South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Zhaoyu Zhao

Supervisor:
Qingyao Wu

Student ID：
201530613900

Grade:
Undergraduate

December 9, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

Abstract—We use Logistic Regression and Linear Classification with four SGD methods to solve classification in this experiment for learning. We gain some perceptual knowledge about the comparision between Logistic Regression and Linear Classification as well as four SGD methods.

I. Introduction

In this experiment, we use SVM and logistic regression for solving classification with different optimized methods(NAG，RMSProp，AdaDelta and Adam).We would like to compare SVM with logistic regression for solving classification.We use Stochastic Gradient Descent to speed up the process of convergence because we use a huge dataset.By using four Stochastic Gradient Descent methods, it reduces some errors in a way. Besides, we can also compare these four optimized methods and draw a conclusion about it.

In a word, there are three purpose in this experiment. First, we hope to compare and understand the difference between gradient descent and stochastic gradient descent. Then, we'll compare and understand the differences and relationships between Logistic regression and linear classification. At last, we can further understand the principles of SVM and practice on larger data.

II. Methods and Theory

Logistic regression maximizes the likelihood that each sample is classified correctly. The proof is as followed.

$$\begin{aligned}
& \max && \prod_{n=1}^{N} P(y_n \mid \mathbf{x}_n) \\
\Leftrightarrow\ & \max && \ln\left(\prod_{n=1}^{N} P(y_n \mid \mathbf{x}_n)\right) \\
\equiv\ & \max && \sum_{n=1}^{N} \ln P(y_n \mid \mathbf{x}_n) \\
\Leftrightarrow\ & \min && -\frac{1}{N}\sum_{n=1}^{N} \ln P(y_n \mid \mathbf{x}_n) \\
\equiv\ & \min && \frac{1}{N}\sum_{n=1}^{N} \ln \frac{1}{P(y_n \mid \mathbf{x}_n)} \\
\equiv\ & \min && \frac{1}{N}\sum_{n=1}^{N} \ln \frac{1}{\theta(y_n \cdot \mathbf{w}^{\mathrm{T}}\mathbf{x}_n)} \\
\equiv\ & \min && \frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n \cdot \mathbf{w}^{\mathrm{T}}\mathbf{x}_n})
\end{aligned}$$

So $\frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n \cdot \mathbf{w}^{\mathrm{T}}\mathbf{x}_n})$ is our loss function. We can calculate the gradient of it. That is

$$\frac{\partial \frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n * w^T * x_n})}{\partial w} = \frac{1}{N}\sum_{n=1}^{N} \frac{-y_n * e^{-y_n * w^T * x_n}}{1 + e^{-y_n * w^T * x_n}} x_n$$

After getting the gradient, we can perform SGD to train theta, which approximately minimizes the loss funciton. SGD methods will be involved after the proof of SVM.

In this experiment, we consider about L1-SVM. It requires the solution of the following unconstrained optimization problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i), \qquad (1)$$

where ξ(w; xi , yi) is a loss function, and C > 0 is a penalty parameter. The loss function is $\max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0)$.We can calculate the gradient.

$$\frac{\partial f(\mathbf{w}, b)}{\mathbf{w}} = \mathbf{w} + C\sum_{i=1}^{N} g_{\mathbf{w}}(\mathbf{x}_i)$$

$$\frac{\partial f(\mathbf{w}, b)}{b} = C\sum_{i=1}^{N} g_{b}(\mathbf{x}_i)$$

$$g_{\mathbf{w}}(\mathbf{x}_i) = \begin{cases} -y_i\mathbf{x}_i & 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b) >= 0 \\ 0 & 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b) < 0 \end{cases}$$

$$g_{b}(\mathbf{x}_i) = \begin{cases} -y_i & 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b) >= 0 \\ 0 & 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b) < 0 \end{cases}$$

Similarily, we can perform SGD to get an ideal theta.

There are four SGD methods involved in this experiment. (NAG，RMSProp，AdaDelta and Adam).

NAG

NAG(Nesterov accelerated gradient)'s main idea is to predict the next step's gradient rather than use the merely theta.

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1} - \gamma\mathbf{v}_{t-1})$$
$$\mathbf{v}_t \leftarrow \gamma\mathbf{v}_{t-1} + \eta\mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \mathbf{v}_t$$

RMSprop

RMSprop is a method to solve the problem that learning speed converges to zero in AdaGrad method. It adds an exponential decal on the accumulated information.

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$
$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t$$

AdaDelta

AdaDelta is another way to solve AdaGrad's problem. It uses $\sqrt{\Delta_{t-1} + \epsilon}$ to estimate the learning speed.

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$
$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$
$$\Delta\boldsymbol{\theta}_t \leftarrow -\frac{\sqrt{\Delta_{t-1} + \epsilon}}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} + \Delta\boldsymbol{\theta}_t$$
$$\Delta_t \leftarrow \gamma \Delta_{t-1} + (1-\gamma)\Delta\boldsymbol{\theta}_t \odot \Delta\boldsymbol{\theta}_t$$

Adam

Adam (adaptive estimates of lower-order moments) can correct initialization bias and be adapted to mean and variance.

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$
$$\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$$
$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$
$$\alpha \leftarrow \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t}$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\mathbf{m}_t}{\sqrt{G_t + \epsilon}}$$

III. Experiment

A dataset

Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. We use a9a as training set and a9a.t as validation set.

B implementation

Logistic Regression and Stochastic Gradient Descent

1. Load the training set and validation set.

```
#读入数据，并将数据转化为array
X_train, y_train = load_svmlight_file("a9a")
X_test, y_test = load_svmlight_file("a9a.t")
X_train = X_train.toarray()
X_test = X_test.toarray()

#测试集上第123个属性全为0，这里补上
temp = np.zeros(shape=[len(y_test),1])
X_test = np.concatenate([X_test,temp], axis=1)
```

2. Initalize logistic regression model parameters, we use initalizing zeros.
3. Select the loss function and calculate its derivation.

```
#给定矩阵X, 一维向量y和w,
#根据Loss=1/N*sigma(ln(1+exp(-y[n]*w.T*X[n])))(1<=n<=N)计算Loss
def get_loss(X, y, w):
    loss = 0
    for i in range(len(y)):
        loss += math.log(1 + math.exp(- y[i] * np.dot(w.T, X[i])))
    return loss / len(y)
```

4. Calculate gradient toward loss function from **partial samples**.

```
#随机取batch_size组数
batch = np.random.choice(len(y_train), batch_size)
X_batch, y_batch = X_train[batch], y_train[batch]
```

```
#给定矩阵X, 一维向量y和w,计算Loss在w上的梯度
#令exp_tem = exp(-y[n]*w.T*X[n])
#则可根据Loss=1/N*sigma((-y[n]*exp_tem/(1+exp_tem))*X[n])(1<=n<=N)计算梯度
def get_gradient(X, y, w):
    gradient = np.zeros(shape=[X.shape[1]])
    for i in range(len(y)):
        exp_tem = math.exp(- y[i] * np.dot(w.T, X[i]))
        gradient += (- y[i] * exp_tem / (1 + exp_tem)) * X[i]
    gradient /= len(y)
    return gradient
```

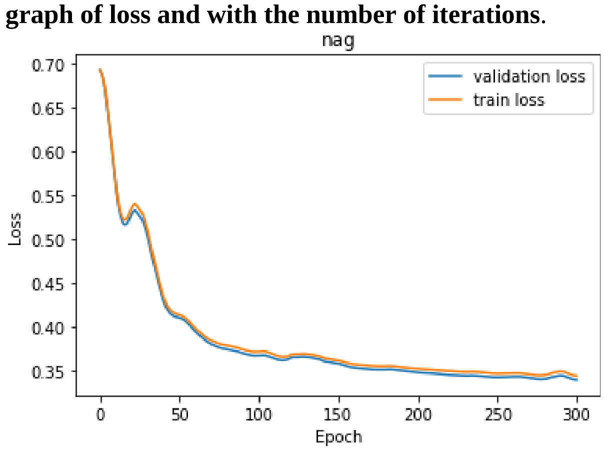
5. **Update model parameters using different optimized methods(NAG，RMSProp，AdaDelta and Adam).**

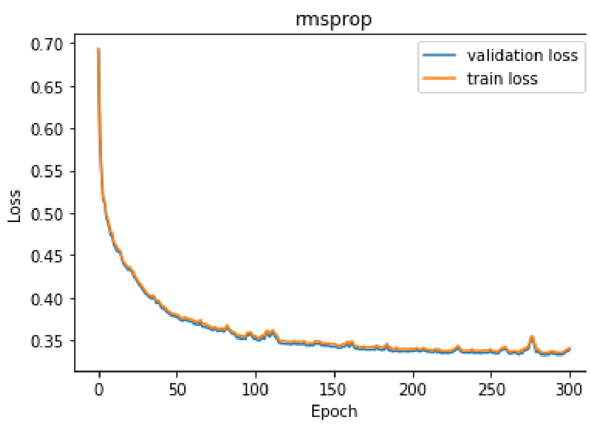
```
for t in range(1, max_epoch + 1): #从1开始，避免adam出现除零问题
    #随机取batch_size组数
    batch = np.random.choice(len(y_train), batch_size)
    X_batch, y_batch = X_train[batch], y_train[batch]

    #以下分别是4种SGD算法的实现，一开始均采用get_gradient来
    #求得对应的梯度，然后按各方法进行相应操作
    #（参考https://blog.slinuxer.com/2016/09/sgd-comparison）

    # nag
    if train_way == "nag":
        gradient = get_gradient(X_batch, y_batch, theta - gama * velocity)
        velocity = gama * velocity + lr * gradient
        theta = theta - velocity

    # rmsprop
    if train_way == "rmsprop":
        gradient = get_gradient(X_batch, y_batch, theta)
        Gradient = gama * Gradient + (1 - gama) * (gradient * gradient)
        theta = theta - (lr / (np.sqrt(Gradient + epsilon))) * gradient

    # adadelta
    if train_way == "adadelta":
        gradient = get_gradient(X_batch, y_batch, theta)
        Gradient = gama * Gradient + (1 - gama) * (gradient * gradient)
        delta_theta = -(np.sqrt(delta_t + epsilon) / np.sqrt(Gradient + epsilon) * gradient)
        theta = theta + delta_theta
        delta_t = gama * delta_t + (1 - gama) * (delta_theta * delta_theta)

    # adam
    if train_way == "adam":
        gradient = get_gradient(X_batch, y_batch, theta)
        moments = beta * moments + (1 - beta) * gradient
        Gradient = gama * Gradient + (1 - gama) * (gradient * gradient)
        alpha = lr * math.sqrt(1 - math.pow(gama, t)) / (1 - math.pow(beta, t))
        theta = theta - alpha * moments / np.sqrt(Gradient + epsilon)
```

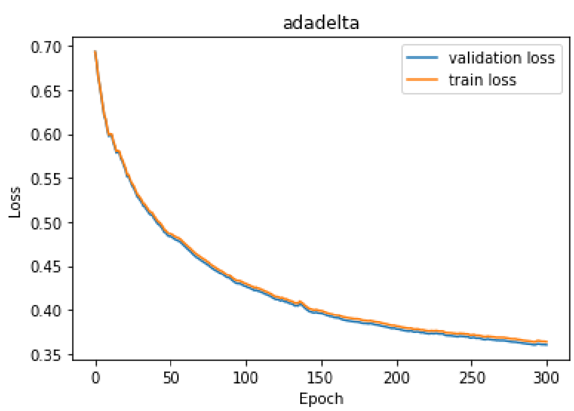
6. Select the appropriate threshold, mark the sample whose predict scores **greater than the threshold as positive, on the contrary as negative**. Predict under validation set and get the different optimized method loss

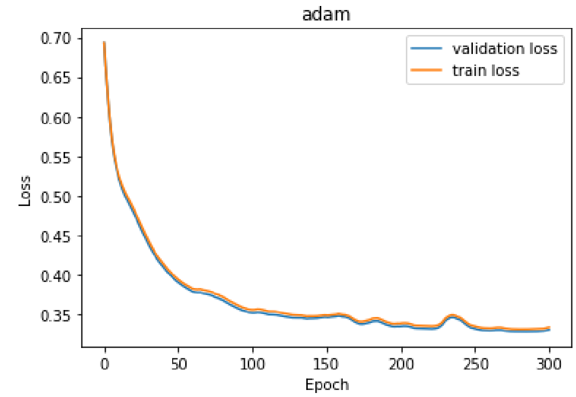
```python
#输出分类的正确率，分类的阈值为0
def print_correct(X, y, w):
    num_correct = 0
    for i in range(X.shape[0]):
        if y[i] * (np.dot(w.T, X[i])) >= 0:
            num_correct += 1
    print("total:", len(y))
    print("correct:", num_correct)
    print("accuracy:", num_correct/len(y))
```

7. Repeate step 4 to 6 for several times, and **drawing graph of loss and with the number of iterations**.









	lr	epochs	Loss_train	Loss_test	Acuracy_train	Acuracy_test
Nag	0.01	300	0.3440	0.3395	0.8413	0.8430
RmsProp	0.01	300	0.3405	0.3385	0.8413	0.8425
AdaDelta	0.01	300	0.3638	0.3603	0.8312	0.8322
Adam	0.01	300	0.3337	0.3303	0.8441	0.8453

Linear Classification and Stochastic Gradient Descent

1. Load the training set and validation set.
2. Initalize SVM model parameters, we use initalizing zeros.
   The basic codes are the same as we do in Logisitic Regression. But we have a trick here. That is, merging b into w. The code is as followed.

```python
#以下处理读入数据，每组X后增加一列1，
#用来将b合并到w中，这样方便处理

#训练集增加一列，X=(X:1)
temp = np.ones(shape=[len(y_train),1])
X_train = np.concatenate([X_train,temp], axis=1)

#测试集增加第123个属性（全为零），同时增加一列1
temp = np.zeros(shape=[len(y_test),1])
X_test = np.concatenate([X_test,temp], axis=1)
temp = np.ones(shape=[len(y_test),1])
X_test = np.concatenate([X_test,temp], axis=1)
```

3. Select the loss function and calculate its derivation, find more detail in PPT.

```python
#给定矩阵X，一维向量y,w,求Loss,这里C自适应，除了组数
#（根据公式Loss=0.5*w*w+C*sigma(max(0,1-y[i]*(w`T*x[i]))))
def get_loss(X, y, w):
    loss = 0 #初始化Loss
    #遍历每组数据
    for i in range(X.shape[0]):
        #如果大于0，累加loss
        if(hinge_judge(X[i], y[i], w)):
            loss += 1 - y[i] * (np.dot(w.T, X[i]))
    loss *= C #乘以系数C
    loss /= len(y) #除组数
    loss += np.dot(w, w) / 2 #加上求和外的0.5*w*w
    return loss
```

4. Calculate gradient toward loss function from **partial**

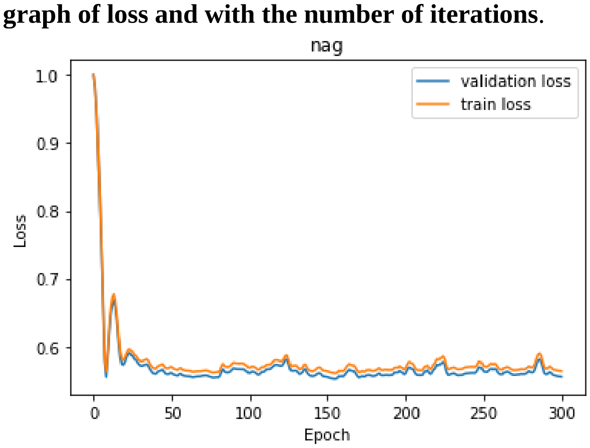
samples.
The basic codes are the same as we do in Logisitic Regression, so we just give the gradient code.

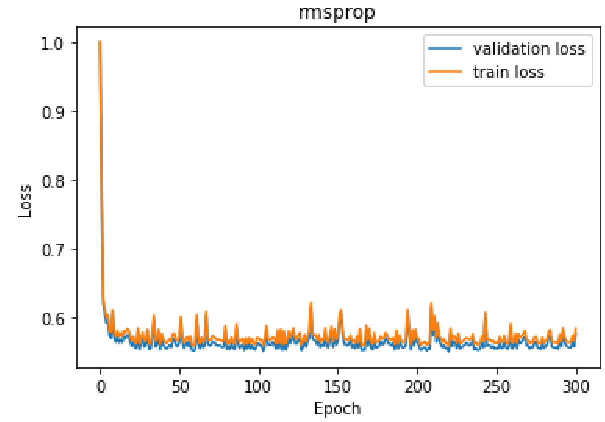
```
#求Loss时分类的判断条件
def hinge_judge(x, y, w):
    return 1 - y * (np.dot(w.T, x)) >= 0

#给定矩阵X, 一维向量y,w,求梯度
#gradient=w+C*sigma((1-y[i]*(w.T*X[i])>=0 ? -y[i]*X[i] : 0)(1<=
def get_gradient(X, y, w):
    gradient = np.zeros(shape=[X.shape[1]])
    #遍历每组数据，根据hinge_judge判断该组数据是否对梯度有贡献
    for j in range(len(y)):
        if hinge_judge(X[j], y[j], w):
            gradient -= C * y[j] * X[j]

    #对每组数据求和后得到的梯度求均值
    gradient /= len(y)

    #w的梯度需要加上w_train
    gradient += w

    return gradient
```

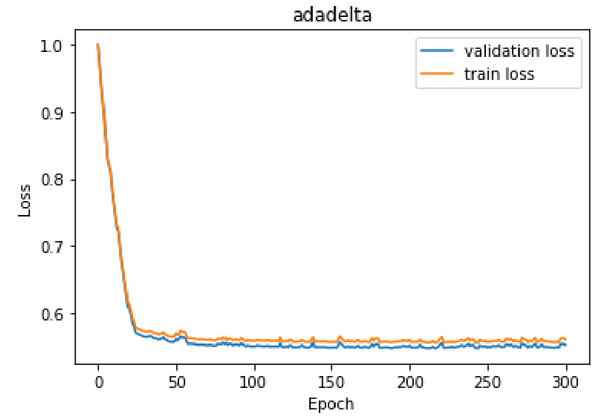
5. **Update model parameters using different optimized methods(NAG，RMSProp，AdaDelta and Adam).**
   The basic codes are the same as we do in Logisitic Regression, we just add some initialization.

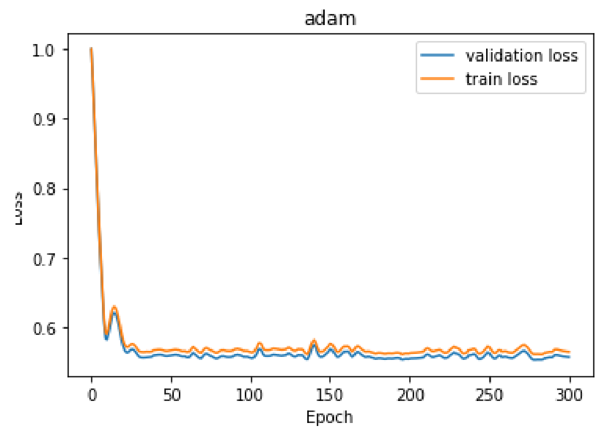
```
#设置SGD的超参数
batch_size = 16
gama = 0.95
beta = 0.9
epsilon = 1e-6
#初始化部分SGD方法需要用到的变量
velocity = np.zeros(shape=[X_train.shape[1]])
moments = np.zeros(shape=[X_train.shape[1]])
Gradient = np.zeros(shape=[X_train.shape[1]])
delta_t = np.zeros(shape=[X_train.shape[1]])
```

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.
7. Repeate step 4 to 6 for several times, and **drawing graph of loss and with the number of iterations**.









	lr	epochs	Loss_train	Loss_test	Acuracy_train	Acuracy_test
Nag	0.01	300	0.5640	0.5554	0.7592	0.7638
RmsProp	0.01	300	0.5844	0.5771	0.7592	0.7638
AdaDelta	0.01	300	0.5607	0.5523	0.7592	0.7638
Adam	0.01	300	0.5645	0.5569	0.7592	0.7638

IV. CONCLUSION

We are regret that the running machine is not so well that we have to wait for a long time to get the result for each experiment. Lack of enough times of experiment, plus the randomness of SGD, the conclusion we draw may not be compelling enough. But we can gain some perceptual knowledge about it, since the experiment is just for learning.

From the comparision between SVM and Logisitic Regression, we find that SVM converges faster, but finally its accuracy is lower than Logisitic Regression. Maybe it's because the parameters we choose are more suitable for Logisitic Regression.

From the comparision between four SGD methods, we find that AdaDelta converges slowest. Maybe it's also because of the parameters. We also find that epsilon is important. If we choose a too small number for it, AdaDelta would be very slow.

Though it's difficult for us to find the relation between parameters and the goodness of classification, we find that if batch_size is larger, the graph will be smoother. It's because the larger the batch_size is, the closer to batch gradient descent the SGD is. We can find that SGD is much faster than normal GD and if batch_size is larger, SGD will be slower and closer to normal GD.