CS 195-5: Machine Learning
Problem Set 5

Douglas Lanman
dlanman@brown.edu
26 November 2006

1 Clustering and Vector Quantization

Problem 1

Part 1: In this problem we will apply Vector Quantization (VQ) to image compression. Using the Matlab function `learnVQ.m`, run VQ on `imagePS5.png` using $k = 256$ clusters with blocks of size $m = 3$. Compare the number of bits required to store the reconstructed image versus the original.

Let's begin by briefly reviewing the Matlab routines required for VQ-based image compression. First, note that the VQ parameters are set on lines 13-19 of `prob1.m`. The general VQ framework is implemented by `learnVQ.m`, which uses the subroutine `kmeans.m` to implement the k-means algorithm. (Note that, as recommended, we use `findnn.m` and `lpnorm.m` within the k-means implementation to allow flexible nearest-neighbor selection under a general L_p-norm.)

Applying VQ to `imagePS5.png` demonstrates that this algorithm can achieve significant compression ratios. First, note that the original image has dimensions 510×768 pixels and is represented using 24-bit color (i.e., 3 bytes per pixel). As a result, `imagePS5.png` requires $510 \cdot 768 \cdot 24 =$ 9,400,320 bits to store. In contrast, the VQ cluster means (using $k = 256$ and $m = 3$) only require $256 \cdot 3 \cdot 3 \cdot 24 =$ 55,296 bits to store. In order to reconstruct the original image we must assign an 8-bit index into the VQ cluster means "codebook" for each 3×3 block, which requires an additional $(510/3) \cdot (768/3) \cdot 8 =$ 348,160 bits. As a result, the compressed image requires a total of 403,456 bits to store, leading to the following compression ratio.

$$\text{compression ratio} \triangleq \frac{\text{compressed size}}{\text{original size}} = \frac{\text{403,456 bits}}{\text{9,400,320 bits}} = 0.0429$$

In conclusion, we find that the compressed image only requires 4.3% as many bits as the original.

Part 2: Display the original image, the VQ reconstruction, and the cluster means (using the provided `showBlockMeans.m` function). What can you say about the compression results? If there are artifacts you find undesirable, what change in parameters of the VQ would reduce their effect?

The original image, the compressed image, and the cluster means are shown in Figures 1, 2 and 4. Note that, for lossy image compression, the quality of a given scheme is subjective; does a human observer perceive significant differences between the original and compressed images? In order to gauge the perceptual differences, we evaluated the Euclidean difference in the $L^*a^*b^*$ color space [6]. As shown in Figure 3, the boundaries between neighboring blocks are visually apparent – since VQ does not guarantee smooth border transitions. These artifacts could be reduced by increasing the total number of blocks or by reducing the size of individual blocks. Either solution would require increasing the size of the VQ codebook and reduce the compression ratio. This demonstrates the central tradeoff between codebook complexity and reconstruction accuracy in VQ compression.

Figure 1: Original image from `imagePS5.png`.

Figure 2: Image reconstructed using Vector Quantization (VQ) with 256 3×3 blocks.

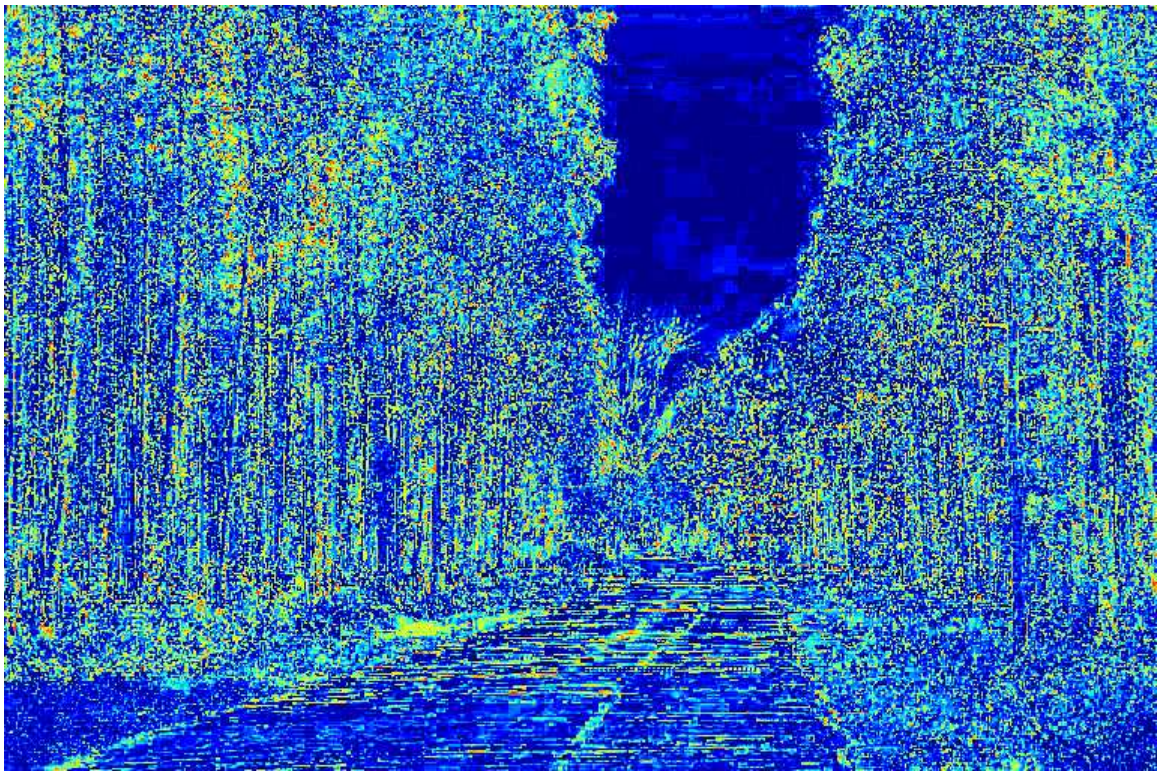


Figure 3: Difference between original image and VQ reconstruction (in the $L^*a^*b^*$ color space) [6]. Red pixels correspond to large reconstruction errors, whereas blue pixels represent small errors.

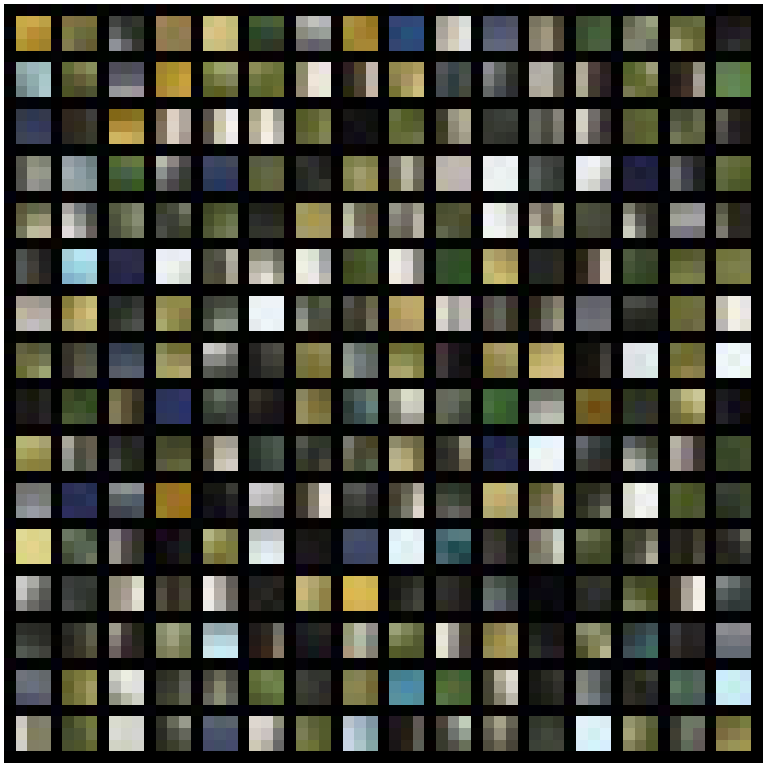


Figure 4: Mosaic containing the 256 3×3 blocks selected by k-means clustering. Note that Figure 2 was constructed by estimating the best block to represent each original 3×3 region in Figure 1.

Problem 2

If we want to store/transmit many images, it would be costly to run VQ for each image separately. Describe a scheme that significantly reduces both the computation time and storage/communication costs for this scenario. What assumptions are necessary to make this scheme work well?

In the previous problem, we found that VQ could reduce the storage/transmission requirements for a single image to 4.3% of the number of bits in the original. While this reduction is impressive, we also found experimentally that VQ is rather slow. For example, `imagePS5.png` required k-means be applied to over $(510/3) \cdot (768/3) = 43,520$ samples, each with 27 dimensions. If we want to store/transmit many images, then it is not computationally feasible to run VQ separately for each image. Instead, we need to develop a method for selecting either one or several codebooks which capture the variation within the entire image set. Given these codebooks, we note that the relatively efficient nearest-neighbor algorithm can be used to assign cluster means to each image block. In the following paragraphs, we propose three schemes: two for uncorrelated images and one specifically-designed for video sequences.

Let use begin by studying the case in which the images are not expected to be correlated. For example, suppose a user wants to store or transmit a large photo collection. In general, we do not expect strong inter-image correlations across the entire image set. Note, however, that some images may be taken in similar locations and, as a result, are correlated. We propose first clustering the images using k-means, where the individual samples correspond to vector representations of the entire original images (or down-sampled versions of the originals). This process will naturally identify correlated image sets. Within each cluster we will select an exemplar image which is closest to the mean (alternatively we can cluster using the k-mediods algorithm). Afterwards, we propose applying VQ to each exemplar and compressing the remaining images within each cluster using the cluster codebook. This process should significantly reduce the computational requirements by only running VQ on a limited set of images. In addition, the storage/transmission requirements are reduced (at the possible expense of additional image artifacts).

As an alternative to the previous scheme we can omit the clustering step to determine exemplars. (If the image set does not possess several strongly-correlated clusters this approach will lead to a single exemplar image and will result in large reconstruction errors.) Instead, we observe that the luminance channel of an image should share certain local patterns which are repeated in a general image database. For example, within small image blocks, we expect the luminance channel to share edges, junctions, and other image features across images. As a result, we propose first converting the images to the $L^*a^*b^*$ color space [6]. VQ can then be applied to a subset of randomly selected images to obtain a set of cluster means defined in the luminance channel. The remaining images are compressed by representing the luminance information using the VQ codebook and by transmitting the original uncompressed color channels. This approach will only compress one of three color channels, but should achieve lower reconstruction errors for uncorrelated image sets.

As a final example, we consider the specific case of compressing video sequences. In this case we expect *a priori* that neighboring frames will be strongly correlated. As a result, we can obtain a "keyframe" codebook for a given frame using VQ. The following frames can then be compressed using this codebook. Periodically, we propose recomputing a new keyframe codebook. (This scheme is similar to approaches used by a variety of video codecs.) In general, we could also transmit sparse images containing the differences between VQ reconstructions and the original images – further reducing artifacts without significantly increasing storage/transmission requirements.

2 Feature Selection

Problem 3

Part 1: In this problem we will implement feature selection based on the mutual information (MI) between a feature and the class labels. Begin by writing `[h,hcond] = EntropyGaussian(x,y)` – a Matlab function that computes the entropy $H(X)$ of a 1D variable x and the conditional entropy $H(X|Y)$, assuming the class-conditionals $p(x|y)$ are Gaussian. Use this function to compute the mutual information between the 1D variable x and the class label y as `mi = MIGaussian(x,y)`.

First, recall that the entropy $H(X)$ of a random variable X is defined as

$$H(X) \triangleq -\int_X p(x)\log p(x)dx, \tag{1}$$

whereas the conditional entropy $H(X|Y)$ is defined as

$$\begin{aligned} H(X|Y) &\triangleq -\int_Y \int_X p(x,y)\log p(x|y)dxdy \\ &= -\int_Y p(y)\left\{\int_X p(x|y)\log p(x|y)dx\right\}dy. \end{aligned}$$

Note that, in this problem, Y is a discrete binary random variable representing the class labels and X is a particular feature x_j corresponding to a dimension of the input vector. As a result, the conditional entropy $H(X|Y)$ can be expressed as

$$H(x_j|y) = -\sum_{y\in\{0,1\}} P_y \int_{z\in\text{range}(x_j)} p(z|y)\log p(z|y)dz, \tag{2}$$

where P_0 and P_1 are the class priors. Also recall that the Gaussian probability density function $\mathcal{N}(x;\mu,\sigma^2)$ is given by

$$\mathcal{N}(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{3}$$

Substituting Equation 3 into Equations 1 and 2 gives the following closed-form expressions for the entropy $H(X)$ and the conditional entropy $H(X|Y)$ when the class-conditionals are Gaussian.

$$H(X) = \frac{1+\log(2\pi\sigma^2)}{2} \tag{4}$$

$$H(X|Y) = P_0\left(\frac{1+\log(2\pi{\sigma_0}^2)}{2}\right) + P_1\left(\frac{1+\log(2\pi{\sigma_1}^2)}{2}\right) \tag{5}$$

Note that $\{\sigma_0,\sigma_1\}$ represent the standard derivations of each class under a Gaussian model.

Equations 4 and 5 were implemented using `EntropyGaussian.m`. On lines 19-24 we extract the class labels and estimate the prior probabilities $\{P_0,P_1\}$. Lines 26-32 use ML estimators to determine the parameters of a Gaussian model for the complete set of observations, as well as the individual observations within each class. Finally, we note that line 35 implements Equation 4 and lines 38-39 implement Equation 5.

At this point we recall that the mutual information $I(X;Y)$, between random variables X and Y, is defined as

$$I(X;Y) \triangleq H(X) - H(X|Y). \tag{6}$$

As a result, we can directly evaluate the mutual information, assuming Gaussian class-conditions, using the outputs from `EntropyGaussian.m`. Specifically, we evaluate the entropies on line 19 of `MIGaussian.m` and compute the mutual information, using Equation 6, on line 20.

Part 2: Rather than assuming a parametric form for the class-conditions, now let's use a Gaussian kernel density estimator to model the distributions directly from the observations. Begin by writing `[h,hcond] = EntropyKernel(x,y,sigma)` and modify `kernelPDF.m` to implement Gaussian kernel density estimation. Use these functions to evaluate MI as `mi = MIKernel(x,y,sigma)`.

Recall that we can estimate the probability $p(x)$, corresponding to the likelihood of a given value x of the random variable X, using a kernel density estimator

$$\hat{p}(x) = \frac{1}{N}\sum_{i=1}^{N} K(x, x_i),$$

where $K(x, x_i)$ is a given kernel function (itself a valid probability density function) and where $\{x_i\}$ are observations drawn from $p(X)$. In this problem we will use a Gaussian kernel defined such that

$$\hat{p}(x) \triangleq \frac{1}{N\sigma\sqrt{2\pi}}\sum_{i=1}^{N} \exp\left(-\frac{(x-x_i)^2}{2\sigma^2}\right), \tag{7}$$

where σ is the user-selected standard deviation of the Gaussian kernel function.

Similar to Part 1, the entropy $H(X)$ and conditional entropy $H(X|Y)$ were evaluated in the Matlab function `EntropyKernel.m`, using the Gaussian kernel density estimator given by `kernelPDF.m`. (Note that we simply added the evaluation of the mean of the kernel estimates to complete line 21 of `kernelPDF.m`.) Once again, we implement Equation 6 using `MIKernel.m`.

Problem 4

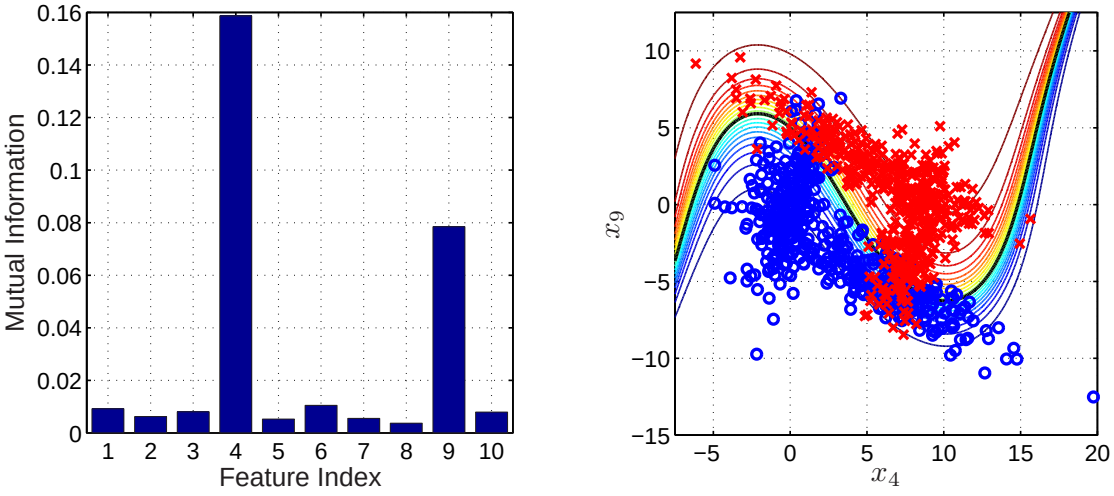
Part 1: In this problem we will examine the effect of feature selection on classifier performance. Specifically, we will evaluate the behavior of cubic polynomial logistic regression (LR) as we reduce the set of training features. Begin by training a LR model (with regularization parameter $\lambda = 1$) using all ten dimensions of the data in `gadgetTrain.mat`. Report the test errors on `gadgetTest.mat`.

Let's begin by briefly examining the Matlab solution script `prob4.m`. First, note that we set the logistic regression and kernel density estimator parameters on lines 15-17. Next, we parse the training and test data on lines 23-36. On lines 42-53 we use `logisticRegression.m` and `degexpandScale.m` to implement cubic polynomial logistic regression using all ten features. The resulting training and test errors are tabulated below (together with the results from Part 2).

Number of Features	Error on Training Data	Error on Test Data
10	9.8%	9.2%
2	10.7%	10.0%

Part 2: Now we will select two features (i.e., dimensions of the training data) using the maximum MI criterion. Using Gaussian kernel density estimation with $\sigma = 0.4$, plot the MI values for each of the ten features. Train a LR model using the top two features and report the resulting test error.

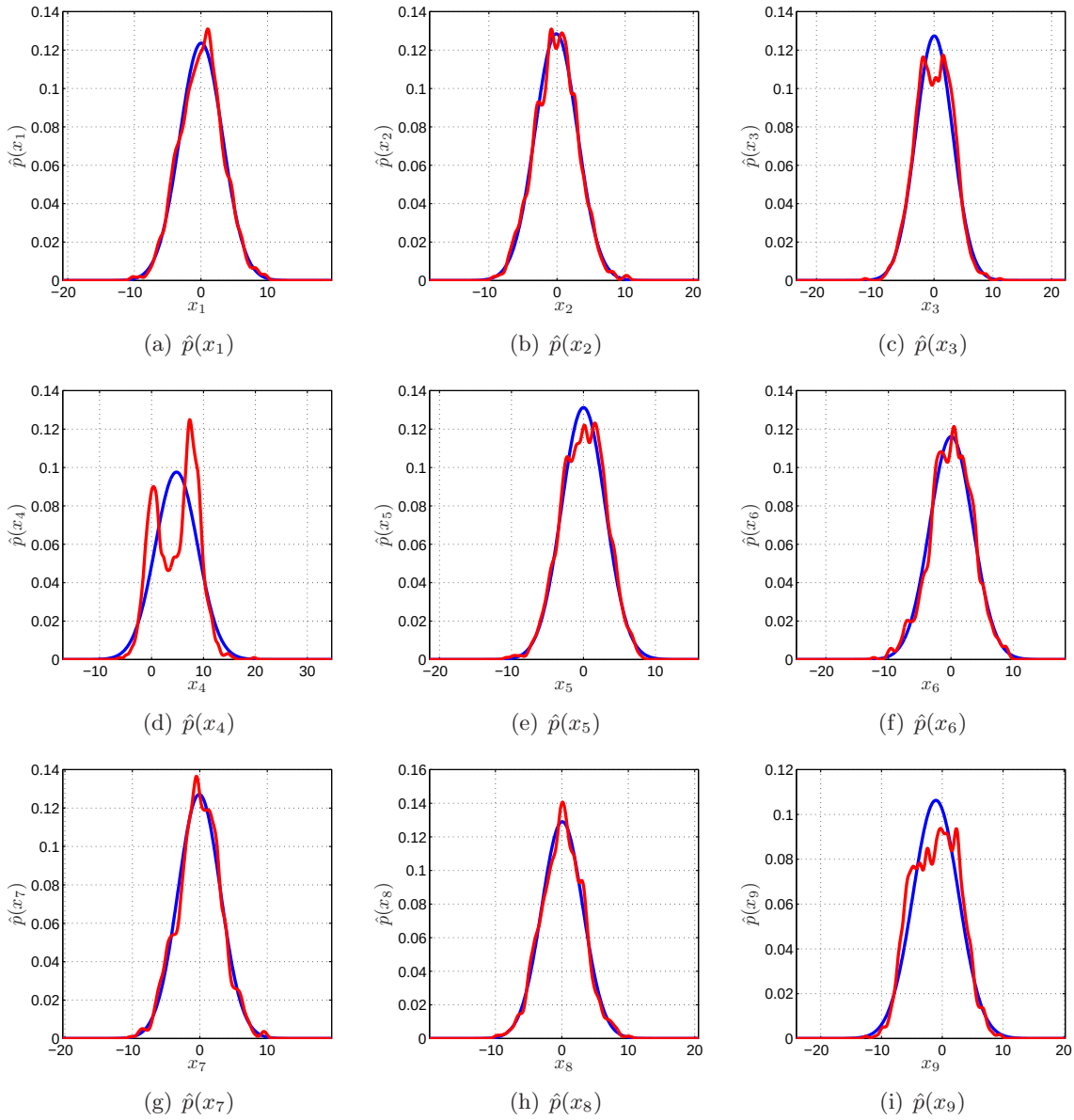
Using `MIKernel.m` from Problem 3, we evaluate the MI for each feature on lines 59-79 of `prob4.m`. From the plot in Figure 5(a), **we select features $\{\mathbf{x_4}, \mathbf{x_9}\}$** based on the maximum mutual information criterion. Training the logistic regression classifier using only the selected features results in the classification errors reported above. Note that the test error using all ten features is only slightly



(a) Mutual information of each feature

(b) Training samples and decision boundary

Figure 5: Feature selection using mutual information. (a) Estimated mutual information using a Gaussian kernel with $\sigma = 0.4$. (b) Training samples projected on the MI-selected features and the LR decision boundary, where $\circ$ denotes members of Class 0 and $\times$ indicates members of Class 1.



(a) $\hat{p}(x_1)$ (b) $\hat{p}(x_2)$ (c) $\hat{p}(x_3)$

(d) $\hat{p}(x_4)$ (e) $\hat{p}(x_5)$ (f) $\hat{p}(x_6)$

(g) $\hat{p}(x_7)$ (h) $\hat{p}(x_8)$ (i) $\hat{p}(x_9)$

Figure 6: Estimated probability density functions for the first nine dimensions of `gadgetTrain.mat`. The ML Gaussian fit is shown in blue and the Gaussian kernel density estimate is shown in red.

lower than that obtained using only the two selected features. This result can best be understood by examining the individual values of the mutual information tabulated in Figure 5(a). Recall from Equation 6 that the mutual information will be maximized, and equal to $H(X)$, when $H(X|Y) = 0$ (i.e., for situations in which knowledge of the class label completely determines the value of X). Similarly, the mutual information will be minimized, and equal to zero, when $H(X|Y) = H(X)$ (i.e., for situations in which the class labels Y and random variable X are independent). As a result, features with very small MI values will not provide a large amount of information regarding the class label. Since the MI values for all dimensions expect $\{x_4, x_9\}$ are close to zero, excluding them from the classifier does not have a significant effect on the test error.

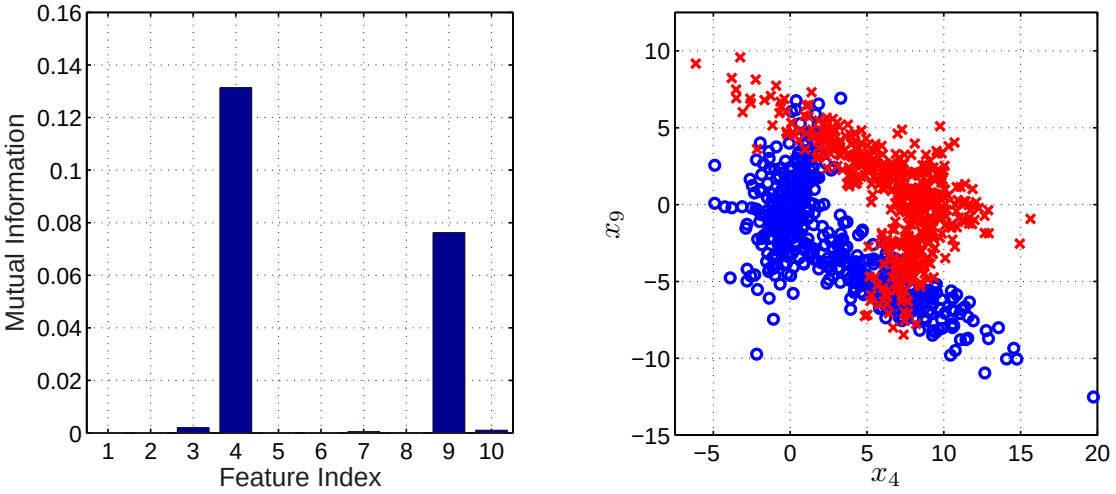
Problem 5

Part 1: While the previous problem utilized kernel density estimation, we can reduce the computational complexity by making certain parametric assumptions. Specifically, re-evaluate the MI of each feature under the assumption that the class-conditional density $p(x_j|y)$ is Gaussian for each feature and each class. Plot the resulting MI estimates and select the top two features.

Let's begin by reviewing the solution script `prob5.m`. We note that this script is nearly identical to `prob4.m`, except that on lines 60-78 we use `MIGaussian.m` to evaluate the MI for each feature. From the plot in Figure 7(a), again **we select features** $\{\mathbf{x_4}, \mathbf{x_9}\}$ based on the maximum mutual information criterion. As a result, the classification errors are identical to those in Problem 4.

Part 2: Plot the training data projected on the two features selected in the previous problem. State your conclusions from this plot, the plots of estimated MI, and the test errors. What can you say about the validity of the Gaussian assumption and its effect on the MI estimates?

The projected training data is shown in Figure 7(b). From this plot, it is apparent that the class-conditionals for $\{x_4, x_9\}$ are not well-modeled by a Gaussian distribution; instead, both classes appear to be composed of a mixture of two Gaussians. Comparing MI in Figures 5(a) and 7(a), it is apparent that the Gaussian model underestimates the values for $\{x_4, x_9\}$, in comparison to the kernel density estimator. This is confirmed by the individual density estimates in Figure 6. Note that only the kernel density estimator is capable of modeling the bimodal distributions of $\{x_4, x_9\}$. In conclusion, the Gaussian class-conditional assumption is more computationally efficient when it holds, however the kernel density estimator appears more robust (assuming an appropriate width σ is known and sufficient samples are available to approximate the underlying distribution).



(a) Mutual information of each feature

(b) Training samples projected on MI features

Figure 7: Feature selection using mutual information. (a) Estimated mutual information assuming Gaussian class-conditionals. (b) Training samples projected on features with the highest estimated mutual information, where ∘ denotes members of Class 0 and × indicates members of Class 1.

3 Dimensionality Reduction

Problem 6

Part 1: In this problem we will apply principal component analysis (PCA) to a database of face images [3] in a manner similar to Turk and Pentland's "Eigenfaces" [5]. Begin by using the built-in Matlab function `princomp` to apply PCA to the training images in `faceData.mat` as follows.

```
[phi,xlowd,lambda] = princomp(Xtrain,'econ')
```

Plot the mean face and the first five principal components. Also plot the the eigenvalues of the covariance matrix (i.e., `lambda`) and explain how to use them to select a suitable dimension m.

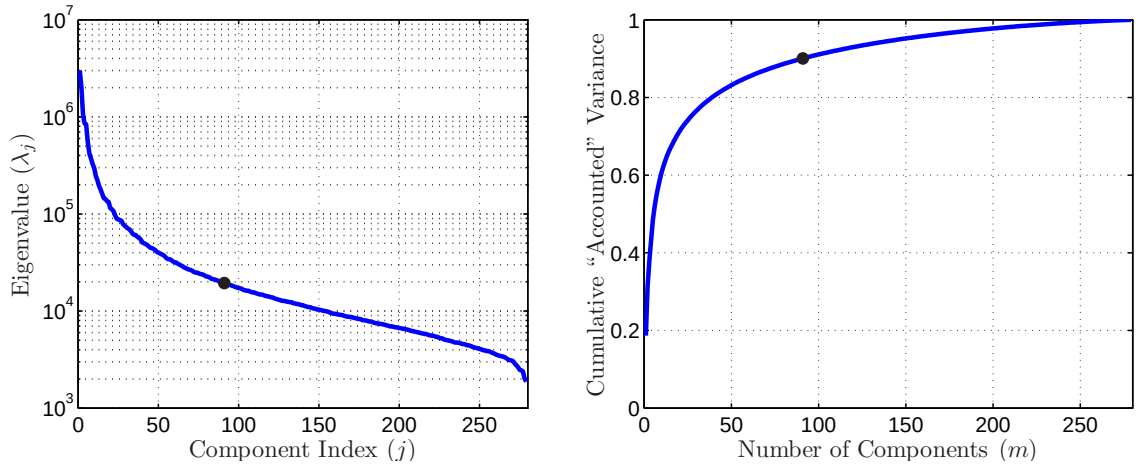
Let's begin by reviewing the basic properties of PCA. Given a particular representation of the data in terms of a vector $\mathbf{x} = [x_1, \ldots, x_d]$ in $\mathbb{R}^d$, we recall that we can define its projection on a linear subspace $\mathbf{x}'$ as

$$\mathbf{x}' = \mathbf{\Phi}^T(\mathbf{x} - \bar{\mathbf{x}}), \tag{8}$$

where $\mathbf{\Phi}$ is a $d \times m$ matrix (with $m < d$), whose columns form an orthonormal basis for a linear subspace of $\mathbb{R}^d$ with dimension m. Also note that we have subtracted $\bar{\mathbf{x}} = \sum_{i=1}^{N} \mathbf{x}_i$ to ensure that the samples have zero mean. Recall that, for the particular case of PCA, the columns of $\mathbf{\Phi}$ correspond to the first m unit-length eigenvectors (sorted by decreasing eigenvalue) of the $d \times d$ covariance matrix given by

$$\frac{1}{N} \sum_{i=1}^{N} (\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T. \tag{9}$$

We refer to these m eigenvectors as the *principal components* of the data. In general, it can be shown that the first principal component defines the direction of highest variance in the data. Similarly, the second principal component corresponds to the direction of highest variance among



(a) The eigenspectrum of the training covariance matrix (b) Variance retained in principal subspace

Figure 8: Selecting the number of components m. (a) Plot of the eigenvalues of the training covariance matrix. (b) Plot of the variance retained in the principal subspace defined by Equation 11.

all directions orthogonal to the first principal component (and so on for the remaining components). Recall that we can reconstruct a given sample from its lower-dimensional PCA projection as

$$\tilde{\mathbf{x}} = \bar{\mathbf{x}} + \mathbf{\Phi}\mathbf{x}'. \tag{10}$$

Now that we understand the basic properties of PCA, we turn our attention to its application to the training images in `faceData.mat` using `prob6.m`. First, on lines 16 and 17 we select the number of components m and five random faces for reconstruction. We load and pre-process the training images on lines 23-50. Next, on line 57 we apply `princomp` to determine the principal components, their eigenvalues, and the low-dimensional projections of the training images. On lines 60-62 we reconstruct the training images using Equation 10. Finally, on lines 76-142 we plot and analyze the PCA results. The mean training image is evaluated on lines 43-50 and is shown in Figure 10. Similarly, the first five eigenvalues are plotted on lines 76-89 and are tabulated in Figure 11.

At this point we must determine how many principal components m are required to sufficiently model the variation in the training database. As described in class on 11/15/06, the distribution of eigenvalues is particularly useful for determining m – since each eigenvalue λ_j gives the amount of variance in the data captured by the j^{th} principal component. From the plot of the sorted eigenvalues in Figure 8(a), it is apparent that there is a characteristic "elbow" around $m = 50$ components. Note, however, that a more principled method for selecting m is given by plotting the variance $V(m)$ retained in the principal subspace, as defined by the following expression.

$$V(m) = \frac{\sum_{i=1}^{m} \lambda_i}{\sum_{j=1}^{d} \lambda_j} \tag{11}$$

Using this equation, we obtain the plot in Figure 8(b). In conclusion, we select the 90% retained-variance level, which requires $\mathbf{m = 91}$ components for the training data in `faceData.mat`.

Part 2: Using the selected number of components m, plot five training faces along with their reconstructions. Explain any artifacts you see and how the reconstructions are affected by m.

As shown in Figure 12, five random faces were selected from the training data in Figure 9. Their reconstructions, from the lower-dimensional PCA representation using $m = 91$ components, are also shown. Note that several artifacts are apparent. First, we notice that the reconstructions do not retain high spatial frequencies (i.e., fine details). This is expected since these features will be specific to each image and, as a result, will only be captured by components with small eigenvalues which are not likely to be included in the limited principal component set. In addition, we notice certain patterns are present in the reconstructions which are not in the original images. For example, consider the reconstruction shown in Figure 12(i); the outline of a pair of glasses is clearly visible, yet this feature is not present in the original image. This is explained by the fact that a particular component used in the reconstruction has this feature and, since the reconstruction is formed as a linear combination of components, this feature cannot be completely removed.

In general, we find that increasing the number of components m reduces the apparent artifacts. In the limit that $m = 279$ (i.e., the number of independent dimensions in the training covariance matrix), we find the the PCA-based reconstruction is equal to the original face to within small numerical errors. This demonstrates the central tradeoff between dimensionality m and reconstruction accuracy in PCA compression.

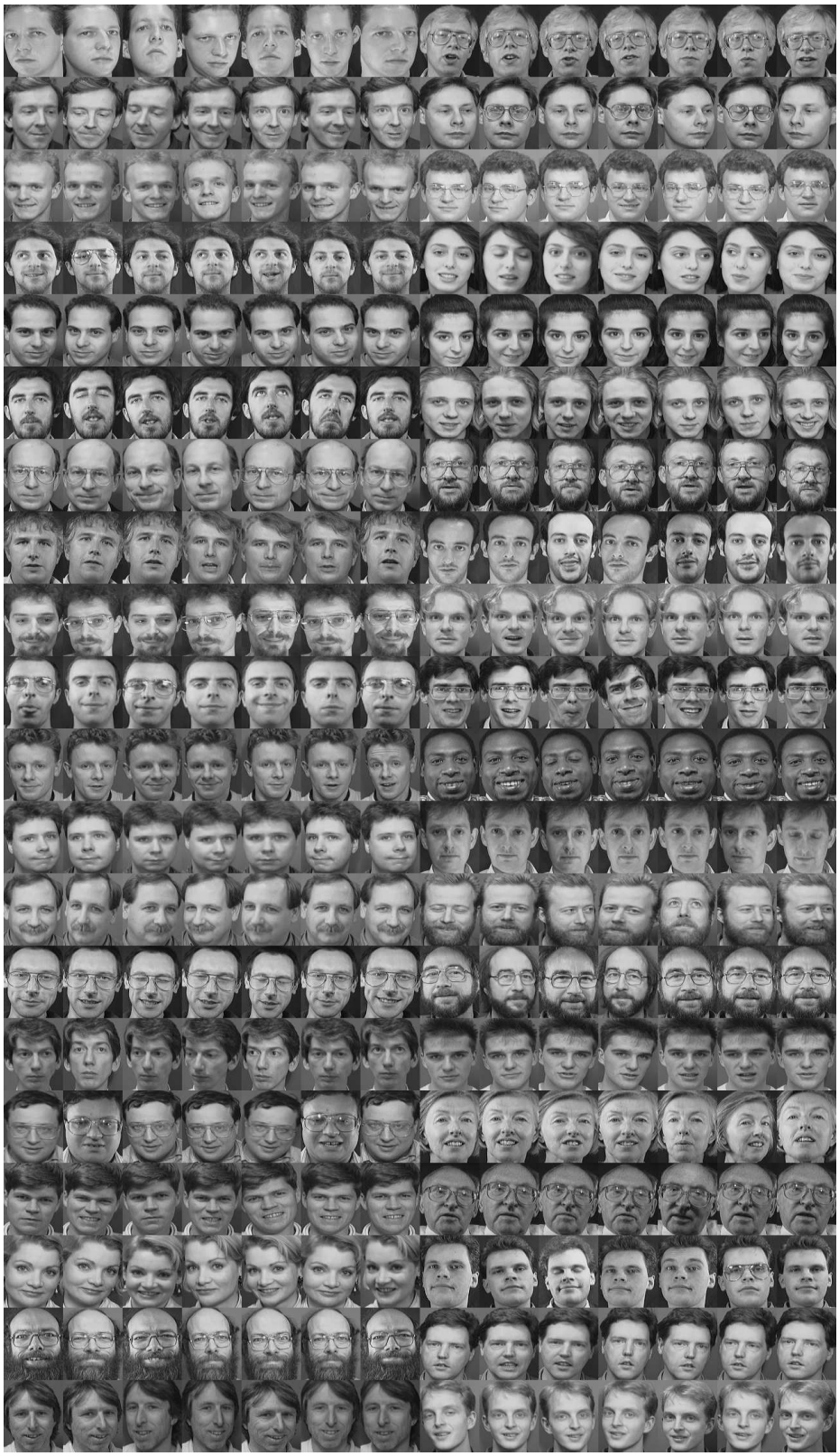


Figure 9: The AT&T face database composed of 40 individuals, each with 7 different pose and expression variations. Each sample is given by a grayscale image of dimension 112×92 pixels [3].

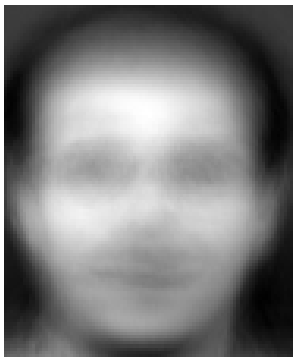
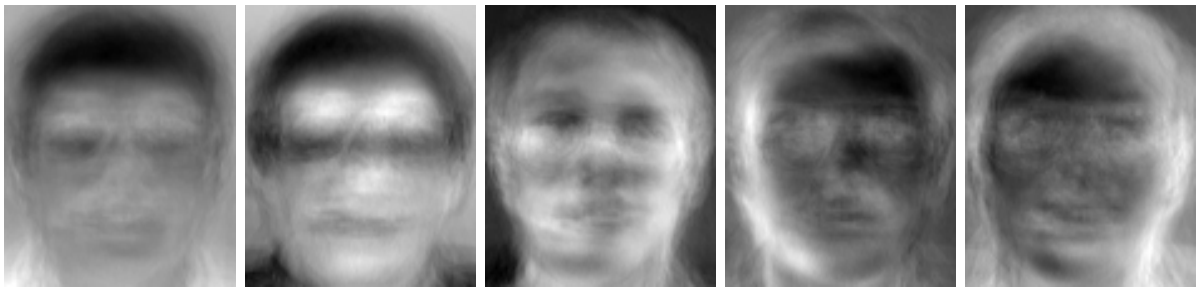
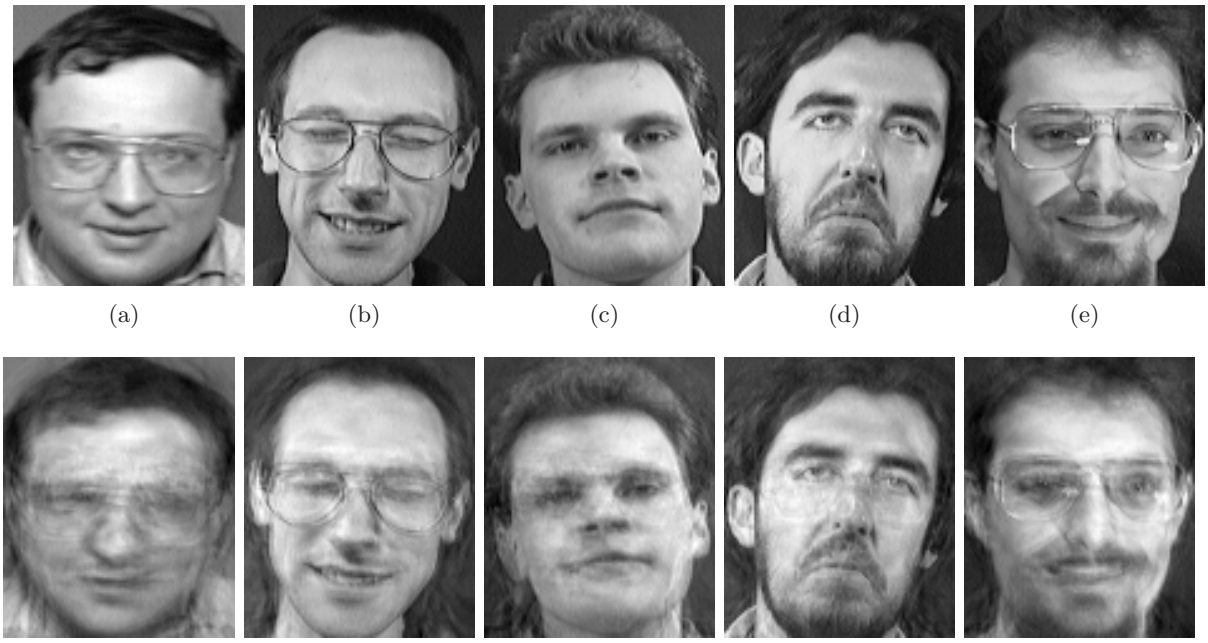


Figure 10: The mean face averaged over the training samples in Figure 9.



(a) (b) (c) (d) (e)

Figure 11: The first five principal components of the training samples in Figure 9.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 12: Modeling faces using PCA. (a)-(e) Five random faces from Figure 9. (f)-(j) The reconstruction of each face, from its low-dimensional projection, using $m = 91$ principal components.

Problem 7

Part 1: In this problem we will apply PCA to the task of face recognition using the "Eigenfaces" approach of Turk and Pentland [5]. Compute the low-dimensional PCA representation of the test data in `faceData.mat` using the dimensionality m selected in Problem 6. Apply the k-nearest neighbor classifier (in the low-dimensional subspace) to predict the identities of the test subjects. Set k using leave-one-out cross validation for $k = \{1, 3, 7, 15, 30, 50\}$. Report the test error and the "chance level" performance. Comment on the usefulness of this approach for face recognition.

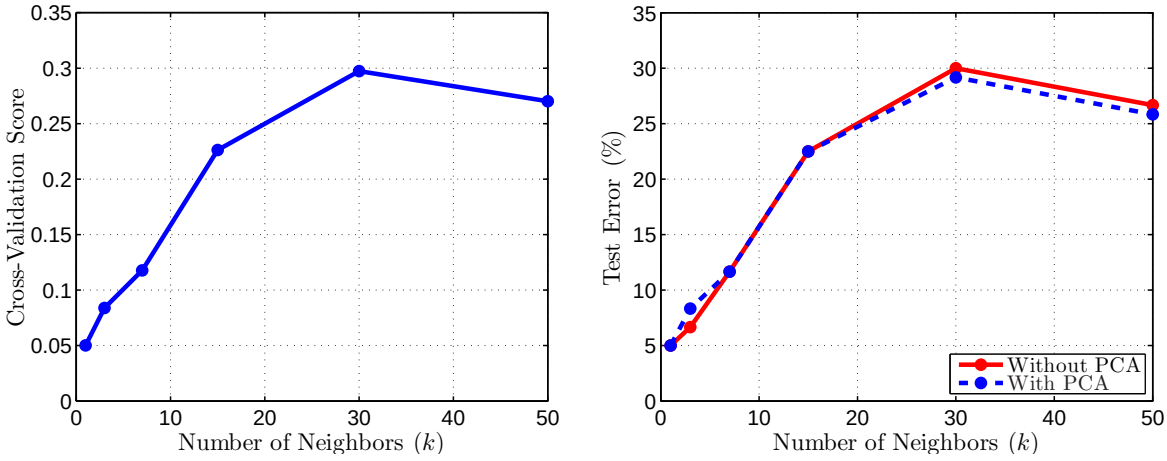
Let's begin by briefly reviewing the k-nearest neighbor (k-NN) classifier, as presented in class on 10/23/06. Recall that if we are provided a labeled set of N training samples $\{\mathbf{x}_1, y_1, \ldots, \mathbf{x}_N, y_N\}$ in $\mathbb{R}^d$, then the k-nearest neighbor classifier will assign a label to a query point $\mathbf{x}_0$ by majority voting among its k-nearest neighbors (typically measured using the Euclidean distance in $\mathbb{R}^d$). To complete our solution to this problem, the k-NN classifier was implemented as `knnclassifier.m`. Note that, as recommended, we used `findnn.m` and `lpnorm.m` within the k-NN classifier to allow flexible nearest-neighbor selection under a general L_p-norm.

The general solution script is provided as `prob7.m`. First, on lines 17 and 18 we specify the dimensionality m and the number of neighbors k for cross-validation testing. On lines 23-50 we load and pre-process the training and test data in `faceData.mat`. Next, on lines 56-70 we compute the low-dimensional representation of the training and test data. (Note that the mean of the *training* images is subtracted from the test data to ensure consistent projections.) On lines 76-92 we apply leave-one-out cross-validation to select the best value of k to use with the k-NN classifier. The test errors (both with and without PCA-based dimensionality reduction) are tabulated below.

k	Cross-Validation Score (with PCA)	Test Error (with PCA)	Test Error (without PCA)
1	0.0502	5.00%	5.00%
3	0.0839	8.33%	6.67%
7	0.1177	11.67%	11.67%
15	0.2262	22.50%	22.50%
30	0.2973	29.17%	30.00%
50	0.2701	25.83%	26.67%

From Figure 13(a) and the tabulated cross-validation scores, we select **$\mathbf{k = 1}$ neighbor** (i.e., the nearest-neighbor classifier). Using the nearest-neighbor classifier and the low-dimensional PCA-based representations, we report a test error of only 5%. As shown in Figure 14, the selected classifier only misclassifies 6 of 120 test samples – a surprisingly strong result! For example, a random classification (into one of the 40 subject classes) would be expected to correctly classify $(1/40) \cdot 120 = 3$ faces on average – resulting in a **"chance level" test error of 97.5%**.

In conclusion, we find that the "Eigenfaces" approach achieves low test errors for the AT&T database. However, these results should be examined more closely before concluding this is a valid approach for the general face recognition task. If we compare the training and test images (in Figures 9 and 14, respectively) we observe strong correlations. In fact, many subject images appear to be drawn from a video sequence and, as a result, the images are very similar. The selection of the nearest-neighbor classifier confirms this observation. If the low-dimensional representation was truly capturing the inherent variation within a class, we'd expect at least $k = \{3, 7\}$ neighbors would achieve better results (since there are 7 pose and expression variations for each subject in the training database). As a result, we conclude that the "Eigenfaces" approach achieves low test errors only when the training database sufficiently models all possible variations of each test



(a) Cross-validation scores using PCA and k-NN (b) Test errors obtained using the k-NN classifier

Figure 13: Face recognition performance using "Eigenfaces". (a) Cross-validation scores using the low-dimensional PCA representations and the k-nearest neighbor classifier. (b) Comparison of test errors obtained using the k-NN classifier with and without the PCA dimensionality reduction.

subject. This is not a desirable property for the general face recognition problem, since lighting, pose, and expression variations will necessitate a very large training database. In addition, the proposed method is not tolerant to changes in scale or surface properties (i.e., beards, masks, etc.).

Part 2: Compare the results you obtained using PCA to those obtained by applying the k-nearest neighbor classifier directly to the original 10,304-dimensional representations. What advantages, if any, do you see in using the low-dimensional representation instead of the original images?

The k-nearest neighbor classifier was applied directly to the original 10,304-dimensional samples on lines 98-103 of `prob7.m`. The resulting test errors are tabulated with the results reported in Part 1 and are also plotted in Figure 13(b). Overall, we find that the test error is nearly identical to the results obtained using the $m = 91$ principal component representation. Recall from Problem 6 that $m = 91$ was selected to capture 90% of the variation in the training data. As a result, the negligible change in test error is expected since the remaining 10% of variance does not significantly enhance the separation of training samples used by the k-NN classifier.

In conclusion, we find that principal component analysis is a very effective compression scheme for recognition tasks. By design, PCA compresses the training data in a manner which retains as much variance as possible in the projected subspace. In general, we conclude that PCA will reduce the computational complexity of similar classification tasks. As discussed in class on 11/15/06, however, the the principal components do not always select the most discriminative dimensions. As a result, projection onto a linear subspace may make discrimination impossible – demonstrating that PCA cannot be applied blindly to all classification problems.

Figure 14: 120 test images from the AT&T face database [3]. Samples shaded in red were misclassified using the nearest-neighbor classifier with the low-dimensional PCA-based representations.

References

[1] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1996.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[3] AT&T Laboratories Cambridge. Database of faces. `http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`.

[4] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (Second Edition)*. Wiley-Interscience, 2000.

[5] Matthew A. Turk and Alex P. Pentland. Face recognition using eigenfaces. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1991.

[6] Günther Wyszecki and W. S. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulae*. John Wiley & Sons, 1982.