华南理工大学

South China University of Technology

The Experiment Report of

Machine Learning

College South China University of Technology

Subject Software Engineering

Members Haixu Liu

Student ID 201530612347

E-mail 1059350952@qq.com

Tutor Mingkui Tan

Date submitted 2017. 12. 15

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

2. Time: 12/9/2017

3. Reporter: Haixu Liu

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:

Load the training set and validation set.
Initalize logistic regression model parameters, you can consider initalizing zeros, random numbers or normal distribution.
Initalize SVM model parameters, you can consider initalizing zeros, random numbers or normal distribution.
Select the loss function and calculate its derivation, find more detail in PPT.
Calculate gradient 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 Lnag，Lrmsprop，Ladadelta and Ladam.
Repeate step 4 to 6 for several times, and drawing graph of Lnag，Lrmsprop，Ladadelta and Ladam with the number of iterations.

7. Code:

Logistic Regression:

1. data reading

```
x_train, y_train = load_svmlight_file("a9a.txt", n_features=123)
x_test, y_test = load_svmlight_file("a9a_test.txt", n_features=123)
```

2. matrix

```
x_train = x_train.toarray()
x_test = x_test.toarray()
y_train = y_train.reshape(y_train.shape[0], 1)
y_test = y_test.reshape(y_test.shape[0], 1)

index_range = range(0, x_train.shape[0])
```

3. randomly choose several samples to perform SGD

```
def stochastic_samples(x, y, n):
    samples_index = random.sample(index_range, n)
    x_samples = np.zeros((0, x.shape[1]))
    y_samples = np.zeros((0, y.shape[1]))

    for i in samples_index:
        x_samples = np.r_[x_samples, x[i].reshape(1, x.shape[1])]
        y_samples = np.r_[y_samples, y[i].reshape(1, y.shape[1])]

    return x_samples, y_samples
```

4. four advanced algorithms

```
# NAG
def NAG(x, y, w, params):
    w_new = w - params['GAMMA'] * params['Vt']
    gradient_temp = np.zeros(w.shape)
    for index in range(0, y.shape[0]):
        xi = x[index].reshape(x.shape[1], 1)
        yi = y[index][0]
        gradient_temp += yi / (1 + mt.exp(yi * np.dot(w_new.T, xi)[0][0])) * xi
    gradient = LAMBDA * w_new - 1 / batch_num * gradient_temp
    params['Vt'] = params['GAMMA'] * params['Vt'] + params['learning_rate'] *
gradient
    w = w - params['Vt']
    return w, params

# RMSProp
def RMSProp(x, y, w, params):
    EPSILON = 1e-8
```

```
    loss_gradient = np.zeros(w.shape)
    for index in range(0, y.shape[0]):
        xi = x[index].reshape(1, x.shape[1]).T
        yi = y[index][0]
        loss_gradient += yi / (1 + mt.exp(yi * np.dot(w.T, xi)[0][0])) * xi
    gradient = LAMBDA * w - 1 / batch_num * loss_gradient
    params['Gt'] = params['GAMMA'] * params['Gt'] + (1 - params['GAMMA'])
* (gradient * gradient)
    w = w - (params['learning_rate'] / np.sqrt(params['Gt'] + EPSILON)) *
gradient
    return w, params

# AdaDelta
def AdaDelta(x, y, w, params):
    EPSILON = 1e-8
    GAMMA = params['GAMMA']

    loss_gradient = np.zeros(w.shape)
    for index in range(0, y.shape[0]):
        xi = x[index].reshape(1, x.shape[1]).T
        yi = y[index][0]
        loss_gradient += yi / (1 + mt.exp(yi * np.dot(w.T, xi)[0][0])) * xi
    gradient = LAMBDA * w - 1 / batch_num * loss_gradient
    params['Gt'] = GAMMA * params['Gt'] + (1 - GAMMA) * (gradient *
gradient)
    Delta_W = -(np.sqrt(params['Delta_t'] + EPSILON) / np.sqrt(params['Gt'] +
EPSILON)) * gradient
    params['Delta_t'] = GAMMA * params['Delta_t'] + (1 - GAMMA) *
(Delta_W * Delta_W)
    w = w + Delta_W
    return w, params


# Adam
def Adam(x, y, w, params):
    BETA1 = 0.9
    GAMMA = params['GAMMA']
    EPSILON = 1e-8
    gradient_temp = np.zeros(w.shape)
    for index in range(0, y.shape[0]):
        xi = x[index].reshape(x.shape[1], 1)
        yi = y[index][0]
        gradient_temp += yi / (1 + mt.exp(yi * np.dot(w.T, xi)[0][0])) * xi
    gradient = LAMBDA * w - 1 / batch_num * gradient_temp
```

```
    params['mt'] = BETA1 * params['mt'] + (1 - BETA1) * gradient
    params['Gt'] = GAMMA * params['Gt'] + (1 - GAMMA) * (gradient *
gradient)
    ALPHA = params['learning_rate'] * mt.sqrt(1 - GAMMA **
params['iteration']) / (1 - BETA1 ** params['iteration'])
    w = w - ALPHA * (params['mt'] / np.sqrt(params['Gt'] + EPSILON))
    return w, params
```

5. perform logistic regression

```
for algorithm_index in range(int(len(algorithm_list))):
    w = np.zeros((x_train.shape[1], 1))
    algorithm_params = {'Vt': np.zeros(w.shape),
                        'Gt': np.zeros(w.shape),
                        'Delta_t': np.zeros(w.shape),
                        'mt': np.zeros(w.shape),
                        'learning_rate': 0.1,
                        'GAMMA': 0.9,
                        'BETA1': 0.9,
                        'iteration': 1,
                        }

    if algorithm_list[algorithm_index] == AdaDelta:
        algorithm_params['GAMMA'] = 0.999

    for i in range(0, iter_num):
        x_samples, y_samples = stochastic_samples(x_train, y_train,
batch_num)
        w, algorithm_params = algorithm_list[algorithm_index](x_samples,
y_samples, w, algorithm_params)
        algorithm_params['iteration'] = i + 1

        loss_sum = 0
        for index in range(0, y_test.shape[0]):
            xi = x_test[index].reshape(1, x_test.shape[1]).T
            yi = y_test[index][0]

            wtxi = np.dot(w.T, xi)[0][0]
            loss_sum += mt.log(1 + mt.exp(-yi * wtxi))
        test_loss = LAMBDA * 1 / 2 * np.dot(w.T, w)[0][0] + 1 / x_test.shape[0]
* loss_sum
        loss_list[algorithm_names[algorithm_index]].append(test_loss)
```

linear classification:

1. four advanced algorithms

```
# NAG
def NAG(x, y, w, params):
    w_new = w - params['GAMMA'] * params['Vt']
    gradient_temp = np.zeros(w.shape)
    gradient_result = np.zeros(w.shape)
    for index in range(0, y.shape[0]):
        xi = x[index].reshape(x.shape[1], 1)
        yi = y[index][0]
        if yi * np.dot(w_new.T, xi)[0][0] <= 1:
            gradient_temp = - yi * xi
        elif yi * np.dot(w_new.T, xi)[0][0] > 1:
            gradient_temp = np.zeros(w.shape)
        gradient_result += gradient_temp
    gradient = w_new + initial_c * 1 / batch_num * gradient_result

    params['Vt'] = params['GAMMA'] * params['Vt'] + params['learning_rate'] *
gradient
    w = w - params['Vt']

    return w, params


# RMSProp
def RMSProp(x, y, w, params):
    EPSILON = 1e-8

    gradient_temp = np.zeros(w.shape)
    gradient_result = np.zeros(w.shape)
    for index in range(0, y.shape[0]):
        xi = x[index].reshape(x.shape[1], 1)
        yi = y[index][0]
        if yi * np.dot(w.T, xi)[0][0] <= 1:
            gradient_temp = - yi * xi
        elif yi * np.dot(w.T, xi)[0][0] > 1:
            gradient_temp = np.zeros(w.shape)
        gradient_result += gradient_temp
    gradient = w + initial_c * 1 / batch_num * gradient_result
    params['Gt'] = params['GAMMA'] * params['Gt'] + (1 - params['GAMMA'])
* (gradient * gradient)
    w = w - (params['learning_rate'] / np.sqrt(params['Gt'] + EPSILON)) *
gradient
    return w, params
```

```
# AdaDelta
def AdaDelta(x, y, w, params):
    EPSILON = 1e-8
    GAMMA = params['GAMMA']
    gradient_temp = np.zeros(w.shape)
    gradient_result = np.zeros(w.shape)
    for index in range(0, y.shape[0]):
        xi = x[index].reshape(x.shape[1], 1)
        yi = y[index][0]
        if yi * np.dot(w.T, xi)[0][0] <= 1:
            gradient_temp = - yi * xi
        elif yi * np.dot(w.T, xi)[0][0] > 1:
            gradient_temp = np.zeros(w.shape)
        gradient_result += gradient_temp
    gradient = w + initial_c * 1 / batch_num * gradient_result
    params['Gt'] = GAMMA * params['Gt'] + (1 - GAMMA) * (gradient *
gradient)
    Delta_W = -(np.sqrt(params['Delta_t'] + EPSILON) / np.sqrt(params['Gt'] +
EPSILON)) * gradient
    params['Delta_t'] = GAMMA * params['Delta_t'] + (1 - GAMMA) *
(Delta_W * Delta_W)
    w = w + Delta_W
    return w, params


# Adam
def Adam(x, y, w, params):
    BETA1 = 0.9
    GAMMA = params['GAMMA']
    EPSILON = 1e-8
    gradient_temp = np.zeros(w.shape)
    gradient_result = np.zeros(w.shape)
    for index in range(0, y.shape[0]):
        xi = x[index].reshape(x.shape[1], 1)
        yi = y[index][0]
        if yi * np.dot(w.T, xi)[0][0] <= 1:
            gradient_temp = - yi * xi
        elif yi * np.dot(w.T, xi)[0][0] > 1:
            gradient_temp = np.zeros(w.shape)
        gradient_result += gradient_temp
    gradient = w + initial_c * 1 / batch_num * gradient_result
    params['mt'] = BETA1 * params['mt'] + (1 - BETA1) * gradient
```

```
    params['Gt'] = GAMMA * params['Gt'] + (1 - GAMMA) * (gradient *
gradient)
    ALPHA = params['learning_rate'] * mt.sqrt(1 - GAMMA **
params['iteration']) / (1 - BETA1 ** params['iteration'])
    w = w - ALPHA * (params['mt'] / np.sqrt(params['Gt'] + EPSILON))
```

2. perform linear regression:

```
for algorithm_index in range(int(len(algorithm_list))):
    w = np.zeros((x_train.shape[1], 1))
    algorithm_params = {'Vt': np.zeros(w.shape),
                        'Gt': np.zeros(w.shape),
                        'Delta_t': np.zeros(w.shape),
                        'mt': np.zeros(w.shape),
                        'learning_rate': 0.01,
                        'GAMMA': 0.9,
                        'BETA1': 0.9,
                        'iteration': 1,
                        }

    if algorithm_list[algorithm_index] == AdaDelta:
        algorithm_params['GAMMA'] = 0.999

    for i in range(0, iter_num):
        x_samples, y_samples = stochastic_samples(x_train, y_train,
batch_num)
        w, method_params = algorithm_list[algorithm_index](x_samples,
y_samples, w, algorithm_params)
        algorithm_params['iteration'] = i + 1

        loss_sum = 0
        for index in range(0, y_test.shape[0]):
            xi = x_test[index].reshape(1, x_test.shape[1]).T
            yi = y_test[index][0]
            wtxi = np.dot(w.T, xi)[0][0]
            if yi * wtxi <= 1:
                hinge_loss = 1 - yi * wtxi
            elif yi * wtxi > 1:
                hinge_loss = 0
            loss_sum += hinge_loss
        test_loss = 1 / 2 * (np.dot(w.T, w)[0][0]) ** 2 + (initial_c / batch_num)
* loss_sum
        loss_list[algorithm_names[algorithm_index]].append(test_loss)
```

8. Selection of validation (hold-out, cross-validation, k-folds cross-validation, etc.):

Cross-validation

9. The initialization method of model parameters:

For the logistic regression experiment, the initialization is:

```
LAMBDA = 1
batch_num = 100
threshold = 0.5
iter_num = 100

NAG_loss_list = []
RMSProp_loss_list = []
AdaDelta_loss_list = []
Adam_loss_list = []
loss_list = {'NAG': NAG_loss_list,
                    'RMSProp': RMSProp_loss_list,
                    'AdaDelta': AdaDelta_loss_list,
                    'Adam': Adam_loss_list
                    }

algorithm_list = [NAG, RMSProp, AdaDelta, Adam]
algorithm_names = ['NAG', 'RMSProp', 'AdaDelta', 'Adam']
```

For the linear regression experiment, the initialization is：

```
LAMBDA = 1
batch_num = 100
threshold = 0.5
iter_num = 100
hinge_loss = 0
initial_c = 0.9

NAG_loss_list = []
RMSProp_loss_list = []
AdaDelta_loss_list = []
Adam_loss_list = []
loss_list = {'NAG': NAG_loss_list,
                    'RMSProp': RMSProp_loss_list,
                    'AdaDelta': AdaDelta_loss_list,
```

'Adam': Adam_loss_list
}

algorithm_list = [NAG, RMSProp, AdaDelta, Adam]
algorithm_names = ['NAG', 'RMSProp', 'AdaDelta', 'Adam']

10. The selected loss function and its derivatives:

For the logistic regression experiment:

$$J(w) = -\frac{1}{n}[\sum_{i=1}^{N} y_i \log h_w(x_i) + (1 - y_i) \log(1 - h_w(x_i))]$$

In Python:
test_loss = LAMBDA * 1 / 2 * np.dot(w.T, w)[0][0] + 1 / x_test.shape[0] * loss_sum

For the linear classification experiment:

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

In Python:
test_loss = 1 / 2 * (np.dot(w.T, w)[0][0]) ** 2 + (initial_c / batch_num) * loss_sum

11. Experimental results and curve:

Hyper-parameter selection (η , epoch, etc.):

$$y = w^T x + b$$

w = np.zeros((x_train.shape[1], 1)) η = 0.01 epoch = 100

Assessment Results (based on selected validation):

The essential idea of NAG is using Momentum to predict the next step of gradient, in another word, when we calculate the gradient, we not only consider the w, but also another vector: v, which means the gradient will be pushed forward by γv_{t-1} step.

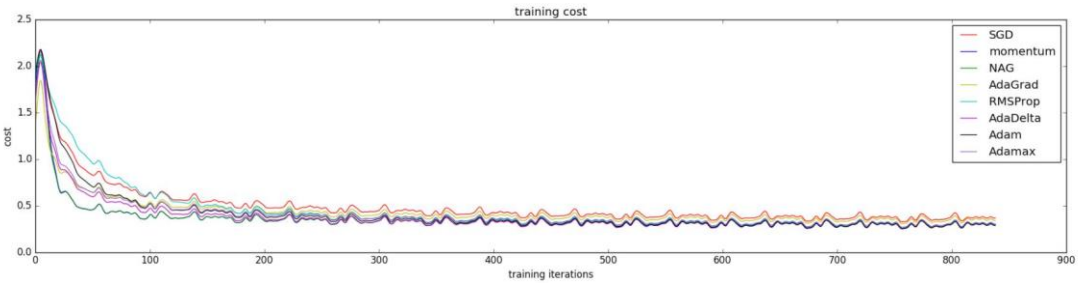
When we perform the RMSProp advanced algorithm, we should remind that there is another vector G_t should be considered, the damaged

exponential γ can be set to 0.9, and ε is a small value 1e-8, it is an upgraded method of AdaGRad.

AdaDelta also can solve the disadvantages of AdaGrad, and it is more effective than RMSProp, compared with RMSProp

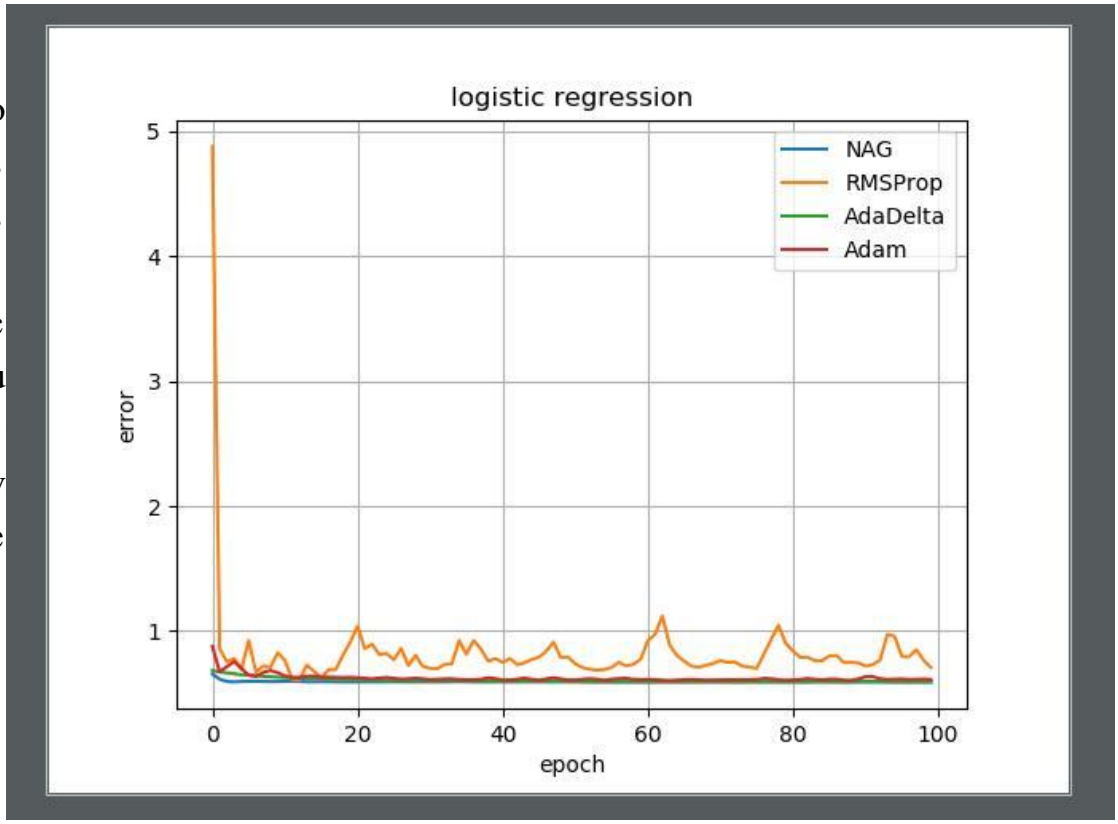
Adam is the most advanced algorithm, though it does not perform more effective than other algorithms, or it has a giant advantage over other methods, however, we can figure out that it solves problems of all the three methods mentioned above and provides a better idea to calculate the gradient and do the upgrade.

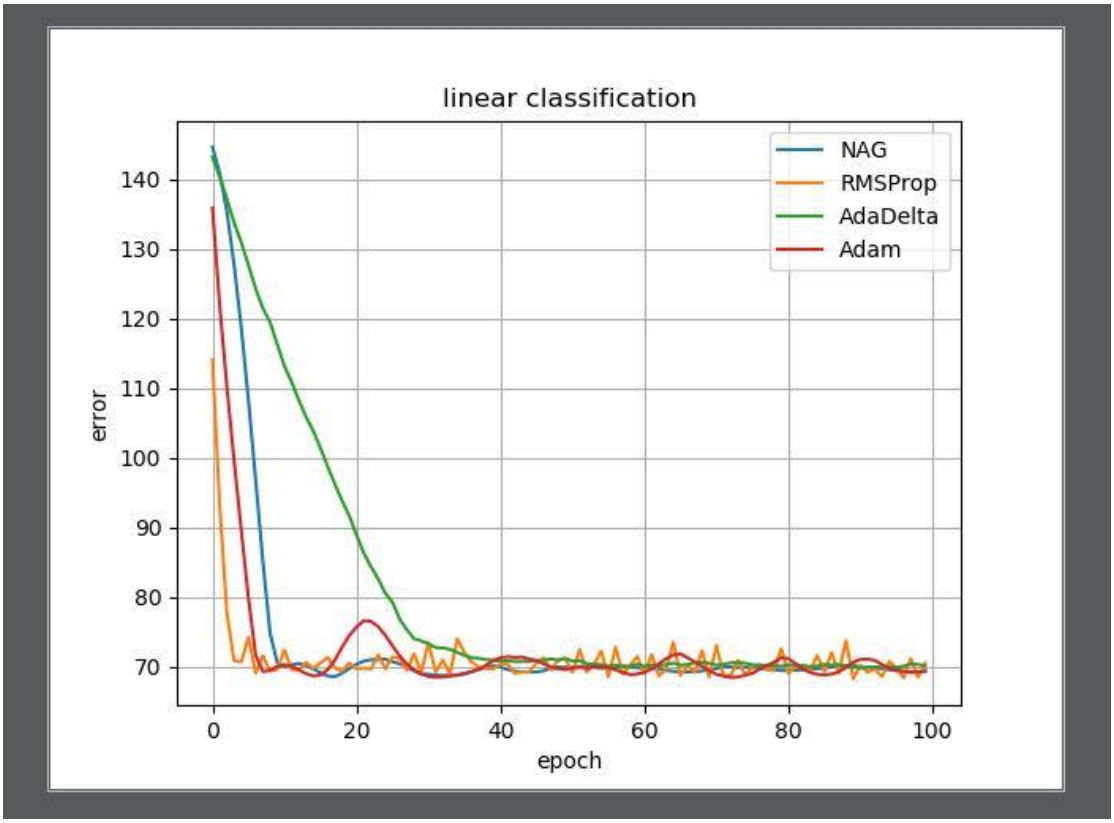
Predicted Results (Best Results):



o
s
s

c
u
r
v
e
:





12. Results analysis:

According to the first graph, we can see that the yellow line is the most obvious, which means it has more features that can be analysed, and of course, the yellow refers to RMSProp approach, it can be inferred that the loss rate decreases really fast since the first ten rounds. All those phenomenon can be attributed to the usage of learning rate or other parameters. Moreover, it is obvious that only the RMSProp approach brings much more fluctuation than other algorithms, and it brings the worst output, which means it does not remain stable to a certain value.

As for the second experiment, four lines can be told easily, from the graph we can infer that AdaDelta is the slowest way, however it is not the worst way since it decrease to a certain value just liker other methods do, like we mention before, RMSProp also brings a lot fluctuation. In addition, the red, which belongs to Adam, shows a little fluctuation, too, it may due to the code I write, or the algorithm itself, since time is limited I can not spend too much time on this kind of research.

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

Linear regression can be regarded as a Perception, its activation function is f(x)=x, logistic regression can be also regarded as a Perception, however, its activation function is sigmoid function.

In general, linear regression is a true regression, we find a line or a hyperplane to fit into several samples, in order to have the best fitting hyperplane, there are several problems needed us to solve, but logistic regression is more life a classifier, rather than a regression.

However, there are some regression ideas contained in logistic regression, such like we need a sigmoid function, calculate loss function and gradient, we also need gradient descent to continuously upgrade or w and gradient. Therefore, the name “logistic regression” is also reasonable.

Logistic Regression:

$$\mathrm{f(x)} = sigmoid(w^T x + b)$$

Logistic Classifier:

$$y = \begin{cases} 1 & f(x) \geq 0.5 \\ -1 & f(x) < 0.5 \end{cases}$$

Linear Regression:

$$\mathrm{f(x)} = w^T x + b$$

Linear Classifier:

$$y = \begin{cases} 1 & f(x) \geq 0 \\ -1 & f(x) < 0 \end{cases}$$

f(x) in Linear Classifier is not computed through Linear Regression, Linear Classifier uses hyperplane as a decision boundary, logistic regression and SVM can be seen as Linear Classifier.

14. Summary:

From the two experiments, I understand the basic idea of logistic regression, I am capable to tell the difference between logistic regression and linear regression. Logistic regression can be used for classification, different from support vector machine, it can deal with non-continuous features, and continuous features as well. Since it has a sigmoid function, which can limit the value between 0 and 1, we give the model a threshold, due to the result output, samples can be classified.

Different from GD, SGD randomly choose several samples and compute their gradient, instead of calculating all the samples, SGD saves a lot of time and computer space in order to save time, and it provides a more effective method.