### README

Heng zhe Duan(hd79), Yu Cheng(yc489)

Usage:

Main.m deals with problem 1a and 1c.

Oneb.m deals with problem 1b.

Oned.m deals with problem 1d.

Before running oneb.m and oned.m, make sure the global variables:

global featurenum;

global datanum;

featurenum= 29328;

datanum = 1000;

are defined.

This is how Main.m works:

It calls on readData() to read the data into matrices usable by Matlab.

It runs getW() on the matrices to get the w and b matrix/vector.

getErrorRate() predicts the label using the produced w and b, and compares with the real w and b to determine the error rate.

The solution is then graphed.

This is how oneb.m works:

It simply serves as a plot tool. I ran SVM\_light with the different parameters and produced the results.

oneb.m simply takes this result and plots it.

This is how oned.m works:

It calls readData() on the result generated with SVM\_light, as well as the result from classifying using perceptron primal (explained in Main.m). Then it compares each of the predicted result to the real results and tally up the exclusive errors each made. This is used in the documentation to complete the statistical test.

### readData.m (Used to read data into matlab)

function [labelArr, featureM] = readData(filename)

global datanum;

global featurenum;

labelArr = zeros(datanum, 1);

featureM = zeros(datanum, featurenum);

fid = fopen(filename);

tline = fgetl(fid);

i = 1;

while ischar(tline)

[l, f] = readLine(tline, featurenum);

labelArr(i) = l;

featureM(i, :) = f;

tline = fgetl(fid);

i = i + 1;

if i ~= 1001 && mod(i, 50) == 0

fprintf('%3.0f%% complete\n', i/10);

end

end

fclose(fid);

function [label, features] = readLine(line, featurenum)

[label, line] = strtok(line);

label = str2double(label);

features = zeros(1, featurenum);

while ~isempty(line)

[l, line] = strtok(line);

[k, v] = strtok(l, ':');

[v, tt] = strtok(v, ':');

features(str2double(k)) = str2double(v);

end

### getW.m(Used to calculate w/b)

function [wM, bA] = getW(labelArr, featureM)

global featurenum;

global datanum;

n = 1;

wM = zeros(datanum, featurenum);

bA = zeros(datanum, 1);

k = 1;

totalk = 0;

normv = zeros(datanum, 1);

for i = 1:datanum

normv(i) = norm(featureM(i, :));

end

R = max(normv);

cont = 1;

j = 1;

while (cont && j < 50)

cont = 0;

wM(j, :) = wM(k, :);

bA(j) = bA(k);

totalk = totalk + k;

k = j;

for i = 1:datanum

if labelArr(i) \* (sum(wM(k, :) .\* featureM(i, :)) + bA(k)) <= 0

wM(k+1, :) = wM(k, :) + n \* labelArr(i) \* featureM(i, :);

bA(k+1) = bA(k) + n \* labelArr(i) \* R ^ 2;

k = k + 1;

cont = 1;

end

end

j = j+1;

end

for p = j : datanum

wM(p, :) = wM(j-1, :);

bA(p) = bA(j-1);

end

### getErrorRate.m(Used to calculate the error rate of certain test data)

function errorRate = getErrorRate(label, feature, w, b)

errors = 0.;

sampleSize = length(label);

for i = 1:sampleSize

if label(i) \* (sum(w .\* feature(i, :)) + b) <= 0

errors = errors + 1;

end

end

errorRate = errors / sampleSize;

### classify.m(Used to classify a feature with given hyperplane w/b)

function classifiedLabels = classify(feature, w, b)

sampleSize = size(feature,1);

classifiedLabels = zeros(sampleSize,1);

for i = 1:sampleSize

classifiedLabels(i) = sum(w .\* feature(i, :)) + b;

end

### main.m

global featurenum;

global datanum;

featurenum= 29328;

datanum = 1000;

disp 'reading validation data...'

[validY, validX] = readData('movie/polarity.validation');

totIter = 20;

validErrorRate = zeros(1, totIter);

trainErrorRate = zeros(1, totIter);

validErrorRate1 = zeros(1, totIter);

trainErrorRate1 = zeros(1, totIter);

validErrorRate2 = zeros(1, totIter);

trainErrorRate2 = zeros(1, totIter);

validErrorRate3 = zeros(1, totIter);

trainErrorRate3 = zeros(1, totIter);

disp 'reading training data...'

[trainY, trainX] = readData('movie/polarity.train');

disp 'getting W for training data...'

[w, b] = getW(trainY, trainX);

disp 'getting Error rates...'

for i = 2:(totIter+1)

validErrorRate(i-1) = getErrorRate(validY, validX, w(i,:), b(i));

trainErrorRate(i-1) = getErrorRate(trainY, trainX, w(i,:), b(i));

end

clear w;

clear b;

clear trainx;

clear trainY;

disp 'reading training data1...'

[trainY1, trainX1] = readData('movie/polarity-reorder-1.train');

disp 'getting W for training data1...'

[w1, b1] = getW(trainY1, trainX1);

disp 'getting Error rates1...'

for i = 2:(totIter+1)

validErrorRate1(i-1) = getErrorRate(validY, validX, w1(i,:), b1(i));

trainErrorRate1(i-1) = getErrorRate(trainY1, trainX1, w1(i,:), b1(i));

end

clear w1;

clear b1;

clear trainY1;

clear trainX1;

disp 'reading training data2...'

[trainY2, trainX2] = readData('movie/polarity-reorder-2.train');

disp 'getting W for training data2...'

[w2, b2] = getW(trainY2, trainX2);

disp 'getting Error rates2...'

for i = 2:(totIter+1)

validErrorRate2(i-1) = getErrorRate(validY, validX, w2(i,:), b2(i));

trainErrorRate2(i-1) = getErrorRate(trainY2, trainX2, w2(i,:), b2(i));

end

clear w2;

clear b2;

clear trainY2;

clear trainX2;

disp 'reading training data3...'

[trainY3, trainX3] = readData('movie/polarity-reorder-3.train');

disp 'getting W for training data3...'

[w3, b3] = getW(trainY3, trainX3);

disp 'getting Error rates3...'

for i = 2:(totIter+1)

validErrorRate3(i-1) = getErrorRate(validY, validX, w3(i,:), b3(i));

trainErrorRate3(i-1) = getErrorRate(trainY3, trainX3, w3(i,:), b3(i));

end

clear w3;

clear b3;

clear trainY3;

clear trainX3;

figure

hold on

plot (1:totIter, validErrorRate, 'g')

plot (1:totIter, trainErrorRate, 'r')

xlabel ('Iteration Number')

ylabel ('Error Rate')

legend ('validation error rate', 'training error rate')

title ('1a')

figure

hold on

plot (1:totIter, validErrorRate, 'g')

plot (1:totIter, validErrorRate1, 'r')

plot (1:totIter, validErrorRate2, 'k')

plot (1:totIter, validErrorRate3, 'm')

xlabel ('Iteration Number')

ylabel ('Error Rate')

legend ('validErrorRate on train', 'validErrorRate on train1', 'validErrorRate on train2', 'validErrorRate on train3')

title ('1c part1')

figure

hold on

plot (1:totIter, trainErrorRate, 'g')

plot (1:totIter, trainErrorRate1, 'r')

plot (1:totIter, trainErrorRate2, 'k')

plot (1:totIter, trainErrorRate3, 'm')

xlabel ('Iteration Number')

ylabel ('Error Rate')

legend ('trainErrorRate on train', 'trainErrorRate on train1', 'trainErrorRate on train2', 'trainErrorRate on train3')

title ('1c part2')

### oneb.m

a = 1 - [0.722 0.780 0.844 0.910 0.963 0.991 1.000 1.000 1.000];

b = 1 - [0.667 0.702 0.736 0.764 0.779 0.795 0.781 0.784 0.784];

c = [0.25 0.5 1 2 4 8 16 32 64];

c = log(c)/log(2);

figure

hold on

plot(c, a, 'r')

plot(c, b, 'g')

legend('training set', 'validation set')

xlabel('log\_2(C)')

ylabel('error rate')

### oned.m

global datanum;

disp 'reading validation data...'

[validY, validX] = readData('movie/polarity.validation');

disp 'reading training data...'

[trainY, trainX] = readData('movie/polarity.train');

disp 'getting W for training data...'

[w, b] = getW(trainY, trainX);

trainingLabels\_primal = sign(classify(validX, w(20,:), b(20)));

realLabels = validY;

[trainingLabels\_SVM, ~] = readData('movie/output1c/8c');

trainingLabels\_SVM = sign(trainingLabels\_SVM);

primalError = 0;

SVMError = 0;

bothError = 0;

for i = 1:datanum

if realLabels(i) ~= trainingLabels\_primal(i) || realLabels(i) == 0

primalError = primalError + 1;

if realLabels(i) ~= trainingLabels\_SVM(i) || realLabels(i) == 0

bothError = bothError + 1;

end

end

if realLabels(i) ~= trainingLabels\_SVM(i) || realLabels(i) == 0

SVMError = SVMError + 1;

end

end

SVMonlyError = SVMError - bothError

primalonlyError = primalError - bothError