**IE 598: Big Data Optimization** **Fall 2016**

# Lecture 13: Case Study on Logistic Regression – October 06

*Lecturer: Niao He* *Scriber: Jialin Song*

**Overview:** In this lecture we summarize algorithms we have covered so far for smooth convex optimization problems. To better understanding the pros and cons of different algorithms, we conduct a case study on logistic regression model to investigate the their empirical performances.

## 13.1 Summary

We summarize performance of algorithms on smooth convex optimization (including Interior Point Method (IPM), Gradient Descent (GD), Accelerated Gradient Descent (AGD), Projection Gradient Descent (PGD), Frank-Wolfe Algorithm (FW), Block Coordinated Gradient Descent (BCGD)) in terms of convergence rate, iteration complexity and iteration cost in Table 13.1.

Table 13.1: Performance of algorithms on structured convex optimization and smooth convex optimization

| | Algorithm | Convergence Rate | | Iteration Complexity | | Iteration Cost |
|---|---|---|---|---|---|---|
| Structured Convex Optimization (LP/SOCP/DCP) | IPM | $O(\nu \exp(-\frac{t}{\sqrt{\nu}}))$ | | $O(\sqrt{\nu}\log(\frac{1}{\epsilon}))$ | | one Newton step |
| | | Convex | Strongly Convex | Convex | Strongly Convex | |
| Smooth Convex Optimization | GD | $O(\frac{LD^2}{t})$ | $O((\frac{1-\kappa}{1+\kappa})^{2t})$ | $O(\frac{LD^2}{\epsilon})$ | $\frac{L}{\mu}\log(\frac{1}{\epsilon})$ | one gradient |
| | AGD | $O(\frac{LD^2}{t^2})$ | $O((\frac{1-\sqrt{\kappa}}{1+\sqrt{\kappa}})^{2t})$ | $O(\frac{\sqrt{L}D}{\sqrt{\epsilon}})$ | $O(\sqrt{\frac{L}{\mu}}\log(\frac{1}{\epsilon}))$ | one gradient |
| | PGD | $O(\frac{LD^2}{t})$ | $O((1-\frac{\mu}{L})^t)$ | $O(\frac{LD^2}{\epsilon})$ | $\frac{L}{\mu}\log(\frac{1}{\epsilon})$ | one gradient<br>one projection |
| | FW | $O(\frac{LD^2}{t})$ | | $O(\frac{LD^2}{\epsilon})$ | | one gradient<br>one linear minimization |
| | BCGD | $O(\frac{bLD^2)}{t}$ | $O((1-\frac{\mu}{bL})^t)$ | $O(\frac{bLD^2}{\epsilon})$ | $O(\frac{bL}{\mu}log(\frac{1}{\epsilon}))$ | $O(1)$-block gradient (randomized)<br>$O(b)$-block gradient (cyclic)<br>$O(b)$-block gradient (Gauss Southwell) |

Notations:

- L is L-smooth constant
- $\mu$ is strong convexity parameter
- $\kappa$=$\frac{L}{\mu}$ ibs the condition number
- $D$ is either $\|x_0 - x_*\|_2$ or diameter of compact set X.

## 13.2 Logistic Regression

In this section, we first introduce the basic fundamentals of classification models and then formulate logistic regression model.

### 13.2.1 Preliminaries on Classification Model

We consider a binary classification problem: Given $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, $x_i \in R^d$, $y_i \in \{-1, 1\}$. Note that input $x_i$ is called feature vector, output $y_i$ is label. Suppose there is a family of function $f(x, w)$, binary classification aims to specify the best $w$ such that x and y are related.

Based on the input feature vectors and certain type of function we adopt, the output can be predicted based on certain prediction rules. For binary classification problem, the prediction rule can be shown as follows:

$$y = \begin{cases} 1, & f(x,w) \geq 0 \\ -1, & f(x,w) < 0 \end{cases}$$

An error term can be defined as follows to denote whether the output is successfully predicted or not:

$$\text{error} = \begin{cases} 1, & yf(x,w) < 0 \\ 0, & yf(x,w) \geq 0 \end{cases}$$

Obviously our goal is to find the best parameter $w$ in order to minimize the total error in the process of prediction. The optimization step in the binary classification model can be extracted below:

- ◇ empirical risk minimization:

$$\min_w \sum_{i=1}^{n} \mathbb{1}_{\{y_i f(x_i,w)<0\}}$$

- ◇ expected risk minimization:

$$\min_w \mathrm{E}_{(x,y)} \ \mathbb{1}_{\{yf(x,w)<0\}} \Longrightarrow \min_w \mathrm{P}(yf(x,w) < 0)$$

For the purpose of simplicity, we define of 0-1 loss function

$$l(v) = \begin{cases} 1, & v < 0 \\ 0, & v \geq 0 \end{cases}$$

Thus the minimization step in classification model becomes

$$\min_w \sum_{i=1}^{n} l(y_i f_i(x_i, w))$$

which is usually called 0-1 loss minimization problem.

It can be easily seen that 0-1 loss minimization problem is a non-convex optimization problem, which is NP-hard. Instead of solving directly the 0-1 loss minimization problem, one can resort to convex relaxation and solve instead convex upper bound of the 0-1 loss function. We introduce 4 common convex loss functions here:

- hinge loss function: $l(v) = \max\{1 - v, 0\}$
- exponential loss function: $l(v) = \exp(-v)$
- modified least square loss function: $l(v) = \max(1 - v, 0)^2$
- logistic loss function: $l(v) = \log(1 + \exp(-v))$

The comparison between the original non-convex 0-1 loss function and different commonly applied loss functions is shown in Figure 13.1.

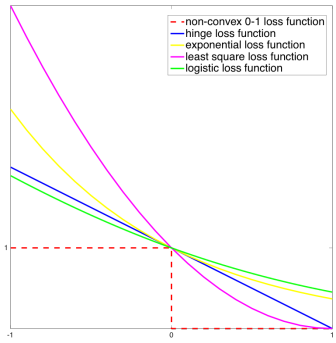


Figure 13.1: 0-1 loss function and common loss functions

### 13.2.2 Logistic Regression Model

When using the logistic loss as the convex surrogate function, we end up with the logistic regression model

$$\min_w f(w) := \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)) + \frac{\lambda}{2}\|w\|_2^2$$

Logistic regression model can also be obtained through MLE by assuming the likelihood

$$P(y = 1|x, w) = \frac{1}{1 + \exp(-w^T x)} = \sigma(w^T x),$$

where $\sigma(u) = \frac{1}{1+\exp(-u)}$ is known as the sigmoid function.

The first order information of regression model is given by

$$\nabla f(w) = \sum_{i=1}^{n} \frac{-y_i x_i \exp(-y_i w^T x_i)}{1 + \exp(-y_i w^T x_i)} + \lambda w = \sum_{i=1}^{n} y_i x_i (\sigma(y_i w^T x_i) - 1) + \lambda w$$

The second order information of regression model is

$$\nabla^2 f(w) = \sum_{i=1}^{n} x_i \sigma(y_i w^T x_i)(1 - \sigma(y_i w^T x_i)) {x_i}^T + \lambda \mathrm{I} = X^T A X + \lambda \mathrm{I}$$

where $A$ is a diagonal matrix with diagonals given by $A_{ii} = \sigma(y_i w^T x_i)(1 - \sigma(y_i w^T x_i)), i = 1, \ldots, n$, $X^T = [x_1, \ldots, x_n]$ is the observation matrix. Note that $0 < A_i i \leq 1/4, \forall i$, hence, we can conclude that

$$L \leq \frac{1}{4}\lambda_{\max}(X^T X) + \lambda.$$

In practice, when $L$ is not known, one can adopt back-tracking (online search) approach to search for the appropriate $L$ until it satisfies the condition $f(x_{t+1}) - f(x_t) \leq \nabla f(x_t)(x_{t+1} - x_t) + \frac{L}{2}\|x_{t+1} - x_t\|^2$. In the following experiments, we will simply use the fixed stepsize $\gamma_t = 1/L$ with the $L$ given above.

## 13.3 Numerical Implementation

We run Python codes to illustrate how to apply our algorithms to solve logistic regression and investigate their empirical performances.

### 13.3.1 Data Generation

We use the following codes to generate our inputs x and outputs y in preparation for doing the logistic regression.

```
N = 100
dim = 30
lamda = 1
sigmoid = lambda x: 1/(1+np.exp(-x))
np.random.seed(50)
w = np.matrix(np.random.multivariate_normal([0.0]*dim, np.eye(dim))).T
X = np.matrix(np.random.multivariate_normal([0.0]*dim, np.eye(dim), size = N))
y = 2 * (np.random.uniform(size = (N, 1)) < sigmoid(X*w)) - 1
```

### 13.3.2 Optimization via CVXPY

Before implementing the algorithm, we apply the CVXPY module solving the optimization problem to specify the optimal solution for sake of conducting comparative investigation later. CVXPY is a Python-embedded modeling language for convex optimization problems based on Interior-Point-Methods. We run the following Python codes to implement CVXPY.

```
import cvxpy as cvx

w = cvx.Variable(dim)
loss = cvx.sum_entries(cvx.logistic(-cvx.mul_elemwise(y, X*w))) + lamda/2 * cvx.sum_squares(w)

problem = cvx.Problem(cvx.Minimize(loss))
problem.solve(verbose=True)
opt = problem.value
```

### 13.3.3 Gradient-based Algorithms

For all the algorithms below, we initialize $w_0 = 0$.

- Gradient Descent

$$w_{t+1} = w_t - \frac{1}{L}\nabla f(w_t)$$

```
w = np.matrix([0.0]*dim).T
for t in range(0, max_iter):
    obj_val = obj(w)
    w = w - 1/L * grad(w)
```

- Accelerated Gradient Descent

$$w_{t+1} = v_t - \frac{1}{L}\nabla f(v_t)$$
$$v_{t+1} = w_{t+1} + \frac{t}{t+3}(w_{t+1} - w_t)$$

```
w = np.matrix([0.0]*dim).T
v = w
for t in range(0, max_iter):
    obj_val = obj(w)
    w_prev = w
    w = v - 1/L * grad(v)
    v = w + t/(t+3) * (w - w_prev)
```

- FISTA

$$w_{t+1} = v_t - \frac{1}{L}\nabla f(v_t)$$
$$v_{t+1} = w_{t+1} + \frac{a_t - 1}{a_{t+1}}(w_{t+1} - w_t)$$

where $a_0 = 0, \quad a_{t+1} = \frac{1+\sqrt{1+4a_t{}^2}}{2}$.

```
w = np.matrix([0.0]*dim).T
v = w
a = 0.0
for t in range(0, max_iter):
    obj_val = obj(w)
    w_prev = w
    w = v - 1/L*grad(v)
    a_prev = a
    a = (1 + np.sqrt(1 + 4 * a_prev**2))/2
    v = w + (a_prev - 1) / a * (w - w_prev)
```

- Block Coordinate Gradient Descent

$$\text{pick i from } i \in \{1, 2, ..., n\}$$
$$w_{t+1} = w_t - \frac{1}{L}U_i\nabla_i f(w_t)$$

◇ Cyclic Coordinate Gradient Descent

```
w = np.matrix([0.0]*dim).T
for t in range(0, max_iter):
    obj_val = obj(w)
    for i in range(0,dim):
        w[i] = w[i] - 1/L * grad(w)[i]
```

◇ Gauss-Southwell Coordinate Gradient Descent

```
w = np.matrix([0.0]*dim).T
for t in range(0, max_iter):
```

```
    obj_val = obj(w)
    i = np.argmax(abs(grad(w)))
    w[i] = w[i] - 1/L * grad(w)[i]
```

◇ Randomized Coordinate Gradient Descent

```
w = np.matrix([0.0]*dim).T
for t in range(0, max_iter):
    obj_val = obj(w)
    for i in range(0, dim):
        i = np.random.randint(0, dim)
        w[i] = w[i] - 1/L * grad(w)[i]
```

### 13.3.4 Numerical Comparisons

Next we compare the convergence performance of different algorithms. Particularly we want to examine the effect of strong convexity and choices of stepsizes on the convergence rate of different algorithms. We can set the $\lambda$ to be 1 to make the regression loss function strongly convex, and set $\lambda$ to be 0 for the convex case.

**Gradient Descent and Accelerated Gradient Descent**

First we compare the convergence performance of gradient descent and accelerated gradient descent under strongly convex and convex case respectively. We plot the objective error in each iteration step

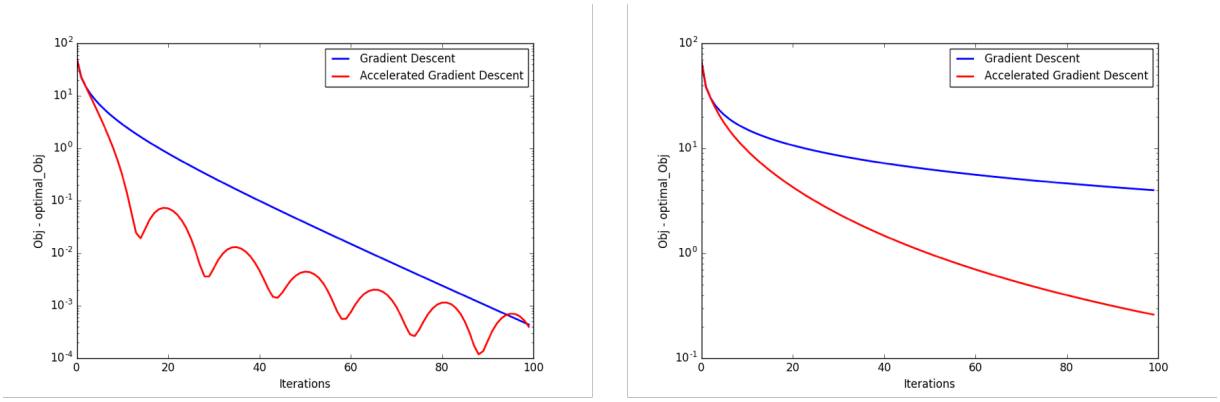


Figure 13.2: Convergence performance of GD and AGD

As you can see from Figure 13.2, accelerated gradient descent (AGD) always achieves a better convergence performance compared with the gradient descent (GD) under both strongly convex case and convex case. It is noticeable that the convergence performance is rather stable and robust in the convex case. In the strongly convex case, convergence error curve end up with drastic oscillation.

**Nesterov's Accelerated Gradient Descent and FISTA**

We pay attention to two different kinds of accelerated gradient descent methods, i.e., AGD and FISTA. As you can see from Figure 13.3, both AGD and FISTA have pretty similar convergence performance in the convex case and strongly convex case.

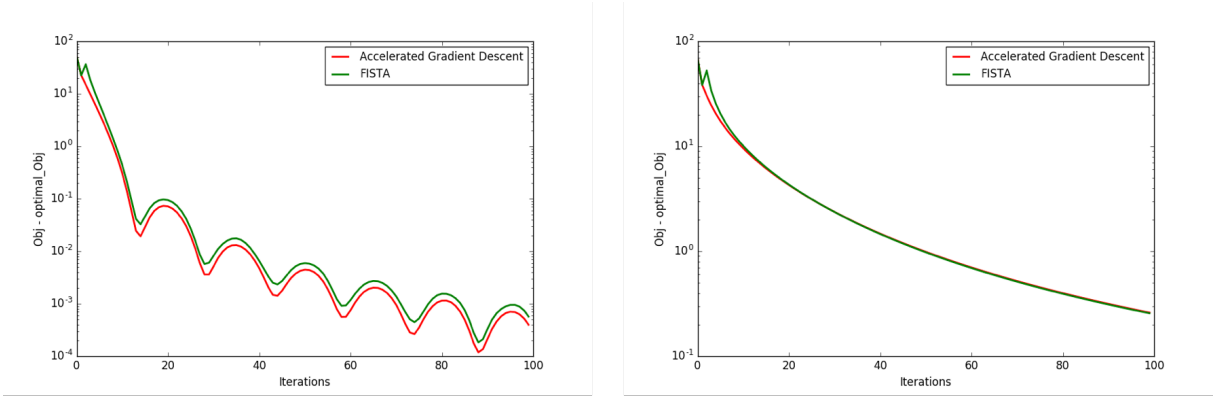


Figure 13.3: Convergence performance of AGD and FISTA

**Coordinated Gradient Descent**

We also analyze several coordinated gradient descent algorithms with Gauss-Southwell, randomized and cyclic updating rules. We adopt the strongly convex assumption ($\lambda = 1$) and vary the stepsize to explore the convergence performance under small ($\gamma_t = 1/L$) and big stepsizes ($\gamma_t = 1/L_i$).

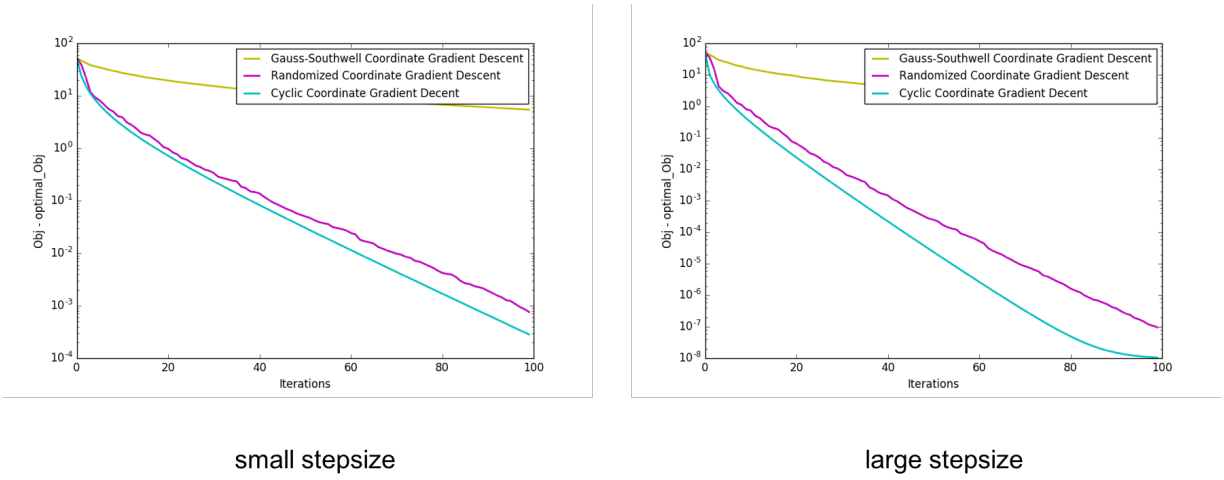


Figure 13.4: Convergence performance of CGD

As Figure 13.4 suggests, Both randomized-CD and cyclic-CD perform far better than Gauss-Southwell-CD, though cyclic-CD outweighs the randomized-CD a bit in both small stepsize case and large stepsize case.

Note that the larger iteration stepsize can lead to more accurate result compared with the small iteration stepsize.

**Overall Performance**

We have analyzed and compared the performance of 5 different algorithms that can be applied to solve logistic regression problem. Naturally we tend to understand which algorithm behaves the best. Note that our comparison is conducted under the strongly convex case. Like previous case, we consider both a small and large stepsize for the two variants of coordinated gradient descent algorithms.

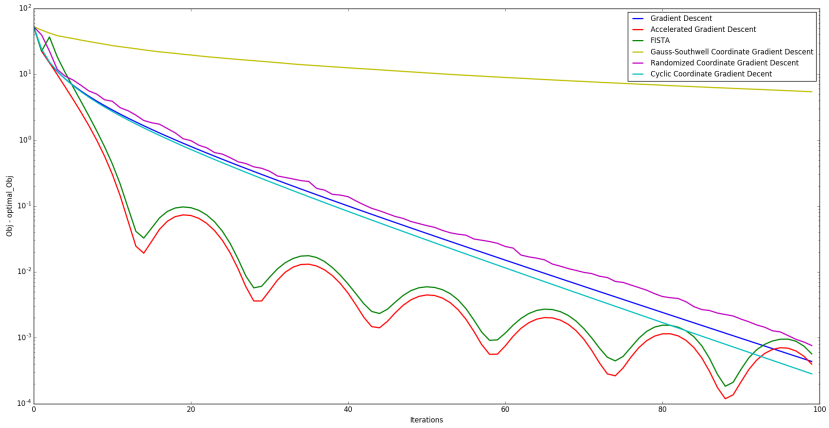


Figure 13.5: Convergence performance summary of 5 different algorithms under small stepsize

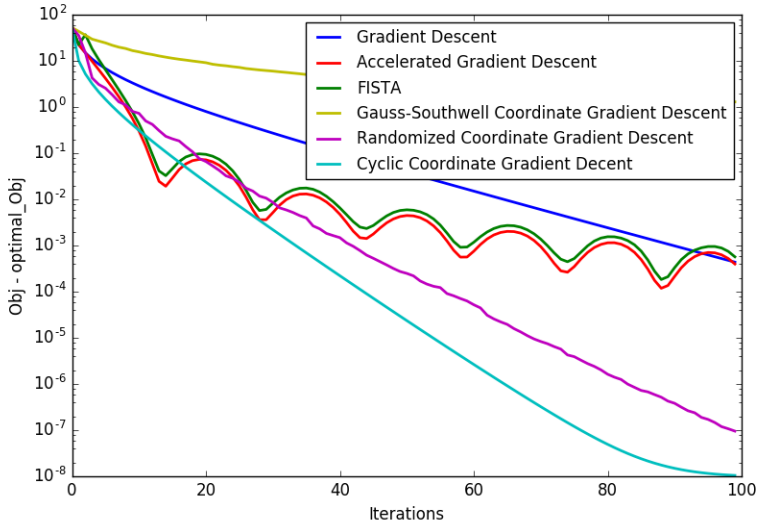


Figure 13.6: Convergence performance summary of 5 different algorithms

Figure 13.5 and 13.6 suggests that when large stepsize is adopted, cyclic-CD and randomized-CD can perform

much better than AGD and FISTA.

There are still several points that haven't been completely addressed in the numerical case study. We see how different stepsize could change drastically on the behavior for CD algorithms. How robust are these algorithms? What happens if we change the stepsize for GD, AGD? Under which regimes would CD be favorable comparing to GD, or cyclic-CD be favorable comparing to randomized-CD? To make a proper conclusion on which algorithm works best among all certainly requires a deeper investigation.