1.

Firstly preprocessing is done for 200000 instances. Nominal attributes are replaced by unique integer values. Assigning unique integer value is based on count of attribute value,more the count max value is assigned. For some attributes one-hot encoding is applied. Continuous attributes are binned while some of them are dropped out from the table.
Age,Capital losses,Capital Gain,Dividend from stock continuous attributes are considered while others are dropped from the table. Continuous attributes are assigned discrete values based on the binning.
Missing values are replaced by highly occurring value in the attribute i.e. mode value.
Probability density function used in Naive Bayes is Gaussian .
Log probability is taken to avoid numerical errors.

Code:

```
function NB_classifier(n)
        D = csvread('train.csv',1,0);
%       TestData = csvread('train_new.csv',1,0);
        Ds = D(randperm(size(D, 1)), :);
        TrainData=Ds(1:n/2,:);
        TestData=Ds(n/2+1:n,:);
        TrainPos = TrainData(TrainData(:, 38)== 1, :);
        TrainNeg = TrainData(TrainData(:, 38)== 0, :);
        Trainlabel = TrainData(:,38);
        xtrain = TrainData(:,1:37);
        TrainPos = TrainPos(:,1:37);
        TrainNeg = TrainNeg(:,1:37);
        MeanPos = mean(TrainPos)';
        MeanNeg = mean(TrainNeg)';
        SigmaPos = cov(TrainPos);
        SigmaNeg = cov(TrainNeg);
        Testlabel = TestData(:,38);
        gTrain = [];
        for i = 1:size(TestData,1)
        inputvector = transpose(TestData(i,1:37));
        PosProb = (-0.5 * (inputvector-MeanPos)' * inv(SigmaPos) * (inputvector-MeanPos)) -
(0.5 *log(abs(det(SigmaPos)))) + log(abs(size(TrainPos,1)/50000));
        NegProb = (-0.5 * (inputvector-MeanNeg)' * inv(SigmaNeg) * (inputvector-MeanNeg))
- (0.5 *log(abs(det(SigmaNeg)))) + log(abs(size(TrainNeg,1)/50000));
        if PosProb >= NegProb
        gTrain(i,1)=1;
        else
        gTrain(i,1)=0;
        end
        end
```

```
    accuracy = mean(double(gTrain == Testlabel) * 100);
    error = mean(double(gTrain ~= Testlabel) * 100);
    disp(accuracy);
    disp(error);
end
```

No of instances	Testing Accuracy	Training Accuracy
50000	63.75	64.14
60000	65.09	65.98
67000	65.17	66.07
70000	63.63	64.05
76000	63.9053	64.7698
80000	60.2875	62.5213
83000	63.1904	64.3452
85000	61.4353	62.30
87000	61.1655	61.98
90000	62.944	64.09

Mean Testing Accuracy : 63.05
Mean Training Accuracy : 64.12

In some cases the matrix is becoming singular due to selecting samples randomly which makes the matrix singular or its determinant value is tending to zero.

2.

Univariate case

$$p(x|D) = \int p(x|\mu)\, p(\mu|D)\, d\mu$$

$$= \int \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left[\left(\frac{\mu-\mu_n}{\sigma_n}\right)^2 \times -\frac{1}{2}\right] d\mu$$

$$= \frac{1}{2\pi\sigma\sigma_n} \exp\left[-\frac{1}{2}\frac{(x-\mu_n)^2}{\sigma^2+\sigma_n^2}\right] f(\sigma,\sigma_n)$$

where $f(\sigma,\sigma_n) = \int \exp\left[-\frac{1}{2}\frac{\sigma^2+\sigma_n^2}{\sigma^2\sigma_n^2}\left(\mu - \frac{\sigma_n^2 x + \sigma^2\mu_n}{\sigma^2+\sigma_n^2}\right)^2\right] d\mu$

This is a function of x hence $p(x|D) \propto \exp\left[-\frac{1}{2}\frac{(x-\mu_n)^2}{\sigma^2+\sigma_n^2}\right]$

~~and~~ which implies $p(x|D)$ is normally distributed with mean μ_n and variance $\sigma^2+\sigma_n^2$

$$p(x|D) \sim N(\mu_n, \sigma^2+\sigma_n^2)$$

Multivariate case

Let us assume that

$$p(x|\mu) \sim N(\mu, \Sigma) \quad \text{and} \quad p(\mu) \sim N(\mu_0, \Sigma_0)$$

where Σ, Σ_0 and μ_0 are assumed to be known.

After observing a set D of n independent samples $x_1, x_2, \ldots x_n$, we use Bayes formula to obtain

$$p(\mu|D) = \alpha \prod_{k=1}^{n} p(x_k|\mu)\, p(\mu)$$

$$= \alpha' \exp\left[-\frac{1}{2}\left(\mu^t\left(n\Sigma^{-1} + \Sigma_0^{-1}\right)\mu - 2\mu^t\left(\Sigma^{-1}\sum_{k=1}^{n} x_k + \Sigma_0^{-1}\mu_0\right)\right)\right]$$

which is of the form

$$p(\mu|D) = \alpha'' \exp\left[-\frac{1}{2}(\mu - \mu_n)^t \Sigma_n^{-1} (\mu - \mu_n)\right]$$

This implies that $p(\mu|D) \sim N(\mu_n, \Sigma_n)$

On equating coefficients, we get

$$\Sigma_n^{-1} = n\Sigma^{-1} + \Sigma_0^{-1} \quad \text{--- (1)}$$

and

$$\Sigma_n^{-1}\mu_n = n\Sigma^{-1}\hat{\mu}_n + \Sigma_0^{-1}\mu_0 \quad \text{--- (2)}$$

where $\hat{\mu}_n = \frac{1}{n}\sum_{k=1}^{k} x_k$

To solve equations (1) & (2), matrix identity is required

$$(A^{-1} + B^{-1})^{-1} = A(A+B)^{-1}B = B(A+B)^{-1}A$$

valid for non-singular matrices A, B

$$\mu_n = \Sigma_0 \left(\Sigma_0 + \frac{1}{n} \Sigma \right)^{-1} \hat{\mu}_n + \frac{1}{n} \Sigma \left(\Sigma_0 + \frac{1}{n} \Sigma \right)^{-1} \mu_0$$

and $$\Sigma_n = \Sigma_0 \left(\Sigma_0 + \frac{1}{n} \Sigma \right)^{-1} \frac{1}{n} \Sigma$$

We know $p(x|D) = \int p(x|\mu)\, p(\mu|D)\, d\mu$

here x can be viewed as sum of two mutually independent random variables, a random vector μ with $p(\mu|D) \sim N(\mu_n, \Sigma_n)$ and an independent random vector y with $p(y) \sim N(0, \Sigma)$

sum of two normally distributed independent vectors is again normally distributed vector whose mean is sum of means and whose covariance matrix is sum of covariance matrices

$$p(x|D) = p(\mu_n, \Sigma + \Sigma_n)$$

3.

Code for PCA:

```
function PCA(k)
	Data = dlmread('dorothea_train.data',' ');
	%Data = csvread('train2.csv');
	x=zeros(800,100000);
	for i=1:size(Data,1)
	for j=1:size(Data(i,:),2)
	if Data(i,j)>0
		x(i,Data(i,j))=1;
	end
	end
	end
	disp(size(x));
	x=sparse(x);
	[N,M]=size(x);
	nm=mean(x);
	x=x-repmat(nm,N,1);
	covariance= x * x';
	%k=500;
```

```
        [eigvectors,eigvalues]=eigs(covariance,k);
        eigvalues=diag(eigvalues);
        M=x'*eigvectors;
        final_x=x*M;
        TestLabel = csvread('train_labels.csv');
        TrainPos = final_x(TestLabel(:,1)== 1, :);
        TrainNeg = final_x(TestLabel(:,1)== -1, :);
        MeanPos = mean(TrainPos)';
        MeanNeg = mean(TrainNeg)';
        SigmaPos = cov(TrainPos);
        SigmaNeg = cov(TrainNeg);
        gTrain = [];
        for i = 1:size(final_x,1)
        inputvector = transpose(final_x(i,1:k));
        PosProb = (-0.5 * (inputvector-MeanPos)' * inv(SigmaPos) * (inputvector-MeanPos)) -
(0.5 *log(abs(det(SigmaPos)))) +
log(abs(size(TrainPos,1)/(size(TrainPos,1)+size(TrainNeg,1))));
        NegProb = (-0.5 * (inputvector-MeanNeg)' * inv(SigmaNeg) * (inputvector-MeanNeg))
- (0.5 *log(abs(det(SigmaNeg)))) +
log(abs(size(TrainNeg,1)/(size(TrainPos,1)+size(TrainNeg,1))));
        if PosProb >= NegProb
        gTrain(i,1)=1;
        else
        gTrain(i,1)=-1;
        end
        end
        accuracy = mean(double(gTrain == TestLabel) * 100);
        error = mean(double(gTrain ~= TestLabel) * 100);
        disp(accuracy);
        disp(error);
end
```

Value of K	Accuracy
100	43.35
500	50.56
800	49.12

Code for LDA:

```
function LDA(k)
        Data = dlmread('dorothea_train.data',' ');
```

```
%Data = csvread('train2.csv');
x=zeros(800,100000);
for i=1:size(Data,1)
for j=1:size(Data(i,:),2)
if Data(i,j)>0
        x(i,Data(i,j))=1;
end
end
end
x=sparse(x);
[N,M]=size(x);
nm=mean(x);
x=x-repmat(nm,N,1);
covariance= x * x';
%k=500;
[eigvectors,eigvalues]=eigs(covariance,k);
eigvalues=diag(eigvalues);
M=x'*eigvectors;
final_x=x*M;
TestLabel = csvread('train_labels.csv');
TrainPos = final_x(TestLabel(:,1)== 1, :);
TrainNeg = final_x(TestLabel(:,1)== -1, :);
MeanPos = mean(TrainPos)';
MeanNeg = mean(TrainNeg)';
SigmaPos = cov(TrainPos);
SigmaNeg = cov(TrainNeg);
Sigma=SigmaPos+SigmaNeg;
opt_w=inv(Sigma)*(MeanPos-MeanNeg);
y=[];
for i=1:size(final_x,1)
temp= transpose(final_x(i,1:500));
y(i,1)=opt_w' * temp;
end
inputvector=y;
TrainPos = y(TestLabel(:,1)== 1, :);
TrainNeg = y(TestLabel(:,1)== -1, :);
MeanPos = mean(TrainPos)';
MeanNeg = mean(TrainNeg)';
SigmaPos = cov(TrainPos);
SigmaNeg = cov(TrainNeg);
for i=1:size(y,1)
inputvector=y(i,:);
PosProb = (-0.5 * (inputvector-MeanPos)' * inv(SigmaPos) * (inputvector-MeanPos)) -
(0.5 *log(abs(det(SigmaPos)))) +
log(abs(size(TrainPos,1)/(size(TrainPos,1)+size(TrainNeg,1))));
```

```
        NegProb = (-0.5 * (inputvector-MeanNeg)' * inv(SigmaNeg) * (inputvector-MeanNeg))
- (0.5 *log(abs(det(SigmaNeg)))) +
log(abs(size(TrainNeg,1)/(size(TrainPos,1)+size(TrainNeg,1))));
        if PosProb>=NegProb
        gTrain(i,1)=1;
        else
        gTrain(i,1)=-1;
        end
        end
        accuracy = mean(double(gTrain == TestLabel) * 100);
        error = mean(double(gTrain ~= TestLabel) * 100);
        disp(accuracy);
        disp(error);
end
```

Value of k	Accuracy
100	83.13
500	82.45
800	82.76