Machine Learning EECS 545 HW2

Chien-Wei Lin

1.

(1)

A is PSD iff $\forall \alpha \in \mathbb{R}^n$, $\alpha^T A \alpha \geq 0$

We can rewrite $\alpha = \sum_{i=1}^{n} a_i v_i$, where v_i is the eigenvector of A

$$\therefore \quad 0 \leq \alpha^T A \alpha = \left(\sum_{i=1}^{n} a_i v_i\right)^T A \left(\sum_{i=1}^{n} a_i v_i\right) = \sum_{i=1}^{n} \lambda_i a_i^2 \|v_i\|_2^2 \quad \text{iff} \quad \lambda_i \geq 0$$

$\Rightarrow$ A is PSD iff $\lambda_i \geq 0$

(2)

Same as (1)

A is PD iff $\forall \alpha = \sum_{i=1}^{n} a_i v_i \in \mathbb{R}^n$, where v_i is eigenvector of A, $\alpha^T A \alpha > 0$

$$\text{iff} \quad 0 < \left(\sum_{i=1}^{n} a_i v_i\right)^T A \left(\sum_{i=1}^{n} a_i v_i\right) = \sum_{i=1}^{n} \lambda_i a_i^2 \|v_i\|_2^2 \quad \text{iff} \quad \lambda_i > 0$$

$\Rightarrow$ A is PD iff $\lambda_i > 0$

2.

(1)

$$h(x) = \frac{1}{1+e^{-w^T x}} = \frac{e^{w^T x}}{e^{w^T x}+1}$$

$$\Rightarrow l_i(w) = -y_i \log h(x_i) - (1-y_i)\log(1-h(x_i)) = -y_i w^T x_i + \cancel{y_i \log(e^{w^T x_i}+1)} + \log(e^{w^T x_i}+1) - \cancel{y_i \log(e^{w^T x_i}+1)}$$

$$= -y_i w^T x_i + \log(e^{w^T x_i}+1)$$

$$\Rightarrow \frac{\partial l_i(w)}{\partial w} = -y_i x_i + \frac{e^{w^T x_i}}{e^{w^T x_i}+1} x_i = -y_i x_i + h(x_i) x_i$$

$$\Rightarrow \nabla l(w) = \sum_{i=1}^{n} -y_i x_i + h(x_i) x_i = X^T\left(-y + h(x)\right)$$

(3) From (1)

$$\nabla l_i(w) = -y_i x_i + h(x_i) x_i = x_i [h(x_i) - y_i]$$

HW2

Problem 2

(2)

$$w = \begin{bmatrix} -1.8492 \\ -0.6281 \\ 0.8585 \end{bmatrix}$$

It takes 6158 steps to converge.

(4)

$$w = \begin{bmatrix} -1.7762 \\ -0.6288 \\ 0.8442 \end{bmatrix}$$

My while loops will stop either

$$\left| l(w)^{(t+1)} - l(w)^{t} \right| < 10^{-8}$$

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 ε to be larger. The following table is my result.

ε	Gradient Descent (steps)	Stochastic Gradient Descent (steps)
10^{-2}	137	70
10^{-3}	930	817
10^{-4}	1935	3544
10^{-5}	2987	10850

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

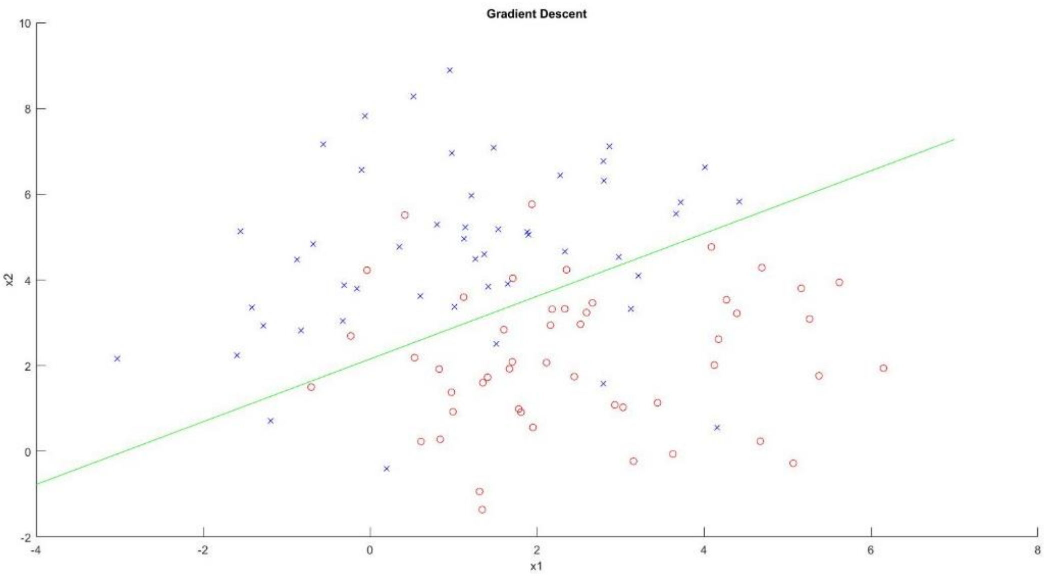
(6)

$$w = \begin{bmatrix} -1.8492 \\ -0.6281 \\ 0.8585 \end{bmatrix}$$

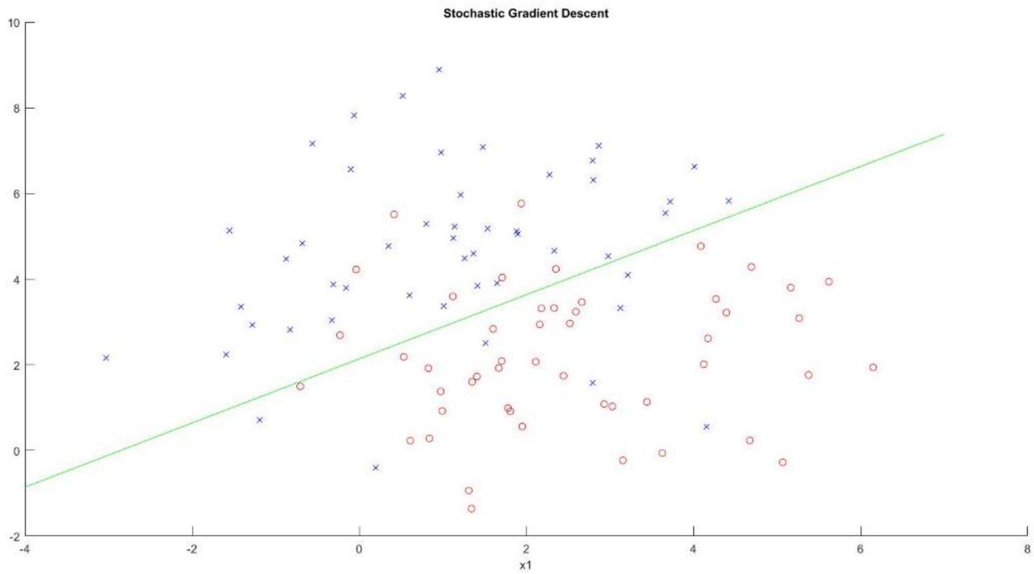
It takes only 7 steps to converge. Compare to the other 2 methods, Newton's Method converge much faster.

(7)

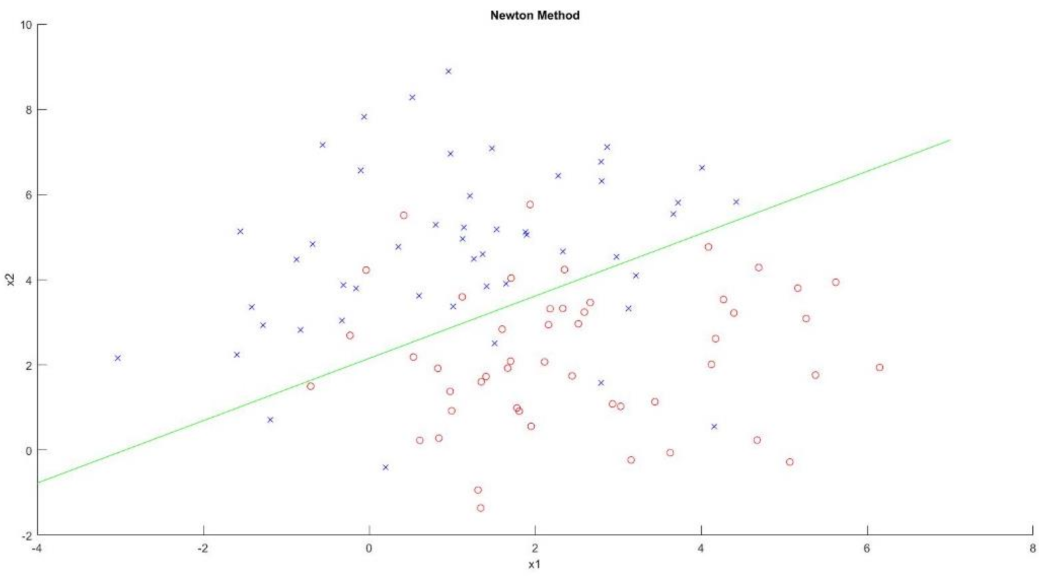
Gradient Descent



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
```

(5)

$$\underline{H} = \nabla^2 l(w), \quad \nabla^2 l_i(w) = \nabla\left(x_i\left(h(x_i) - y_i\right)\right) = x_i e^{-w^T x_i}\left(1+e^{-w^T x_i}\right)^{-2} x_i$$
$$= x_i h(x_i)\left(1-h(x_i)\right) x_i$$

$$\Rightarrow H = \sum_{i=1}^{n} x_i h(x_i)\left(1-h(x_i)\right) x_i = X^T A X, \text{ where } A = \begin{bmatrix} h(x_1)(1-h(x_1)) & & 0 \\ & \ddots & \\ 0 & & h(x_n)(1-h(x_n)) \end{bmatrix}$$

$\forall z \in \mathbb{R}^3$

$$z^T H z = z^T X^T A X z = \sum_{i=1}^{n}\sum_{j=1}^{3} a_i z_j x_{ij} x_{ij} z_j = \sum_{i=1}^{n}\sum_{j=1}^{3} a_i z_j^2 x_{ij}^2 \geq 0$$

$\Rightarrow H$ is PSD

4.

(1)

The principle of Maximum Entropy is carried out by finding λ_0, λ_1 such that Lagrange Multipliers is made the Largest.

$$\mathcal{L} = \max_{P_k}\left[-\sum_{k=1}^{K} P_k \log P_k - (\lambda_0 - 1)\left(\sum_{k=1}^{K} P_k - 1\right)\right] \Rightarrow 0 = \frac{\partial \mathcal{L}}{\partial P_k} = -(\log P_k + 1) - (\lambda_0 - 1) = -\log P_k - \lambda_0$$

$\Rightarrow P_k = e^{-\lambda_0}$, for $k = 1 \sim K$ then $\lambda_0 = \ln K$

$\therefore P_k = \frac{1}{K} \Rightarrow$ the maximum entropy probability is a discrete uniform distribution

(2)

The Lagrangian is now

$$\mathcal{L} = \max_{P_i}\left[-\sum_{i=1}^{3} P_i \log P_i - (\lambda_0 - 1)\left(\sum_{i=1}^{3} P_i - 1\right) - \lambda_1\left(\sum_{i=1}^{3} P_i X_i - 2.5\right)\right]$$

$$0 = \frac{\partial \mathcal{L}}{\partial P_i} = -\log P_i - \lambda_0 - \lambda_1 X_i \Rightarrow P_i = e^{-\lambda_0 - X_i \lambda_1} \text{ for } i = 1, 2, 3$$

$\lambda_0 = 2.987$ $\qquad P_1 = 0.116$

$\lambda_1 = -0.834$ $\qquad P_2 = 0.268$

$\qquad P_3 = 0.616$

5

(1)

Since $k\langle\cdot,\cdot\rangle$ is an inner product kernel

$\therefore$ Let $x, y \in \mathbb{R}^n$ $\quad k\langle x,y\rangle = \sum_{i=1}^{n} \Phi_i(x)\,\Phi_i(y) = k\langle y,x\rangle \Rightarrow$ symmetric

The Gram matrix K, where $K_{ij} = k\langle x_i, x_j\rangle$, $x_i \in$ Finite Sample sets $S = \{x_1 \cdots x_m\}$

$\forall \alpha \in \mathbb{R}^n \quad \alpha^T K \alpha = \sum_i \sum_j \alpha_i \alpha_j K_{ij} = \sum_i \sum_j \alpha_i \alpha_j \Phi(x_i)\Phi(x_j) = \left(\sum_i \alpha_i \Phi(x_i)\right)\left(\sum_j \alpha_j \Phi(x_j)\right)$

$= \|\sum_i \alpha_i \Phi(x_i)\|_2^2 > 0$

$\Rightarrow K$ is PD

(2)

If $g(k\langle\cdot,\cdot\rangle)$ is a valid kernel $\Rightarrow g(k\langle\cdot,\cdot\rangle)$ is PD & symmetric

Let $g(x) = a_n x^n + \cdots + a_1 x + a_0$, where $a_i \geq 0$ for $i = 0 \sim n$

$\Rightarrow g(k\langle x,x'\rangle) = a_n k\langle x,x'\rangle^n + \cdots + a_1 k\langle x,x'\rangle + a_0 = a_n k\langle x',x\rangle^n + \cdots + a_1 k\langle x',x\rangle + a_0$

$= g(k\langle x',x\rangle) \Rightarrow$ symmetric

Let Q be the Gram matrix of $g(k\langle\cdot,\cdot\rangle)$, then $Q_{ij} = g(k\langle x_i, x_j\rangle)$

$\forall \alpha \in \mathbb{R}^n, \ \alpha^T Q \alpha = \sum_i \sum_j \alpha_i \alpha_j Q_{ij} = \sum_i \sum_j \sum_k \alpha_i \alpha_j a_k k\langle x_i, x_j\rangle^k = \sum_i \sum_j \sum_k \alpha_i \alpha_j a_k \Phi(x_i)^k \Phi(x_j)^k$

$= \left(\sum_i \alpha_i \sum_k a_k^{\frac{1}{2}} \Phi(x_i)^k\right)\left(\sum_j \alpha_j \sum_k a_k^{\frac{1}{2}} \Phi(x_j)^k\right) = \|\sum_i \alpha_i \sum_k a_k^{\frac{1}{2}} \Phi(x_i)^k\|^2 > 0$

$\Rightarrow Q$ is PD $\Rightarrow g(k\langle\cdot,\cdot\rangle)$ is a valid kernel

(3)

$(a_1 k_1 + a_2 k_2)\langle x,y\rangle = a_1 k_1\langle x,y\rangle + a_2 k_2\langle x,y\rangle = a_1 k_1\langle y,x\rangle + a_2 k_2\langle y,x\rangle = (a_1 k_1 + a_2 k_2)\langle y,x\rangle$

$\Rightarrow$ symmetric

Let $K_{ij} = (a_1 k_1 + a_2 k_2)\langle x_i, x_j\rangle$

$\forall \alpha \in \mathbb{R}^n, \ \alpha^T K \alpha = \sum_i \sum_j \alpha_i \alpha_j \left[a_1 \Phi_1(x_i)\Phi_1(x_j) + a_2 \Phi_2(x_i)\Phi_2(x_j)\right]$

$= a_1\left(\sum_i \alpha_i \Phi_1(x_i)\right)\left(\sum_j \alpha_j \Phi_1(x_j)\right) + a_2\left(\sum_i \alpha_i \Phi_2(x_i)\right)\left(\sum_j \alpha_j \Phi_2(x_j)\right) = a_1\|\sum_i \alpha_i \Phi_1(x_i)\|^2 + a_2\|\sum_i \alpha_i \Phi(x_i)\|^2 > 0$

$\Rightarrow$ PD

$\therefore a_1 k_1 + a_2 k_2$ is valid kernel

6.

(1)

Let $W^{(0)} = 0$, and by (4.55) $W^{(\tau+1)} = W^{(\tau)} + \eta \phi_n(x) t_n$

$\therefore W = \sum_{n=1}^{N} \alpha_n t_n \phi(x_n)$, where n indexes a pattern which is misclassified and α_n denotes how many times pattern n is used.

$\therefore y(n) = \text{sign}(W^T \Phi(x)) = \text{sign}\left(\sum_{n=1}^{N} \alpha_n t_n \phi(x_n)^T \Phi(x)\right) = \text{sign}\left(\sum_{n=1}^{N} \alpha_n t_n k\langle x_n, x\rangle\right)$

(2)

$\|x - x_n\|^2 = x^T x - 2x^T x_n + x_n^T x_n = k\langle x, x\rangle - 2k\langle x, x_n\rangle + k\langle x_n, x_n\rangle$

7.

(1)

$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2}\|w\|_2^2 + \sum_i C_i \xi_i + \sum_i \alpha_i (1 - y_i(w^T x_i + b) - \xi_i) - \sum_i r_i \xi_i$

, where $C = \begin{cases} C_{+1}, & \text{if } y_i = 1 \\ C_{-1}, & \text{otherwise} \end{cases}$

(2)

$\frac{\partial \mathcal{L}}{\partial W} = W - \sum_{i=1}^{N} \alpha_i y_i x_i = 0 \Rightarrow W = \sum_{i=1}^{N} \alpha_i y_i x_i$

$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i = 0 \Rightarrow \sum_{i=1}^{N} \alpha_i y_i = 0 = \alpha^T y$

$\frac{\partial \mathcal{L}}{\partial \xi_i} = C_i - \alpha_i - r_i = 0 \Rightarrow r_i = C_i - \alpha_i$, since $r_i \geq 0$ $\therefore \alpha_i \leq C_i \quad \forall i$

(3)

$\frac{1}{2}\|w\|^2 + \sum_{i=1}^{N} \alpha_i (1 - y_i(w^T x_i)) + \sum_{i=1}^{N} \overbrace{\alpha_i y_i}^{=0} b + \sum_{i=1}^{N} (C_i - \alpha_i - r_i) \xi_i$

$= \frac{1}{2}\|w\|^2 + \sum \alpha_i - W^T \overbrace{\sum \alpha_i y_i x_i}^{=W} = \sum \alpha_i - \frac{1}{2} W^T W = \sum_i \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j x_i x_j$

$\Rightarrow$ dual problem $= \max.\ 1^T \alpha - \frac{1}{2} \alpha^T \hat{K} \alpha$, where $\hat{K}_{ij} = y_i y_j x_i x_j$

subject to $\alpha^T y = 0$, $0 \leq \alpha_i \leq C_i, \forall i$

(4)

At the Prime problem, we change the constraint $y_i(W^T x_i + b) \ge 1 - \xi_i$ to $y_i(W^T \Phi(x_i) + b) \ge 1 - \xi_i$

the procedure of (1) to (3) will not change.

the dual problem $= \max \ 1^T \alpha - \frac{1}{2} \alpha^T \tilde{K} \alpha$, where $\tilde{K}_{ij} = y_i y_j K\langle x_i, x_j \rangle$

subject to $\alpha^T y = 0$

$0 \le \alpha_i \le C_i$, $\forall i$

Problem 3

(1)

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

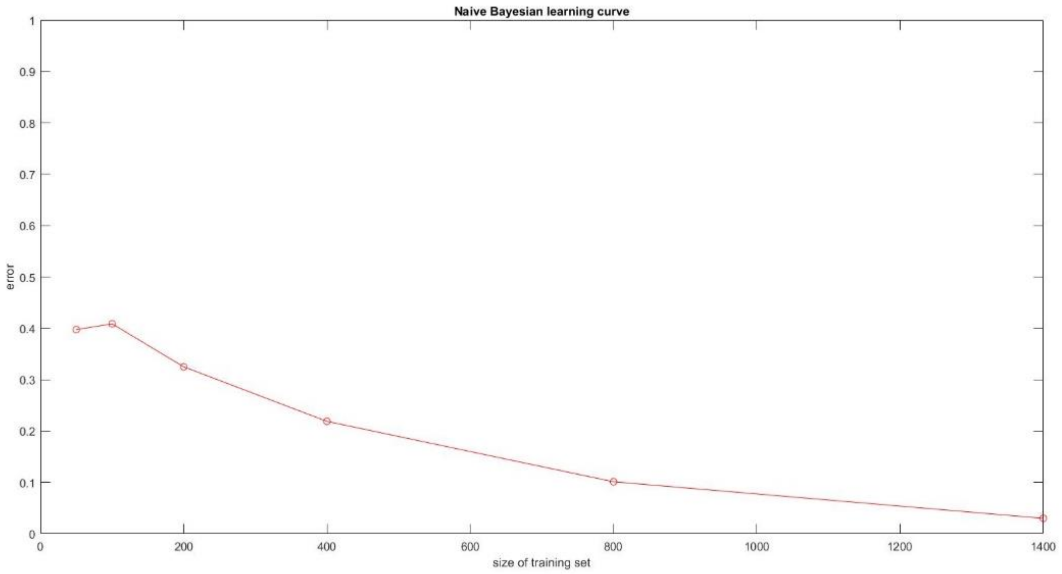
(2)

$$token_{list} = [616 \quad 1210 \quad 1357 \quad 194 \quad 1369]$$

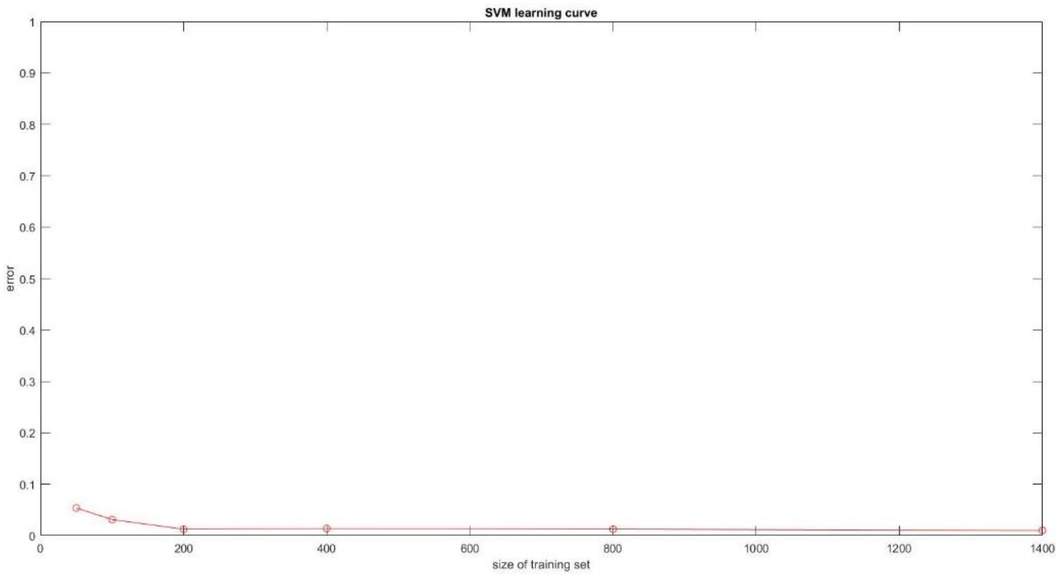
The corresponding words are httpaddr, spam, unsubscribe, cent, and valet.

(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
```