ECE8560

Pattern Recognition

Takehome #2

Shreyas Prabhakar

sprabha

1. **Assessing Classification Results from Takehome #1**

In the takehome #1, we were asked to design and test a Bayesian, minimum error classifier for a given H and S_T. The problem was a c=3 class, d=4-dimensional problem. The training data and testing data were provided to us. Training data (H) of 15000 samples were provided to us in a file name train_sp2017_v19 and the testing data (S_T) of 15000 samples were provided to us in a file name test_sp2017_v19.

We trained our Bayesian classifier using the training data H and then tested our classifier for the test data S_T and the corresponding class of each of the sample in S_T was stored in an ASCII file named *sprabha-classified-takehome1.txt* with the integer for the class per line.

In takehome #2, we are given the true class sequence of S_T as 3-1-2-3-2-1, which repeats throughout S_T. We are asked to build a confusion matrix for the output classified in Takehome #1, knowing the true class sequence and also to estimate P(error).

Confusion Matrix (CM)

The confusion matrix for the Bayesian Classifier implemented in Takehome #1 is a 3*3 matrix where the rows are the true classes and the columns are the predicted classes.

N = 15000	Class 1 (Predicted)	Class 2 (Predicted)	Class 3 (Predicted)
Class 1 (True)	4218	547	235
Class 2 (True)	395	4568	37
Class 3 (True)	148	48	4804

P(error)

The P(error) is calculated using

$$P(error) = 1 - \frac{sum(diag(CM))}{N}$$

Where, N = 15000 (samples in S_T)

$$P(error) = 1 - \frac{(4218+4568+4804)}{15000}$$

$$\boldsymbol{P(error) = 0.094 = 9.4\%}$$

The P(error) for the designed Bayesian Classifier is found to be 0.094 or 9.4%

2. Ho-Kashyap Hyperplanar Classifier

In this part of Takehome #2, we are asked to implement Ho-Kashyap Hyperplanar classifier for the given training set H. The Ho-Kashyap hyperplanar classification method is an iterative method, where the margin 'b' and the parameter of the hyperplane 'w' are estimated iteratively. In Ho-Kashyap, we first transform our data to (d+1) dimension and the 'w' is

$$\boldsymbol{\omega} = [\boldsymbol{\omega}\ \boldsymbol{\omega_0}]$$

$$g(x) = \boldsymbol{\omega}_{ij}^{T} x - \boldsymbol{\omega}_0$$

For the 3-class training set provided, I found the Hyperplane parameter 'w' to be –

1. Between class 1 and class 2 –

$$\boldsymbol{\omega} = [\mathbf{0.0057},\ \mathbf{-0.0032},\ \mathbf{7.8486 * 10^{-4}},\ \mathbf{-0.0131},\ \mathbf{-0.5248}]$$

$$\boldsymbol{x} = \begin{cases} \boldsymbol{class\ 1}; & \boldsymbol{if\ g1(x) > 0} \\ \boldsymbol{class\ 2}; & \boldsymbol{if\ g1(x) \leq 0} \end{cases}$$

P(error) = (misclassification of class1 – misclassification of class 2)/15000 = **0.1823**

2. Between class 1 and class 2 –

$$\boldsymbol{\omega} = \left[\mathbf{4.1242 * 10^{-4}},\ \mathbf{-0.0017}, \mathbf{-0.0236},\ \mathbf{3.1864 * 10^{-5}},\ \mathbf{-0.0217}\right]$$

$$\boldsymbol{x} = \begin{cases} \boldsymbol{class\ 1}; & \boldsymbol{if\ g2(x) > 0} \\ \boldsymbol{class\ 3}; & \boldsymbol{if\ g2(x) \leq 0} \end{cases}$$

P(error) = (misclassification of class1 – misclassification of class 2)/15000 = **0.1075**

3. Between class 1 and class 2 –

$$\boldsymbol{\omega} = [\mathbf{-0.0022}, \mathbf{-6.4909 * 10^{-4}},\ \mathbf{-0.0202},\ \mathbf{0.0082},\ \mathbf{0.2797}]$$

$$\boldsymbol{x} = \begin{cases} \boldsymbol{class\ 2}; & \boldsymbol{if\ g3(x) > 0} \\ \boldsymbol{class\ 3}; & \boldsymbol{if\ g3(x) \leq 0} \end{cases}$$

P(error) = (misclassification of class1 – misclassification of class 2)/15000 = **0.0493**

4. The final decision conditions for classification between the 3 classes is determined by the pairwise classification rules (from above)

$$\boldsymbol{x} = \begin{cases} \boldsymbol{class\ 1};\ \boldsymbol{if\ (g1(x) > 0) and (\ g2(x) > 0)} \\ \boldsymbol{class\ 2};\ \boldsymbol{if\ (g1(x) \leq 0) and (\ g3(x) > 0)} \\ \boldsymbol{class\ 3};\ \boldsymbol{if\ (g2(x) \leq 0) and (\ g3(x) \leq 0)} \end{cases}$$

Confusion Matrix (CM)

The confusion matrix for the Bayesian Classifier implemented in Takehome #1 is a 3*3 matrix where the rows are the true classes and the columns are the predicted classes.

N = 15000	Class 1 (Predicted)	Class 2 (Predicted)	Class 3 (Predicted)
Class 1 (True)	4195	582	223
Class 2 (True)	1283	3717	0
Class 3 (True)	739	17	4244

P(error)

The P(error) is calculated using

$$P(error) = 1 - \frac{sum(diag(CM))}{N}$$

Where, N = 15000 (samples in S_T)

$$P(error) = 1 - \frac{(4195+3717+4244)}{15000}$$

$$\boldsymbol{P(error) = 0.1896 = 18.96\%}$$

The P(error) for the designed Bayesian Classifier is found to be 0.1896 or 18.96%

3. *k*-NNR Strategies

In this part of Takehome #2, we are asked to implement k-NNR classifier of k = 1,3, and 5 for samples in S_T using H. The nearest neighbours of a sample in ST is taken as the datapoints in H which has the minimum Euclidian distance between the sample in ST and all the samples in H.

I have implemented k-NNR in MATLAB using the 'bsxfun' function. This functions converts the input data from any numerical data type (eg: double) to gpuArray datatype and runs the computation on the GPU, and then returns the data after converting it from gpuArray datatype back to the original datatype. Also, it is known that the computations on matrices as a whole is faster than doing computations iteratively using loops for a subset of data. I implemented the k-NNR by exploiting the above 2 mentioned points for computation.

I found that a simple brute force for calculating the Euclidian distance for every sample in ST with all the sample in H, sorting the distances and taking the 'k' samples with lowest distances took about 28.3 minutes for k = 1,3, and 5. By exploiting the computations using GPU and matrix computations, k-NNR for k = 1,3, and 5 together are all calculated within 10 seconds.

In 3-NNR and 5-NNR, when there are more than 1 competing class, with equal number of nearest points. The classifier I implemented chooses the class which has a data point with closest (nearest) to the required test point.

K=1

Confusion Matrix (CM)

The confusion matrix for the Bayesian Classifier implemented in Takehome #1 is a 3*3 matrix where the rows are the true classes and the columns are the predicted classes.

N = 15000	Class 1 (Predicted)	Class 2 (Predicted)	Class 3 (Predicted)
Class 1 (True)	4101	631	268
Class 2 (True)	654	4264	82
Class 3 (True)	289	63	4648

P(error)

The P(error) is calculated using

$$P(error) = 1 - \frac{sum(diag(CM))}{N}$$

Where, N = 15000 (samples in S_T)

$$P(error) = 1 - \frac{(4101+4264+4648)}{15000}$$

$$\boldsymbol{P(error) = 0.1325 = 13.25\%}$$

The P(error) for the designed Bayesian Classifier is found to be 0.1325 or 13.25%

K=3

Confusion Matrix (CM)

The confusion matrix for the Bayesian Classifier implemented in Takehome #1 is a 3*3 matrix where the rows are the true classes and the columns are the predicted classes.

N = 15000	Class 1 (Predicted)	Class 2 (Predicted)	Class 3 (Predicted)
Class 1 (True)	4283	491	226
Class 2 (True)	626	4311	63
Class 3 (True)	247	54	4699

P(error)

The P(error) is calculated using

$$P(error) = 1 - \frac{sum(diag(CM))}{N}$$

Where, N = 15000 (samples in S_T)

$$P(error) = 1 - \frac{(4283+4311+4699)}{15000}$$

$$\boldsymbol{P(error) = 0.1138 = 11.38\%}$$

The P(error) for the designed Bayesian Classifier is found to be 0.1138 or 11.38%

K=5-

Confusion Matrix (CM)

The confusion matrix for the Bayesian Classifier implemented in Takehome #1 is a 3*3 matrix where the rows are the true classes and the columns are the predicted classes.

N = 15000	Class 1 (Predicted)	Class 2 (Predicted)	Class 3 (Predicted)
Class 1 (True)	4351	432	217
Class 2 (True)	592	4352	56
Class 3 (True)	235	51	4714

P(error)

The P(error) is calculated using

$$P(error) = 1 - \frac{sum(diag(CM))}{N}$$

Where, N = 15000 (samples in S_T)

$$P(error) = 1 - \frac{(4351+4352+4714)}{15000}$$

$$\boldsymbol{P(error) = 0.1055 = 10.55\%}$$

The P(error) for the designed Bayesian Classifier is found to be 0.1055 or 10.55%

4. **PCA**

In this part of Takehome #2, we are asked to implement PCA classifier, by using the 2 'best' features which are the linear combinations of the original features. We are supposed to use PCA on H to determine the projected features which maximize the variance.

The PCA is of the form

$$Y = AX$$

The brief algorithm is as given below:

1. Given the training set H. We first calculate the means (m) of the entire training set. We then subtract the mean from all the samples in H to make it zero mean H.

$$\mu_i = \frac{1}{n}\sum_{k=1}^{n} x_k$$

2. We find the covariance of the entire training set H, which is Σ_X**.** Σ_X is highly correlated.

$$\sum = \frac{1}{n-1}\sum_{k=1}^{n}(x_k - \mu)(x_k - \mu)^T$$

3. We find the covariance Y i.e Σ_Y, by taking the eigen of Σ_X. Σ_Y is a diagonal and uncorrelated.

$$\Sigma_Y = M^T \Sigma_X M$$

4. The diagonal elements of Σ_Y (which are the eigen values) gives us the correlation of every feature. So, to find the 'best' 2 features, we pick the features which have high correlation.
5. So, for this, we sort the diagonal of Σ_Y in descending order, and select the first 2 features, which has the largest correlation (eigen values).
6. We reorder A (features in H) according to the sorting done in the previous step.
7. We pick the first 2 (best) columns of A (features of H) according to the largest eigen values (e(i)). Let us call this H1.
8. We add the means of the correspond features back.
9. We reorder ST and pick the 2 features, correspond to the largest eigen values calculated earlier. Let us call this ST1.
10. We send this 2-dimensional reduced feature vectors H1 to the Bayesian Classifier designed in Takehome #1 and classify ST1.

Confusion Matrix (CM)

The confusion matrix for the Bayesian Classifier implemented in Takehome #1 is a 3*3 matrix where the rows are the true classes and the columns are the predicted classes.

N = 15000	Class 1 (Predicted)	Class 2 (Predicted)	Class 3 (Predicted)
Class 1 (True)	4606	273	123
Class 2 (True)	1608	3392	0
Class 3 (True)	640	0	4360

P(error)

The P(error) is calculated using

$$P(error) = 1 - \frac{sum(diag(CM))}{N}$$

Where, N = 15000 (samples in S_T)

$$P(error) = 1 - \frac{(4606+3392+4360)}{15000}$$

$$\boldsymbol{P(error) = 0.1763 = 17.63\%}$$

The P(error) for the designed Bayesian Classifier is found to be 0.1763 or 17.63%

5. Comparison and Analysis

Combining Takehome #1 and Takehome #2, we have implemented 4 different types of classifiers –

1. Supervised Classification - Bayesian Classifier (Maximum Likelihood Estimation and Bayesian Parametric Estimation),
2. Hyperplanes classifer (Ho-Kashyap),
3. Direct Classification from training set - k-Nearest Neighbour (k=1,3, and 5), and
4. Feature Reduction (PCA).

P(error)

Classifiers	P(error)
Bayesian Classifier	0.094
Ho-Kashyap	0.1896
1-NNR	0.1325
3-NNR	0.1138
5-NNR	0.1055
PCA	0.1763

From the above table, we can see that Ho-Kashyap Hyperplanar Classification and PCA is does equally bad at around 17.5-19% error rate. This is to be expected as PCA is a dimensional reduction, and you lose the information because of this reduction. But it is very fast to compute as the features and reduced and number of computations becomes lower. Hyperplanar Classifier like Ho-Kashyap works iteratively trying to find the best hyperplanar boundary between classes. A hyperplane cannot be a perfect fit for non – linear data points. A 'best' fit of hyperplane can only be estimated after many number of iterations. It becomes computationally expensive to iteratively estimate the hyperplane.

The Nearest Neighbour Classifiers work very well compared to other classifiers here, but is very computationally expensive. We need to compute distances for each sample of the test set with every single sample of the training set to find which is the closest data point in the training set for a given test sample. The Bayesian Classifer works well for a Gaussain type of PDFs.

APPENDIX A:

MATLAB CODE:

assign2.m

```
clear all

C = 3;

fp = fopen('train_sp2017_v19');
A = textscan(fp,'%f%f%f%f');
[~] = fclose(fp);
for i = 1:size(A,2)
H(:,i) = A{1,i}(1:end, :);
end
clear A fp

fp = fopen('test_sp2017_v19');
A = textscan(fp,'%f%f%f%f');
[~] = fclose(fp);
for i = 1:size(A,2)
St(:,i) = A{1,i}(1:end, :);
end
clear A fp

trueVal = [3, 1, 2, 3, 2, 1];
fp = fopen('sprabha-classified-takehome1.txt');
A = textscan(fp,'%d');
[~] = fclose(fp);
predicted = A{1,1};
clear A fp

conf1 = confMat(predicted, trueVal);
x = pca1(H,St,2);
fx = fopen('sprabha-classified-pca.txt', 'wt');
for i = 1 : length(x)
        fprintf(fx, '%d\n', x(i));
end
[~] = fclose(fx);
confPCA = confMat(x,trueVal);

y = hoKashyap(H,St,1);
confHoK = confMat(y, trueVal);
fx = fopen('sprabha-classified-hoKayshap.txt', 'wt');
for i = 1 : length(y)
        fprintf(fx, '%d\n', y(i));
end
[~] = fclose(fx);

[x1,x3,x5] = kNNR1(H, St);
confNNR1 = confMat(x1, trueVal);
confNNR3 = confMat(x3, trueVal);
confNNR5 = confMat(x5, trueVal);
fx = fopen('sprabha-classified-nnr1.txt', 'wt');
fy = fopen('sprabha-classified-nn3.txt', 'wt');
fz = fopen('sprabha-classified-nnr5.txt', 'wt');
for i = 1 : length(x1)
```

```
        fprintf(fx, '%d\n', x1(i));
        fprintf(fy, '%d\n', x3(i));
        fprintf(fz, '%d\n', x5(i));
end
[~] = fclose(fx);
[~] = fclose(fy);
[~] = fclose(fz);
```

confMat.m

```
function A = confMat(predicted,trueVal)
n = length(trueVal);
c = max(trueVal);
A = zeros(c,c);
for i = 0:length(predicted)-1
    A(trueVal(mod(i,n)+1), predicted(i+1)) = A(trueVal(mod(i,n)+1),
predicted(i+1)) + 1;
end
end
```

pca1.m

```
function x = pca1(H,St,dim)
C = 3;
m = mean(H);
H1 = bsxfun(@minus, H,m);
sigmaY = cov(H1);
[~,Z] = eig(sigmaY);
[~,i] = sort(diag(Z),'descend');
H3 = bsxfun(@plus, H1(:,i(1:dim)), m(i(1:dim)));
St2 = St(:,i(1:dim));
x = bayesian(H3,St2,C);
end
```

bayesian.m

```
function predict = bayesian(H,xi,C)
totalSize = size(H,1);
sizeC = totalSize/C;
N = 0.9 * sizeC;
dim = size(H,2);

for i = 1:C
    starting = ((i-1)*sizeC)+1;
    ending = ((i-1)*sizeC)+N;
    w{1,i} = H(starting:ending,:);
end
clear starting ending

%Maximum Likelihood Estimation
for i = 1:C
MLEm{1,i} = sum(w{1,i})/N;
MLEc{1,i} = cov(w{1,i});
end

sig0 = eye(dim);
mu0 = zeros(1,dim)';
```

```
% Bayesian Parameter Estimation
for i = 1:C
   b = (1/N) * MLEc{1,i};
   a = inv(sig0 + b);
   sign{1,i} = sig0 * a * b;
   mu{1,i} = ((sig0 * a * MLEm{1,i}') + (b * a * mu0))';
end
clear a b

%Discriminant Function g(i)
for i = 1 : size(xi, 1)
    for j = 1 : C
        mahal_dist = ((xi(i,:)-mu{1,j}) * inv(sign{1,j}) * (xi(i,:)-
mu{1,j})');
        logc = log(det(sign{1,j}));
        g(i,j) = -(1/2) * (mahal_dist + logc);
    end

    if ((g(i, 1) >= g(i, 2)) && (g(i, 1) >= g(i, 3)))
        predict(i) = 1;
    elseif ((g(i, 2) >= g(i, 1)) && (g(i, 2) >= g(i, 3)))
        predict(i) = 2;
    else
        predict(i) = 3;
    end
end
end
```

hoKashyap.m

```
function x = hoKashyap(Hi,St,alp)
C = 3;
nC = size(Hi,1);
dim = size(Hi,2);
Hi(:,dim+1) = ones(1,nC);
St(:,dim+1) = ones(1,nC);
nC = nC/C;
for i = 0 : C-1
    J{1,i+1} = Hi((i*nC)+1: (i+1)*nC, :);
end

a = 1;
for i = 1:C
    for j = i+1:C
        k = 0;
        A = [J{1,i};-J{1,j}];
        AA = [J{1,i};J{1,j}];
        b = ones(size(A,1),1);
        w = inv(A'*A)*A'*b;
        while true
            e = A*w-b;
            b1 = b + alp*(e+abs(e));
            w = inv(A'*A)*A'*b;
            if k>100 | e>=0 | b1==b
                break;
            else
                b = b1;
            end
            k = k + 1;
```

```
        end
        omega{1,a} = w;
        a = a+1;
    end
end

for t = 1 : size(St,1)
    g1 = omega{1,1}'*St(t,:)';
    g2 = omega{1,2}'*St(t,:)';
    g3 = omega{1,3}'*St(t,:)';
    if g1 > 0 && g2 > 0
        x(t) = 1;
    elseif g1<0 && g3>0
        x(t) = 2;
    else
        x(t) = 3;
end
end
```

kNNR1.m

```
function [x1,x3,x5] = kNNR1(H, St)
for i = 1 : size(St,1)
    cc = (bsxfun(@minus,H,St(i,:))).^2;
    disti = sqrt(sum(cc,2));
    [~,ind] = sort(disti);
    nnr5 = ind(1:5);
    for j = 1:5
        if nnr5(j) <= 5000
            nnr5Class(j) = 1;
        elseif nnr5(j) <= 10000
            nnr5Class(j) = 2;
        else
            nnr5Class(j) = 3;
        end
    end
    x5(i) = knnDecision(nnr5Class);
    x3(i) = knnDecision(nnr5Class(1:3));
    x1(i) = nnr5Class(1);
end
end
```

knnDecision.m

```
function x = knnDecision(nnr)
    order = [1, 2, 3];
    counts = [sum(nnr==1), sum(nnr==2), sum(nnr==3)];
    [maxi, ii] = max(counts);
    if sum(counts == maxi)>1
        ti = order(counts == maxi);
        ty = logical(zeros(1,length(nnr)));
        for f = 1: length(ti)
            ty = ty | (nnr==ti(f));
        end
        x = nnr(find(ty,1));
    else
        x = ii;
    end
```