Building and Optimising a Binary Image Classifier

With the exception of the code written by Nick Westlake and Christian Wolf, which is used with permission and cited within the source files, all work is entirely my own. I have read and adhered to the University's Examination and Assessment Offences Code of Practice statement (QA53) and the guidelines concerning plagiarism and academic integrity.

Damask Talary-Brown
BSc. Computer Science
dtstb20@bath.ac.uk

CM20220
April 23, 2015

Abstract

Gaussian Mixture Model classifiers are a simple and well known subset of pattern classifiers within the field of machine learning. In this paper, I build and optimise a GMM classifier for binary shape classes in MATLAB. Optimisations explored include the use of Zernike moments as a replacement for Fourier chain-code features, altered ratios of training and test set sizes and the oversampling of minority classes. The optimised classifier attains an average of 92.7% accuracy across 23 image classes.

Contents

1 Introduction and Background Theory **1**
1.1 Classification . . . 1
1.2 Multivariate Gaussian Distributions . . . 1
1.3 Eigenvalue Decomposition . . . 2
1.4 Feature Selection . . . 2
1.4.1 Fourier-transformed Chain Code Method . . . 2
1.4.2 Complex Zernike Moments Method . . . 3

2 Methodology **4**
2.1 Splitting Testing and Training Data . . . 4
2.2 Feature Extraction . . . 4
2.3 Estimating Gaussian Parameters . . . 5
2.4 MAP Classification . . . 5
2.5 Visualising a Confusion Matrix . . . 6
2.6 Comparing Performance . . . 6

3 Optimisation, Testing and Results **7**
3.1 Determining the Optimal Fourier Feature Vector Length . . . 7
3.2 Chain Code First-Order Differences and Normalisation . . . 7
3.3 Calculating the Optimal Ratio of Training and Testing Sets . . . 8
3.4 Synthetic Oversampling of Minority Class Data . . . 8

4 Conclusion **10**

Appendix A Additional Optimisation Data **13**

List of Figures

1.1 Example of the 8-Direction Freeman Chain Code for a binary image . . . 3

3.1 Varying feature vector length to optimise performance . . . 7
3.2 Un-normalised chain code . . . 8
3.3 Normalised 1st difference chain code . . . 8
3.4 Varying training and test ratios to optimise performance . . . 8
3.5 Standard and synthetic oversampling performance . . . 9

A.1 Images used to test rotational invariance after the alternative chain code computation . . . 13

List of Tables

3.1 Varying training and test ratios to optimise performance . . . 8
3.2 Standard and synthetic oversampling performance . . . 9

A.1 Raw data for figure 3.1 - Performance for 1-30 features . . . 13

1 Introduction and Background Theory

1.1 Classification

Classification is an important application of machine learning which aims to examine objects or patterns in data and assign labels which separate input data into distinct classes. For humans, past experience allows us to learn incrementally in order to recognise the similarities and differences between objects [1]. Training a machine to classify using existing data with known labels is known as supervised learning, whereas building a machine that can recognise patterns in unlabelled data is known as an unsupervised learning approach [2].

Typically, for a supervised approach, data is randomly (but not necessarily evenly) divided into two sets; testing and training. The classifier learns from the labels associated with each training datum and then attempts to classify the test set. Some measure of performance is generally used to assess the accuracy of the classification.

Gaussian Mixture Models (GMMs) are parametric probabilistic models, comprised of a finite number of weighted *Components*; Gaussian distributions with estimated parameters which each cover an area of the feature space in K dimensions. GMMs are a useful approach to classification because they provide a good generalisation with few parameters, but since data is assumed to be normally distributed, performance suffers when this is not the case [3]. It should be noted that there are many other parametric and non-parametric classification methods in addition to GMMs, including the simple to implement k-Nearest Neighbours[1] algorithm and the generally more accurate Support Vector Machine[2] method.

1.2 Multivariate Gaussian Distributions

To model each datum uniquely in K-dimensional feature space, points can be represented as column vectors $x = [x_1, x_2, x_3, ..., x_n]^T$. These vectors are known as feature vectors and the information they contain describes an object's important characteristics. Ideally, objects of the same class should have similar feature vectors. The parameters required for each individual multivariate Gaussian within the GMM are the population mean μ and covariance σ for that class. When these are unknown, the sample mean $\bar{x}$ and sample covariance C are used. The sample mean ($\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$) can be calculated via matrix multiplication rather than a summation using $\bar{x} = \frac{1}{N}X\vec{1}$ (where $\vec{1}$ is the column vector consisting entirely of 1s and X is the matrix whose columns x_i are individual feature vectors in class X.)

$$\bar{x} = \frac{1}{N}X\vec{1} = \frac{1}{N} \times \begin{bmatrix} x_{a1} & x_{a2} & x_{a3} & \dots \\ x_{b1} & x_{b2} & x_{b3} & \dots \\ x_{c1} & x_{c2} & x_{c3} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \end{bmatrix} = \frac{1}{N} \times \begin{bmatrix} x_{a1} + x_{a2} + x_{a3} + \dots \\ x_{b1} + x_{b2} + x_{b3} + \dots \\ x_{c1} + x_{c2} + x_{c3} + \dots \\ \vdots \end{bmatrix} = \begin{bmatrix} \bar{x}_a \\ \bar{x}_b \\ \bar{x}_c \\ \vdots \end{bmatrix} = \frac{1}{N}\sum_{i=1}^{N} x_i \quad (1.1)$$

The sample covariance is defined as follows

$$C = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T \quad (1.2)$$

The denominator $N-1$ is used rather than N as the sample covariance relies on the variance between the population mean and each observation, but the sample mean is defined in terms of all observations and is therefore marginally correlated with each [4]. A degree of freedom is lost to using the sample mean rather than the population mean in the calculation, so $N-1$ gives an unbiased estimator [5]. The likelihood of a class X_i producing a K-dimensional feature vector x is modelled by a sample Gaussian with parameters $\bar{x}_i$ and C_i:

$$P(x|X_i) = \frac{1}{(2\pi)^{K/2}|C_i|^{1/2}} \exp\left(-\frac{1}{2}(x - \bar{x}_i)^T C_i^{-1}(x - \bar{x}_i) \right) \quad (1.3)$$

[1] http://www.statsoft.com/textbook/k-nearest-neighbors
[2] http://www.statsoft.com/textbook/support-vector-machines

Maximising the above, i.e. calculating $\text{argmax}_i\, P(x|X_i)$ gives us a maximum likelihood (ML) estimate. A maximum a posteriori (MAP) estimate can be found by maximising $\text{argmax}_i\, P(X_i|x)$ which is the posterior probability, defined by Bayes' Theorem [6] as follows:

$$P(X_i|x) = \frac{P(x|X_i)P(X_i)}{P(x)} \text{ where } P(X_i) \text{ is the prior for class } X_i. \tag{1.4}$$

The class priors are the probabilities that any given image will belong to a certain class before knowing anything about the image such as its feature vector [7]. This is simply the proportion $\frac{N_x}{N}$ of N_x training images for each class X out of the total set of N images. The sum of all the class priors is therefore 1. The denominator $P(x)$ (known as the *evidence*) can be ignored in the maximisation calculation, as it does not depend on the parameters X_i in respect to maximising the right hand side of the equation [7]. We therefore only need to calculate $\text{argmax}_{i\in\{1,\ldots,K\}}\, P(x|X_i)P(X_i)$. In practice, it is easier to work with larger numbers so the natural logarithm of each posterior is calculated.

1.3 Eigenvalue Decomposition

In the case of some classes, classes which have less images in their testing sets than the dimensionality of the classifier's feature space, the covariance matrix will be rank deficient. Any $n \times n$ matrix with rank $< n$ is singular and therefore not invertible [8], but the inverse of C is required to calculate the Mahalanobis distance for each Gaussian.

The Eigenvalue Decomposition (EVD) of a matrix A gives the expansion of A as a product of its eigenvalues and eigenvectors.

$$A = ULU^T \text{ where } U \text{ is orthonormal } (U^T = U^{-1}). \tag{1.5}$$

U is the matrix of eigenvectors of A, and L is a square KxK matrix with normalised eigenvalues on the leading diagonal [9] and zeroes everywhere else. Since the rank of a matrix is equal to the number of non-zero eigenvalues of that matrix, for small classes, some elements on the diagonal be zero. By replacing these elements in L with a larger number to give L_1, A can be recomputed as UL_1U^T and will have full rank, meaning the inverse of the covariance can be computed.

1.4 Feature Selection

Feature selection, that is, choosing a subset of relevant attributes which consistently provide a distinction between classes without overfitting, is an important area of research. The two methods for feature selection that I investigated are filtered chain code and Zernike moments, which take significantly different approaches to extracting features from binary images.

1.4.1 Fourier-transformed Chain Code Method

Since the binary shapes in the testing and training sets are composed of a continuous area of black on a white background or vice-versa, the images can be encoded by the direction of the boundary between the two colours, which gives a chain code. After removing noise and truncating the resulting frequencies, this chain code can be used as a set of features.

First, the 8-direction Freeman chain code [10] (Figure 1.1) is computed for each image. This provides a set of features for the class with positional invariance [11], since the chain code directions alone could not be used to recreate the position of the original shape. The Fast-Fourier Transform of each chain code is then filtered using a top hat filter, which removes high-frequency noise that could lead images of the same class having significantly different feature vectors.

The absolute value of each of the discrete set of Fourier coefficients is taken, which removes the phase component [12] (the complex argument) and makes the data rotationally invariant. The largest N positive and negative frequencies become the feature vectors. This truncation occurs because "reducing the dimension of the feature

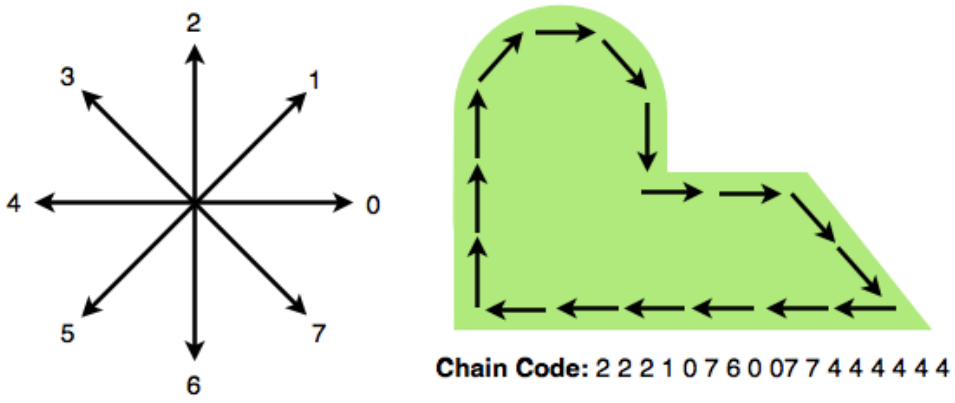


Figure 1.1: Example of the 8-Direction Freeman Chain Code for a binary image

space is necessary to infer reliable conclusions"[13] when working with a large number of possibly explanatory variables such as the filtered transformed chain code, in order to avoid overfitting.

1.4.2 Complex Zernike Moments Method

Zernike moments are projections of images onto a set of orthogonal complex polynomials (Zernike polynomials [14]), which form a complete basis on the unit circle [15]. There is no overlap in the information represented by each moment, as all moments are orthogonal. An additional benefit to this is increased robustness against high-frequency noise [16]. For these reason, a series of 1-n$^{\text{th}}$ degree Zernike moments can make suitable feature vectors for classification. A major advantage of Zernike descriptors, which describe image regions, over chain-code descriptors, which can only describe contours, is that Zernike descriptors are not limited to simple shapes with single connected regions [17].

The Zernike moment of order $n \in \{0, 1, 2, ..., \infty\}$ and repetition m (with $n - |m| \bmod 2 = 0$ and $|m| \leq n$) for a digital image function $f(x, y)$ is defined as

$$A_{nm} = \frac{n+1}{\pi} \sum_x \sum_y f(x,y) V^*_{nm}(x,y) \quad \text{where } x^2 + y^2 \leq 1. \tag{1.6}$$

This is a generalised version of the definition of Zernike moments for continuous image functions, which is

$$A_{nm} = \frac{n+1}{\pi} \int_x \int_y f(x,y) V^*_{nm}(x,y)\, dx\, dy \quad \text{where } x^2 + y^2 \leq 1. \tag{1.7}$$

In both cases, V^*_{nm} is the complex conjucate of the Zernike polynomial of the same order and repetition [18]. The Zernike polynomials, which act as basis functions, can be expressed in polar co-ordinate form as

$$V_{nm}(r, \theta) = R_{nm}(r) e^{im\theta} \quad \text{where } i = \sqrt{-1} \text{ and } \theta = \tan^{-1}\left(\frac{y}{x}\right). \tag{1.8}$$

$R_{nm}(r)$ is the orthogonal radial polynomial [19], defined as

$$R_{nm}(r) = \sum_{s=0}^{\frac{n-|m|}{2}} (-1)^s \frac{(n-s)!}{s!(\frac{n+|m|}{2} - s)!\,(\frac{n-|m|}{2} - s)!} r^{n-2s} \text{ [20]} \tag{1.9}$$

Taking the absolute value of each complex moment (in the same manner as for the filtered Fourier coefficients) disregards the complex argument of the moment and therefore the phase, making the resulting feature rotationally invariant. It has been observed that, although phase coefficients contain much of an image's information (therefore rotational invariance should in theory come at the cost of weakened performance), magnitude-only Zernike moment descriptors still outperform many other moment descriptors [18].

2 Methodology

2.1 Splitting Testing and Training Data

The first stage of building the classifier was to separate data into training data (from which the classifier would estimate means and covariances for each class) and testing data, which it would then attempt to classify. I used various combinations of the two sets of binary image classes as many of my tests examined the differences in performance of the classifier for classes of different sizes. To ensure the split was random across each class, I generate a random permutation for each class directory and apply it to the file names to randomise the order of images before splitting them into test and training sets.

```
% Randomly shuffle image names
shuffled = randperm(numel(images));
images = reshape(images(shuffled), size(images));
```

Initially I chose a training to testing set ratio of 50:50 without much consideration for the implications of this, but I later optimised it to 75:25, the mathematical justification for which can be found in Section 3.3.

```
% split the training and testing data in a 75:25 ratio
testSet = images(ceil(3*length(images)/4)+1:end);
images = images(1:ceil(3*length(images)/4));
```

2.2 Feature Extraction

For the means and covariances needed to be calculated from the training data for each class, I needed to select a subset of useful features from all the images without overfitting. I experimented with two different methods of feature extraction both shown below; filtered Fourier-transformed chain codes and Zernike moments, respectively. My decision to ultimately use Zernike descriptors rather than Fourier chain code descriptors was based on the results in Section 2.6.

```
% Get the chain code for the image
angles = chainCode(im);
angles = (2*pi/8) * angles(3,:);
% Take the fourier transform
anglesFFT = fft(angles);
% Filter using a 'top hat' filter.
filter = zeros(size(angles));
% Both the positive and negative low frequencies must be kept
filter(1:FV_SIZE) = 1;
filter(end-FV_SIZE+2:end) = 1;
filteredFFT = anglesFFT .* filter;
% abs() gets rid of the phase component (the complex argument)
featureVector = (abs(filteredFFT(1:FV_SIZE)))';
```

To use Zernike descriptors, each image is transformed to be centred on a square of uniform size. To compute the Zernike basis functions and moments from the set of training data, I used MATLAB functions written by Christian Wolf, publicly available from his website[1]. The Zernike basis functions up to degree 6 are calculated, which are used to compute the Zernike moments for each image. The set of absolute values of the Zernike moments up to degree 6 provide a 16-dimensional feature vector.

```
imsize = 45;
% The image needs to be centred on a unit square
m = centersquare(im,imsize);
% abs() gets rid of the phase component (the complex argument)
% Calculate the Zernike moments up to degree 6
fv = abs(zernike_mom(m,zernike_bf(imsize,6)));
featureVector = fv ./ sum(fv);
```

[1] http://liris.cnrs.fr/christian.wolf/software/zernike/

2.3 Estimating Gaussian Parameters

Once I had a method of feature extraction and classes of images divided into testing and training sets, I could compute the estimators for the mean and covariance for each class, as well as returning other information such as the size of the training set in order for the prior to be computed.

```
% Compute the sample mean
mu = allFeatureVecs * ones(length(images),1) / length(images);
C = zeros(FV_SIZE);
for idx = 1:maxTrainingSet
   C = C + (allFeatureVecs(:,idx) - mu) * (allFeatureVecs(:,idx) - mu)';
end
% This is the sample covariance matrix (with Bessel's correction applied)
% Change length(images)-1 to length(images) to use the biased estimator
C = C / (length(images)-1);
```

The feature vectors x_i for each image in the class are used to compute the sample mean vector $\bar{x}$. (In the code I call this variable μ, since it acts as an estimator for the population mean.) The sample covariance matrix C, (which is an unbiased estimator for the population covariance matrix) is computed by iterating through the images, summing the products of each feature vector minus the mean by its transpose.

After the parameters for every class have been estimated, the size of each training set is divided by the total number of training images giving the class priors, which sum to 1.

2.4 MAP Classification

Once all the class parameters had been estimated, the classifier could then iterate through every class for any given image and compute the log posterior probability $ln(P(x|X_i) + ln(P(X_i)$, disregarding the evidence $P(x)$.

```
% Choose the class with the highest posterior probability
    maxClass = struct('idx',0,'probability',0);
    for idx = 1:length(classData)
        cdx = classData(idx);
        % Use the pseudoinverse because c is singular or close to singular
        p = exp(-1/2 * (fv-cdx.xbar)' * pinv(cdx.c) * (fv-cdx.xbar)) / ...
            ((sqrt(det(cdx.c)))*2*pi^(F_V_SIZE/2));
        % Taking logs lets us work with bigger numbers
        prob = log(p) + log(classData(idx).prior);
        if idx == 1
            maxClass.probability = prob;
            maxClass.idx = 1;
        elseif prob > maxClass.probability
            maxClass.probability = prob;
            maxClass.idx = idx;
        end
    end
    classification = maxClass.idx;
```

For classes whose covariance matrices were rank deficient, I initially used the Moore-Penrose pseudoinverse [21] in my code rather than the standard inverse. However, I found on average that manually adding eigenvalues to the matrix to inflate it to full rank as in Section 1.3 performed 15% more accurately than using the pseudoinverse.

```
if rank(C) < FV_SIZE
    [U,L] = eig(C);
    % U is the matrix of eigenvectors, L is the matrix of normalised eigenvalues
    L = diag(L); % Get the leading diagonal of E
    L(L < 0.0001) = 0.0001; % Replace tiny diagonal elements with a larger value
    L = diag(L); % Turn it back into a square matrix
    C = U * L * U';
end
```

The classification returned by the code is an integer; the index of the predicted class in the struct which contains all classes initially seen.

2.5 Visualising a Confusion Matrix

A confusion matrix is computed for the testing set as a whole, with each row indicating the actual class of the image and the column indicating the predicted class. Ideally, this matrix would be diagonal, as every image would be classified correctly [22]. Since my classify function returns an index rather than a class name, the confusion matrix is easily populated by iterating through each class, and within that loop, through each image.

```
    for idx = 1:classNum
        for idx2 = 1:length(classData(idx).testSet)
            predicted = classify(classData,idx,idx2,FV_SIZE);
            confusionMat(idx,predicted) = confusionMat(idx,predicted) + 1;
        end
    end
```

A simple and computationally light measure of performance is to calculate the sum of elements lying on the diagonal divided by the total number of elements, formally, using Iverson bracket notation[2]:

$$\text{Performance} = \frac{100}{N}\sum_{i=1}^{N}[\hat{X}(x_i) = X(x_i)] \text{ where } \hat{X} \text{ is the predicted class and } X \text{ is the actual class.} \quad (2.1)$$

```
% Calculate the percentage of elements which lie on a matrix diagonal
% For confusion matrices, this is the accuracy of classification
function spread = matrixSpread(cMatrix)
    count = sum(sum(cMatrix));
    spread = 100*(1 - (trace(cMatrix)/count));
end
```

This gives the percentage of images that were classified correctly, and is the measure I use for the testing and optimisation section of this report.

2.6 Comparing Performance

Using the covariance pseudoinverse rather than EVD to inflate the covariance results in poor accuracy for classes with extreme differences in size, which I attempted to address in Section 3.4. The average accuracy attained over 10 attempts classifying `Alien`, `Arrow`, `Butterfly`, `Face`, `Star` and `Toonface` was 92.7273%. The average accuracy attained for 10 attempts classifying the SIID set[3] of 17 classes was 88.2353%. Classifying the two sets together with 23 classes attained an average accuracy of 91.9255%. These performance statistics are with the optimisations in the following section applied.

For the set of large classes I consistently used throughout development (`Alien`, `Arrow`, `Butterfly`, `Face`, `Star`, and `Toonface`), there was no significant difference in average performance between chain code/Fourier descriptors (91.7622%) and Zernike moment descriptors (92.7273%.) Zernike moments also performed fractionally worse for sets with high variance in class size. However, for the SIID set of classes which all contained 12 images (`Bird`, `Brick`, etc.) Zernike moment description performed consistently in the order of 40% more accurately than the Fourier transformed chain code descriptors.

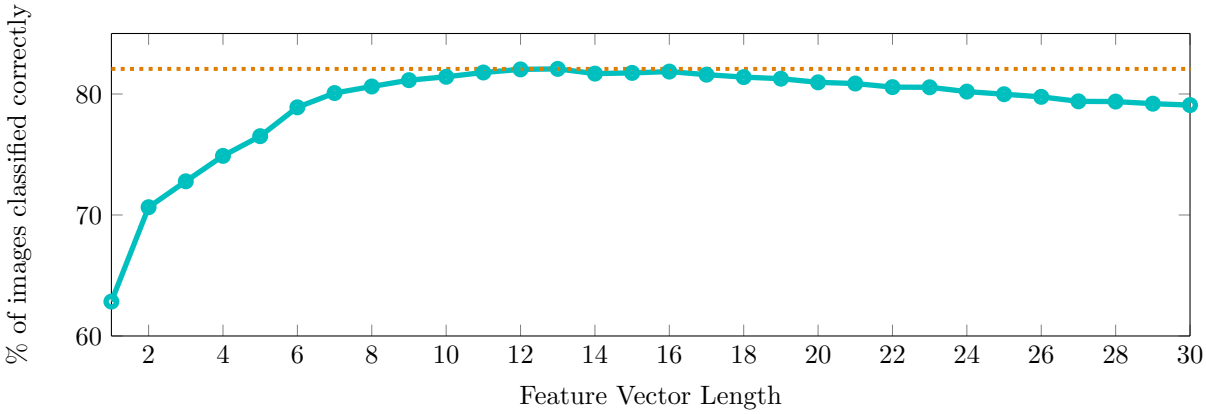
[2] http://mathworld.wolfram.com/IversonBracket.html
[3] 216 shapes from http://www.lems.brown.edu/vision/researchAreas/SIID/

3 Optimisation, Testing and Results

3.1 Determining the Optimal Fourier Feature Vector Length

The first experiment I conducted was to measure the performance of the classifier using Fourier descriptors while varying the size of the feature vectors between 1 and 30. The curse of dimensionality (a common problem arising from multivariate data analysis) means in practice that there exists a maximum feature vector size for any given sample which, if exceeded, will degrade the performance of a classifier rather than further increasing it [20]. High dimensionality data in statistical analysis and pattern recognition is also considered a problem due to the computational cost and associated time required to process it [23], alongside the previously mentioned problems caused to the covariance when the vector space dimensionality is higher than that of the the sample space [8]. The results of my testing can be seen in figure 3.1 with corresponding data in Appendix A.1, showing a maximum for performance of 82.07% correctness with 13 features and a test/training set ratio of 50:50.

Figure 3.1: Varying feature vector length to optimise performance



The testing and training data was taken from the four classes (`Alien`, `Butterfly`, `Face` and `Star`) with a total 219 images in the combined training sets. The ratio of optimal features to sample size is therefore 16.8:1, which is similar to that found by Liu and Wan [24], whose optimal ratio for GMM classification was 15.9:1 and whose performance against features data was similarly positively skewed.

3.2 Chain Code First-Order Differences and Normalisation

Another optimisation I attempted and tested was to calculate the first-order difference [25] of the chain code in an attempt to improve rotational invariance. I tested this by creating a new image directory, containing four versions of the same image, all altered in some way, either by rotation, mirroring, scale alteration or a combination of the three (see Appendix A.) The chain code was then replaced by a new code of the differences between each element and the subsequent one in the original code. For example, the 8-direction chain code 222446567 would produce the clockwise first-order difference 00202711. Using the original chain code computation produced significantly dissimilar feature vectors for each altered image, which was undesirable since they should all have been classified as arrows. The difference in the first feature alone between two of the vectors was over 2000. After normalising their first-order differences, the chain codes and therefore the feature vectors became orders of magnitude more tightly grouped. Note the scales of the graphs below to observe the difference in variation of individual feature vectors.

In theory, I would have expected for the altered chain code computation to improve the classifier's performance. In practice however, it actually performed on average 4.41% worse than its counterpart. I confirmed that this was due to the effects of the Fourier Transform by taking the average performance across both methods while bypassing the FFT and filtering, which showed the normalised first difference performed 31.2% better than the default method. However, both methods still performed significantly worse than when using the filtered FFT.

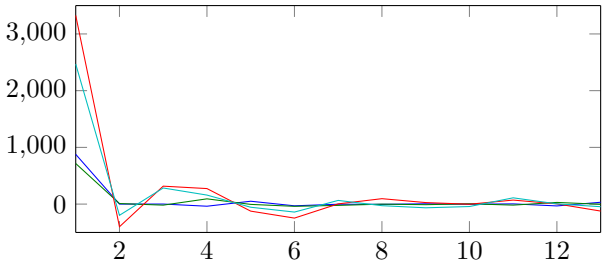


Figure 3.2: Un-normalised chain code

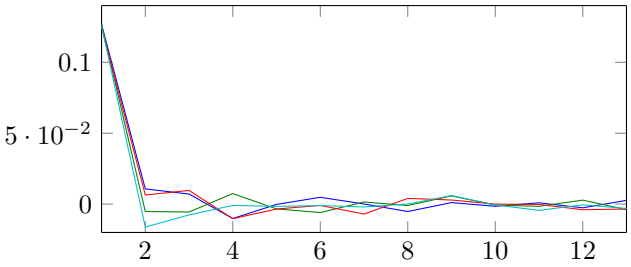


Figure 3.3: Normalised 1st difference chain code

3.3 Calculating the Optimal Ratio of Training and Testing Sets

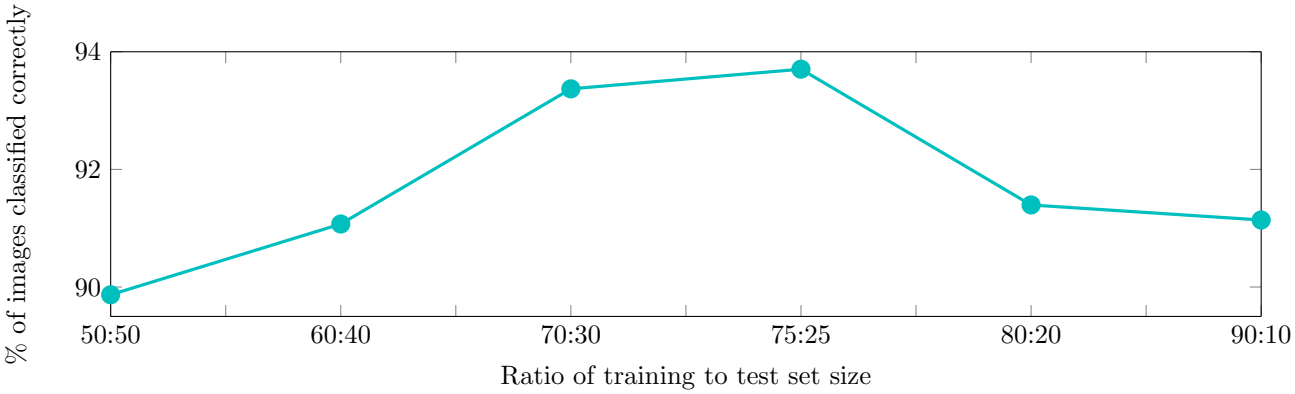
When I first started building the classifier, I used a training to test set ratio of 50:50. However, after researching this area of optimisation further, I experimented with varying the ratio as high as 90:10. A concern with extreme ratios was that if the training set were too large then the classifier's performance would have a large variance, but if the training set were too small then there would be too much variance in the estimation of the mean and covariance for each class [26].

Ultimately, I chose a training to testing set ratio of 75:25, based on the work of Guyon [27], who determined that the size of the testing set should be close to inversely proportional to the square root of the Vapnik-Chervonenkis (VC) dimension[1] (for my classifier, the recommended ratio would in reality be closer to 73:27, due to the fact that the VC-dimension for d-dimensional linear classifiers is $d + 1$ [28].) The average performance across 10 randomised selections for the two sets at ratios between 50:50 and 90:10 (for the classes `Alien`, `Butterfly`, `Face` and `Star`) peaked at a ratio of 75:25, as is illustrated in figure 3.4.

Table 3.1: Varying training and test ratios to optimise performance

Training to Testing Set Ratio	Percentage of images classified correctly
50:50	89.8684
60:40	91.0714
70:30	93.3693
75:25	93.7037
80:20	91.3953
90:10	91.1394

Figure 3.4: Varying training and test ratios to optimise performance



3.4 Synthetic Oversampling of Minority Class Data

Another optimisation I tested relates to the significant differences in class prior probabilities when attempting to classify images of type `Arrow` or `Toonface` alongside those of classes up to 20 times the size such as `Butterfly`.

[1] http://mathworld.wolfram.com/Vapnik-ChervonenkisDimension.html

This unequal distribution is known as between-class imbalance [29] and cause serious deterioration in classifier performance even for class ratios not considered 'extreme' (1:100) [30]. Class imbalances further complicate the dimensionality problem I had already encountered (my optimal feature vector length is over twice the size of the smallest classes), since "the combination of small sample size and high dimensionality hinders learning because of difficultly involved in forming conjunctions over the high degree of features with limited samples [29]." Japkowicz and Stephen [31] also concluded that class imbalances had a greater effect on small sets.

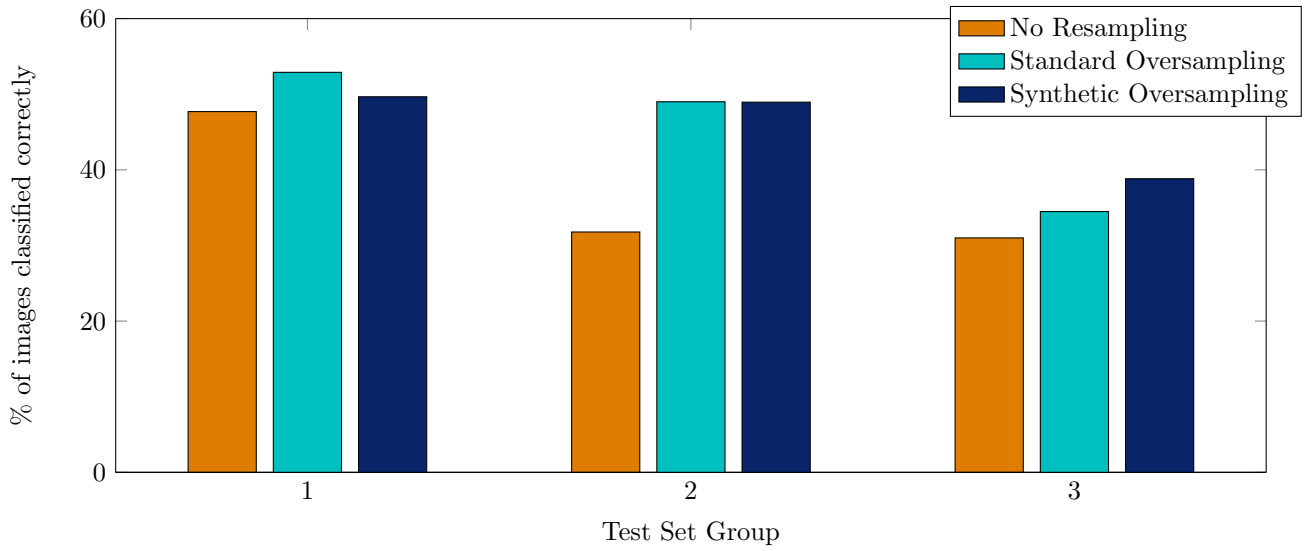
There are three simple high-level approaches to rebalancing classes; oversampling minority classes, undersampling majority classes and synthetic minority oversampling (SMOTE [32].) Of these, undersampling majority classes has been the most heavily criticised, since it "may throw out potentially useful data [30]." I therefore decided to experiment by recording the average classification performance without covariance rank inflation using standard oversampling (equivalent to sampling with replacement) and synthetic oversampling (interpolating between data by taking the average of two random feature vectors within the class to create a composite) with three subsets of five classes.

Group I: `Alien`, `Arrow`, `Butterfly`, `Face`, `Toonface`. Range: 194. Standard deviation: 80.3387
Group II: `Alien`, `Arrow`, `Camel`, `Star`, `Toonface`. Range: 79. Standard deviation: 35.6749
Group III: `Arrow`, `Bird`, `Camel`, `Key`, `Toonface`. Range: 7. Standard deviation: 3.8341

Table 3.2: Standard and synthetic oversampling performance

Sampling Method	Group I Performance	Group II Performance	Group III Performance
No Oversampling	47.7045	31.7808	30.9986
Standard Oversampling	52.8952	49.0041	34.4802
Synthetic Oversampling	49.6632	48.9487	38.8258

Figure 3.5: Standard and synthetic oversampling performance



My results found that for every group, both oversampling methods performed with a higher degree of accuracy than the classifier with no oversampling. As expected, performance in general decreased for the groups with larger ranges and standard deviations. Synthetic oversampling performed better then standard as the range of each group size decreased, but for the larger ranges, standard oversampling produced the best performance. It was unsurprising that there was no one optimal oversampling method, since studies have shown that often two-class techniques (such as SMOTE) are not as effective when applied to multi-class classification [30].

4 Conclusion

In this paper, I detailed the methodology behind building a Gaussian Mixture Model binary image classifier and implemented a series of optimisations.

The combination of optimisations which I found lead to the most significant increase in performance were using Zernike moments as features rather than Fourier chain code descriptors, using a training set to testing set ratio of 75:25 based on the VC-dimension of d-dimensional linear classifiers, inflating covariance matrices to full rank using EVD and using sampling with replacement to attempt to mitigate the problems caused by minority classes having significantly smaller priors.

When using Fourier chain code features, I found the optimal feature vector length to be 13, with accuracy reduced upwards of 17 features and significantly reduced below 7. Taking the first difference of the chain code did not improve classification accuracy for rotated images due to the effect of the Fast-Fourier transform.

For both methods of feature extraction, oversampling increased classification accuracy, most notably for groups of classes with high variance in class size. Synthetic oversampling of minority classes performed with more accuracy than sampling with replacement for groups with extreme low variance, but was outperformed by sampling with replacement for groups with higher variance.

Using eigenvalue decomposition to add artificial eigenvalues and therefore inflate the covariance matrix to full rank had a significant ($> 10\%$) effect on performance over using the Moore-Penrose pseudoinverse in the Gaussian calculation, as well as an increase in speed, since the pseudoinverse function in MATLAB is an expensive operation computationally.

Areas I would consider for further detailed research include investigating the effect of altering the degree of Zernike moments calculated, and using a variation on Zernike descriptors such as the independent component analysis Zernike moment shape descriptors (ICAZMSD) method proposed by Androutsos and Mei [17], which uses independent component analysis techniques[33] to give a canonical form which provides features invariant under affine transformations.

An alternative potential research area would be an investigation into the weighting of significant features in order to better differentiate between classes, and finding the optimal threshold at which eigenvalues should be replaced during rank inflation to provide a non-singular covariance matrix.

References

[1] S. Marzukhi, W. N. Browne, and M. Zhang, "Two-cornered learning classifier systems for pattern generation and classification," in *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '12. New York, NY, USA: ACM, 2012, pp. 895–902. [Online]. Available at: http://doi.acm.org/10.1145/2330163.2330287

[2] S. Amershi and C. Conati, "Unsupervised and supervised machine learning in user modeling for intelligent learning environments," in *Proceedings of the 12th International Conference on Intelligent User Interfaces*, ser. IUI '07. New York, NY, USA: ACM, 2007, pp. 72–81. [Online]. Available at: http://doi.acm.org/10.1145/1216295.1216315

[3] Z. Ma, "Non-gaussian statistical modelsand their applications," 2011.

[4] S. So, "Why is the sample variance a biased estimator?" Signal Processing Laboratory, Griffith School of Engineering, 2008. [Online]. Available at: http://maxwell.ict.griffith.edu.au/sso/biased_variance.pdf

[5] D. Barber, *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[6] M. F. Triola, "Bayes' Theorem," 2010. [Online]. Available at: http://faculty.washington.edu/tamre/BayesTheorem.pdf

[7] A. Kak, "ML, MAP, and Bayesian - The Holy Trinity of Parameter Estimation and Data Prediction."

[8] G. H. Tucci and K. Wang, "An innovative approach for analysing rank deficient covariance matrices," in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 2596–2600.

[9] H. Abdi, "The eigen-decomposition: eigenvalues and eigenvectors."

[10] L.-D. Wu, "On the chain code of a line," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-4, no. 3, pp. 347–353, May 1982.

[11] R. C. Gonzales and P. Wintz, *Digital Image Processing (2Nd Ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1987.

[12] F. Lin and R. D. Brandt, "Towards absolute invariants of images under translation, rotation, and dilation," *Pattern Recognition Letters*, vol. 14, no. 5, pp. 369–379, 1993.

[13] S. Gadat and L. Younes, "A stochastic algorithm for feature selection in pattern recognition," *J. Mach. Learn. Res.*, vol. 8, pp. 509–547, May 2007. [Online]. Available at: http://dl.acm.org/citation.cfm?id=1248659.1248678

[14] E. W. Weisstein, "Zernike polynomial," Wolfram Research, Inc., 2002. [Online]. Available at: hhttp://mathworld.wolfram.com/ZernikePolynomial.html

[15] A. Tahmasbi, F. Saki, H. Aghapanah, and S. B. Shokouhi, "A novel breast mass diagnosis system based on zernike moments as shape and density descriptors," in *Biomedical Engineering (ICBME), 2011 18th Iranian Conference of*. IEEE, 2011, pp. 100–104.

[16] T.-W. Lin and Y.-F. Chou, "A comparative study of zernike moments," in *Web Intelligence, 2003. WI 2003. Proceedings. IEEE/WIC International Conference on*, Oct 2003, pp. 516–519.

[17] Y. Mei and D. Androutsos, "Robust affine invariant shape image retrieval using the ica zernike moment shape descriptor," in *Image Processing (ICIP), 2009 16th IEEE International Conference on*. IEEE, 2009, pp. 1065–1068.

[18] S. Li, M.-C. Lee, and C.-M. Pun, "Complex zernike moments features for shape-based image retrieval," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 39, no. 1, pp. 227–237, 2009.

[19] D. R. Iskander, M. R. Morelande, M. J. Collins, and B. Davis, "Modeling of corneal surfaces with radial polynomials," *Biomedical Engineering, IEEE Transactions on*, vol. 49, no. 4, pp. 320–328, 2002.

[20] J. D. Shutler and M. S. Nixon, "Zernike velocity moments for sequence-based description of moving features," *Image and Vision Computing*, vol. 24, no. 4, pp. 343–356, 2006.

[21] E. W. Weisstein, "Moore-penrose matrix inverse," Wolfram Research, Inc., 2002. [Online]. Available at: http://mathworld.wolfram.com/Moore-PenroseMatrixInverse.html

[22] K. Ting, "Confusion matrix," in *Encyclopedia of Machine Learning*. Springer US, 2010, pp. 209–209.

[23] A. Janecek, W. N. Gansterer, M. Demel, and G. Ecker, "On the relationship between feature selection and classification accuracy," in *FSDM*, 2008, pp. 90–105.

[24] M. Liu and C. Wan, "Feature selection for automatic classification of musical instrument sounds," in *Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries*, ser. JCDL '01. New York, NY, USA: ACM, 2001, pp. 247–248. [Online]. Available at: http://doi.acm.org/10.1145/379437.379663

[25] K. Fating and A. Ghotkar, "Performance analysis of chain code descriptor for hand shape classification," *International Journal of Computer Graphics & Animation*, vol. 4, no. 2, April 2014. [Online]. Available at: http://airccse.org/journal/ijcga/papers/4214ijcga02.pdf

[26] S. E. of Mathematics, "Law of large numbers," 2001. [Online]. Available at: http://www.encyclopediaofmath.org/index.php/Law_of_large_numbers

[27] I. Guyon, "A scaling law for the validation-set training-set size ratio," in *AT & T Bell Laboratories*, 1997.

[28] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, "Classifying learnable geometric concepts with the vapnik-chervonenkis dimension," in *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '86. New York, NY, USA: ACM, 1986, pp. 273–282. [Online]. Available at: http://doi.acm.org/10.1145/12130.12158

[29] H. He and E. A. Garcia, "Learning from imbalanced data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 9, pp. 1263–1284, 2009.

[30] V. G. J. S. R. Mollineda and R. A. J. Sotoca, "The class imbalance problem in pattern classification and learning," *Pattern Analysis and Learning Group*.

[31] N. Japkowicz and S. Stephen, "The class imbalance problem - a systematic study," *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429–449, Oct. 2002. [Online]. Available at: http://dl.acm.org/citation.cfm?id=1293951.1293954

[32] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, no. 1, pp. 321–357, 2002.

[33] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent component analysis*. John Wiley & Sons, 2004, vol. 46.

A Additional Optimisation Data

Table A.1: Raw data for figure 3.1 - Performance for 1-30 features

Features	Percentage of images classified correctly
1	62.8440366972477
2	70.6422018348624
3	72.782874617737
4	74.8853211009174
5	76.5137614678899
6	78.8990825688073
7	80.0786369593709
8	80.6192660550459
9	81.1416921508665
10	81.4220183486239
11	81.7764804003336
12	82.0336391437309
13	82.0748059280169
14	81.6841415465269
15	81.743119266055
16	81.8520642201835
17	81.5974096060443
18	81.3965341488277
19	81.2650893288267
20	80.9633027522936
21	80.86500655308
22	80.5671392827356
23	80.5544475468688
24	80.1987767584098
25	79.9816513761468
26	79.7635850388144
27	79.3917770981991
28	79.3741808650066
29	79.1996203732996
30	79.0825688073394

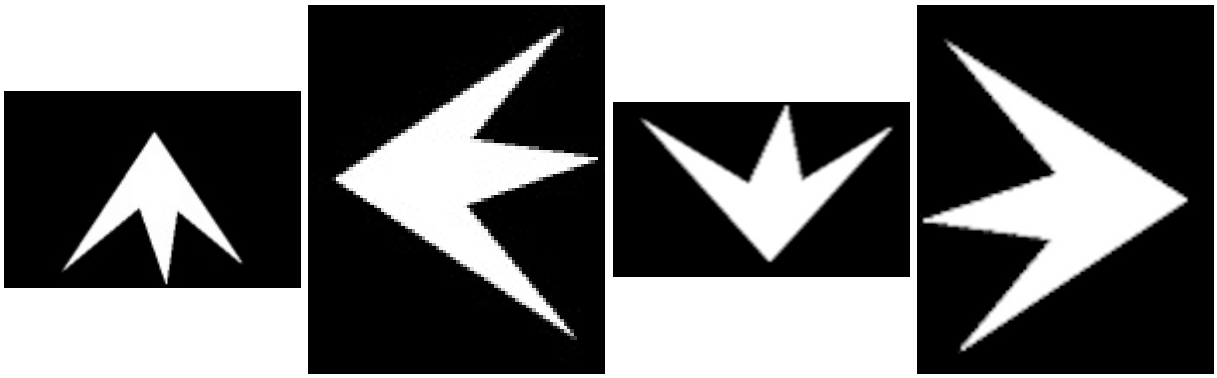


Figure A.1: Images used to test rotational invariance after the alternative chain code computation