Search Engine and Web Mining 11-641 HW#3

Name: Yitong Zhou **Andrew ID:** yitongz

1 Derivation of Logistic Regression Model and Algorithm

1.1 Logistic Regression Model

The likelihood function equals to:

$$L(D|\boldsymbol{w}) = \prod_{i=1}^{n} P_w\left(y^{(i)}\middle|\boldsymbol{x}^{(i)}\right) = \prod_{i=1}^{n}\left(\sigma(\boldsymbol{w^T x})\right)^{y^{(i)}}\left(1-\sigma(\boldsymbol{w^T x})\right)^{1-y^{(i)}}$$

The log-likelihood:

$$l(D|\boldsymbol{w}) = \ln L(D|\boldsymbol{w}) = \sum_{i=1}^{n}\{y^{(i)} ln\, \sigma(\boldsymbol{w^T x}) + (1-y^{(i)})\ln\left(1-\sigma(\boldsymbol{w^T x})\right)\}$$

So we want:

$$\widehat{\boldsymbol{w}}^{LR} = \arg\max\{l(\boldsymbol{w})\}$$

We achieve the optimal solution we need to solve the following equations:

$$\begin{aligned}\frac{\partial l}{\partial w_i} &= \left(y_i\frac{1}{\sigma(z)} - (1-y_i)\frac{1}{1-\sigma(z)}\right)\sigma(z)\left(1-\sigma(z)\right)\frac{\partial z}{\partial w_j}\\ &= \left(y_i - \sigma(\boldsymbol{w^T x})\right)x_i\\ &= 0 \quad , i = 0 \dots n\end{aligned}$$

2.2 Regularized Logistic Regression Model

(1) Gradient Derivation Add a regularization term, the new objective becomes:

$$\widehat{\boldsymbol{w}}^{RLR} = \arg\max\{\, l(\boldsymbol{w}) + \lambda\boldsymbol{w^T w}\}$$

So if we set $F(\boldsymbol{w}) = l(\boldsymbol{w}) + \lambda\boldsymbol{w^T w}$, the equations become:

$$\begin{aligned}\frac{\partial F}{\partial w_j} &= \frac{\partial l}{\partial w_i} + 2\lambda w_j\\ &= \left(y_j - \sigma(\boldsymbol{w^T x})\right)x_j + 2\lambda w_j\\ &= \left(y_j - \sigma(\boldsymbol{w^T x})\right)x_j + cw_j \ (set\ \lambda = 0.5c)\\ &= 0, i = 0 \dots n\end{aligned}$$

(2) Batch Version Algorithm

So the batch version of logistic regression algorithm should become:

Loop until (convergence){

$(\varepsilon > 0, j = 0 \dots n)$

$w_j := w_j + \varepsilon\sum_{i=1}^{|D|}\left(y^{(i)} - \sigma\left(\boldsymbol{w^T x}^{(i)}\right)\right)x_i^{(i)} + \varepsilon c w_j;$

}

(3) Convergence Criteria Choice

Generally the convergence criteria is that the gradient vector satisfies:

$$\left\|\frac{\partial \boldsymbol{l}}{\partial \boldsymbol{w}}\right\| \le threshold\ (a\ typical\ threshold\ can\ be\ 10^{-4} to\ 10^{-2})$$

Notice, I do not include cw_j part while considering checking convergence, because adding this part into the vector, will make the vector does not converge to 0 but to the expected mode of w. With w increasing, the condition can fail to converge to be smaller.

But as in this batch version, for each update of a parameter w, we have a batch of documents all combined together, so actually we have |D| number of different gradient vector. But the convergence condition better be one, so I mainly come up with three different conditions to use as the convergence criteria:

Table 1-1 Different Convergence Criteria Comparison

#	Criteria	Index	Performance
1	Set $\frac{\partial l}{\partial w}^{main} = \frac{1}{\|D\|}(\frac{\partial l}{\partial w}^{(1)} + \frac{\partial l}{\partial w}^{(2)} + \cdots + \frac{\partial l}{\partial w}^{\|D\|})$ **Condition:** $\left\|\frac{\partial l}{\partial w}^{main}\right\| \leq threshold$	Wallclock MI-PREC MI-REC MI-F1 MA-PREC MA-REC MA-F1	3090s 0.6501 0.6501 0.6501 0.6402 0.6432 0.6350
2	**Condition:** $\frac{1}{\|D\|}\sum_{i=1}^{\|D\|}\left\|\frac{\partial l}{\partial w}^{(i)}\right\| \leq threshold$	RUN MI-PREC MI-REC MI-F1 MA-PREC MA-REC MA-F1	150s 0.6299 0.6299 0.6299 0.6512 0.6092 0.6094
3	Count=0; If $(\left\|\frac{\partial l}{\partial w}^{(i)}\right\| \leq threshold)$ Then count+=1; (i=1,2, … |D|) **Condition:** Count=|D|	Wallclock MI-PREC MI-REC MI-F1 MA-PREC MA-REC MA-F1	N/A | 1390s * 0.6571 0.6571 0.6571 0.6534 0.6462 0.6446

* All performance check conducted on the same computer, with $threshold = 10^{-2}, \varepsilon = 10^{-4}, c = 0.5$
* The #3 method is too slow that it still cannot converge after running an hour, the result is when the count value reaches 2000 and stop.

We can see in the table that criteria #1 and #3 can generate a better result, but at a cost of far longer time. Especially for criteria #3, its convergence speed is so slow that I have to stop it when the count value reached 2000. But since the epsilon and the iteration algorithm of these methods are exactly the same, despite their different criteria for when to end the loop. As a result, it is reasonable to say that different criteria choice actually only influence the convergence time, given all the same parameters. If we can set the threshold more strictly for criteria #2 to make it also run 20~50 minutes, most likely it can achieve the same performance level.

So basically, I decided to use the criteria #2 for all the following experiments, since its time cost is much lower than other criteria, with an acceptable good result.

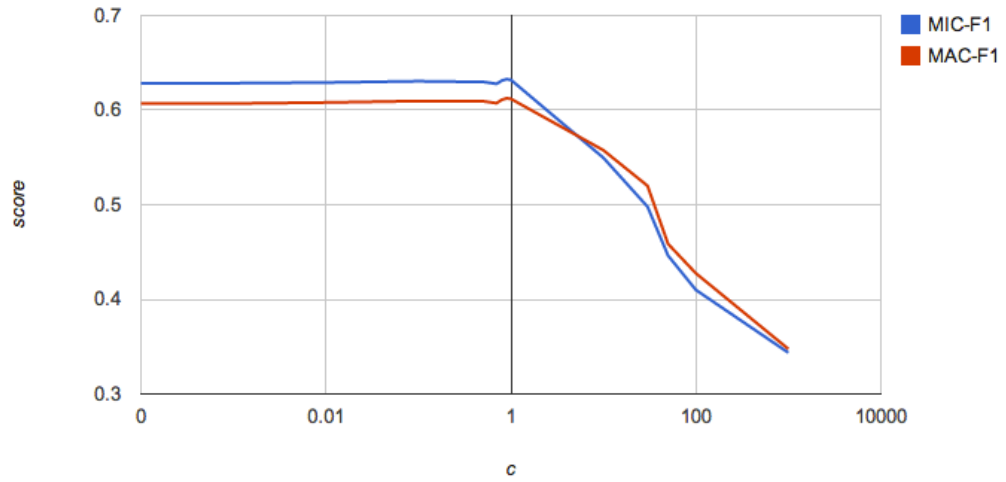
2 Experiment Result

2.1 Logistic Regression Result

The learning rate is set as 0.0001, threshold as 0.01 and x_0 as 0.

Table 2-1: LR Test Results with changing c values

c	Micro-Average			Macro-Average			Wallclock(s)
	Precision	Recall	F1	Precision	Recall	F1	
0.0001	0.6284916	0.6284916	0.6284916	0.6488168	0.6079071	0.6070756	86
0.001	0.6284916	0.6284916	0.6284916	0.6488168	0.6079071	0.6070756	84
0.01	0.6291899	0.6291899	0.6291899	0.6498608	0.6087241	0.6080903	82
0.1	0.6305866	0.6305866	0.6305866	0.6511356	0.6101344	0.6096188	82
0.4	0.6277933	0.6277933	0.6277933	0.6493827	0.6072045	0.6073674	76
0.5	0.6298883	0.6298883	0.6298883	0.6512273	0.6092393	0.6094051	77
0.7	0.6305866	0.6305866	0.6305866	0.6524182	0.6098185	0.6103904	71
0.8	0.6312849	0.6312849	0.6312849	0.6541176	0.6097079	0.6110863	73
0.9	0.6326816	0.6326816	0.6326816	0.6553096	0.6110223	0.6125105	70
1	0.6319832	0.6319832	0.6319832	0.6546112	0.6103829	0.6119229	68
10	0.549581	0.549581	0.549581	0.6664304	0.5500723	0.5576134	60
30	0.497905	0.497905	0.497905	0.6954983	0.5040545	0.5199457	26
50	0.4462291	0.4462291	0.4462291	0.6364169	0.4495707	0.458575	20
100	0.4099162	0.4099162	0.4099162	0.6620001	0.4084508	0.4274808	13
1000	0.3435754	0.3435754	0.3435754	0.5256506	0.3470754	0.3475115	4



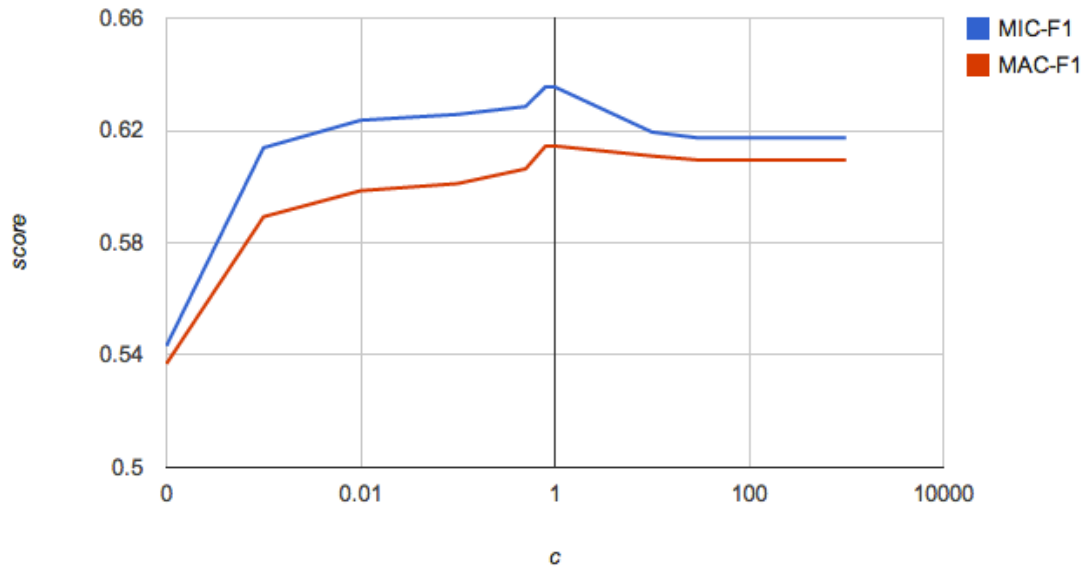
Graph 2-1: LR F1 scores with c range from 0 to 1000

2.2 SVM Result

Choose the default model of svm_light and use c from 0 to 100.

Table 2-2: SVM Test Results with changing c values

c	Micro-Average			Macro-Average			Wallclock(s)
	Precision	Recall	F1	Precision	Recall	F1	
0.0001	0.5432961	0.5432961	0.5432961	0.6206298	0.5422486	0.5369033	9
0.001	0.6138268	0.6138268	0.6138268	0.5992586	0.6050103	0.5892401	9
0.01	0.6236034	0.6236034	0.6236034	0.6129197	0.6134529	0.5985171	8
0.1	0.6256983	0.6256983	0.6256983	0.6136163	0.616132	0.6010657	9
0.5	0.6284916	0.6284916	0.6284916	0.6133924	0.6202887	0.6063037	8
0.8	0.6354749	0.6354749	0.6354749	0.6217983	0.6261279	0.6143666	9
1	0.6354749	0.6354749	0.6354749	0.6222243	0.6247685	0.6144372	9
10	0.6194134	0.6194134	0.6194134	0.6161295	0.6122009	0.6108887	9
30	0.6173184	0.6173184	0.6173184	0.6144542	0.6108113	0.6094651	9
50	0.6173184	0.6173184	0.6173184	0.6144542	0.6108113	0.6094651	9
100	0.6173184	0.6173184	0.6173184	0.6144542	0.6108113	0.6094651	9
1000	0.6173184	0.6173184	0.6173184	0.6144542	0.6108113	0.6094651	9



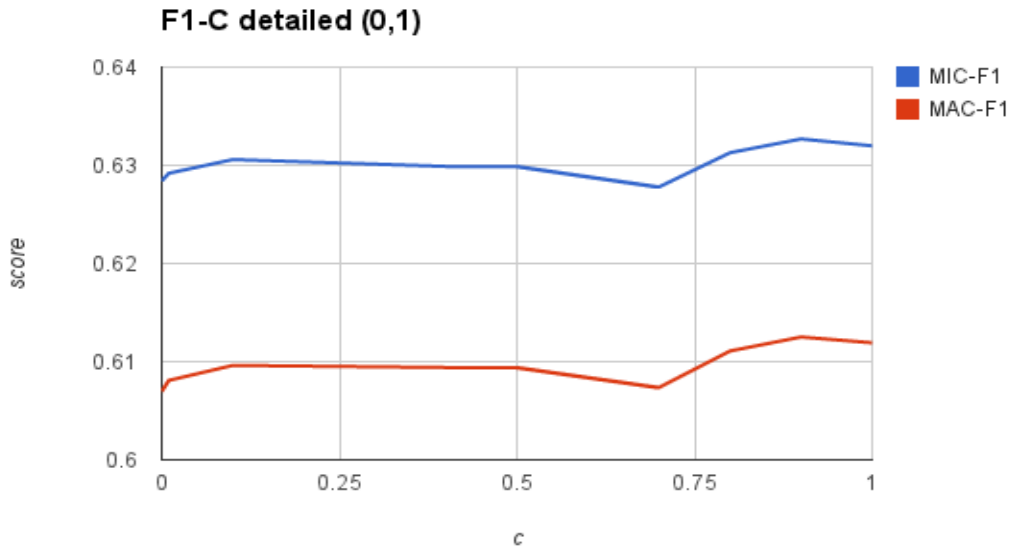
Graph 2-2: SVM F1 scores with c range from 0 to 1000

3 Analysis

3.1 F1 score with the variation of c value

(1) Logistic Regression

The table 2-1 and graph 2-1 indicates that, while the F1 score performance is mainly stable with slight variation. And when the c is larger than 1, the performance of the LR model drop very quickly, with MIC-F1 only around 33% when c reaches 1,000.



Graph 3-1: LR F1 Scores with c from 0 to 1

The Graph 3-1 is a detailed graph that only includes the performance for c within [0,1] range. As we can see in the graph, even though the performance of c within 1 much better than the performance of c larger than 1, the F1 score is not necessarily random distributed. There is a distinct drop of performance when the c is around 0.7 and the best performance is achieved around 0.9.

Such pattern indicates that as for logistic regression model, a no regularized model (c=0) can be already quite effective, and it is much safer to tune the c within, so that even it is hard to gain a performance improvement, at least we will not destroy the accuracy of the model. The c actually can hardly improve the accuracy, but it is for accelerate the convergence speed.

(2) SVM

Actually we can see that the pattern of SVM is quite the opposite of LR model. When the c is near 0, the performance is not satisfying and with c increasing the F1 increases very fast. The F1 performance gains the highest score around c=0.9 and then drop slightly but almost stay stable when the c is very large. When c is larger than 30, the performance scores are even not changing any more.

(3) Comparison

Similarity:

- Both reaches best performance around c=0.9; And their best performance are very close, all around 63%;
- Both have a plateau range where the performance basically maintains stable at a quite good level.
- Also, MICRO-F1 is always larger than the MICRO-F1 for a certain amount, which means there could be some categories with comparable low precisions, making the entire MICRO-F1 score skewed.

Difference:

- The LR's performance is stable and good when c is between 0 and 1; while SVM's is stable and quite good when c is a very large number.
- No matter what the c value is (not considering negative values), SVM can safely produce a result with F1 at least around 50%, while for LR, if the c is too large, the F1 can drastically drop towards 0.3 or even 0.
- The time cost of SVM is quite stable, no matter what the c value is, while for the LR model, when the c is very large, the time cost drops significantly together with performance scores. Maybe this is resulted from too fast convergence.

3.2 Precision and Time Cost of LR with variation of learning rate ε, and x_0

Since we basically use the default settings of SVM, this section only discuss the parameters effect in LR model.

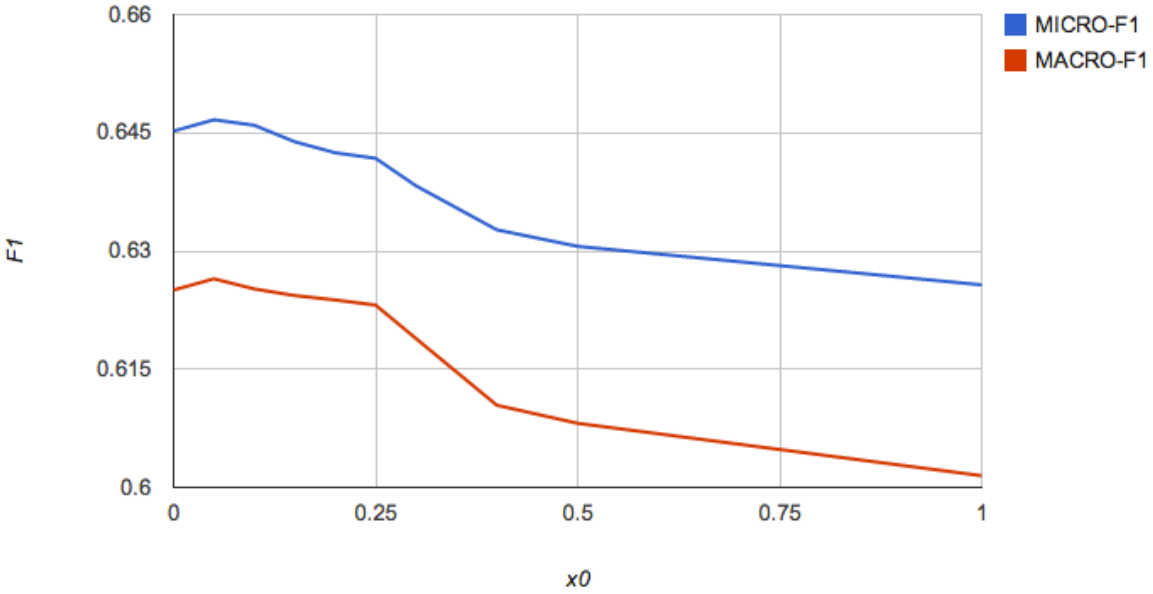
(1) Biased Hyper Plane Effect

Typically in the previous experiment, we set the $\boldsymbol{x_0 = 0}$, so that there is not any bias for the parameter vector. In the course slides (17-LR page 3), it mentions that $\boldsymbol{x_0 = 1}$, but with the real experiment result, 1 can be a very bad value choice since all the other x values are around 0.1~0.15. So I carried out a series of experiment with different $\boldsymbol{x_0 = 1}$, to see the effects of this parameter.

I set c=0.5, threshold=0.005 and learning_rate=0.0001 in these experiments.

Table 3-1: Hyperplane result with different x0

x0	MICRO-F1	MACRO-F1	Wallclock(s)
0	0.645214	0.6250394	190
0.05	0.646648	0.6264673	234
0.1	0.6459497	0.6251768	231
0.15	0.6438547	0.6243389	231
0.2	0.6424581	0.6237828	228
0.25	0.6417598	0.6231324	222
0.3	0.6382682	0.6188866	219
0.4	0.6326816	0.6104237	211
0.5	0.6305866	0.6081303	208
1	0.6256983	0.601483	208



Graph 3-2: Hyperplane result with different x0

As we can see from the result, when the x0 is around 0.1, we can achieve a better performance for the given data, but when the x0 is larger, it can lower the accuracy. Better result goes around the average score for a word, which demonstrates my previous guess that x0 has better in the same scale with other x values.

Only if the score scale can be very large, or I think there is no need to explicitly tune this parameter. For example, even if we tuned a better x0=0.1 for this particular model, when the test data changes, it can also results into accuracy changes. But at least the Graph 3-2 tells us that even set default x0 to be 0, the performance will be considerably good and safe.

Also it is interesting to notice that the time cost of running the program will first increase than drop down with x0 increasing from 0 to 1. Such time cost goes the same with variation of accuracy, which indicate that given the same learning rate and convergence threshold, time cost can be proportional to the accuracy.

(2) Learning Rate Effect

During real experiment, I have found that not necessarily a smaller learning rate means better performance, sometimes if the threshold is quite large, a very small

learning rate actually cannot perform better but in deed, cost much more time. So I carry out the following experiment to explore the effect of learning rate.

In the experiment, we set c=0.5 and x0=0.

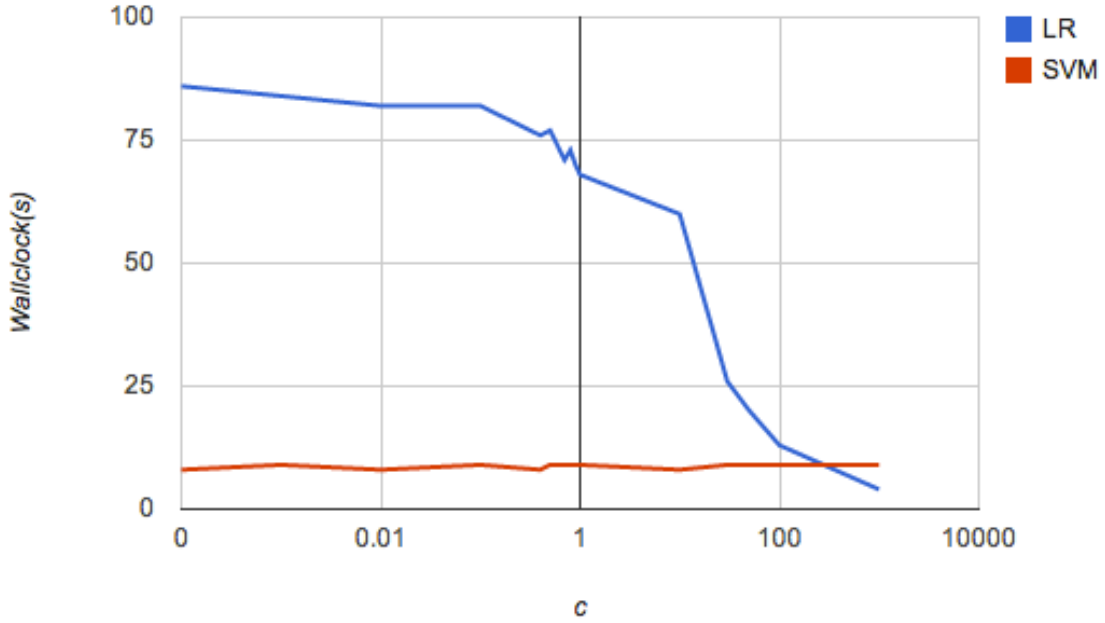
Table 3-2: F1 scores change with learning rate

Learning Rate	MICRO-F1	MACRO-F1	Wallclock(s)
I. Threshold=0.005			
0.0001	0.6452514	0.6250394	233
0.001	0.6452514	0.6250394	24
0.01	0.6459497	0.626259	4
0.1	0.650838	0.6352383	2
0.5	0.353352	0.3847594	2
1	0.1396648	0.1384467	2
II. Threshold=0.01			
0.0001	0.6298883	0.6094051	76
0.001	0.6298883	0.609432	24
0.01	0.6298883	0.6096502	2
0.1	0.5460894	0.4953405	2
0.5	0.5551676	0.5325102	2
1	0.1836592	0.1449369	2

The most significant pattern is that enlarging the learning rate can accelerate the learning speed (reduce the wall clock time very significantly). However, the accuracy may or may not increase, since in threshold 0.01 result the decrease effect is more obvious and in the threshold 0.005 result there is also an increase effect when learning rate is approaching the 0.1.

3.3 Time Cost

Since actual performance of SVM and LR are very close, it becomes more important to examine their time cost.



Graph 3-3 Time Cost of LR vs. SVM

Graph 3-3 indicates clearly SVM's advantage ----it time cost is almost stable. The SVM's algorithm time complexity only depends on the scale of data and does not depends on the c. However, for the LR model a larger c can significantly reduce the running time at the cost of loss of accuracy.
From this comparison, it is obvious that why SVM is currently a more advanced and popular model even though LR has already achieved good accuracy more than 20 years ago ---- it is because it is much faster and stable. If consider the large IO cost for handling the data and that actually in my implementation of using the svm_light, I deliberately call off the multi process and make the svm_light process learn one by one, the efficiency of SVM algorithms can even be higher.

4 Software Implementation

Since this assignment is not very complicated in programming, I will not include unity graph, UML design in this part. Just list some simple ideas.

(1) Class and Usage

Table 4-1 Class and Basic Explanation

Class	Usage
Package com.yitongz.lr	
Main	The entrance of the program
Parameter	Store the important constants like total word numbers in this Class
ParameterOperator	Interface for standard learn operation
Train	Handle configuration, read in train and test data and instantiate operator instance to get the result
GradientOperator	The implantation of batch version algorithm
DocVector	The data structure for a document, implement some iterator methods to accelerate speed
DocElement	The data structure for one dimension element of the DocVector (storing a word and its score)
Package com.yitongz.svm	
SVM	Very simple and all-together class to run the svm_light and combine result files.

(2) Important Algorithm

a) The iterating methods for DocVector

Since for each document, it has a score vector more than 14,000 long, and the w vector is also very long because of this. Doing multiplying calculations can be very very costly if every time I iterate for about 14,000 times.

So I implement a more cost saving of iterating method in DocVector. Basically the idea of algorithm is as below:
Store all x scores bigger than 0 in an ArrayList,
Implement the following methods:

Method: hasNextX()
If still has more x sores, return true;
Else return false;
Method: getNextX()
If currently the index is not equal to next non-zero x score, return 0 and increase index;

Else return current-non-zero x score and increase index;
Method: resetIterate()
Move the index back to the beginning;

Using this version of iterator, in every loop of the batch version algorithm, we will only read the non-zero scores of document once and largely saving the cost of iteration. (This algorithm made my previous 58s task only takes 2s now.)

b) The batch version algorithm
I already discussed the batch version algorithm design in the previous part, but actually, there is one more thing to notice: when the c is very large or the learning rate is too small in the LR model, sometimes it will become quite hard to converge or even impossible, to prevent the endless loop situation, I also add one more out of loop condition, that if the old criteria equals to the new criteria, it also needs to be out of the loop.

(3) Strength and weakness
Strength:

- Used batch version algorithms, adapted many efficiency improvement techniques
- Choosed a quite effective convergence criteria to converge faster while maintain high precision scores.

Weakness:

- The convergence criteria I choose are not necessarily very correct in theory. I choose the criteria because it is faster to converge. But it may also leave some shortcomings by doing so.
- Every time I improve the algorithm speed, I reuse more smaller learning rate and convergence threshold, so the default running time is not very fast, may take 3~5 minutes.
- The svm implementation is way too simple and written in one file, which I barely plan for temporary usage, if in the future, we still need the svm_light, I may need to rewrite the class.