

CS195-5: Introduction to Machine Learning  
Fall 2006

Problem Set #5 : Rambutan

Out: November 11, 2006

**Due: November 22, 11am EDT**

## Instructions

In order to identify problem sets to the automated submission system we will assign code names to each problem set. This one will be called “rambutan”.

**How and what to submit?** Please submit your solutions using the automated system in the following way.

1. Create a directory (say, `ps5`) in which you will work on your solutions. All files mentioned below are assumed to reside in that directory.
2. Typeset the written parts of your solution in  $\text{\LaTeX}$ . The final results should be a document in PDF (preferable) or in PostScript. please name this document `rambutansolution.pdf` or `rambutansolution.ps`, according to the format. **Important: please start the solution of every problem with `\newpage`.**
3. Create Matlab code (extension `.m`) and data (`.mat`) files needed.
4. Create a file named `README`, and include in it a short description of all Matlab files you are submitting.
5. **From the directory `ps5`, run**  
`/course/cs195-5/bin/cs195-5handin rambutan`

You can resubmit your work as many times as needed, and each subsequent version will override the previous one.

**Late submissions: there will be a penalty of 25 points for any solution submitted past the deadline until 11am on Monday, November 27. No submissions will be accepted past then.**

**What is the required level of detail?** When asked to derive something, please clearly state the assumptions, if any, and strive for balance: justify any non-obvious steps, but try to avoid superfluous explanations. When asked to plot something, please include the figure as well as the code used to plot it; if multiple entities appear on a plot, make sure that they are clearly distinguishable (by color or style of lines and markers). When asked to provide a brief explanation or description, try to make your answers concise, but do not omit anything you believe is important.

## 1 Clustering and Vector Quantization

In this section we explore the application of Vector Quantization (VQ) to image compression. We will break the input image into a set of non-overlapping blocks,  $m \times m$  pixels; each block is a vector in the  $3m^2$ -dimensional space (since there are three color channels, representing for each pixel the red, green and blue values). We will then run  $k$ -means clustering in this space. Once we have obtained the cluster centers, we can replace each block in the original image with the cluster it has been assigned to. We will call this the “reconstructed” image.

### Problem 1 [15 points]

We have provided a Matlab function `learnVQ.m` that relies on `kmeans.m` to run the  $k$ -means algorithm. Write `kmeans.m` and run VQ for the image `imagePS5.png`, using  $k=256$  and block size  $m=3$ . To store the reconstructed image, we need to store the cluster means and the index of the assigned cluster for each block. Assuming no further compression, compare the number of bits required to store the reconstructed image versus the original (that requires three bytes per pixel); what is the compression factor?

Display the two images: the original and the one reconstructed by VQ. Also display the cluster means, using the provided `showBlockMeans.m` function. What can you say about the results of this compression? If there are artifacts you find undesirable, what change in parameters of the VQ (block size and/or  $k$ ) would in your opinion address those?

**End of problem 1**

*Advice: We recommend that in this problem as well as in the problems in Section 3, you use the functions `findnn.m` and `lpnorm.m` provided with PS Grape.*

Note that in this case we should really use a  $k$ -medoids algorithm, to ensure that the values in the cluster means represent valid pixels. In particular, each value should be integer between 0 and 255. To make matters simpler and save some computation, use a “hack”: after the  $k$ -means has converged, round and “clip” the cluster means to force them to represent valid image blocks. Note that you won’t be able to display the color image using `image` or `imagesc` without such rounding, and converting the values to type `uint8`.

**Problem 2** [10 points]

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

**End of problem 2**

## 2 Feature Selection

Inspector Gadget needs your help! Every morning for the past 6 years he has been collecting data from ten of his various on-board sensors to try to determine the factors that will make it rain that day. This is so he can make a decision whether to take his trusty umbrella attachment from home. He hopes to perform two-class classification on the data so he can predict, as accurately as possible, whether it is going to rain. Unfortunately Inspector Gadget only has 50K of on-board memory and an 8MHz processor under his hat so he needs you to perform feature selection on the data so he can use his logistic regression toolbox on a smaller set of features and still get good results. The 10-dimensional data set he has collected over a set of 1000 days, along with the class labels for whether it rained or not ( $y = 1$  if it rained,  $y = 0$  otherwise) can be found in `gadgetTrain.dat`. He is also providing you with `gadgetTest.dat` from another set of 1000 days for you to test the classification on. Note that based on Gadget’s experience, the prior probability of a rainy day is exactly one half.

To help Gadget, you will need to perform *feature selection*. In this problem we will explore the feature selection method based on *mutual information* (MI) between a feature and the class label. MI between random variables  $X$  and  $Y$  is denoted  $I(X; Y)$  and is defined as

$$I(X; Y) \triangleq H(X) - H(X|Y), \quad (1)$$

where  $H(X)$  is the entropy of the random variable  $X$

$$H(X) = - \int p(X) \log p(X) dX, \quad (2)$$

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

$$H(X|Y) \triangleq \int p(X, Y) \log p(X|Y) dX dY = \quad (3)$$

$$= \int p(Y) \left\{ \int p(X|Y) \log p(X|Y) dX \right\} dY. \quad (4)$$

Intuitively, this measures the amount of information about  $X$  contained in  $Y$ . Recall that the entropy is a measure of uncertainty; thus the MI measure how much of the inherent uncertainty regarding  $X$  is reduced by knowledge of  $Y$ .

Note that in our case,  $Y$  is the discrete (binary) variable representing the class, and  $X$  is a particular feature (dimension of the input vector)  $x_j$ . The conditional entropy then becomes

$$H(x_j | y) = - \sum_{y \in \{0,1\}} p(y) \int_{z \in \text{Range}(x_j)} p(z | y) \log p(z | y) dz \quad (5)$$

For some distributions one can compute the entropy of a random variable in closed form. In particular, for a Gaussian random variable  $X \sim \mathcal{N}(X; \mu, \sigma^2)$  the entropy is

$$H(X) = \frac{1}{2} (1 + \log 2\pi\sigma^2). \quad (6)$$

In a more general case, even for a mixture of Gaussians, we can no longer compute entropy in closed form. If the pdf  $p(X)$  can be calculated in closed form, e.g., if we make a certain assumptions about the parametric form of the pdf and estimate the parameters, then we can still evaluate  $H(X)$  fairly easily, by numerically approximating the integral (2).

If no parametric assumptions are made regarding the form of  $p(X)$  but we have a set of observation drawn from it, we can estimate the density at any given value of  $X$  using the kernel density estimate as described in class:

$$\hat{p}(x) = \frac{1}{N} \sum_{i=1}^N K(x, x_i), \quad (7)$$

where  $K(x, x_i)$  is the kernel function which itself is a valid pdf. In this problem set we will use the Gaussian kernel, i.e.

$$\hat{p}(x) = \frac{1}{N\sqrt{2\pi}\sigma} \sum_{i=1}^N \exp\left(-\frac{1}{2\sigma^2}(x - x_i)^2\right). \quad (8)$$

As shown in class, the parameter  $\sigma$  determines the *width* of the kernel. We have provided a Matlab function `kernelPDF.m` (with a part missing) which computes this estimate.

**Problem 3 [15 points]**

Fill in the missing line in `kernelPDF.m`, and write the following Matlab functions:

`[h,hcond]=EntropyGaussian(x,y)` that computes the entropy  $H(X)$  of a 1D  $x$  as well as the conditional entropy  $H(X|Y)$ , assuming that class-conditionals  $p(x|y)$  are Gaussian. The function should take care of estimating the parameters of the Gaussian for each class (use ML estimates).

`mi=MIgaussian(x,y)` which computes MI of 1D  $x$  with the class label  $y$ , assuming that class conditional  $p(x|y)$  is Gaussian. Note: this should be an extremely simple function once you write `EntropyGaussian`.

`[h,hcond]=EntropyKernel(x,y,sigma)` that computes the entropy  $H(X)$  of a 1D  $x$  as well as the conditional entropy  $H(X|Y)$ , using kernel density estimator (with Gaussian kernel) for the class-conditionals. The kernel width is given by `sigma`.

`mi=MIKernel(x,y,sigma)` which computes MI of 1D  $x$  with the class label  $y$ , using kernel density estimator. Again, this should be very simple given `EntropyKernel`.

Try to make your code as efficient as possible (as usual, this means in particular avoiding loops).

**End of problem 3**

*Advice: To calculate the entropy, you will need to estimate density over a regularly spaced range of values and approximate the integral in (2) Think carefully about the range, and also don't forget the  $dx$  in the integral.*

Now we can start experimenting with feature selection for classification. Throughout this section we will use *cubic* polynomial LR, i.e. applied to features such as generated by `degexpandScale(X,3,ones(1,3*d+1))` where  $d$  is the dimension of the input data. Note that we will compute MI of the original input dimensions, not these polynomial features. Code for regularized logistic regression is provided in `logisticRegression.m`; use regularization parameter  $\lambda = 1$  in all experiments.

**Problem 4 [15 points]**

First, using all 10 dimensions, train the logistic regression model on data in `gadgetTrain.mat`, test on `gadgetTest.mat`, and report the test error.

Calculate the MI for each feature, using Gaussian kernel density estimate with  $\sigma = 0.4$ . Obviously, you should only use training data. Plot the MI values for each of the ten features, and specify which two features you have selected. Train logistic regression again, now using just the selected features.

Now train and test LR using only the two selected features, and report the test error. Briefly explain the results, in particular, what is causing the change in the test error (or lack thereof).

**End of problem 4**

**Problem 5 [15 points]**

The previous problem asked you to estimate the probabilities using kernel density estimation. We can try to make the task less computationally demanding by making certain parametric assumptions. Here, we will assume that the marginal class-conditional density  $p(x_j | y)$  is Gaussian for each feature and each class.

Using the code you have written, evaluate MI for each feature under this Gaussian assumption. Again, plot the estimated MI and select the two features with highest estimated MI. If you selected a different set of two features than in the previous problem, report the test error you get with these features.

Plot the training data (with different markers for the two classes) projected on the two features selected in the previous problem. State your conclusions from this plot, the plots of estimated MI (in this and the previous problems), and the test errors. In particular, what can you say about the validity of the Gaussian assumptions and the effect it has on the MI estimate? What are pros and cons of using the two density models (Gaussian and non-parametric)

in this case?

**End of problem 5**

*Advice: Careful: the assumption of Gaussian class-conditionals does not mean that the compound  $p(x_j)$  is Gaussian! Under the stated assumptions it is, in fact, a mixture of two Gaussians. Of course, since the class labels are provided for the training data, you can find the ML estimate for the mixture*

### 3 Dimensionality reduction

When we are given a particular representation of the data in terms of a vector  $\mathbf{x} = [x_1, \dots, x_d]$  in  $\mathbb{R}^d$ , feature selection can be seen as dimensionality reduction by *projection*

$$\mathbf{x}' = \mathbf{\Phi}^T \mathbf{x}. \quad (9)$$

The projection operator  $\mathbf{\Phi}$  is in this case limited to a particular form, namely axes-parallel projection. For instance, selecting second and fourth out of five features can be expressed as the following projection:

$$\mathbf{x}' = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{x}.$$

We can generalize this to a more general form of the linear operator  $\mathbf{\Phi}$ . In particular, suppose that the columns of the  $d \times k$  matrix  $\mathbf{\Phi}$ , with  $k < d$ , form an orthonormal basis of a linear subspace of  $\mathbb{R}^d$ . Then, each new feature,<sup>1</sup> that is, each coordinate of the  $k$ -dimensional  $\mathbf{x}'$  in (9) is a linear combination of the original features:

$$x'_j = \sum_{l=1}^d \Phi_{jl} x_l = \phi_j^T \mathbf{x},$$

where  $\phi_j$  is the  $j$ -th column of  $\mathbf{\Phi}$ . One widely used principled way of constructing the projection  $\mathbf{\Phi}$  is by means of principal component analysis (PCA), also known as the *Karhunen-Loeve transform* or the *Hotelling*

---

<sup>1</sup>Since none of these features is one of the original features, this is sometimes referred to as “feature generation”.

*transform*. In PCA, the  $m$  basis vectors (columns of  $\Phi$ ) are the unit-length eigenvectors of the  $d \times d$  data covariance matrix

$$\sum_{i=1}^N (\mathbf{x}_i - \mu_{\mathbf{x}})(\mathbf{x}_i - \mu_{\mathbf{x}})^T,$$

corresponding to the largest eigenvalues.  $\mu_{\mathbf{x}}$  is the mean of the data  $\frac{1}{N} \sum_{i=1}^N \mathbf{x}_i$ . These vectors are called the *principal components* of the data, and it can be shown that the first principal component  $\phi_1$  is the direction of highest variance in the observed data. The second principal component  $\phi_2$  is the direction of highest variance among all directions *orthogonal* to  $\phi_1$ ; the third is the direction of highest variance among all directions orthogonal to both  $\phi_1$  and  $\phi_2$ , etc.

We can reconstruct the original data from the PCA representation in the following way: since the feature  $x'_j$  is the projection of  $\mathbf{x}$  on a basis vector  $\phi_1$ , we can calculate

$$\tilde{\mathbf{x}} = \mu_{\mathbf{x}} + \Phi(\mathbf{x}' - \mu_{\mathbf{x}})^T. \quad (10)$$

In general, this is not going to reconstruct  $\tilde{x}$  precisely, since we have lost some information by reducing the dimension from  $d$  to  $k$ . Specifically, there will generally be a non-zero *Euclidean residual*  $\epsilon = \|\mathbf{x} - \tilde{\mathbf{x}}\|$ . It can be shown however that the basis obtained with PCA produces reconstruction with of the training data with the lowest residuals, i.e. PCA is the optimal dimensionality reduction in terms of the Euclidean residual.

In this problem we will apply PCA to a set of face images, originally collected at AT&T.<sup>2</sup> These are graylevel images  $112 \times 92$  pixels, of 40 subjects in a variety of poses and expressions. The data in Matlab format can be found in `faceData.mat`. It has been partitioned to training and test set, in `Xtrain`, `Xtest` (faces as column vectors) and `Ytrain`, `Ytest` (subject identities, from 1 to 40). To visualize a particular face, use a command like

```
imagesc(reshape(Xtrain(:,1),112,92));axis image;colormap gray
```

In the first part of the problem, we will restrict our attention to the training set, and will ignore the subject identities. To apply PCA we will use Matlab's builtin routine `princomp`. The terminology in the help printout is a bit different from ours, so to clarify:

<sup>2</sup><http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html>

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

will produce in the columns of `phi` the basis of the principal subspace, and in the rows of `xlowd` the low-dimensional representations  $\mathbf{x}'$ . `lambda(j)` gives the amount of variance  $\lambda_j$  in the data accounted for by the  $j$ -th principal component. `princomp` also subtracts the mean  $\mu_{\mathbf{x}}$  from the data.

Note that with  $N < d$  we can not possibly get more than  $N$  meaningful linearly independent components in  $\Phi$ . In fact, since we subtract the data mean from each observation, there are at most  $N - 1$  independent dimensions. The directive `'econ'` indicates this fact and causes the eigen-decomposition to only extract the  $N - 1$  largest eigenvalues; the dimension of `phi` will therefore be  $d \times N - 1$ —in this case,  $N - 1 = 279$ , since there are 280 training examples. This significantly speeds up the computation.

### **Problem 6 [15 points]**

Apply PCA to the training face data in `Xtrain`. Plot the mean face (average of the training data) and the first five principal components. Also, plot the variances `lambda`, and explain how one could use the values of `lambda` to select a suitable dimension  $m$ ?

In addition, plot five randomly selected faces from the training data along with their reconstruction from the low-dimensional representation with  $m$  you selected. Explain any artifacts you see; how are the reconstructions affected by the value of  $m$ ?

**End of problem 6**

*Advice: Things to look at are the shape of the graph as well as the actual values of  $\lambda_j$ . Think, and check empirically, how they relate to the variance of the original data `Xtrain` and to the variance in the low-dimensional representation `xlowd`, and how the variance is in turn related to the reconstruction residuals.*

*As a sanity check, try to reconstruct a face in the training data using all 279 PCA dimensions; the reconstruction should be equal to the original face, up to very small numerical errors.*

PCA as a method of dimensionality reduction is unsupervised. However we can use the results to produce feature representations used in classification. In the next problem we will experiment with this idea applied to face recognition; this method is known as “Eigenfaces” (in reference to the eigen-decomposition which is a part of the PCA computation).

**Problem 7 [15 points]**

Compute the low-dimensional representation of the test data in `Xtest` using the PCA results from the previous problem and the dimension you have selected. Use  $k$ -nearest neighbor classifier to predict the identities of the subjects in the test set, when the training and test data are both represented in the low-dimensional space. To set  $k$ , use leave-one-out cross-validation for the following values of  $k$ : 1,3,7,15,30,50. Report the results (i.e. the test error obtained using the selected  $k$ ); what is the “chance level” performance here, i.e. the error obtained by random guessing? What is your opinion about the usefulness of this method for face recognition?

Compare the results you obtained with PCA to those of  $k$ -nearest neighbor classification using the original, 10304-dimensional representation. What advantages, if any, do you see in using low-dimensional representation instead of the original images?

**End of problem 7**

*Advice: Think carefully about how to project the test data onto the principal subspace. In particular, what is the mean vector you need to add/subtract in (10)?*

*Leave-one-out CV with the  $k$ -NN classifier can be expensive if done naively. You can make it very efficient by running the NN search once, taking care not to include  $x_i$  as its own neighbor. This can be done using the `'noself'` argument in `findnn.m`; see documentation for details.*