# MLPR Assignment

Chris Swetenham (s1149322)

2nd April 2012

## Q1

### a)

I understand the question as meaning: we have a dataset X, and two different models M1 and M2. We fit the parameters of each model based on the entire dataset X. We then evaluate the likelihood of the data under each model, and select the model with the highest likelihood.

The likelihood of the training data under the learned model will only tell you how well the model fits your training data; it is easy to construct a model which will maximise this measure, but not generalise well at all.

A better way of comparing the classifiers would be to evaluate their performance under cross-validation.

### b)

When we optimise the parameters of the rbf network, for example using the EM algorithm, the positions of the rbf functions in our data space will be moved towards regions where we have training data, and will be useless for predicting in regions where we did not have training data.

In general it will be difficult to predict anything sensible in regions were we do not have training data, especially with a nonlinear model, since the model could take on any values at all in those regions and still be a good predictor of our training data. A linear model might do better in this case, although it is hard to say without any training data in those regions. Another possible slight improvement would be using some kind of prior over the model parameters.

### c)

Since a 2-layer model can approximate any function at all, it wouldn't tell us anything about regions outside our training instances, since it could approximate any function at all that includes those training instances. Another way to put this would be that this is a very complex model with a lot of parameters and a lot of scope for overfitting the training data. In addition, training this network will be very slow.

Instead, it would be preferable to actually look at the data, and use the simplest model that could possibly work with it.

# Q2

## a)

We perform Principal Components Analysis on the data by computing the eigenvectors of the sample covariance matrix. We sort them in descending order by eigenvalue.

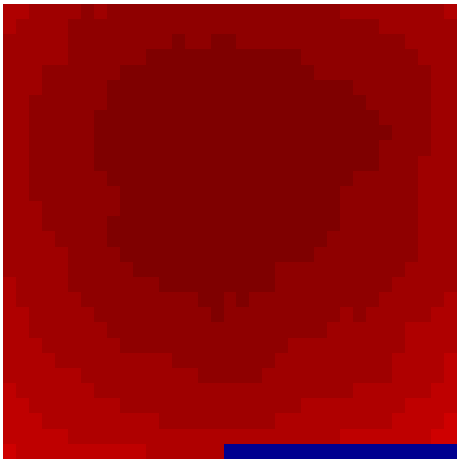
```
function [Mu, E, Lambda, P] = getEigenvectors(Sequence)
% getEigenvectors computes the eigenvectors and eigenvalues of a dataset.
% INPUT Sequence: [NFrames x NFeatures]
%           Matrix with instances as rows and features as columns.
% OUTPUT Mu: [NFeatures x 1]
%           Column vector of means for each feature.
%        Lambda: [NFeatures x 1]
%           Column vector eigenvalues of covariance matrix, in descending
%           order.
%        E: [NFeatures x NFeatures]
%           Matrix whose columns are the eigenvectors of the covariance
%           matrix in corresponding order with the eigenvalues in lambda.
%        P: [NFeatures x 1]
%           Column vector of the cumulative percentage of variance
%           explained by each of the eigenvalues in lambda.

% Compute the mean vector for the sequence.
Mu = mean(Sequence)';
% Compute the eigendecomposition of the covariance matrix.
% V: [NFeatures x NFeatures] has the eigenvectors as columns.
% D: [NFeatures x NFeatures] is a diagonal matrix of the eigenvalues.
[V, D] = eig(cov(Sequence));
% Extract and sort the eigenvalues in descending order.
% I: [NFeatures x 1] stores the corresponding indices of the sorted entries
% in the original matrix.
[Lambda, I] = sort(diag(D), 1, 'descend');
% Use the index vector to get the sorted eigenvectors.
E = V(:, I);
% Get the cumulative sum of the variance % explained by each eigenvector.
P = 100 * cumsum(Lambda) / sum(Lambda);
```

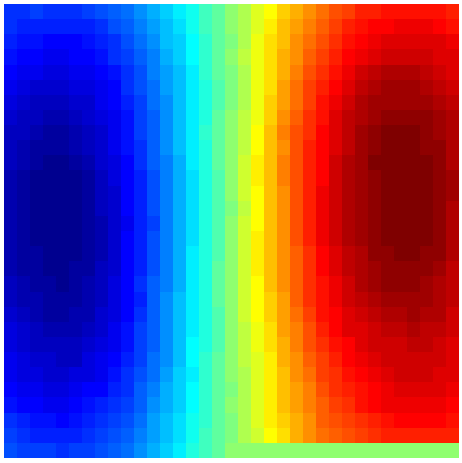
Listing 1: getEigenvectors.m

We visualise the mean patch and the first three eigenvalues in Figure 1. Patches are shown in greyscale; eigenvectors are rescaled and use the 'jet' colourmap, since they contain negative and positive components. The mean is more or less a uniform dark patch, and the first 3 eigenvectors resemble basis functions like those in the DCT transform used on patches in the JPEG file format.

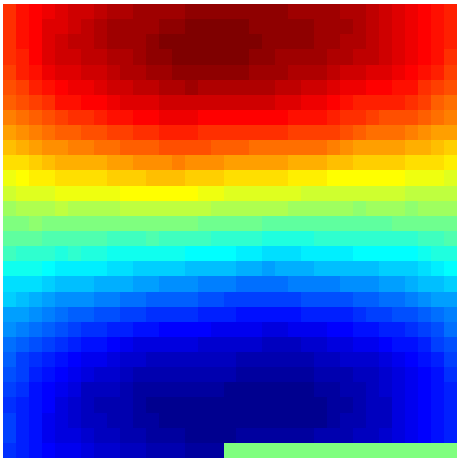
(a) Mean Patch



(b) Eigenvector 1



(c) Eigenvector 2



(d) Eigenvector 3

Figure 1: Results of PCA Analysis

## b)

We project all the patches onto the first 3 eigenvectors and then reconstruct them. We find the patch with the greatest reconstruction error.

```
function [ZSeq] = projectSequence(Mu, E, Seq, ZDims)
% projectSequence projects a data sequence onto the first ZDims components.
% INPUT Seq: [NFrames x NFeatures]
%           Input data sequence.
%       Mu: [NFeatures x 1]
%           Average of the features in the data sequence.
%       E: [NFeatures x NFeatures]
%           Column eigenvector matrix.
%       ZDims:
%           Number of
```

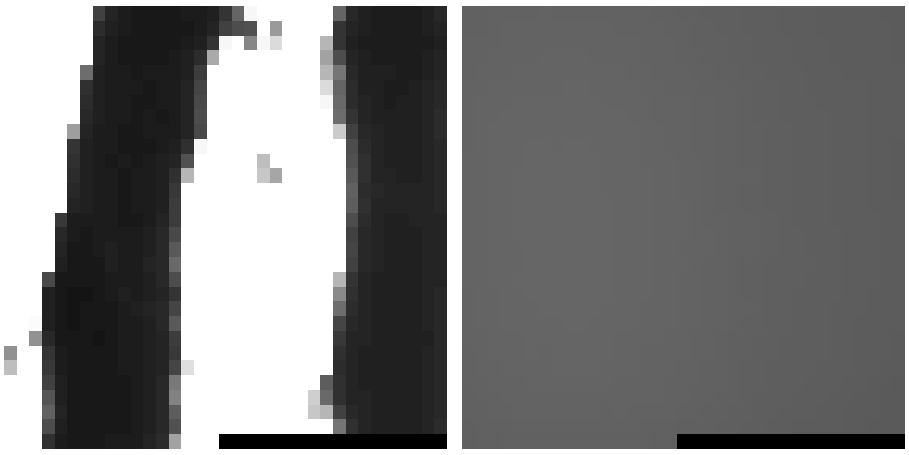
```
% OUTPUT ZSeq: [NFrames x ZDims]
%           The projected sequence.

[NFrames, ~] = size(Seq);
% Transformation to the reduced space. W: [NFeatures x ZDims]
W = E(:, 1:ZDims);
% Replicate the mean into a matrix. MMu: [NFrames x NFeatures]
MMu = repmat(Mu', [NFrames 1]);
% Perform the transformation.
ZSeq = (Seq - MMu) * W;
```

Listing 2: projectSequence.m

```
[Q2b] Patch with largest reconstruction error: 34729.
```

We visualise this patch and its reconstruction in Figure 2. The pattern of alternating stripes cannot be reproduced from the three first eigenvectors we saw above.



(a) Original Patch (b) Reconstruction

Figure 2: Patch with worst reconstruction error

## c)

We visualise the histogram of the target values. This is shown in Figure 3. It is bimodal with the second mode being very small. It looks like underlying unimodal with saturation at 63.

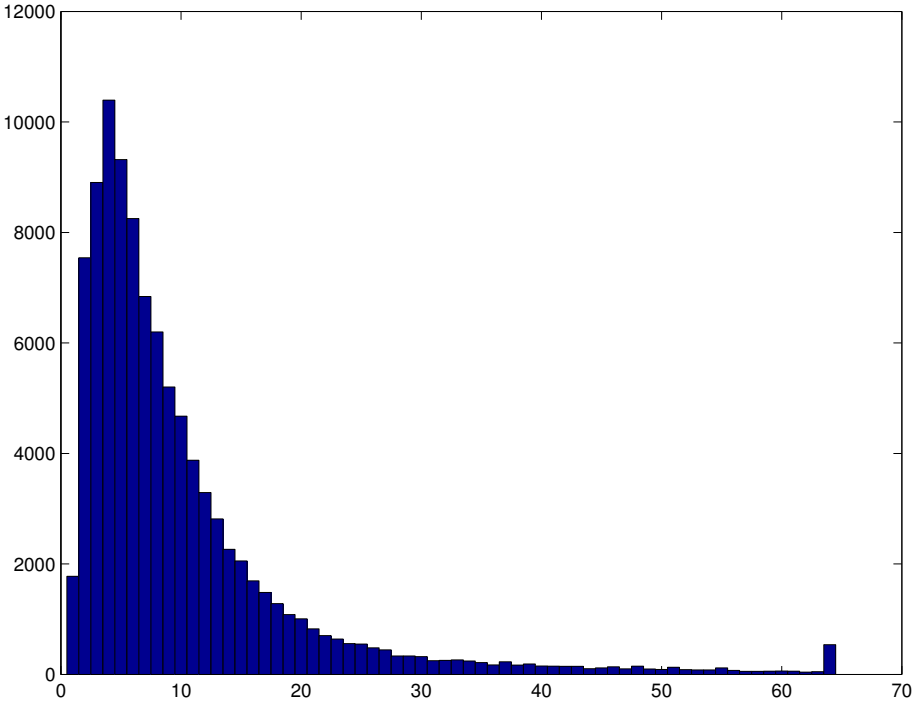



Figure 3: Histogram of Targets

## d)

We visualise the histogram of differences between x(end) and y – between the last data pixel and the target pixel. Although the question specifies 64 bins for the histogram, I later opted to use as many bins as we had unique values since I found the resulting histogram clearer. This is shown in Figure 4.

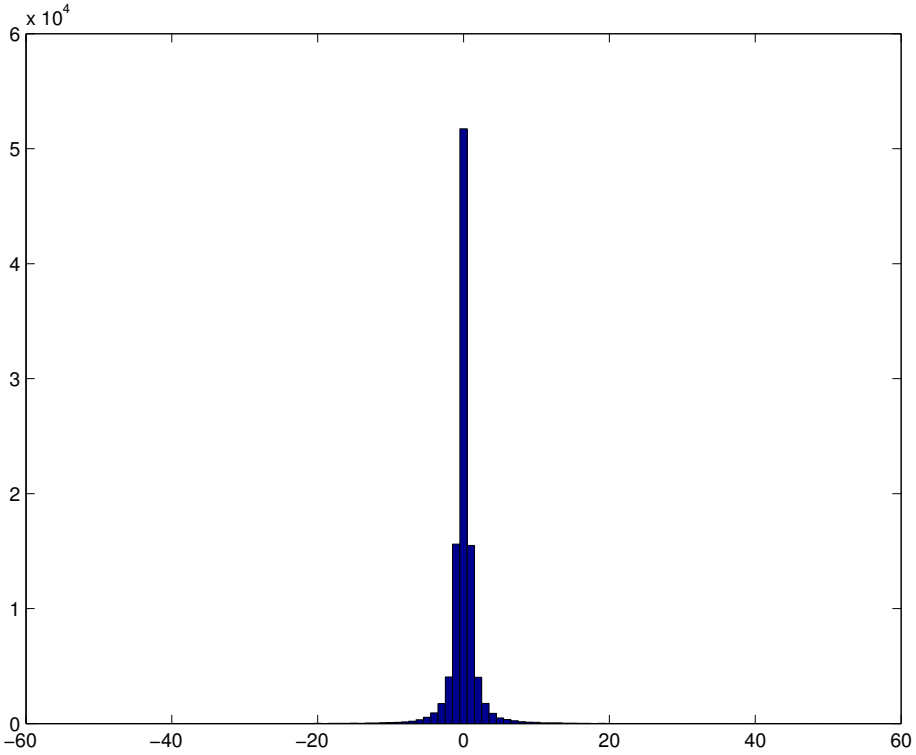



Figure 4: Histogram of Differences

```
function [Mu, E] = q2(Data)
%% Q2a
imgPatch = @(X) reshape([X zeros(1,18)],35,30)'/63;

fprintf('[Q2a] Computing Eigenvectors...'); tic;
[Mu, E, Lambda, P] = getEigenvectors(Data.x);
fprintf('Done.\n'); toc;

fprintf('[Q2a] Producing figures...'); tic;
figurePatch(imgPatch(Mu'));
writeFigureEPS('Q2a-mean.eps');
close;

figureEig(imgPatch(E(:, 1)'));
writeFigureEPS('Q2a-eig1.eps');
close;

figureEig(imgPatch(E(:, 2)'));
writeFigureEPS('Q2a-eig2.eps');
close;

figureEig(imgPatch(E(:, 3)'));
writeFigureEPS('Q2a-eig3.eps');
close;
fprintf('Done.\n'); toc;

%% Q2b
```

```
fprintf('[Q2b] Projecting patches...'); tic;
ProjectedX = projectSequence(Mu, E, Data.x, 3);
fprintf('Done.\n'); toc;

fprintf('[Q2b] Reconstructing patches...'); tic;
ReconstructedX = ProjectedX * E(:, 1:3)';
fprintf('Done.\n'); toc;

fprintf('[Q2b] Computing error...'); tic;
ErrorsX = mean((Data.x - ReconstructedX).^2, 2);
[Dummy MaxI] = max(ErrorsX);
[Log, Cleanup] = makeLogFile(['q2b.log']);
fprintf(Log, '[Q2b] Patch with largest reconstruction error: %d.\n', MaxI);
fprintf('Done.\n'); toc;

fprintf('[Q2b] Producing figures...'); tic;
figurePatch(imgPatch(Data.x(MaxI,:)));
writeFigureEPS('Q2b-patch.eps');
close;
figurePatch(imgPatch(ReconstructedX(MaxI, :)));
writeFigureEPS('Q2b-reconstructed.eps');
close;
fprintf('Done.\n'); toc;

%% Q2c

fprintf('[Q2c] Producing figures...'); tic;
CountsY = hist(Data.y, 0:63);
figure();
bar(CountsY, 'hist');
writeFigureEPS('Q2c-hist.eps');
close;
fprintf('Done.\n'); toc;

%% Q2d

fprintf('[Q2d] Producing figures...'); tic;
D = Data.y - Data.x(:, end);
[H2, X2] = hist(D, min(D):max(D));
figure();
bar(X2, H2, 'hist');
writeFigureEPS('Q2d-hist.eps');
close;
fprintf('Done.\n'); toc;
```

Listing 3: q2.m

# Q3

## a)

The Maximum Likelihood solution to Naive Bayes with categorical features and output simply counts instances of each value of $y$ and of $x|y$ in the training set. The main issue with this is that any value of $y$, or any value of $x$ for a certain $y$, that was not seen in the training set will be assigned a probability of 0. One solution to this is to use additive solution by adding a small increment (for example, 1) to each count.

## b)

### i)

In the Bayesian formulation of the categorical Naive Bayes model, we compute $P(Y|X)$ by modelling $P(Y)$ and $P(X|Y)$. Using the Naive Bayes assumption that the components of $X$ are independent given the class $Y$, we further break this down into $P(X1|Y)$, $P(X3|Y)$, $P(X3|Y)$.

We place a conjugate prior over the parameters of our categorical distributions. The parameters of $P(Y)$ are simply a vector of probabilities for each value of $Y$; and for each value of $Y$, the parameters of $P(X_1|Y)$ are simply a vector of probabilities for each value of $X_1$; similarly for $X_2$ and $X_3$.

The conjugate prior for a multinomial distribution is the Dirichlet distribution, and the categorical distribution is a multinomial distribution in the case of a single trial. Let $P(Y = y) = \theta_y$ be the parameters of $P(Y)$. Our prior over the parameters $\theta$ is $P(\theta) = Dir(\theta; \alpha)$, where $\alpha$ are the parameters of the Dirichlet distribution. We use $\alpha_k = 1$ to give us a uniform prior over $\theta$.

Since the Dirichlet distribution is a conjugate prior to our $P(Y|\theta)$, $P(\theta|D) = Dir(\theta; \alpha')$ for some $\alpha'$. Looking this up on Wikipedia's page on Conjugate Priors we find that $\alpha'_k = \alpha_k + \sum_i I(x^{(i)} = k)$; in other words, we add to each component of alpha the counts of the corresponding value in the training data.

We now have a distribution over the parameters of $P(Y|\theta)$, but we wish to find $P(Y)$ so that we can calculate $P(Y|X)$. To to this we multiply $P(Y|\theta)$ by $P(\theta)$ and marginalise out $\theta$ (marginalising out each of its components).

$P(Y = y) = \int_{\{\theta_i\}} P(Y = y|\theta)P(\theta)$

$= \int_{\theta_i} \theta_y Dir(\theta; \alpha)$

We split the integral between the one over \theta_y and the ones over the other components of \theta.

$= \int_{\theta_y} \theta_y \int_{\{\theta_i \neq y\}} Dir(\theta; \alpha)$

The marginalisation of a Dirichlet distribution is a Beta distribution.

$= \int_{\theta_y} \theta_y Beta(\theta; \alpha_y, \alpha_0 - \alpha_y)$

Where $\alpha_0$ is $\sum_i \alpha_i$. This expression is now equivalent to the mean or expectation of our Beta distribution, which is:

$= \frac{\alpha_y}{\alpha_0}$

Similar calculations occur for the parameters of $P(X_k|Y = y)$ for each value of Y. We now see that, with a uniform Dirichlet prior with $\alpha_k = 1$, the computations and results of the Bayesian formulation are precisely the same as those of the Maximum Likelihood formulation with +1 Additive Smoothing.

We train and evaluate the model using 4-fold cross-validation.

```
function Results = crossValidation(DataY, DataX, NFolds, trainAndEvaluate)
    [NInstances NFeatures] = size(DataX);
    assert(size(DataY, 1) == NInstances);
    assert(size(DataY, 2) == 1);
    FoldSize = NInstances / NFolds;
    assert(FoldSize == floor(FoldSize)); % don't want to handle an uneven
        split
    Results = cell(NFolds, 1);
    for FoldIdx = 1:NFolds
```

```
        FoldStart = 1 + (FoldSize * (FoldIdx-1));
        FoldEnd = FoldStart + FoldSize - 1;
        TrainDataX = [DataX(1:(FoldStart-1), :); DataX((FoldEnd+1):NInstances
            , :)];
        TrainDataY = [DataY(1:(FoldStart-1), :); DataY((FoldEnd+1):NInstances
            , :)];
        TestDataX = DataX(FoldStart:FoldEnd, :);
        TestDataY = DataY(FoldStart:FoldEnd, :);
        Results{FoldIdx} = trainAndEvaluate(FoldIdx, TrainDataY, TrainDataX,
            TestDataY, TestDataX);
    end
end
```

Listing 4: crossValidation.m

```
function [Likelihoods] = trainAndTestNB(FoldIdx, TrainDataY, TrainDataX,
    TestDataY, TestDataX)
    [AlphaY AlphaX] = trainNB(TrainDataY, TrainDataX);
    Likelihoods = probNB(AlphaY, AlphaX, TestDataY, TestDataX);

    Dists = distNB(AlphaY, AlphaX, TestDataX);
    PlotIdx = 9465;
    figure();
    plot(Dists(PlotIdx, :));
    line([TestDataY(PlotIdx) TestDataY(PlotIdx)], [0, max(Dists(PlotIdx, :))
        ],'Color','r');
    writeFigureEPS(['Q3bii-hist-' int2str(FoldIdx) '-' int2str(PlotIdx) '.eps
        ' ]);
    close;
end
```

Listing 5: trainAndTestNB.m

```
function [AlphaY AlphaXYk] = trainNB(DataY, DataX)
    [NInstances NFeatures] = size(DataX);
    AlphaXYk = ones(64, 64, 3); % prior
    AlphaY = ones(64, 1); % prior

    for I = 1:NInstances
        Y = DataY(I)+1;
        X1 = DataX(I, 1)+1;
        X2 = DataX(I, 2)+1;
        X3 = DataX(I, 3)+1;

        AlphaY(Y) = AlphaY(Y) + 1;
        AlphaXYk(X1, Y, 1) = AlphaXYk(X1, Y, 1) + 1;
        AlphaXYk(X2, Y, 2) = AlphaXYk(X2, Y, 2) + 1;
        AlphaXYk(X3, Y, 3) = AlphaXYk(X3, Y, 3) + 1;
    end
end
```

Listing 6: trainNB.m

```
function [P] = probNB(AlphaY, AlphaXYk, TestY, TestX)
    [NInstances NFeatures] = size(TestX);
    assert(size(TestY, 1) == NInstances);
    Ps = distNB(AlphaY, AlphaXYk, TestX);

    P = zeros(NInstances, 1);
    % Just select the observed y(i) from the
    % distribution p(y|x(i))
    for I = 1:NInstances
```

```
        Y = TestY(I) + 1;
        P(I) = Ps(I, Y);
    end
end
```

Listing 7: probNB.m

```
function [P] = distNB(AlphaY, AlphaXYk, TestX)
% P(y|x) = P(y, x)/P(x)
% P(y, x) = P(x1|y)P(x2|y)P(x3|y)P(y)
% P(x1|y) = AlphaX1Y(x1, y)/Alpha(y)
% P(y) = AlphaY(y)/TotY
% P(x) = sum(P(y, x), y)
    [NInstances NFeatures] = size(TestX);
    TotY = sum(AlphaY);
    P = zeros(NInstances, 1);
    for I = 1:NInstances
        X1 = TestX(I, 1) + 1;
        X2 = TestX(I, 2) + 1;
        X3 = TestX(I, 3) + 1;
        for Y = 1:64
            PY = AlphaY(Y)/TotY;
            PX1 = AlphaXYk(X1, Y, 1)/AlphaY(Y);
            PX2 = AlphaXYk(X2, Y, 2)/AlphaY(Y);
            PX3 = AlphaXYk(X3, Y, 3)/AlphaY(Y);
            P(I, Y) = PX1 * PX2 * PX3 * PY;
        end

        % Normalise
        P(I, :) = P(I, :) / sum(P(I, :), 2);
    end
end
```

Listing 8: distNB.m

```
% Smaller perplexity is better - best possible is 1.
function Perplexity = perplexity(Likelihoods)
    Perplexity = exp(-mean(log(Likelihoods)));
end
```

Listing 9: perplexity.m

We compute the following perplexity value:

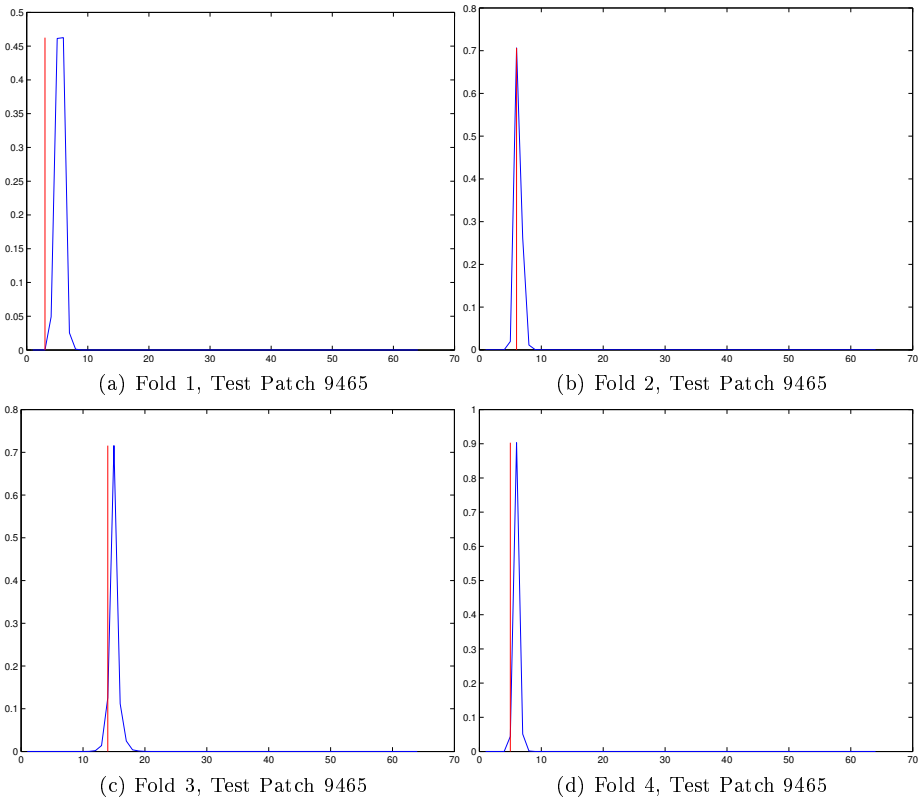
```
[Q3bi] Naive Bayes Perplexity: 4.688695.
```

This seems like a pretty good value, since the best possible perplexity score would be 1.

### ii)

We visualise $P(y|x^{(i)})$ and $y^{(i)}$ for each fold for (arbitrarily) the $9465^{th}$ patch in each fold. This is shown in Figure 5. The distributions have a sharp peak, and we see that we can get a mode very close to the true value yet still assign a low probability to the true value. This is most visible in the example from fold 1. The disadvantage of our model is that the target values are categorical, but we know that the underlying reality of the data is a real value which has been

captured with noise and then quantised; if we see an output value of 45 in the training data it ought to increase our estimated probability of seeing a 44 or 46, for example.



(a) Fold 1, Test Patch 9465

(b) Fold 2, Test Patch 9465

(c) Fold 3, Test Patch 9465

(d) Fold 4, Test Patch 9465

Figure 5: Distributions and actual values for selected test patches

### iii)

The strong assumption made by Naive Bayes is that the input features are distributed independently given the output value. This seems unlikely, especially since we could have picked any other pixel in the patch as the target; and if the Naive Bayes assumption held for one target then it seems like it should equally hold for another, since there is little to distinguish them; but it would therefore have to hold between any sets of pixels in the patch.

```
function q3(Data)
%% Q3a
% No code

%% Q3b
% i)
fprintf('[Q3b] Running Naive Bayes Cross-Validation...'); tic;
```

```
[NInstances NFeatures] = size(Data.x);
SubSetX = [Data.x(:, end), Data.x(:, end - 34), Data.x(:, end - 35)];
ResultsNB = crossValidation(Data.y, SubSetX, 4, @trainAndTestNB);
LikelihoodsNB = reshape(cell2mat(ResultsNB), NInstances, 1);
PerplexityNB = perplexity(LikelihoodsNB);
fprintf('Done.\n'); toc;

[Log, Cleanup] = makeLogFile(['q3bi.log']);
fprintf(Log, '[Q3bi] Naive Bayes Perplexity: %f.\n', PerplexityNB);

% ii)
% Figures generated in trainAndTestNB.
% iii, iv)
% No code
```

Listing 10: q3.m

# Q4

I interpret this question as asking us to now treat the intput and output as numerical rather than categorical data and perform linear regression on that.

## a)

We use Netlab to perform linear regression, although this is more to save time debugging than due to any particular difficulty of implementation. We select only the final, 34th and 35th from final pixels in the patch data for our input, as specified.

```
function [Likelihoods] = trainAndTestLR(FoldIdx, TrainDataY, TrainDataX,
    TestDataY, TestDataX)
    [Net V] = trainLR(TrainDataY, TrainDataX);
    Likelihoods = probLR(Net, V, TestDataY, TestDataX);
end
```

Listing 11: trainAndTestLR.m

```
function [Net V] = trainLR(TrainDataY, TrainDataX)
    [NInstances NFeatures] = size(TrainDataX);
    Net = glm(NFeatures, 1, 'linear');
    Options = zeros(14, 1);
    Options(1) = 1;
    Net = glmtrain(Net, Options, TrainDataX, TrainDataY);
    Pred = glmfwd(Net, TrainDataX);
    % MLE estimate of V
    V = mean((Pred - TrainDataY).^2);
end
```

Listing 12: trainLR.m

```
function Likelihoods = probLR(Net, V, TestDataY, TestDataX)
    [NInstances NFeatures] = size(TestDataX);
    Likelihoods = zeros(NInstances, 1);
    Pred = glmfwd(Net, TestDataX);
    for I = 1:NInstances
        Likelihoods(I) = gauss(Pred(I), V, TestDataY(I));
```

```
    end
end
```

Listing 13: probLR.m

We measure the perplexity under cross-validation.

```
[Q4a] Linear Regression Perplexity: 6.747893.
```

It might seem wrong to compare this to the perplexity in the last section, since here $p(y|x)$ is a probability density rather than a discrete probability. Perhaps it would have been better to approximate $P(y|x)$ as $1/2*(p(y-0.5|x)+p(y+0.5|x))$, thus approximating the integral of $p(y|x)$ in the interval $(y-0.5,\ y+0.5)$. But numerically this would give very similar results to just our evaluation at the midpoint, so we do allow ourselves to compare the results using a $p(y|x)$ as a density function to those using $P(y|x)$.

### b)

We project the data onto the first ten eigenvectors of our PCA model from Q2, and use these as our features for linear regression.

```
[Q4b] Linear Regression PCA Perplexity: 15.603067.
```

### c)

The perplexity score for the PCA method is notably worse. Although the PCA construction gives us a good way of discarding data while still reconstructing a reasonable approximation of the original patches to human eyes, when it comes to predicting specifically the value of the target pixel, using its neighbours seems better.

## Q5

The results of Q2d) suggest we could get a decent result by just predicting y=x(end). We find the standard deviation of the error to give us a narrow gaussian around x(end) for a distribution p(y|x).

```
function q5(Data)
[Log, Cleanup] = makeLogFile(['q5.log']);
[NInstances NFeatures] = size(Data.x);
ResultsLP = crossValidation(Data.y, Data.x, 4, @trainAndTestLP);
LikelihoodsLP = reshape(cell2mat(ResultsLP), NInstances, 1);
PerplexityLP = perplexity(LikelihoodsLP);
fprintf(Log, '[Q5] Last-Pixel Perplexity: %f.\n', PerplexityLP);
```

Listing 14: q5.m

```
[Q5] Last-Pixel Perplexity: 9.674745.
```

The resulting perplexity is comparable to our predictors from Q4 but not as good as the naive bayes result from Q3.

```
function q4(Data, Mu, E)
[NInstances NFeatures] = size(Data.x);
%% Q4a
[Log, Cleanup] = makeLogFile(['q4a.log']);
fprintf('[Q4a] Running Linear Regression Cross-Validation...'); tic;
SubSetX = [Data.x(:, end), Data.x(:, end - 34), Data.x(:, end - 35)];
ResultsRL3 = crossValidation(Data.y, SubSetX, 4, @trainAndTestLR);
LikelihoodsRL3 = reshape(cell2mat(ResultsRL3), NInstances, 1);
PerplexityRL3 = perplexity(LikelihoodsRL3);
fprintf('Done.\n'); toc;
fprintf(Log, '[Q4a] Linear Regression Perplexity: %f.\n', PerplexityRL3);

%% Q4b
[Log, Cleanup] = makeLogFile(['q4b.log']);
fprintf('[Q4b] Running Linear Regression PCA Cross-Validation...'); tic;
ProjectedX = projectSequence(Mu, E, Data.x, 10);
ResultsRL10 = crossValidation(Data.y, ProjectedX, 4, @trainAndTestLR);
LikelihoodsRL10 = reshape(cell2mat(ResultsRL10), NInstances, 1);
PerplexityRL10 = perplexity(LikelihoodsRL10);
fprintf('Done.\n'); toc;
fprintf(Log, '[Q4b] Linear Regression PCA Perplexity: %f.\n', PerplexityRL10)
    ;
%% Q4c
% No code
```

Listing 15: q4.m

# Appendix A - Additional Code

```
%% Housekeeping
clc;
close all;
clear all;
dbstop if error;
dbstop if naninf;
addpath('netlab/');

%% Load data
fprintf('Loading data...'); tic;
Data = load('imdata.mat');
Data.x = double(Data.x);
Data.y = double(Data.y);
Data.i = double(Data.i);
TestData = load('imtestdata.mat');
TestData.x = double(TestData.x);
TestData.i = double(TestData.i);
[NInstances NFeatures] = size(Data.x);
fprintf('Done.\n'); toc;
%% Q1
% No code

%% Q2
[Mu, E] = q2(Data);

%% Q3
q3(Data);

%% Q4
q4(Data, Mu, E);
```

```
%% Q5
q5(Data);

%% End
fprintf('Done.\n');
```

Listing 16: main.m

```
function figurePatch(Img)
    figure();
    imshow(Img);
    colormap('gray');
    axis('square', 'off');
end
```

Listing 17: figurePatch.m

```
function figureEig(Eig)
    figure();
    imagesc(Eig);
    colormap('jet');
    axis('square', 'off');
end
```

Listing 18: figureEig.m

```
function [F, C] = makeLogFile(FileName)
% makeLogFile opens a log file and makes an onCleanup object to close it.
F = fopen(FileName, 'w');
C = onCleanup(@()fclose(F));
```

Listing 19: makeLogFile.m

```
function [] = writeFigureEPS(Fig, FileNameEPS)
% writeFigureEPS writes the current figure to an eps file
% INPUT Fig: [optional] the figure to write; default is the current figure.
%       FileName: a string containing the path of the file to save to.

if nargin < 2
    FileNameEPS = Fig;
    Fig = gcf;
end
SavedBreakpoints = dbstatus;
dbclear if infnan; % Yes, this is a thing
print(Fig, '-depsc', FileNameEPS);
dbstop(SavedBreakpoints);
```

Listing 20: writeFigureEPS.m