华南理工大学

South China University of Technology

The Experiment Report of

Machine Learning

College Software College

Subject Software Engineering

Members Feng Chen

Student ID 201530611159

E-mail 781909556@qq.com

Tutor Mingkui Tan

Date submitted 2017.12.15

1. **Topic:** Logistic Regression, Linear Classification and Stochastic Gradient Descent

2. Time: 2017/12/15

3. Reporter: Feng Chen

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.

6. Experimental steps:

- *Logistic Regression and Stochastic Gradient Descent*
  a. Load the training set and validation set.
  b. Initalize logistic regression model parameters, you can consider initalizing zeros, random numbers or normal distribution.
  c. Select the loss function and calculate its derivation, find more detail in PPT.
  d. Calculate gradient toward loss function from partial samples.
  e. Update model parameters using different optimized methods(NAG, RMSProp, AdaDelta and Adam).
  f. 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}.
  g. Repeate 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*

a. Load the training set and validation set.
b. Initalize SVM model parameters, you can consider initalizing zeros, random numbers or normal distribution.
c. Select the loss function and calculate its derivation, find more detail in PPT.
d. Calculate gradient toward loss function from **partial samples**.
e. Update model parameters using different optimized methods(NAG, RMSProp, AdaDelta and Adam).

h. 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}.
i. Repeate 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:

- *Logistic Regression and Stochastic Gradient Descent*

```
# sigmoid function
def function_g(x, w):
    return 1 / (1 + math.exp(-np.dot(x, w)[0]))
```

```
# compute each gradient and then average
def compute_gradient(x, y, w):
    gradient = 0.0
    for i in range(len(x)):
        gradient += (function_g(x[i], w) - y[i]) * x[i]
    return gradient / len(x)
```

```
# compute each loss and then average
def loss_function(x, y, w):
    loss = 0.0
    for i in range(len(x)):
        loss += y[i] * math.log(function_g(x[i], w)) + (1 - y[i]) * math.log(1 - function_g(x[i], w))
    return -loss / len(x)
```

```python
# NAG update parameters
def NAG(w, gradient, v, mu=0.9, eta=0.05):
    v_prev = v
    v = mu * v - eta * gradient
    w += (-mu * v_prev + (1 + mu) * v).reshape((123, 1))
    return w, v

# RMSProp update parameters
def RMSProp(w, gradient, cache, decay_rate=0.9, eps=1e-5, eta=0.01):
    cache = decay_rate * cache + (1 - decay_rate) * (gradient ** 2)
    w += (- eta * rmsprop_gradient / (np.sqrt(cache) + eps)).reshape((123, 1))
    return w, cache

# AdaDelta update parameters
def AdaDelta(w, gradient, cache, delta_t, r=0.95, eps=1e-6):
    cache = r * cache + (1 - r) * (gradient ** 2)
    delta_theta = - np.sqrt(delta_t + eps) / np.sqrt(cache + eps) * gradient
    w = w + delta_theta.reshape((123, 1))
    delta_t = r * delta_t + (1 - r) * (delta_theta ** 2)
    return w, cache, delta_t

# Adam update parameters
def Adam(w, gradient, m, i, t, beta1=0.9, beta2=0.999, eta=0.01, eps=1e-8):
    m = beta1 * m + (1 - beta1) * gradient
    mt = m / (1 - beta1 ** i)
    t = beta2 * t + (1 - beta2) * (gradient ** 2)
    vt = t / (1 - beta2 ** i)
    w += (-eta * mt / (np.sqrt(vt) + eps)).reshape((123, 1))
    return w, m, t
```

```python
# update parameters
for i in range(1, max_epoch):
    index = list(range(len(x_train)))
    random.shuffle(index)

    # NAG update parameters
    nag_gradient = compute_gradient(x_train[index][:batch_size], y_train[index][:batch_size], nag_w)
    nag_w, v = NAG(nag_w, nag_gradient, v)
    nag_loss.append(loss_function(x_valid, y_valid, nag_w))

    # RMSprop update parameters
    rmsprop_gradient = compute_gradient(x_train[index][:batch_size], y_train[index][:batch_size], rmsprop_w)
    rmsprop_w, cache = RMSProp(rmsprop_w, rmsprop_gradient, cache)
    rmsprop_loss.append(loss_function(x_valid, y_valid, rmsprop_w))

    # AdaDelta update parameters
    adadelta_gradient = compute_gradient(x_train[index][:batch_size], y_train[index][:batch_size], adadelta_w)
    adadelta_w, adadelta_cache, delta_t = AdaDelta(adadelta_w, adadelta_gradient,
                                                   adadelta_cache, delta_t)
    adadelta_loss.append(loss_function(x_valid, y_valid, adadelta_w))

    # Adam update parameters
    adam_gradient = compute_gradient(x_train[index][:batch_size], y_train[index][:batch_size], adam_w)
    adam_w, m, t = Adam(adam_w, adam_gradient, m, i, t)
    adam_loss.append(loss_function(x_valid, y_valid, adam_w))
```

- *Linear Classification and Stochastic Gradient Descent*

```python
# compute each hinge loss and the average
def loss_function(x, y, w):
    losses = (1 - y * np.dot(x, w))
    hinge_loss = 0
    for one_loss in losses:
        hinge_loss += max(0, one_loss)
    return hinge_loss / len(x)
```

```python
# compute each gradient and average
def compute_gradient(x, y, w):
    gradient = np.zeros((1, x.shape[1]))
    losses = (1 - y * np.dot(x, w))
    for i, loss in enumerate(losses):
        if loss <= 0:
            gradient += w.T
        else:
            gradient += w.T - y[i] * x[i]
    return gradient / len(x)
```

```python
# NAG update parameters
def NAG(w, gradient, v, mu=0.9, eta=0.001):
    v_prev = v
    v = mu * v - eta * gradient
    w += (-mu * v_prev + (1 + mu) * v).reshape((123, 1))
    return w, v

# RMSProp update parameters
def RMSProp(w, gradient, cache, decay_rate=0.9, eps=1e-8, eta=0.001):
    cache = decay_rate * cache + (1 - decay_rate) * (gradient ** 2)
    w += (- eta * rmsprop_gradient / (np.sqrt(cache) + eps)).reshape((123, 1))
    return w, cache

# AdaDelta update parameters
def AdaDelta(w, gradient, cache, delta_t, r=0.95, eps=1e-6):
    cache = r * cache + (1 - r) * (gradient ** 2)
    delta_theta = - np.sqrt(delta_t + eps) / np.sqrt(cache + eps) * gradient
    w = w + delta_theta.reshape((123, 1))
    delta_t = r * delta_t + (1 - r) * (delta_theta ** 2)
    return w, cache, delta_t

# Adam update parameters
def Adam(w, gradient, m, i, t, beta1=0.9, beta2=0.999, eta=0.001, eps=1e-8):
    m = beta1 * m + (1 - beta1) * gradient
    mt = m / (1 - beta1 ** i)
    t = beta2 * t + (1 - beta2) * (gradient ** 2)
    vt = t / (1 - beta2 ** i)
    w += (-eta * mt / (np.sqrt(vt) + eps)).reshape((123, 1))
    return w, m, t
```

```python
# update parameters
for i in range(1, max_epoch):
    index = list(range(len(x_train)))
    random.shuffle(index)

    # NAG update parameters
    nag_gradient = compute_gradient(x_train[index][:batch_size], y_train[index][:batch_size], nag_w)
    nag_w, v = NAG(nag_w, nag_gradient, v)
    nag_loss.append(loss_function(x_valid, y_valid, nag_w))

    # RMSprop update parameters
    rmsprop_gradient = compute_gradient(x_train[index][:batch_size], y_train[index][:batch_size], rmsprop_w)
    rmsprop_w, cache = RMSProp(rmsprop_w, rmsprop_gradient, cache)
    rmsprop_loss.append(loss_function(x_valid, y_valid, rmsprop_w))

    # AdaDelta update parameters
    adadelta_gradient = compute_gradient(x_train[index][:batch_size], y_train[index][:batch_size], adadelta_w)
    adadelta_w, adadelta_cache, delta_t = AdaDelta(adadelta_w, adadelta_gradient,
                                                   adadelta_cache, delta_t)
    adadelta_loss.append(loss_function(x_valid, y_valid, adadelta_w))

    # Adam update parameters
    adam_gradient = compute_gradient(x_train[index][:batch_size], y_train[index][:batch_size], adam_w)
    adam_w, m, t = Adam(adam_w, adam_gradient, m, i, t)
    adam_loss.append(loss_function(x_valid, y_valid, adam_w))
```

8. The initialization method of model parameters:

- *Logistic Regression and Stochastic Gradient Descent*

```python
max_epoch = 201
batch_size = 2000

# initial each w to zero
nag_w = np.zeros((x_train.shape[1], 1))
rmsprop_w = np.zeros((x_train.shape[1], 1))
adadelta_w = np.zeros((x_train.shape[1], 1))
adam_w = np.zeros((x_train.shape[1], 1))

# some parameters used in the update process are initialized to zero
v = np.zeros(x_train.shape[1])
cache = np.zeros(x_train.shape[1])
adadelta_cache = np.zeros(x_train.shape[1])
delta_t = np.zeros(x_train.shape[1])
m = np.zeros(x_train.shape[1])
t = np.zeros(x_train.shape[1])
```

for NAG: mu=0.9, eta=0.05
for RMSProp: decay_rate=0.9, eps=1e-5, eta=0.01
for AdaDelta: r=0.95, eps=1e-6
for Adam: beta1=0.9, beta2=0.999, eta=0.01, eps=1e-8

- *Linear Classification and Stochastic Gradient Descent*

```
max_epoch = 201
batch_size = 5000

# initial each w to zero
nag_w = np.zeros((x_train.shape[1], 1))
rmsprop_w = np.zeros((x_train.shape[1], 1))
adadelta_w = np.zeros((x_train.shape[1], 1))
adam_w = np.zeros((x_train.shape[1], 1))

# some parameters used in the update process are initialized to zero
v = np.zeros(x_train.shape[1])
cache = np.zeros(x_train.shape[1])
adadelta_cache = np.zeros(x_train.shape[1])
delta_t = np.zeros(x_train.shape[1])
m = np.zeros(x_train.shape[1])
t = np.zeros(x_train.shape[1])
```

for NAG: mu=0.9, eta=0.001
for RMSProp: decay_rate=0.9, eps=1e-8, eta=0.001
for AdaDelta: r=0.95, eps=1e-6
for Adam: beta1=0.9, beta2=0.999, eta=0.001, eps=1e-8

9. The selected loss function and its derivatives:

- *Logistic Regression and Stochastic Gradient Descent*
  Loss function:

$$J(\mathbf{w}) = -\frac{1}{n}\left[\sum_{i=1}^{n} y_i \log h_{\mathbf{w}}(\mathbf{x}_i) + (1-y_i)\log\left(1-h_{\mathbf{w}}(\mathbf{x}_i)\right)\right]$$

  Gradient:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{n}\sum_{i=1}^{n}\left(h_{\mathbf{w}}(\mathbf{x}_i) - y\right)\mathbf{x}_i$$

- *Linear Classification and Stochastic Gradient Descent*

  Loss function:

$$\text{Hinge loss} = \xi_i = \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))$$

  Gradient:

$$\min_{\mathbf{w},b} \; L(\mathbf{w},b) = \frac{\|\mathbf{w}\|^2}{2} + \frac{C}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))$$

  - So we have:

$$\nabla_{\mathbf{w}} L(\mathbf{w},b) = \mathbf{w} + \frac{C}{n}\sum_{i=1}^{n} g_{\mathbf{w}}(\mathbf{x}_i)$$

$$\nabla_b L(\mathbf{w},b) = \frac{C}{n}\sum_{i=1}^{n} g_b(\mathbf{x}_i)$$

10. Experimental results and curve:(Fill in this content for various methods of gradient descent respectively)

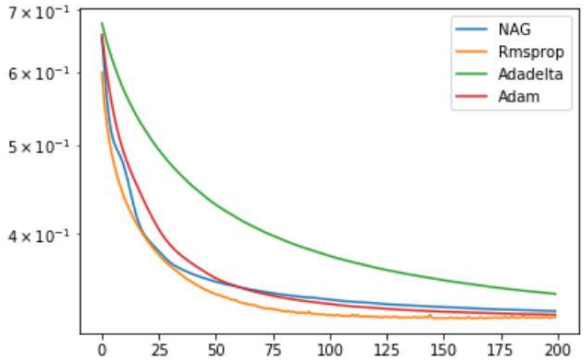
Hyper-parameter selection:

- *Logistic Regression and Stochastic Gradient Descent*
  for NAG: mu=0.9, eta=0.05
  for RMSProp: decay_rate=0.9, eps=1e-5, eta=0.01
  for AdaDelta: r=0.95, eps=1e-6
  for Adam: beta1=0.9, beta2=0.999, eta=0.01, eps=1e-8

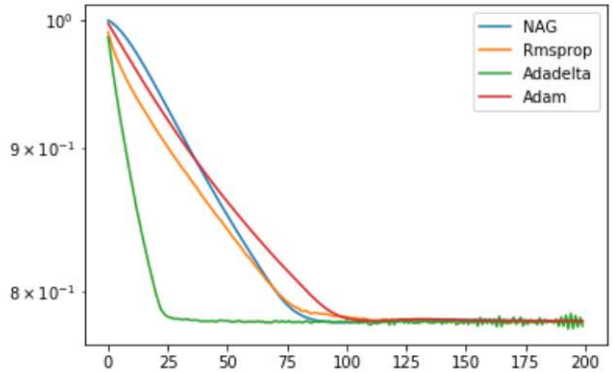
- *Linear Classification and Stochastic Gradient Descent*
  for NAG: mu=0.9, eta=0.001
  for RMSProp: decay_rate=0.9, eps=1e-8, eta=0.001
  for AdaDelta: r=0.95, eps=1e-6
  for Adam: beta1=0.9, beta2=0.999, eta=0.001, eps=1e-8

Loss curve

- *Logistic Regression and Stochastic Gradient Descent*



- *Linear Classification and Stochastic Gradient Descent*



11. Results analysis:

- For sparse data, try to use the learning rate can be adaptive optimization method, without manual adjustment, and the best default value
- SGD generally takes longer to train, but with good initialization and learning rate scheduling schemes, the results are more reliable
- If you are concerned about faster convergence and need to train deeper and more complex networks, we recommend that you use a learning rate adaptive optimization method.
- Adadelta, RMSprop, Adam are relatively similar algorithms that perform similarly under similar conditions.

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

- The two methods are common classification algorithms.
- objective function: logistic regression uses logistical loss, but svm uses hinge loss. The purpose of these two loss functions are to increase the classification of data Point 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.Logical regression through nonlinear mapping, greatly 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 is the same.In addition, both methods can add different regular The terms, such as l1, l2, etc. So in many experiments, the results of the two algorithms are very close.
- However, 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. However, the theoretical basis of SVM is more solid, The theoretical basis for minimizing the risk, although not commonly used by people in general, is not very relevant and it is important to note that once a SVM is converted into a dual problem, the classification needs only to calculate the distance to a few support vectors, The advantages of function calculation are obvious, which can greatly simplify the model and calculation
- svm more belongs to the non-parametric model, and logistic regression is a parametric model, the essence is different, the difference can refer to the difference between parametric model and non-parametric model just fine.
- Logic can do svm can do, but there may be problems in accuracy, svm can do some logic can not do

13. Summary:

Through this experiment, I know the similarities and differences between logistic regression and linear classification, understand the process and the advantages and disadvantages of various optimization algorithms