CM20220: Fundamentals of Machine Learning

Student Declaration:

By signing the student declaration, you agree that you have read and understood the University's Examination and Assessment Offences Code of Practice statement (QA53) and your student handbook including the particular references to and penalties for unfair practices such as plagiarism, fabrication or falsification.

If you are in any doubt that all the material in this submission is all your own work (except where you have included full references in the required form), you should consult your handbook for further explanations and guidance or seek help from appropriate members of staff before submission, in order to avoid any later accusation and possible penalty.

Abstract:

This report will discuss and analyse a classifier that uses a statistical model to classify binary images in MATLAB. Firstly, the algorithm will be contextualised with theory on machine learning, the model used, and the method used for testing it. The next section will detail the implementation of this classifier, describing precisely how it is trained and tested. Test results will then be presented along with an analysis of how well the classifier performed and what can be done to improve it.

Introduction and Theory:

Classification is a form of supervised machine learning where the classifier aims to identify the class of a previously unseen observation or image by making observations on a set of known and labelled training data.

The process of creating a classifier starts with obtaining a set of images for training and another for testing. Using chain-coding and Fast Fourier Transforms, the binary images are converted to feature vectors to be labelled for training the classifier on what kind of images belong to each class. A Fourier Transform is a method for processing images and decomposing them into sinusoidal functions. The relevant one, the Fast Fourier Transform, takes a function of time and returns the frequencies in waveform, and vice versa when using its inverse. To take the Fast Fourier Transform of an image or object, frequencies need to be calculated as sinusoids at specified intervals and then joined, potentially using a 2D array. Equation 1 shows how to calculate a sinusoid where a is the amplitude, ω is the frequency, and φ is the phase. Equation 2 shows how to calculate the FFT by summing discrete and finite sets of waves/frequencies instead of integrating over an infinite range. As for chain codes, they are a convention used to represent the boundary of connected

$$f(x) = a\sin(\omega x + \varphi)$$

Equation 1

$$F_\omega = \sum_{n=0}^{N-1} f_n e^{-i2\pi\omega n/N}$$

Equation 2

components, or the size and shape of an object, in a certain image using a set of numerical symbols. There are 8 directional symbols that trace horizontally, vertically, or diagonally along the shape of the selected component(s). For example, a 1 would correspond to RIGHT, a 2 corresponds to UP RIGHT, and so on. By taking the absolute value, or the magnitude component from the frequencies, of a specified number of components of the Fourier Transform of the chain code, we can produce a feature vector for the image in processing. The feature vector is a measurable characteristic of an object under observation, and is crucial to the development of a classifier. Features can otherwise be constructed or taught, however in this case it is extracted from each image in every class in the training set. By numerically representing objects and storing all their labels appropriately, the classifier is able to predict the class of an unknown input image by means of probability calculated using a Gaussian Mixture Model.

Gaussian mixture models use a collection of feature vectors to create a Gaussian for every class in the training data set. All the individual Gaussian distributions are joined together and scaled according to their priors, where their summation adds up to 1. This is clearly visible in figure 1 where the blue sinusoidal waves represent the individual Gaussians, and the red one represents the mixture model. Each Gaussian, $p(x|\theta)$ for a given feature vector x and class θ, covers a part of feature space and indicates the density of data points in that 2-dimensional space. The parameter in which a Gaussian mixture model operates in is determined by the components' means, covariances, and priors, meaning that they determine the shape of probability distribution. The covariance is a measure of the variability between two

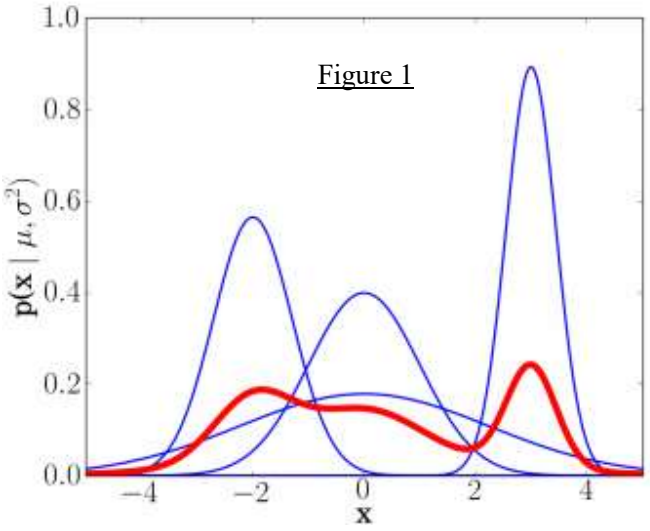

Figure 1

$$arg_\theta \max p(\theta|x) = arg_\theta \max[\log(p(\theta)) - 0.5\log(|C_\theta|) - 0.5\,(x-\mu_\theta)^T {C_\theta}^{-1}(x-\mu_\theta)]$$

Equation 3

variables in the form of a matrix. It is calculated by subtracting the mean of an image's data matrix from the data matrix itself, multiplying its transposed version by the original, and multiplying the result by one over the number of rows. The prior is the component weight, or the number of data points corresponding to one class over all of them. It indicates the probability that a new data point belongs to a class prior to calculating the maximum a posteriori (MAP) estimate. Upon the completion of the model, the MAP probability that the image being classified belongs to a certain class is determined using equation 3.

A Gaussian mixture model is the fastest algorithm for learning mixture models, and only attempts to maximise the likelihood without affecting the means negatively or the cluster sizes to have definite structures. However, it can be improved upon by using an iterative process named Expectation-Maximisation, where it alternates between two steps to converge on a more accurate maximum likelihood. The first (Expectation) step uses current estimates of parameters to predict the log-likelihood and the second

(Maximisation) step maximises the prediction by computing further parameters. Alternatively, unsupervised learning could be used instead of supervised, where all data is unlabeled, and the classifier must model the underlying distributions to learn more about the input data.

A confusion matrix was used to test the Gaussian mixture model produced. Every column in the matrix corresponds to every class in the model, and every row corresponds to every class in the test data set, in the same order. Each image in the test set was labelled to be compared against the mixture model's prediction, and if the comparison showed a correct prediction, then the corresponding element in the matrix would increment. Thus, after classifying and verifying all the classifications, the diagonal of the matrix will show the number of correctly classified images and the number of misclassifications in any other element in the matrix. The quality of the classifier can be assessed using this confusion matrix, as it shows where the algorithm went wrong and what it misclassified. Additionally, it gives us the rate of true classification.

Methodology:

To prepare the classifier to get features and train, each binary image need to be characterised using chain code. The function 'chainCode' takes an image as an input, and immediately forms a grid overlaying the input. Next, it iteratively generates the chain code using method 'findWhiteSpot' to search for and trace along the boundary of the object in the image using numerical symbols indicating direction. The function also constantly checks if it is still on the boundary as to avoid errors.

The 'getFeatures' function, shown to the right, oversees returning an N number of features as a row vector. It reads an image and converts its numeric values into logical values of 1s and 0s to easily suppress noise. Then, it proceeds to generate a chain code to convert the numerical directions into transformable angles. By taking their absolute value after they have undergone a Fast Fourier Transform, the feature vector for that image is produced with N features. This function is used in conjunction with the 'getDataMatrix' to return a data matrix for a specified class.

```
function [features] = getFeatures(imagePath, N)
    im = imread(imagePath);
    im = logical(im);

    c = chainCode(im);

    angles = c(3,:)*(2*pi/8);
    anglesFFT = fft(angles);

    absFFT = abs(anglesFFT);

    features = absFFT(1:N);
end
```

The 'getDataMatrix' function is used for training the classifier, as it returns a matrix that has a row for each feature vector of every image in a certain class. The function loops the entire list of images, calling 'getFeatures' and adds it into the matrix. In the 'train' function, shown below, a list of classes and the number of images are immediately obtained. Next, we enter a for loop that has as many iterations as there are classes. In each iteration, the class name is allocated to a variable using the previously mentioned list to store the name and the statistics that are produced afterwards into the 'models' structural array. Function 'getDataMatrix' is called upon to calculate the mean, covariance, and prior. The mean is transposed as if it isn't, no further calculations regarding the mean can be performed due to the nature of matrix multiplication. As for the covariance matrix, a function 'ensurePSD' was applied to it to prevent the matrix from being rank deficient by ensuring that it is positive and semi-definite, meaning that eigenvalues are always a little greater than zero. The prior is calculated by taking the size of the data matrix and dividing it by the number of total images. Those statistics are all stored respectively under the appropriate class in 'models' for use in classification.

```
function train(imagedir, N)
    %Train a GMM models uses shapes in imagedir using N features
    classes = getClasses(imagedir);
    totalImages = getNumImages(imagedir);
    for idx = 1:length(classes)
        class = classes{idx};
        models(idx).name = class;
        dataMatrix = getDataMatrix(imagedir, class, N);
        models(idx).mean = transpose(calcMean(dataMatrix));
        models(idx).cov = ensurePSD(calcCov(dataMatrix));
        models(idx).prior = size(dataMatrix, 1)/totalImages;
    end
    save('models');
end
```

The 'classify' function takes in the path directory of the image. Firstly, it loads the 'models' array that was created and filled out when training the classifier. All models have the same number of features, as entered when calling the 'train' function. The number is extracted for further use. The image's features are then obtained, and an arbitrary maximum score is set. The following loop identifies the correct model, performs the calculation of probability according to equation 3 and reiterates until the class with the highest probability, or score, is found. Finally, the class name is returned.

```
function [classname] = classify(imagepath)
    %Classify the image specified for the given image path
    load('models');
    %Assume all models use the same number of features
    N = length(models(1).mean);
    features = getFeatures(imagepath, N);
    maxscore = -inf;
    %Find out which class has the highest score
    for idx = 1:length(models)
        model = models(idx);
        score = log(model.prior) - 0.5*log(det(model.cov)) - 0.5 *  ~~
          ~~(features-model.mean')*inv(model.cov)*(features-model.mean')';
        if score > maxscore
            maxscore = score;
            bestidx = idx;
        end
    end
    classname = classes(bestidx);
end
```

As previously mentioned, the classifier was tested using a confusion matrix. The method 'test' is used to train the classifier, initialise and appropriately increment elements in the confusion matrix. The matrix was created to have as many columns and rows as there are classes, and all of which is filled with zeroes. Firstly, the test directory is set, and the lists of images and classes extracted. Next, the classifier is trained using the images in the training folder, and is set to observe 6 features. The flow of execution then goes onto a loop that iterates through every image in the test folder and performs a set of instructions each time that builds the confusion matrix up to completion. In each iteration, the name if the image is extracted for the classifier to find it and classify it. Then, the actual class name is labelled and the classifier predicts the label. The class index is then identified so that the appropriate element in the confusion matrix can be incremented. At the end, the confusion matrix is displayed.

```
function test()
    %test classifier using confusion matrix
    imagedir = 'images/test/';
    imagelist = dir(sprintf('%s/*.gif', imagedir));
    classes = getClasses(imagedir);
    train('images/train/', 6);
    confusionMatrix = zeros(length(classes));
    for idx = 1:length(imagelist)
        name = imagelist(idx).name;
        actual = name(1:end-7);
        predicted = classify(strcat(imagedir, name));
        index_actual = getClassIndex(actual);
        index_predicted = getClassIndex(predicted);
        confusionMatrix(index_actual, index_predicted) =
                confusionMatrix(index_actual, index_predicted) + 1;
    end
    confusionMatrix
end
```

Results:

Table 1: confusion matrix.

		Predicted					
		Alien	Arrow	Butterfly	Face	Star	Toonhead
Actual	Alien	39	0	3	0	0	0
	Arrow	0	0	2	0	0	0
	Butterfly	2	0	48	0	0	0
	Face	26	0	0	73	0	0
	Star	5	0	0	0	21	0
	Toonhead	2	0	0	0	0	0

Table 1 shows the confusion matrix as computed by my 'test' function. By adding all the diagonal elements and dividing it by the total number of elements, the rate of true classification can be calculated. In the case of my classifier, that rate is 82%. I chose to have N = 6 as the number of features observed through an iterative trial and error process. Choosing a number between 5 and 10 seemed to have a similar true classification rate of 80%-82%. Going below or above that range significantly caused more misclassifications.

$$Rate\ of\ true\ classification = \frac{correct\ classifications}{total\ classifications} * 100\% = \frac{181}{221} * 100\% = 82\%$$

The classifier clearly struggled to correctly classify the Toonheads, probably due to the fact that there are only 3 training images for that class. Such a low amount of images is insufficient for predictions to be made within the data cluster. The same applied to the class Arrow, where there are only 3 training images as well. It is clear that the classifier found the Alien class most confusing, as that column has the most misclassifications of all. The data cluster for that class must be spread broadly across the feature space, causing misclassifications with every other class except for Arrow.

Conclusion:

I have built a classifier using a Gaussian Mixture Model in MATLAB. It observes 6 features in every image and has a rate of true classification of 82%. It is a form of supervised machine learning, and could very much be optimised using Expectation-Maximisation. The confusion matrix was extremely useful in identifying how my classifier misclassified some images.

Bibliography:

[1] http://scikit-learn.org/stable/modules/mixture.html