South China University of Technology

The Experiment Report of *Machine Learning*

College Software College

Subject Software Engineering

Members

Student ID 201530612651

E-mail 835380624@qq.com

Tutor mingkui Tan

Date submitted 2017.12. 11

1. **Topic:**

   Comparison of Various Stochastic Gradient Descent Methods for Solving Classification Problems

2. **Time:**

   2017-12-02 2:00-5:00 PM

3. **Reporter:**

   任嘉宁

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：

   - Load the training set and validation set.
   - Initialize logistic regression model parameters with random

numbers.

- Select the loss function and calculate its derivation.
- Calculate gradient $\mathcal{G}$ toward loss function from partial samples.
- Update model parameters using different optimized methods(NAG， RMSProp， AdaDelta and Adam).
- 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 $\mathcal{L}_{NAG}$， $\mathcal{L}_{RMSProp}$， $\mathcal{L}_{AdaDelta}$ and $\mathcal{L}_{Adam}$.
- Repeate step 4 to 6 for several times, and drawing graph of $\mathcal{L}_{NAG}$， $\mathcal{L}_{RMSProp}$， $\mathcal{L}_{AdaDelta}$ and $\mathcal{L}_{Adam}$ with the number of iterations.

Linear Classification and Stochastic Gradient Descent:

- Load the training set and validation set.
- Initialize SVM model parameters with random numbers.
- Select the loss function and calculate its derivation.
- Calculate gradient $\mathcal{G}$ toward loss function from partial samples.
- Update model parameters using different optimized

methods(NAG， RMSProp， AdaDelta and Adam).

- 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 $\mathcal{L}_{NAG}$, $\mathcal{L}_{RMSProp}$, $\mathcal{L}_{AdaDelta}$ and $\mathcal{L}_{Adam}$.
- Repeate step 4 to 6 for several times, and drawing graph of $\mathcal{L}_{NAG}$, $\mathcal{L}_{RMSProp}$, $\mathcal{L}_{AdaDelta}$ and $\mathcal{L}_{Adam}$ with the number of iterations.

7. Code:

Logistic Regression and Stochastic Gradient Descent：

NAG:

```
# NAG

learning_rate=0.05
w=0.001*np.random.random((1,124))
v=np.zeros((1,124))
gamma=0.9

loss_NAGhistory=[]
test_NAGhistory=[]
for k in range(time):
    mask=np.random.choice(X_train.shape[0],256,replace=False)
    X_batch=X_train[mask]
    y_batch=y_train[mask]
    NAGloss=np.mean(np.log(1+np.exp(-y_batch.reshape(-1,1)*(X_batch.dot(w.T)))))+lamda/2*(np.linalg.norm(w, 2)**2)
    NAGtestloss=np.mean(np.log(1+np.exp(-y_test.reshape(-1,1)*(X_test.dot(w.T)))))+lamda/2*(np.linalg.norm(w, 2)**2)
    dw=lamda*(w-gamma*v) - np.mean((y_batch.reshape(-1,1) * X_batch)/(1 + np.exp(y_batch.reshape(-1,1) * X_batch.dot((w-gamma*v).T))), \
                                axis = 0, keepdims = True)
    v=gamma*v+learning_rate*dw
    w=w-v
    loss_NAGhistory.append(NAGloss)
    test_NAGhistory.append(NAGtestloss)

print(Accuracy(X_test,w))
```

```
0.8506848473680978
```

RMSProp:

```
# RMSProp

learning_rate=0.005
gamma=0.9
epsilon=1e-8
loss_RMSProphistory=[]
test_RMSProphistory=[]
w=0.001*np.random.random((1,124))
G=np.zeros((1,124))
for k in range(time):
    mask=np.random.choice(X_train.shape[0],256,replace=False)
    X_batch=X_train[mask]
    y_batch=y_train[mask]
    RMSProploss=np.mean(np.log(1+np.exp(-y_batch.reshape(-1,1)*(X_batch.dot(w.T)))))+lamda/2*(np.linalg.norm(w, 2)**2)
    RMSProptestloss=np.mean(np.log(1+np.exp(-y_test.reshape(-1,1)*(X_test.dot(w.T)))))+lamda/2*(np.linalg.norm(w, 2)**2)

    dw=lamda*w - np.mean((y_batch.reshape(-1,1) * X_batch)/(1 + np.exp(y_batch.reshape(-1,1) * X_batch.dot(w.T))), \
                     axis = 0, keepdims = True)
    G=gamma*G+(1-gamma)*np.multiply(dw,dw)

    w=w-np.multiply(learning_rate/np.sqrt(G+epsilon),dw)

    loss_RMSProphistory.append(RMSProploss)
    test_RMSProphistory.append(RMSProptestloss)


print(Accuracy(X_test,w))
```

0.8503777409250046

Adadelta:

```
# AdaDelta

gamma=0.95
epsilon=1e-5
loss_AdaDeltahistory=[]
test_AdaDeltahistory=[]
w=0.001*np.random.random((1,124))
G=np.zeros((1,124))
delta_t=np.zeros((1,124))
for k in range(time):
    mask=np.random.choice(X_train.shape[0],256,replace=False)
    X_batch=X_train[mask]
    y_batch=y_train[mask]
    AdaDeltaloss=np.mean(np.log(1+np.exp(-y_batch.reshape(-1,1)*(X_batch.dot(w.T)))))+lamda/2*(np.linalg.norm(w, 2)**2)
    AdaDeltatestloss=np.mean(np.log(1+np.exp(-y_test.reshape(-1,1)*(X_test.dot(w.T)))))+lamda/2*(np.linalg.norm(w, 2)**2)
    dw=lamda*w - np.mean((y_batch.reshape(-1,1) * X_batch)/(1 + np.exp(y_batch.reshape(-1,1) * X_batch.dot(w.T))), \
                     axis = 0, keepdims = True)
    G=gamma*G+(1-gamma)*np.multiply(dw,dw)
    delta_w=-np.multiply(np.sqrt(delta_t+epsilon)/np.sqrt(G+epsilon),dw)
    w=w+delta_w
    delta_t=gamma*delta_t+(1-gamma)*np.multiply(delta_w,delta_w)
    loss_AdaDeltahistory.append(AdaDeltaloss)
    test_AdaDeltahistory.append(AdaDeltatestloss)


print(Accuracy(X_test,w))
```

0.8517290092746146

Adam:

```
# Adam

beta=0.9
gamma=0.999
eta=0.001
epsilon=1e-8
loss_Adamhistory=[]
test_Adamhistory=[]
w=0.001*np.random.random((1,124))
m=np.zeros((1,124))
G=np.zeros((1,124))
for k in range(time):
    mask=np.random.choice(X_train.shape[0],4096,replace=False)
    X_batch=X_train[mask]
    y_batch=y_train[mask]
    Adamloss=np.mean(np.log(1+np.exp(-y_batch.reshape(-1,1)*(X_batch.dot(w.T)))))+lamda/2*(np.linalg.norm(w, 2)**2)
    Adamtestloss=np.mean(np.log(1+np.exp(-y_test.reshape(-1,1)*(X_test.dot(w.T)))))+lamda/2*(np.linalg.norm(w, 2)**2)
    dw=lamda*w - np.mean((y_batch.reshape(-1,1) * X_batch)/(1 + np.exp(y_batch.reshape(-1,1) * X_batch.dot(w.T))), \
                         axis = 0, keepdims = True)
    m=beta*m+(1-beta)*dw
    G=gamma*G+(1-gamma)*np.multiply(delta_w,delta_w)
    alpha=learning_rate*(np.sqrt(1-gamma**(k+1)))/(1-beta**(k+1))
    w=w-alpha*m/(np.sqrt(G+epsilon))
    loss_Adamhistory.append(Adamloss)
    test_Adamhistory.append(Adamtestloss)


print(Accuracy(X_test,w))
```

```
0.8510533750998096
```

Linear Classification and Stochastic Gradient Descent:

NAG:

```
# NAG

v=np.zeros((1,124))
gamma=0.9
w=np.random.random((1,124))*0.001
learning_rate=0.05
loss_NAGhistory=[]
test_NAGhistory=[]

for k in range(time):
    mask=np.random.choice(X_train.shape[0],256,replace=False)
    X_batch=X_train[mask]
    y_batch=y_train[mask]
    dw=np.zeros((1,X_batch.shape[1]))
    NAGloss = lamda * 0.5*(np.linalg.norm(w,2)**2)+np.mean(np.maximum(1-y_batch.reshape(-1,1)*(X_batch.dot(w.T)),0))
    NAGtestloss = lamda * 0.5*(np.linalg.norm(w,2)**2)+np.mean(np.maximum(1-y_test.reshape(-1,1)*(X_test.dot(w.T)),0))
    loss_NAGhistory.append(NAGloss)
    test_NAGhistory.append(NAGtestloss)
    for i in range(X_batch.shape[0]):
        if 1-y_batch[i]*(X_batch[i].dot((w-gamma*v).T))>=0:
            dw += -y_batch[i]*X_batch[i].reshape((1,124))
    dw= dw/X_batch.shape[0]+ lamda*(w-gamma*v)
    v=gamma*v+learning_rate*dw
    w=w-v

print(Accuracy(X_test,w))
```

```
0.8480437319574965
```

RMSProp:

```
#  RMSProp

gamma=0.95
epsilon=1e-8
learning_rate=0.005
w=np.random.random((1,124))*0.001
G=np.zeros((1,124))
loss_RMSProphistory=[]
test_RMSProphistory=[]

for k in range(time):
    mask=np.random.choice(X_train.shape[0],256,replace=False)
    X_batch=X_train[mask]
    y_batch=y_train[mask]
    dw=np.zeros((1,X_batch.shape[1]))
    RMSProploss = lamda * 0.5*(np.linalg.norm(w,2)**2)+np.mean(np.maximum(1-y_batch.reshape(-1,1)*(X_batch.dot(w.T)),0))
    RMSProptestloss = lamda * 0.5*(np.linalg.norm(w,2)**2)+np.mean(np.maximum(1-y_test.reshape(-1,1)*(X_test.dot(w.T)),0))
    loss_RMSProphistory.append(RMSProploss)
    test_RMSProphistory.append(RMSProptestloss)
    for i in range(X_batch.shape[0]):
        if 1-y_batch[i]*(X_batch[i].dot(w.T))>=0:
            dw += -y_batch[i]*X_batch[i].reshape((1,124))
    dw= dw/X_batch.shape[0]+ lamda*w
    G=gamma*G+(1-gamma)*np.multiply(dw,dw)
    w=w-np.multiply(learning_rate/np.sqrt(G+epsilon),dw)

print(Accuracy(X_test,w))
```

0.8502548983477674

Adadelta:

```
#  AdaDelta

gamma=0.95
epsilon=1e-5
G=np.zeros((1,124))
delta_t=np.zeros((1,124))
loss_AdaDeltahistory=[]
test_AdaDeltahistory=[]
w=np.random.random((1,124))*0.001
for k in range(time):
    mask=np.random.choice(X_train.shape[0],512,replace=False)
    X_batch=X_train[mask]
    y_batch=y_train[mask]
    dw=np.zeros((1,X_batch.shape[1]))
    AdaDeltaloss = lamda * 0.5*(np.linalg.norm(w,2)**2)+np.mean(np.maximum(1-y_batch.reshape(-1,1)*(X_batch.dot(w.T)),0))
    AdaDeltatestloss = lamda * 0.5*(np.linalg.norm(w,2)**2)+np.mean(np.maximum(1-y_test.reshape(-1,1)*(X_test.dot(w.T)),0))
    loss_AdaDeltahistory.append(AdaDeltaloss)
    test_AdaDeltahistory.append(AdaDeltatestloss)
    for i in range(X_batch.shape[0]):
        if 1-y_batch[i]*(X_batch[i].dot(w.T))>=0:
            dw += -y_batch[i]*X_batch[i].reshape((1,124))
    dw= dw/X_batch.shape[0]+ lamda*w
    G=gamma*G+(1-gamma)*np.multiply(dw,dw)
    delta_w=-np.multiply(np.sqrt(delta_t+epsilon)/np.sqrt(G+epsilon),dw)
    w=w+delta_w
    delta_t=gamma*delta_t+(1-gamma)*np.multiply(delta_w,delta_w)

print(Accuracy(X_test,w))
```

0.8482279958233524

Adam:

```
# Adam

beta=0.9
gamma=0.999
eta=0.001
epsilon=1e-8
learning_rate=0.001
m=np.zeros((1,124))
G=np.zeros((1,124))
loss_Adamhistory=[]
test_Adamhistory=[]
w=np.random.random((1,124))*0.001
for k in range(time):
    mask=np.random.choice(X_train.shape[0],4096,replace=False)
    X_batch=X_train[mask]
    y_batch=y_train[mask]
    dw=np.zeros((1,X_batch.shape[1]))
    Adamloss = lamda * 0.5*(np.linalg.norm(w,2)**2)+np.mean(np.maximum(1-y_batch.reshape(-1,1)*(X_batch.dot(w.T)),0))
    Adamtestloss = lamda * 0.5*(np.linalg.norm(w,2)**2)+np.mean(np.maximum(1-y_test.reshape(-1,1)*(X_test.dot(w.T)),0))
    loss_Adamhistory.append(Adamloss)
    test_Adamhistory.append(Adamtestloss)
    for i in range(X_batch.shape[0]):
        if 1-y_batch[i]*(X_batch[i].dot(w.T))>=0:
            dw += -y_batch[i]*X_batch[i].reshape((1,124))
    dw= dw/X_batch.shape[0]+ lamda*w
    m=beta*m+(1-beta)*dw
    G=gamma*G+(1-gamma)*np.multiply(delta_w,delta_w)
    alpha=learning_rate*(np.sqrt(1-gamma**(k+1)))/(1-beta**(k+1))
    w=w-alpha*m/(np.sqrt(G+epsilon))

print(Accuracy(X_test,w))
```

```
0.8494564215957251
```

8. The initialization method of model parameters:

W: set with random numbers and multiply 0.001;

G, v, m, delta_t :set all to zeros.

9. The selected loss function and its derivatives:

Logistic Regression and Stochastic Gradient Descent：

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} log(1 + e^{-y_i w^T x_i}) + \frac{\lambda}{2}\|w\|_2^2$$

$$\frac{\partial \mathcal{L}}{\partial w} = -\frac{1}{n}\sum_{i=1}^{n} \frac{y_i x_i}{1 + e^{y_i w^T x_i}} + \lambda w$$

Linear Classification:

$$\mathcal{L} = \frac{\lambda\|w\|^2}{2} + \frac{1}{n}\sum_{i=1}^{n} max(0, 1 - yi(w^T xi))$$

$$g_w(xi) = -yixi \quad 1 - yi(w^T xi) \geq 0$$

$$g_w(xi) = 0 \quad 1 - yi(w^T xi) < 0$$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{1}{n}\sum_{i=1}^{n} g_w(xi) + \lambda w$$

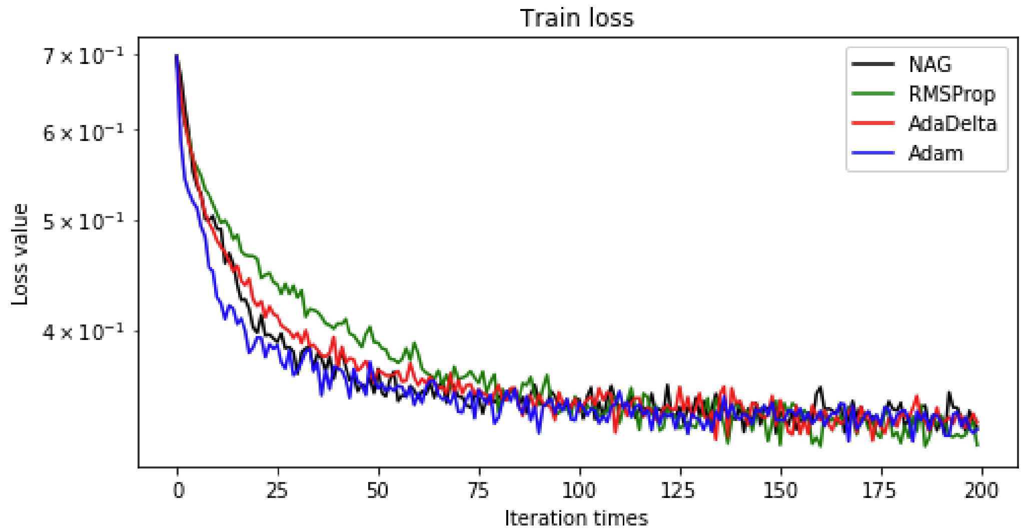
10.Experimental results and curve:

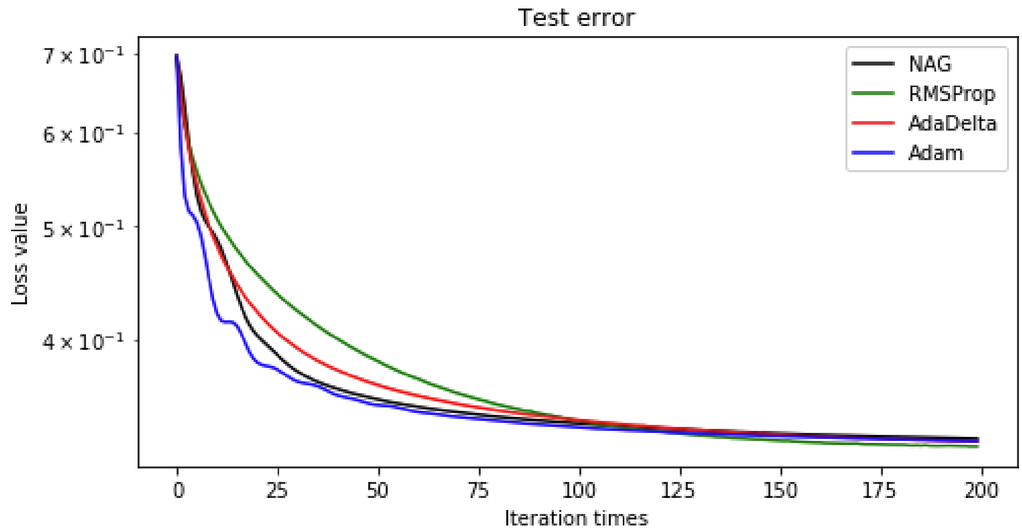
Logistic Regression and Stochastic Gradient Descent：

$\lambda = 0$, Iteration times=200, Batch_size=4096

	Hyper-parameter selection	Predicted Results (Best Results)
NAG	gamma=0.9 learning_rate=0.05	print(Accuracy(X_test,w)) 0.8506848473680978
RMSProp	learning_rate=0.005 gamma=0.9 epsilon=1e-8	print(Accuracy(X_test,w)) 0.8503777409250046
AdaDelta	gamma=0.95 epsilon=1e-5	print(Accuracy(X_test,w)) 0.8517290092746146
Adam	beta=0.9 gamma=0.999 eta=0.001 epsilon=1e-8 learning_rate=0.001	print(Accuracy(X_test,w)) 0.8510533750998096

Loss curve:





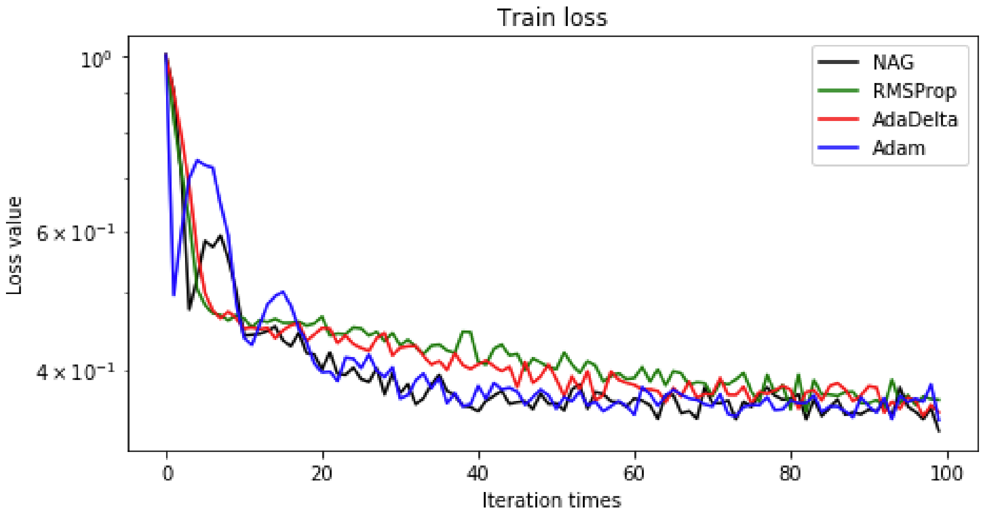
Linear Classification and Stochastic Gradient Descent:

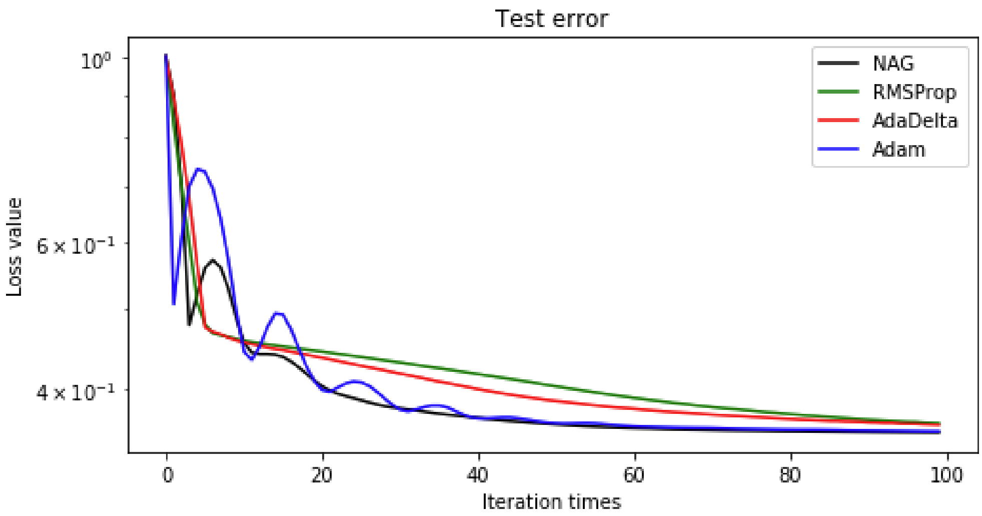
$\lambda = 0$, Iteration times=100, Batch_size=4096

	Hyper-parameter selection	Predicted Results (Best Results)
NAG	gamma=0.9 learning_rate=0.05	print(Accuracy(X_test,w)) 0.8480437319574965
RMSProp	gamma=0.95 epsilon=1e-8 learning_rate=0.005	print(Accuracy(X_test,w)) 0.8502548983477674
AdaDelta	gamma=0.95 epsilon=1e-5	print(Accuracy(X_test,w)) 0.8482279958233524

Adam	beta=0.9 gamma=0.999 eta=0.001 epsilon=1e-8 learning_rate=0.001	print(Accuracy(X_test,w)) 0.8494564215957251

Loss curve:





11.Results analysis:

1. When the Regular modulus equal to 0, the accuracy is the highest. And the loss is much smaller.
2. Batch size is association with the amplitude of train loss, the larger the batch size, the smaller the amplitude of train loss.

3. If learning_rate in NAG =0.001, NAG should iterate a lot of time to convergence(probably 3000times), so I change the learning_rate to 0.05 in NAG. And the iteration time reduce a lot. So do the RMSProp, so I change the learning_rate to 0.005 in RMSProp.
4. If epsilon in AdaDelta = 1e-8, AdaDelta should iterate a lot of time to convergence(probably 3000times), so I change the epsilon to 1e-5, it perform much better.

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

Both methods are common classification algorithms. Look at the objective function, the difference is that logistic loss is used in logistic regression, svm uses hinge loss. The purpose of these two loss functions is to increase the weight of the data points that have a greater impact on the classification and reduce the weight of the data points with less classification. SVM approach is to consider only support vectors, which is the most relevant and classification of the few points to learn classifier. Logistic regression reduces the weights of points farther away from the classification plane through nonlinear mapping, and increases the weight of the data points most relevant to the classification.

13.Summary:

The algorithm behaves differently on different models and also has a relationship with hyperparameter learning rate η and so on.