ECE 693

Big Data Security

Lab 3

By

xxxxx

1. Use some data as examples, explain how logistic regression and belief propagation work. Search online codes for those two algorithms. Then link to our lectures to explain how those two tools could be used to recognize the attacks.

Logistic Regression

The training of logistic regression is done using generated random data (Gaussian distributed). The train data has 1000 samples and 2 features. The logistic regression output/label is defined as

$$y = w_0 + w_1 x_1 + w_2 x_2$$

Where, x1 = 1st feature and x2 = 2nd feature while w0, w1, and w2 are the respective weights. During the training session those weight values are calculated.

For testing, we generate a new set of random data (Gaussian distributed) which is tested on the trained model and find the label of test data. The size of test data is 200 samples and 2 features.

Code is given below

```
%% Logistic logistic regression for binary classification
clear; close all;
k = 2;      % no. of features
n = 1000;   % no. of samples
[traindata,response] = kmeansRnd(2,k,n);     % to generate gaussian data
[model, llh] = logitBin(traindata,response-1);
plot(llh);
[testdata,testlabel] = kmeansRnd(2,k,n);     % to generate gaussian data
[y,p] = logitBinPred(model,testdata);
y = y+1;
binPlot(model,testdata,y)
```

```
function [X, z, mu] = kmeansRnd(d, k, n)
% Generate samples from a Gaussian mixture distribution with common
variances (kmeans model).
% Input:
%   d: dimension of data
%   k: number of components
%   n: number of data
% Output:
%   X: d x n data matrix
%   z: 1 x n response variable
%   mu: d x k centers of clusters
% Refernce: Written by Mo Chen (sth4nth@gmail.com).
alpha = 1;
beta = nthroot(k,d); % in volume x^d there is k points: x^d=k

X = randn(d,n);
w = dirichletRnd(alpha,ones(1,k)/k);
z = discreteRnd(w,n);
E = full(sparse(z,1:n,1,k,n,n));
mu = randn(d,k)*beta;
X = X+mu*E;

function x = dirichletRnd(a, m)
```

```
% Generate samples from a Dirichlet distribution.
% Input:
%   a: k dimensional vector
%   m: k dimensional mean vector
% Outpet:
%   x: generated sample x~Dir(a,m)
% Written by Mo Chen (sth4nth@gmail.com).
if nargin == 2
    a = a*m;
end
x = gamrnd(a,1);
x = x/sum(x);

function x = discreteRnd(p, n)
% Generate samples from a discrete distribution (multinomial).
% Input:
%   p: k dimensional probability vector
%   n: number of samples
% Ouput:
%   x: k x n generated samples x~Mul(p)
% Written by Mo Chen (sth4nth@gmail.com).
if nargin == 1
    n = 1;
end
r = rand(1,n);
p = cumsum(p(:));
[~,x] = histc(r,[0;p/p(end)]);
```

```
function [model, LLLH] = logitBin(data, label, lambda)
% Logistic regression for binary classification optimized by Newton-Raphson
method.
% Input:
%   data: d x n data matrix
%   label: 1 x n label (0/1)
%   lambda: regularization parameter
% Output:
%   model: trained model structure
%   LLLH: loglikelihood
% Refernce: Written by Mo Chen (sth4nth@gmail.com).
if nargin < 3
    lambda = 1e-2;
end
data = [data; ones(1,size(data,2))];
[d,n] = size(data);     % d = no. of features, n = no. of samples
tol = 1e-4;     % tolerance
maxiter = 100;
LLLH = -inf(1,maxiter);
idx = (1:d)';
dg = sub2ind([d,d],idx,idx);    % diagonal index
label1 = ones(1,n);             % label1 = label {-1,1}
label1(label==0) = -1;
w = zeros(d,1);     % Logistic regression Coefficient/weight
a = w'*data;
for iter = 2:maxiter
    y = sigmoid(a);
    r = y.*(1-y);
```

```
    Xw = bsxfun(@times, data, sqrt(r));
    H = Xw*Xw';
    H(dg) = H(dg)+lambda;
    U = chol(H);
    g = data*(y-label)'+lambda.*w;
    p = -U\(U'\g);
    wo = w;
    w = wo+p;
    a = w'*data;
    LLLH(iter) = -sum(log1pexp(-label1.*a))-0.5*sum(lambda.*w.^2);  %
calculate log likelihood
    incr = LLLH(iter)-LLLH(iter-1);
    if incr < tol; break; end
end
LLLH = LLLH(2:iter);
model.w = w;
```

```
function [y, p] = logitBinPred(model, testdata)
% Prediction of binary logistic regression model
% Input:
%   model: trained model structure
%   X: d x n testing data
% Output:
%   y: 1 x n predict label (0/1)
%   p: 1 x n predict probability [0,1]
% Refernce: Written by Mo Chen (sth4nth@gmail.com).
testdata = [testdata;ones(1,size(testdata,2))];
w = model.w;
p = exp(-log1pexp(w'*testdata));
y = round(p);
```

```
function y = log1pexp(x)
% Accurately compute y = log(1+exp(x))
seed = 33.3;
y = x;
idx = x<seed;
y(idx) = log1p(exp(x(idx)));
```

```
function binPlot(model, X, label)
% Plot binary classification result for 2d data
% Input:
%   model: trained model structure
%   X: 2 x n data matrix
%   label: 1 x n label
% Refernce: Written by Mo Chen (sth4nth@gmail.com).
assert(size(X,1) == 2);
w = model.w;
xi = min(X,[],2);
xa = max(X,[],2);
[x1,x2] = meshgrid(linspace(xi(1),xa(1)), linspace(xi(2),xa(2)));

color = 'brgmcyk';
m = length(color);
figure(gcf);
axis equal
clf;
```

```
hold on;
view(2);
for i = 1:max(label)
    idc = label==i;
    scatter(X(1,idc),X(2,idc),36,color(mod(i-1,m)+1));
end
y = w(1)*x1+w(2)*x2+w(3);
contour(x1,x2,y,[-0 0]);
hold off;
```

Results of Logistic regression

Sample of train data

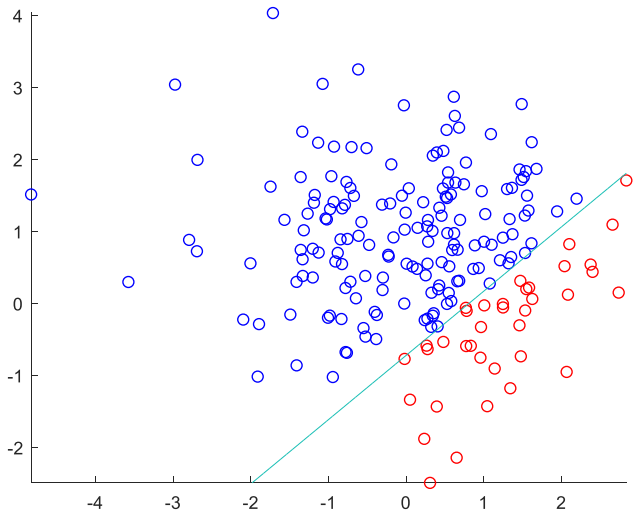
Label	1	2	2	2	1	2	2	2	2	1
X1	0.30	-3.97	-1.68	-3.29	-0.79	-1.77	-2.09	-1.39	-1.71	1.21
X2	-1.30	0.25	0.98	1.18	-3.21	0.39	1.52	1.03	-1.06	-1.86

Sample of test data

Label	2	1	1	1	1	1	1	2	1	1	1
X1	-2.69	0.96	1.24	-0.40	1.37	-0.83	1.35	-1.34	0.46	0.01	0.63
X2	1.99	-0.75	-0.01	-0.12	0.96	-0.22	0.64	2.38	0.57	0.55	1.68

Corresponding probability of being class 2

Probability	0%	85%	69%	13%	29%	8%	42%	0%	15%	8%
Predicted label	1	2	2	1	1	1	1	1	1	1



Here red dot is representing class 1 and blue dot represents class 2.

belief propagation

For belief propagation I am using an example of error correction. It calculates belief using marginal probability. The code is given below:

```
clear;
disp("parity check matrix for (7,4) hamming code");
pchkAry = [1, 1, 1, 0, 1, 0, 0;
           0, 1, 1, 1, 0, 1, 0;
           1, 0, 1, 1, 0, 0, 1];          % parity check matrix for (7,4)
hamming code
disp(pchkAry);
codeMsgNoisy =  [1, 0, 1, 1, 0, 0, 0];      % input code word with 1 bit
error
decodeMsg = zeros(1, numel(codeMsgNoisy));  % array for storing decoded word
 % build the edge array which stores the connection between variable nodes
and factor nodes
edgeAry = [eye(7); pchkAry];
[i, j, s] = find(edgeAry);                  % find the index of edgeArray;
edgeMap = sparse(i, j, 1:length(s));        % map edge id to node id;

max_iter = 10;                              % max number of iteration for
bp;
prob = 0.01;                                % error probability for BSC
% get number of factor nodes and variable nodes,
% where the num of factor nodes is equal to n + m, and num of factor nodes
is n;
[m,n] = size(pchkAry);
no_Edge = sum(edgeAry(:));                  % get number of edges between
variable nodes and factor nodes
% set the num of bp labels, where 2 means we only use 0 and 1 as the label
in our bp algorithm.
no_Label = 2;
LabelVal = [0, 1];                        % set the value
msgF2V = ones(no_Edge, no_Label);           % message from variable nodes to
factor nodes
msgV2F = ones(no_Edge, no_Label);           % message from factor nodes to
variable
belief = ones(n, no_Label);                 % belief of each variable node.
epsVal = 10E-5;                             % a small value to make sure
there is no zero denominator;


%% update factor node 1 to 7. since they only have one variable node
neighbor, they only need to be update once.
for i = 1:n
    for j = i:i
        edgeID = edgeMap(i,j); % find the edgeID of current neighbor j for
factor node i.
        for k = 1: no_Label
            if(LabelVal(k) == codeMsgNoisy(i))
                msgF2V(edgeID,k) = 1 - prob;
            else
                msgF2V(edgeID,k) = prob;
            end
        end
```

```
    end
end

%% start belief propagation iteration
for iter = 1:max_iter

    %% update variable nodes 1 to 7
    for j = 1:n
        % get the message from Left side factor node;
        prodVal = msgF2V(edgeMap(j,j), :);
        % find neighbor's edgeID;
        edgeIDArray = full(edgeMap(find(edgeMap(j+1:n+m,j)~= 0) + j, j));

        % mutliply all the incoming message from its neighbor
        for i = 1:length(edgeIDArray)
            edgeID = edgeIDArray(i); % find the index of current neighbor i.
            prodVal = prodVal .* msgF2V(edgeID,:);
        end

        %send out the message, divide prodVal by the incoming message you
        %get the sending out message;
        for i = 1:length(edgeIDArray)
            % find the index of current neighbor i.
            edgeID = edgeIDArray(i);
            msgV2F(edgeID,:) = prodVal ./ msgF2V(edgeID,:);
            % normalization
            msgV2F(edgeID,:) = msgV2F(edgeID,:) / sum(msgV2F(edgeID,:));
        end

    end

    %% update factor nodes  8 to 10
    for i = n+1 : n+m
        edgeIDArray = full(edgeMap(i,edgeMap(i, :)~= 0));  % find neighbor's
edgeID;
        for j = 1: length(edgeIDArray)
            edgeID = edgeIDArray(j); % find the index of current neighbor j.
            for k = 1:no_Label
                msgF2V(edgeID,k) = sumMsg(LabelVal(k), edgeIDArray, msgV2F,
j);
            end
            msgF2V(edgeID,:) = msgF2V(edgeID,:) / sum(msgF2V(edgeID,:));
% normalization
        end
    end

    %% calculate belief
    for j = 1:n
        belief(j, :) = msgF2V(edgeMap(j,j), :);  % get the message from Left
side factor node;
        edgeIDArray = full(edgeMap(find(edgeMap(j+1:n+m,j)~= 0) + j, j));  %
find neighbor's edgeID;
        % mutliply all the incoming message
        for i = 1 : length(edgeIDArray)
```

```
            edgeID = edgeIDArray(i); % find the edgeID of current neighbor i
for variable node j.
            belief(j, :) = belief(j, :) .* msgF2V(edgeID,:);
        end
    end

    %% get decoded codeword by belief
    decodeMsg = fix(belief(:,2) ./ belief(:,1))';
    decodeMsg(decodeMsg ~= 0) = 1;
    %% validate the decoded codeword
    checkVal = mod(sum(pchkAry * decodeMsg'),2);
    if(iter >= max_iter || checkVal == 0)
        break;
    end

end

if(checkVal ==0)
    disp('Successfully decoded');
else
    disp('decoding failed;')

end
disp(['input noisy message: ', int2str(codeMsgNoisy)]);
disp(['    decoded message: ', int2str(decodeMsg)]);
```

```
function val = sumMsg(x, edgeIDArray, messageV2F, jV)
    edgeIDArray(jV) = [];
    nV = length(edgeIDArray);
    len = 2^nV;
    val = 0;
    for i = 1: len
        binArray = rot90(dec2binvec(i-1,nV),2);
        prodVal = fun2(x, binArray);
        if(prodVal ~= 0)
            for j = 1 : nV
                edgeID = edgeIDArray(j);
                prodVal = prodVal * messageV2F(edgeID,binArray(j)+1);
            end
        end
        val = val + prodVal;
    end
end
```

Output

parity check matrix for (7,4) hamming code

1	1	1	0	1	0	0
0	1	1	1	0	1	0
1	0	1	1	0	0	1

Successfully decoded
input noisy message: 1 0 1 1 0 0 0
decoded message: 1 0 1 1 0 0 1

Belief/ marginal probability

0	0.02%	54.22%	0.06%	0.02%	73.74%	73.74%	0.06%
1	1.00%	0.07%	0.58%	1.00%	0.26%	0.26%	1.00%

parity check matrix for (7,4) hamming code	Belief/ marginal probability
1 1 1 0 1 0 0 0 1 1 1 0 1 0 1 0 1 1 0 0 1 Successfully decoded input noisy message: 1 0 1 0 1 0 0 decoded message: 1 1 1 0 1 0 0	0 0.03% 0.09% 0.03% 2.83% 0.97% 2.91% 96.09% 1 2.83% 0.94% 0.08% 0.03% 2.91% 0.97% 0.03%

Using the belief propagation algorithm, it is capable to detect error and as well as correct bit error in communication.

In order to detect attack and benign, those above-mentioned algorithms are used. Logistic regression is used as a classifier. It classifies the new data as an attack or non-attack category using conditional probability. For logistic regression it uses four features they are port change, unknown exect, in-connect and out-connect. After that they used belief propagation to make sure it is an attack or not. In belief propagation algorithm they used marginal probability. Belief propagation gives a label whether the data is coming from attack or benign. If it is coming from and attack, then it will add those features to the database and retrained the logistic regression classifier again.