华南理工大学

South China University of Technology

The Experiment Report of
Machine Learning

College Software College

Subject Software Engineering

Members Kenan Ye

Student ID 201530613467

E-mail Conan.ye@gmail.com

Tutor Mingkui Tan

Date submitted 2017.12.14

1. Topic:

Comparison of Various Stochastic Gradient Descent Methods for Solving Classification Problems

2. Time:

2017-12-02 2:00-5:00 PM B7-138/238

3. Reporter:

Kenan Ye

4. Purposes:

Compare and understand the difference between gradient descent and stochastic gradient descent.

Compare and understand the differences and relationships between Logistic regression and linear classification.

Further understand the principles of SVM and practice on larger data.

5. Data sets and data analysis:

Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. Please download the training set and validation set.

6. Experimental steps:

Logistic Regression and Stochastic Gradient Descent

1) Load the training set and validation set.
2) Initialize logistic regression model parameters, you can consider initializing zeros, random numbers or normal distribution.
3) Select the loss function and calculate its derivation, find more detail in PPT.
4) Calculate gradient G toward loss function from **partial samples.**
5) Update model parameters using different optimized methods(NAG，RMSProp，AdaDelta and Adam).
6) Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Predict under validation set and get the different optimized method loss L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam}.
7) Repeat step 4 to 6 for several times, and drawing graph of L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam}. with the number of iterations.

Linear Classification and Stochastic Gradient Descent

1) Load the training set and validation set.
2) Initialize SVM model parameters, you can consider initializing zeros, random numbers or normal distribution.
3) Select the loss function and calculate its derivation, find more detail in PPT.
4) Calculate gradient G toward loss function from **partial samples.**
5) Update model parameters using different optimized methods(NAG，RMSProp，AdaDelta and Adam).
6) Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Predict under validation set and get the different optimized method loss L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam}.
7) Repeat step 4 to 6 for several times, and drawing graph of L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam}. with the number of iterations.

7. Code:

1) Code of Logistic Regression (only include gradient decent)

```
    for i in range(iteration_num):
        # print current iteration number
        print("\rProducing (" + str(i + 1) + "/" + str(iteration_num) + ")", end="")

        # define batch number
        batch_number = 100

        # select samples' index
        index = random.randint(0, y_train.size - batch_number)

        # calculate gradient
        SGD_gradient = calculate_gradient(X_train[i:i + batch_number], y_train[i:i + batch_number], SGD_omega)
        NAG_gradient = calculate_gradient(X_train[i:i + batch_number], y_train[i:i + batch_number], NAG_omega + mu * momentum)
        AdaDelta_gradient = calculate_gradient(X_train[i:i + batch_number], y_train[i:i + batch_number], AdaDelta_omega)
        RMSProp_gradient = calculate_gradient(X_train[i:i + batch_number], y_train[i:i + batch_number], RMSProp_omega)
        Adam_gradient = calculate_gradient(X_train[i:i + batch_number], y_train[i:i + batch_number], Adam_omega)

        # calculate new momentum (NAG)
        momentum = mu * momentum - eta * NAG_gradient

        # calculate E_Theta and E -g (AdaDelta)
        Ada_E_Theta = gamma * Ada_E_Theta + (1 - gamma) * np.dot(Ada_delta_theta, Ada_delta_theta)
        Ada_E_g = gamma * Ada_E_g + (1 - gamma) * np.dot(AdaDelta_gradient, AdaDelta_gradient)

        # calculate delta theta (AdaDelta)
        Ada_delta_theta = (-np.sqrt(Ada_E_Theta + epsilon) / np.sqrt(Ada_E_g + epsilon)) * AdaDelta_gradient
```

Figure 1 Code of Logistic Regression

```
# calculate E_Theta and E -g (RMSProp)
RMS_E_Theta = gamma * RMS_E_Theta + (1 - gamma) * np.dot(RMS_delta_theta, RMS_delta_theta)
RMS_E_g = gamma * RMS_E_g + (1 - gamma) * np.dot(RMSProp_gradient, RMSProp_gradient)

# calculate delta theta (RMSProp)
RMS_delta_theta = (- eta / np.sqrt(RMS_E_g + epsilon)) * RMSProp_gradient

# calculate mt and vt (Adam)
m_t = beta_1 * m_t + (1 - beta_1) * Adam_gradient
v_t = beta_2 * v_t + (1 - beta_2) * np.dot(Adam_gradient, Adam_gradient)

# revises mt and vt (Adam)
m_t_r = m_t / (1 - np.power(beta_1, i + 1))
v_t_r = v_t / (1 - np.power(beta_2, i + 1))

# calculate delta theta (Adam)
Adam_delta_theta = (- eta / (np.sqrt(v_t_r) + epsilon)) * m_t_r

# update parameters
SGD_omega = SGD_omega - eta * SGD_gradient
NAG_omega = NAG_omega + momentum
AdaDelta_omega = AdaDelta_omega + Ada_delta_theta
RMSProp_omega = RMSProp_omega + RMS_delta_theta
Adam_omega = Adam_omega + Adam_delta_theta
```

Figure 2 Code of Logistic Regression

2) Code of Linear Classification (only include gradient decent)

```
    for i in range(iteration_num):
        # print current iteration number
        print("\rProducing (" + str(i + 1) + "/" + str(iteration_num) + ")", end="")

        # define batch number
        batch_number = 100

        # select samples' index
        index = random.randint(0, y_train.size - batch_number)

        # calculate gradient
        SGD_gradient = calculate_gradient(X_train[i:i + batch_number], y_train[i:i + batch_number], SGD_omega)
        NAG_gradient = calculate_gradient(X_train[i:i + batch_number], y_train[i:i + batch_number], NAG_omega + mu * momentum)
        AdaDelta_gradient = calculate_gradient(X_train[i:i + batch_number], y_train[i:i + batch_number], AdaDelta_omega)
        RMSProp_gradient = calculate_gradient(X_train[i:i + batch_number], y_train[i:i + batch_number], RMSProp_omega)
        Adam_gradient = calculate_gradient(X_train[i:i + batch_number], y_train[i:i + batch_number], Adam_omega)

        # calculate new momentum (NAG)
        momentum = mu * momentum - eta * NAG_gradient

        # calculate E_Theta and E -g (AdaDelta)
        Ada_E_Theta = gamma * Ada_E_Theta + (1 - gamma) * np.dot(Ada_delta_theta, Ada_delta_theta)
        Ada_E_g = gamma * Ada_E_g + (1 - gamma) * np.dot(AdaDelta_gradient, AdaDelta_gradient)

        # calculate delta theta (AdaDelta)
        Ada_delta_theta = (-np.sqrt(Ada_E_Theta + epsilon) / np.sqrt(Ada_E_g + epsilon)) * AdaDelta_gradient
```

Figure 3 Code Linear Classification

```
# calculate E_Theta and E -g (RMSProp)
RMS_E_Theta = gamma * RMS_E_Theta + (1 - gamma) * np.dot(RMS_delta_theta, RMS_delta_theta)
RMS_E_g = gamma * RMS_E_g + (1 - gamma) * np.dot(RMSProp_gradient, RMSProp_gradient)

# calculate delta theta (RMSProp)
RMS_delta_theta = (- eta / np.sqrt(RMS_E_g + epsilon)) * RMSProp_gradient

# calculate mt and vt (Adam)
m_t = beta_1 * m_t + (1 - beta_1) * Adam_gradient
v_t = beta_2 * v_t + (1 - beta_2) * np.dot(Adam_gradient, Adam_gradient)

# revises mt and vt (Adam)
m_t_r = m_t / (1 - np.power(beta_1, i + 1))
v_t_r = v_t / (1 - np.power(beta_2, i + 1))

# calculate delta theta (Adam)
Adam_delta_theta = (- eta / (np.sqrt(v_t_r) + epsilon)) * m_t_r

# update parameters
SGD_omega = SGD_omega - eta * SGD_gradient
NAG_omega = NAG_omega + momentum
AdaDelta_omega = AdaDelta_omega + Ada_delta_theta
RMSProp_omega = RMSProp_omega + RMS_delta_theta
Adam_omega = Adam_omega + Adam_delta_theta
```

Figure 4 Code of Linear Classification

8. The initialization method of model parameters:

Both experiment initialize parameters with Zeros

9. The selected loss function and its derivatives:

1) Logistic Regression

Assume that the labels are binary: $y_i \in \{0,1\}$

$$h_w(X) = g(W^T X) = \frac{1}{1 + e^{-W^T X}}$$

Then, probability can be expressed as follows:

$$p = \begin{cases} h_w(X_I) & y_i = 1 \\ 1 - h_w(X_i) & y_i = 0 \end{cases}$$

Then the optimizing problem becomes maximum the probability of all samples by fitting the parameters of model with data set.

That is:

$$\max \prod_{\mathrm{i}=1}^{\mathrm{n}} P(y_i|X_i) \Leftrightarrow maxlog\left(\prod_{i=1}^{n} P(y_i|X_i)\right)$$

$$\equiv \max \sum_{\mathrm{i}=1}^{\mathrm{n}} logP(y_i|X_i)$$

$$\Leftrightarrow \min -\frac{1}{\mathrm{n}}\sum_{i=1}^{n} logP(y_i|X_i)$$

In the expression below:

$$\mathrm{P}(\mathrm{y_i}|X_i) = h_w(X_i)^{y_i} \cdot \left(1 - h_w(X_i)\right)^{1-y_i}$$

Consequently, the loss function is:

$$\mathrm{J(W)} = -\frac{1}{n}\left(\sum_{i=1}^{n} y_i log h_w(X_i) + (1-y_i)\log\left(1 - h_w(X_i)\right)\right)$$

For a sample, the gradient is:

$$\frac{\partial \mathrm{J(W)}}{\partial \mathrm{W}} = -\frac{1}{\partial \mathrm{W}} \cdot \partial\left(y \cdot log h_w(X) + (1-y) \cdot \log\left(1 - h_w(X)\right)\right)$$

$$= -\mathrm{y} \cdot \frac{1}{\mathrm{h_w}(X)} \cdot \frac{\partial h_w(W)}{\partial W} + (1-\mathrm{y}) \cdot \frac{1}{1-\mathrm{h_w(X)}} \cdot \frac{\partial h_w(X)}{\partial W}$$

$$= -\mathrm{y} \cdot \frac{1}{\mathrm{h_w}(X)} \cdot \frac{\partial h_w(X)}{\partial W} + (1-y) \cdot \frac{1}{1-h_w(X)} \cdot \frac{\partial h_w(X)}{\partial W}$$

$$= -\mathrm{y} \cdot \frac{1}{\mathrm{h_w}(X)} \cdot \frac{\partial g(W^T X)}{\partial W} + (1-y) \cdot \frac{1}{1-h_w(X)} \cdot \frac{\partial g(W^T X)}{\partial W}$$

$$= \left(-\frac{\mathrm{Xy}}{\mathrm{h_w}(X)} + \frac{x(1-y)}{1-h_w(X)}\right) \cdot g(W^T X) \cdot [1 - g(W^T X)]$$

$$= (\mathrm{h_w}(X) - y)X$$

For several samples:

$$\frac{\partial \mathrm{J(W)}}{\partial \mathrm{W}} = \frac{1}{\mathrm{n}}\sum_{i=1}^{n} (h_w(X_i) - y)X_i$$

$$= \frac{1}{\mathrm{n}}(h_w(X) - y) \cdot X$$

$$\mathrm{W} \coloneqq \mathrm{W} - \frac{1}{\mathrm{n}}\sum_{i=1}^{n} \alpha(h_w(X_i) - y_i)X_i$$

$$= \mathrm{W} - \frac{1}{\mathrm{n}}(h_w(X) - y) \cdot X$$

2) Linear Classification

For getting best classification performance in unseen data set, we use support vector machine to do the linear classification (SVM), which selects two parallel hyperplanes that separate the two classes of data and let the distance between them as large as possible. The region bounded by these two hyperplanes is called the "margin"

Margin:

$$\frac{\mathrm{w}}{||\mathrm{w}||}(\mathrm{x}_+ - \mathrm{x}_-) = \frac{\mathrm{w}^{\mathrm{T}}(\mathrm{x}_+ - \mathrm{x}_-)}{||\mathrm{x}||} = \frac{2}{||\mathrm{w}||}$$

Therefore, learning the SVM can be formulated as an optimization:

$$\max_{\mathrm{w,b}} \frac{2}{||\mathrm{w}||}$$

$$\text{s.t.} \quad \mathrm{w}^{\mathrm{T}}\mathrm{x}_{\mathrm{i}} + \mathrm{b} \begin{cases} \geq 1 & \mathrm{y} = +1 \\ \leq 1 & \mathrm{y} = -1 \end{cases}$$

However, data set given might, and usually certainly, have noises. If just maximum the margin the model will overfitting. In general, there is a trade-off between the margin and the number of mistakes on the training data.

Therefore, we introduce $\xi_{\mathrm{i}} \geq 0$, for each i, which represents how much example i is on wrong side of margin boundary.

- If $\xi_{\mathrm{i}} = 0$ then it is ok.
- If $0 < \xi_{\mathrm{i}} < 1$ it is correctly classified, but with a smaller margin than $\frac{1}{||\mathrm{w}||}$
- If $\xi_{\mathrm{i}} > 1$ then it is in correctly classified.

The optimization problems become:

$$\min_{\mathrm{w,b}} \frac{||\mathrm{w}||^2}{2} + \mathrm{C}\sum_{\mathrm{i}=1}^{\mathrm{n}} \xi_{\mathrm{i}}$$

$$\text{s.t.} \quad \mathrm{y_i}(\mathrm{w}^{\mathrm{T}}\mathrm{x_i} + \mathrm{b}) \geq 1 - \xi_{\mathrm{i}}, \quad \mathrm{i} = 1,2,\dots,\mathrm{n}$$

Hinge loss = $\xi_{\mathrm{i}} = \max(0, 1 - \mathrm{y_i}(\mathrm{w}^{\mathrm{T}}\mathrm{x_i} + \mathrm{b}))$

The optimization problems become

$$\min_{\mathrm{w,b}} \frac{||\mathrm{w}||^2}{2} + \mathrm{C}\sum_{\mathrm{i}=1}^{\mathrm{n}} \max(0, 1 - \mathrm{y_i}(\mathrm{w}^{\mathrm{T}}\mathrm{x_i} + \mathrm{b}))$$

Gradient of optimization function:

$$\nabla \mathrm{f} = \begin{bmatrix} \nabla_{\mathrm{w}} \mathrm{f}(\mathrm{w}, \mathrm{b}) \\ \nabla_{\mathrm{b}} \mathrm{f}(\mathrm{w}, \mathrm{b}) \end{bmatrix}$$

$$\mathrm{w} = [w_1, w_2, \dots, w_\mathrm{n}]^\mathrm{T}$$

$$\left|\left|\mathrm{w}\right|\right|^2 = \left|\left|\mathrm{w}\right|\right|_2^2 = w_1^2 + w_2^2 + \cdots + w_\mathrm{n}^2$$

Therefore:

$$\frac{\partial\left(\left|\left|\mathrm{w}\right|\right|^2\right)}{\partial \mathrm{w}} = \left[\frac{\partial(w_1^2 + w_2^2 + \cdots + \mathrm{w}_\mathrm{n}^2)}{\partial w_1}, \dots, \frac{\partial(w_1^2 + w_2^2 + \cdots + \mathrm{w}_\mathrm{n}^2)}{\partial w_\mathrm{n}}\right]^\mathrm{T}$$

$$= [2w_1, \dots, 2w_\mathrm{n}]$$

$$= 2\mathrm{w}$$

Let $\mathrm{g_w}(\mathrm{x_i}) = \frac{\partial\left(\max\left(0, 1-\mathrm{y_i}(\mathrm{w^T x_i}+\mathrm{b})\right)\right)}{\partial \mathrm{w}}$, *so:*

$$\mathrm{g_w}(\mathrm{x_i}) = \begin{cases} -\mathrm{y_i x_i} & 1-\mathrm{y_i}(\mathrm{w^T x_i}+\mathrm{b}) \geq 0 \\ 0 & 1-\mathrm{y_i}(\mathrm{w^T x_i}+\mathrm{b}) < 0 \end{cases}$$

Let $\mathrm{g_b}(\mathrm{x_i}) = \frac{\partial\left(\max\left(0, 1-\mathrm{y_i}(\mathrm{w^T x_i}+\mathrm{b})\right)\right)}{\partial \mathrm{b}}$, *so:*

$$\mathrm{g_b}(\mathrm{x_i}) = \begin{cases} -\mathrm{y_i} & 1-\mathrm{y_i}(\mathrm{w^T x_i}+\mathrm{b}) \geq 0 \\ 0 & 1-\mathrm{y_i}(\mathrm{w^T x_i}+\mathrm{b}) < 0 \end{cases}$$

Consequently,

$$\frac{\partial \mathrm{f(w,b)}}{\partial \mathrm{w}} = \mathrm{w} + \mathrm{C}\sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{g_w}(\mathrm{x_i})$$

$$\frac{\partial \mathrm{f(w,b)}}{\partial \mathrm{b}} = \mathrm{w} + \mathrm{C}\sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{g_b}(\mathrm{x_i})$$

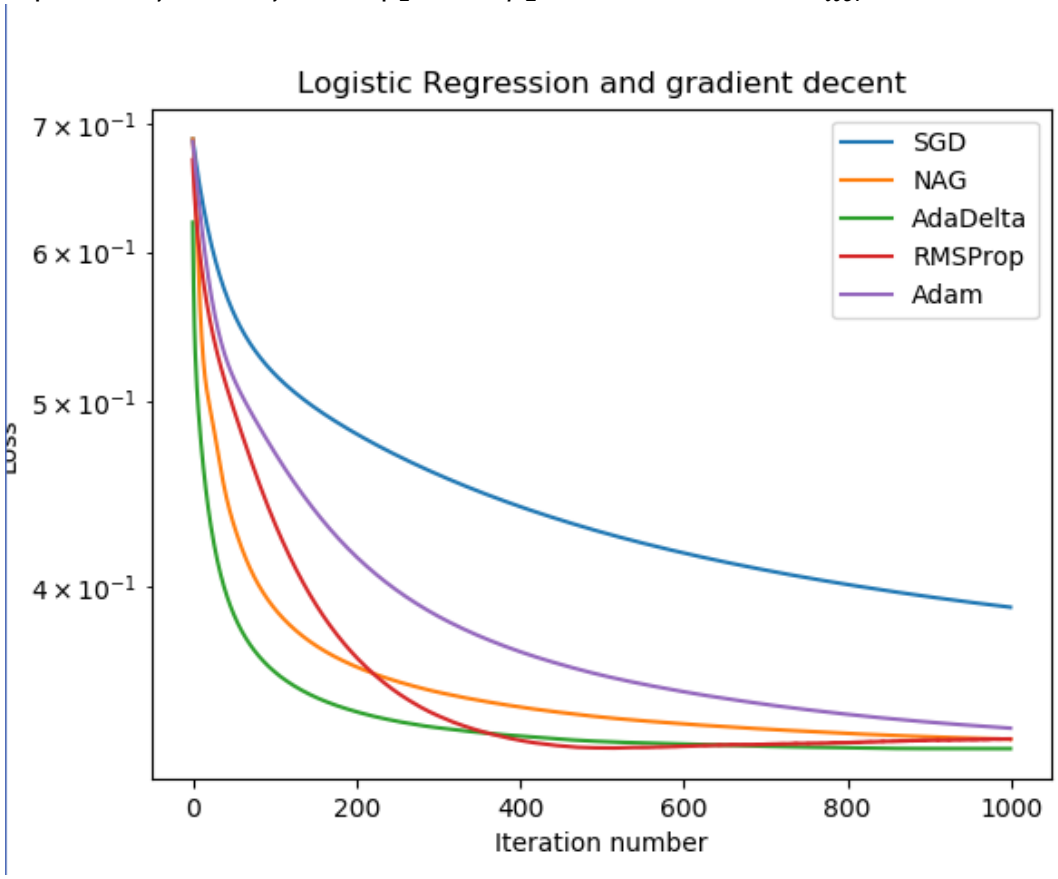
Because expressing w and b is inconvenient, we can add a column with ones to the training data X, and append b to the end to parameters w.

10.Experimental results and curve:

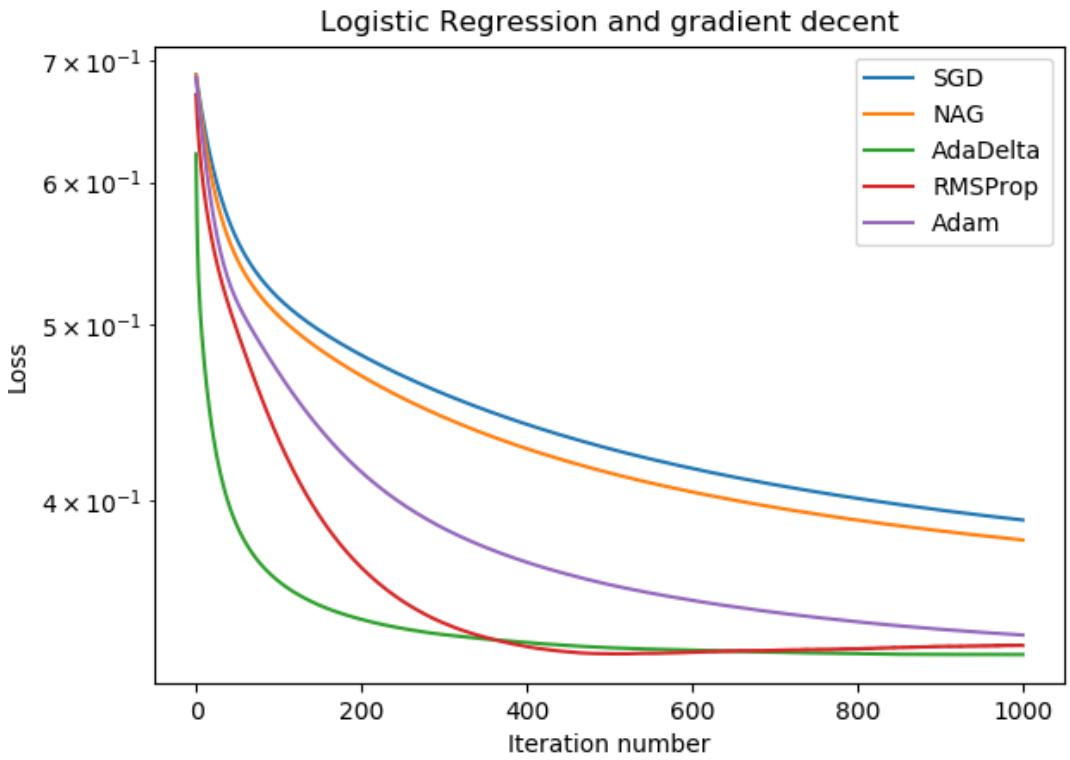
Hyper-parameter selection:

1) Logistic Regression

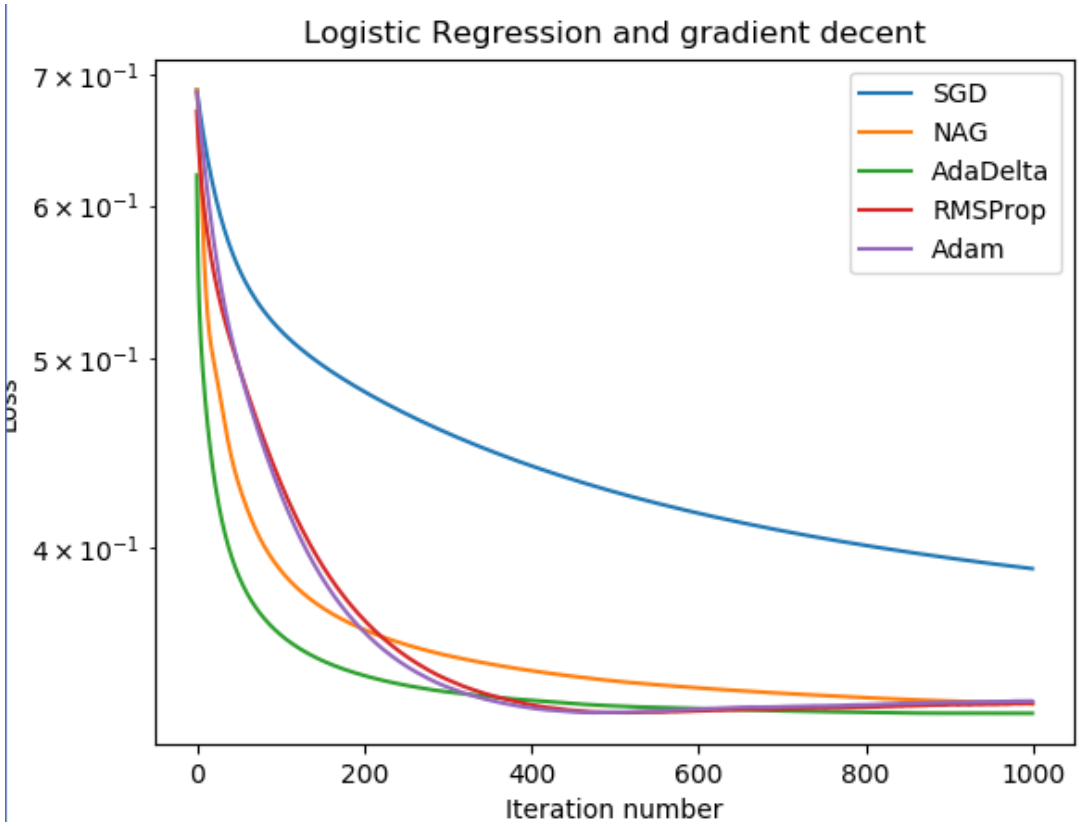
$\eta = 0.01, \mu = 0.9, \gamma = 0.9, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, n_{iter} = 1000$



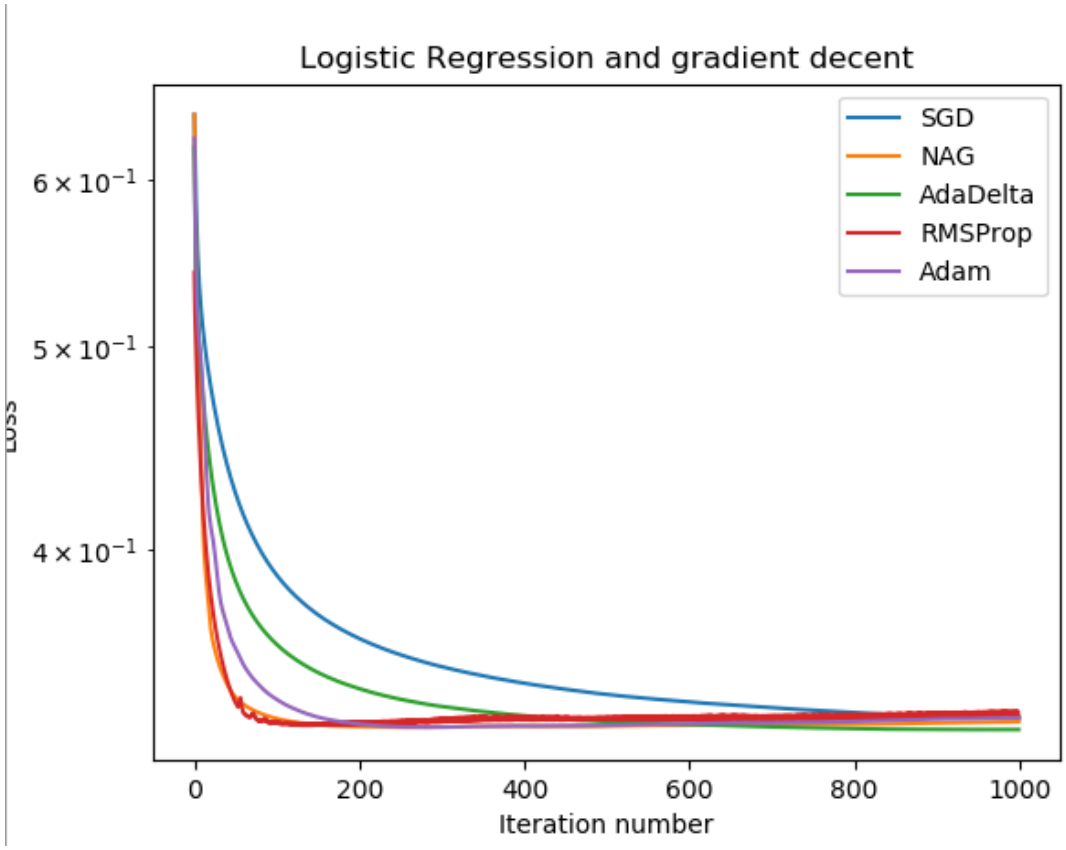
$\eta = 0.01, \mu = 0.2, \gamma = 0.9, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, n_{iter} = 1000$



$\eta = 0.01, \mu = 0.9, \gamma = 0.9, \beta_1 = 0.2, \beta_2 = 0.5, \epsilon = 10^{-8}, n_{iter} = 1000$

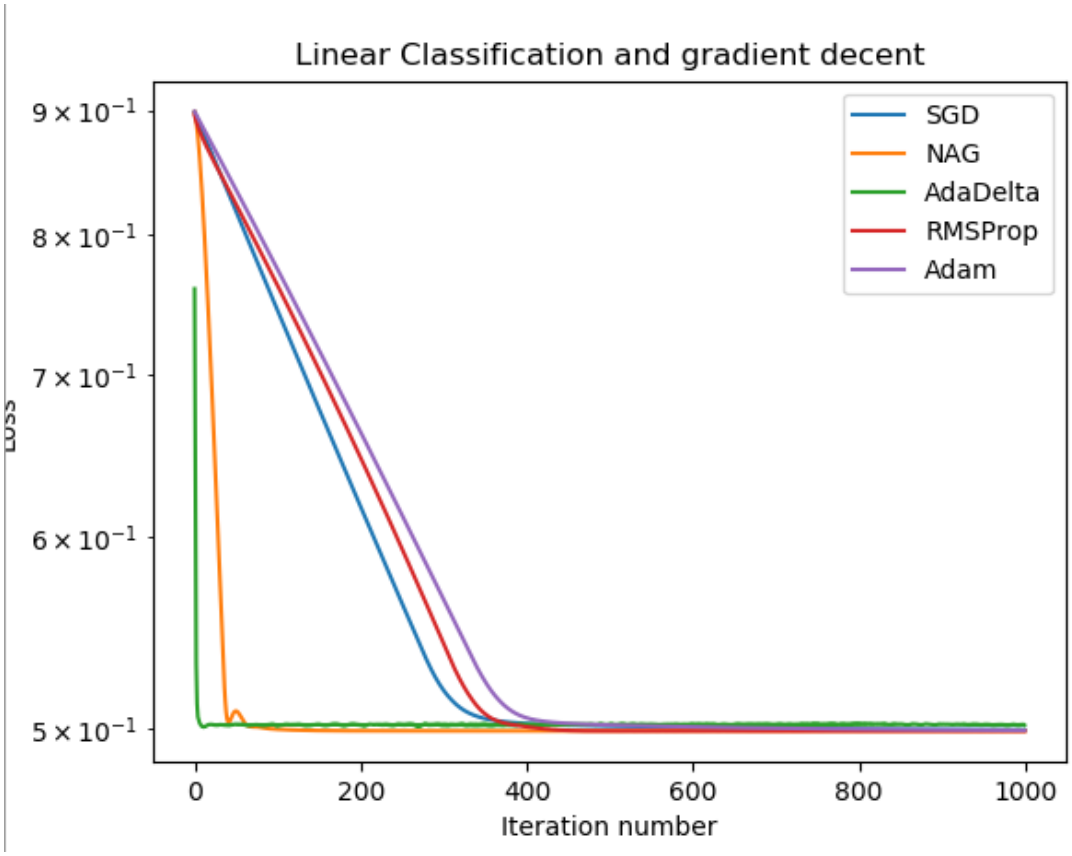


$\eta = 0.1, \mu = 0.9, \gamma = 0.9, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, n_{iter} = 1000$

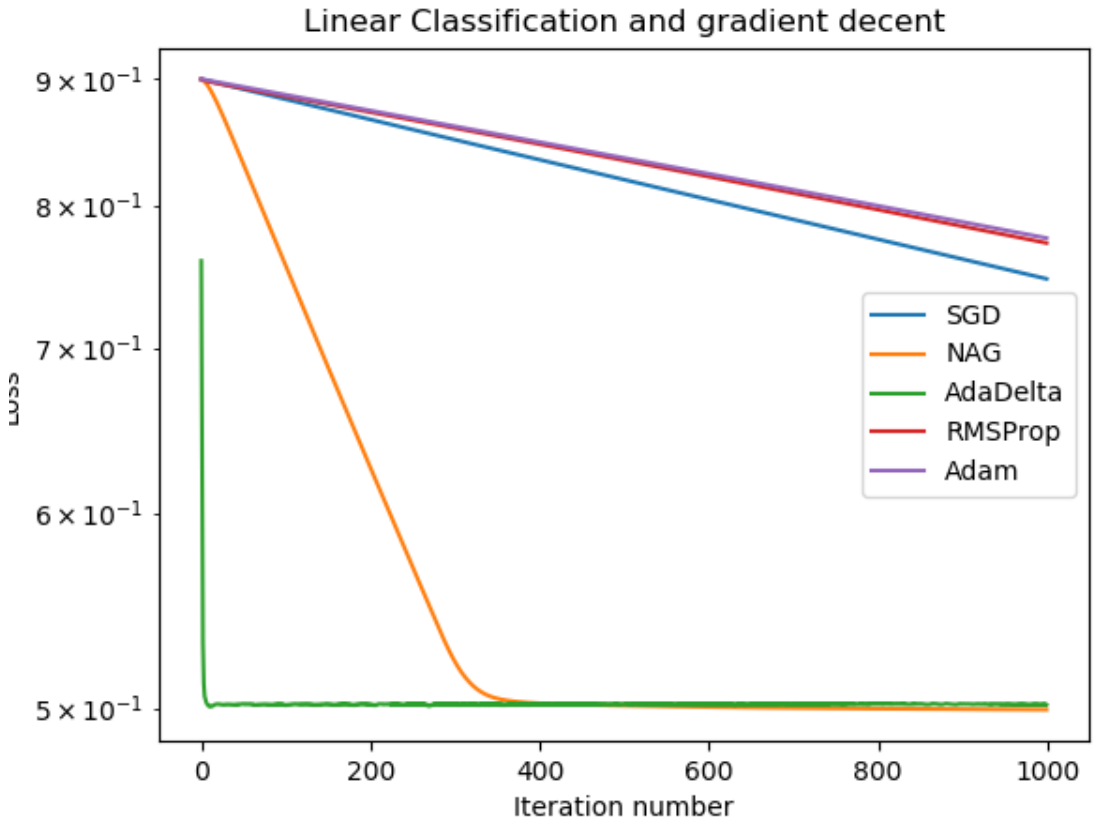


2) Linear Classification

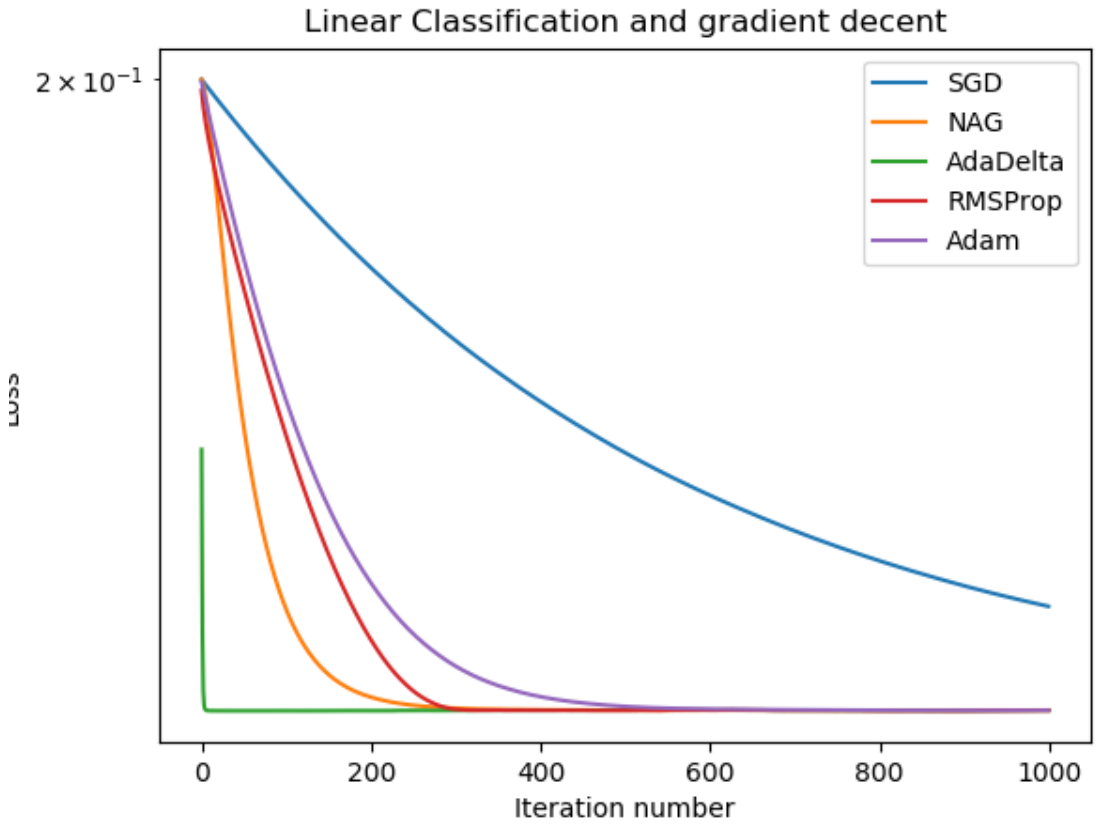
$\eta = 0.001, C = 0.9, \mu = 0.9, \gamma = 0.9, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, n_{iter} = 1000$



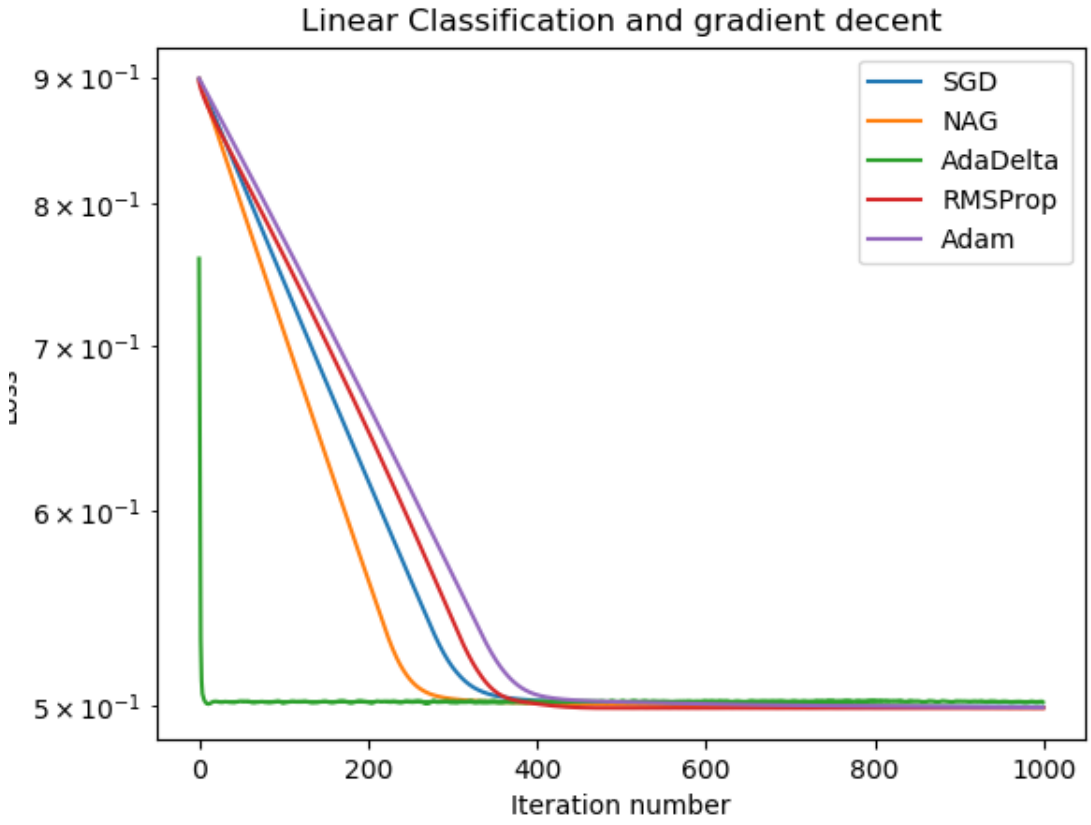
$\eta = 0.0001, C = 0.9, \mu = 0.9, \gamma = 0.9, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, n_{iter} = 1000$



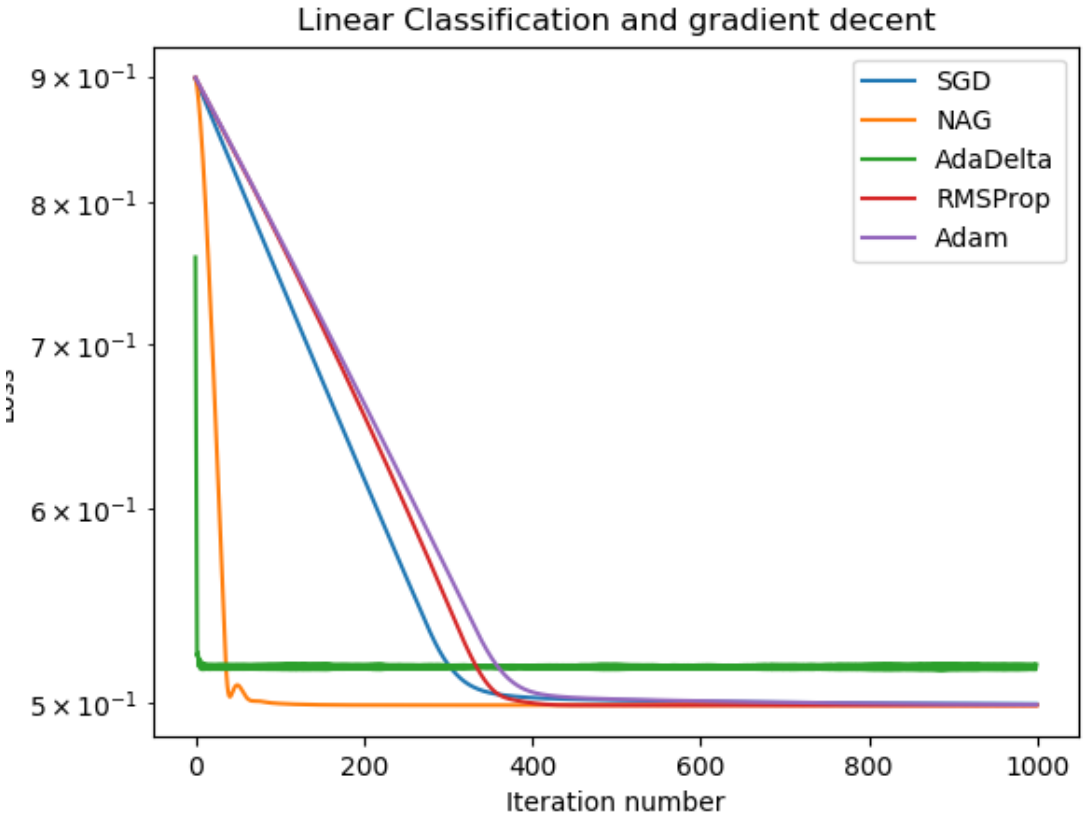
$\eta = 0.001, C = 0.2, \mu = 0.9, \gamma = 0.9, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, n_{iter} = 1000$



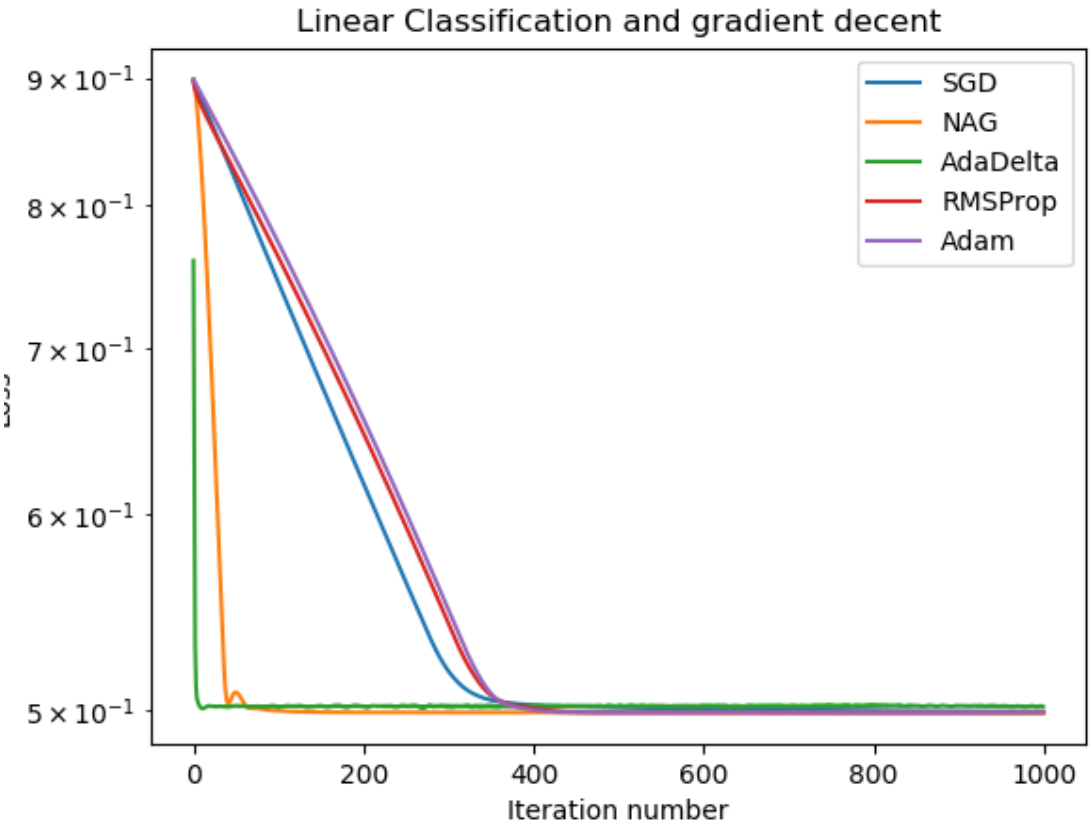
$\eta = 0.001, C = 0.9, \mu = 0.2, \gamma = 0.9, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, n_{iter} = 1000$



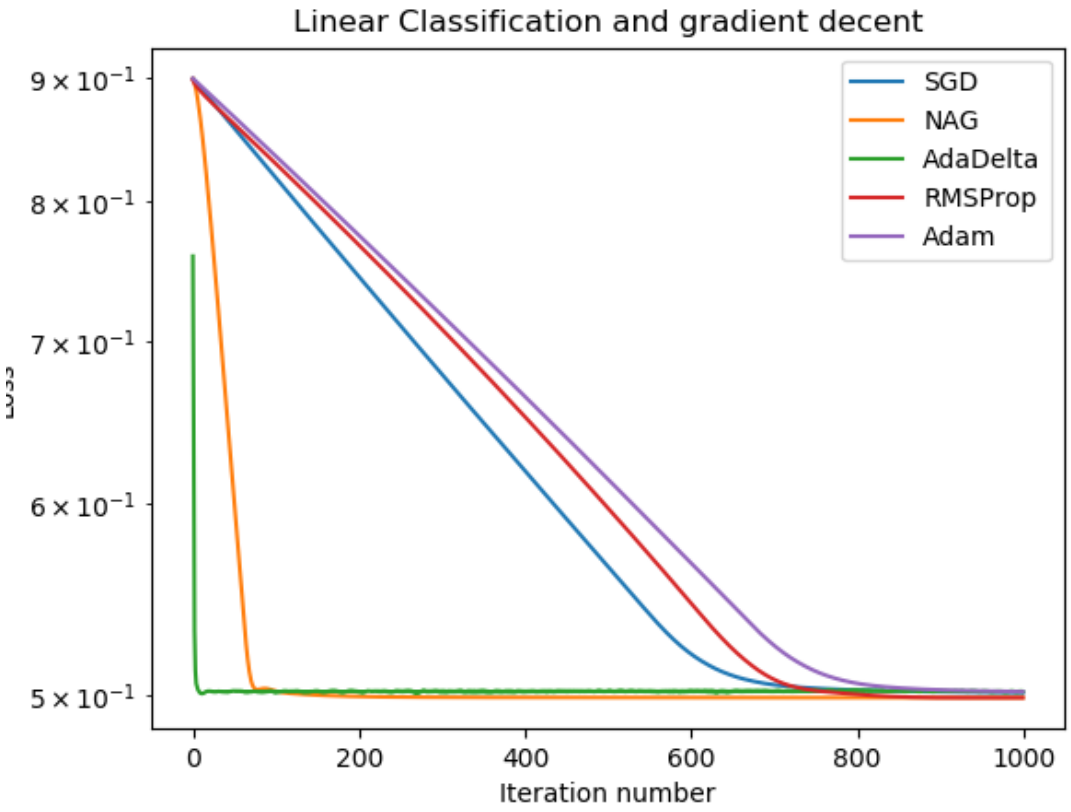
$\eta = 0.001, C = 0.9, \mu = 0.9, \gamma = 0.2, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, n_{iter} = 1000$



$\eta = 0.001, C = 0.9, \mu = 0.9, \gamma = 0.9, \beta_1 = 0.2, \beta_2 = 0.5, \epsilon = 10^{-8}, n_{iter} = 1000$



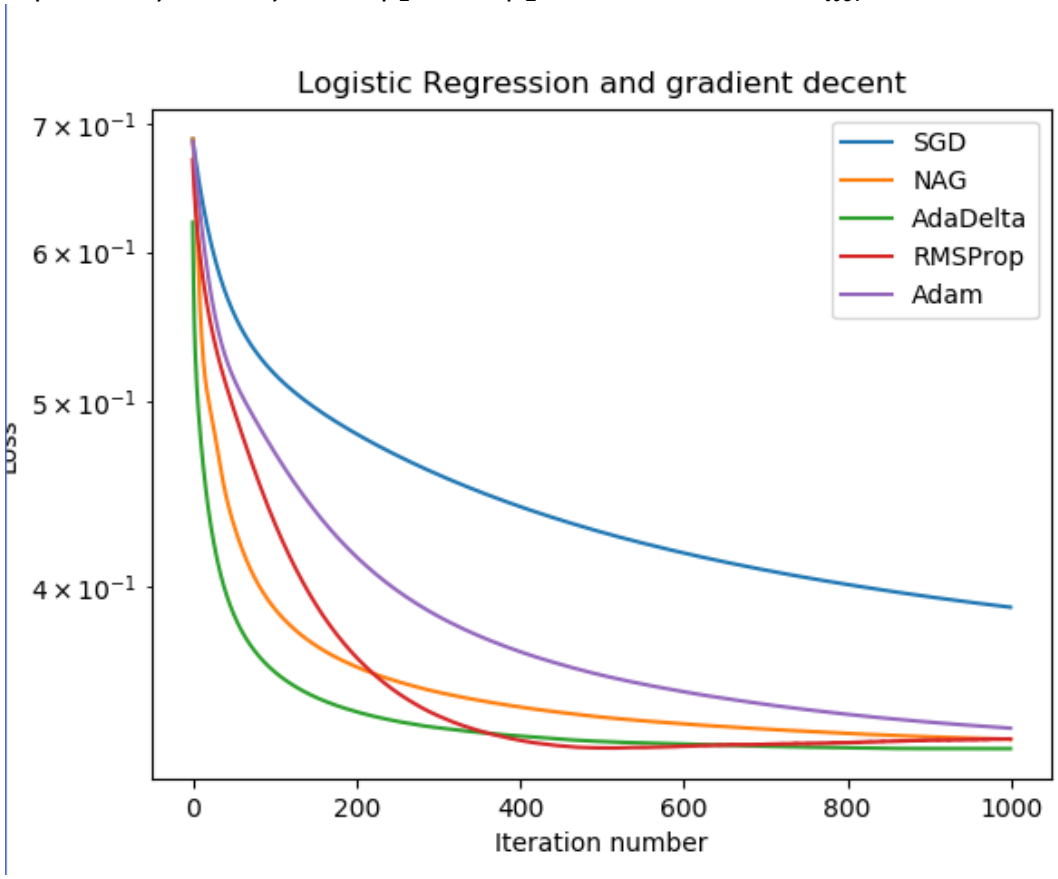
$\eta = 0.0005, C = 0.9, \mu = 0.9, \gamma = 0.9, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, n_{iter} = 1000$



Predicted Results (Best Results) and Loss curve:

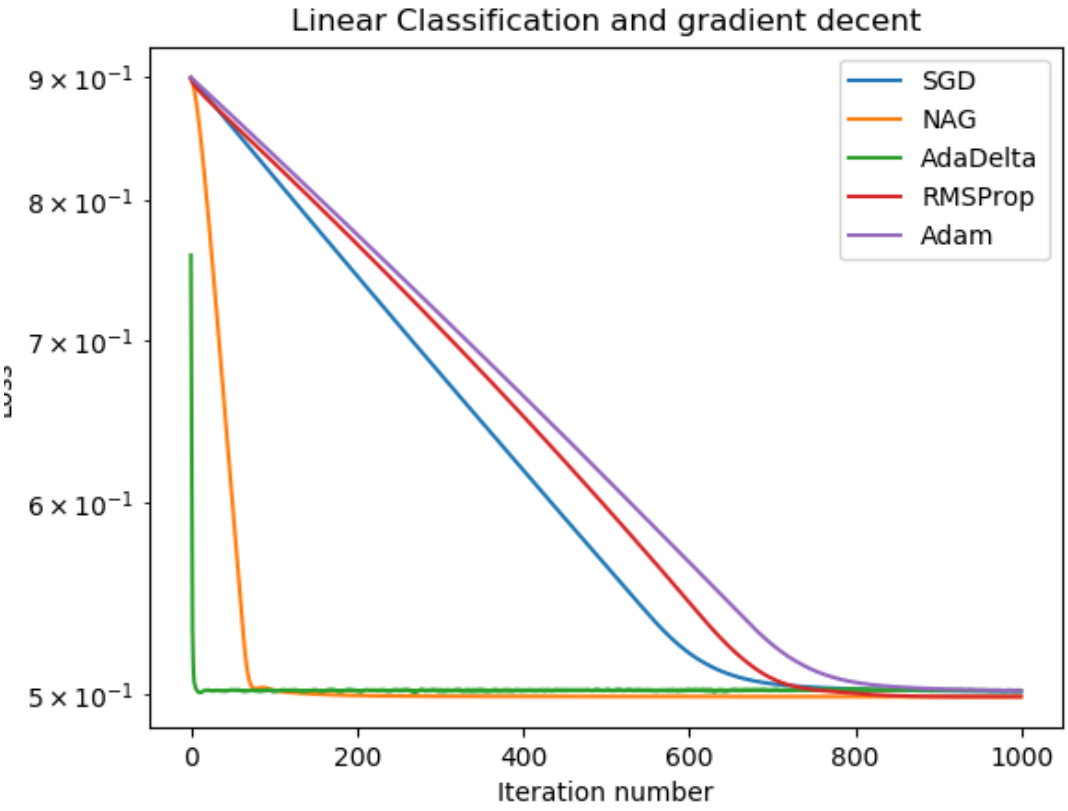
1) Logistic Regression

$\eta = 0.01, \mu = 0.9, \gamma = 0.9, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, n_{iter} = 1000$



2) Linear Classification

$\eta = 0.0005, C = 0.9, \mu = 0.9, \gamma = 0.9, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, n_{iter} = 1000$



11. Calculation performance and the method of computing gradient and loss:

During the experiment, I used to calculate gradient and loss using simple loop function. Then I found that it's too slow to conveniently do the experiment. Then I start to optimize the program's performance.

Firstly, I changed the type of dataset from sparse matrix to simple ndarray, and gained about 5 times speed-up.

In addition, I edited the method of calculating gradient and loss, by using lot of numpy function and combining loop calculation to matrix operation.

After all these optimization, the time of program's execution reduced about 95%, which really matters in practice.

This practice gives me a lesson, which I used to know but never cares. It warns me though our computer is quite good at calculating and the bottle neck of Machine Learning is I/O (I thought, but I don't think about it carefully, that in this experiment, thought the scale of dataset is huge, it still can be held in memory.), optimize algorithm is still of great importance.

12.Results analysis:

SGD:

Advantages: Very simple to implement, low computation, guaranteed to convergent to good globally optimal solution.

Disadvantages: α (learning rate) is a fixed hyper parameter, leading to direct causal relationship between α selection and good or bad result. A too small learning rate leads to low speed to convergent, whereas a too big learning rate leads to vibration which means can't convergent to optimal solution. What's more, if the problem is not convex problem, it can only convergent to Local optimum, and can't get over it.

NAG:

It is possible to accelerate the convergence of SGD by adaptively updating parameters according to the slope of the loss function in each learning process. In the next step, each parameter needs to be adaptively updated according to the importance of the parameters.

AdaDelta:

Advantages: Fully adaptive global learning rate, good performance of acceleration.

Disadvantages: In late learning progress, usually have shock in a small area.

RMSProp:

Advantages: Good performance of acceleration.

Disadvantages: In late learning progress, usually have shock in a small area. Need to set global learning rate.

Adam:

Combined with Momentum and Adaprop, it has good stability and at the same time it does not have to store all global gradients compared to Adagrad, so it is suitable for processing large-scale data.

Consequently, in my practice, AdaDelta has the fastest decent speed, but it is more likely to not get the best result, where as SGD is the lowest but nearly always decent to optimal result.

13.Similarities and differences between logistic regression and linear classification：

These two methods are both common classification algorithms. As for

the objective function, the difference is that, the logistic regression using logistical loss, SVM using hinge loss. The purpose of these two loss functions are both to increase the data points', that matter classification, weight and reduce the weight of the data points less relevant to the classification. SVM processing method is to consider only support vectors, which is the most relevant and the classification of a few points to learn the classifier. Logistic regression through nonlinear mapping, the weight of the points farther away from the classification plane is reduced and the weight of the data points most relevant to the classification is raised. The basic purpose of both method is the same. In addition, both methods can add different regular terms, such as l1, l2, etc. So, in many experiments, the results of the two algorithms are very close.

One the one hand, the logistic regression is relatively simple, easy to understand and implement, especially for large-scale linear classification, while the understanding and optimization of SVM are relatively complex. But on the other hand, the theoretical basis of SVM is more solid, it has a theoretical basis for minimizing the risk, although not commonly considered by people in general. There is a very important point. After the SVM transforms into a dual problem, the classification only needs to calculate the distance to a few support vectors. This is an obvious advantage when working with complex kernel functions and greatly simplifies models and calculations.

SVM more belongs to the non-parametric model, whereas logistic regression is a parameter model, essentially different. What logistic regression can do SVM can do too, but there may be problems that SVM can do but logistic regression cannot do

13. Summary:

This experiment is about the comparison of different decent method. After implementing such different optimization methods, I learnt a lot.

Firstly, I learnt the significance of optimize algorithm. In actual usage, the data set might much bigger than this, so the performance of program is of great importance. Though a better decent method might reduce the impact of bad practice of algorithm, but the performance between same algorithm can be large.

Secondly, difference decent method has difference feature, we shall select decent method carefully.

Thirdly, when using mini batch stochastic gradient decent, the bigger batch is, the smaller the shock is.

Fourthly, when plotting the graph of loss, using log scale with y dimension can have a better picture of decent progress. The line will be more smooth.