Machine Vision – Assignment 1
John Goodacre

PracticalMixGaussA.m

To do a) Here we are asked to implement the function fitGaussianModel. The function receives an array of 3d points (in this case corresponding to R,G,B). It returns the mean and the covariance of these points. Because the data is 3xn – then the mean will be 3x1 (i.e. the mean of each row). The covariance will be 3x3 – I implemented this very simplistically using the mean and covariance functions in matlab – for the covariance each element was calculated separately i.e. cov(R,B) etc.

To do b) Here we are asked to calculate the likelihood of each data point. To be specific the function receives a 3x1 data point and returns the likelihood of this point using a multivariate Gaussian (with model parameters mean/ cov as mentioned above). We were asked to not use say mvnpdf from Matlab, but instead to implement this ourselves – which I did in a function I called multivarGauss (included). MultivarGauss uses the usual formula for the Gaussian and in this case is implemented with the inv function (one has to be careful with large matrices).

To do c) Here we are asked to calculate posterior probabilities for each pixel of being skin or non skin. Using Bayes rule we can create a posterior for each pixel using our priors and likelihoods. Specifically –
posterior of skin(pixel) = prior of skin*likelihood of skin(pixel)/(prior of skin * likelihood of skin(pixel)) + (prior of non skin*likelihood of non-skin(pixel).

This is done for each pixel. Here the prior is an arbitrary number 0.3.



The figure above shows the results. On the left we have the original image, the middle image is our ground truth image. If one is trying to identify skin, then this image is the absolute truth. The figure to the right shows our posteriors for each pixel. As one can see this is less than perfect. Note the model knows nothing about the test image and was trained on other skin/ non skin data. However, this is a basic model. Given the simplicity of the model the results aren't terrible, but there are weaknesses. Note the inability of the model to differentiate between the guitar and the skin. Particularly in the bottom right of the picture. Clearly this is down to the colour. Model improvements could be using a mixture of Gaussians (done later), using t-distributions (fat tails and should pixel colours be normally distributed? I.e. is that model a good representation?). And finally we go through pixel by pixel only. Clearly there are relations across pixels.

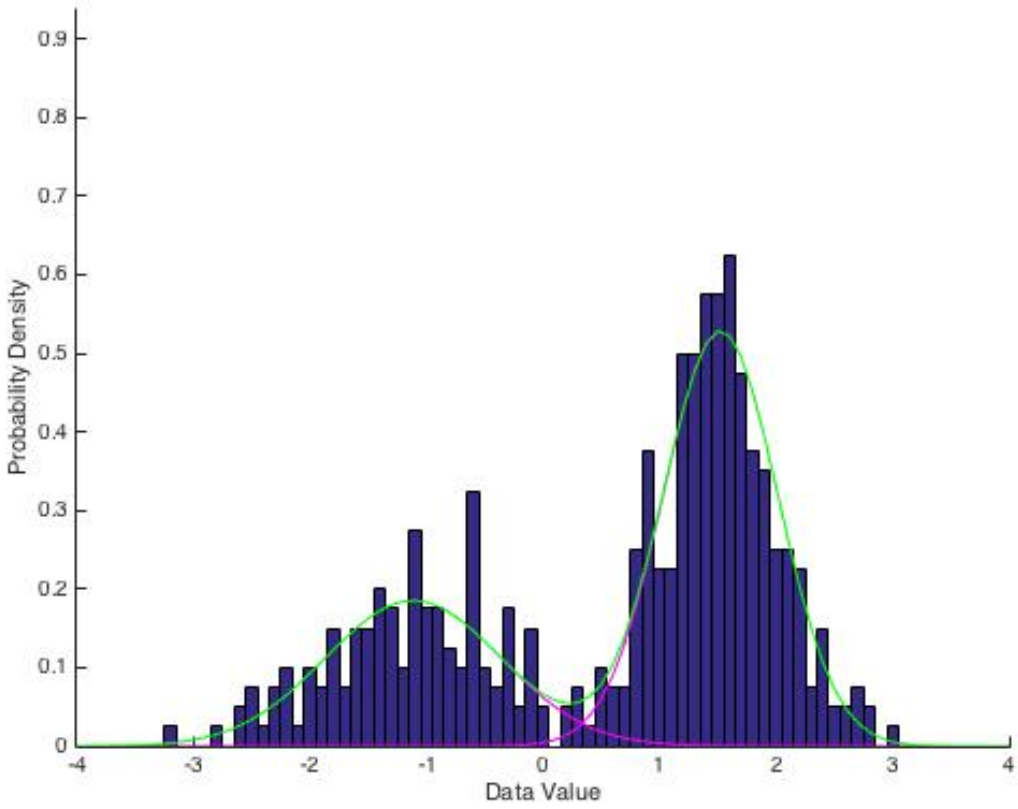
practicalMixGaussB

To do d) We now shift to practicalMixGaussB. Here we generate 1-dimensional data from two different Gaussians and attempt to infer the parameters of the respective distributions. In d) we are asked to generate data.

If we assume the data is generated by weighting two separate Gaussians then we can generate data first by randomly drawing values from a Bernoulli – here the weights are 0.3, 0.7 and so we have a probability of 0.3 that the data should be generated from the first distribution and 0.7 from the second. (This can of course be generalised to multiple distributions).

Once we have chosen which distribution we wish to use to generate the data this is implemented by using the matlab randn(1) function to receive a member n of N(0,1). And this is scaled to an N(mu, sigma_sq) by taking mu+sigma*n. This is our generated value. We then do this multiple times to give the raw data.

e) To do – calculate the likelihood of each data point. This to do perhaps needs a step backwards to explain. Here we are calculating the likelihood for every data point. Given a set of data points and a number of Gaussian models (in this case 2), we are generating a likelihood for each data point which will then return a single log-likelihood for the whole data set under these models. Within the expectation maximisation algorithm each iteration guarantees that the log-likelihood will not decrease. To calculate the log-likelihood for a single data point we multiply the likelihood of that data point for each model with its respective weighting (each data point is 'closer' to one model that another and so one can consider weightings to be responsibilities of each model for that data point). We sum the likelihoods for all the data points under each model and then take the log (often better for computational purposes given floating point errors).



The figure above shows the result of applying the expectation-maximisation algorithm. If one examined the original data and unknown distributions (supplied) then one would see the pictures are almost identical with in this case a very good convergence. Above I was only asked to describe only the likelihood calculation for EM. To illustrate the picture in EM there is an expectation part, which calculates likelihoods for each data point for each model and then uses this to update the weights or responsibilities (hidden variables which we are marginalising over). The maximisation part updates the parameters (aggregate weighting, means and covariances) for the respective models using these new 'responsibilities'. This in turn gives a new model which again takes us to the expectation part. In this case we used 20 iterations.

The result here was good. But we knew the data was generated by two Gaussians. Convergence is not guaranteed and is also dependent upon the initial guesses for our model parameters.

To do f) Again here we are asked to generate data. The dimensionality has increased. We are now generating two dimensional data from three Gaussians. This is similar to before – in this case we draw from a discrete distribution to decide which of the three Gaussians we should use. Unlike before the data is 2-dimensional so each of the three means will be 2-d and each covariance matrix a 2x2 matrix. In the same way as before we generate normal random variables and scale them. Because we are in 2-d we use the chol function multiplied by the covariance matrix to scale the distribution correctly and add the mean vector to shift it. (Cholesky is effectively a way of taking the square root of a covariance matrix. A covariance matrix is positive semi-definite x'Cx>=0 for all x, and so C can be factored into LL'. This is the Cholesky factorisation).

To do g) Fill in the column of 'hidden' posterior probabilities that each data point came from each Gaussian.

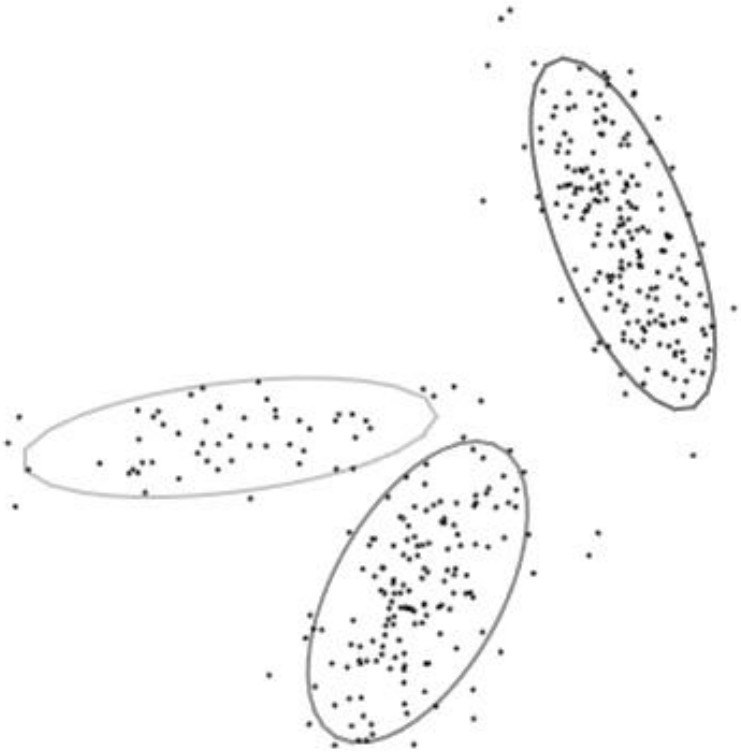
This is the 'expectation' part of the EM algorithm. In EM we solve by having a layer of hidden weights (responsibilities) – later these will be marginalised away. Each data point will be 'closer' to one of our proposed Gaussian models than another. Thus will have a higher likelihood, this likelihood is multiplied by that models current weight for that data point. If we repeat for all data points over each model this gives a new set of hidden responsibilities.

To do h) Now we are within the Maximisation step. Specifically, we are asked to update the weights for each model. This is calculated as the sum of each models responsibilities to all the data points divided by the sum of all the responsibilities overall. This gives a new overall weight for each model to the data.

To do i) still within the Maximisation step we also need to adjust each models mean vector. This is done by taking the weighted sum of the data for each model respectively (weights being the hidden variables), divided by the sum of the weights. This will give a new mean vector for each model.

To do j) again within the Maximisation step we update the covariances for each model. The way I have implemented this is to take the difference of each data point from the model mean – I call this xs_mu. This is then multiplied by the model's hidden weights. To give weighted xs_mu. The two are multiplied together and element by element divided by the sum of the weights for that model. This gives a new covariance matrix for that model.

The figure below shows the results. As mentioned above the data came from three Gaussians and is clearly 2-d data. Again the fit of the data is dependent upon our initial guesses. In my case I didn't get convergence initially and made 'fatter' covariances. My initial guesses were `2*randn(nDim,k)` for the means and `(5.5+1.5*rand(1))*eye(nDim,nDim)` for the covariances.



Part II

B) 'Train a mixture of Gaussians model for distinguishing apple vs. non-apple pixels. Use red, green, and blue as your dimensions for now. Make any other decisions you need to, and document them in your report'.

The assignment provided three pictures containing apples, and three images containing ground masks. My approach was to load each of the three pictures into two training sets. The first containing a very large array of positive examples of apple pixels. The second containing an array of non-apple pixels. Although pixels are RBG containing numbers between 0-255, in this case the data was scaled between 0 and 1. With RGB being my dimensions of course.

The aim from here was to create two models. The first a mixture of Gaussian models for apple pixels, the second another mixture of Gaussian models for non-apple data.

My approach was based on that for the earlier skin/ non skin code. Given test images the pixels would be run through in turn to calculate a likelihood of being an apple pixel.

A number of changes had to be made. Skin/ Non skin was much smaller in terms of data requirements and I found the loops involved particularly time consuming. I thus created a function vcalcGaussianProb (included), in order to do the Gaussian pdf estimates for all the data in a vectorised manner. Also because of the size of the data, I first got the code working by using a sample of the training data and testing other samples of the training data.

My choice of the number of models was somewhat trial and error but also informed by these experiments. Three models, was reasonable but I eventually settled upon five.

fitmixGauss

The main workhorse function is the fitmixGauss function which executes the expectation maximization algorithm described before. The data had been scaled between 0 and 1 so I chose uniform (0,1) random variables for the initial guesses for the means of the models and `rand(1)*eye(nDim,nDim) for the covariances`. After the code was vectorised I was able to increase the number of iterations to 50.

The philosophy in the code was to allow for loops only looping though the small number of models but not through each data point due to time considerations. I did use the mvn function but could have chosen to use my multivarGauss function from before.

Otherwise the implementation is reasonably similar to the mixGauss practical bar additional complexity and vectorisation.

I decided to count the number of apple/ non apple pixels and use these as my priors.

C) Download the file testApples.zip. Put figures into your report, showing each pixel's posterior probability of being "apple.

LoadApplesScript

After loading the training data and creating the two models the next part of the exercise involved looking at the test pictures and calculating the posterior probabilities.

Again here the idea was to loop through each pixel and use Bayes to calculate the posteriors from the likelihoods and the priors. Again I found the existing implementation in PracticalMixGauss painfully slow and so this was also vectorised.

The method was to take the test images' pixels. Change the dimensionality to 3xn and feed in one go to my vectorised calcGaussianProb function to receive the likelihoods in one vector. This could then be used to calculate a vector of posteriors, which was reconverted back into the dimensions of the initial image.

The posteriors for the images are shown below. In the first image the model worked well on the red apples, and reasonably on the green apples but poorly with the yellow apples. If one notes the training set with pears and other yellow non-apples then this becomes less surprising.

The second image for the most part identifies the apples, although the background contains green leaves some of which it is mis-identifying as apples pixels.

The results for the third image are excellent. The model clearly differentiates between the apple and the orange?/Satsuma!. Almost anything non-apple is correctly rejected.

D) For the test image with a ground-truth mask, quantify and report your result.

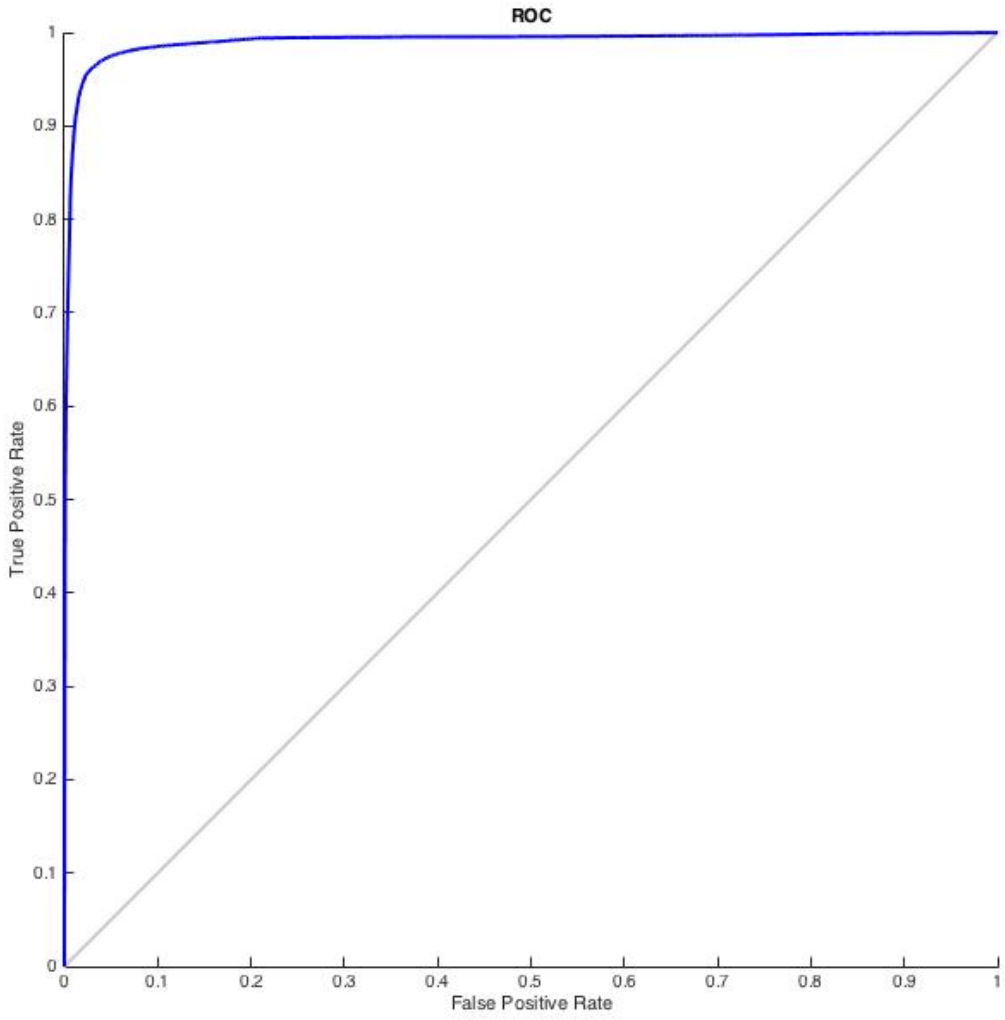
The image above with the excellent posterior also had a ground mask provided. We can use this to quantify and report upon its success.

My method was to read in the ground mask together with the posterior for the image, then to reshape both matrices to an array. One would contain an array of posterior values between 0 and 1, the other an array of 0,1s where ones report the location of an apple pixel.

As suggested by the assignment Wikipedia provides an article on ROC curves, where one plots the TPR (true positive rate) against the FPR (false positive rate) as the threshold is changed. For example – if my posterior for a pixel was say 0.8, and the pixel actually was an apple pixel – then this would be a true positive if my threshold was less than 0.8 and a false one otherwise.

TPR = TP/(TP+FN), FPR = FP/(FP+TN), where FP (false positive – 'says its apple but it isn't') and TN (true negative – says its non apple and is right), follow definitions analogous to type I and type II errors in statistics.

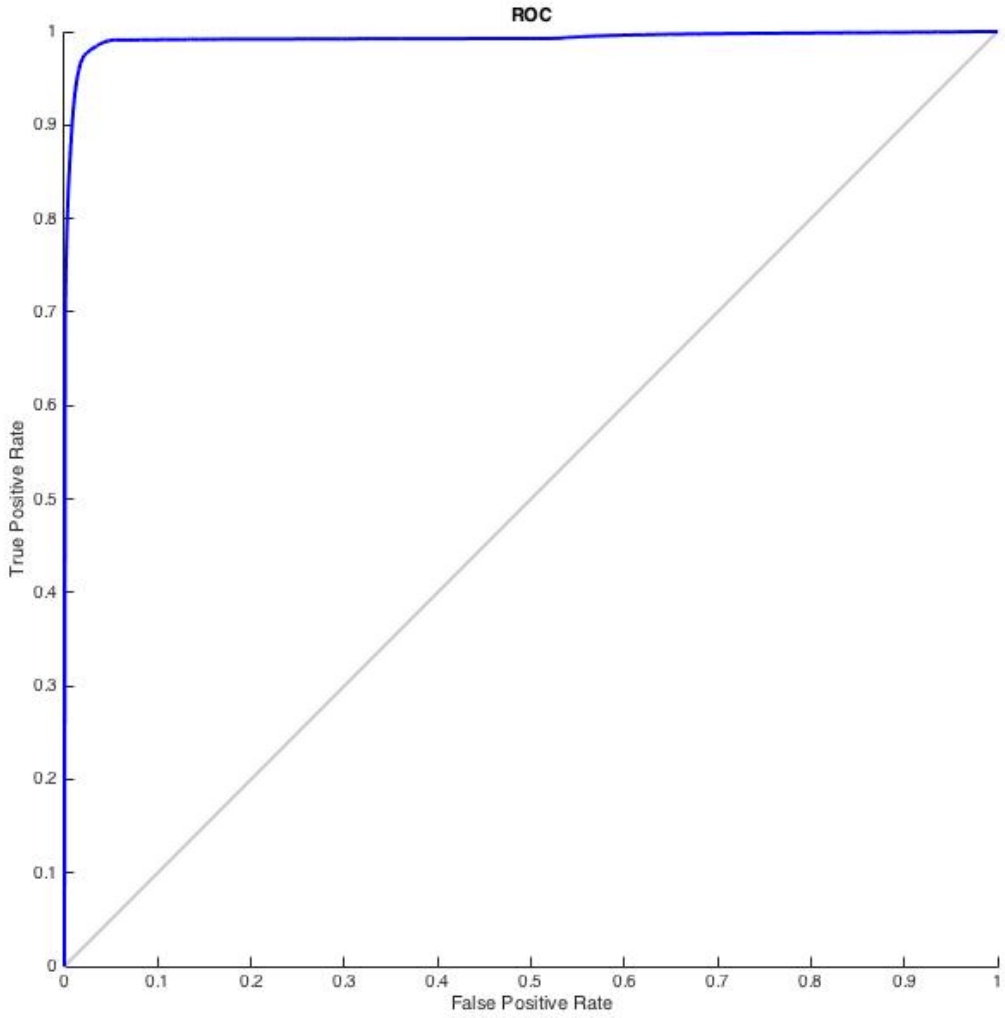
One can calculate this explicitly by counting and changing the threshold, or use the plotroc function offered by matlab as I did.



The ROC curve above quantifies the intuition behind the good result for the test image. The intuition is that no skill would be represented by the diagonal, and the better the skill the greater the area under the curve to a maximum of 1. The was done with 50 iterations and 5

Gaussian models. The quantification number would be the area under the curve. Or visually we could simply plot multiple ROC's together.

The result for three Gaussian models on the test image is shown below and is again extremely good.

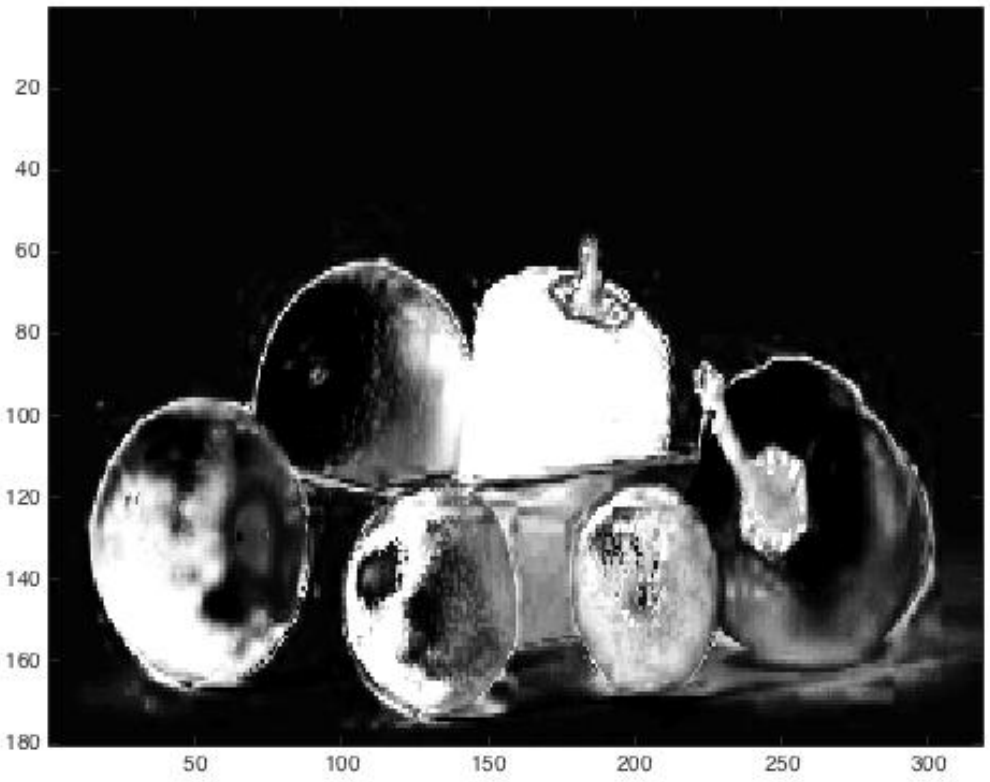


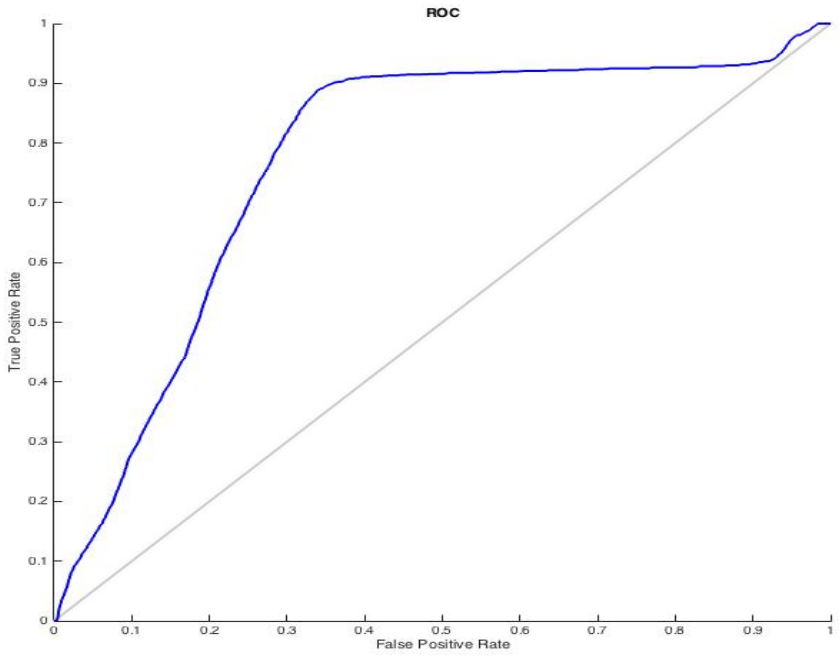
E) Download two non-copyrighted photos with apples (maybe http://search.creativecommons.org/ or some other source of images that are not copyrighted). Make good ground-truth masks for them. You can use Windows Paint, or more sophisticated programs like Gimp (free). Use these as extra test-images. Report your qualitative and quantitative results.

I used creative commons to find two test images of apples. I suspect it would have been reasonably easy to find 'gimme' images. However, I was curious as to how badly the model may do under certain circumstances. The image below contains one apple, and with not ideal colours given what we know from before and a number of other fruits and veg.

I created the ground image below using GIMP.

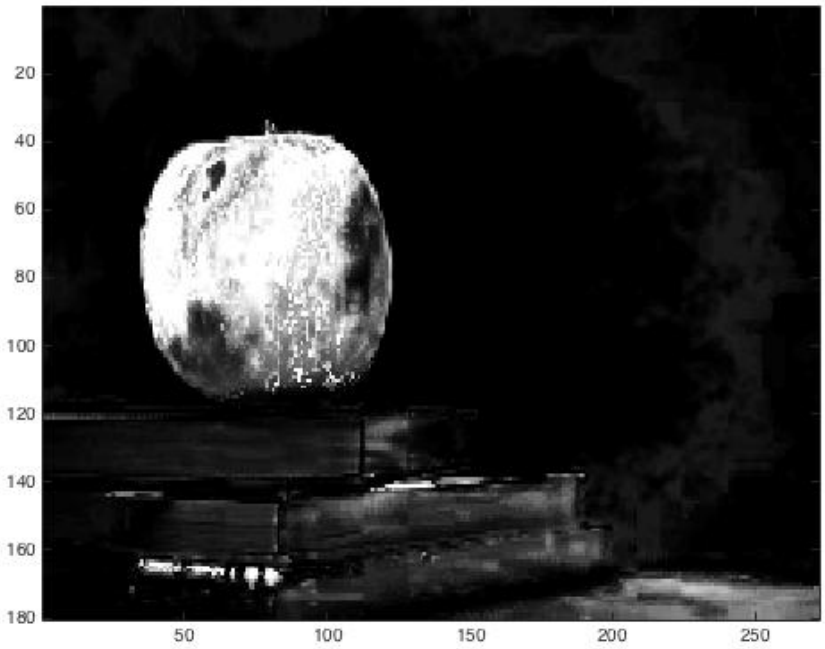
The posterior for the image is given below. Note the model does identify about half the apple however gets completely thrown by the red pepper. I would expect worse ROC results, which can be seen clearly – although it appears I hadn't given credit for the majority of the backgound image it correctly classified as non apple. Again this was five Gaussians.



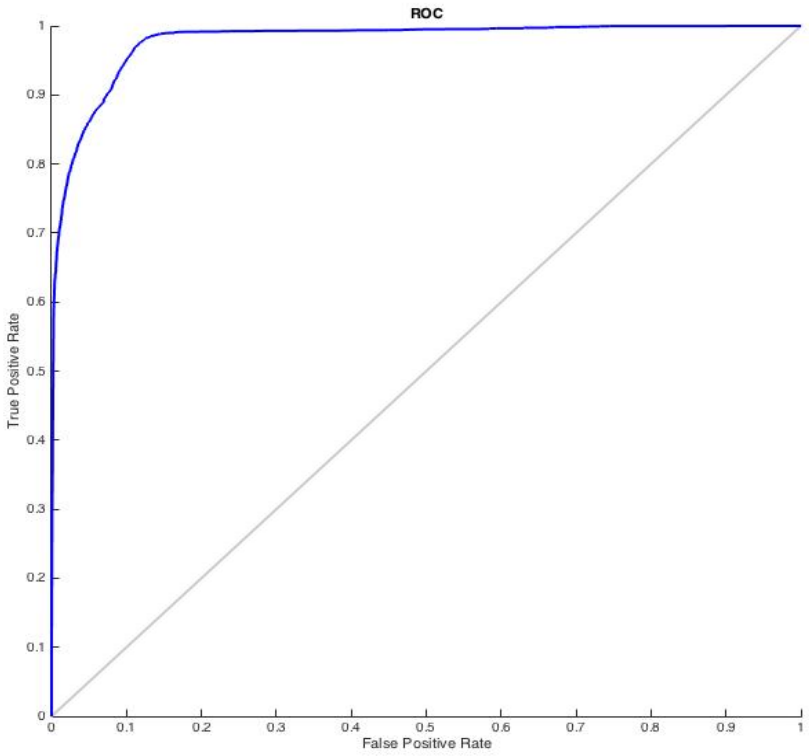


The second image I suspected would be far easier given what we know so far. The image and ground mask are given below.

As can be seen by the posterior of the image the results are good with the vast majority of the apple being correctly identified as well as almost everything that is not apple.



As expected the ROC curve quantifies the good result.



F) We should really be using three separate sets of files: a training set, a validation set, and a test set! Explain why.

The short answer is that by holding back some of the training data to validate upon we are more likely to have a model that performs similarly under test conditions. Hence techniques like 'leave one out' and 'cross-validation'. We must be particularly careful about what we do to our model after training (ideally nothing) – as this is not information we had in training but afterwards. Meaning our test results are likely to be worse than training. When I chose my number of models it was by holding back some of the training data and experimenting. (Perhaps I should also have looked to the characteristics of the data), however it was not by training on all the data and then choosing the best number for the number of Gaussians (aside, I believe this error is particularly prevalent in Finance!).

The more comprehensive answer from 'The Elements of Statistical Learning, Hastie, Tibshirani, Freidman' p222.

'There are two separate goals that we have in mind….Model Selection, Model Assessment…The best approach for both problems is to randomly divide between a training set, validation set and test set. The training set is used to fit the models, validation for assessment of the generalisation error of the final chosen model. Ideally the test set should be kept in a vault'.

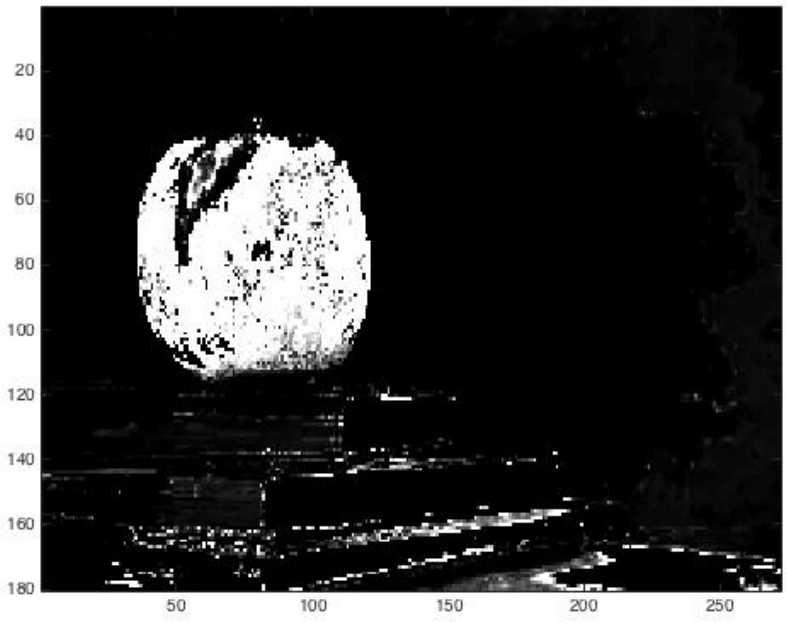
Extra Credit

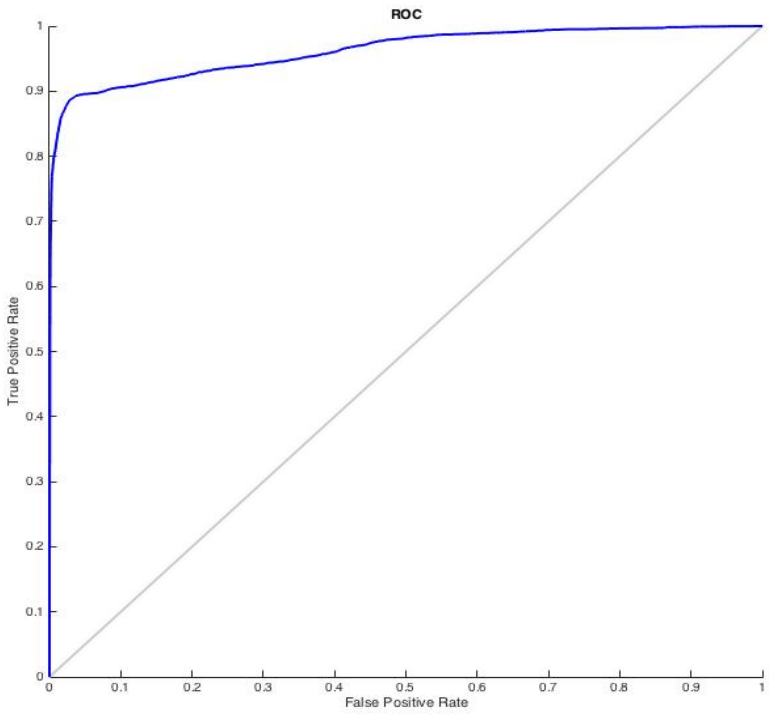
Consider manipulating the photographs' colours to improve the classification.

I created a further test image after playing with the colour map in gimp.

The ground mask is of course the same.

The posterior and ROC curve is shown below.





Apologies I should perhaps have plotted these on the same graph. However, a careful look shows that for a lower threshold we have a higher true positive rate. Unfortunately, thus far I can only conclude the result is different, not better – as the result was already extremely good. Given more time I would experiment further but the point is clear that one can achieve different results by manipulating the image's colour-map.

Files included –

All of the part 1 code implementations are in 02_PracticalMixGauss
The Part 2 assignment code is in Ass1Code folder run from the load apples script
My original images and image masks are in TestIm_origs and TestIm_masks respectively.
The code has been tested to run given the same directory structure as the zip…note that if one wishes to change images that file paths are hard coded re images and certain images relate to certain ground images, there are two places manually in the code where this is done. The is in loadapples.m