The Experiment Report of Machine Learning

SCHOOL： SCHOOL OF SOFTWARE ENGINEERING

SUBJECT： SOFTWARE ENGINEERING

Author:

ABAS MOHAMMED NAGI ABORAS

Supervisor:

Pro. Mingkui Tan

Student ID：201722800094

Grade:

Undergraduate or Graduate

December 22 , 2017

CONTENT OF REPORT:

Abstract

This experiment focuses on two parts, one is logistic regression, the other is linear classification. Both models are updated with four different gradient descent methods and tested in the a9a in LIBSVM Data. For logistic regression, the accuracy of validation set used NAG, RMSProp, AdaDelta and Adam.

Logistic Regression and Stochastic Gradient Descent

Introduction

In the first part of this exercise, we'll build a logistic regression model. Now we need to implement logistic regression so we can train a model to predict the outcome. The theory is that this helps to minimize over fitting and improve the model's ability to generalize.

Experimental steps:

1. Load the training set and validation set.
2. Initialize logistic regression model parameters, you can consider initializing zeros, random numbers or normal distribution.
3. Select the loss function and calculate its derivation.
4. Calculate gradient toward loss function from partial samples.
5. Update model **parameters using different optimized methods(NAG，RMSProp，AdaDelta and Adam).**
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 L_{NAG} ，L_{RMSPRO} ，$L_{AdaDelta}$ and L_{Adam}.
7. Repeater step 4 to 6 for several times, and **drawing graph of** L_{NAG} ，L_{RMSPRO} ，$L_{AdaDelta}$ and L_{Adam} **with the number of iterations**.

EXPERIMENTS :

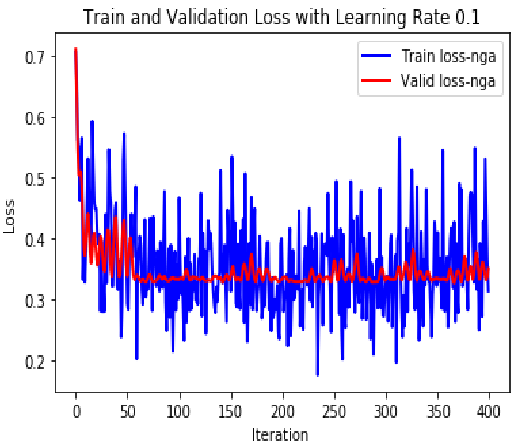
1. Dataset

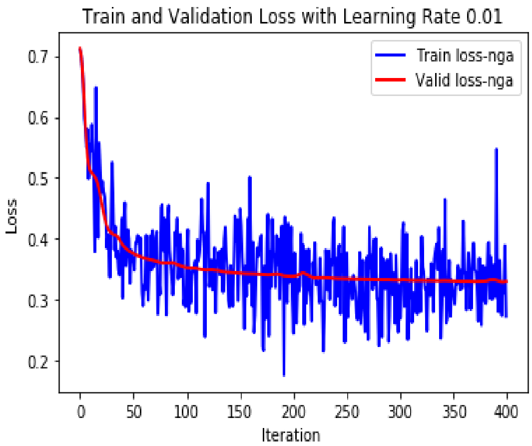
In this experiment use Data set (a9a) in LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. Please download the training set and validation set.

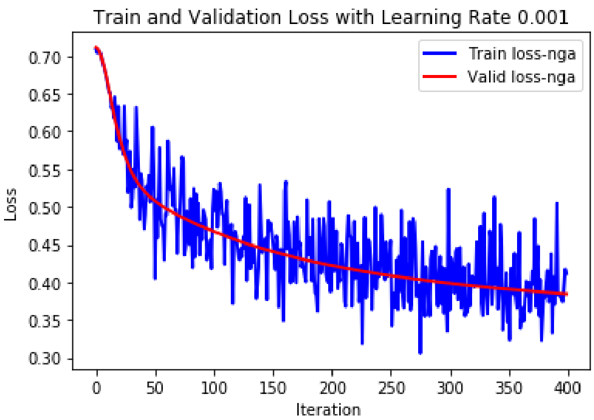
2. Implementation:

First: Methods (NAG):

This result method NAG with different learning rate: 0.1 ,0.01, 0.001

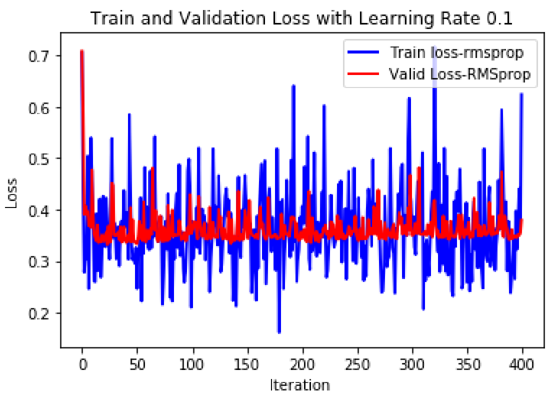


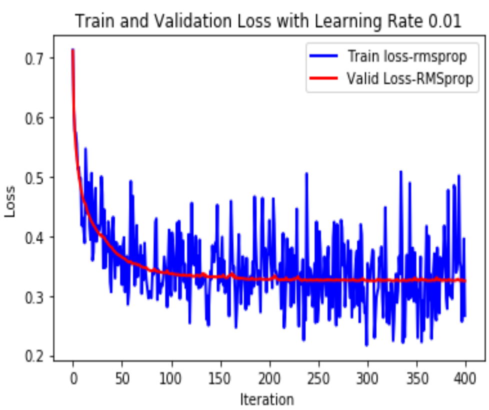


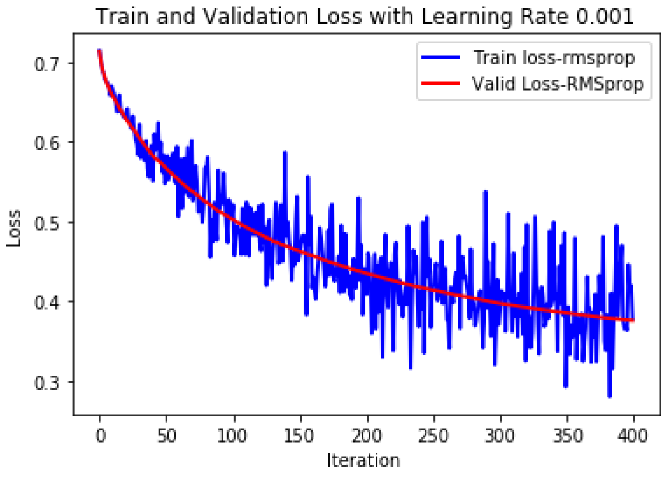


Second: Methods (RMS pop):

This result method RMS pop with different learning rate: 0.1 ,0.01, 0.001

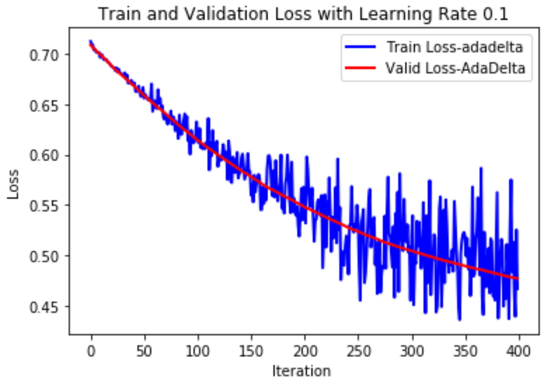


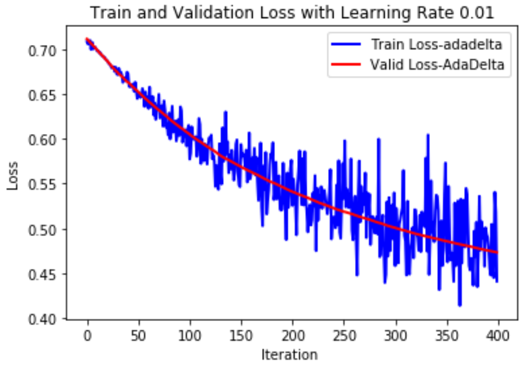




Third: Methods (AdaDelta)

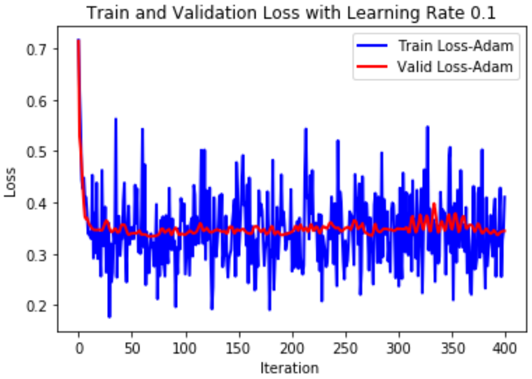
This result method AdaDelta with different learning rate: 0.1 ,0.01, 0.001

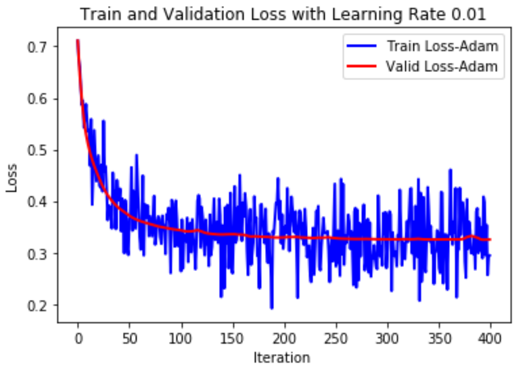


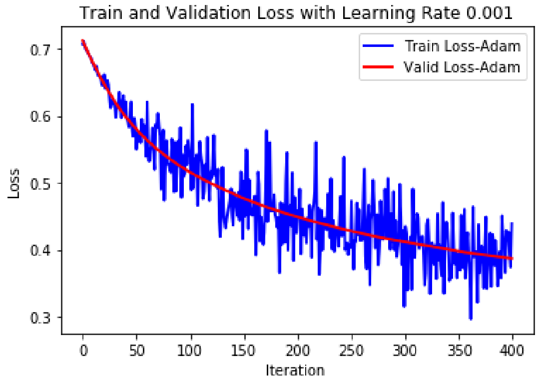


Forth: Methods(Adam)

This result method ADAM with different learning rate: 0.1 ,0.01, 0.001







Code:

This some picture from code logistic regression.

Logistic Regression and Stochastic Gradient Descent

```
# import Data set
import numpy as np
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split

iteration_times = 400
train_valid_ratio = 0.9
```

```
# lode Data
a_train = load_svmlight_file('./DATA/a9a')
a_test = load_svmlight_file('./DATA/a9a.t', n_features=a_train[0].shape[1])
print('\n')
print('This value train and test data')
print('Train Data Shape :', a_train[0].shape, 'Test Data Shape:', a_test[0].shape)
print('Each sample has  : 123/123 ')
```

```
This value train and test data
Train Data Shape : (32561, 123) Test Data Shape: (16281, 123)
Each sample has  : 123/123
```

```
# Return Train &  Validation

def prepare_data(train, test, shuffle = False):
    x_train, y_train, x_valid, y_valid = train[0], train[1], test[0], test[1]

    y_train[y_train == -1] = 0
    y_valid[y_valid == -1] = 0
    return x_train, x_valid, y_train, y_valid
```

```
def init_param(n, method='randn', scale=0.01):

  #  initilize param

    if n < 0:
        return
    if method == 'randn':
        if n > 1:
            return scale * np.random.random(size=n)
        else:
            return scale * np.random.random()
    if method == 'zero':
        if n > 1:
            return np.zeros(n)
        else:
            return 0
```

```
# Choose loss function, calculate gradient and update params
def logistic_model(x, w):
    try:
        y = 1 / (1 + np.exp(-x * w))
        return y
    except Exception as e:
        print("THE VALUE X don't Match W")


def loss_function(y, y_pred):
    return -1 * np.mean(y * np.log(y_pred) + (1 - y) * np.log(1 - y_pred))  # use log loss function

def compute_gradient(x, y, y_pred):
    # compute gradient, perhaps different loss fucntion.
    return (y_pred - y) * x / x.shape[0]


def get_batch(x, batch_size=20):
    if batch_size < 0:
        return
    # default batch_size 20
    return np.random.choice(range(x.shape[0]) , size=batch_size)
```

```
#*************************** Update adadelta ***********************************************
def update_params_adadelta(x, y, y_pred, w, decay_rate, eps, cum_grad, cum_u_w, u_w_list):

    # Comput Root Mean square of Cumulative Gradient
    d_w = compute_gradient(x, y, y_pred)  # alpha is learning rate
    cum_grad = decay_rate * cum_grad + (1 - decay_rate) * (d_w ** 2)
    rms_grad = np.sqrt(cum_grad + eps)

    # Comput Root Mean Square Of Cumulative Update Value of w
    cum_u_w = decay_rate * cum_u_w + (1 - decay_rate) * (u_w_list[-1] ** 2)
    rms_u_w = np.sqrt(cum_u_w + eps)

    # Update weight
    u_w = rms_u_w * d_w / rms_grad
    w = w - u_w  # alpha is learning rate

    # Store params
    u_w_list.append(u_w)

    return w, cum_grad, cum_u_w
```

```
#********************************** Update Adam ***********************************************

def update_params_adam(x, y, y_pred, w, m, v, betal_one, belta_two, eps, alpha, iter_step):


    d_w = compute_gradient(x, y, y_pred)  # alpha is learning rate, compute the gradient of w
    m = betal_one * m + (1 - betal_one) * d_w
    mt = m / (1 - np.power(betal_one, iter_step + 1))
    v = belta_two * v + (1 - belta_two) * (d_w ** 2)
    vt = v / (1 - np.power(belta_two, iter_step + 1))
    w = w - alpha * mt / (np.sqrt(vt) + eps)
    return w, m, v
```

```python
#  Different Method Of Updating Params
def nga_model(x_train, x_valid, y_train, y_valid, iteration_times, mu, alpha):

    train_loss = []
    valid_loss = []
    # initial params
    w = init_param(x_train.shape[1])  # initialize weight and bias
    v = init_param(x_train.shape[1], method='zero')  # initial with zeros
    for i in range(iteration_times):
        # Get Train & Test Data
        index = get_batch(x_train, batch_size=int(x_train.shape[0]/iteration_times))
        x_train_batch = x_train[index, :]
        y_train_batch = y_train[index]

        # Logistic Model & Train loss
        y_pred = logistic_model(x_train_batch, w)
        loss_t = loss_function(y_train_batch, y_pred)  # square loss function of train data
        train_loss.append(loss_t)

        # Logistic Model & Validation loss
        y_valid_pred = logistic_model(x_valid, w)
        loss_v = loss_function(y_valid, y_valid_pred)  # square loss function of validation data
        valid_loss.append(loss_v)


        w, v = update_params_nga(x_train_batch, y_train_batch, y_pred, w, v, mu, alpha)

    return train_loss, valid_loss
```

```python
#****************** Draw The Graph Of Train And Validation

import matplotlib.pyplot as PL


print("****************************************************************** ")
print("This Result Logistic Regression and Stochastic Gradient Descent ")
print("****************************************************************** ")

def plot_result(train_loss, valid_loss, fig_config):

    for i in range(len(alpha)):
        PL.figure()
        PL.title('Train and Validation Loss with Learning Rate '+ str(alpha[i]))
        PL.plot(range(len(train_loss[i])), train_loss[i], linewidth=2.0,
                 color=fig_config['color'][0], label=fig_config['label'][0])
        PL.plot(range(len(valid_loss[i])), valid_loss[i], linewidth=2.0,
                 color=fig_config['color'][1], label=fig_config['label'][1])
        PL.xlabel('Iteration ')
        PL.ylabel('Loss ')
        PL.legend(fig_config['label'])
        PL.show()
        print('\n')
```

Linear Classification and Stochastic Gradient Descent

Introduction:

In the second part linear classification, it is solved by four different gradient descent methods.Through analyzing the different updating process, we can further understand the principle of gradient descent. We will practice on a biggerscale datasets to understand the process of optimization andparameter adjustment. We expect linear classification can get higher accuracy.

Experimental steps:

1. Load the training set and validation set.
2. Initialize SVM model parameters, you can consider initializing zeros, random numbers or normal distribution.
3. Select the loss function and calculate its derivation.
4. Calculate gradient toward loss function from partial samples.
5. Update model **parameters using different optimized methods (NAG, RMSProp， AdaDelta and Adam).**
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 lossL_{NAG} ,L_{RMSPRO} ,$L_{AdaDelta}$ and L_{Adam}.
7. Repeater step 4 to 6 for several times, and **drawing graph of** L_{NAG} , L_{RMSPRO} ,$L_{AdaDelta}$ and L_{Adam}.**With the number of iterations**.

EXPERIMENTS :

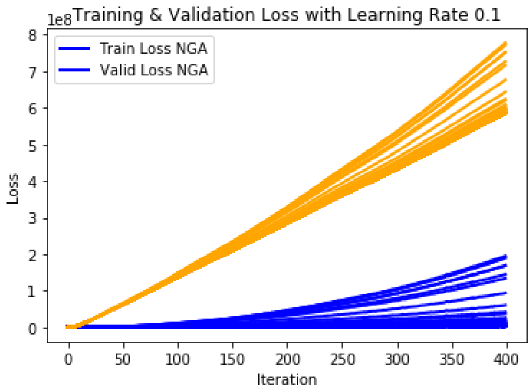
1. Dataset

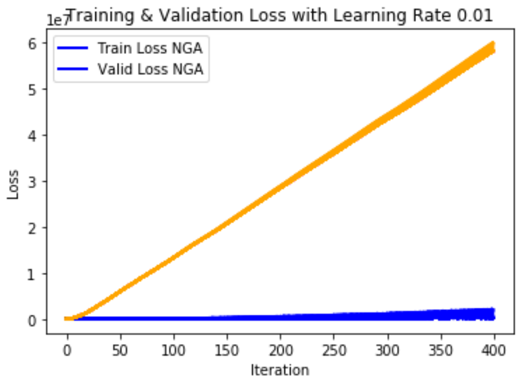
In this experiment use Data set (a9a) in LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. Please download the training set and validation set.

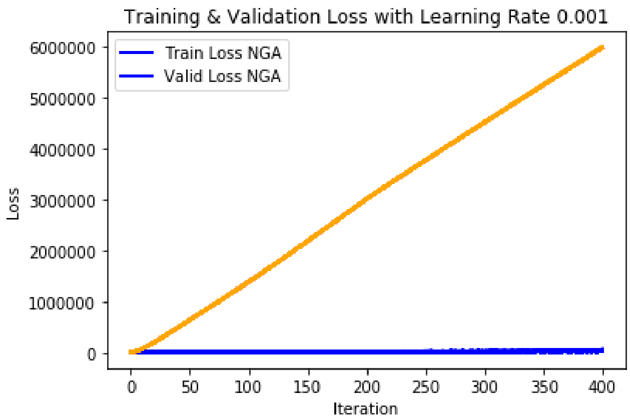
2. Implementation:

First: Methods (NAG):

This result method NAG with different learning rate: 0.1 ,0.01, 0.001

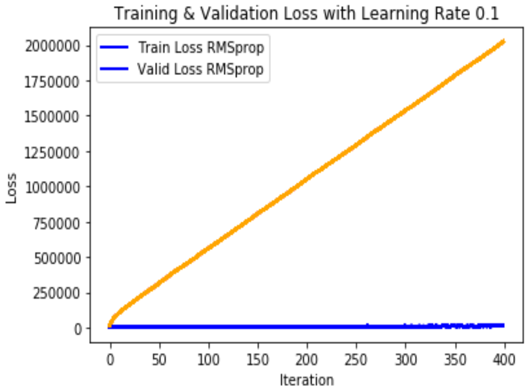


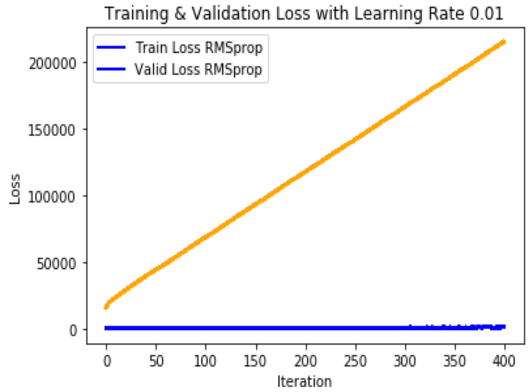


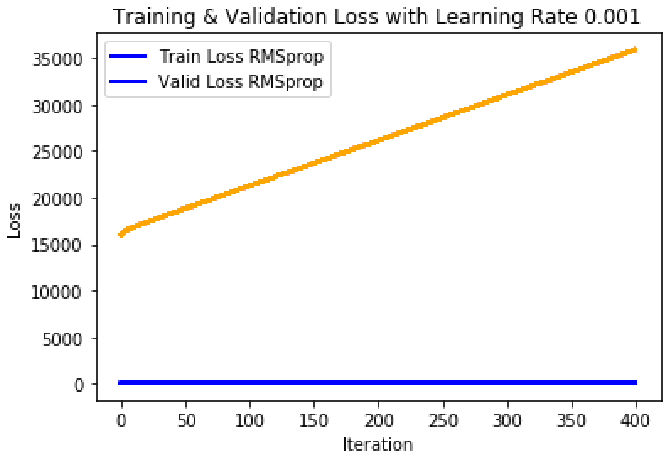


Second: Methods (RMS pop):

This result method RMS pop with different learning rate: 0.1 ,0.01, 0.001

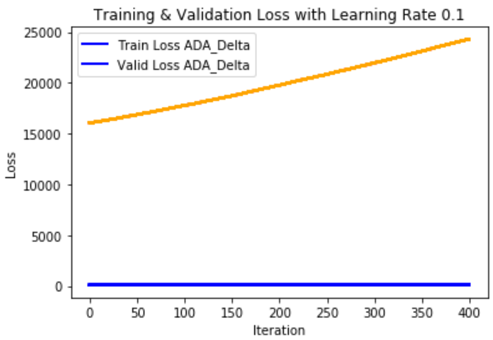


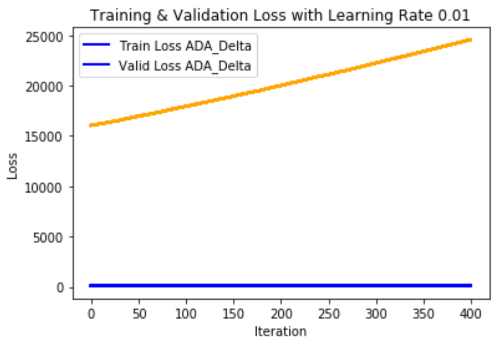


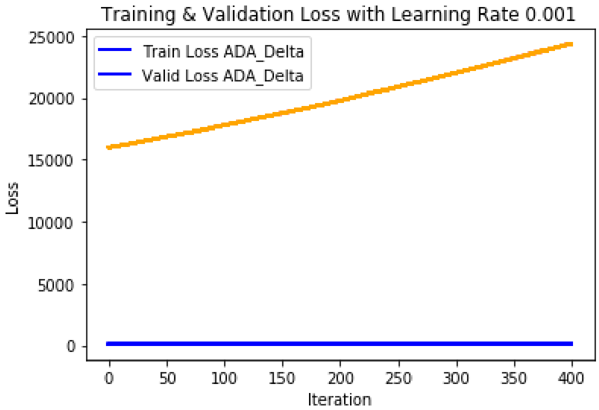


Third: Methods(AdaDelta)

This result method AdaDelta with different learning rate: 0.1 ,0.01, 0.001

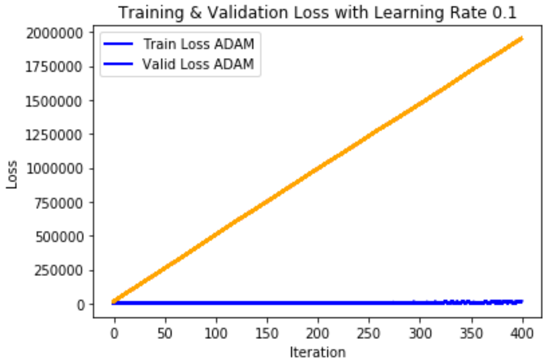


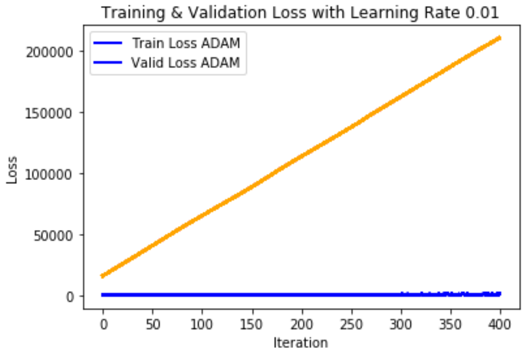




Third: Methods(Adam)

This result method Adam with different learning rate: 0.1 ,0.01, 0.001





Code:

This some picture from code linear classification SGD.

Linear Classification and Stochastic Gradient Descent

```
# import Data set
import numpy as np
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split

iteration_times = 400   # Iteration Times
train_valid_ratio = 0.9   # Determine the Ratio Of Train To Validation Data
```

```
# lode Data
a_train = load_svmlight_file('./DATA/a9a')
a_test = load_svmlight_file('./DATA/a9a.t', n_features=a_train[0].shape[1])
print('\n')
print('This value train and test data')
print('Train Data Shape :', a_train[0].shape, 'Test Data Shape:', a_test[0].shape)
print('Each sample has  : 123/123 ')
```

```
This value train and test data
Train Data Shape : (32561, 123) Test Data Shape: (16281, 123)
Each sample has  : 123/123
```

```
# Return Train and Validation.
def prepare_data(train, test, shuffle = False):
    x_train, y_train, x_valid, y_valid = train[0], train[1], test[0], test[1]
    y_train[y_train == -1] = 0
    y_valid[y_valid == -1] = 0
    return x_train, x_valid, y_train, y_valid
```

```
# initial parameter with normal distribution

def init_param(n, method='randn', scale=0.01):

    if n < 0:
        return
    if method == 'randn':
        if n > 1:
            return scale * np.random.random(size=n)
        else:
            return scale * np.random.random()
    if method == 'zero':
        if n > 1:
            return np.zeros(n)
        else:
            return 0
```

```
# Choose Loss Function

def linear_model(x, w):
    try:
        y_prob = x*w
        # judge the class of linear model
        y = np.array([0]*y_prob.shape[0])
        y[y_prob < 0.5] = -1
        y[y_prob > 0.5] = 1
        return y_prob
    except Exception as e:
        print(" The value X Don't Match value w ")

def loss_function(y, y_pred, w):
    hinge = np.max([[0]*y.shape[0],1 - y*y_pred], axis=0)
    return np.sum(hinge) + 0.5 * np.square(w)     # use hinge loss function

def compute_gradient(x, y, y_pred):
    # compute gradient, perhaps different loss fucntion, now it's log loss function
    return y[y * y_pred < 1]*x[y * y_pred < 1]
```

```
def get_batch(x, batch_size=20):
    if batch_size < 0:
        return
    # default batch_size 20
    return np.random.choice(range(x.shape[0]) , size=batch_size)
def update_params_nga(x, y, y_pred, w, v, mu, alpha):

    d_w = compute_gradient(x, y, y_pred)  # evaluate dx_head
    v_prev = v
    v = mu * v - alpha * d_w  # alpha is learning rate
    w = w + mu * v_prev + (1 + mu) * v
    return w, v

# **************************** Update ParamsRMS prop ******************************
def update_params_rmsprop(x, y, y_pred, w, cache, decay_rate, eps, alpha):

    d_w = compute_gradient(x, y, y_pred)  # alpha is learning rate, compute the gradient of w
    cache = decay_rate * cache + (1 - decay_rate) * (d_w ** 2)  # compute cache, larger cache, smaller learning rate
    w = w - alpha * d_w / (np.sqrt(cache) + eps)  # update weight
    return w, cache
```

```
# *************************** Update Params adadelta*******************************
def update_params_adadelta(x, y, y_pred, w, decay_rate, eps, cum_grad, cum_u_w, u_w_list):

    # comput root mean square of cumulative gradient
    d_w = compute_gradient(x, y, y_pred)  # alpha is learning rate, compute the gradient of w
    cum_grad = decay_rate * cum_grad + (1 - decay_rate) * (d_w ** 2)
    rms_grad = np.sqrt(cum_grad + eps)

    # comput root mean square of cumulative update value of w
    cum_u_w = decay_rate * cum_u_w + (1 - decay_rate) * (u_w_list[-1] ** 2)
    rms_u_w = np.sqrt(cum_u_w + eps)

    # update weight
    u_w = rms_u_w * d_w / rms_grad
    w = w - u_w  # alpha is learning rate

    # store params
    u_w_list.append(u_w)

    return w, cum_grad, cum_u_w
```

```
#********************************* Update Params adam **************************
def update_params_adam(x, y, y_pred, w, m, v, betal_one, belta_two, eps, alpha, iter_step):


    d_w = compute_gradient(x, y, y_pred)  # alpha is learning rate, compute the gradient of w
    m = betal_one * m + (1 - betal_one) * d_w
    mt = m / (1 - np.power(betal_one, iter_step + 1))
    v = belta_two * v + (1 - belta_two) * (d_w ** 2)
    vt = v / (1 - np.power(belta_two, iter_step + 1))
    w = w - alpha * mt / (np.sqrt(vt) + eps)
    return w, m, v
```

```python
# Different Method of Updating params

def nga_model(x_train, x_valid, y_train, y_valid, iteration_times, mu, alpha):

    train_loss = []
    valid_loss = []
    # initial params
    w = init_param(x_train.shape[1])  # initialize weight and bias
    v = init_param(x_train.shape[1], method='zero')  # initial with zeros
    for i in range(iteration_times):
        # Get Train & Test Data
        index = get_batch(x_train, batch_size=int(x_train.shape[0]/iteration_times))
        x_train_batch = x_train[index, :]
        y_train_batch = y_train[index]

        # logistic Model & Train loss
        y_pred = linear_model(x_train_batch, w)
        loss_t = loss_function(y_train_batch, y_pred, w)  # square loss function of train data
        train_loss.append(loss_t)

        # logistic Model  &  Validation Loss
        y_valid_pred = linear_model(x_valid, w)
        loss_v = loss_function(y_valid, y_valid_pred, w)  # square loss function of validation data
        valid_loss.append(loss_v)

        w, v = update_params_nga(x_train_batch, y_train_batch, y_pred, w, v, mu, alpha)

    return train_loss, valid_loss
```

```python
#******************************** RMS prop Model *************************************
def rmsprop_model(x_train, x_valid, y_train, y_valid, iteration_times, decay_rate, eps, alpha):

    train_loss = []
    valid_loss = []
    # initial params
    w = init_param(x_train.shape[1])  # initialize weight and bias
    cache = init_param(1, method='zero')  # initial with zeros
    for i in range(iteration_times):
        # Get Train & Test Data
        index = get_batch(x_train, batch_size=int(x_train.shape[0]/iteration_times))
        x_train_batch = x_train[index, :]
        y_train_batch = y_train[index]

        # Logistic Model & Train Loss
        y_pred = linear_model(x_train_batch, w)
        loss_t = loss_function(y_train_batch, y_pred, w)  # square loss function of train data
        train_loss.append(loss_t)

        # Logistic Model &  Validation loss
        y_valid_pred = linear_model(x_valid, w)
        loss_v = loss_function(y_valid, y_valid_pred, w)  # square loss function of validation data
        valid_loss.append(loss_v)


        w, cache = update_params_rmsprop(x_train_batch, y_train_batch, y_pred, w, cache, decay_rate, eps, alpha)

    return train_loss, valid_loss
```

```python
#****************************** ADA Delta Model **************************************************

def adadelta_model(x_train, x_valid, y_train, y_valid, iteration_times, decay_rate, eps):

    train_loss = []
    valid_loss = []
    # initial params
    w = init_param(x_train.shape[1])  # initialize weight and bias
    cum_grad = init_param(1, method='zero')  # initial with zero
    cum_u_w = init_param(1, method='zero')  # initial with zero
    u_w_list = [init_param(x_train.shape[1], method='zero')]

    for i in range(iteration_times):
        # Get Train & Test Data
        index = get_batch(x_train, batch_size=int(x_train.shape[0]/iteration_times))
        x_train_batch = x_train[index, :]
        y_train_batch = y_train[index]

        # Logistic Model & Train loss
        y_pred = linear_model(x_train_batch, w)
        loss_t = loss_function(y_train_batch, y_pred, w)  # square loss function of train data
        train_loss.append(loss_t)

        # Logistic Model  & Validation Loss
        y_valid_pred = linear_model(x_valid, w)
        loss_v = loss_function(y_valid, y_valid_pred, w)  # square loss function of validation data
        valid_loss.append(loss_v)

        w, cum_grad, cum_u_w = update_params_adadelta(x_train_batch, y_train_batch, y_pred, w, decay_rate, eps, cum_grad
                                                      u_w_list)
    return train_loss, valid_loss
```

```python
#draw the graph of train and validation
import matplotlib.pyplot as PL


print("**************************************************************** ")
print("This Result Linear Classification and Stochastic Gradient Descent ")
print("**************************************************************** ")
def plot_result(train_loss, valid_loss, fig_config):
    # train loss plt with different alpha
    for i in range(len(alpha)):
        PL.figure()
        PL.title('Training & Validation Loss with Learning Rate '+ str(alpha[i]))
        PL.plot(range(len(train_loss[i])), train_loss[i], linewidth=2.0,
                 color=fig_config['color'][0], label=fig_config['label'][0])
        PL.plot(range(len(valid_loss[i])), valid_loss[i], linewidth=2.0,
                 color=fig_config['color'][1], label=fig_config['label'][1])
        PL.xlabel('Iteration')
        PL.ylabel('Loss')
        PL.legend(fig_config['label'])
        PL.show()
        print('\n')
```

Conclusion:

In this experiment consist of two part, part one: logistic regression, Second part: linear classification implemented on larger data using SGD In addition to four different optimization methods (**NAG， RMSProp， AdaDelta and Adam**). Through this experiment, further understand the improved version of gradient descent. Through the experiment, I compared and understand the differences and relationships between Logistic regression and linear classification and further understood the principles of SVM and practice on larger data.