HW2

Problem 2

(2)

It takes 6158 steps to converge.

(4)

My while loops will stop either

or steps more than 20000. The steps result is always equal 20000. It violates the concept that SGD converges faster. Therefore, I change the value of to be larger. The following table is my result.

|  |  |  |
| --- | --- | --- |
|  | Gradient Descent (steps) | Stochastic Gradient Descent (steps) |
|  | 137 | 70 |
|  | 930 | 817 |
|  | 1935 | 3544 |
|  | 2987 | 10850 |

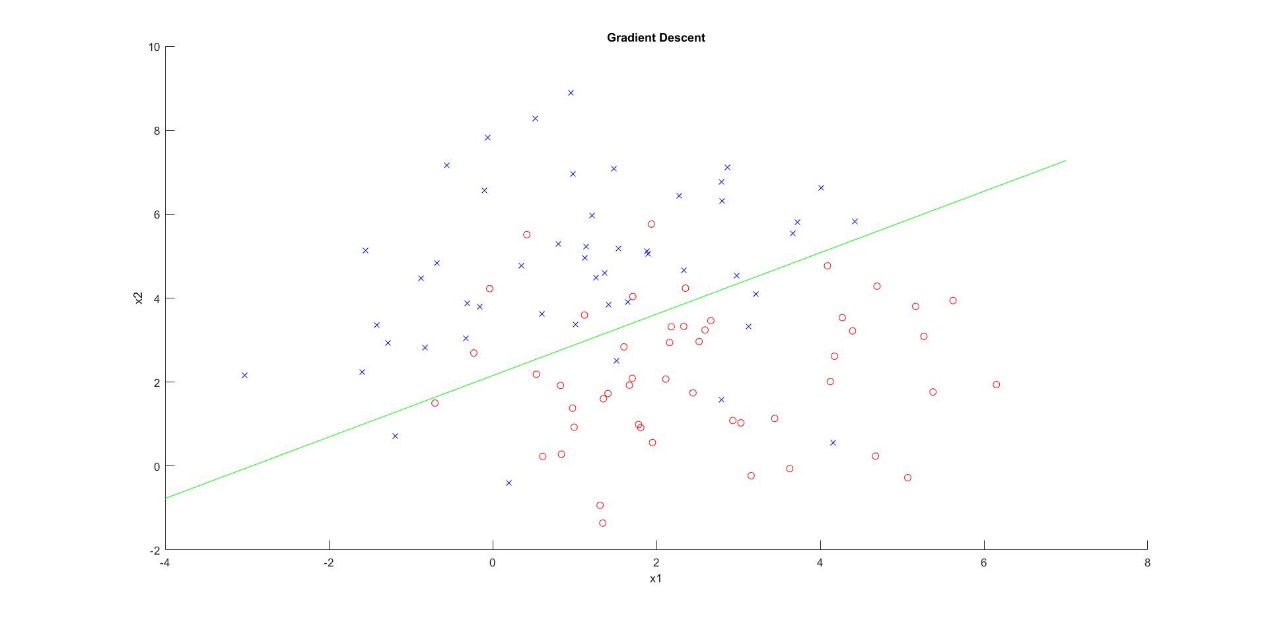
At the beginning, the SGD converges faster than GD. However, when it comes to smaller , SGD seems to has more oscillation.

(6)

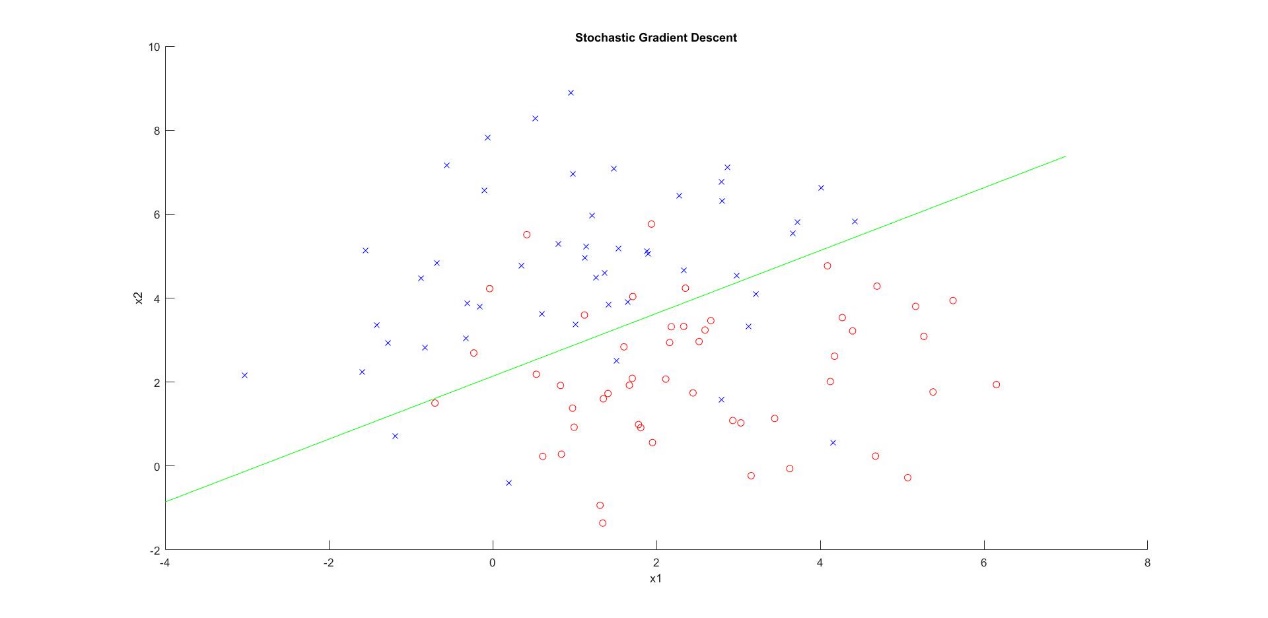
It takes only 7 steps to converge. Compare to the other 2 methods, Newton’s Method converge much faster.

(7)

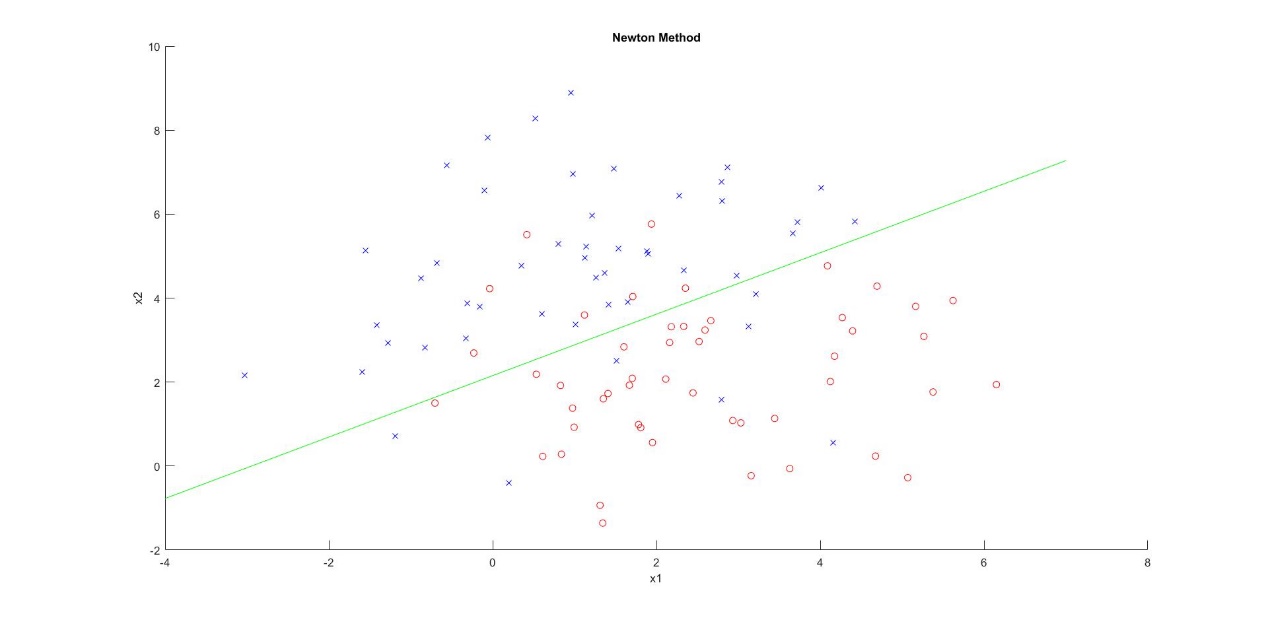
Gradient Descent



Stochastic Gradient Descent



Newton’s Method



Appendix

Logistic Regression

clear, clc, close all;

load q1x.dat;

load q1y.dat;

nIters = 20000;

epsilon = 1e-8;

learning\_rate = 0.001;

x = [ones(size(q1x,1), 1), q1x];

y = q1y;

%% Gradient Descent

[w, steps] = GD( x, y, learning\_rate, nIters ,epsilon);

drawResult( x, y, w, 'Gradient Descent');

display(['GD steps:', num2str(steps)]);

%% Stochastic Gradient Descent

learning\_rate = 1;

[w, steps] = SGD( x, y, learning\_rate, nIters ,epsilon);

drawResult( x, y, w, 'Stochastic Gradient Descent');

display(['SGD steps:', num2str(steps)]);

%% Newton's Method

[w, steps] = Newton( x, y, nIters ,epsilon);

drawResult( x, y, w, 'Newton Method');

display(['Newton Method steps:', num2str(steps)]);

function [ w, steps] = GD( x, y, learning\_rate, nIters, epsilon)

[m,~] = size(x);

w = zeros(3, 1);

logistic = @(x,w,m)ones(m,1)./(ones(m,1)+exp(-x\*w));

loss = sum(y-logistic(x, w, m));

pre\_loss = 0;

steps=0;

while abs(loss - pre\_loss) > epsilon && steps < nIters

pre\_loss = loss;

h = logistic(x, w, m);

w = w - learning\_rate\* x' \* (h - y);

loss = sum(y-logistic(x, w, m));

steps = steps + 1;

end

end

function [w, steps] = SGD( x, y, r0, nIters, epsilon)

[m,~] = size(x);

w = zeros(3, 1);

logistic = @(x,w,m)ones(m,1)./(ones(m,1)+exp(-x\*w));

loss = sum(y-logistic(x, w, m));

pre\_loss = 0;

steps=0;

while abs(loss - pre\_loss) > epsilon && steps < nIters

pre\_loss = loss;

for j =1:m

i = ceil(99 \* rand(1));

learning\_rate = r0 / ((1+r0 \* (steps\*m+j) )^0.75);

xi = x(i,:);

h = logistic(xi, w, 1);

w = w - learning\_rate \* xi' \* (h - y(i));

end

loss = sum(y-logistic(x, w, m));

steps = steps + 1;

end

end

function [w, steps] = Newton( x, y, nIters, epsilon)

[m,~] = size(x);

w = zeros(3, 1);

logistic = @(x,w,m)ones(m,1)./(ones(m,1)+exp(-x\*w));

loss = sum(y-logistic(x, w, m));

pre\_loss = 0;

steps=0;

while abs(loss - pre\_loss) > epsilon && steps < nIters

pre\_loss = loss;

h = logistic(x, w, m);

A = diag(h.\*(1-h));

H = x' \* A \* x;

w = w - H\( x' \* (h - y));

loss = sum(y-logistic(x, w, m));

steps = steps + 1;

end

end

Problem 3

(1)

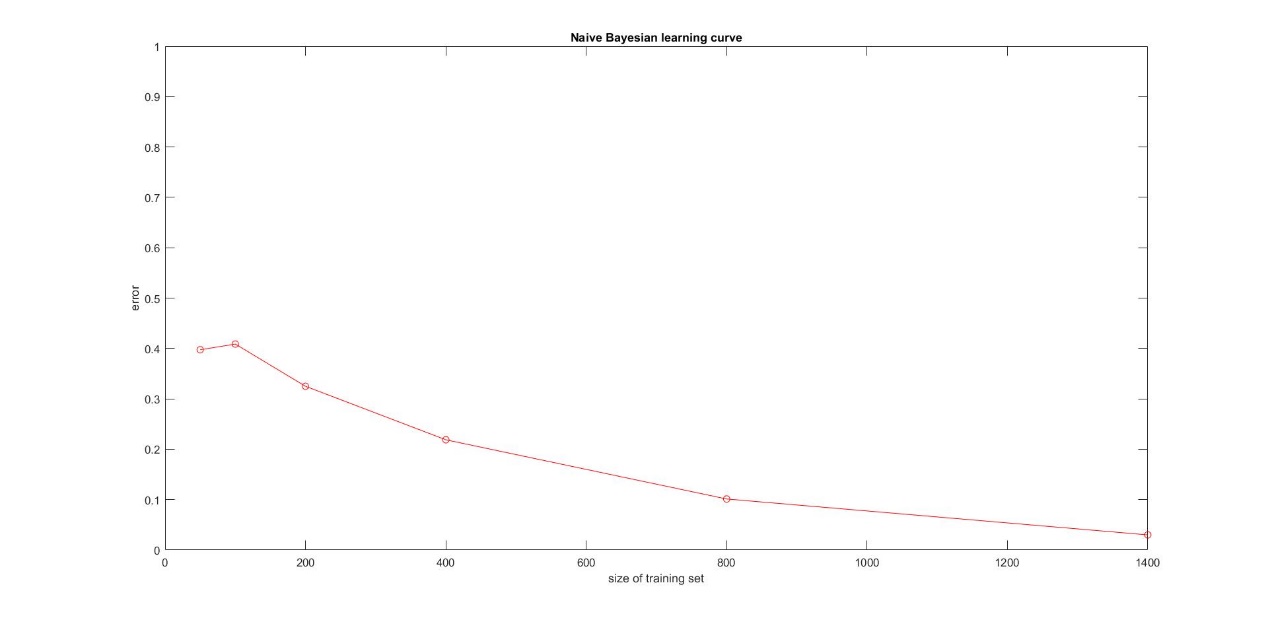
The error rate of Naïve Bayesian in the spam mail classification is 1%

(2)

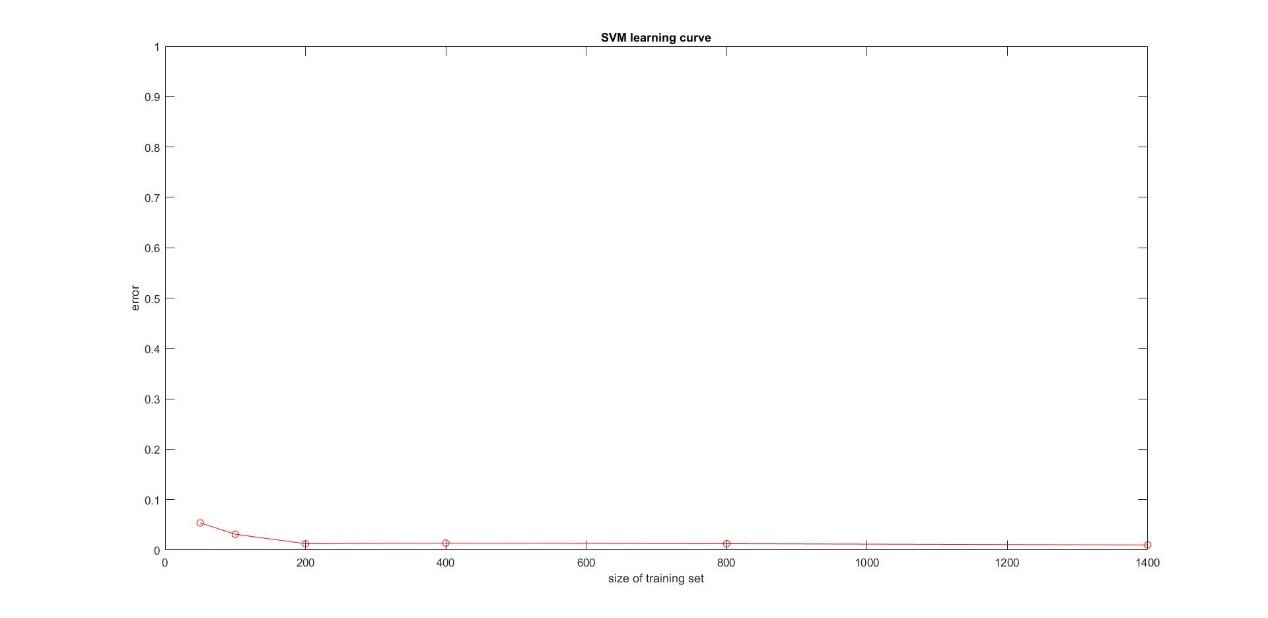
The corresponding words are httpaddr, spam, unsubscribe, cent, and valet.

(3)

The data set with size 1400 has lowest testing error 3%. It is reasonable to have a lowest generalization error using the largest training data. Because Naïve Bayesian uses data to estimate the probability.



(4)



(5)

The testing errors are generally lower than the testing error of Naïve Bayesian. The reason why SVM don’t need many data to train is that SVM use only support vectors, which normally are few, to maximize the classification margin.

Appendix

Spam

clear, clc;

file = {'50', '100', '200', '400', '800', '1400'};

%% read data from file and save in mat

readWord('SPARSE.TRAIN', 'train');

for i = 1: length(file)

fileName = ['SPARSE.TRAIN.', file{i}];

saveName = ['train', file{i}];

readWord(fileName, saveName);

end

readWord('SPARSE.TEST', 'test');

%% Naive Bayesian

load('data\train.mat');

xtrain = x;

ytrain = y;

clear x y;

load('data\test.mat');

xtest = x;

ytest = y;

clear x y;

ypredict = NB(xtrain, ytrain, xtest);

error = sum(abs(ypredict-ytest)/2) / length(ytest);

disp(['error rate: ', num2str(100\*error), '%']);

%% most indicative tokens of spam

[B, I] = tokens( xtrain, ytrain);

%% learning curve

train\_size = [50, 100, 200, 400, 800, 1400];

gernalization\_error = learningCurve(file, xtest, ytest);

figure(1)

plot(train\_size, gernalization\_error, 'ro-');

xlabel('size of training set');

ylabel('error');

ylim([0, 1]);

title('Naive Bayesian learning curve');

%% SVM

gernalization\_error = learningCurveSVM(file, xtest, ytest);

figure(1)

plot(train\_size, gernalization\_error, 'ro-');

xlabel('size of training set');

ylabel('error');

ylim([0, 1]);

title('SVM learning curve');

function readWord(fileName, saveName )

%READWORD Summary of this function goes here

% Detailed explanation goes here

row = 1;

col = 1448;

fid = fopen(fileName);

document = 1;

tline = fgetl(fid);

y = [];

x = sparse(row, col);

while ischar(tline)

C = strsplit(tline);

y = [y;str2double(C{1})];

for i=2:size(C,2)

element = strsplit(C{i},':');

x(document, str2double(element{1})) = str2double(element{2});

end

%disp(tline);

tline = fgetl(fid);

document = document + 1;

end

fclose(fid);

save(['data\', saveName, '.mat'],'x','y');

end

function predict = NB( xtrain, ytrain, xtest)

% the index of spam mail

indexSpam = find(ytrain==1);

% P(D|spam): probability of a word appear in a spam mail

wordBagSpam = sum(sign(xtrain(indexSpam,:)),1) ./ length(indexSpam);

% P(D): probability of a word appear in mails

wordBag = sum(sign(xtrain), 1) ./ length(ytrain);

% the index of words appear in mails

indexwithWord = find(wordBag~=0);

% P(spam): probability of spam mail

probOfSpam = length(indexSpam) / length(ytrain);

predict = zeros(size(xtest, 1), 1);

for i=1:size(xtest, 1)

prob = probOfSpam;

for j=indexwithWord

if xtest(i,j)~=0

% P(Di|spam)/P(Di): probability of a word appear in spam mail

prob = prob \* wordBagSpam(j) / wordBag(j);

else

% (1-P(Di|spam))/P(Di): probability of a word not appear in spam mail

prob = prob \* (1 - wordBagSpam(j)) / wordBag(j);

end

end

if prob > 0.5

predict(i) = 1;

else

predict(i) = -1;

end

end

end

function [ B, I ] = tokens( xtrain, ytrain)

col = size(xtrain, 2);

indexSpam = find(ytrain==1);

indexNotSpam = find(ytrain==-1);

indicator = zeros(2, col);

for j=1:col

indicator(1, j) = 1 + sum(xtrain(indexSpam, j));

indicator(2, j) = 1 + sum(xtrain(indexNotSpam, j));

end

[B, I] = sort(log(indicator(1,:)./ indicator(2,:)), 'descend');

B = B(1:5);

I = I(1:5);

end

function [ gernalization\_error ] = learningCurve(file, xtest, ytest)

gernalization\_error = zeros(length(file),1);

for i=1:length(file)

load(['data\train', file{i}, '.mat']);

xtrain = x;

ytrain = y;

clear x y;

predict = NB( xtrain, ytrain, xtest);

error = sum(abs(predict-ytest)/2) / length(ytest);

gernalization\_error(i) = error;

disp(['error rate of file ', file{i} ': ', num2str(100\*error), '%']);

end

end

function [ gernalization\_error ] = learningCurveSVM(file, xtest, ytest)

gernalization\_error = zeros(length(file),1);

for i=1:length(file)

load(['data\train', file{i}, '.mat']);

xtrain = x;

ytrain = y;

clear x y;

model = svmlib.matlab.train(ytrain, xtrain ,['liblinear\_options', 'row']);

[~, accuracy, ~] = svmlib.matlab.predict(ytest, xtest, model,...

['liblinear\_options', 'col']);

gernalization\_error(i) = 1-accuracy(1)/100;

disp(['error rate of file ', file{i} ': ', num2str(100\*gernalization\_error(i)), '%']);

end

end