MDS 6106 – Introduction to Optimization

Final Project

Logistic Regression and Support Vector Machines

Haoyu KANG 220041025
Yuxuan WANG 220041054
Jiadong LOU 220041045
Wenhao QU 220041062
Yuqing HUANG 220041018

December, 2020

Contents

1 Background to the project

1.1 Introduction

In this project, in order to achieve a better dichotification, we use the support vector machine model and the logistic regression model to construct the optimization problem, and apply different optimization algorithms to solve the model.

In the process of solving the solution, we picked the optimization algorithm with best performance by comparing the convergence rate, the number of iterations, calculating time and other parameters when applying different optimization methods.

Afterwards, we compute and compare the test accuracy within the picked algorithm, for different choices of the binary classification model parameters.

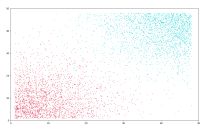
1.2 Data Preparation

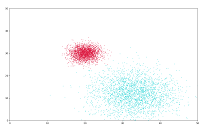
The data used in this project are divided into two categories, one is self-generated binary data, and the other is high-dimensional data from the real world.

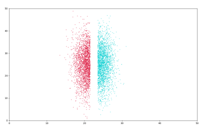
We have generated four sets of data for the dichotomy:

- In the first data set, the two central points are far apart and the variances are large, there are some but a small number of points overlap
- In the second set of data, the two central points are moderately distanced, points of Class 1 are relatively concentrated near the central point, points of Class 2 are relatively discrete
- In the third data set, two data centers are close, but the two classes of data, with a clear demarcation line, do not overlap;
- In the fourth data set, the interlaced area is relatively large, and the data is relatively scattered.

The data sets is shown in Figure 1, the size of the four datasets are 20000, 5000, 1000, 40000 respectively.


(a) 1st Data Set


(b) 2nd Data Set


(c) 3rd Data Set

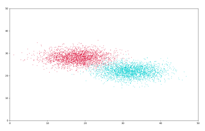

(d) 4th Data Set

Figure 1: Data Sets

2 Support Vector Machines

The smoothed support vector machine problem :

$$\min_{x,y} \quad f_{svm}(x,y) := \frac{\lambda}{2}\left\|x\right\|^2 + \sum_{i=1}^{m} \varphi_+(1 - b_i(a_i^T x + y)) \tag{1}$$

$\varphi_+(t)$ denotes a Huber-type version of the max-function max $\{0, t\}$:

$$\varphi_+(t) = \begin{cases} \frac{1}{2\delta}(max\left\{0,t\right\})^2 & if \quad t \le \delta, \\ t - \frac{\delta}{2} & if \quad t > \delta. \end{cases}$$

2.1 The Basic Gradient Method with Backtracking

2.1.1 Introduction and Preparation

Implement the basic gradient method with backtracking ($\gamma = 0.1$, $\sigma = 0.5$, $s = 1$) for the smoothed support vector machine problem (1) .
Pick a stepsize α_k by backtracking:
Let γ and σ be given, choose α_k as the largest element in $\left\{s, s\sigma, s\sigma^2, ...\right\}$such that

$$f(x^k + \alpha_k d^k) - f(x^k) \le \gamma\alpha_k \cdot \triangledown f(x^k)^T d^k$$

To calculate gradient of the function $f_{svm}(x,y)$:

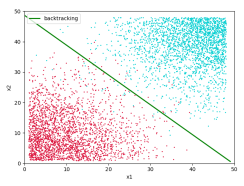
$$\frac{\partial f_{svm}(x,y)}{\partial x} = \lambda x + \sum_{i=1}^{m} \frac{\partial \varphi_+(1 - b_i(a_i^T x + y))}{\partial x}$$

$$\frac{\partial f_{svm}(x,y)}{\partial y} = \sum_{i=1}^{m} \frac{\partial \varphi_+(1 - b_i(a_i^T x + y))}{\partial y}$$

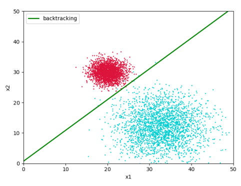
For every $i = 1, 2, ..., m$, we calculate the $\frac{\partial \varphi_+(1-b_i(a_i^T x+y))}{\partial x}$ and $\frac{\partial \varphi_+(1-b_i(a_i^T x+y))}{\partial y}$ respectively, then calculate the sum and recorde them as sum_x and sum_y.

2.1.2 Results of Backtracking

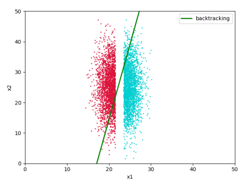
Run the basic gradient method with backtracking on our four synthetic datasets, the output is shown in Figure 2.



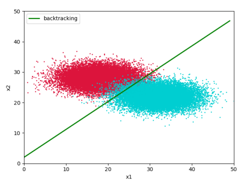
(a) Result on 1st Data Set



(b) Result on 2nd Data Set



(c) Result on 3rd Data Set



(d) Result on 4th Data Set

Figure 2: Basic Gradient Method with Backtracking on Four Synthetic Datasets

2.2 Accelerated Gradient Method (AGM)

2.2.1 Introduction

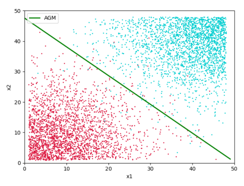
Implement the accelerated gradient method for the smoothed support vector machine problem (1). The principle idea of many acceleration techniques is to perform an extrapolation step $y^{k+1} = x^k + \beta_k(x^k - x^{(}k-1))$, $\beta_k > 0$ to approximate and extrapolate the next iterate $y^{k+1} \approx x^{x+1}$. Since the Lipschitz constant of ∇f_{sum} is unknown, so we choose the Algorithm 1 to estimate the Lipschitz constant.

Use the same gradient vector as subsection 2.1 has calculated.

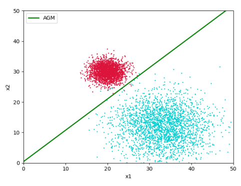
Choose $x^0 \in \mathbb{R}^n$, set $x^{-1} = x^0$ and $t_{-1} = t_0 = 1$. Choose $\alpha_{-1} > 0$ and $\eta \in (0, 1)$.
For $k = 0, 1, 2, ...,$ compute the extrapolation parameter $\beta_k = t_k^{-1}(t_{k-1} - 1)$ and set $y^{k+1} = x^k + \beta_k(x^k - x^{k-1})$, $\alpha_k = \alpha_{k-1}$ and $\bar{x}^{k+1} = y^{k+1} - \alpha_k \nabla f(y^{k+1})$.
While $f(\bar{x}^{k+1}) - f(y^{k+1}) > -\frac{\alpha_k}{2} \left\| \nabla f(y^{k+1}) \right\|^2$, set $\alpha_k = \eta\alpha_k$ and recompute $\bar{x}^{k+1} = y^{k+1} - \alpha_k \nabla f(y^{k+1})$.
Set $t_{k+1} = \frac{1}{2} \cdot (1 + \sqrt{1 + 4t_k^2})$ and $x^{k+1} = \bar{x}^{k+1}$.

2.2.2 Results of AGM

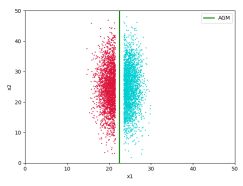
Run the AGM on our four synthetic datasets, the output is shown in Figure 3.



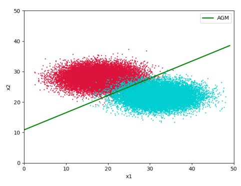
(a) Result on 1st Data Set



(b) Result on 2nd Data Set



(c) Result on 3rd Data Set



(d) Result on 4th Data Set

Figure 3: AGM on Four Synthetic Datasets

2.3 The Globalized BFGS Method

2.3.1 Introductiont to BFGS

Implement the globalized BFGS method for problem (2)
Quasi-Newton methods approximate the Hessian by a suitable and "easier", invertible matrix such that:

- Less memory storage is required.
- The resulting quasi-Newton step is much cheaper.

Quasi-Newton methods generate Hessian approximations $(B_k)_k$ using specific update rules that follow the framework:

- $S^k = x^{k+1} - x^k$, $y^k = \triangledown f(x^{k+1}) - \triangledown f(x^k)$, B_k. It will genegrate a new symmetric B_{k+1} when satisfying quasi-Newton equation $y^k = B_{k+1}s^k$, which equals to $x^{k+1} = x^k - \alpha_k B_k^{-1} \triangledown f(x^k$.
- Symmetric rank-1 update: $B_{k+1} = B_k + \gamma uu^T$.
- Symmetric rank-2 update: $B_{k+1} = B_k + \gamma uu^T + \delta vv^T$.

There are several possible updates rules:
The Broyden-Fletcher-Goldfarb-Shanno-update (BFGS):

$$B_{k+1}^{BFGS} = B_k + \frac{y^k (y^k)^T}{(y^k)^T s^k} - \frac{(B_k s^k)(B_k s^k)^T}{(s^k)^T B_k s^k}$$

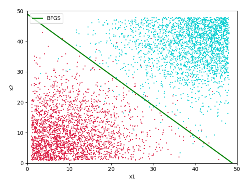
The Davidon-Fletcher-Powell-update (DFP): $h^k = y^k - B_k s^k$,then:

$$B_{k+1}^{DFP} = B_k + \frac{h^k (y^k)^T - y^k (h^k)^T}{(y^k)^T s^k} - \frac{(h_k)^T (s^k)}{((y^k)^T s^k)^2} y^k (y^k)^T.$$

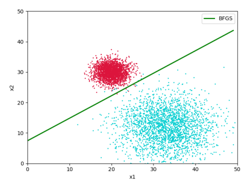
BFGS is one of the most efficent Quasi-Newton method. The properties of BFGS-Updates:
If $(s^k)y^k \neq 0$ and $(s^k)^T B_k s^k \neq 0$, the matrix B_{k+1}^{BFGS} is well-defined, symmetric, and the quasi-Newton equation holds.
If B_k is positive definite and $(s^k)^T y^k > 0$, then B_{k+1}^{BFGS} is positive condition.
Globalized BFGS-Method
Initialization: Select an initial point $x^0 \in \mathbb{R}^n$ and a symmetric, positive, definite matrix $H^0 \in \mathbb{R}^{n*n}$, here we choose $H^0 = I$. Choose $\sigma, \gamma \in (0,1)$, for k =0,1... Here we set $\sigma = 0.5, \gamma = 0.1$.
Compute the quasi-Newton direction $d^k = -H_k \triangledown f(x^k)$.
Implement backtracing line search (Armijo line search), which ia showed as above, to find a step α_k.
Set $x^{k+1} = x^k - \alpha_k d^k$. If $\left\| \triangledown f(x^{k+1}) \right\| \leq \varepsilon$, then stop.
Set $s^k = x^{k+1} - x^k$ and $y^k = \triangledown f(x^{k+1}) - \triangledown f(x^k)$. If $(s^k)^T y^k \leq 10^{-14}$, set $H_{k+1} = H_k$, otherwise

$$H_{k+1} = H_k + \frac{(s^k - H_k y^k)(s^k)^T + s^k (s^k - H_k y^k)^T}{(s^k)^T y^k} - \frac{(s_k - H_k y^k)^T (y^k)}{((s^k)^T y^k)^2} s^k (s^k)^T.$$

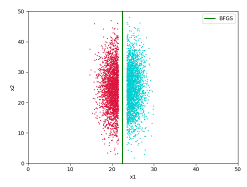
and the pair will be added to current curvature pairs if the condition if $(s^k)^T y^k > 10^{-14}$.



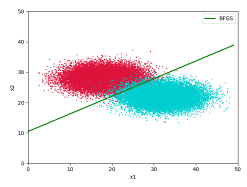
(a) Result on 1st Data Set



(b) Result on 2nd Data Set



(c) Result on 3rd Data Set



(d) Result on 4th Data Set

Figure 4: BFGS on Four Synthetic Datasets

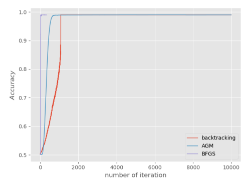
2.3.2 Results of BFGS

Run the BFGS on our four synthetic datasets, and the results is shown in Figure 4.

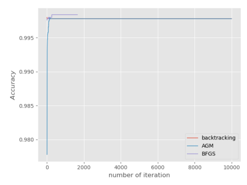
2.4 Comparison of the three methods

2.4.1 Accuracy and Convergence

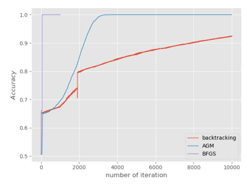
Compare the accuracy of the four methods. Shown in Figure 5



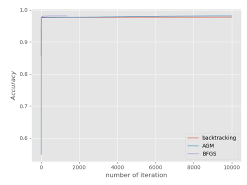
(a) Accuracy of the results on 1st Data Set



(b) Accuracy of the results on on 2^{ed} Data Set



(c) Accuracy of the results on on 3rd Data Set

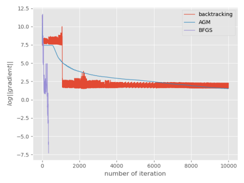


(d) Accuracy of the results on on 4th Data Set

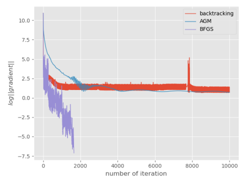
Figure 5: Compare the Accuracy of the Four Methods

When performing different optimization algorithms for synthetic data sets, it can be found that except the backtracking method cannot get an accuracy of approximately 100% accuracy rate within 10,000 iterations for the third data set, all three optimization algorithms can achieve an approximate 100% accuracy rate within 10000 iterations. However, BFGS methods can achieve a high accuracy rate within fewer iterations, which proofs that the algorithm performs better than the other two methods.

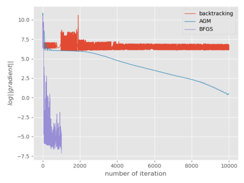
Compare the convergence of the four methods. Shown in Figure 6.
When testing the self-generated data set, we set a maximum iterations to 10,000. We found that when using the backtracking algorithm, the gradient of the SVM cannot converge, and oscillates continuously when the norm of it is still large. When applying the AGM algorithm, the gradient can be observed to decline, but it still cannot converge to the stop criterion within 10,000 iterations. Compared with the first two optimization algorithms, BFGS can converge to the stop criterion within 2,000 times in four data sets.



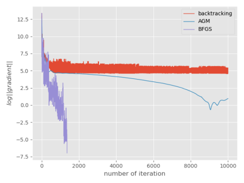
(a) Convergence of the results on 1st Data Set



(b) Convergence of the results on on 2nd Data Set



(c) Convergence of the results on on 3rd Data Set



(d) Convergence of the results on on 4th Data Set

Figure 6: Compare the Convergence of the Four Methods

2.4.2 Performance

According to the Table 1 to Table 4,when performing different optimization algorithms for synthetic data sets, we can get that backtracking method and accelerated gradient method(AGM) is not convergance. They iterate 10000 times, the maximum number of iterations we set, in the performance of each synthetic data sets. At the meantime, BFGS can get convergence in cycling less than 1700 times of all kinds of synthetic data sets, which means that it converges faster than backtracking method and AGM.
Moreover, BFGS spends the least total CPU-time of these three method. As for the accuracy, although the backtracking method and AGM do not converge, the accuracy is basically close to 100% after 10000 iterations. In addition, the accuracy of the BFGS method on this SVM problem is also colse to 100%, and it can can also converge after as few iterations as possible, so we can think that BFGS method is the best performer among the threee methods to solve this SVM problem.

Method	Iteration	Total CPU-time	Average Time	Accuracy
Backtracking	10000	132.3167	13.2317	0.9897
AGM	10000	83.9594	8.3959	0.9897
BFGS	344	27.8616	80.9931	0.9895

Table 1: Performance of Three Methods on 1st Data Set

Method	Iteration	Total CPU-time	Average Time	Accuracy
Backtracking	10000	112.8153	11.2815	0.9978
AGM	10000	74.8296	7.4829	0.9978
BFGS	1652	24.3455	14.7370	0.9984

Table 2: Performance of Three Methods on 2^{ed} Data Set

3 Logistic Regression and L-BFGS

3.1 Introduction to Logistic Regression

The classification optimization problem is given by

Method	Iteration	Total CPU-time	Average Time	Accuracy
Backtracking	10000	168.7077	16.8707	0.9231
AGM	10000	81.1285	8.1128	1.0000
BFGS	1009	60.7123	60.1707	1.0000

Table 3: Performance of Three Methods on 3^{rd} Data Set

Method	Iteration	Total CPU-time	Average Time	Accuracy
Backtracking	10000	857.9151	85.7915	0.9770
AGM	10000	595.3396	59.5339	0.9807
BFGS	1362	557.9624	409.6640	0.9807

Table 4: Performance of Three Methods on 4^{th} Data Set

$$\min_{x,y} f_{log}(x,y) = \frac{1}{m}\sum_{i=1}^{m} log(1+exp(-b_i \cdot (a_i^T x + y))) + \frac{\lambda}{2}\|x\|^2 \tag{2}$$

After we get the answer x and y of problem (2), then we should use sigmoid function to varify test data into $C1$ or $C2$.
In the case of logistic regression,the sigmoid function $\sigma : \mathbb{R} \to \mathbb{R}$, $\sigma(a) = \frac{1}{1+exp(-a)}$ is chosen as underlying probability model.
In logistic regression, we want to train the linear model $l_{(x,y)}(a) = a^T x + y$ such that

$$\sigma(l_{(x,y)}(a_i)) = \sigma(a_i^T x + y) \approx \begin{cases} 1 & if \quad a_i \quad belongs \quad to \quad class \quad C_1, i.e., b_i = +1, \\ 0 & if \quad a_i \quad belongs \quad to \quad class \quad C_2, i.e., b_i = -1. \end{cases}$$

A new data $a \in \mathbb{R}^n$ can then be classified via

$$\begin{cases} +1 & if \quad \sigma(l_{(x,y)}(a)) > \frac{1}{2}, \\ -1 & if \quad \sigma(l_{(x,y)}(a)) \le \frac{1}{2}. \end{cases} \quad or \quad \begin{cases} C_1 & if \quad \sigma(l_{(x,y)}(a)) > \frac{1}{2}, \\ C_2 & if \quad \sigma(l_{(x,y)}(a)) \le \frac{1}{2}. \end{cases}$$

This, methodology is based on a probabilistic idea. Specifically, we try to find a good parametric model of the probablity that a given feature vector belongs to the first class $C1$. The probablity is close to 1 for every feature vector which was assigned to the class $C1$. After the training has been finished, the model can be used to classify any feature vector according to the probabilistic estimate.

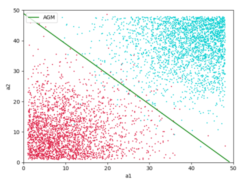
Agenda of implementing Logistic Regression

- Test the AGM and L-BFGS methods with the data set generated by ourselves, and obtain the optimal solution of the objective function respectively.
- Apply sigmoid function by using answer of objective function to classify test data set.
- Compare the effects of the two optimization methods by testing indicators.
- Choose a better performing algorithm to test the actual data set.
- Theoretically, the reason why the performance of the selected method changed to excellent

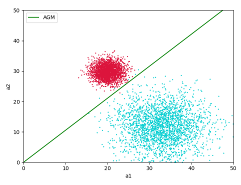
3.2 AGM on Logistic Regression and Results

The introduction of AGM method please refer to 2.2, the difference here is that this time we have known the Lipschitz constant which is $L = \frac{1}{4m}\sum_{i=1}^{m}\|a_i\|^2$, and $alpha_k = 1/L$

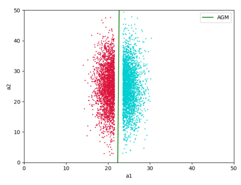
We apply four data sets to test the performence of AGM method, and the conclusion is shown in Figure 7.



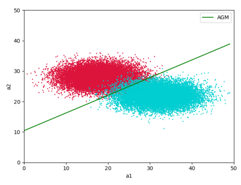
(a) Result on 1^{st} Data Set



(b) Result on 2^{nd} Data Set



(c) Result on 3^{rd} Data Set



(d) Result on 4^{th} Data Set

Figure 7: AGM on Logistic Regression

3.3 L-BFGS

3.3.1 Introduction to L-BFGS

L-BFGS method is a kind of Quasi-Newton method, which is an improvement on the basis of BFGS Method.In BFGS Merhod, we use B_k to approximate $\nabla^2 f(x^{k+1})$.

Let

$$H_{K+1} = B_{k+1}^{-1}$$

We get BFGS formula:

$$H_{k+1} = H_k + \frac{(s^k - H_k y^k)(s^k)^T + s^k(s^k - H_k y^k)^T}{(s^k)^T y^k} - \frac{(s^k - H_k y^k)^T y^k}{((s^k)^T y^k)^2} \cdot s^k (s^k)^T$$

According to the above formula, we know that each iteration of the Quasi-Newton method needs to be based on the previous iteration H_k. The storage space of the matrix is at least $N(N+1)/2$, and N is the feature dimension. For high-dimensional application scenarios, the required storage space will be very huge.The basic idea of L-BFGS is to replace the previous H_k matrix by storing a small amount of data from the previous m iterations.

We are only intrerested in:

$$d^k = -H_k \nabla f(x^k)$$

For a ventor $v \in \mathbb{R}^n$, it holds that

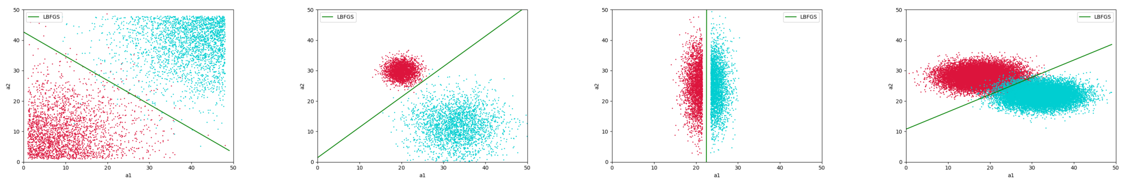
$$H_{k+1}v = H_k\left[v - \frac{(s^k)^T v}{(s^k)^T y^k} \cdot y^k\right] + \frac{(s^k)^T v}{(s^k)^T y^k} \cdot s^k - \frac{(y^k)^T H_k \left[v - \frac{(s^k)^T v}{(s^k)^T y^k} \cdot y^k\right]}{(s^k)^T y^k} s^k$$

Hence, $H_{k+1}v$ can be calculated recursively:

$$\beta_k = \frac{(s^k)^T v}{(s^k)^T y^k}, \quad q^k = v - \beta_k y^k, \quad p^k = H_k q^k, \quad H_{k+1} v = p^k + \left[\beta_k - \frac{(y^k)^T p^k}{(s^k)^T y^k}\right] s^k$$

3.3.2 Results

Run the L-BFGS on our synthetic datasets, and the results is shown in Figure 8.



(a) Result on 1st Data Set (b) Result on 2nd Data Set (c) Result on 3rd Data Set (d) Result on 4th Data Set

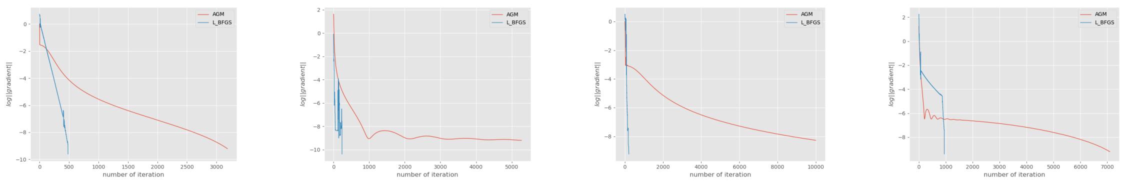
Figure 8: L-BFGS on our synthetic datasets

3.4 Comparison of the two methods

We can see from the test index that the performance of L-BFGS is better than that of AGM. This is mainly because BFGS is a kind of Quasi-Newton method, its convergence type is quadratic convergence, and the convergence type of AGM is Linear convergence. Moreover, This difference is more obvious when using higher-dimensional real data detection. Because L-BFGS is designed to be a kind of BFGS in dealing with high-dimensional problems.

3.4.1 Convergence and Accuracy

Compare the convergence of AGM and L-BFGS. Shown in Figure 9.



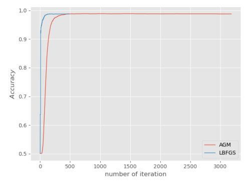
(a) Result on 1st Data Set (b) Result on 2nd Data Set (c) Result on 3rd Data Set (d) Result on 4th Data Set

Figure 9: Compare Convergence of AGM and L-BFGS

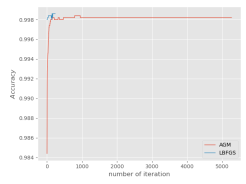
In Figure 9, L-BFGS algorithm displays higher convergence rate than AGM algorithms for all the four synthetic data sets, which is consistent to our expectations since AGM aims to approximate and extrapolate next iteration step based on gradient method, however, L-BFGS is an algorithm to approximate Hessian matrix based on the Newton's Method.

Compare the accuracy of AGM and L-BFGS. Shown in Figure 10.

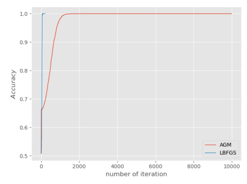
Figure 10 shows the result of testing the accuracy rate when applying AGM and L-BFGS algorithm, both algorithm can achieve nearly 100% accuracy rate after several iterations.



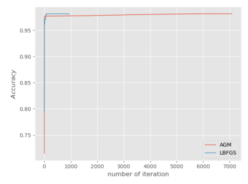
(a) Result on 1st Data Set



(b) Result on 2nd Data Set



(c) Result on 3rd Data Set



(d) Result on 4th Data Set

Figure 10: Compare Accuracy of AGM and L-BFGS

3.4.2 Performance

Table 5 to Table 8 shows the performance of AGM and L-BFGS on our four synthetic datasets.

From Table 5 to Table 8, the iterations needed to converge for L-BFGS method is significantly less than the iterations needed to converge for AGM method, however, each iteration costs more time when applying the L-BFGS algorithm. When comparing the Total computation time of both algorithms, AGM algorithm is more time consuming due to the number of iterations it needs to reach the optimal point.

Method	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
AGM	3186	22.1752	6.9602	0.9875
L_BFGS	483	29.5265	61.1316	0.9875

Table 5: Performance of Two Methods on 1st Data Set

Method	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
AGM	5273	33.8302	6.4157	0.9982
L_BFGS	238	2.4870	10.4496	0.9986

Table 6: Performance of Two Methods on 2nd Data Set

4 Performance on Real-world Datasets

To compare the performance of Support Vector Machine with BFHS (SVM_BFGS) method , Support Vector Machine with L-BFGS and Logistic Regression with L-BFGS (SVM_L-BGFGS) method, we use real-world data sets breast-cancer, mushrooms and gisette (description of the datasets is shown in Table 9, m refers to size, n refers to dimension) to run the optimization models, and compare their performance by comparing iteration times, CPU-time, average time and accuracy.

Method	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
AGM	10000	66.1526	6.6152	1.0000
L_BFGS	210	2.1659	10.3139	1.0000

Table 7: Performance of Two Methods on 3^{rd} Data Set

Method	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
AGM	7089	355.6361	50.1673	0.9817
L_BFGS	946	232.8000	246.0888	0.9815

Table 8: Performance of Two Methods on 4^{th} Data Set

4.1 Dataset - breast-cancer

4.1.1 SVM with BFGS and LR with L-BFGS

From Table 10, we can see that on breast-cancer, logistic regression with L-BFGS has higher accuracy than SVM with BFGS, at the meantime, LR_L-BFGS has less iteration times and shorter CPU-Time, so we can say that LR_L-BFGS performances better in this small scale dataset breast-cancer.
Convergence of LR_L-BFGS and SVM_BFGS is shown in Figure 11.

4.1.2 SVM with L-BFGS and LR with L-BFGS

As for L-BFGS in both SVM and Logistic Regression method, it's obvious as shown in Table 11 that in logistic regression model, it has a better performance. Less iteration times, shorter CPU-time but still a little higher accuracy.
Accuracy and convergence of LR_L-BFGS and SVM_L-BFGS is shown in Figure 12 and 13.

4.1.3 Adjust Parameters of LR_L-BFGS on breast-cancer

Adjust parameter m:
Adjust the parameter m, and run the logistic regression model with L-BFGS on small scale dataset breast-cancer, performance is shown in Table 12.
We apply different values for parameter m in Logistic Regression model performed by L-BFGS algorithm to find out difference in the convergence rate when parameter m takes different value. From Figure 14, it's obvious that when m equals to 12, the model is much more time consuming, while when m equals to 3, the model is much more effective.
Adjust parameter λ:

Data Set	m	n
breast-cancer	638	10
mushrooms	8124	112
gisette	6000	5000

Table 9: A description of the datasets used in the numerical comparison

Method	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
SVM_BFGS	972	9.1696	9.4338	0.9791
LR_L_BFGS	99	0.3810	3.8493	1.0000

Table 10: Performance of LR_LBFGS and SVM_BFGS on breast-cancer

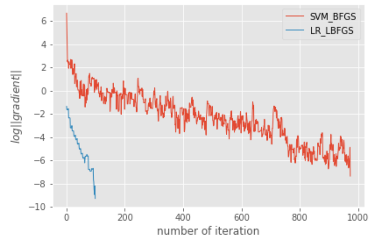


Figure 11: Compare Convergence of LR_L-BFGS and SVM_BFGS on breast-cancer

We apply different values for parameter λ in Logistic Regression model performed by L-BFGS algorithm to find out difference in the accuracy rate when parameter λ takes different value. From Figure 15, it's clearly to see that when equals to 0.3, the accuracy rate is significantly small since much weight is put on the margin and less weight is put on the misclassification error. In the breast-cancer datasets, we found that $\lambda = 0.0001$ displays a higher accuracy rate.

4.2 Dataset - mushrooms

4.2.1 SVM with BFGS and LR with L-BFGS

To avoid time consuming, we set the iteration time limit as 500 for this middle scale dataset mushrooms.
As provided in Table 13, both of them have high accuracy. LR with L_BFGS has less iterations and shorter CPU-times, so it's more effective than SVM_BFGS.
The plot of accuracy and convergence is shown in Figure 16 and 17.

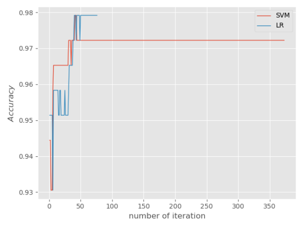


Figure 12: Compare Accuracy of LR_L-BFGS and SVM_L-BFGS on breast-cancer

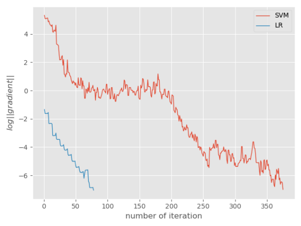


Figure 13: Compare Convergence of LR_L-BFGS and SVM_L-BFGS on breast-cancer

Method	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
SVM_L_BFGS	375	0.9741	2.5977	0.9722
LR_L_BFGS	78	0.1998	2.5615	0.9791

Table 11: Performance of LR_L-BFGS and SVM_L-BFGS on breast-cancer

m	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
5	162	0.4638	2.8631	0.9791
7	198	1.2345	4.2412	0.9791
10	244	1.0348	4.2412	0.9791
12	362	2.2304	6.1615	0.9791

Table 12: Performance of LR_L-BFGS with Different Parameter m on breast-cancer

4.2.2 SVM with L-BFGS and LR with L-BFGS

Table 14 shows that both of logistic regression with L-BFGS and support vactor machine with L-BFGS has high accurcy. However, the LR-L-BFGS has shorter CPU-time and less Iteration times, so LR_L-BGFS wins again!
The accuracy and convergence figure is shown in Figure 18 and 19.

4.2.3 Adjust Parameters of LR_L-BFGS on mushrooms

Adjust the parameter m, and run the logistic regression model with L-BFGS on medium scale dataset mushrooms, performance is shown in Table 15.
Adjust parameter m:
Convergence of different parameter m is shown in Figure 20.

Above all, although the accuracy is the same, but as the m become large, iteration and CPU-time also grows. So, we think that the smaller m is better.
Adjust parameter λ:
Convergence of different parameter λ is shown in Figure 21.

4.3 Dataset - gisette

4.3.1 SVM_BFGS and LR_L-BFGS

To avoid time consuming, we set the iteration time limit as 100 for this large scale dataset gisette. As provided in Table 16, LR with L_BFGS has a little bit higher accuracy, shorter CPU-time and both of them reach the maximum iteration times. So, we can say that LR with L_BFGS is more offective.

Method	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
SVM_BFGS	500	123.1250	246.2500	1.000
LR_L-BFGS	116	2.6862	23.1569	1.0000

Table 13: Performance of SVM_BFGS and LR_LBFGS on Mushroom

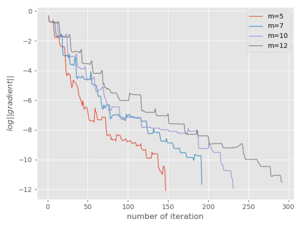


Figure 14: Compare Convergence of LR__L-BFGS with Different Parameter m on breast-cancer

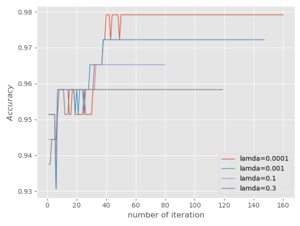


Figure 15: Compare Convergence of LR__L-BFGS with Different Parameter λ on breast-cancer

4.3.2 SVM with L-BFGS and LR with L-BFGS

Table 17 shows that both of logistic regression with L-BFGS and support vactor machine with L-BFGS has high accurcy. However, the LR-L-BFGS has shorter CPU-time and less Iteration times, so we think LR__L-BFGS has better performance.

4.3.3 Adjust Parameters of LR__L-BFGS on gisette

Adjust parameter m:

Adjust the parameter m, and run the logistic regression model with L-BFGS on big scale dataset gisette, performance is shown in Table 18.

Convergence of different parameter m is shown in Figure 25.

Adjust parameter λ:

Convergence of different parameter λ is shown in Figure 26.

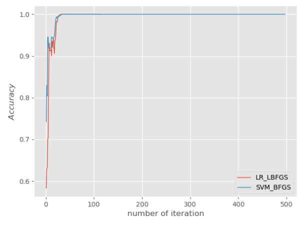


Figure 16: Compare Accuracy of LR__L-BFGS and SVM__BFGS on mushroom

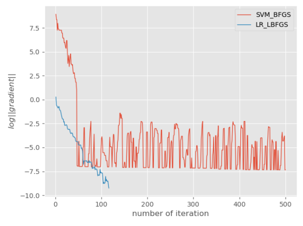


Figure 17: Compare Convergence of LR__L-BFGS and SVM__BFGS on mushroom

Method	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
SVM_L_BFGS	99	1.4639	14.7878	1.0000
LR_L_BFGS	62	0.69400	11.1940	1.0000

Table 14: Performance of LR_L-BFGS and SVM_L-BFGS on Breastcancer

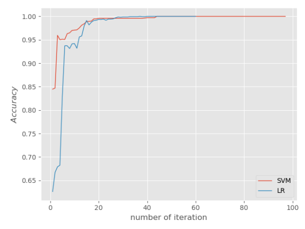


Figure 18: Compare Accuracy of LR_L-BFGS and SVM_L-BFGS on Mushroom

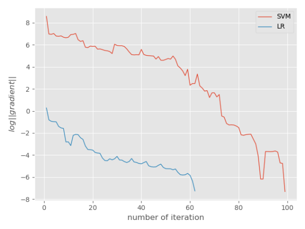


Figure 19: Compare Convergence of LR_L-BFGS and SVM_L-BFGS on Mushroom

m	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
5	147	2.3492	15.9815	1.0000
7	192	4.4258	23.0514	1.0000
10	231	6.2011	26.8447	1.0000
12	292	10.8416	37.1288	1.0000

Table 15: Performance of LR_L-BFGS with Different Parameter m on mushrooms

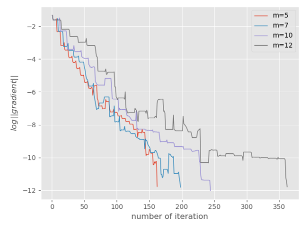


Figure 20: Compare Convergence of LR_L-BFGS with Different Parameter m on mushrooms

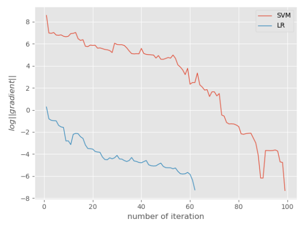


Figure 21: Compare Convergence of LR_L-BFGS with Different Parameter λ on mushrooms

Method	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
SVM_BFGS	100	3452.3679	34523.6795	0.9720
LR_L_BFGS	100	499.1347	4991.3479	0.9770

Table 16: Performance of LR_L-BFGS and SVM_L-BFGS on gisette

Method	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
SVM_L_BFGS	95	386.6717	4070.2293	0.9770
LR_L_BFGS	100	518.7887	5187.8872	0.9770

Table 17: Performance of LR_LBFGS and SVM_L-BFGS on Gisette

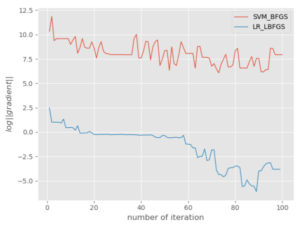


Figure 22: Compare Convergence of LR__L-BFGS and SVM__L-BFGS on Gisette

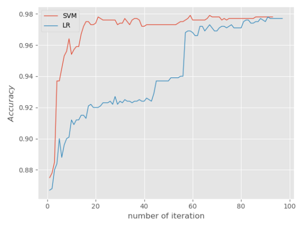


Figure 23: Compare Accuracy of LR__L-BFGS and SVM__L-BFGS on Gisette

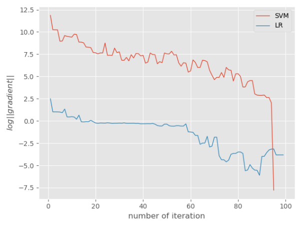


Figure 24: Compare Gradient Norm of LR__L-BFGS and SVM__L-BFGS on Gisette

m	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
5	100	515.5406	5155.4061	0.9970
7	100	719.5999	7195.9997	0.9970
10	100	689.7119	6897.1192	0.9710
12	100	565.3267	5653.2676	0.9700

Table 18: Performance of LR__L-BFGS with Different Parameter m on mushrooms

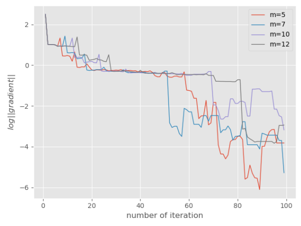


Figure 25: Compare Convergence of LR__L-BFGS with Different Parameter m on gisette

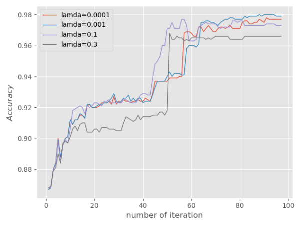


Figure 26: Compare Convergence of LR__L-BFGS with Different Parameter λ on gisette

5 Stochastic Gradient Descent

The gradient descent method needs to traverse the entire data set every time the weight is updated. When the amount of data is small, we can still accept this algorithm. Once the amount of data is too large, using this method will make the convergence process extremely slow, and when there are multiple local minima, the global optimal solution cannot be guaranteed. In order to solve such problems, an advanced form of gradient descent method is introduced: stochastic gradient descent method.

We assume that $f_i(x)$ is the loss function of the training dataset with n examples, an index of i, and parameter vector of x, then we have the objective function.

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

$$\triangledown f(x) = \frac{1}{n}\sum_{i=1}^{n} \triangledown f_i(x)$$

If gradient descent is used, the computing cost for each independent variable iteration is $O(n)$, which grows linearly with n. Therefore, when the model training dataset is large, the cost of gradient descent for each iteration will be very high.

Stochastic gradient descent (SGD) reduces computational cost at each iteration. At each iteration of stochastic gradient descent, we uniformly sample an index $i1,,n$ for data examples at random, and compute the gradient $f_i(x)$ to update x:

$$x \leftarrow x \leftarrow \eta \triangledown f_i(x)$$

Here, is the learning rate. We can see that the computing cost for each iteration drops from $O(n)$ of the gradient descent to the constant $O(1)$. We should mention that the stochastic gradient $f_i(x)$ is the unbiased estimate of gradient $f(x)$.

$$\mathbb{E}_i \triangledown f_i(x) = \frac{1}{n}\sum_{i=1}^{n} \triangledown f_i(x) = \triangledown f(x)$$

This means that, on average, the stochastic gradient is a good estimate of the gradient.
In each iteration we sample one index i_k from $\{1, ..., m\}$ uniformly at random and perform the update

$$\begin{pmatrix} x^{k+1} \\ y^{k+1} \end{pmatrix} = \begin{pmatrix} x^{k} \\ y^{k} \end{pmatrix} - \alpha_k \triangledown f_{ik}(x^k, y^k).$$

We choose the step size

$$\alpha_k = \frac{10}{1 + 0.01k}$$

In this section, we minimize the computation task for each iteration by randomly selecting several batches of different size as substitution of the original large-scale dataset. We test this approach for

the Dataset ”rcv1” which is of 47236 dimension and has 20242 observations. It’s worth noting that before we apply the stochastic algorithm, none of the algorithm could handle this dataset since it needs really much computation power to execute even one iteration.
The result is exhibited in Figure 27 and Table 19. When the batch size equals to half of the observations of the Dataset which is 10000, the norm of the stochastic gradient performs no oscillation.

Batch	Iteration	Total CPU-time(s)	Average Time(ms)	Accuracy
1024	154	138.5369	899.5907	0.8
2048	158	147.2822	932.1660	0.8
4096	152	130.0817	855.8011	0.8
10000	155	132.3243	853.7057	0.8

Table 19: Performance of SGD

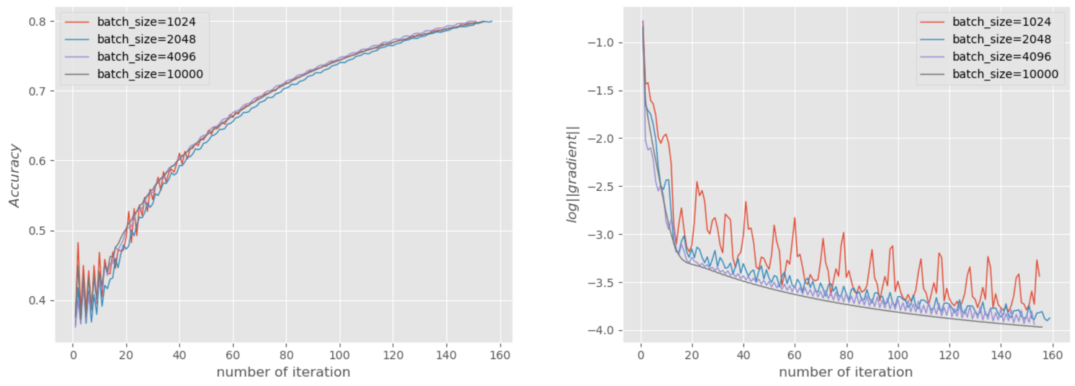


Figure 27: Accuracy and Convergence of SGD