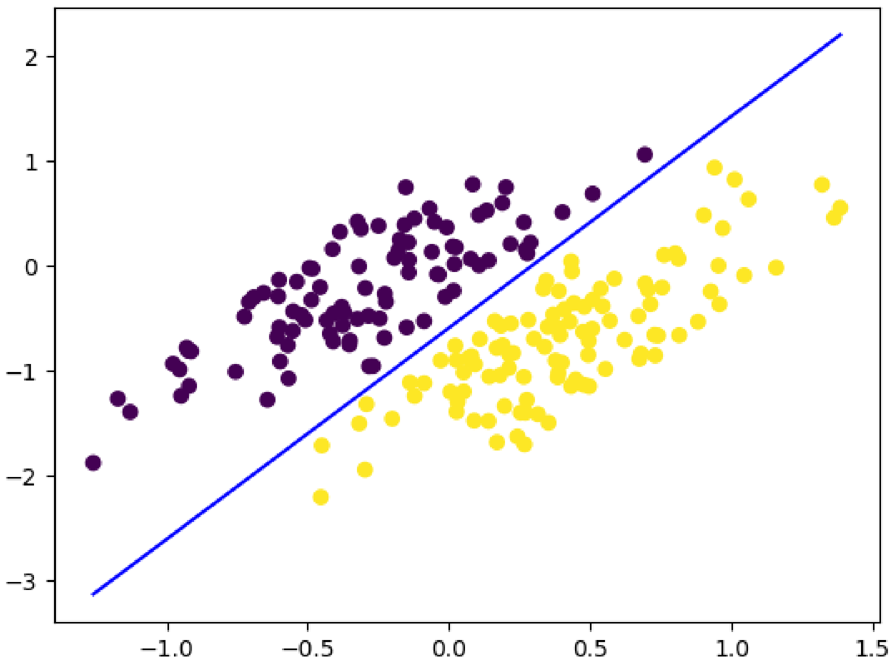
1.1 Least Square

Here is my figure for least square model with accuracy 1.00



1.2 Perceptron

Here is my figure for Perceptron algorithm with accuracy 1.00



Part II

2.1 Requirement

Using logistic regression, we can classify the given text to corresponding category. In order to process the text in a convenient way, we should first convert the document to a multi-hot vector, including Tokenizing the document into a list of words, splitting them, building a vocabulary and so on. Then we can explore the classification with generalized logistic regression for multi-class, softmax function. Since the loss function is convex, we can reach convergence through the gradient descent.

2.2 Outline

The rest of the report is organized as follow : In 2.3 we start with the preprocessing of the raw data. Mathematical derivation for the calculation will be complemented in 2.4 and plots for the loss function and details for learning will be shown in 2.5. Comparison of the 3 different ways of gradient descent will discussed in 2.6.

2.3 Preprocess

Since the document is difficult to learn the underlying category directly, we always convert it into a multi-hot vector before training, and Regular Expression in Python is the perfect tool to do it. We can first remove the punctuation like this :

```
re.sub(r'[^a-zA-Z0-9\s]', '', train.data[i])
```

Then make all whitespace characters a space :

```
re.sub(r'['+string.whitespace+']+',' ',train_set[i])
```

Further, we'll split the sentences into words with the space. Additionally, strip( ) can remove the space at the beginning and end of a string and lower( ) can convert all characters into lowercase.

`re.split(r' +', train_set[i].strip().lower())`

Then we can build a vocabulary on all the words in the training set. To simplify the following operations, we can set a threshold to control the size of the dictionary. Finally map every word to a number and represent each sentence with a multi-hot vector.

2.4 Mathematical derivation

In this Section, we'll show the derivation for the differentiation of the loss function for the softmax function. First, we have the cross entropy loss function defined as :

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log \hat{y}_i + \frac{\lambda}{2}\|W\|^2 \tag{1}$$

where $\hat{y}_i$ is the predicted probability of the correct class y_i.

So we have the differentiation for the loss function

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_j}\frac{\partial z_j}{\partial W} + \lambda W \tag{2}$$

and we have the probability of y_i in softmax model, where k represents the number of categories

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{3}$$

Equipped with these, we can begin with our derivation, we can verify with ease that

$$\frac{\partial L}{\partial \hat{y}_i} = -\frac{1}{\hat{y}_i} \tag{4}$$

as for $\frac{\partial \hat{y}_i}{\partial z_j}$, when i = j

$$\frac{\partial \hat{y}_i}{\partial z_j} = \frac{e^{z_i}\sum_{j=1}^{k} e^{z_j} - e^{z_i}e^{z_i}}{(\sum_{j=1}^{k} e^{z_j})^2} = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} - (\frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}})^2 = \hat{y}_i(1-\hat{y}_i) \tag{5}$$

when i $\neq$ j

$$\frac{\partial \hat{y}_i}{\partial z_j} = \frac{-e^{z_j}e^{z_i}}{(\sum_{j=1}^{k} e^{z_j})^2} = -\frac{e^{z_j}}{\sum_{j=1}^{k} e^{z_j}}\frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} = -\hat{y}_j\hat{y}_i \tag{6}$$

$$\begin{aligned}\frac{\partial L}{\partial W} &= -\frac{1}{\hat{y}_i}[-\hat{y}_1\hat{y}_i, \ldots, \hat{y}_i(1-\hat{y}_i), \ldots, -\hat{y}_k\hat{y}_i]^T X + \lambda W \\ &= [\hat{y}_1, \ldots, \hat{y}_i - 1, \ldots, \hat{y}_k]^T X + \lambda W\end{aligned} \tag{7}$$

and similar to $\frac{\partial L}{\partial W}$, we have

$$\frac{\partial L}{\partial b} = [\hat{y}_1 \ldots \hat{y}_i - 1 \ldots \hat{y}_k]^T \tag{8}$$

To overcome over-fitting, we should add the $L2$ regularization, and the regularization of bias term is not a must, since the bias has nothing to do with the shape of the curve but the site.

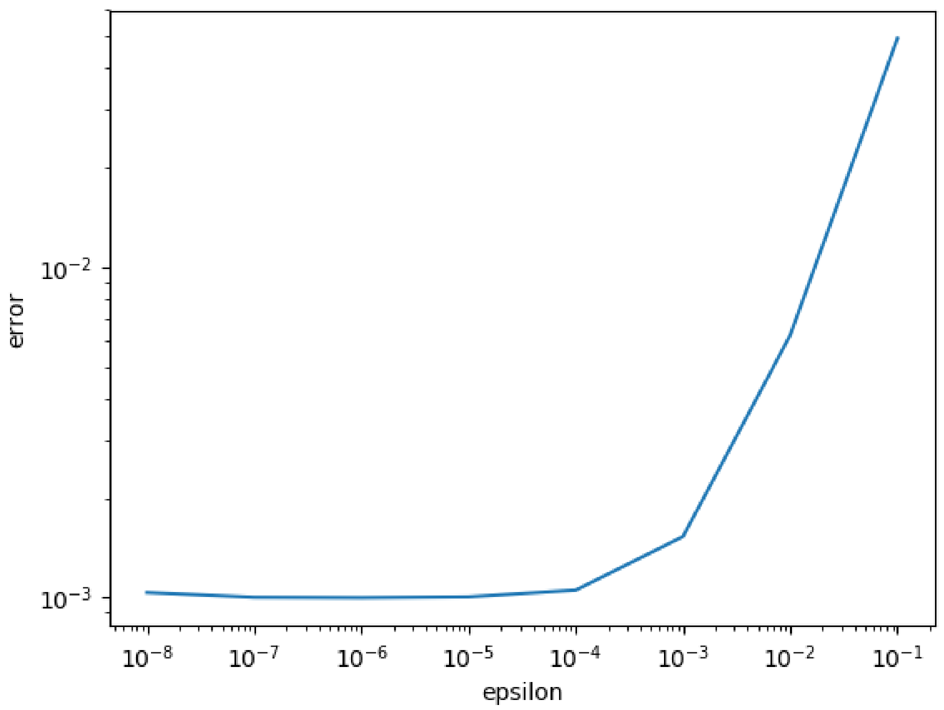
As for checking of gradient calculation, we can introduce the gradient checking :

Consider a vector $\theta = [\theta_1, \theta_2, \ldots, \theta_n]$

$$\frac{\partial J}{\partial \theta_i}(\theta) \approx \lim_{\epsilon \to 0} \frac{J(\theta_1, \ldots, \theta_i + \epsilon, \ldots, \theta_n) - J(\theta_1, \ldots, \theta_i, \ldots, \theta_n)}{\epsilon} \tag{9}$$

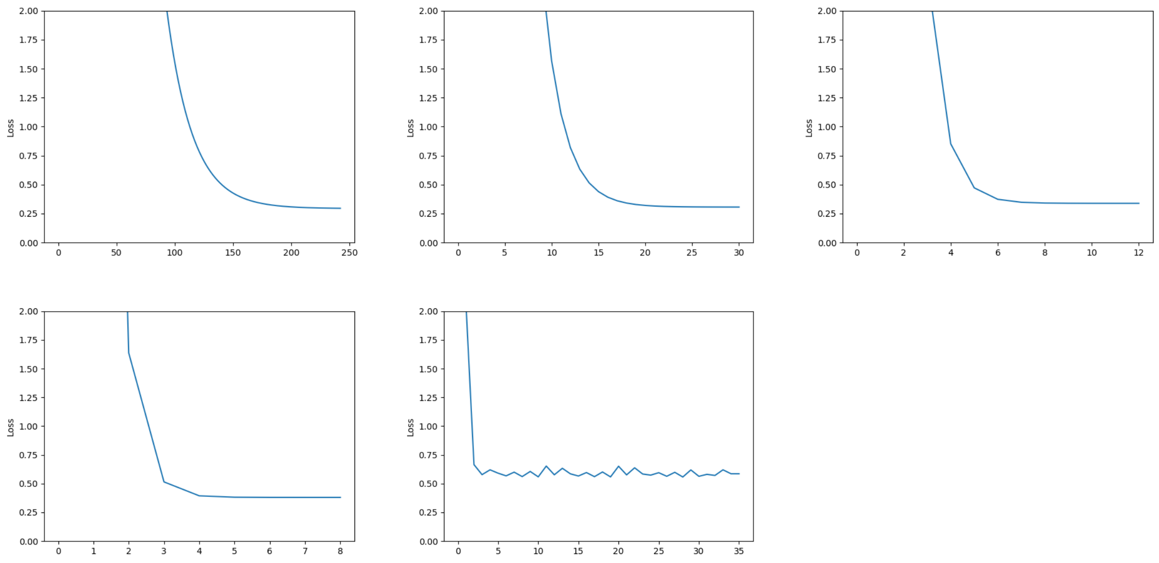
The result is guaranteed because the gradient is the directional derivative at θ, so if you give an ϵ in any direction, the change of J can be approximated by the inner product of $\frac{\partial J}{\partial \theta}(\theta)$ and ϵ, here $\frac{\partial J}{\partial \theta}(\theta)$ and ϵ are vectors.

We have gradient checking before training. Considering the efficiency, we just have 20 multi-hot vectors to check with stochastic gradient descent, and from the plot below we see that the error tend to 0 when the ϵ gets smaller.



2.5 Plot and Details for Learning

Here we will show 5 pictures obtained from our training procedure with different training rate **gama** through stochastic descent including: 10^{-3}, 10^{-2}, 3×10^{-2}, 5×10^{-2} and 10^{-1}.



As we can see, when we set a low learning rate, it takes many epochs to train, and when the rate exceeds some critical value, its epoch increases and the curve vibrates before convergence. So we'd better choose 3×10^{-2} or 5×10^{-2} to have a rapid convergence and flat curve. Certainly, the parameter selection is very crude, and there should be other good ideas to help make choices like cross validation and so on.

Since the loss function is convex, it will converge to its minimum. So we can set a threshold (Here we set as 10^{-4}), and the training procedure terminates when the absolute value of the difference between adjacent results is less than it.

2.6 Comparison

We first set accuracy as 10^{-4}, and **stochastic gradient descent** converge within **7** epochs (penalization λ = 0.01, learning rate α = 0.03), with pretty results: in training set, the correct rate is 0.9928, and for test set it's 0.9231, while **full batch gradient descent** converge within much longer epochs (penalization λ = 0.1, learning rate α = 0.03) : **1689**, having a similar result to stochastic procedure, 1.00 for training set and 0.9171 for test set. As for **batched gradient descent**, transition between the two procedures above, its correct rate and convergence change when we set different batch size, so to get a good result for different situations, we should adjust the hyper-parameter like penalization λ, learning rate α and so on. And in my trial, when I set the batch size = 2 (penalization λ = 0.01, learning rate α = 0.03), and it converge within **12** epochs and correct rate 0.9265 in training set, 0.8582 in test set.

As we can see that stochastic gradient descent reach convergence much faster than the other two procedures and it's consistent with our intuitive that the stochastic procedure can respond in time while the other two delay. And the full batch gradient descent taking all the data into consideration should perform better in the correct rate intuitively, while the batched one should compromise between the training speed and the correct rate.