The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:

梁子豪

Student ID：2015306121630

Supervisor:

Qingyao Wu

Grade:

Undergraduate

December 15, 2017

Linear Regression, Linear Classification and Gradient Descent

Abstract—

I. Introduction

1. Compare and understand the difference between gradient descent and stochastic gradient descent.
2. Compare and understand the differences and relationships between Logistic regression and linear classification.
3. Further understand the principles of SVM and practice on larger data.

II. Methods and Theory

Logistic Regression and Stochastic Gradient Descent

1. Load the training set and validation set.
2. Initalize logistic regression model parameters, you can consider initalizing zeros, random numbers or normal distribution.
3. Select the loss function and calculate its derivation, find more detail in PPT.
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 LNAG , LRMSProp, LAdadelta and LAdam .
7. Repeate step 4 to 6 for several times, and **drawing graph of L and with the number of iterations.**

Linear Classification and Stochastic Gradient Descent

1. Load the training set and validation set.
2. Initalize SVM model parameters, you can consider initalizing zeros, random numbers or normal distribution.
3. Select the loss function and calculate its derivation, find more detail in PPT.
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 LNAG , LRMSProp, LAdadelta and LAdam .
7. Repeate step 4 to 6 for several times, and **drawing graph of L and with the number of iterations.**

III. EXPERIMENT

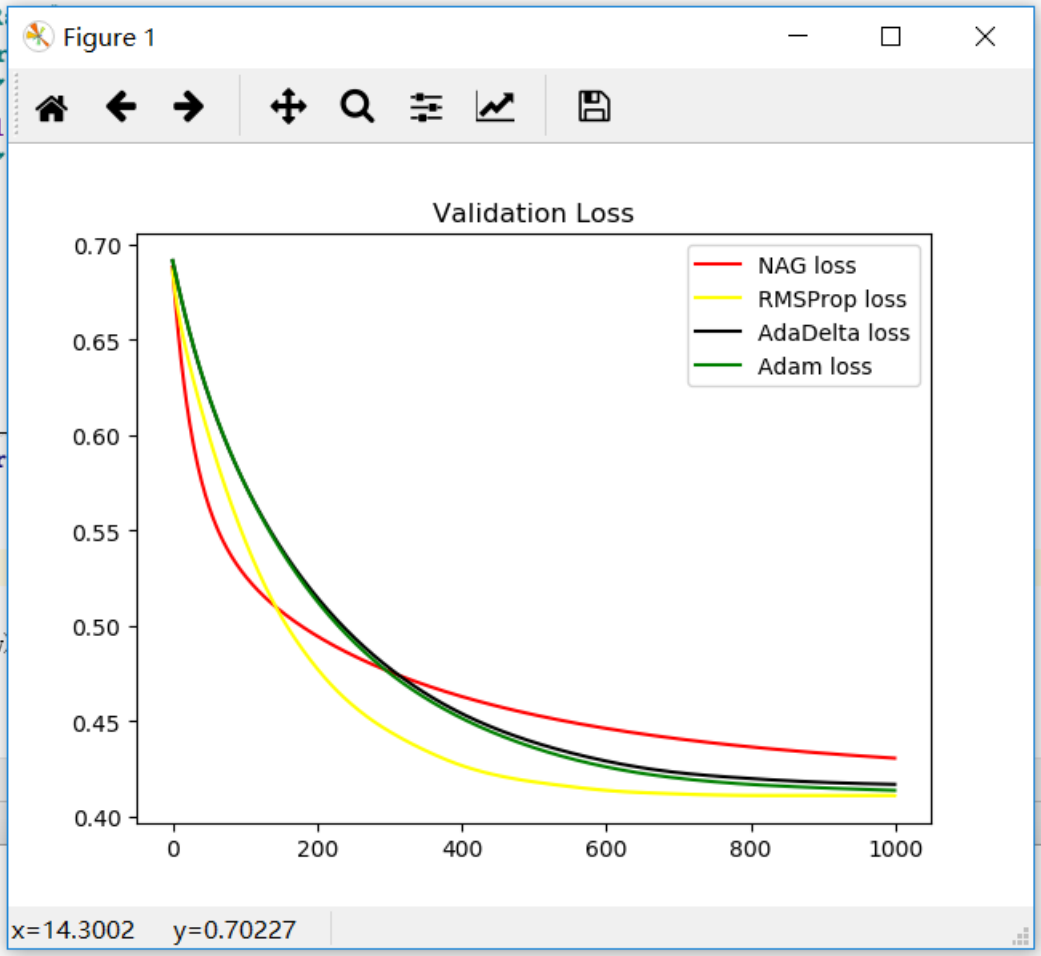
```
# write your code here
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt


def h(w, x):
    return 1 / (1 + np.exp(- x.dot(w)))


# 计算梯度，x 中包含 n 个样本，根据这 n 个样本计算梯度
def gradient(X, y, w, λ=0.1):
    row, col = w.shape
    grad = - y * X / (1 + np.exp(y * X.dot(w)))
    return np.mean(grad, axis=0).reshape(row, col) + w * λ
```

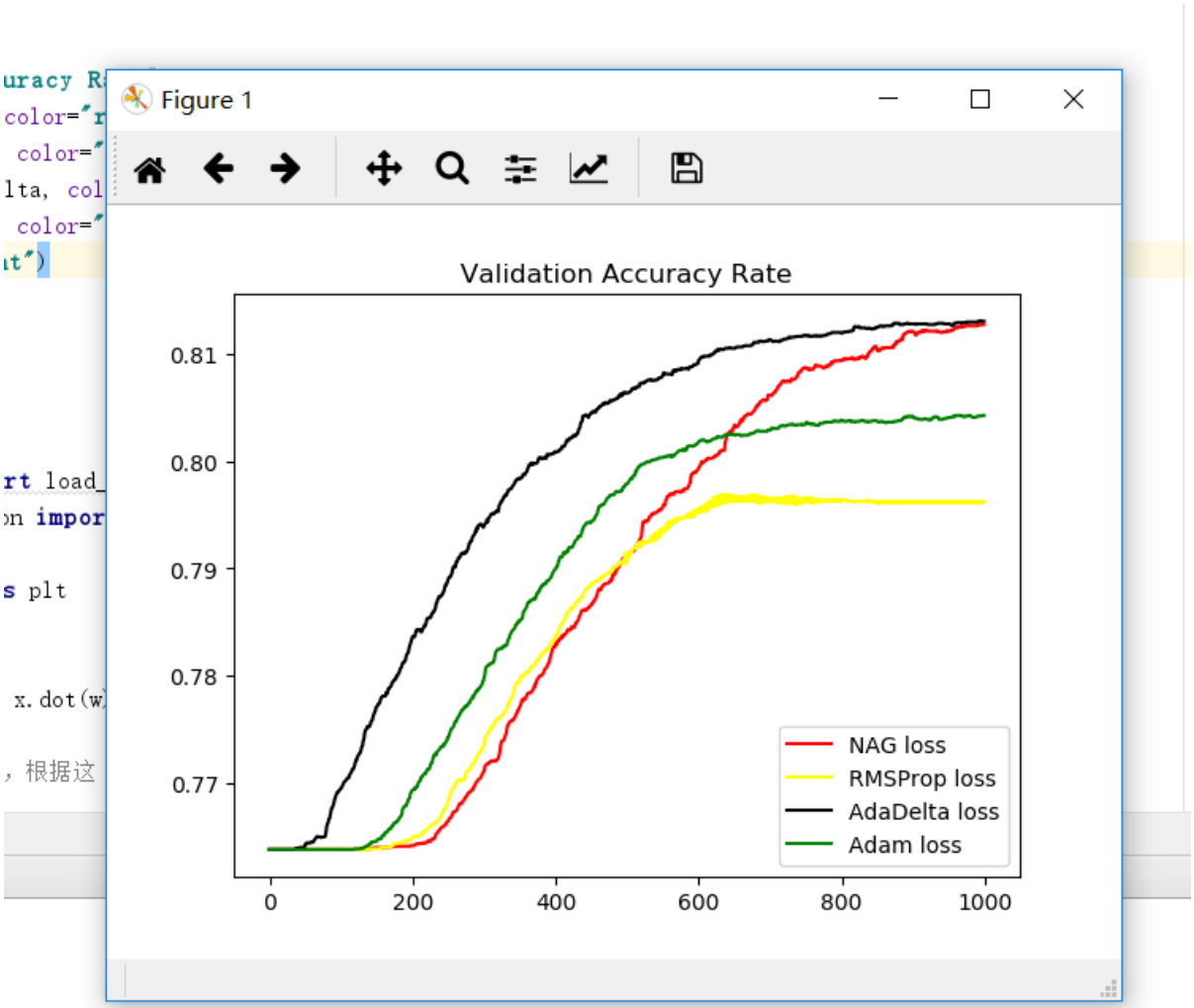
```
# 根据模型预测， θ 为阈值
def prediction(X, w, θ=0.5):
    row, col = X.shape
    p = np.ones((row, 1))
    b = h(w, X) < θ
    b = b.ravel()
    p[b, :] = -1
    return p


def Loss(X, y, w, λ=0.01):
    row, col = X.shape
    loss_mat = np.log(1 + np.exp(-y * X.dot(w)))
    return loss_mat.mean() + λ * 0.5 * w.T.dot(w) / row


# 计算分类正确率
def Rate(y_real, y_prediction):
    row, col = y_real.shape
    mat = np.zeros([row, 1])
    b = (y_real == y_prediction)
    b = b.ravel()
    mat[b, :] = 1
    return mat.sum() / row


# NAG, T是训练轮数， λ 为正则化参数
def NAG(X, y, X_test, y_test, T, batch, η=0.01, γ=0.01, λ=0.01):
    row, col = X.shape
    random = np.array(np.random.random_sample(batch) * row, dtype=int)
    x = X[random, :]
    y = y[random, :]
    w = np.zeros((col, 1))
    v = np.zeros((col, 1))
    loss = np.zeros(T)
    rate = np.zeros(T)
    for i in range(T):
        g = gradient(x, y, w - γ * v)
        v = γ * v + η * g
        w = w - v
        loss[i] = Loss(X_test, y_test, w)
        y_prediction = prediction(X_test, w)
        rate[i] = Rate(y_test, y_prediction)
    return loss, rate


def RMSProp(X, y, X_test, y_test, T, batch, η=0.0005, γ=0.9):
    row, col = X.shape
    random = np.array(np.random.random_sample(batch) * row, dtype=int)
    x = X[random, :]
    y = y[random, :]
```

```
    w = np.zeros((col, 1))
    G = np.zeros((col, 1))
    e = 1 * 10 ** (-8)
    loss = np.zeros(T)
    rate = np.zeros(T)
    for i in range(T):
        g = gradient(x, y, w)
        G = γ * G + (1 - γ) * g * g
        w = w - (η / np.sqrt(G + e)) * g
        # w = w - η*g
        loss[i] = Loss(X_test, y_test, w)
        y_prediction = prediction(X_test, w)
        rate[i] = Rate(y_test, y_prediction)
    return loss, rate


def AdaDelta(X, y, X_test, y_test, T, batch, η=0.0005, γ=0.95):
    row, col = X.shape
    random = np.array(np.random.random_sample(batch) * row, dtype=int)
    x = X[random, :]
    y = y[random, :]
    w = np.zeros((col, 1))
    G = np.zeros((col, 1))
    delta = np.zeros((col, 1))
    e = 1 * 10 ** (-8)
    loss = np.zeros(T)
    rate = np.zeros(T)
    for i in range(T):
        g = gradient(x, y, w)
        G = γ * G + (1 - γ) * g * g
        w_delta = - np.sqrt(delta + e) / np.sqrt(G + e) * g
        w = w + w_delta
        delta = γ * delta + (1 - γ) * w_delta * w_delta
        loss[i] = Loss(X_test, y_test, w)
        y_prediction = prediction(X_test, w)
        rate[i] = Rate(y_test, y_prediction)
    return loss, rate


def Adam(X, y, X_test, y_test, T, batch, η=0.001, γ=0.999, β=0.9):
    row, col = X.shape
    random = np.array(np.random.random_sample(batch) * row, dtype=int)
    x = X[random, :]
    y = y[random, :]
    w = np.zeros((col, 1))
    m = np.zeros((col, 1))
    G = np.zeros((col, 1))
    delta = np.zeros((col, 1))
    e = 1 * 10 ** (-8)
    loss = np.zeros(T)
    rate = np.zeros(T)
```

```
    for i in range(T):
        g = gradient(x, y, w)
        m = β * m + (1 - β) * g
        G = γ * G + (1 - γ) * g * g
        α = η * np.sqrt(1 - γ ** i) / (1 - β ** i)
        w = w - α * m / np.sqrt(G + e)
        loss[i] = Loss(X_test, y_test, w)
        y_prediction = prediction(X_test, w)
        rate[i] = Rate(y_test, y_prediction)
    return loss, rate


# 从文件中加载数据并对数据进行预处理
def load(file_name, features=123):
    X, y = load_svmlight_file(file_name, n_features=features)
    row, col = X.shape
    X = X.toarray()
    X = np.column_stack((X, np.ones((row, 1))))
    y = y.reshape(row, 1)
    return X, y


# 加载数据并分离测试集和验证集
X_train, y_train = load('a9a.t')
X_validation, y_validation = load('a9a.t')

# 初始化数据, batch 是批量梯度下降的批量数,
iteration = 1000
batch = 500

loss_NAG, rate_NAG = NAG(X_train, y_train, X_validation, y_validation, batch=batch, T=iteration)
loss_Prop, rate_Prop = RMSProp(X_train, y_train, X_validation, y_validation, batch=batch, T=iteration)
loss_AdaDelta, rate_AdaDelta = AdaDelta(X_train, y_train, X_validation, y_validation, batch=batch, T=iteration)
loss_Adam, rate_Adam = AdaDelta(X_train, y_train, X_validation, y_validation, batch=batch, T=iteration)

# 绘图
x_axis = range(iteration)

plt.figure()
plt.title("Validation Loss")
plt.plot(x_axis, loss_NAG, color="blue", label="NAG loss")
plt.plot(x_axis, loss_Prop, color="red", label="RMSProp loss")
plt.plot(x_axis, loss_AdaDelta, color="yellow", label="AdaDelta loss")
plt.plot(x_axis, loss_Adam, color="green", label="Adam loss")
plt.legend(loc="upper right")
plt.show()

plt.figure()
plt.title("Validation Accuracy Rate")
plt.plot(x_axis, rate_NAG, color="blue", label="NAG loss")
plt.plot(x_axis, rate_Prop, color="red", label="RMSProp loss")
```

```
plt.plot(x_axis, rate_AdaDelta, color="yellow", label="AdaDelta loss")
plt.plot(x_axis, rate_Adam, color="green", label="Adam loss")
plt.legend(loc="lower right")
plt.show()
```






```
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

def h(w, x):
    return 1/(1 + np.exp(- x.dot(w)))

# 计算梯度，x 中包含 n 个样本，根据这 n 个样本计算梯度
def gradient(X, y, w):
    grad = y * X /(1 + np.exp(y * X.dot(w)))
    return np.mean(grad, axis=0)

# 根据模型预测， θ 为阈值
def prediction(X, w, θ):
    row, col = X.shape
    p = np.zeros((row, 1))
    b = h(w, X) < θ
    b = b.ravel()
```

```
    p[b,:] = -1
    return p

def Loss(X, y, w):
    loss_mat = - np.log(1 + np.exp(-y * X.dot(w)))
    return loss_mat.mean()

# 计算分类正确率
def rate(y_real, y_prediction):
    row, col = y_real.shape
    mat = np.zeros([row, 1])
    b = (y_real == y_prediction)
    b = b.ravel()
    mat[b,:] = 1
    return mat.sum()/row

# 加载数据并分离测试集和验证集
X, y = load_svmlight_file('a9a.txt')
row, col = X.shape
X = X.toarray()
y = y.reshape(row, 1)
#X = np.column_stack((X, np.ones((row, 1))))
X_train, X_validation, y_train, y_validation = train_test_split(X, y , test_size=0.33, random_state=42 )

# 初始化数据, μ 为动量因子, η 为学习率, θ 为阈值
row, col = X_train.shape
μ = 1
η = 0.5
θ = 0.5
T = 2000
min_batch = 1
v = np.zeros((col,1))
w = np.zeros((col,1))

# 训练集和测试集 Loss, 准确率 rate
loss_train = np.zeros(T)
rate_train = np.zeros(T)
loss_validation = np.zeros(T)
rate_validation = np.zeros(T)

 # 训练模型
for i in range(T):
    random = np.array(np.random.random_sample(min_batch)*row,dtype=int)
    x = X_train[random,:]
    y = y_train[random,:]
#         print(y)
#          g = gradient(w+μ*v, x, y).reshape(col,1)
#          v = μ* v - η*g
#          w = w + v
    g = gradient(x, y, w).reshape(col,1)
    w = w - η*g
```

```
    prediction_train = prediction(X_train, w, θ)
    loss_train[i] = Loss(X_train, y_train, w)
    rate_train[i] = rate(y_train, prediction_train)
    prediction_validation = prediction(X_validation, w, θ)
    loss_validation[i] = Loss(X_validation, y_validation, w)
    rate_validation[i] = rate(y_validation, prediction_validation)

# 绘图
plt.figure()
x_axis = range(T)
plt.title("Trainint and Validation Loss in Linear Classification")
plt.plot(x_axis, loss_train,color="blue",linestyle="-",label="Training loss")
plt.plot(x_axis, loss_validation,color="red",linestyle="-",label="Validation loss")
plt.legend(loc="upper right")
plt.show()

plt.figure()
plt.title("Trainint and Validation Accuracy Rate")
plt.plot(x_axis, rate_train,color="green",linestyle="-",label="Training rate")
plt.plot(x_axis, rate_validation,color="yellow",linestyle="-",label="Validation rate")
plt.legend(loc="upper right")
plt.show()
```

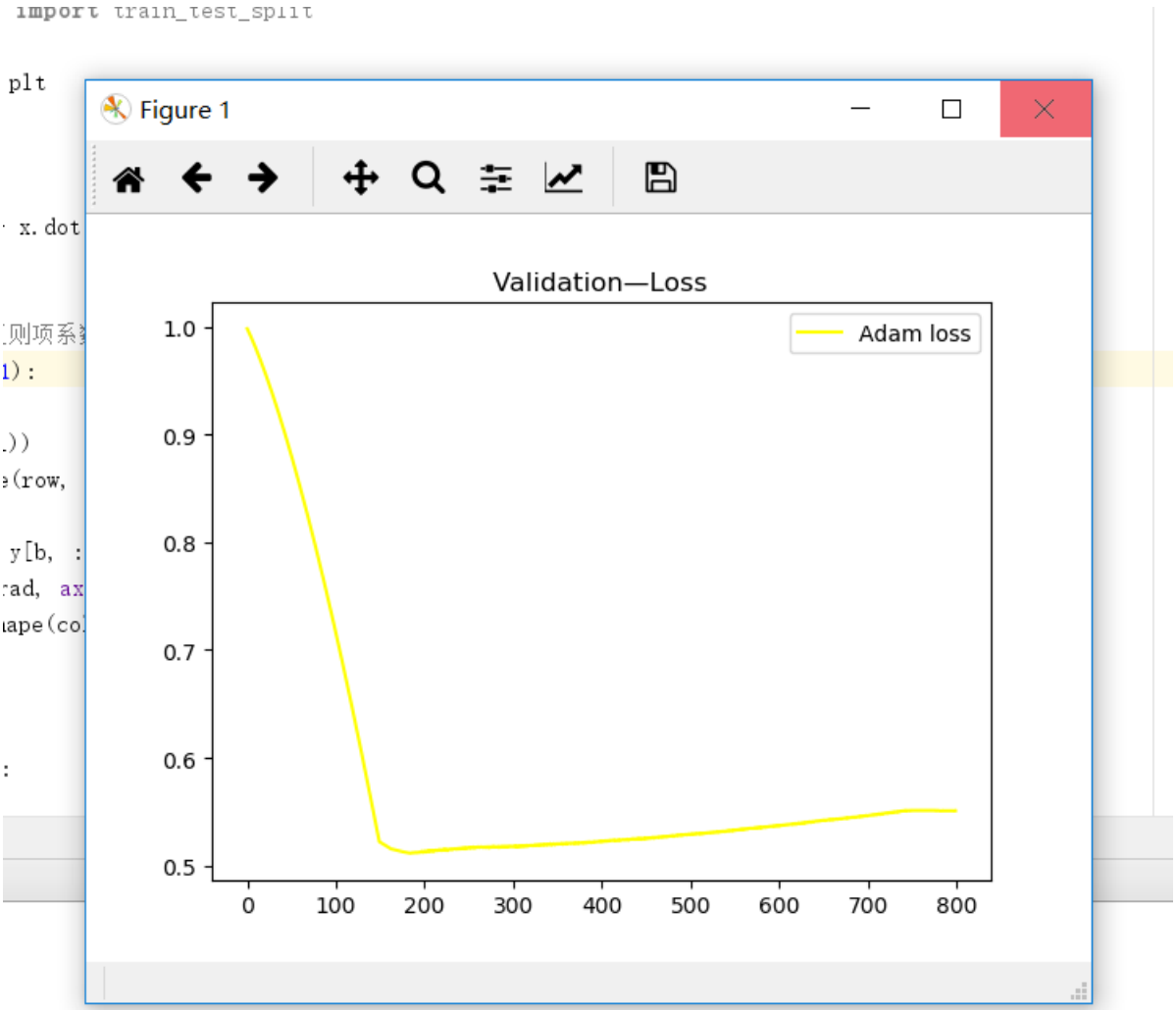
```
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt


def h(w, x):
    return 1 / (1 + np.exp(- x.dot(w)))


# 计算 hinge loss 的梯度, λ 为正则项系数
def gradient(X, y, w, λ=0.01):
    row, col = X.shape
    grad = np.zeros((row, col))
    b = y * (X.dot(w).reshape(row, 1)) < 1
    b = b.ravel()
    grad[b, :] = - X[b, :] * y[b, :]
    grad_average = np.mean(grad, axis=0)
    return grad_average.reshape(col, 1) + λ * w


# 用所得模型进行分类, θ 为阈值
def prediction(X, w, θ=0.5):
    row, col = X.shape
    prediction_mat = np.ones((row, 1))
```

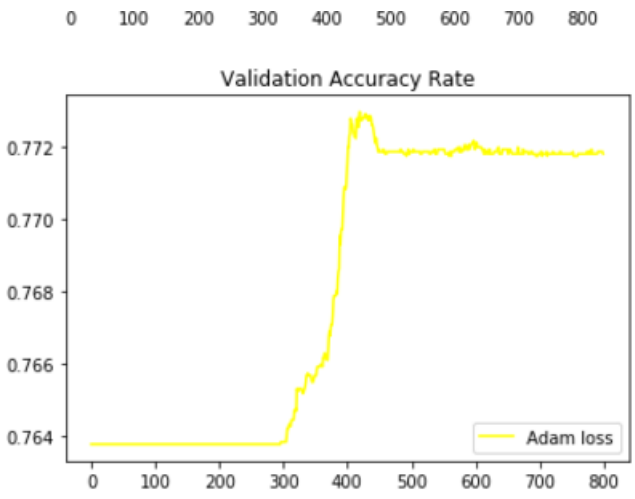
```
    b = X.dot(w) < θ
    b = b.ravel()
    prediction_mat[b, :] = -1
    return prediction_mat


def Loss(X, y, w, λ=0.01):
    row, col = X.shape
    loss = np.zeros((row, 1))
    b = y * (X.dot(w).reshape(row, 1)) < 1
    b = b.ravel()
    loss[b, :] = 1 - y[b, :] * X[b, :].dot(w)
    return loss.mean() + λ * 0.5 * w.T.dot(w)


# 计算分类正确率
def Rate(y_real, y_prediction):
    row, col = y_real.shape
    mat = np.zeros([row, 1])
    b = (y_real == y_prediction)
    b = b.ravel()
    mat[b, :] = 1
    return mat.sum() / row


# NAG, T是训练轮数, λ为正则化参数
def NAG(X, y, X_test, y_test, T, batch, η=0.07, γ=0.01, λ=0.003):
    row, col = X.shape
    random = np.array(np.random.random_sample(batch) * row, dtype=int)
    x = X[random, :]
    y = y[random, :]
    w = np.zeros((col, 1))
    v = np.zeros((col, 1))
    loss = np.zeros(T)
    rate = np.zeros(T)
    for i in range(T):
        g = gradient(x, y, w - γ * v, λ=λ)
        v = γ * v + η * g
        w = w - v
        loss[i] = Loss(X_test, y_test, w, λ=λ)
        y_prediction = prediction(X_test, w)
        rate[i] = Rate(y_test, y_prediction)
    return loss, rate


def RMSProp(X, y, X_test, y_test, T, batch, η=0.0008, γ=0.9, λ=0.00005):
    row, col = X.shape
    random = np.array(np.random.random_sample(batch) * row, dtype=int)
    x = X[random, :]
    y = y[random, :]
    w = np.zeros((col, 1))
```

```
    G = np.zeros((col, 1))
    e = 1 * 10 ** (-8)
    loss = np.zeros(T)
    rate = np.zeros(T)
    for i in range(T):
        g = gradient(x, y, w, λ=λ)
        G = γ * G + (1 - γ) * g * g
        w = w - (η / np.sqrt(G + e)) * g
        loss[i] = Loss(X_test, y_test, w, λ=λ)
        y_prediction = prediction(X_test, w)
        rate[i] = Rate(y_test, y_prediction)
    return loss, rate


def AdaDelta(X, y, X_test, y_test, T, batch, γ=0.80):
    row, col = X.shape
    random = np.array(np.random.random_sample(batch) * row, dtype=int)
    x = X[random, :]
    y = y[random, :]
    w = np.zeros((col, 1))
    G = np.zeros((col, 1))
    delta = np.zeros((col, 1))
    e = 1 * 10 ** (-8)
    loss = np.zeros(T)
    rate = np.zeros(T)
    for i in range(T):
        g = gradient(x, y, w)
        G = γ * G + (1 - γ) * g * g
        w_delta = - np.sqrt(delta + e) / np.sqrt(G + e) * g
        w = w + w_delta
        delta = γ * delta + (1 - γ) * w_delta * w_delta
        loss[i] = Loss(X_test, y_test, w)
        y_prediction = prediction(X_test, w)
        rate[i] = Rate(y_test, y_prediction)
    return loss, rate


def Adam(X, y, X_test, y_test, T, batch, η=0.1, γ=0.999, β=0.9):
    row, col = X.shape
    random = np.array(np.random.random_sample(batch) * row, dtype=int)
    x = X[random, :]
    y = y[random, :]
    w = np.zeros((col, 1))
    m = np.zeros((col, 1))
    G = np.zeros((col, 1))
    delta = np.zeros((col, 1))
    e = 1 * 10 ** (-8)
    loss = np.zeros(T)
    rate = np.zeros(T)
    for i in range(T):
        # ηt = η/np.sqrt(t)
```

```
        g = gradient(x, y, w)
        m = β * m + (1 - β) * g
        G = γ * G + (1 - γ) * g * g
        α = η * np.sqrt(1 - γ ** i) / (1 - β ** i)
        w = w - α * m / np.sqrt(G + e)
        loss[i] = Loss(X_test, y_test, w)
        y_prediction = prediction(X_test, w)
        rate[i] = Rate(y_test, y_prediction)
    return loss, rate


def load(file_name, features=123):
    X, y = load_svmlight_file(file_name, n_features=features)
    row, col = X.shape
    X = X.toarray()
    X = np.column_stack((X, np.ones((row, 1))))
    y = y.reshape(row, 1)
    return X, y


# 加载数据并分离测试集和验证集
X_train, y_train = load('a9a.t')
X_validation, y_validation = load('a9a.t')

# 初始化数据, batch 是批量梯度下降的批量数,
iteration = 800
batch = 10

# loss_NAG, rate_NAG = NAG(X_train, y_train, X_validation, y_validation, batch=batch,T = iteration)
# loss_Prop, rate_Prop = RMSProp(X_train, y_train, X_validation, y_validation, batch=batch,T = iteration)
# loss_AdaDelta, rate_AdaDelta = AdaDelta(X_train, y_train, X_validation, y_validation, batch=batch,T = iteration)
loss_Adam, rate_Adam = AdaDelta(X_train, y_train, X_validation, y_validation, batch=batch, T=iteration)

# 绘图
x_axis = range(iteration)

plt.figure()
plt.title("Validation—Loss")
# plt.plot(x_axis, loss_NAG,color="blue",label="NAG loss")
# plt.plot(x_axis, loss_Prop,color="red",label="RMSProp loss")
# plt.plot(x_axis, loss_AdaDelta,color="yellow",label="AdaDelta loss")
plt.plot(x_axis, loss_Adam, color="yellow", label="Adam loss")
plt.legend(loc="upper right")
plt.show()

plt.figure()
plt.title("Validation—Accuracy Rate")
# plt.plot(x_axis, rate_NAG,color="blue",label="NAG loss")
# plt.plot(x_axis, rate_Prop,color="red",label="RMSProp loss")
# plt.plot(x_axis, rate_AdaDelta,color="yellow",label="AdaDelta loss")
plt.plot(x_axis, rate_Adam, color="yellow", label="Adam loss")
```

```
plt.legend(loc="lower right")
plt.show()
```





IV. CONCLUSION

This experiment is a little difficult, but it is intersting.

Although aiming at different tasks, different optimization algorithms should be tried. But adjusting the learning rate during the training process is necessary for complex optimization goals, such as an epoch times 0.5. No one knows what happens in the process of training complex models, and SGD is the most promising.