Pattern Recognition

Mini-Project #2

Instructor: Dr. Yazdi

St: Hamze Ghaedi

Stn:9831419

Aban 99

Abstract

This project includes two problem, the first one is focused on implementing a classifier for MNIST handwritten digit dataset using Bayesian Decision theory and Gaussian Parameter Model.

I separate the first problem into 4-steps as follows:

1) Model definition
2) Estimate model parameters
3) Define discriminant function and address classification problem
4) Test the classifier and evaluation

The second problem is about MAP Detector, finding some parameters and proof some relations.

Problem #1

Model Definition

Our main patterns are 28*28 images of handwritten digits available at MNIST dataset. But in this project, we have been given a dataset of 5000 samples each containing 62 extracted features along with their labels for training and 2500 samples for test. So, our goal is to classify these samples into 10 classes (digits (0-9))

It is assumed that the samples are normally distributed so the class conditional density for i^{th} class would be:

$$p(x|w_i) = \frac{\exp\left(-\frac{(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}{2}\right)}{(2\pi)^{\frac{d}{2}}|\Sigma_i|^{\frac{1}{2}}}$$

Where

$$\Sigma_i: covariance\ matrix\ of\ i^{th} class$$

$$\mu_i: Mean\ of\ i^{th} class$$

$$d: dimension\ of\ samples$$

Parameter Estimation

In the previous section, we define a model for our problem. Our model has some unknown parameters (means and covariance matrices) that must be estimated.

Using ML-Estimator to finding the model parameters gives following results:

$$\mu_i = \frac{1}{n}\Sigma_1^n x_j \qquad (all\ x\ belongs\ to\ i^{th} class)$$

$$\Sigma_i = \frac{1}{n}\Sigma(x-\mu_i)(x-\mu_i)^T \quad (for\ all\ x\ belongs\ to\ i^{th} class1)$$

Where n in the number of samples in class i

the above formula for Σ_i is biased, which means it doesn't approach to the exact covariances if the number of samples approaches infinity.

It can be proved that dividing Σ_i by (n - 1) instead of n leaves the covariance matrix unbiased.

In order to estimating the priori probabilities, we count the number of samples per each class and dividing it by the total number of samples. Thus,

$$p(w_i) = \frac{number\ of\ samples\ in\ class\ i}{total\ number\ of\ samples}$$

Discriminator function

Given class conditional densities and priori probabilities for classes, we can compute the posteriori probability for each class as follow:

$$p(w_i|x) = \frac{p(x|w_i)p(w_i)}{p(x)}$$

Our decision rule based on Bayesian decision theory becomes:

$$if\ p(w_i|x) > p\left(w_j|x\right)\ \ for\ all\ j \neq i\ then\ choose\ w_i$$

But sometimes it is possible to make the classifier simpler. Since applying a monotonically increasing function to both side of an inequality leaves it unaffected, in our gaussian model, we can define a function $g_i(x) = lnp(w_i|x)$ so the decision rule becomes:

$$if\ g_i(x) > g_j(x)\ \ for\ all\ j \neq i\ then\ choose\ w_i$$

such a function g_i is called the Discriminator function. After some algebraic calculation we'll reach the following formula for g_i

$$g_i(x) = x^T W_i x + w_i^T x + w_{io}$$

Where:

$$W_i = -\frac{1}{2}\Sigma^{-1}$$

$$w_i = \Sigma^{-1}\mu_i$$

$$W_{io} = -\frac{1}{2}\mu^T\Sigma^{-1}\mu_i - \frac{1}{2}\ln|\Sigma^{-1}| + lnp(w_i)$$

Singular Covariance matrix

So, our discriminator function needs the inverse of the covariance matrices. But, estimating the covariance matrices based on few samples, usually, leads to a singular matrix (the estimated covariance matrix is not invertible)

I choose the Maximum Entropy Covariance Estimator method to address this problem.

Based on this approach:

First, we compute the mixture of covariances matrix which is:

$$\Sigma_p = \text{average of all } \Sigma_\text{i} s$$

Then

Finding ϕ which is eigen vectors of $\Sigma_i + \Sigma_p$

Compute

$$diag\left(\phi^T \Sigma_p \phi\right) = \left[\lambda_{p1}, \lambda_{p2}, \dots, \lambda_{pk}\right] I$$

$$diag(\phi^T \Sigma_i \phi) = [\lambda_{i1}, \lambda_{i2}, \dots, \lambda_{ik}] I$$

The form

$$\Sigma_{new} = [\max\left(\lambda_{p1}, \lambda_{i1}\right), \max\left(\lambda_{p2}, 2\right), \dots] I$$

Then form

$$\Sigma_{mecs} = \phi \Sigma_{new} \phi^T$$

And use the $\Sigma_{mecs}\ instead\ of\ \Sigma_i$

Model Evaluation

We test the model against the 2500 new samples and compute the accuracy criteria using following formula:

$$acc = \frac{number\ of\ true\ prediction}{total\ number\ of\ samples}$$

Also, we define the confusion matrix as follows:

$$c_{ij} = number\ of\ samples\ from\ class\ i\ incorrectly\ assigned\ to\ class\ j$$

The discussed model reaches the overall accuracy of 86.7%

Conclusion

we adopt a parameter model of the form of the gaussian normal for our patterns and such a model, gained the overall accuracy about 87%, so it seems our assumptions is fairly enough and the designed parameter model is fit to the patterns. The method discussed is an example of parametric model method to dealing with pattern recognition problems. But there is no general way to find a parameter model for a given problem, Non-parametric models tackle such situations.

Problem #2

$$p(D) = \int_{-\infty}^{\infty} p(D|\mu)p(\mu)d\mu = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N \left(\frac{1}{\sigma_0\sqrt{2\pi}}\right) \int_{-\infty}^{\infty} e^{\left(-\left(\frac{\Sigma(x_i-\mu)^2}{2\sigma}\right)-\frac{(\mu-\mu_0)^2}{2\sigma_0}\right)} d\mu$$

$$= \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N \left(\frac{1}{\sigma_0\sqrt{2\pi}}\right) \mathrm{e}^{-\frac{\sigma_0 \Sigma x_i^2+\sigma\mu_0^2}{2\sigma^2\ \sigma_0^2}} \int_{-\infty}^{\infty} e^{-\frac{\sigma_0^2 N\mu^2+\sigma\mu^2-2\sigma_0^2\mu\Sigma x_i-2\sigma\mu_0\mu}{2\sigma^2\sigma_0^2}}$$

$$= K \int_{-\infty}^{\infty} e^{-\left(\frac{A\left(\mu-\left(\frac{B}{A}\right)\right)^2}{2\sigma_0^2\sigma^2}\right)} d\mu$$

Where

$$A = \sigma_0^2 N + \sigma^2$$

$$B = \sigma_0^2 \Sigma x_i + \sigma^2\mu_0$$

$$K = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N \left(\frac{1}{\sigma_0\sqrt{2\pi}}\right) \mathrm{e}^{-\frac{\sigma_0 \Sigma x_i^2+\sigma\mu_0^2}{2\sigma^2\ \sigma_0^2}}\ e^{-\left(\frac{B^2}{2A\sigma^2\sigma_0^2}\right)}$$

We know that:

$$\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} e\hat{\ }(-\frac{(x-m)^2}{2\sigma^2})\, d\mu = 1$$

So,

$$\int_{-\infty}^{\infty} e^{-\left(\frac{A\left(\mu-\left(\frac{B}{A}\right)\right)^2}{2\sigma_0^2\sigma^2}\right)} d\mu = \left(\frac{\sigma\sigma_0}{\sqrt{A}}\right)\sqrt{2\pi}$$

$$p(D) = K\left(\frac{\sigma\sigma_0}{\sqrt{A}}\right)\sqrt{2\pi}$$

$$p(\mu|D) = \frac{p(D|\mu)p(\mu)}{p(D)} = \frac{\left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N \left(\frac{1}{\sigma_0\sqrt{2\pi}}\right)}{p(D)} e^{-\left(\frac{\Sigma(x_i-\mu)^2}{2\sigma^2}\right)-\left(\frac{(\mu-\mu_0)^2}{2\sigma^2}\right)}$$

$$= \frac{\left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N \left(\frac{1}{\sigma_0\sqrt{2\pi}}\right)}{p(D)} e^{-\frac{\sigma_0 \Sigma x_i^2+\sigma^2\mu_0^2}{2\sigma_0^2\sigma^2}} * e^{-\frac{(N\sigma_0^2+\sigma^2)\mu^2-(2\sigma^2\mu_0+2\sigma_0^2\Sigma x_i)\mu}{2\sigma^2\sigma_0^2}}$$

$$= \alpha' e^{-\frac{A\mu^2-2B\mu}{2\sigma^2\sigma_0^2}} = \alpha'\left(\sqrt{2\pi\sigma_N^2}\right)\left(\frac{1}{\sqrt{2\pi\sigma_N^2}}\right)\exp\left(\frac{\left(\mu-\left(\frac{B}{A}\right)\right)^2}{2\left(\frac{\sigma\sigma_0}{\sqrt{A}}\right)^2}\right)$$

So,

$$\sigma_N^2 = \frac{\sigma^2\sigma_0^2}{A} = \frac{\sigma^2\sigma_0^2}{N\sigma_0^2+\sigma^2}$$

$$\mu_N = \frac{B}{A} = \frac{\sigma_0^2\Sigma x_i + \sigma^2\mu_0}{N\sigma_0^2+\sigma^2} = \frac{N\sigma_0^2}{N\sigma_0^2+\sigma^2}\bar{\mu} + \frac{\sigma^2}{N\sigma_0^2+\sigma^2}\mu_0$$

$$\bar{\mu} = \frac{1}{N}\Sigma_{i=1}^N x_i$$

$$\alpha = \alpha'\left(\sqrt{2\pi\sigma_N^2}\right) = \frac{\left(\left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N\left(\frac{1}{\sigma_0\sqrt{2\pi}}\right)\right)e^{-\frac{\sigma_0\Sigma x_i^2+\sigma^2\mu_0^2}{2\sigma_0^2\sigma^2}}}{\left(\left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N\left(\frac{1}{\sigma_0\sqrt{2\pi}}\right)e^{-\frac{\sigma_0\Sigma x_i^2+\sigma\mu_0^2}{2\sigma^2\ \sigma_0^2}}\ e^{-\left(\frac{B^2}{2A\sigma^2\sigma_0^2}\right)}\right)\left(\frac{\sigma\sigma_0}{\sqrt{A}}\right)\sqrt{2\pi}}\left(\left(\frac{\sigma_0\sigma}{A}\right)\sqrt{2\pi}\right)$$

$$= e^{\frac{B^2}{2A\sigma^2\sigma_0^2}} = e^{\left(\frac{1}{2}\right)\left(\frac{B}{A}\right)^2\left(\frac{A}{\sigma_0\sigma}\right)^2\left(\frac{1}{A}\right)} = e^{\frac{1}{2A}\left(\frac{\mu_N}{\sigma_N}\right)^2}$$

Appendix #1

MATLAB script for estimating Priories, Means and Covariances:

```
function [priories,means,covariances] = params_estimator(x_train,y_train,n_classes)

    %---------Specs----------------
    sz = size(x_train);
    n_features = sz(2); %input dimension (or number of input features)
    n_samples = sz(1); %number of samples
    %------------------------------


    %-----------Declaration--------
    frequencies = zeros(n_classes,1); %frequency of occurrence for each class
    means = zeros(n_classes,n_features);
    covariances = zeros(62,62,10);
    priories = zeros(n_classes,1);
    %------------------------------


    %-------Calculate Priories and means-------------------------

    %MATLAB arrays indexed from 1 (Not 0). In order to map class labels
    %to arrays indices, one added to each sample's label!
    %thus, frequencies(1) is the number of samples of class 0 and so on

    for i=1:n_samples

        c = y_train(i,1) + 1; %corresponding class(label) of i'th sample + 1
        frequencies(c) = frequencies(c) + 1;
        means(c,:) = means(c,:) + x_train(i,:);

    end

    priories =  frequencies ./ n_samples;
    means = means ./ frequencies;
    %-----------------------------------------------------------------

    %-------------------Calculate covariance matricies---------------

    for i=1:n_samples
        c = y_train(i,1)+1;
        covariances(:,:,c) = covariances(:,:,c) +...
            (x_train(i,:)-means(c))'*(x_train(i,:) - means(c));
    end

    for i =1:10
        covariances(:,:,i) = covariances(:,:,i) ./ (frequencies(i) - 1);
    end
    %------------------------------------------------------------------

end
```

MATLAB script for computing Discriminator parameters

```
function [W,w,w_0] = disc_params_calculator(priories,means,covariances)

    n_classes = size(priories);

    inv_covariances = zeros(62,62,10);
   % mecs = max_ent_cov_estimator(covariances);

    %-------------Inverting covariance matrices-------
    for i=1:n_classes
        inv_covariances(:,:,i) = inv(covariances(:,:,i));
    end
    %--------------------------------------------------

    %----------------------------------------------------------------------
    %-----------Computing Bayes Discriminator parameters-------------------
    %----------------------------------------------------------------------
    W = (-0.5) .* inv_covariances;

    for i=1:n_classes
        w(:,i) = inv_covariances(:,:,i) * means(i,:)';
    end

    %-------Thresholds(or biases)----
    for i=1:n_classes
        w_0(i) = (-0.5)*means(i,:)*w(:,i)-(0.5)*log(det(inv_covariances(:,:,i)))+log(priories(i));
    end


end
```

MATLAB script for Discriminator Function

```
function [g] = discriminator(W,w,w_0,x)
    g = zeros(10,1);
    for i=1:10
        g(i) = x * W(:,:,i) * x' + w(:,i)'*x' + w_0(i);
    end
end
```

MATLAB script for Maximum Entropy Covariance Estimation

```
%----------Maximum Entropy Covariance Estimator---------------
function [mecs] = max_ent_cov_estimator(covariances)

    sz = size(covariances);
    cov_p = zeros(sz(1));

    for i=1:sz(3)
       cov_p = cov_p + covariances(:,:,i);
    end
    cov_p = cov_p ./ sz(3); %mixture covariance matrix (covariances pool)

    mecs = zeros(62,62,10);
    for i=1:10
       [evec,eval] = eig(covariances(:,:,i) + cov_p);
       d_i = evec' * covariances(:,:,i) * evec * eye(62,62);
       d_p = evec' * cov_p * evec * eye(62,62);
       mecs(:,:,i) = evec * max(d_i,d_p) * evec';
    end

end
```

MATLAB script for Classifier:

```
function [c] = classifier(g)
     [v,c] = max(g); %c is argmax(g)
end
```

MATLAB script for Testing and evaluating the Model:

```
%-----------Load data----------------------------
x_train = load('mnist\Train_Data.csv');
y_train = load('mnist\Train_labels.csv');
x_test = load('mnist\Test_Data.csv');
y_test = load('mnist\Test_labels.csv');
%-----------------------------------------------


%----------------Specs---------------------------
sz = size(x_train);
n_training_samples = sz(1);
[n_test_samples,te] = size(y_test);
n_features = sz(2);
n_classes = 10;
%-----------------------------------------------

%---------------Model Definition-----------------

%assuming Gaussian distribution for each class
%We need to estimate mean(which is a vector)
%and covaraince matrix for each of the 10 classes
%so we have 20 parameters
%and 10 priori probabilities corresponding to each class

priories = zeros(n_classes,1);
means = zeros(n_features,n_classes);
covariances = zeros(n_features,n_features,n_classes);

%-------------------------------------------------


%---------------Estimate Model Parameters--------------

%our parameter estimator is based on Maximum likelihood

[priories,means,covariances] = params_estimator(x_train,y_train,n_classes);

%----------------------------------------------------

%-----------------Define Discriminator-----------

W = zeros(n_features,n_features,n_classes);
w = zeros(n_features,n_classes);
w_0 = zeros(n_classes,1);

%Calculate Discriminator parameters--------------
covariances = max_ent_cov_estimator(covariances);
[W,w,w_0] = disc_params_calculator(priories,means,covariances);
%------------------------------------------------

%-------------Testing----------------------------

predicted = zeros(2500,1);

for i=1:n_test_samples
    g = discriminator(W,w,w_0,x_test(i,:));
    predicted(i) = classifier(g) - 1;
end

%--------------------------evaluation-----------------
acc = length(find((predicted == y_test))) / n_test_samples;
acc_mat = zeros(10,10);
for i =1:n_test_samples
   acc_mat(y_test(i)+1,predicted(i)+1) =  acc_mat(y_test(i)+1,predicted(i)+1) + 1;
end
%----------------------------------------------------
```