華南理工大學

South China University of Technology

The Experiment Report of
Machine Learning

College **Software College**

Subject **Software Engineering**

Members

Student ID 201530661741

E-mail yimingzhao0@qq.com

Tutor Mingkui Tan

Date submitted 2017. .

1. Topic: Logistic regression, logistic classification and gradient decent method

2. Time: 2017.12.2 PM 2:00-5:00

3. Reporter: Yiming Zhao

4. Purposes:

- **Understand the differences between the gradient decent and stochastic gradient decent method.**
- **Understand the connection between the logistic regression and linear regression.**
- **Practicing the SVM on the bigger data set.**

5. Data sets and data analysis:

Using a9a data set from LIBSVM Data, including 32561 / 16281(testing) samples and 123 features in each sample.

6. Experimental steps:

- **Read the data set using sklearn's function**
- **Randomly initializing the parameters.**
- **Selecting a proper loss function and using NAG, Adam, Adadelta, RMSProp to optimize the loss function.**

7. Code:

```
from sklearn import datasets as ds
import numpy as np
```

```python
from numpy import random
import matplotlib.pyplot as plt


def sigmoid(input_):
    return 1 / (1 + np.exp(-input_))


def train(x_train, y_train, x_test, y_test,
method, iters, test_errors):
    max_iterations = 100
    theta = random.rand(num_features + 1)
    num_test_samples, num_test_features =
x_test.shape

    if method == 'sgd':
        lr = 0.01

        for i in range(max_iterations):
            output = sigmoid(np.dot(x_train[i],
theta))
            error = output - y_train[i]
```

```
            theta = theta - lr * np.dot(x_train[i],
error)

            predict_error = 0
            for j in range(num_test_samples):
                predict_output =
sigmoid(np.dot(x_test[j], theta))
                predict_error -= y_test[j] *
np.log(predict_output) + (1 - y_test[j]) *
np.log(1 - predict_output)
            print(str(i) + '\t' +
str(predict_error / num_test_samples))

            iters.append(i)
            test_errors.append(predict_error /
num_test_samples)

    if method == 'nag':
        lr = 0.01
        miu = 0.9
        momentum = np.zeros(num_features + 1)
```

```
    for i in range(max_iterations):
        output = sigmoid(np.dot(x_train[i],
theta - lr * miu * momentum))
        error = output - y_train[i]
        grad = np.dot(x_train[i], error)
        momentum = momentum * lr + grad
        theta = theta - lr * momentum

        predict_error = 0
        for j in range(num_test_samples):
            predict_output =
sigmoid(np.dot(x_test[j], theta))
            predict_error -= y_test[j] *
np.log(predict_output) + (1 - y_test[j]) *
np.log(1 - predict_output)
        print(str(i) + '\t' +
str(predict_error / num_test_samples))

        iters.append(i)
        test_errors.append(predict_error /
num_test_samples)
```

```
    if method == 'rmsprop':
        lr = 0.1
        expectation = 1
        rho = 0.95
        delta = 10e-7

        for i in range(max_iterations):
            output = sigmoid(np.dot(x_train[i],
theta))
            error = output - y_train[i]
            grad = np.dot(x_train[i], error)
            norm = grad * grad
            expectation = rho * expectation + (1 -
rho) * norm
            theta = theta - lr * grad /
(np.sqrt(expectation) + delta)

            predict_error = 0
            for j in range(num_test_samples):
                predict_output =
sigmoid(np.dot(x_test[j], theta))
                predict_error -= y_test[j] *
```

```
np.log(predict_output) + (1 - y_test[j]) *
np.log(1 - predict_output)
            print(str(i) + '\t' +
str(predict_error / num_test_samples))
            iters.append(i)
            test_errors.append(predict_error /
num_test_samples)

    if method == 'adam':
        delta = 10e-8
        rho1 = 0.9
        rho2 = 0.999
        lr = 0.1
        s = 0
        r = 0

        for i in range(max_iterations):
            output = sigmoid(np.dot(x_train[i],
theta))
            error = output - y_train[i]
            grad = np.dot(x_train[i], error)
```

```
            s = rho1 * s + (1 - rho1) * grad
            r = rho2 * r + (1 - rho2) * grad * grad
            s_hat = s / (1 - rho1)
            r_hat = r / (1 - rho2)
            delta_theta = (-lr * s_hat) /
(np.sqrt(r_hat) + delta)
            theta = theta + delta_theta

            predict_error = 0
            for j in range(num_test_samples):
                predict_output =
sigmoid(np.dot(x_test[j], theta))
                predict_error -= y_test[j] *
np.log(predict_output) + (1 - y_test[j]) *
np.log(1 - predict_output)
            print(str(i) + '\t' +
str(predict_error / num_test_samples))
            iters.append(i)
            test_errors.append(predict_error /
num_test_samples)

    if method == 'adadelta':
```

```
        r = 0
        e = 0
        miu = 0.9
        delta = 10e-7
        lr = 10

        for i in range(max_iterations):
            output = sigmoid(np.dot(x_train[i],
theta))
            error = output - y_train[i]
            grad = np.dot(x_train[i], error)

            r = miu * r + (1 - miu) * grad * grad
            delta_theta = (-lr * grad * np.sqrt(e
+ delta)) / (np.sqrt(r + delta))
            theta = theta + delta_theta
            e = miu * e + (1 - miu) * e * e



            predict_error = 0
            for j in range(num_test_samples):
                predict_output =
```

```
sigmoid(np.dot(x_test[j], theta))
             predict_error -= y_test[j] *
np.log(predict_output) + (1 - y_test[j]) *
np.log(1 - predict_output)
          print(str(i) + '\t' +
str(predict_error / num_test_samples))
          iters.append(i)
          test_errors.append(predict_error /
num_test_samples)

if __name__ == '__main__':
   x_train, y_train =
ds.load_svmlight_file('./data/a9a')
   x_test, y_test =
ds.load_svmlight_file('./data/a9a.t')

   num_samples, num_features = x_train.shape
   num_test_samples, num_test_features =
x_test.shape

   x_train = x_train.toarray()
   temp = np.ones(shape=[32561, 1],
```

```
dtype=np.float32)
    x_train = np.concatenate([x_train, temp],
axis=1)
    x_test = x_test.toarray()
    temp = np.zeros(shape=[16281, 1],
dtype=np.float32)
    temp1 = np.ones(shape=[16281, 1],
dtype=np.float32)
    x_test = np.concatenate([x_test, temp, temp1],
axis=1)

    for i in range(0, len(y_train)):
        if y_train[i] == -1:
            y_train[i] = 0
    for i in range(0, len(y_test)):
        if y_test[i] == -1:
            y_test[i] = 0


    methods = ['sgd', 'nag', 'rmsprop',
'adadelta', 'adam']
    for method in methods:
```

```
        iters = []
        test_errors = []
        train(x_train, y_train, x_test, y_test,
method, iters, test_errors)
        plt.plot(iters, test_errors,
label=method)

plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
from sklearn import datasets as ds
import numpy as np
from numpy import random
import matplotlib.pyplot as plt
import os

C = 1
feature_size = 123
bias = np.zeros(shape=[feature_size + 1, 1])
bias[len(bias)-1][0] = 1.
```

```
def compute_loss(x, y, w):
    # x.shape = [batch_size, feature_size + 1],
y.shape = [batch_size, 1], w.shape =
[feature_size + 1, 1]
    pred = np.matmul(x, w)
    hinge_loss = np.maximum(1 - y * pred, 0)
    loss = np.mean(hinge_loss ** 2) + C * np.sum((w
- bias) ** 2)
    return loss


def compute_gradient(x, y, w):
    # x.shape = [batch_size, feature_size + 1],
y.shape = [batch_size, 1], w.shape =
[feature_size + 1, 1]
    pred = np.matmul(x, w)
    hinge_loss = np.maximum(1 - y * pred, 0)
    hinge_loss_gradient =
-np.matmul(x.transpose(), hinge_loss * y) / len(y)
    norm_gradient = 2 * C * (w - bias)
```

```
    gradient = hinge_loss_gradient +
norm_gradient
    return gradient  # gradient.shape =
[feature_size + 1, 1], the same as w


global_list = {'NAG_momentum': 0,
'RMS_expectation': 1, 'ADAM_s': 0, 'ADAM_r': 0,
               'ADADELDA_r': 0, 'ADADELDA_e': 0}


def optimizer(method, parameter_list, x, y, w):
    global global_list
    if method == 'SGD':
        # sgd_para_list = {'learning_rate':0.01}
        lr = parameter_list['learning_rate']
        w -= lr * compute_gradient(x, y, w)
    if method == 'NAG':
        # nag_para_list = {'miu':0.9,
'learning_rate':0.01}
        lr = parameter_list['learning_rate']
        miu = parameter_list['miu']
```

```
        momentum = global_list['NAG_momentum']
        gradient = compute_gradient(x, y, w - 
momentum * lr * miu)
        momentum = momentum * lr + gradient
        w -= lr * momentum
        global_list['NAG_momentum'] = momentum
    if method == 'RMSProp':
        # rmsprop_para_list = {'delta':10e-7, 
'rho':0.95, 'learning_rate':0.1}
        lr = parameter_list['learning_rate']
        expectation = 
global_list['RMS_expectation']
        rho = parameter_list['rho']
        delta = parameter_list['delta']
        gradient = compute_gradient(x, y, w)
        expectation = rho * expectation + (1 - rho) 
* (gradient ** 2)
        global_list['RMS_expectation'] = 
expectation
        w -= lr * gradient / (np.sqrt(expectation) 
+ delta)
    if method == 'ADAM':
```

```
        # adam_para_list = {'delta':10e-8,
'rho1':0.9, 'rho2':0.999, 'learning_rate':0.1}
        delta = parameter_list['delta']
        rho1 = parameter_list['rho1']
        rho2 = parameter_list['rho2']
        lr = parameter_list['learning_rate']
        s = global_list['ADAM_s']
        r = global_list['ADAM_r']
        gradient = compute_gradient(x, y, w)
        s = rho1 * s + (1 - rho1) * gradient
        r = rho2 * r + (1 - rho2) * (gradient **
2)
        s_hat = s / (1 - rho1)
        r_hat = r / (1 - rho2)
        w -= (lr * s_hat) / (np.sqrt(r_hat) + delta)
        global_list['ADAM_s'] = s
        global_list['ADAM_r'] = r
    if method == 'Adadelta':
        # adadelta_para_list = {'delta':10e-7,
'miu':0.9, 'learning_rate':0.1}
        r = global_list['ADADELDA_r']
        e = global_list['ADADELDA_e']
```

```
        miu = parameter_list['miu']
        delta = parameter_list['delta']
        lr = parameter_list['learning_rate']
        grad = compute_gradient(x, y, w)
        r = miu * r + (1 - miu) * (grad ** 2)
        w -= (lr * grad * np.sqrt(e + delta)) /
(np.sqrt(r + delta))
        e = miu * e + (1 - miu) * (w ** 2)
        global_list['ADADELDA_r'] = r
        global_list['ADADELDA_e'] = e


if __name__ == '__main__':
    x_train, y_train =
ds.load_svmlight_file('./data/a9a')
    x_test, y_test =
ds.load_svmlight_file('./data/a9a.t')

    num_samples, num_features = x_train.shape
    num_test_samples, num_test_features =
x_test.shape
```

```
    x_train = x_train.toarray()
    temp = np.ones(shape=[num_samples, 1],
dtype=np.float32)
    x_train = np.concatenate([x_train, temp],
axis=1)
    x_test = x_test.toarray()
    temp = np.zeros(shape=[num_test_samples, 1],
dtype=np.float32)
    temp1 = np.ones(shape=[num_test_samples, 1],
dtype=np.float32)
    x_test = np.concatenate([x_test, temp, temp1],
axis=1)
    y_train = y_train.reshape([len(y_train), 1])
    y_test = y_test.reshape([len(y_test), 1])


    def shuffle_train():
        global x_train, y_train
        rng_state = np.random.get_state()
        np.random.shuffle(x_train)
        np.random.set_state(rng_state)
        np.random.shuffle(y_train)
```

```
    batch_size = 1024
    data_size = num_samples

    def feed_data(batch_count):
        if (1 + batch_count) * batch_size <=
data_size:
            feed_dict = {'x': x_train[batch_count
* batch_size:(batch_count + 1) * batch_size],
                         'y': y_train[batch_count *
batch_size:(batch_count + 1) * batch_size]}
        else:
            feed_dict = {'x': x_train[batch_count
* batch_size:data_size],
                         'y': y_train[batch_count *
batch_size:data_size]}
        return feed_dict

    methods = ['SGD', 'NAG', 'RMSProp',
'Adadelta', 'ADAM']
    sgd_para_list = {'learning_rate': 0.01}
    nag_para_list={'miu':0.9, 'learning_rate':
```

```
0.01}
    rmsprop_para_list = {'delta': 10e-7, 'rho':
0.95, 'learning_rate': 0.1}
    adam_para_list = {'delta': 10e-8, 'rho1': 0.9,
'rho2': 0.999, 'learning_rate': 0.1}
    adadelta_para_list = {'delta': 10e-7, 'miu':
0.9, 'learning_rate': 0.1}
    para_list = {'SGD': sgd_para_list, 'NAG':
nag_para_list, 'RMSProp': rmsprop_para_list,
                 'Adadelta': adadelta_para_list,
'ADAM': adam_para_list}
    for method in methods:
        iters = []
        test_errors = []
        w = np.random.rand(feature_size+1, 1)
        parameter_list = para_list[method]
        count = 0
        for i in range(0, 3):
            shuffle_train()
            for batch_count in range(0,
int(data_size / batch_size) + 1):
                feed_dict = feed_data(batch_count)
```

```
                    iters.append(count)
                    count += 1
                    optimizer(method=method,
parameter_list=parameter_list,
x=feed_dict['x'], y=feed_dict['y'], w=w)

test_errors.append(compute_loss(x_test, y_test,
w))
          plt.plot(iters, test_errors,
label=method)

plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

(Fill in the contents of 8-11 respectively for logistic regression and linear classification)

8. The initialization method of model parameters:

Randomly initializing.

9. The selected loss function and its derivatives:

$$\boldsymbol{w} = (\boldsymbol{w}; \mathbf{1})$$

$$p_i = sigmoid(w^T x_i)$$

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p_i) + (1-y_i)log(1-p_i)$$

$$\frac{\partial Loss}{\partial w} = -\frac{1}{N}\sum_{i=1}^{N} x_i(p_i - y_i)$$

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

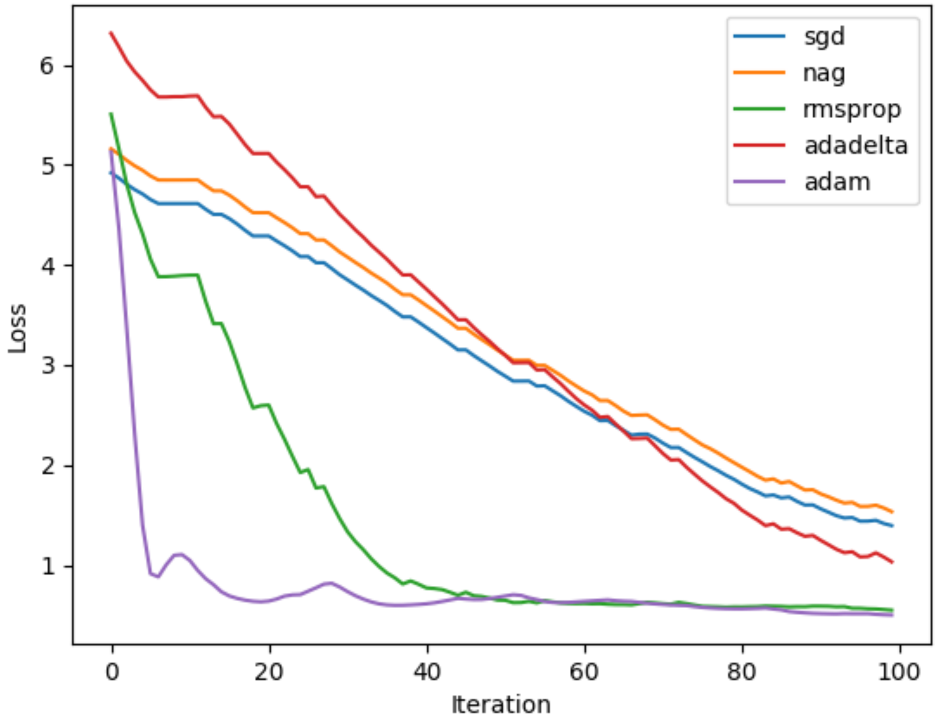
Hyper-parameter selection:

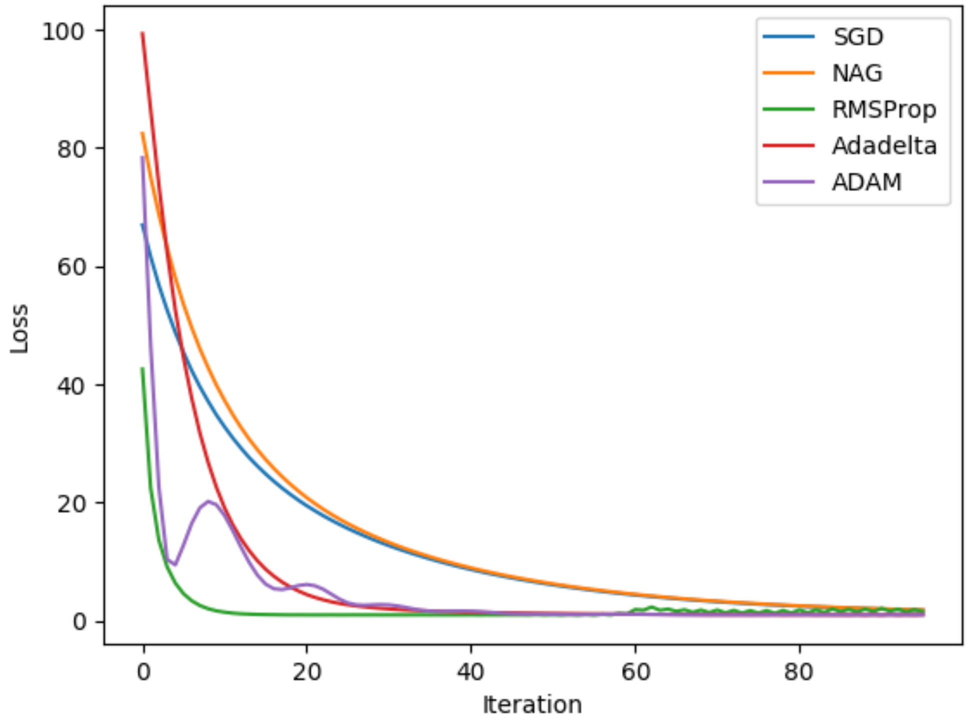
```
SGD:
Learning rate = 0.01

NAG:
Learning rate = 0.01
Miu = 0.9
Momentum_ini = [0,0 ... ,0]^T

RMSProp:
Learning rate = 0.1
Expectation = 1
   ρ = 0.95
delta=10e − 7
```

```
Adam:
delta = 10e-8
rho1 = 0.9
rho2 = 0.999
lr = 0.1
s = 0
r = 0

Adadelta
r = 0
       e = 0
       miu = 0.9
       delta = 10e-7
       lr = 10
```

Predicted Results (Best Results):

Loss curve:





11. Results analysis:

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

13. Summary: