CAB420 Machine Learning

Queensland University of Technology

Assignment 2 Report

Group 22

Nathan Armishaw – n9157191

Quang Huy Tran - n10069275

Due: 23/05/20

Table of Contents

[Part A: SVMs and Bayes Classifiers [45 Marks] 2](#_Toc9197962)

[Support Vector Machines [30 Marks] 2](#_Toc9197963)

[Bayes Classifiers [15 Marks] 6](#_Toc9197964)

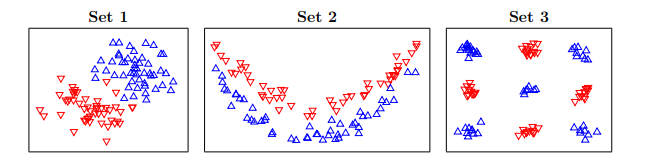
[Part B: PCA & Clustering [45 marks] 9](#_Toc9197965)

[EigenFaces [25 Marks: 5 marks for each section (a) to (e)] 9](#_Toc9197966)

[Clustering [20 Marks: 5 marks for each section (a) to (d)] 13](#_Toc9197967)

# Part A: SVMs and Bayes Classifiers [45 Marks]

### Support Vector Machines [30 Marks]



%% Section A: Support Vector Machines

% Load the datasets

svm\_data = load('data\_ps3\_2.mat');

% Training Data

set1\_train = svm\_data.set1\_train; % Set 1

set2\_train = svm\_data.set2\_train; % Set 2

set3\_train = svm\_data.set3\_train; % Set 3

set4\_train = svm\_data.set4\_train; % Set 4

% Testing Data

set1\_test = svm\_data.set1\_test; % Set 1

set2\_test = svm\_data.set2\_test; % Set 2

set3\_test = svm\_data.set3\_test; % Set 3

set4\_test = svm\_data.set4\_test; % Set 4

%Set constant values

C = 1000;

polyOrder = 2;

SD1 = 1;

SD2 = 1.5;

From these results the best model for datasets 1,2 and 3 were chosen. These were Linear for set 1 because the data was linearly clustered. 2nd order polynomial for set 2 because the 2 classes were separable by a curved boundary. 1 SD gaussian for set 3 because the data was separately clustered. The plots of the decision boundary and test errors for each of these three is shown below. Note that the plots below show the decision boundary on the training data, then the testing result on the test data.

%% Question 1:

% For the first three datasets, consider the linear, second

% order polynomial, Gaussian of standard deviation 1 kernels

%% Part 1:

% Train first 3 datasets, plot the decision boundary and print the test

% errors using svm\_test

% Plot the decision boundary and test errors with the linear model

svm\_test(@Klinear,[],C,set1\_train,set1\_test);

svm\_test(@Klinear,[],C,set2\_train,set2\_test);

svm\_test(@Klinear,[],C,set3\_train,set3\_test);

% Plot the decision boundary and test errors with the 2nd order

% polynomial model

svm\_test(@Kpoly,polyOrder,C,set1\_train,set1\_test);

svm\_test(@Kpoly,polyOrder,C,set2\_train,set2\_test);

svm\_test(@Kpoly,polyOrder,C,set3\_train,set3\_test);

% Plot the decision boundary and test errors with the gaussian model of

% standard deviation 1

svm\_test(@Kgaussian,SD1,C,set1\_train,set1\_test);

svm\_test(@Kgaussian,SD1,C,set2\_train,set2\_test);

svm\_test(@Kgaussian,SD1,C,set3\_train,set3\_test);













%% Part 2

% Train 4th dataset with a linear, polynomial of degree 2 and Gaussian of

% standard deviation 1.5 kernels

% Preallocate TestError and set Kernel

TestError = zeros(3,1);

Kernel = {'Linear';'Polynomial of degree 2';'Gaussian of std 1.5'};

% For the following chunks of code the following process was used: First the

% SVM model was trained on the training data, then the errors between the

% prediction and actual results in the test data was calculated. Then the

% fraction of errors was displayed.

% For linear model

svm\_linear4 = svm\_train(set4\_train,@Klinear,[],C);

y\_linear4 = sign(svm\_discrim\_func(set4\_test.X,svm\_linear4));

errors\_linear = find(y\_linear4 ~= set4\_test.y);

TestError(1) = length(errors\_linear)/length(set4\_test.y);

fprintf('Linear SVM: %g of 4th test examples were misclassified.\n',...

length(errors\_linear)/length(set4\_test.y));

% For polynomial model

svm\_poly4 = svm\_train(set4\_train,@Kpoly,polyOrder,C);

y\_poly4 = sign(svm\_discrim\_func(set4\_test.X,svm\_poly4));

errors\_poly = find(y\_poly4 ~= set4\_test.y);

TestError(2) = length(errors\_poly)/length(set4\_test.y);

fprintf('Polynomial SVM: %g of 4th test examples were misclassified.\n',...

length(errors\_poly)/length(set4\_test.y));

% For gaussian model

svm\_gaussian4 = svm\_train(set4\_train,@Kgaussian,SD2,C);

y\_gaussian4 = sign(svm\_discrim\_func(set4\_test.X,svm\_gaussian4));

errors\_gaussian = find(y\_gaussian4 ~= set4\_test.y);

TestError(3) = length(errors\_gaussian)/length(set4\_test.y);

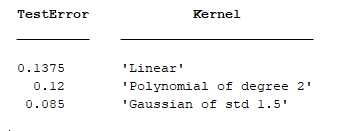
fprintf('Gaussian SVM: %g of 4th test examples were misclassified.\n',...

length(errors\_gaussian)/length(set4\_test.y));

% The test errors of 4th dataset trained on different kernels is as below:

display(table(TestError,Kernel));

This results in the data below. Clearly the gaussian model is the most accurate, followed by the polynomial then the linear.



## Bayes Classifiers [15 Marks]

%% Section A: Bayes Classifiers

% Code is relevant to parts (A) and (B)

% Set the Training and Test data as well as other constants related to them

Xtr1 = [0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1];

Xtr2 = [0,0,0,0,1,1,1,1,0,0,0,1,1,1,1,1];

Ytr = [0,1,1,1,0,1,1,1,0,0,0,0,0,0,1,1];

Xtest = [0 1; 1 0; 1 1];

YtrLength = length(Ytr);

Ytr0Length = length(Ytr(Ytr==0));

Ytr1Length = length(Ytr(Ytr==1));

% Find the probabilities needed to create Joint/naive Bayes classifier

% Find out the percentage occurrence of each possible class(Ytr) value

% to do this divide number of occurrences by the length of the total array

% considered. Repeat this process to calculate all probabilities needed for

% classification

P\_0 = length(Ytr(Ytr==0)) / YtrLength;

P\_1 = length(Ytr(Ytr==1)) / YtrLength;

% Find probabilities for Joint Bayes classifier, these are the percentage

% occurrence rates of an (X1,X2) combination for a specific y value

% P(x1,x2|y). The naming logic specifies the x1 value, x2 value then y

% value.

jbc\_P\_000 = length(Xtr1(Xtr1==0 & Xtr2==0 & Ytr == 0))/Ytr0Length;

jbc\_P\_010 = length(Xtr1(Xtr1==0 & Xtr2==1 & Ytr == 0))/Ytr0Length;

jbc\_P\_100 = length(Xtr1(Xtr1==1 & Xtr2==0 & Ytr == 0))/Ytr0Length;

jbc\_P\_110 = length(Xtr1(Xtr1==1 & Xtr2==1 & Ytr == 0))/Ytr0Length;

jbc\_P\_001 = length(Xtr1(Xtr1==0 & Xtr2==0 & Ytr == 1))/Ytr1Length;

jbc\_P\_011 = length(Xtr1(Xtr1==0 & Xtr2==1 & Ytr == 1))/Ytr1Length;

jbc\_P\_101 = length(Xtr1(Xtr1==1 & Xtr2==0 & Ytr == 1))/Ytr1Length;

jbc\_P\_111 = length(Xtr1(Xtr1==1 & Xtr2==1 & Ytr == 1))/Ytr1Length;

% Find probabilities for naive Bayes classifier, these are the percentage

% occurrence rates of P(x1|y) and P(x2|y). The naming logic specifies the

% x value then y value.

nbc\_x1\_P\_00 = length(Xtr1(Xtr1==0 & Ytr == 0))/Ytr0Length;

nbc\_x1\_P\_10 = length(Xtr1(Xtr1==1 & Ytr == 0))/Ytr0Length;

nbc\_x2\_P\_00 = length(Xtr1(Xtr2==0 & Ytr == 0))/Ytr0Length;

nbc\_x2\_P\_10 = length(Xtr1(Xtr2==1 & Ytr == 0))/Ytr0Length;

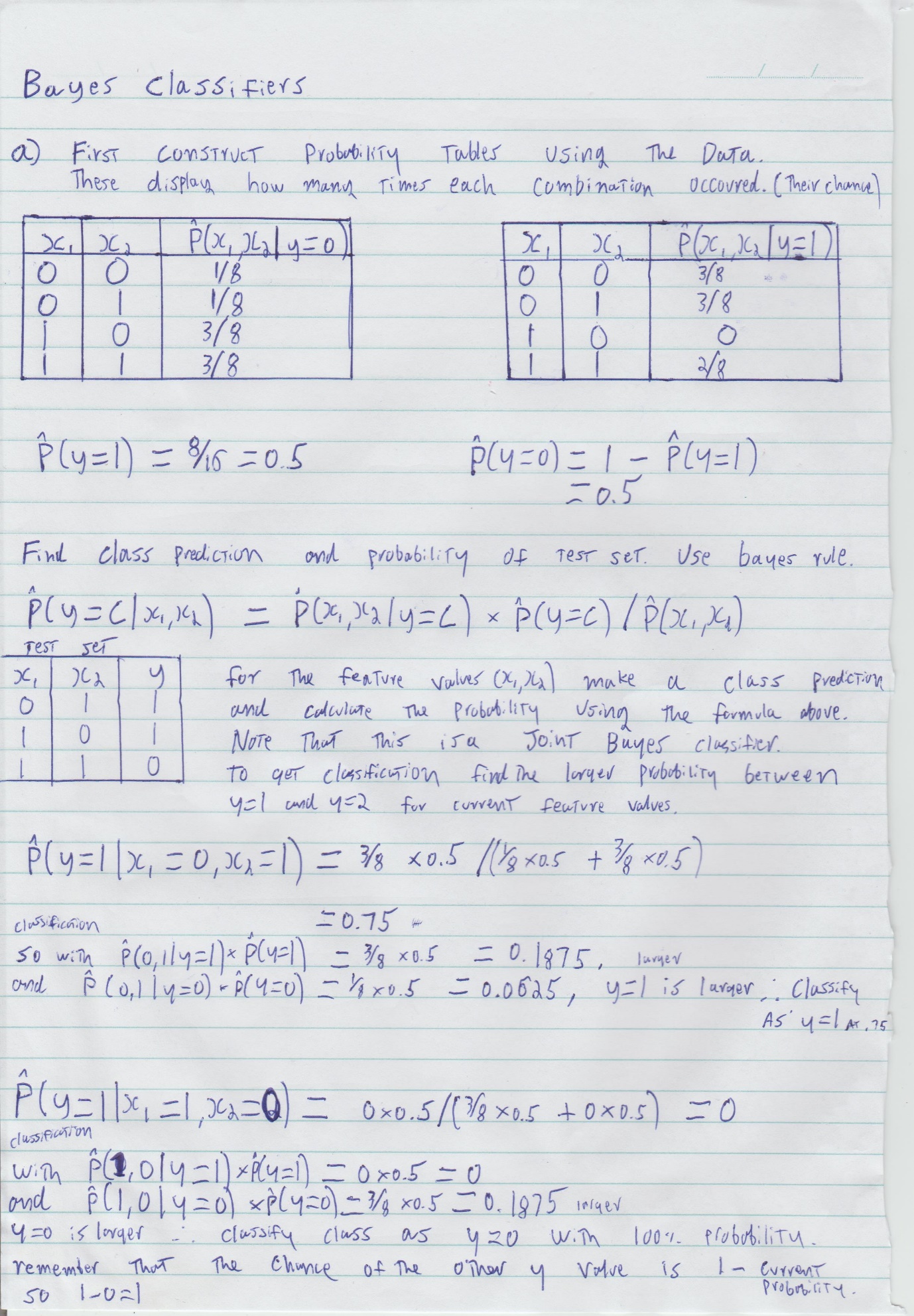
nbc\_x1\_P\_01 = length(Xtr1(Xtr1==0 & Ytr == 1))/Ytr1Length;

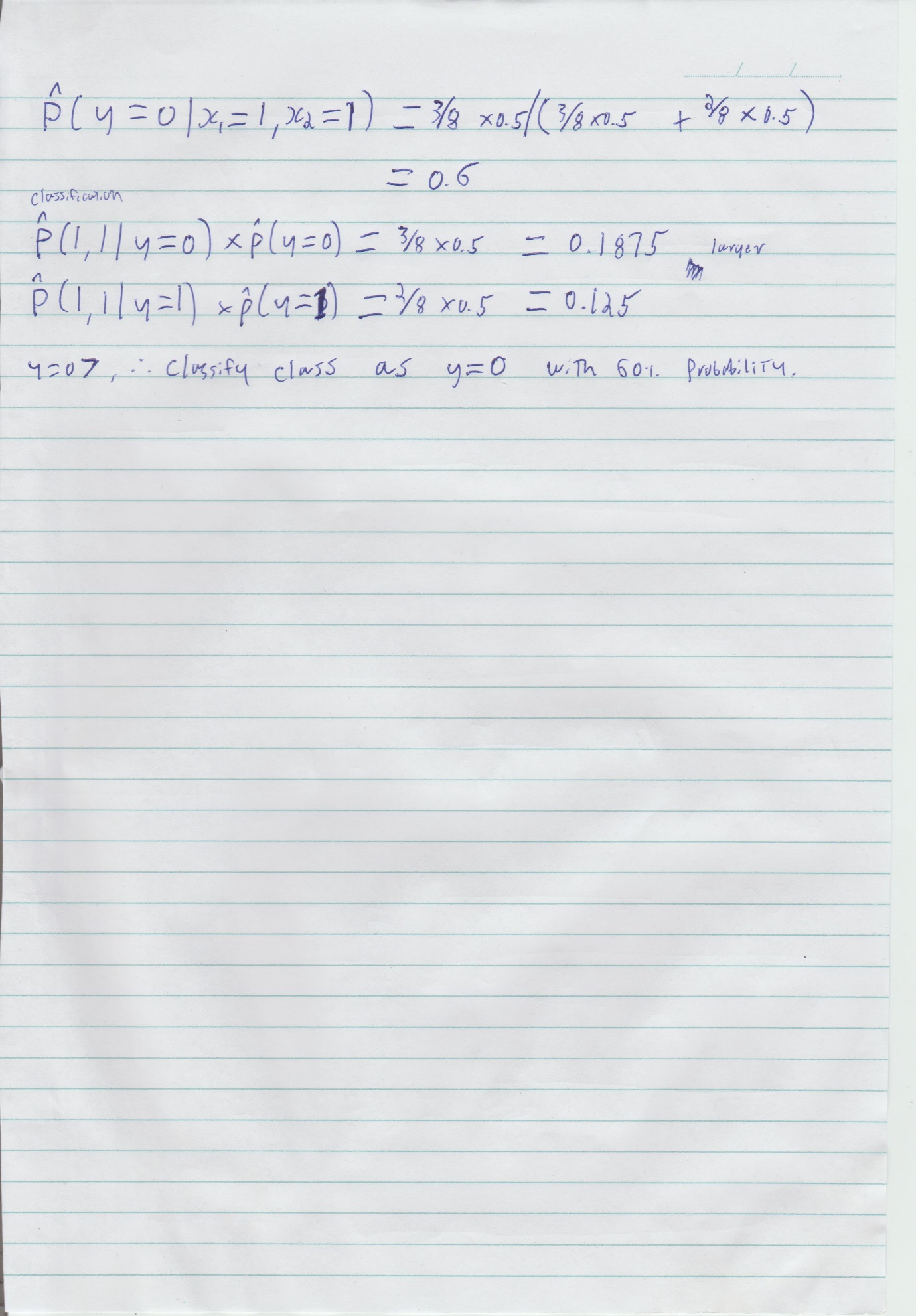
nbc\_x1\_P\_11 = length(Xtr1(Xtr1==1 & Ytr == 1))/Ytr1Length;

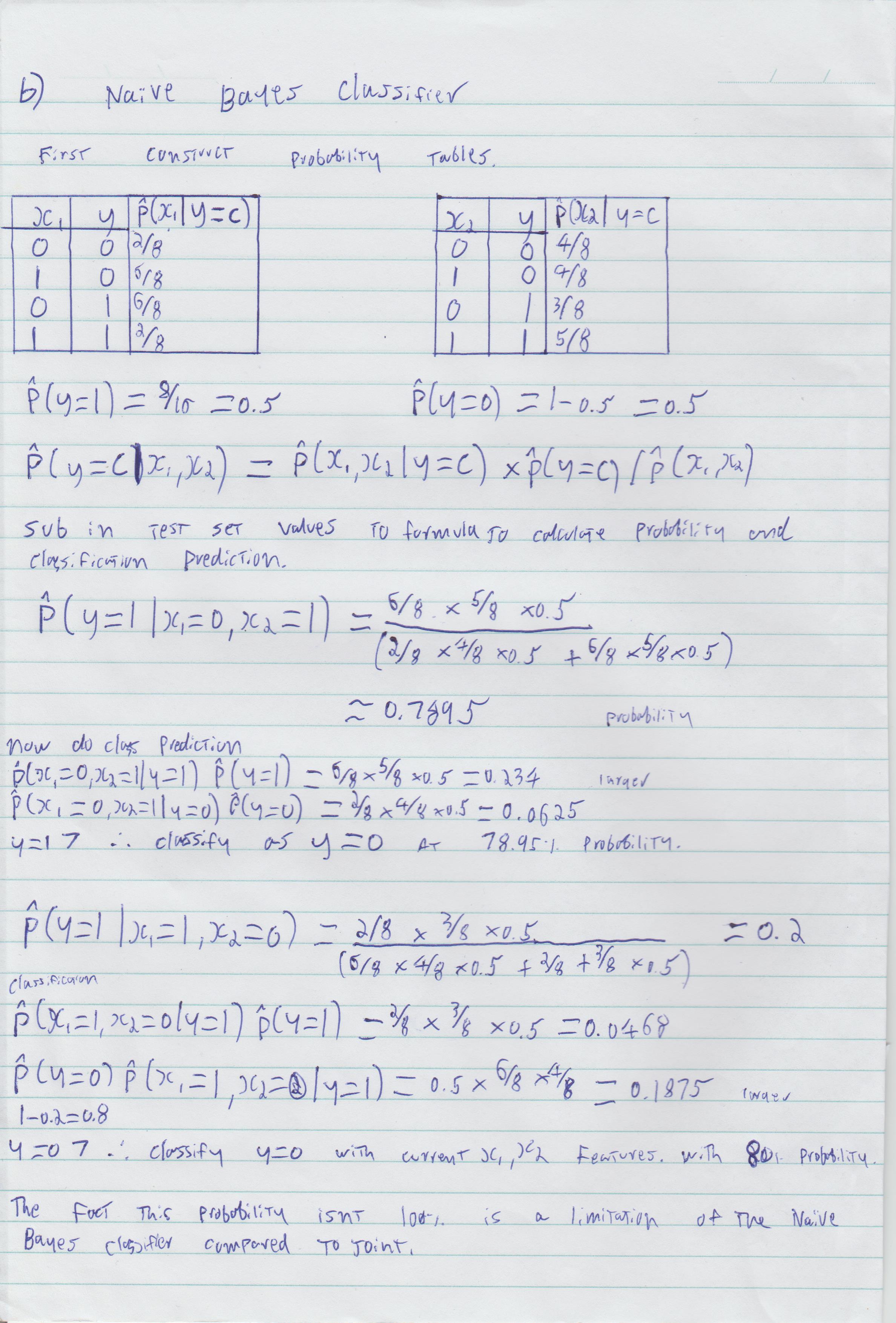
nbc\_x2\_P\_01 = length(Xtr1(Xtr2==0 & Ytr == 1))/Ytr1Length;

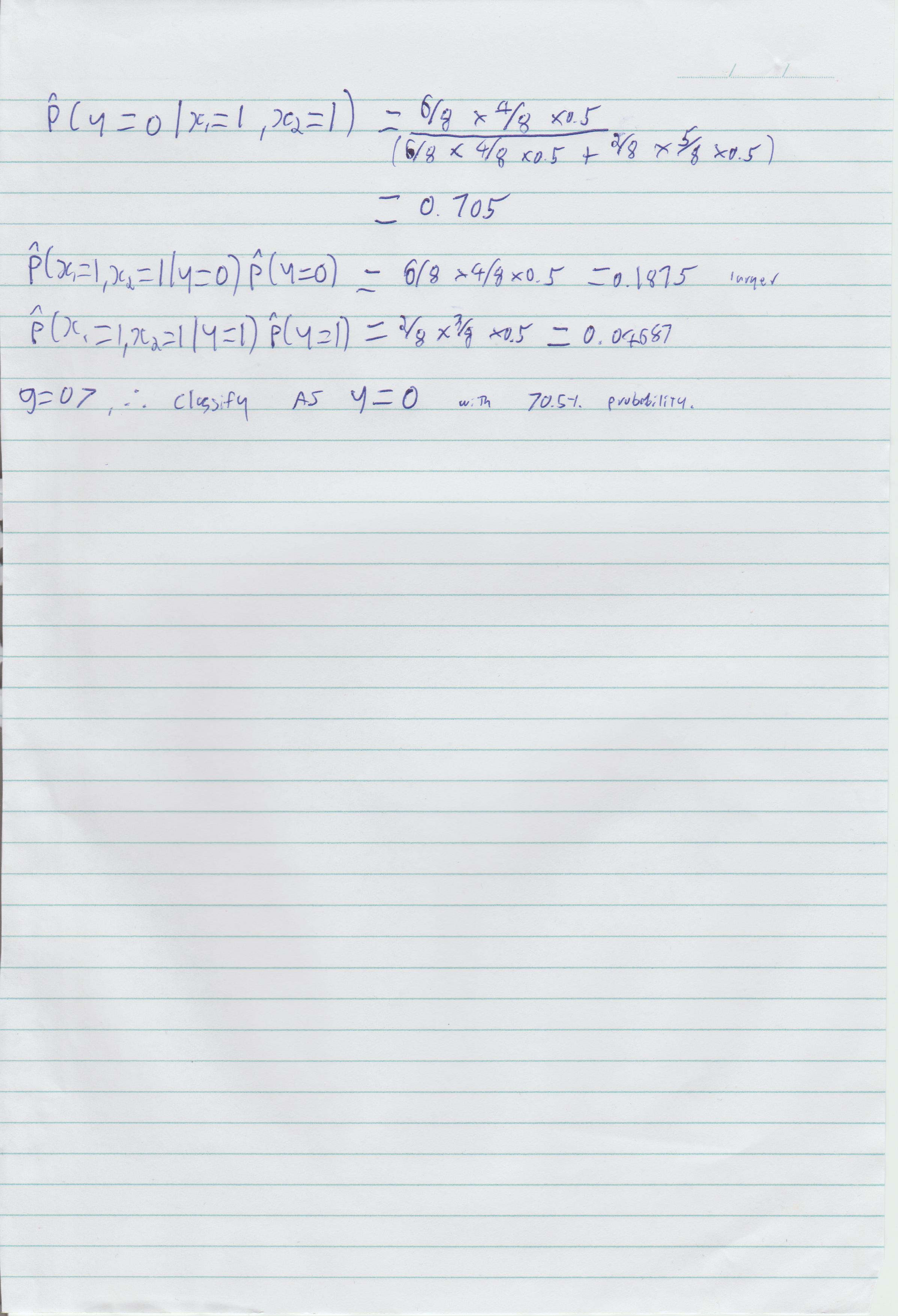
nbc\_x2\_P\_11 = length(Xtr1(Xtr2==1 & Ytr == 1))/Ytr1Length;

Parts A and B have been finished in hand written form as seen below. The data in the tables is taken from the MATLAB results.









# Part B: PCA & Clustering [45 marks]

## EigenFaces [25 Marks: 5 marks for each section (a) to (e)]

%% Section B: PCA

% Eigenfaces - Part A to E

% Load the data and display a few faces

X = load('data/faces.txt');

img = reshape(X(2,:),[24 24]);

imagesc(img); axis square; colormap gray;





%% Part B: Compute the mean square error in SVD's approximation

errors = zeros(1,10); % Store the MSE in SVD's approximation

K = 1:10;

for i = 1:length(K)

[U\_k S\_k V\_k] = svds(X0,K(i));

X0\_svd = U\_k \* S\_k \* V\_k';

mse\_svd = mean(mean((X0 - X0\_svd).^2));

errors(i) = mse\_svd;

end

figure(1);

plot(K,errors);

title('Plot the MSE in SVD approximation against the K');

xlabel('K'); ylabel('MSE');

%% Part A: Subtract the mean of the face images to make the data zero-mean

mean\_X = mean(X);

X0 = X - mean\_X;

% Take the SVD of the data

[U, S, V] = svd(X0);

W = U \* S;



%% Part C: Display a first few principal directions of the data

alpha = 2 \* median(abs(W(:,10))); % Scale factor

direction1 = reshape(mean\_X + alpha \* V(:,10)',[24,24]);

direction2 = reshape(mean\_X - alpha \* V(:,10)',[24,24]);

figure(2);

imagesc(direction1); axis square; colormap gray;

figure(3);

imagesc(direction2); axis square; colormap gray;

This results in the following figures.

%% Part D: Latent Space methods

idx = 15:25; % random indices of data

figure(4); title('Latent Space methods'); hold on; axis ij; colormap(gray);

% find range of coordinates to be plotted

range = max(W(idx,1:2)) - min(W(idx,1:2));

% want 24x24 to be visible

scale = [200 200]./range;

for i=1:length(idx)

imagesc(W(idx(i),1) \* scale(1), W(idx(i),2) \* scale(2), reshape(X(idx(i),:),24,24));

end

{}

The resulting figure is below.





%% Part E: Choose two faces and reconstruct using only K principal directions

K\_recover = [5,10,50]; % Use K principal directions

% Choose random two faces

indices = randperm(size(X0,1));

index1 = indices(1); index2 = indices(2);

img1 = X(index1,:);

img2 = X(index2,:);

figure(5);

title('Construct image using K principal directions');

% Iterate over the 3 k values then reconstruct and plot the image using that k % value, as well as the original image

for i = 1:length(K\_recover)

[U1 S1 V1] = svds(img1,K\_recover(i));

recovered\_img1 = U1 \* S1 \* V1';

subplot(2,4,i);

imagesc(reshape(recovered\_img1,24,24)); axis square; colormap gray;

title([num2str(K\_recover(i))]);

[U2 S2 V2] = svds(img2);

recovered\_img2 = U2 \* S2 \* V2';

subplot(2,4,i+4);

imagesc(reshape(recovered\_img2,24,24)); axis square; colormap gray;

title([num2str(K\_recover(i))]);

end

subplot(2,4,4); imagesc(reshape(img1,24,24)); axis square; colormap gray;

title('Original Image');

subplot(2,4,8); imagesc(reshape(img2,24,24)); axis square; colormap gray;

title('Original Image');

## Clustering [20 Marks: 5 marks for each section (a) to (d)]

%% Part A: Load the usual Iris data with 2 features and plot

iris = load('data/iris.txt'); % Load Iris data

X\_iris = iris(:,1:2); % Use only two first features

figure(6);

plot(X\_iris(:,1),X\_iris(:,2),'ro');

title('Plot of Iris data with 2 first features');



%% Part B: K-Means on the data

% Choose the initialization with the best score

% Run k-means with k = 5 with farthest initialization

figure(7);

title('k-means on the data with k = 5');

[z5 c5 sumd5] = kmeans(X\_iris,5,'farthest',100);

plotClassify2D([],X\_iris,z5);

hold on

plot(c5(:,1),c5(:,2),'kx');

title('K-means K = 5');

% Run k-means with k = 20 with k++ intialization

figure(8);

title('k-means on the data with k = 20');

[z20 c20 sumd20] = kmeans(X\_iris,20,'k++',100);

plotClassify2D([],X\_iris,z20);

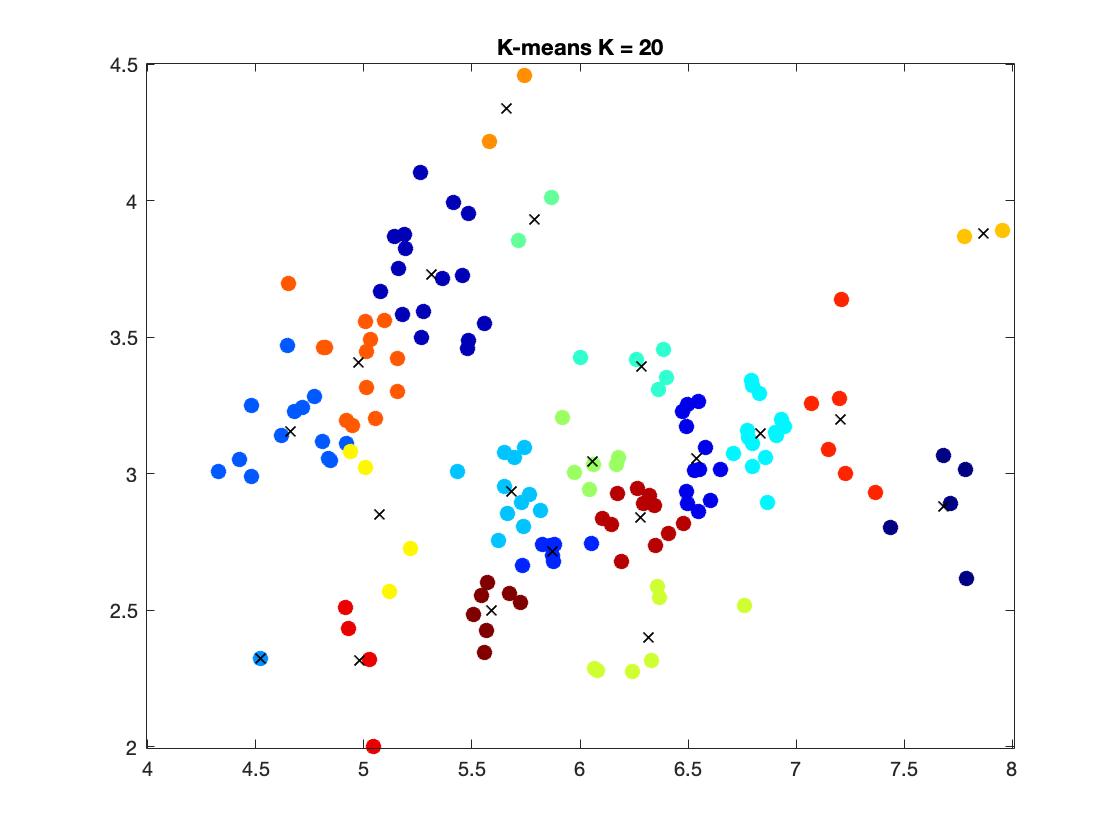
hold on

plot(c20(:,1),c20(:,2),'kx');

title('K-means K = 20');

A close up of a map

Description automatically generated



The resulting plots of using single and complete linkage are demonstrated below.

%% Part C: Agglomerative clustering

% Using single linkage on Iiris data

[z5\_min\_agg join] = agglomCluster(X\_iris,5,'min'); % k = 5

[z20\_min\_agg join] = agglomCluster(X\_iris,20,'min'); % k = 20

figure(9);

subplot(1,2,1);

plotClassify2D([],X\_iris,z5\_min\_agg);

title('K = 5');

subplot(1,2,2);

plotClassify2D([],X\_iris,z20\_min\_agg);

title('K = 20');

% Using complete linkage on Iris data

[z5\_max\_agg join] = agglomCluster(X\_iris,5,'max'); % k = 5

[z20\_max\_agg join] = agglomCluster(X\_iris,20,'max'); % k = 20

figure(10);

subplot(1,2,1);

plotClassify2D([],X\_iris,z5\_max\_agg);

title('K = 5');

subplot(1,2,2);

plotClassify2D([],X\_iris,z20\_max\_agg);

title('K = 20');

In hierarchical clustering, clusters have a tree like structure or parent child relationship. Here, the two most similar clusters are combine together and continue to combine until all objects are in the same cluster

In single linkage clustering, we merge in each step the two clusters with the smallest minimum pairwise distance

In complete linkage clustering, we merge in each step the two clusters with the smallest maximum pairwise distance

Therefore, it is observed from the graphs that the data points are distributed equally on clusters created using complete linkage compared to single linkage. In single linkage clustering, some clusters are very large.

Unlike hierarchical clustering mentioned above, k-means algorithm assigns each point to the cluster whose center (called centroid) is nearest. The center is the average of all the points in the cluster – that is, its coordinates are the arithmetic mean of each dimension separately over all points in the cluster.

A close up of a map

Description automatically generatedA close up of a map

Description automatically generated



%% Part D: Run the EM Gaussian Mixture Model

% Use doPlot = true in emCluster to observe the evolution of mixture

% components’ locations and shapes

% EM Gaussian mixture model with k = 5

[z5\_em,T,soft,ll] = emCluster(X\_iris,5,'farthest',10);

% EM Gaussian mixture model with k = 20

[z20\_em,T,soft,ll] = emCluster(X\_iris,20,'farthest',10);

The plots of EM Gaussian mixture model with 5 and 20 components and the loglikehood of the returned model against the number of iterations are illustrated above on the left and right side respectively.

Unlile two clustering methods above, Expectation Maximization works the same way as K-means except the data is assigned to each cluster with the weights being soft probabilities instead of distances. The advantage is that the model becomes generative as we define the probability distribution for each model