Practical exercises (1)

Contents

PCA – practice (1)

p CMU, 2011f, EXing, HW5, pr. 1 (eigenfaces (with eig) + 1NN)

1 Principal Component Analysis: Eigenfaces [25 points, Bin]

PCA is applied to recognize faces by Matthew Turk and Alex Pentland in their "Eigenface" approach, which is considered as the first successful example of face recognition. In this question, you will implement your own face recognition system.

Please print out your codes and attach them at the end of your answer sheet for this question.

1.1 ORL Face Data

The face data set used in this question is the ORL database of faces[1]. There are 4 files in HW4_data.zip:

- X_train.csv: each of the 360 lines contains data for a 112×92 face image. If you are using MATLAB, you can load the images using 'data = csvread('X_train.csv')', and display the i-th image using 'colormap(gray); imagesc(reshape(data(i,:),112,92))'.
- Y_train.csv: each of the 360 lines contains the label for the corresponding line in X_train.csv
- X_test.csv: 40 test images organized in the same way as in X_train.csv
- Y_test.csv: labels for face images in X_test.csv

1.2 Calculating Eigenfaces

This step will only use X_train.csv:

- **[3 points]** First compute the mean Ψ of the 360 training faces, and subtract the mean from each training data point.
- **[7 points]** Compute the covariance matrix for the 360 training faces, then apply eigen-decomposition on the obtained covariance matrix. In order to save computation time, you could use 'eigs' in MATLAB to compute only the eigenvectors corresponding to the 50 largest eigenvalues.
- **[5 points]** Visualize the Eigenfaces: display the eigenvectors corresponding to the 10 largest eigenvalues as 112×92 images. Print out your results.

- **[5 points]** Face reconstruction. Plot 2 lines of images in your write-up: (1) (first line) the face images corresponding to lines 1, 5, 20, 30, 40 in X_train.csv; (2) (second line) for faces in lines 1, 5, 20, 30, 40 of X_train.csv, plot the face image obtained by projecting the original face into the subspace spanned by the 50 eigenvectors you computed in the second step. To do the projection for a new data point x, first compute $u_i = x^T V_i$ where V_i is the eigenvector you computed, then your reconstructed data point can be obtained as $x' = \sum_i u_i V_i$.

1.3 Using Eigenfaces to Classify a Face Image

Now let's put the eigenfaces to work in face recognition. First subtract each data point in X_train.csv and X_test.csv by the mean Ψ you computed on X_train.csv. Then project each data point in X_train.csv and X_test.csv into the subspace spanned by the 50 eigenvectors you computed previously. Now your face data lives in a 50-dimensional space, instead of the original 112×92-dimensional space.

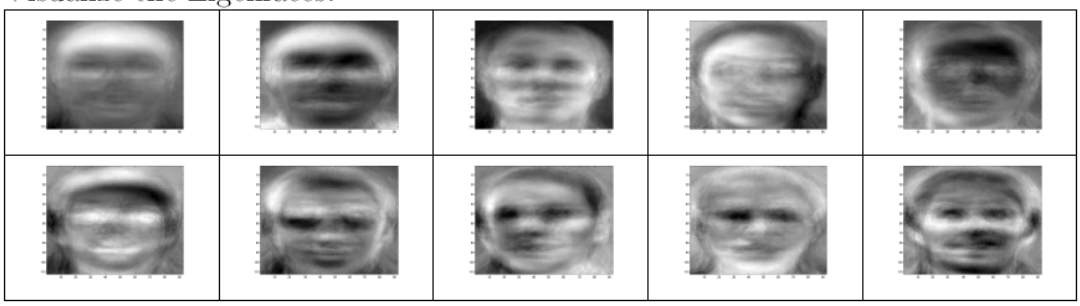
- **[5 points]** The faces in the data set correspond to 40 different people. Apply 1-Nearest neighbor classifier in this 50-dimensional space to predict labels for the faces in X_test.csv. Report your classification accuracy.

1 Principal Component Analysis: Eigenfaces

1.1 ORL Face Data

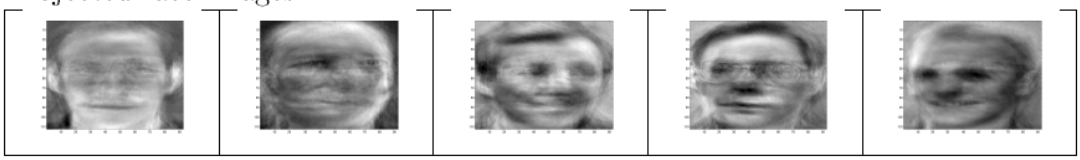
1.2 Calculating Eigenfaces

- Visualize the Eigenfaces:



- Face images 1, 5, 20, 30, 40:

Projected face images:



1.3 Using Eigenfaces to Classify a Face Image

Classification accuracy = 95%

p d CMU, 2014f, EXing, BPoczos, HW3, pr. 1.4 (3vPCA, ASI, FA: implementation + questions!)

1.4 Experiment

Here we will compare the above three methods on two data sets. For ASI and FA we will take $\Psi = I_d$.

A common preprocessing step before applying PCA is to subtract the mean. As we will see, without this preprocessing just taking the SVD of X will give very bad results. For the purposes of this problem we will call these two variants "demeaned PCA" and "buggy PCA". Sometimes, after subtracting the mean we also apply a diagonal scaling so that each dimension has unit variance. We will call this normalized PCA.

One way to study how well the low dimensional representation captures the linear structure in our data is to project it back to D dimensions and look at the reconstruction error. For PCA, if we mapped it to d dimensions via $z = Vx$ then the reconstruction is $V^\top z$. For the preprocessed versions, we first do this and then reverse the preprocessing steps too. For ASI we just compute $Az + b$. For FA, we will use the posterior mean $\mathbb{E}[z|x]$ as the lower dimensional representation and $Az + b$ as the reconstruction. We will compare all four methods by the reconstruction error on the datasets.

Please implement code for the five methods: Buggy PCA (just take the SVD of X) , Demeaned PCA, Normalized PCA, ASI, FA. In all cases your function should take in an $n \times d$ data matrix and d as an argument. It should return the the d dimensional representations, the estimated parameters, and the reconstructions of these representations in D dimensions. For FA, use the values obtained from ASI as initializations for A. Set η based on the reconstruction errors of ASI. Use 10 iterations of EM.

You are given two datasets: A two Dimensional dataset with 50 points `data2D.mat` and a thousand dimensional dataset with 500 points `data1000D.mat`.

For the $2D$ dataset use $d = 1$. For the $1000D$ dataset, you need to choose d. For this, observe the singular values in ASI and see if there is a clear "knee point" in the spectrum. Attach any figures/ Statistics you computed for this to justify your choice.

For the $2D$ dataset you need to attach the a plot comparing the orignal points with the reconstructed points for all five methods. For both datasets you should also report the reconstruction errors. The given starter code does all of this for you so you need to just attach the results.

These were our errors for the $2D$ dataset. If your answers do not tally, please check with the TAs.

```
>> q14
Reconstruction Errors:
Buggy PCA: 0.365284
Demeaned PCA: 0.008448
Normalized PCA: 0.008454
ASI: 0.008448
FA: 0.008526
```

Questions

1. Look at the results for Buggy PCA. The reconstruction error is bad and the reconstructed points don't seem to well represent the original points. Why is this ?
   Hint: Which subspace is Buggy PCA trying to project the points onto ?
2. In both demeaned PCA and ASI the errors are identical. But the Z values are not. You can check this via the command `norm(Z2-Z4, 'fro')`. Explain why ?.
3. The error criterion we are using is the average squared error between the original points and the reconstructed points. In both examples ASI (and demeaned PCA) achieves the lowest error among all methods. Is this surprising ? Why ?

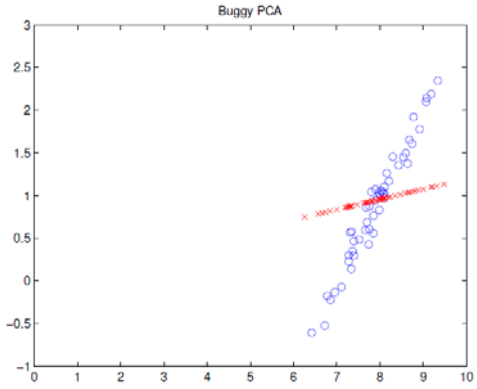
Please submit your code along with your results.

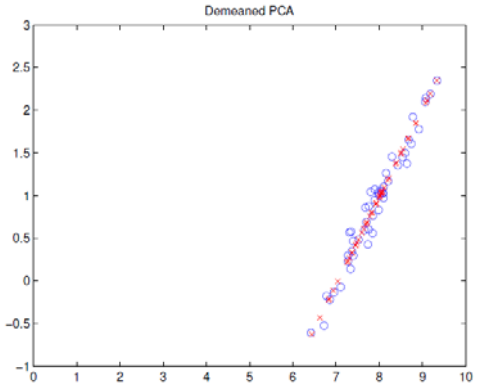
Point Allocation

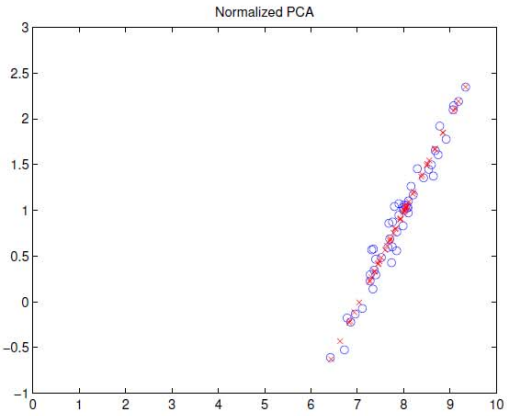
- Implementation of all five methods: **(6 Points)**
- Results - Figures and errors **(3 Points)**
- Choice of d for $1000D$ dataset and appropriate justification: **(1 Point)**
- Questions **(3 Points)**

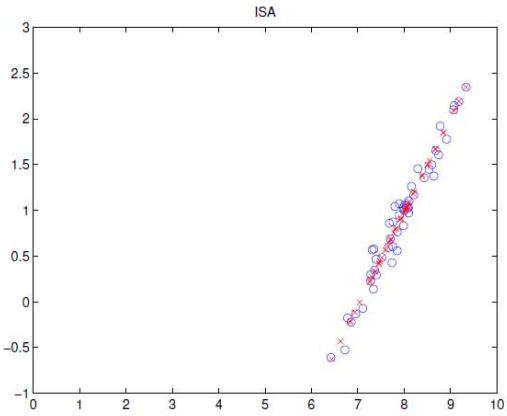
1.4 Experiment

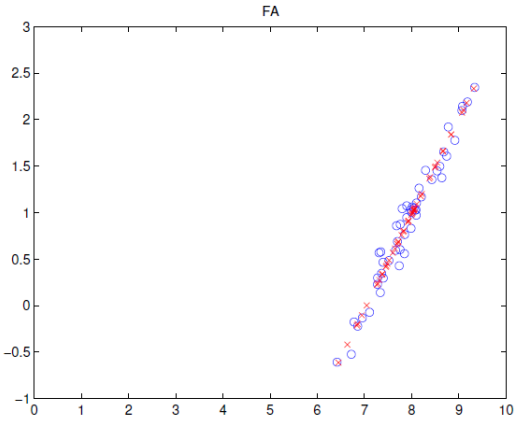
Results for the 2D dataset





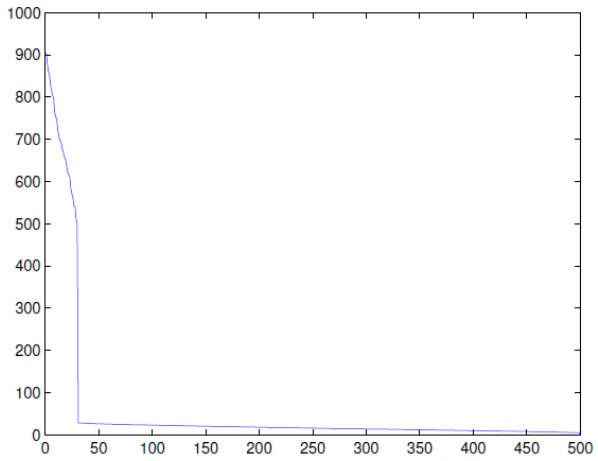






Results for the 1000D dataset

We used $d = 30$ since the singular values fall off sharply after this. See figure below:



This was the output.

```
>> q14
Reconstruction Errors:
Buggy PCA: 777.871447
Demeaned PCA: 272.546928
Normalized PCA: 273.140905
ASI: 272.546928
FA: 272.549605
```

Answers to Questions

1. When you SVD without demeaning on the dataset, you are find the best linear (as opposed to affine) subspace. The first principal component then will then be from the origin towards the data and other principal components will be orthogonal to this.
2. This is because the reconstructions are identical in both cases even if the representations are not.
3. No. Since in ASI we are directly minimizing this error criterion.

This is our implementation.

.....

p CMU, 2014s, BPoczos, ASingh, HW4, pr. 1 (PCA on iris dataset)

Principal Component Analysis (Dani; 10 Points)

For this homework problem, you will implement Principal Component Analysis on the Iris dataset. The Iris dataset contains classifications of iris plants based on four features: sepal length, sepal width, petal length, and petal width. There are three classes of iris plants on this dataset: Iris Setosa, Iris Versicolor, and Iris Virginica. You can download the dataset from `http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data`.

Implement Principal Component Analysis using any language of your choice. Use the first and second principal components to plot this dataset in 2 dimensions. Use different colors for each of the three classes in your plot.

See Figure 1. Grading guidelines:

- **0 point** if there is no plot.
- **2 points** if the plot looks completely wrong.
- **8 points** if the plot looks correct but different from the solution.
- **10 points** if the plot is similar to the solution.

p CMU, 2014s, SKim, HW3, pr. 3.1 (PCA + PCA Embedding?)

3.1 Principal Component Analysis (PCA) [10 pts]

In this question you will implement the PCA algorithm. The dataset for this question can be downloaded at http://www.cs.cmu.edu/~10601b/hw/hw3/data.tar.

1. **PCA [8 pts]**
   Complete the function [P] = pca(X, k).
   - k is the number of principal components to return.
   - P is an $f \times k$ matrix containing the first k principal components, where column i contains the ith principal component.
2. **PCA Embedding [2 pts]**
   Complete the function [Y] = pcaEmbed(X, P).
   - P is an $f \times k$ matrix of principal components, where column i contains the ith principal component.
   - Y is a $n \times k$ matrix, where column i contains the projection of X onto the ith principal component.

p d CMU, s.10-701, HW5, pr. 2 (PCA and FA + questions?)

2 Principal Component Analysis and Factor Analysis (extra credit points: 20%)

Generate 200 three-dimensional points (X_1, X_2, X_3) according to

$$X_1 \sim Z_1 \tag{1}$$

$$X_2 = X_1 + 0.001Z_2 \tag{2}$$

$$X_3 = 10Z_3 \tag{3}$$

where Z_1, Z_2, Z_3 are independent, standard normal random variables ($Z_1, Z_2, Z_3 \sim \mathcal{N}(0, 1)$).

1. Compute the leading principal component and factor analysis directions.
2. Show that the leading principal component aligns itself in the maximal variance direction X_3; while the leading factor essentially ignores the uncorrelated component X_3 and picks up the correlated component $X_2 + X_1$.

You do not need to implement PCA or FA by yourself. Instead, use the matlab functions `princomp` and `factoran` in the statistics toolbox. Use the matlab function `biplot` to visualize the results and hand in the plots.

2.2 Face Recognition using PCA (19 pts)

In this problem we will use PCA to recognize faces. We will work with *AT&T database* which has 10 near frontal images of 40 individuals under different illuminations per individual. The data and code template

can be downloaded from the class website [1].

The `data` folder has `orl_pca.mat` file which contains the train and test datasets. We represent each image as a 1024 dimensional vector (the images have 32×32 pixels). Arrays `xtrain, xtest` in `yale_pca.mat` contain the images for training and testing. Arrays `ytrain, ytest` in `yale_pca.mat` contain the labels of images, where each label corresponds to an individual (labels range from 1 to 40). The `code` folder has the code template and sample scripts for viewing the images.

Question 6

(**4 pts**) First write a function to perform PCA.

1. matlab users complete `findEigenFaces_pca.m` file.
2. python users complete `pca` function in `findEigenFaces.py` file.

This function takes an array $\mathbf{X} \in \mathbb{R}^{n \times p}$ (where n is the total number of images and p is the number of pixels in the image) and k (the number of principal components to compute) as inputs. It returns the mean vector $\hat{\mu}_n \in \mathbb{R}^p$ containing the average of row vectors of $\mathbf{X}$ and an array $\mathbf{V} \in \mathbb{R}^{k \times p}$ that contains k eigenvectors of the covariance matrix of $\mathbf{X}$ (computed after subtracting the mean vector). These should be sorted in ascending order by eigenvalue (i.e., $\mathbf{V}(1,:)$ is the eigenvector with the k^{th} largest associated eigenvalue) and normalized (i.e., $\text{norm}(\mathbf{V}(1,:)) = 1$). MATLAB users can use the following function: `eig` and python users can use the following function: `scipy.linalg.eigh`.

2.2.1 Experiments

With the mean ($\hat{\mu}_n$) and matrix of eigenvectors ($\mathbf{V}$) learned from the training data, you can project other data points into this eigen space. Let x_{test} be a test point. Then to project it into the eigen space simply subtract the mean vector and multiply by the eigenvector matrix (i.e, compute $\mathbf{V}(x_{test} - \hat{\mu}_n)$). This will give you a vector of length k. Given any point $z \in \mathbb{R}^k$ in the eigen space we can reconstruct the point in the original space as $\mathbf{V}^T z + \hat{\mu}_n$.

Question 7

(**2 pts.**) Take a test image and project it into the eigen space (use $k = 5, 10, 20, 40$). Use the projections to reconstruct the test images. Display the reconstructed images along with the original test image. (You can use the commands in `main.py, main.m` files to display images.)

[1] https://sites.google.com/site/10715advancedmlintro2017f/homework-exams

(a) True Image (b) $k = 5$ (c) $k = 10$ (d) $k = 20$ (e) $k = 40$

Figure 3: Image Reconstruction using PCA. We picked a random test point to generate these images.

We will now use PCA to perform face recognition. Given any image our goal is to predict the label of the image. Here is the algorithm we will use

- Perform PCA on `xtrain` and learn top k principal directions.
- Let $\{x_i, y_i\}_{i=1}^n$ be the feature vectors and labels of training data points.
- Let z_i denote the projection of x_i into the eigen space. And for each class l, let Ω_l denote the "projection for class l", which is defined as the mean of the projections of training points belonging to class l

$$\Omega_l = \frac{\sum_{i=1}^n I[y_i = l] z_i}{\sum_{i=1}^n I[y_i = l]},$$

  where $I[\cdot]$ is the indicator function.

- Given any test point x_{test} we predict its label as follows:
  * Project x_{test} into the eigen space. Let z_{test} be the projection.
  * Assign x_{test} to the class l which minimizes $\min_l \|z_{test} - \Omega_l\|_2$.

Question 8

(6 pts) Write a function which computes the centers of each class (Ω_l) in the eigen space.

1. matlab users complete `findEigenFaces_centers.m` file.
2. python users complete `centers` function in `findEigenFaces.py` file.

This function takes a arguments $\{\hat{\mu}_n, V,$ `xtrain, ytrain`$\}$, where `xtrain, ytrain` is the training dataset and $\hat{\mu}_n, V$ are the mean vector and principal directions computed on `xtrain`. It outputs $\Omega \in \mathbb{R}^{l \times k}$, where i^{th} row of Ω contains the center of i^{th} class in the eigen space.

Question 9

(4 pts.) Write a function which predicts the labels of test data points using the algorithm described above

1. matlab users complete `findEigenFaces_pred.m` file.
2. python users complete `pred` function in `findEigenFaces.py` file.

This function takes as arguments $\{\hat{\mu}_n, V,$ `xtest`$\}$, and returns the predicted labels of points in `xtest`.

Question 10

(3 pts.) Plot training and test accuracies as you vary k from 1 to 50. (You can use the `main.py, main.m` files which contain the code to compute accuracy.)

Answer

Test Accuracy at $k = 1$: 0.125.

Test Accuracy at $k = 2$: 0.1625.

Test Accuracy at $k = 5$: 0.562500.

Test Accuracy at $k = 10$: 0.725000.

Test Accuracy at $k = 20$: 0.800000.

Test Accuracy at $k = 40$: 0.837500.

Test Accuracy at $k = 50$: 0.85.

p Radford, Spring 2014 (New folder), PCA + MLP to classify whether images included in webpages are advertisements or not.

STA 414/2104, Spring 2014 — Assignment #3

Due at the start of class on April 3. Please hand it in on 8 1/2 by 11 inch paper, stapled in the upper left, with no other packaging.

This assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own. In particular, you should not leave any discussion with someone else with any written notes (either on paper or in electronic form).

In this assignment, you will use a combination of Principal Components Analysis (PCA) and a multilayer perceptron (MLP) neural network to classify whether images included in webpages are advertisements or not.

The data is from a 1999 paper by Nicholas Kusmerick on "Learning to remove Internet advertisements", and is available from the UC Irving Machine Learning Repository, from the URL `archive.ics.uci.edu/ml/datasets/Internet+Advertisements`. I have processed the data to remove three covariates that are often missing, to remove a few cases in which another covariate is missing, to remove a few covariates that are zero in all but nine or fewer cases, to randomly reorder the cases, and to divide the cases into a training set of 1300 cases and a test set of 1964 cases.

After this processing, there are 1521 covariates, all of which are binary (with values of 0 or 1). One of these covariates indicates whether the URL for the image is for the same server as the main webpage. Others indicate whether or not certain words occur in text associated with the image. Most of these covariates are 0 for most cases — in particular, 1281 of the covariates have the value 1 in less than 1% of the training cases).

The data is available from the course web page in the following files:

`a3trnx`	covariate values for the training cases
`a3trny`	class labels for training cases
`a3tstx`	covariate values for the test cases
`a3tsty`	class labels for test cases

The class label is 1 if the image is an advertisement and 0 if the image is not an advertisement. You should of course look at the labels for test cases only at the very end, to see how well you did. You can read the covariate files with `read.table`, with the `head=FALSE` option, and then convert the data to a matrix with `as.matrix`. You can read the labels with `scan`.

For both computational and statistical reasons, it may be desirable to reduce the dimensionality of the covariates from 1521 to a much smaller number. You will do this using PCA, keeping the first 10, 20, or 40 principal components. You will then see how well you can predict the class using only these 10, 20, or 40 principal components as covariates. You should first try predicting the class using maximum likelihood logistic regression, and then try using a multilayer perceptron network with 10 hidden units. You should try training the MLP network by gradient descent on minus the log likelihood alone, and on minus the log likelihood plus a quadratic penalty on the input-to-hidden weights (only).

To make predictions for the test cases, you will need to make choices of whether to use 10, 20, or 40 principal components, of what learning rate to use, of the number of gradient descent iterations to do, and of whether or not to use a penalty and if so what its magnitude should be. You should make these choices by splitting the 1300 training cases into an estimation set (the first 1000 training cases) and a validation set (the last 300 training cases). You should find principal component directions and fit logistic regression models using only the data in the estimation set. When fitting the multilayer perceptron models, you should do gradient descent using the gradient computed from only the cases in the estimation set. You should compute the average log probability of the cases in the validation set according to the parameters found for the logistic regression and MLP models in order to choose among them, and for the MLP models, to choose how many iterations to train for, and what penalty magnitude to use. (You could also choose the learning rate this way, but you may instead choose it so that minus the log likelihood from the estimation set decreases steadily.)

You should find the principal component vectors using the `pca.vectors` function from the PCA example (for week 10) on the course web page, which you can use without modification. You should find these vectors using the estimation set of 1000 cases, *not* the full training set, or the test set. Once you have found the principal component vectors, you can compute the projections of the estimation, validation, and test cases on these vectors using the `pca.proj` function. You will use these projections on the first 10, 20, or 40 principal component vectors as covariates in your classification models.

You can fit logistic regression models by maximum likelihood using R's `glm` function, with the option `family="binomial"`. For example, `glm(y~X,family="binomial")` models the vector of binary responses `y` in terms of the matrix of covariates `X`.

You should fit multilayer perceptron models using a modified version of the example functions provided on the course web page (for week 10). You will need to modify these functions to model a binary response, replacing the squared error for a training case by minus the log probability of the response. You will also need to modify the network training function so that it can minimize the sum of minus the log likelihood over training cases plus a penalty proportional to the sum of the squares of the weights on input to hidden connections, with the penalty factor being an argument of the training function (set to zero for no penalty). In terms of the week 10 lecture titled "Avoiding Overfitting Using a Penalty", only the first penalty term (sum of squares of $w_{kj}^{(1)}$ multiplied by λ_1) is used, with λ_2 fixed at zero. Note that there is no penalty on the "bias" parameters, $w_{0j}^{(1)}$.

You will need to write a function that computes the average log probability of the responses for the 300 validation cases for all values of the parameters found in a network training run. You can use this to select which set of parameters found during training seems best (which will not necessarily be the parameters from the last iteration). You can also compare the best validation performance for different training runs, that use different numbers of principal components, or different penalty magnitudes. You will also need to compute the average log probability of responses in the validation set using the logistic regression coefficients found using `glm`, using different numbers of principal components. Using these figures on validation performance, you can select which model (and set of model parameters) to use for making predictions for test cases.

For the model and parameter set that you select, you should compute both the average log probability of the responses for test cases, and the classification error rate, when classifying a test case as being an advertisement (label 1) when the model says its probability of being an advertisement is greater than 0.5. These figures indicate how well you would have done if this were a real problem, in which you have to make predictions for test cases before knowing their true values.

You should also find the average log probability of the response and the classification error rate on test cases for all the other models and training runs that you did (using the best parameter set from a run as chosen using the validation set). These performance figures are of interest in seeing how reliable the performance on the validation set was as an indication of performance on the test set.

You should hand in your modified MLP functions, the other functions you wrote, the R script you used to read the data, fit models, and report the results, the output of this R script, and a discussion of the results. In your discussion, you might comment on how easy or hard it was to "tune" the network training (selecting a suitable learning rate, and a suitable number of gradient descent iterations), how fast or slow the training was, whether training with a penalty helped or not, how reliable performance on the validation set was as an indicator of which model was best, and what the effect was of using more or fewer principal components.

p Radford, Fall 2008, (2), PCA on gene expression

STA 437/1005, Fall 2008 — Assignment #3

Due on November 24, at start of lecture. Worth 10% of the course grade.

This assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own. In particular, you should not leave any discussion of this assignment with any written notes or other recordings, nor receive any written or other material from anyone other than your instructor by any other means, such as email.

For this assignment, you will analyse a data set on gene expression in yeast, at different points in its cell cycle. The data was collected for the analysis reported in the following paper:

> P. T. Spellman, *et al.* (1998) "Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization", *Molecular Biology of the Cell*, vol. 9, pp. 3273-3297.

The paper above used other methods of analysis, but for this assignment, you should analyse the data using Principal Component Analysis (PCA). I have written some R functions for doing PCA that you should use. (R has a built-in function for PCA, but it doesn't handle data sets where the number of variables is greater than the number of cases.) These functions, the data, some R hints, and links to the text of the paper and to the original source of the data are available from the course web page:

`http://www.utstat.toronto.edu/~radford/sta437`

Look in the section for Assignment 3. (Note that it isn't necessary for you to read the original paper or consult the original data source to do this assignment.)

Yeast is a single-celled organism whose celluar organization is similar to that of humans (and other "Eukaryotes"). It has approximately 6000 genes, many of which have analogues among the 25000 human genes. Except when they decide to have sex, yeast reproduce by cellular division (producing "buds"). The yeast cells go through a cycle, in which they start out small, grow over time, divide, and then start growing again. During the course of this cycle, the activities of some groups of genes increase and decrease, as their functions are required for only part of the cycle. Some other groups of genes are active during the whole cycle, or may not be active at all during an experiment, if their functions (eg, for mating, or coping with adverse conditions) are not needed in the environment of the experiment.

The purpose of the study above was to determine which genes have activities that vary during the yeast cell cycle. Data on the activity (also called the "expression") of nearly all yeast genes was obtained using a DNA microarray. Measurements of gene activity cannot be obtained in this way for single cells; only the average activity for a large number of cells can be measured. To see what happens during the cell cycle, it is therefore necessary to obtain large numbers of cells that are all at (approximately) the same point in the cell cycle. In the data you will see, this was done in two different ways. For the "alpha" data, a cell culture was exposed to a pheromone that had the effect of synchonizing all the cells at a particular

point in their cycle. Samples of cells were then taken at seven minute intervals after that, as they all grew and divided in approximate synchrony. For the "cdc15" data, the cells were arrested at a certain point in their cycle by low temperature, and then allowed to continue in approximate synchrony when the temperature was raised, with measurements taken every 10 or 20 minutes thereafter. These two methods of synchronization may have unintended side effects, and the different experimental environments result in different growth rates for the cells, so the data is not completely comparable. The experimenters expected that the "alpha" data will show about two cell cycles, and that the "cdc15" data will show about three cell cycles.

The data for the two experiments ("alpha" and "cdc15") that you will see is actually the log base 2 of the ratio of average gene activity in a large number of cells thought to be at a certain point in the cell cycle to the average gene activity of a large number of cells that are in all stages of the cell cycle. Positive numbers therefore indicate that a gene is more active than it typically is; negative numbers that it is less active than it typically is. The technology isn't perfect, so some data was missing. I replaced these missing measurements by zero, since that is a "neutral" value in this context. I also eliminated genes for which all data was missing for one or both experiments.

Previous research had identified 104 genes as probably varying in activity over the cell cycle. I have provided data for you identifying these genes. (Actually, only 92 are identified, since two systems of yeast gene naming are used, and I didn't succeed in converting all the names.) The original analysis used these genes as a guide to identifying other genes that vary in activity over the cell cycle, but for this assignment, you should use this data only at the end of the analysis, as described below.

The data is stored in a file with a header line giving the names of 43 variables, with one line after that for each of 5894 genes. The line for each gene starts with the gene name, after which the values of the 43 variables are given. The 43 variables are as follows:

1	prev	1 if gene was previously identified as varying over the cycle, 0 otherwise
2	alpha_0	Activity ratio of this gene just after administering the alpha pheromone
3	alpha_7	Activity ratio 7 minutes after administering the alpha pheromone
4	alpha_14	Activity ratio 14 minutes after administering the alpha pheromone
	...	
19	alpha_119	Activity ratio 119 minutes after administering the alpha pheromone
20	cdc15_10	Activity ratio 10 minutes after resuming growth at higher temperature
21	cdc15_30	Activity ratio 30 minutes after resuming growth at higher temperature
	...	
43	cdc15_290	Activity ratio 290 minutes after resuming growth at higher temperature

This data can be read with the `read.table` function, which will set the column names of the data frame it reads to the variable names, and the row names of the data frame to the names of the genes. You should convert this data frame to a matrix with the `as.matrix` function, since for this analysis you will need to transpose it (using the `t` function), so that the genes become the variables (ie, $p = 5894$), and the different time points become the cases

(eg, $n = 42$ if both experiments are analysed together). Remember that the `prev` indicator is to be used only at the end of the analysis.

A major problem with analysing this data is that the 5894 genes are far too many to look at manually, and also greatly exceed the number of observations. It therefore makes sense to try to use PCA to reduce the number of variables from 5894 to something more managable. For this assignment, you should look at the first $k = 4$ principal components, and see whether these contain enough information to see the cell cycle, and identify which genes vary in activity during it.

There are several options when applying PCA to this data. First, we could apply PCA using all 42 observations from both experiments, or using just the 18 observatins from the "alpha" experiment, or using just the 24 observations from the "cdc15" experiment. Second, we could use the variables without scaling them (ie, use the covariance matrix), or we could first divide each variable by its standard deviation (ie, use the correlation matrix). This gives six combinations of options which you should try. You should always center the variables (ie, subtract their mean). (Note that centering and scaling will be done automatically by the `pca.vectors` function that I provide, if the right option is set.)

Having found the first 4 principal component vectors, you can find the projections of the observations on these vectors, which reduces the data set to one with 42 observations and 4 variables. Note that you can find the projections for all 42 observations even if you used only a subset of them to find the principal component vectors.

Once you have such a reduced data set with 4 principal component values, you should plot these values for the 18 observations from the "alpha" experiment and for the 24 observations from the "cdc15" experiment, and see whether any of these values seem to follow cycles. For each of the six ways of finding principal components, you should try to identify two variables that vary periodically (not in exactly the same way), and then make a 2D plot of the observations for each experiment with respect to these two principal components. We would hope to see something resembling circular movement as the cycle proceeds. Comment on what you have found, including whether using the covariance or the correlation matrix worked better.

Next, using whatever principal components seem like the best indicators of the cell cycle, you should devise some way of categorizing a gene as varying in activity over the cell cycle, or not. The paper above identified approximately 800 genes as varying in activity over the cell cycle. You should also select approximately 800 genes that seem the most cyclic. Finally, you should see what fraction of the previously identified cyclic genes are in the set that you identified. It is only at this last stage that you should look at the `prev` indicator in the data file, which identifies these genes previously thought to be cyclic. You should discuss why you used the method you chose for categorizing genes, and how well it worked.

You should hand in your discussions, the R output and R plots that support your conclusions, and a listing of the R commands you used.

p Radford, Fall 2010, (3), explore data graphically + PCA on 2 datasets

(1. predicting the average wind speed on Mondays from the wind speeds on the preceding Sundays.

2. gene expression)

STA 437/1005, Fall 2010 — Assignment #1

Due at start of lecture on November 15. Please hand it in on 8 1/2 by 11 inch paper, stapled in the upper-left corner, without any folder or other packaging around it. If really necessary, you can submit it by email (to radford@stat.utoronto.ca*), but please do this only if you can't easily hand in a paper copy.*

This assignment is worth 10% of the course grade. It is to be done by each student individually. You may discuss this assignment in general terms with other students, but the work you hand in should be your own. In particular, you should not leave any discussion of this assignment with any written notes or other recordings, nor receive any written or other material from anyone else by other means such as email.

In this assignment, you will analyse two datasets taken from the scientific literature, exploring them graphically, and applying principal components analysis. You should do your analyses using R.

For both datasets, you should produce a report explaining your findings, and justifying them with suitable plots or textual output from R. You should hand in only a reasonable amount of output, as needed for your analysis, not every conceivable plot. You *must* hand in a listing of the R commands you used for the analysis, including the commands used to produce the plots you hand in. You do not need to repeat in your report what I say below about the datasets.

The datasets, along with some relevant links, are available from the course web page, at

`http://www.utstat.utoronto.ca/~radford/sta437/`

I will also post some hints on using R for this assignment. You should check the web page regularly in case there are any corrections made to the assignment (any corrections will be highlighted in bold at the top of the web page).

The two parts are worth equal marks.

Part I: This dataset contains the daily average wind speeds at 12 locations in the Republic of Ireland, for every Monday from 1961 to 1978, and every preceding Sunday, extracted from a larger dataset containing average wind speeds for all days from 1961 to 1978.

The data is discussed in the following paper:

> Haslett, J. and Raftery, A. E. (1989) "Space-time Modelling with Long-memory Dependence: Assessing Ireland's Wind Power Resource" (with discussion), *Applied Statistics*, vol. 38, pp. 1–50.

These authors' purpose was prediction of wind power resources at possible new locations, but for this assignment, we will just consider predicting the average wind speed on Mondays from the wind speeds on the preceding Sundays. When looking at pairs of days a week apart, the Sunday/Monday pairs are almost independent, so we can ignore the time series aspect of the full dataset. (Of course, a Sunday and the following Monday are not independent; that dependence is what we will be trying to model.)

Two data files, one for Sundays and one for Mondays, are on the webpage. Each has a header line giving the names of the variables, which are abbreviations for the 12 locations.

The order of the 12 locations is roughly from South to North. The observations have labels composed of year/month/day (two digits each). This format is suitable for reading with the `read.table` function using the `head=TRUE` option.

We will sometimes look at average wind speeds over all 12 locations. You can compute these averages using the `rowMeans` function.

The paper by Haslett and Raftery says that taking the square roots of the wind speeds improves normality of the data, and that there is an effect of season on wind speed. You should examine whether these claims are correct, looking at the average wind speed over all 12 locations on Sundays. You should consider three possibilities — no transformation, taking the log of the average wind speed, and taking the square root of the average wind speed. For each possibility, you should fit a linear model of average wind speed or its transformation (using the `lm` function) with sin and cos of the "time of year angle" as covariates. The time of year angle can be computed as 2*pi*((week/52.18)%%1), where "week" is a vector of week numbers, and 52.18 is the number of weeks in a year. Here, "%%1" takes the fractional part of a number. The week numbers start at 1 for the first Sunday, and increase by 1 for each observation; such a sequence can be obtained with something like `1:nrow(data)`.

After fitting this linear model for the seasonal effect, you should find the residuals, and check whether there are still any seasonal effects in the residuals, and whether the residuals appear to be normally distributed. Report your conclusions. Checking for seasonal effects can be done by plotting observations against the year angle. Checking for normality can be done using histograms and QQ plots.

For the rest of the analysis, you should look at the the square roots of the wind speeds minus the fitted value for that day, from the linear model fit to the square root of the average wind speed on Sunday. You can ignore the difference in time of year between a Sunday and a following Monday. (Note that this procedure is not necessarily the best — it assumes that all 12 locations have the same seasonal effects, and that the order of taking square roots and averages isn't important — but it's what we'll do for this assignment.)

You should start by looking at pair-wise scatterplots of this (transformed) Sunday data from the 12 locations. Report any outliers that you find, and comment on the correlations between locations. You can also look at the correlation matrix.

Next, using the transformed data for Sundays, you should find the principal components, using the `prcomp` function, without scaling (ie, use the covariance matrix rather than the correlation matrix). You should look at the components found (the rotation and standard deviations) and comment on what meaning (if any) can be assigned to them.

Finally, you should look at predicting the (transformed) average wind speed on Monday using data from the previous Sunday. You should consider linear models with the following sets of covariates:

- Just the (transformed) average wind speed on the previous Sunday.
- All the (transformed) wind speeds on the previous Sunday.
- The first 1, 2, or 3 principal components from the previous Sunday.

Comment on the performance of each of these methods, using "Adjusted R-Squared" (output by `summary(lm(...))`) as the criterion for how good each linear model is. Can the 12 measurements be condensed to fewer numbers with little or no loss of predictive performance?

Part II: This data set contains the levels of expression of genes in samples of lung cancer cells from 39 patients. The aim is to use the expression levels of these genes to predict whether the lung cancer will recur in a patient after surgery.

The data is from the following study:

> Wigle, D., Jurisica, I., N. Radulovich, M. Pintilie, J. Rossant, N. Liu, C. Lu, J. Woodgett, I. Seiden, M. Johnston, S. Keshavjee, G. Darling, T. Winton, B. Breitkreutz, P. Jorgenson, M. Tyers, F. A. Shepherd, M.S. Tsao. (2002) "Molecular profiling of non-small cell lung cancer and correlation with disease-free survival", *Cancer Research*, vol. 62, pp. 3005–3008.

The measurement for a particular gene and patient is the ratio of the expression level of that gene in the patient's cancer cells divided by the expression level of that gene is a reference sample from other tissues. There may have been some normalization done as well (I can't tell what exactly was done from the paper and data file). All the measurements are positive.

The study above started with about 19,000 genes, and then looked a subset of 2899 genes. I reduced the number further to the 1644 genes for which the expression measurements were missing in at most two patients. I filled in any missing measurements using the median of the non-missing measurements over all 1644 genes and 39 patients. (This is simple, but rather crude; a better method would be desirable in a real application.)

Two data files are provided. One contains the expression levels of the 1644 genes (with missing values filled in) for the 39 patients, with a header line giving information identifying the gene. The observations are labelled with identifiers for the patients. The other data file contains information on the patients (ordered the same as in the gene expression data file), with a header line giving the names of the variables ("id", "type", "stage", and "recur"). Only the "recur" variable is used in this assignment. It is 0 if no recurrence of cancer was observed, and 1 if cancer did recur. Both data files are suitable for reading with the `read.table` function using the `head=TRUE` option.

In this assignment, we will see how well the "recur" variable can be predicted from the gene expression data, by fitting a linear model. A linear model is not really appropriate for a response variable like this, which takes values 0 or 1 — later, we will cover logistic regression models, which would be more appropriate. Nevertheless, it is meaningful to look at how well a linear model does, as a way of seeing how informative the gene expression data is about cancer recurrence.

It is not possible to do a standard least squares fit of a linear model for recurrence using all 1644 gene expression measurements as covariates, with only 39 patients. (Least squares requires that the number of observations be at least as large as the number of covariates.) Accordingly, we will look at predicting recurrence using some small number of principal components found from the 1644 gene expression measurements.

Several issues arise in doing this:

- Should the gene expression measurements be transformed before finding principal components?
- Are there any outliers that should be removed or adjusted before finding principal components?
- Should the principal components be found from the covariance matrix or the correlation matrix?
- How many principal components should be used in a linear model for recurrence?

You should investigate these issues for this assignment, and report your results. For transformations, you should consider at least the possibilities of no transformation and a log transformation. You should assess how well a set of covariates does at predicting the "recur" variable using the "Adjusted R-Squared" output from `summary(lm(...))`. You may also want to look at scatterplots of pairs of principal components (or the original gene expression measurements) in which patients with and without recurrence of cancer are identified by different colours or different plot symbols.

p Radford, Fall 2010, (4), PCA + FA on the same 2 datasets?

STA 437/1005, Fall 2009 — Assignment #3

Due at the start of the lecture on November 30. (Note that due to lack of time, I've had to abandon the idea of an assignment with a two-part solution/critique form that I discussed earlier.)

Please hand this assignment in on 8 1/2 by 11 inch paper, stapled in the upper-left corner, without any folder or other packaging around it.

This assignment is worth 12% of the course grade. It is to be done by each student individually. You may discuss this assignment in general terms with other students, but the work you hand in should be your own. In particular, you should not leave any discussion of this assignment with any written notes or other recordings, nor receive any written or other material from anyone else by other means such as email.

For this assignment, you will look again at the two data sets you used for the second assignment. This time, you will apply principal component analysis (PCA) and factor analysis (FA) to the data, comparing the results you obtain with these methods, and seeing the effect of reducing dimensionality on use of the data for regression or classification. Another purpose of the assignment is to give general insight into properties of PCA and FA.

This handout, the data sets, some hints about useful R commands, and a solution to the second assignment are (or will shortly be) available from the course web page, at `http://www.utstat.utoronto.ca/~radford/sta437/`

Data set 1:

I have provided a version of this data set with nine observations deleted, because they have values for one or more of the variables that are of doubtful accuracy. This version also omits the "density" variable (retaining "pcfat") and the "age" variable. You should use this version for this assignment, and not remove any additional observations as outliers. You should read this data with the "head=TRUE" option. The data frame will contain column names that are the indexes of the observations in the original data set (some indexes are therefore skipped).

You should perform PCA using only the 12 variables other than "pcfat". You should try PCA with and without scaling these variables to all have standard deviation one, and discuss whether scaling or not scaling (or something else) seems most appropriate. You should look at scree plots of the variances in successive principal components, and on this basis comment on how many components should (perhaps) be used.

You should also perform factor analysis on this data, using the 12 variables other than "pcfat". You should try models with one and with two factors.

You should try to interpret the coefficients found by FA and PCA in terms of the meaning of the variables, using your common sense knowledge of the variability of human body proportions.

You should compare the two-factor model with the first two principal components (both with scaling and without scaling). For this, you should look at the coefficients of the linear combinations of the original 12 variables (after centering) that are used to project onto the components found with PCA, and the linear combinations that are used to predict the values of the factors with the FA model. You should take into account the non-uniqueness of the FA solution when doing this, as well as the non-uniqueness of the sign for the principal components.

You should also look at predicting "pcfat" from the other 12 variables, and compare linear regression on all 12 variables with linear regression on a smaller number of variables found using PCA (projections on the components) and with a smaller number of variables found using FA (from the regression estimates of the unobserved factors). You can judge how well "pcfat" can be predicted using the adjusted R-squared value output by the "summary" command applied to the output of "lm". You can also consider adding BMI (weight divided by height squared) as an additional predictor in the regression, and see whether this helps.

Data set 2:

For this data set, you should use the same data file as for the second assignment. It should be read with the "head=TRUE" option. You should not remove any observations as outliers.

You should perform PCA using observations in all classes, and using the 36 variables in this data set with the class variable omitted. You should try PCA with and without scaling these variables to all have standard deviation one, and discuss whether scaling or not scaling (or something else) seems most appropriate. You should also look at scree plots of the variances in successive principal component directions, and on this basis comment on how many components should (perhaps) be used.

You should also perform factor analysis on this data (again, for all classes, 36 variables), using two factors. You should compare the results of PCA and FA. For this, you should look at the coefficients of the linear combinations of the original 36 variables (after centering) that are used to project onto the components found with PCA, and the linear combinations that are used to predict the values of the factors with the FA model. You should take into account the non-uniqueness of the FA solution when doing this, as well as the non-uniqueness of the sign for the principal components.

You should try to interpret the coefficients found by FA and PCA (including how many components seem to be needed) in terms of the meaning of the 36 variables, as observations on 4 spectral bands for 9 pixels, and of the context that observations come from three classes.

You should also look at how effective reducing the 36 variables to only 2 variable with PCA or FA would be if the goal is to classify future observations into one of these three classes. You can do this informally by just looking at scatterplots with the class indicated by colour.

p Radford, Fall 2008, (5), PCA on gene expression

STA 437/1005, Fall 2008 — Assignment #3

Due on November 24, at start of lecture. Worth 10% of the course grade.

This assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own. In particular, you should not leave any discussion of this assignment with any written notes or other recordings, nor receive any written or other material from anyone other than your instructor by any other means, such as email.

For this assignment, you will analyse a data set on gene expression in yeast, at different points in its cell cycle. The data was collected for the analysis reported in the following paper:

P. T. Spellman, *et al.* (1998) "Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization", *Molecular Biology of the Cell*, vol. 9, pp. 3273-3297.

The paper above used other methods of analysis, but for this assignment, you should analyse the data using Principal Component Analysis (PCA). I have written some R functions for doing PCA that you should use. (R has a built-in function for PCA, but it doesn't handle data sets where the number of variables is greater than the number of cases.) These functions, the data, some R hints, and links to the text of the paper and to the original source of the data are available from the course web page:

`http://www.utstat.toronto.edu/~radford/sta437`

Look in the section for Assignment 3. (Note that it isn't necessary for you to read the original paper or consult the original data source to do this assignment.)

Yeast is a single-celled organism whose celluar organization is similar to that of humans (and other "Eukaryotes"). It has approximately 6000 genes, many of which have analogues among the 25000 human genes. Except when they decide to have sex, yeast reproduce by cellular division (producing "buds"). The yeast cells go through a cycle, in which they start out small, grow over time, divide, and then start growing again. During the course of this cycle, the activities of some groups of genes increase and decrease, as their functions are required for only part of the cycle. Some other groups of genes are active during the whole cycle, or may not be active at all during an experiment, if their functions (eg, for mating, or coping with adverse conditions) are not needed in the environment of the experiment.

The purpose of the study above was to determine which genes have activities that vary during the yeast cell cycle. Data on the activity (also called the "expression") of nearly all yeast genes was obtained using a DNA microarray. Measurements of gene activity cannot be obtained in this way for single cells; only the average activity for a large number of cells can be measured. To see what happens during the cell cycle, it is therefore necessary to obtain large numbers of cells that are all at (approximately) the same point in the cell cycle. In the data you will see, this was done in two different ways. For the "alpha" data, a cell culture was exposed to a pheromone that had the effect of synchonizing all the cells at a particular

point in their cycle. Samples of cells were then taken at seven minute intervals after that, as they all grew and divided in approximate synchrony. For the "cdc15" data, the cells were arrested at a certain point in their cycle by low temperature, and then allowed to continue in approximate synchrony when the temperature was raised, with measurements taken every 10 or 20 minutes thereafter. These two methods of synchronization may have unintended side effects, and the different experimental environments result in different growth rates for the cells, so the data is not completely comparable. The experimenters expected that the "alpha" data will show about two cell cycles, and that the "cdc15" data will show about three cell cycles.

The data for the two experiments ("alpha" and "cdc15") that you will see is actually the log base 2 of the ratio of average gene activity in a large number of cells thought to be at a certain point in the cell cycle to the average gene activity of a large number of cells that are in all stages of the cell cycle. Positive numbers therefore indicate that a gene is more active than it typically is; negative numbers that it is less active than it typically is. The technology isn't perfect, so some data was missing. I replaced these missing measurements by zero, since that is a "neutral" value in this context. I also eliminated genes for which all data was missing for one or both experiments.

Previous research had identified 104 genes as probably varying in activity over the cell cycle. I have provided data for you identifying these genes. (Actually, only 92 are identified, since two systems of yeast gene naming are used, and I didn't succeed in converting all the names.) The original analysis used these genes as a guide to identifying other genes that vary in activity over the cell cycle, but for this assignment, you should use this data only at the end of the analysis, as described below.

The data is stored in a file with a header line giving the names of 43 variables, with one line after that for each of 5894 genes. The line for each gene starts with the gene name, after which the values of the 43 variables are given. The 43 variables are as follows:

1	prev	1 if gene was previously identified as varying over the cycle, 0 otherwise
2	alpha_0	Activity ratio of this gene just after administering the alpha pheromone
3	alpha_7	Activity ratio 7 minutes after administering the alpha pheromone
4	alpha_14	Activity ratio 14 minutes after administering the alpha pheromone
	...	
19	alpha_119	Activity ratio 119 minutes after administering the alpha pheromone
20	cdc15_10	Activity ratio 10 minutes after resuming growth at higher temperature
21	cdc15_30	Activity ratio 30 minutes after resuming growth at higher temperature
	...	
43	cdc15_290	Activity ratio 290 minutes after resuming growth at higher temperature

This data can be read with the `read.table` function, which will set the column names of the data frame it reads to the variable names, and the row names of the data frame to the names of the genes. You should convert this data frame to a matrix with the `as.matrix` function, since for this analysis you will need to transpose it (using the `t` function), so that the genes become the variables (ie, $p = 5894$), and the different time points become the cases

(eg, $n = 42$ if both experiments are analysed together). Remember that the `prev` indicator is to be used only at the end of the analysis.

A major problem with analysing this data is that the 5894 genes are far too many to look at manually, and also greatly exceed the number of observations. It therefore makes sense to try to use PCA to reduce the number of variables from 5894 to something more managable. For this assignment, you should look at the first $k = 4$ principal components, and see whether these contain enough information to see the cell cycle, and identify which genes vary in activity during it.

There are several options when applying PCA to this data. First, we could apply PCA using all 42 observations from both experiments, or using just the 18 observatins from the "alpha" experiment, or using just the 24 observations from the "cdc15" experiment. Second, we could use the variables without scaling them (ie, use the covariance matrix), or we could first divide each variable by its standard deviation (ie, use the correlation matrix). This gives six combinations of options which you should try. You should always center the variables (ie, subtract their mean). (Note that centering and scaling will be done automatically by the `pca.vectors` function that I provide, if the right option is set.)

Having found the first 4 principal component vectors, you can find the projections of the observations on these vectors, which reduces the data set to one with 42 observations and 4 variables. Note that you can find the projections for all 42 observations even if you used only a subset of them to find the principal component vectors.

Once you have such a reduced data set with 4 principal component values, you should plot these values for the 18 observations from the "alpha" experiment and for the 24 observations from the "cdc15" experiment, and see whether any of these values seem to follow cycles. For each of the six ways of finding principal components, you should try to identify two variables that vary periodically (not in exactly the same way), and then make a 2D plot of the observations for each experiment with respect to these two principal components. We would hope to see something resembling circular movement as the cycle proceeds. Comment on what you have found, including whether using the covariance or the correlation matrix worked better.

Next, using whatever principal components seem like the best indicators of the cell cycle, you should devise some way of categorizing a gene as varying in activity over the cell cycle, or not. The paper above identified approximately 800 genes as varying in activity over the cell cycle. You should also select approximately 800 genes that seem the most cyclic. Finally, you should see what fraction of the previously identified cyclic genes are in the set that you identified. It is only at this last stage that you should look at the `prev` indicator in the data file, which identifies these genes previously thought to be cyclic. You should discuss why you used the method you chose for categorizing genes, and how well it worked.

You should hand in your discussions, the R output and R plots that support your conclusions, and a listing of the R commands you used.

p Radford, Fall 2006, (8), PCA on gene expression (DNA microarrays) + knn

CSC 411, Fall 2006 — Assignment #4

*Due at **start** of lecture on December 6. Note that this assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own. Handing in work that is not your own is a serious academic offense. Fabricating results, such handing in fake output that was not actually produced by your program, is also an academic offense.*

For this assignment, you will try using PCA to reduce the dimensionality of gene expression data obtained using DNA microarrays, and will also see how such a reduction in dimensionality affects performance of a k-nearest-neighbor classifier.

The data (available from the course web page) gives the expression levels of 2000 genes in tissue samples from 62 people. It was used in the following paper:

Alon, U., Barkai, N., Notterman, D. A., Gish, K., Ybarra, S., Mack, D., and Levine, A. J. (1999) "Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays", *Proceedings of the National Academy of Sciences (USA)*, vol. 96, pp. 6745-6750.

40 of the tissue samples are from people with colon cancer; 20 are from people without cancer. A second data file contains these indicators.

In this assignment, you will look this data in two ways — as data on 62 tissue samples (with 2000 variables for each sample), and as data on 2000 genes (with 62 variables for each gene). In both cases,

you should reduce dimensionality using PCA. You should subtract the sample mean of each variable when doing this, but not divide by the standard deviation. (The variables are all measurements of the same type, so the original scaling may be meaningful.)

You should find the principal component directions and the projections of the points on these directions by applying matrix operations in functions you write yourself, not using some existing package for PCA. You should use the method discussed in the lecture slides for finding principal components when the number of variables is larger than the number of points.

You should reduced the dimensionality of the data on the 2000 genes to the first 10 principal components. You should then look at scatterplots of pairs of principal components, and comment on whether anything that seems of interest can be found.

You should reduce the dimensionality of the data on the 62 tissue samples to the first 25 principal components. You should then consider the performance of a k-nearest-neighbor classifier (with Euclidean distance) for whether the tissue is cancerous or not, with $k = 5$. You should try this using the original data (2000 variables), using just the first principal component, using just the first two principal components, etc., up to using the first 25 principal components. You should evaluate performance using leave-one-out cross validation, in terms of error rate (plus any other measure you find interesting).

You may use the k nearest neighbor classifier (in R) on the course web page, or any other software for doing that. You should write your own function for doing the leave-one-out cross validation, however.

You should hand in listings of the functions you wrote, a discussion of the results, and suitable plots or other output that illustrate and justify your conclusions.

p MIT, Spring 2016, project on digits: MLP + CNN with PCA...

MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.036—Introduction to Machine Learning
Spring 2016

Project 2: Handwritten Digit Recognition Issued: Tue., 3/1 Due: Fri., 4/01 at 9am

Project Submission: Please submit a .zip file of your project on the Stellar web site by 9am, 4/01. The zip file should contain: a PDF of your project report and your Python code.

TAs: Amruth, David H., Haoyang, Jonas, Tony (in alphabetic order)

Introduction

For this project we will use the MNIST[1] (Mixed National Institute of Standards and Technology) database, which contains binary images of handwritten digits commonly used to train image processing systems. The digits were collected from among Census Bureau employees and high school students. The database contains 60,000 training digits and 10,000 testing digits, all of which have been size-normalized and centered in a fixed-size image of 28×28 pixels. Many methods have been tested with this dataset and in this project, you will get a chance to experiment with the task of classifying these images.

You will first implement multinomial logistic regression (aka softmax regression) with gradient descent and run your model on a test dataset. Next, you will use PCA to reduce high-dimensional vectors of pixels to a low-dimensional vector of features. You will also experiment with the reverse by increasing the dimension using kernels and then apply the regression model again, except to these new features. Lastly, you will apply neural network models to the same task.

Other than your code, you will submit a PDF with relevant plots and discussion. What to put in the PDF for each part is indicated with **"include"** annotations below.

0. Setup

As with the last project, please use Python's NumPy numerical library for handling arrays and array operations; use `matplotlib` for producing figures and plots.

Download `project2.zip` from Stellar and unzip it into a working directory. The zip file contains the image dataset in `mnist.pkl.gz`, several relevant Python files, and a `main.py` file where you will be running your code.

1. Multinomial/Softmax Regression and Gradient Descent [40pts]

In this section, you will implement multinomial regression and the gradient descent algorithm to learn a set of parameters used to classify images as digits from 0-9. You will work with the raw pixel values of each image. That is, each pixel in the original image corresponds to a feature in our feature vector.

To get warmed up to the MNIST data set run `main.py`. This file provides code that reads the data from `mnist.pkl.gz` by calling the function `getMNISTData` that is provided for you in `utils.py`.
The call to `getMNISTData` returns Numpy arrays:

- `trainX` : A matrix of the training data. Each row of `trainX` contains the features of one image, which are simply the raw pixel values flattened out into a vector of length $784 = 28^2$. The pixel values are float values between 0 and 1 (0 stands for black, 1 for white, and various shades of gray in-between).
- `trainY` : The labels for each training datapoint, aka the digit shown in the corresponding image (a number between 0-9).
- `testX` A matrix of the test data, formatted like `trainX`.
- `testY` The labels for the test data, which should *only* be used to evaluate the accuracy of different classifiers in your report.

Next, we call the function `plotImages` to display the first 20 images of the training set. Look at these images and get a feel for the data (don't include these in your write-up).

The main function which you will call to run the code you will implement in this section is `runSoftmaxOnMNIST` in `main.py` (already implemented). Below we describe a number of the methods that are already implemented for you in `softmax_skeleton.py` that will be useful.

1. `augmentFeatureVector`: adds the $x_0^{(i)} = 1$ feature for each data point $x^{(i)}$ for $i = 1, 2, ..., n$. The inputs are:

   - `X`, an $n \times (d-1)$ Numpy array of n data points, each with $d-1$ features.

   The function returns:

   - `X_augment`, an $n \times d$ Numpy array of n data points, each with d features.

   The motivation for doing this is to form a compact representation of $\theta \cdot x + \theta_0$. For example, if θ and x were both $(d-1)$ dimensional vectors, we could define a d dimensional vector $\theta' = [\theta_0, \theta_1, \theta_2, \ldots, \theta_{d-1}]$ and $x' = [1, x_1, x_2, \ldots x_{d-1}]$. Then $\theta \cdot x + \theta_0 = \theta' \cdot x'$.

2. `softmaxRegression`: runs batch gradient descent for a specified number of iterations on a dataset, with θ initialized to the all-zeros array. Here, θ is a $k \times d$ Numpy array, where row j represents the parameters of our model for label j for $j = 0, 1, ..., k-1$. The inputs are:

   - `X`, an $n \times d$ Numpy array of n data points, each with d features.
   - `Y`, an $n \times 1$ Numpy array containing the labels (a number between 0-9) for each data point.
   - α, the learning rate (scalar).
   - λ, the regularization constant (scalar).
   - `k`, the number of labels (scalar).
   - `numIterations`, the number of iterations to run gradient descent (scalar).

   The function returns:

   - θ', a $k \times d$ Numpy array that is the final value of θ.
   - `costFunctionProgression`, a Python list containing the cost calculated at each step of gradient descent.

3. `getClassification`: makes predictions by classifying a given dataset. The inputs are:

   - `X`, an $n \times d$ Numpy array of n data points, each with d features.
   - θ, a $k \times d$ Numpy array, where row j represents the parameters of our model for label j.

   The function returns:

   - $\hat{Y}$, an $n \times 1$ Numpy array, containing the predicted label (a number between 0-9) for each data point.

4. `runSoftmaxOnMNIST` in `main.py`: runs the above `softmaxRegression` on the MNIST training set and computes the test error using the test set. It uses the following values for parameters: $\alpha = 0.3$, $\lambda = 10^{-4}$, and $numIterations = 150$. It also plots the cost function over the number of iterations. Once softmax regression has run, you will get the final model parameters θ. We have called the function `writePickleData` that will save the model parameters to a file called `theta.pkl.gz` in the current directory. Rather than rerunning softmax regression, you can read in your model parameters by calling the function `readPickleData` that is provided in `utils.py`. **This model must be saved with this exact name and will be used during grading.**

Below are the three methods that *you* are responsible for. Each method is described in detail with regards to the inputs and the outputs. We have included some methods in a file called `softmax_verification.py` to help you verify that the methods you have implemented are behaving sensibly.

1. Write a function `computeProbablities` [10pts] that computes, for each data point $x^{(i)}$, the probability that $x^{(i)}$ is labeled as j for $j = 0, 1, ..., k-1$. The inputs are:

   - `X`, an $n \times d$ Numpy array of n data points, each with d features.
   - θ, a $k \times d$ Numpy array, where row j represents the parameters of our model for label j.

   The function returns:

   - `H`, a $k \times n$ Numpy array, where entry (j, i) is the probability that $x^{(i)}$ is labeled as j.

   See the appendix for a hint about dealing with overflow errors. `verifyFirstIterationProbabilities` and `verifySecondIterationProbabilities` are available to help you reason that your code is correct. Note that `verifySecondIterationProbabilities` will not pass until you implement `runGradientDescentIteration`.

2. Write a function `computeCostFunction` [10pts] that computes the total cost over every data point. The inputs are:

   - `X`, an $n \times d$ Numpy array of n data points, each with d features.
   - `Y`, an $n \times 1$ Numpy array containing the labels (a number between 0-9) for each data point.
   - θ, a $k \times d$ Numpy array, where row j represents the parameters of our model for label j.
   - λ, the regularization constant (scalar).

   The function returns:

   - `c`, the cost value (scalar).

   For a reminder on the cost function formula, consult the appendix.

   You may use `verifyFirstIterationCostFunction` and `verifySecondIterationCostFunction` to help you reason that your code is correct. Note that `verifySecondIterationCostFunction` will not pass until you implement `runGradientDescentIteration`.

3. Write a function `runGradientDescentIteration` [10pts] that runs one step of batch gradient descent. The inputs are:

   - `X`, an $n \times d$ Numpy array of n data points, each with d features.
   - `Y`, an $n \times 1$ Numpy array containing the labels (a number between 0-9) for each data point.
   - θ, a $k \times d$ Numpy array, where row j represents the parameters of our model for label j.
   - α, the learning rate (scalar).
   - λ, the regularization constant (scalar).

The function returns:

- θ', a $k \times d$ Numpy array that is the next value of θ after one step of gradient descent.

For a hint on the gradient descent step, consult the appendix.

Finally, in your report **include** the final test error [10pts].

If you have implemented everything correctly, the error on the test set should be around 0.1, which implies the linear softmax regression model is able to recognize MNIST digits with around 90% accuracy.

Important: To make sure you receive credit for this section, run the file `checker.py` which will try to check that your functions work with the specified input and return the specified output. If it errors that means there is an issue in your code. If it does not error it will say whether the simple tests in place to check types pass or fail.

2. Using manually crafted features [40pts]

The performance of most learning algorithms depends heavily on the representation of the training data. In this section, we will try representing each image using different features in place of the raw pixel values. Subsequently, we will investigate how well our regression model from the previous section performs when fed different representations of the data.

Dimensionality Reduction via PCA. Principal Components Analysis[2] (PCA) is the most popular method for linear dimension reduction of data and is widely used in data analysis. Briefly, this method finds (orthogonal) directions of maximal variation in the data. By projecting an $n \times d$ dataset X onto $k \ll d$ of these directions, we get a new dataset of lower dimension that reflects more variation in the original data than any other k-dimensional linear projection of X. By going through some linear algebra, it can be proven that these directions are equal to the k eigenvectors corresponding to the k largest eigenvalues of the covariance matrix $\widetilde{X}^T\widetilde{X}$, where $\widetilde{X}$ is a centered version of our original data.

Remark: The best implementations of PCA actually use the Singular Value Decomposition of $\widetilde{X}$ rather than the more straightforward approach outlined here, but these concepts are beyond the scope of this course.

Here are some functions that we have provided for you to use in this part (all located in `features.py`):

`centerData` centers the data (by subtracting off the mean of each feature).
The input is: X, an $n \times d$ Numpy array of n data points, each with d features.
The function returns: $n \times d$ Numpy array $\widetilde{X}$, where for each $i = 1, \ldots, n$ and $j = 1, \ldots, d$:

$$\widetilde{X}_{i,j} = X_{i,j} - \mu_j \quad \text{with } \mu_j = \frac{1}{n}\sum_{i=1}^{n} X_{i,j}$$

`principalComponents` computes the principal component directions of an input data matrix X ($n \times d$ Numpy array). This function first calculates the covariance matrix, $\widetilde{X}^T\widetilde{X}$ and then find its eigenvectors. The function returns: $d \times d$ matrix (Numpy array) whose columns are the principal component directions sorted in descending order by the amount of variation each direction (these are equivalent to the d eigenvectors of $\widetilde{X}^T\widetilde{X}$ sorted in descending order of eigenvalues, so the first column corresponds the eigenvector with largest eigenvalue).

`plotPC` produces a scatter-plot of data after it has first been projected down to its 2-D representation in the first 2 principal components.

`reconstructPC` tries to reconstruct an original data sample from its lower-dimensional PCA-representation. Note that to reconstruct an observation from its representation in the top k principal components, $x_{\text{pca}} \in \mathbb{R}^{1\times k}$ we can use matrix algebra: $x_{\text{pca}} \cdot V^T + \mu$, where V is the $d \times k$ matrix whose columns are the top k eigenvectors of $\widetilde{X}^T\widetilde{X}$, and μ is a $1 \times d$ vector containing the mean of each feature (see the definition of its jth element μ_j above).

Below are the steps you are responsible for in this section:

1. Fill in function `projectOntoPC` in `features.py` that implements PCA dimensionality reduction of dataset X [10 pts]. Your input should be:

[2] In-depth exposition: `www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`

   - `X`, a $n \times d$ Numpy array of n data points, each with d features.
   - `pcs`, a $d \times d$ matrix whose columns are the d eigenvectors of $\widetilde{X}^T\widetilde{X}$, ordered in *descending* order of eigenvalues (i.e. the output of `principalComponents`).
   - `n_components`, the number of principal components to use in the PCA representation.

   Your function should return a $n \times$ `n_components` Numpy array, whose rows represent low-dimensional feature vectors corresponding to the projection of dataset X onto its first `n_components` principal components (we refer to this as the PCA representation).
   Note that to project a given $n \times d$ dataset X into its k-dimensional PCA representation, one can use matrix multiplication (after first centering X): $\widetilde{X}V$, where V is the $d \times k$ matrix whose columns are the top k eigenvectors of $\widetilde{X}^T\widetilde{X}$. This is because the eigenvectors are of unit-norm, so there is no need to divide by their length.

2. Use `projectOntoPC` to compute a 20-dimensional PCA representation of the MNIST training and test datasets, as illustrated in `main.py`.

   Retrain your softmax regression model (using the same settings the previous section) on the MNIST training dataset and report its error on the test data, this time using these 20-dimensional PCA-representations rather than the raw pixel values (**Include** the error in your report [5 pts]).

   If your PCA implementation is correct, the model should perform nearly as well when only given 20 numbers encoding each image as compared to the 784 in the original data (error on the test set using PCA features should be around 0.14). This is because PCA ensures these 20 feature values capture the maximal amount of variation in the original 784-dimensional data.

   Remark: Note that we only use the training dataset to determine the principal components. It is *improper* to use the test dataset for anything except evaluating the accuracy of our predictive model. If the test data is used for other purposes such as selecting good features, it is possible to overfit the test set and obtain overconfident estimates of a model's performance.

3. Use the call to `plotPC` in `main.py` to visualize the first 100 MNIST images, as represented in the space spanned by the first 2 principal components of the training data.
Include this plot in your report [5 pts].

   Remark: Two dimensional PCA plots offer a nice way to visualize some global structure in high-dimensional data, although approaches based on nonlinear dimension reduction may be more insightful in certain cases. Notice that for our data, the first 2 principal components are insufficent for fully separating the different classes of MNIST digits.

4. Use the calls to `plotImages()` and `reconstructPC` in `main.py` to plot the reconstructions of the first two MNIST images (from their 20-dimensional PCA-representations) alongside the originals.
Include these plots in your report [5 pts].

Quadratic Features.

In this section, we will work with a *quadratic feature* mapping which maps an input vector $x = [x_1, \ldots, x_d]$ into a new feature vector $\phi(x)$, defined so that for any $x, x' \in \mathbb{R}^d$:

$$\phi(x)^T \phi(x') = (x^T x' + 1)^2$$

5. In 2-D, let $x = [x_1, x_2]$. Write down the explicit quadratic feature mapping $\phi(x)$ as a vector (**include** in your report).
Hint: $\phi(x)$ should be a 6-dimensional vector.

   Now, fill in function `quadraticFeatures` in `features.py`. Given an input dataset (with d-dimensional features where d is not necessarily equal to 2), this function returns a new dataset where each observation is now represented using quadratic features [10 pts].

6. If we explicitly apply the quadratic feature mapping to the original 784-dimensional raw pixel features, the resulting representation would be of massive dimensionality. Instead, we will apply the quadratic feature mapping to the 20-dimensional PCA representation of our training data which we computed previously. After applying the quadratic feature mapping to the PCA representations for both the train and test datasets, retrain the softmax regression model using these new features and report the resulting test set error (**Include** this error in your report [5 pts]).

   If you have done everything correctly, softmax regression should perform better (on the test set) using these features than either the principal components or raw pixels. The error on the test set using quadratic features should only be around 0.04, demonstrating the power of nonlinear classification models.

Remark: Note that the feature mapping we have been working with actually corresponds to the quadratic kernel, which is defined as: $K(x, x') = (x^T x' + 1)^2$. Recall, that the feature mapping ϕ associated with a kernel K satisfies $K(x, x') = \phi(x)^T \phi(x')$ for any $x, x' \in \mathbb{R}^d$.

In practice, this kernel function is all that is required to perform nonlinear kernel regression and one never has to actually compute the feature mapping (recall the kernel trick). Nevertheless, we have explicitly worked with the feature mapping corresponding to this kernel for educational purposes. Note that working with the explicit representations induced by the feature mapping can vastly increase in the dimensionality of the problem. This is why the kernel trick is a key component of kernel methods, allowing them to run efficiently in practice!

3. Classification using deep neural networks [20pts]

In this section, we are going to use deep neural networks to perform the same classification task as in previous sections. In particular, we will use Keras, a python library built upon the deep learning framework Theano. **Refer to the Appendix for instructions on setting up the necessary deep learning environment**.

1. **Fully-Connected Neural Networks**

   First, we will employ the most basic form of a deep neural network, in which the neurons in adjacent layers are fully connected to one another.

   (a) We have provided a toy example `mnist_nnet_fc.py` in which we have implemented for you a simple neural network. This network has one hidden layer of 128 neurons with a rectified linear unit (ReLU) nonlinearity, as well as an output layer of 10 neurons (one for each digit class). Finally, a softmax function normalizes the activations of the output neurons so that they specify a probability distribution. Reference the Keras website[3] and read through the documentation in order to gain a better understanding of the code. Then, try running the code on your computer with the command `python mnist_nnet_fc.py`. This will train the network with 10 epochs, where an epoch is a complete pass through the training dataset. Total training time of your network should take no more than a couple of minutes; if you find the training to be going slowly on your machine, try using an Athena[4] cluster machine. At the end of training, your model should have an accuracy around 92 to 93% on the test set (**test accuracy**).

   (b) Modify your model to reach over 98% **test accuracy** after 10 epochs of training. Things you can, but don't have to, tweak include the number of fully-connected layers, the number of neurons in each layer, or the parameters of the stochastic gradient descent (SGD) optimizer (such as the learning rate).
   Include a description of your final model architecture in your report, and give its accuracy on the test data. What did you try that worked, and what did you try that didn't work? [10 pts]

2. **Convolutional neural networks**

Next, we are going to apply convolutional neural networks to the same task. These networks have demonstrated great performance on many deep learning tasks, especially in computer vision.

(a) We provide a skeleton code `mnist_nnet_cnn_skeleton.py` which includes examples of some (**not all**) of the new layers you will need in this part. Using the Keras documentation website[5], complete the parts of the code called "Design the model here" and "Import the layer types needed" to implement a convolutional neural network with following layers in order:

- A convolutional layer with 32 filters of size 3×3
- A ReLU nonlinearity
- A max pooling layer with size 2×2
- A convolutional layer with 64 filters of size 3×3
- A ReLU nonlinearity
- A max pooling layer with size 2×2
- A flatten layer
- A fully connected layer with 128 neurons
- A dropout layer with drop probability 0.5
- A fully-connected layer with 10 neurons
- A softmax layer

Without GPU acceleration, you will likely find that this network takes quite a long time to train. For that reason, we don't expect you to actually train this network until convergence. Implementing the layers and verifying that you get approximately 93% **training accuracy** and 98% **validation accuracy** after one training epoch (this should take less than 10 minutes) is enough for this project. If you are curious, you can let the model train for a few hours; if implemented correctly, your model should achieve >99% **test accuracy** after 10 epochs of training. If you have access to a CUDA compatible GPU, you could even try configuring Keras to use your GPU. Include the code for your CNN in your submission. [10 pts]

(b) (**Optional**) To understand why convolutional neural networks work so well, we often need to visualize the intermediate filters to see what's going on. blog[6] and try visualizing the features learned in each layer of your CNN network.

REMEMBER Submit a .zip file containing your source code and your report in PDF format to Stellar.

4. Appendix: some background and details

Dealing with overflow errors when computing H

Computing h of a particular $x^{(i)}$ requires computing

$$h(x^{(i)}) = \frac{1}{\sum_{j=1}^{k} e^{\theta_j \cdot x^{(i)}}} \begin{bmatrix} e^{\theta_1 \cdot x^{(i)}} \\ e^{\theta_2 \cdot x^{(i)}} \\ \vdots \\ e^{\theta_k \cdot x^{(i)}} \end{bmatrix}$$

While the probabilities themselves are in the range [0,1], $e^{\theta_j \cdot x^{(i)}}$ is not. To deal with this we can simply subtract some fixed amount c from each exponent to keep the resulting number from getting too large. See that

$$\begin{aligned} h(x^{(i)}) &= \frac{1}{\sum_{j=1}^{k} e^{\theta_j \cdot x^{(i)}}} \begin{bmatrix} e^{\theta_1 \cdot x^{(i)}} \\ e^{\theta_2 \cdot x^{(i)}} \\ \vdots \\ e^{\theta_k \cdot x^{(i)}} \end{bmatrix} \\ &= \frac{e^{-c}}{e^{-c} \sum_{j=1}^{k} e^{\theta_j \cdot x^{(i)}}} \begin{bmatrix} e^{\theta_1 \cdot x^{(i)}} \\ e^{\theta_2 \cdot x^{(i)}} \\ \vdots \\ e^{\theta_k \cdot x^{(i)}} \end{bmatrix} \\ &= \frac{1}{\sum_{j=1}^{k} e^{\theta_j \cdot x^{(i)} - c}} \begin{bmatrix} e^{\theta_1 \cdot x^{(i)} - c} \\ e^{\theta_2 \cdot x^{(i)} - c} \\ \vdots \\ e^{\theta_k \cdot x^{(i)} - c} \end{bmatrix} \end{aligned}$$

Thus subtracting some fixed amount from each exponent will not change the final probabilities. A good choice for this fixed amount is $c = \max_j \theta_j \cdot x^{(i)}$.

Softmax Cost Function

The cost function $J(\theta)$ is given by:

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^{m} \sum_{j=1}^{k} [[y^{(i)} == j]] \log \frac{e^{\theta_j \cdot x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l \cdot x^{(i)}}} \right] + \frac{\lambda}{2} \sum_{i=1}^{k} \sum_{j=0}^{d-1} \theta_{ij}^2$$

Softmax Gradient

The derivative of $J(\theta)$ wrt a particular θ_j is given by:

$$\nabla_{\Theta_j} J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [x^{(i)}([[y^{(i)} == j]] - p(y^{(i)} = j | x^{(i)}, \theta))] + \lambda \theta_j$$

Environment setup for deep learning

We provide two ways to set up the computing environment for Keras. You are free to choose either one, but we strongly recommend solution 1 as it automatically takes care of version dependency across different packages. This tutorial only works for Mac and Linux users. **Windows users please use MIT Athena**[7] . We have successfully tested the miniconda environment and DNN code on the Athena machines, so if all else fails (or if you find the DNN code to run too slowly on your machine) consider using an Athena machine.

Solution 1

- Download miniconda for **Python 2.7** from here[8]
- Follow the instructions here [9] to install miniconda.
  - When asked to modify the environment config file in your home directory (∼/.bash_profile for Mac or ∼/.bashrc for Linux), **please approve**.
  - After installation, close the terminal and reopen it. Then type 'which python' to make sure the path has 'miniconda' in it.
- We provided a conda enviroment file called `environment.yml` . In the same folder as this file, run `conda env create -f environment.yml` and then `source activate ml6036` in terminal. You will see a 'ml6036' label in front of every line in your terminal now.
- Run `pip install git+git://github.com/Theano/Theano.git` in the terminal
- Additional step for Mac users: please first run `mdfind -name libmkl_intel_lp64.dylib`. Copy the output on the screen. Then run `export DYLD_FALLBACK_LIBRARY_PATH=YOUR_PATH` where `YOUR_PATH` is the output of the previous command
- Things to note:
  - **Every time you close and reopen the terminal, you will have to redo** `source activate ml6036` **to enter this environment again.**
  - At any point, you can switch back to the default python by simply removing the line 'export PATH=/yourhomepath/miniconda2/bin:$PATH' from your environment config file (see the second bulletin point above) and restart your terminal. When you feel you want to switch back to the conda python environment which enables you to run the code in this part, you will then need to first add back this line in your environment config file, restart your terminal, and run `source activate ml6036` in terminal.

Solution 2

Run the following commands to install all the necessary packages. Note that this solution works for Mac/Linux only and is more prone to trouble. If you encounter a package version conflict and don't know how to solve it, we suggest you choose solution 1.

```
sudo pip install numpy
sudo pip install scipy
sudo pip install yaml
sudo pip install cython
sudo pip install h5py
sudo pip install git+git://github.com/Theano/Theano.git
sudo pip install keras
```

Image features

Many other feature representations of images have been developed besides the approaches considered here. Some examples of widely used image features include HOG, SIFT, and GIST. All of them are variations on the same theme: abstract away from raw pixels, and form more general representations of image content. This can be done manually by histogramming and averaging measurements in different image regions. These measurements are often pixel intensities (grayscale or color) or gradients (edges). A histogram over an image region just specifies how much of each measurement there is in a region (e.g. how many horizontal edges, vertical edges, etc.) without specifying the exact pixel location. This allows for a more general representation that will still match images that have similar distributions of features but are not identical at the pixel level.

Recently, deep learning approaches have outperformed complex manually-engineered features like HOG/SIFT in large image datasets. The original appeal of neural networks was a general purpose model that could learn useful features directly from raw pixels. However, in the past few years, the state of the art for computer vision tasks like object recognition has been significantly improved by spending effort to develop complex CNN architectures[10] in place of feature-engineering.

p d Stanford, 2007f, ANg, HW4, pr. 3 (implement! PCA and ICA (with advice)!!! and apply them on natural images)

3. PCA and ICA for Natural Images

In this problem we'll apply Principal Component Analysis and Independent Component Analysis to images patches collected from "natural" image scenes (pictures of leaves, grass, etc). This is one of the classical applications of the ICA algorithm, and sparked a great deal of interest in the algorithm; it was observed that the bases recovered by ICA closely resemble image filters present in the first layer of the visual cortex.

The `q3/` directory contains the data and several useful pieces of code for this problem. The raw images are stored in the *images/* subdirectory, though you will not need to work with these directly, since we provide code for loading and normalizing the images.

Calling the function `[X_ica, X_pca] = load_images;` will load the images, break them into 16x16 images patches, and place all these patches into the columns of the matrices `X_ica` and `X_pca`. We create two different data sets for PCA and ICA because the algorithms require slightly different methods of preprocessing the data.[1]

For this problem you'll implement the `ica.m` and `pca.m` functions, using the PCA and ICA algorithms described in the class notes. While the PCA implementation should be straightforward, getting a good implementation of ICA can be a bit trickier. Here is some general advice to getting a good implementation on this data set:

[1] Recall that the first step of performing PCA is to subtract the mean and normalize the variance of the features. For the image data we're using, the preprocessing step for the ICA algorithm is slightly different, though the precise mechanism and justification is not imporant for the sake of this problem. Those who are curious about the details should read Bell and Sejnowki's paper "The 'Independent Components' of Natural Scenes are Edge Filters," which provided the basis for the implementation we use in this problem.

- Picking a good learning rate is important. In our experiments we used $\alpha = 0.0005$ on this data set.
- Batch gradient descent doesn't work well for ICA (this has to do with the fact that ICA objective function is not concave), but the pure stochastic gradient described in the notes can be slow (There are about 20,000 16x16 images patches in the data set, so one pass over the data using the stochastic gradient rule described in the notes requires inverting the 256x256 W matrix 20,000 times). Instead, a good compromise is to use a hybrid stochastic/batch gradient descent where we calculate the gradient with respect to several examples at a time (100 worked well for us), and use this to update W. Our implementation makes 10 total passes over the entire data set.
- It is a good idea to randomize the order of the examples presented to stochastic gradient descent before each pass over the data.
- Vectorize your Matlab code as much as possible. For general examples of how to do this, look at the Matlab review session.

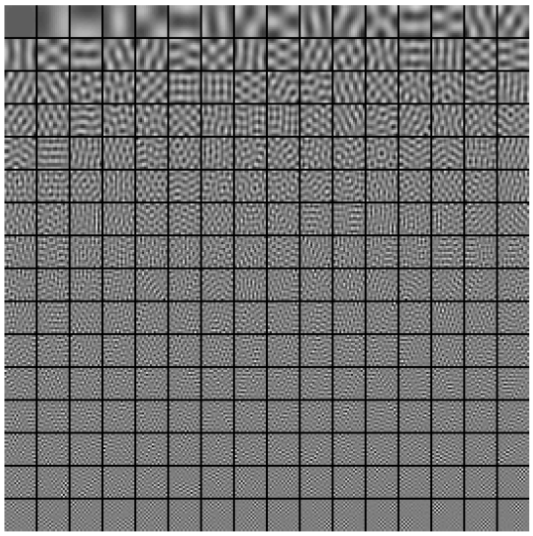
For reference, computing the ICA W matrix for the entire set of image patches takes about 5 minutes on a 1.6 Ghz laptop using our implementation.

After you've learned the U matrix for PCA (the columns of U should contain the principal components of the data) and the W matrix of ICA, you can plot the basis functions using the `plot_ica_bases(W);` and `plot_pca_bases(U);` functions we have provide. Comment briefly on the difference between the two sets of basis functions.

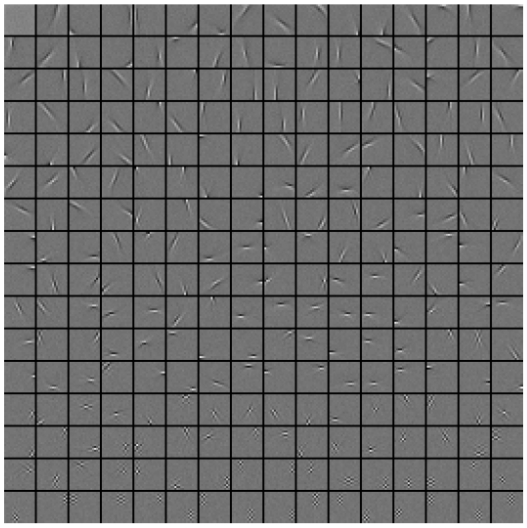
Answer: The following are our implementations of `pca.m` and `ica.m`:

.....

PCA produces the following bases:



while ICA produces the following bases



The PCA bases capture global features of the images, while the ICA bases capture more local features.

CMU, 2007f, HW5, pr. 3 (Clustering and dim. reduction: PCA + GMM(EM) on digits)

3 [30 points] Clustering and Dimensionality Reduction

In this problem we will again be working with the digits data set. The data file we provide contains $60,000$ hand written digits between 0 and 9. Each digit is a 28×28 grayscale image represented as a 784 dimensional vector. The variable X is a $60,000 \times 784$ matrix. The $60,000$ dimensional vector Y contains the true number for each image. Please submit include in your write-up a copy of all plots for this problem.

Update: There is now a smaller data set availabe. In your results please say what data set you use and use only one data set throughout the problem. The smaller data set consists of 5000 10×10pixel images each represented as 100 dimensional row vector. When asked to use the first 1, 2, 5, 10, 21, 44, 94, 200, and 784 principal components instead use the first 1, 2, 3, 6, 10, 18, 32, 56, and 100.

3.1 [15 Points] Principal Components Analysis

A very common technique for dimensionality reduction is principal components analysis typically referred to as PCA. In the first part of this problem you will need to implement PCA. Because this is a relatively large data set, consider using the functions `cov` and `eig` or `eigs`.

1. [5 Points] Plot the first 9 principal components as images. You will probably need the functions `image`, `colormap('Gray')`, `subplot`, and `reshape(v,28,28)`. To plot the principal components rescale the vector so that its values range between 0 and 255.
2. [5 Points] Plot the eigenvalues in decreasing order. From the plot, how many eigenvectors do you believe are necessary to approximately represent the images.
3. [5 Points] Using the first 1, 2, 5, 10, 21, 44, 94, 200, and 784 principal components plot the reconstruction of the the first 2 digits in the data set. Use `subplot(3,3,i)` to save tree of the natural kind. Does the approximation get better with increasing principal components?

3.2 [15 Points] Gaussian Mixture Model

For this problem you will need to implement Expectation Maximization (EM) for the axis aligned Gaussian mixture model. Recall that the axis aligned gaussian mixture model uses the Gaussian Niave bayes assumption that given the class all the features are conditionally independent Gaussians. The specific form of the model is given in Equation 1.

Update: Due to numerical issues on this data set, you may use K-means clustering instead of a Gaussian mixture model. If you have a clever solution to the numerical problems please describe the solution in your write-up for extra credit.

$$
\begin{aligned}
Z_i &\sim Multinomial(p_1, \ldots, p_K) \\
X_i | Z_i = z &\sim N\left(\begin{bmatrix} \mu_1^{(z)} \\ \mu_2^{(z)} \\ \vdots \\ \mu_d^{(z)} \end{bmatrix}, \begin{bmatrix} (\sigma_1^{(z)})^2 & 0 & \cdots & 0 \\ 0 & (\sigma_2^{(z)})^2 & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & (\sigma_d^{(z)})^2 \end{bmatrix} \right)
\end{aligned} \tag{1}
$$

If you find that running your algorithm on the full data set takes too long try using the first $5,000$ digits instead of all $60,000$. You will want to avoid looping over all $n = 60,000$ examples.

1. [5 Points] Run EM to fit a Gaussian mixture model with 16 gaussians on the digits data without applying PCA. Plot each of the means using `subplot(4,4,i)` to save paper.

2. [5 Points] Again run EM clustering with $K = 16$ using only the first 100 principal components. Again, plot each of the means. Do you notice a difference in the output or running time?

3. [5 Points] Evaluating clustering performance is difficult. However, because we have "truth" data, we can roughly assess clustering performance. One possible metric is to label each cluster with the majority label for that cluster using the truth data. Then, for each point we predict the cluster label and measure the mean 0/1 loss. For the digits data set, what score would we get if we placed all the points in one class? Using 1, 2, 5, 10, 21, 44, 94, 200, and 784 principal components what scores does $k = 16$ means clustering obtain?

SVD (1) - practice

d (P:1,2)CMU, 2009f, CGuestrin, HW5, pr. 2 (Randomized SVD:

1. prog: svd approx of a matrix; not optimal

2. prog + idea: eigenvalues + svd

3. theory: projections

4. theory: eig + frob norm ineq)

2 Randomized Singular Value Decomposition [Babis 50 points]

Background

Singular Value Decomposition[2] is a powerful matrix decomposition with many applications: HITS algorithm by J. Kleinberg, Latent Semantic Indexing (Deerwester et al.; Papadimitriou et al), linear dimensionality reduction are some prominent applications of SVD. It is also widely called as "the Swiss Army knife of matrix computation" or even the "sledgehammer of linear algebra".

In a real world scenario one is interested in a low rank approximation of the matrix. Specifically, consider a matrix $A \in \mathbb{R}^{n \times m}$. According to the SVD theorem, A can be expressed as:

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T \tag{1}$$

where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r$ are the singular values, $u_i \in \mathbb{R}^n$ for $i = 1, \ldots, r$ are the left singular vectors and $v_i \in \mathbb{R}^m$ for $i = 1, \ldots, r$ are the right singular vectors and r is the rank of the matrix.

Let A_k be the k rank approximation of A:

$$A = \sum_{i=1}^{k} \sigma_i u_i v_i^T \tag{2}$$

Typically, $k << r$. Matrix A_k, the k-rank approximation of A is optimal with respect to the 2 norm and the Frobenius norm. For example: $||A - A_k||_F \leq ||A - C||_F$ for any matrix C of rank at most k (including k).

In this exercise you will study some fundamental properties of the SVD and implement a practical algorithm which comes with guarantees on its quality.

2.1 Programming Part

In many cases we are willing to compute a sub-optimal k rank approximation, $\hat{A}_k$ if we have important computational savings. You will implement an algorithm that finds $\hat{A}_k$ instead of A_k and comes with guarantees on the quality of the approximation as already mentioned.

The function you will write should have the following input and output arguments:

- Input
  1. Matrix $A \in \mathbb{R}^{n \times m}$
  2. Integer $s \leq n$.
  3. Integer $k \leq s$.
- Output
  1. Matrix $H \in \mathbb{R}^{m \times k}$
  2. $\lambda_1, \ldots, \lambda_k \in \mathbb{R}^+$.

Matrix H will contain the approximation to the top-k right singular vectors and λ_1, λ_k is the approximation to the singular values.

1 [25 points] Implement the following algorithm:

1. For $i = 1$ to n compute $p_i = ||A(i,:)||^2/||A||_F^2$.
2. For $i = 1$ to s
   - Pick an integer j from 1 to n with probability $Pr(\text{pick j}) = p_j$.
   - Include $A(j,:)$ as a row of S.
3. Compute SS^T and its singular value decomposition, i.e., $SS^T = \sum_{t=1}^{s} \lambda_t^2 w_t w_t^T$.
4. Compute $h_t = \frac{S^T w_t}{||S^T w_t||}$ for $t = 1 \ldots k$.
5. Return matrix H whose columns are the vectors $h_1, \ldots, h_k$ and $\lambda_1 \geq \ldots \geq \lambda_k$.

You should hand in your code printed since it should not be more than one page.

2 [10 points] Apply your implementation of the baboon image and compare to the optimal 60-rank approximation. In other words your k equals 60.

Report the following:

1. **[5 points]** Plot the two reconstructed images. The following commands may prove useful: *image(A), olormap(gray(256))*. To read the input use the following command:

   ```
   A = imread('baboon.tif'); A = double(A);
   ```

   Hint: Think of how you should use matrix H.

2. **[5 points]** Report the error in terms of the Frobenius norm for both the optimal 60 rank produced from the SVD and for the 60 rank approximation produced by your implementation.

3 [2 points] Explain how you can adapt your algorithm to approximate the left singular vectors instead of the right singular vectors.

2.2 Eigenvalues and SVD [5 points]

Create a random graph on 100 nodes with probability of 0.5 of having an edge between nodes i and j, for all possible i, j. The following piece of code can do this for you:

```
A = rand(100) <=0.5;
A = triu(A,1);
A = A + A';
```

Compute the eigendecomposition using command *eig* and its singular value decomposition using command *svd*. What do you observe on the values of the eigenvalues and the singular values? How can you recover the signs of the eigenvalues through the SVD?

.....

4. Write a function that given the data `Z` and the parameters θ, μ, and σ predicts the probability that $Y = 1$ (i.e., the face is hot). Provide the code in your solution.

.....

5. We now will train a Gaussian Naive Bayes classifier to predict whether a face is hot using the low dimensional feature space. To determine the best value for `dim` we will use cross validation on `tr`. *Do not* use the `te` data. For each `dim` from 1 to 100 do 100 random splits of `tr` into $\frac{4}{5}$ training and $\frac{1}{5}$ test. For efficiency, you may use the original eigenfaces and xbar computed from the earlier sections. You may want to use the following fragment of code:

.....

2.3 Projections (again)[4+4 points]

Suppose you are given a set of points $\{x_1, x_2, \ldots, x_n\}$ where $x_i \in \mathbb{R}^m$. You create a matrix $A \in \mathbb{R}^{n \times m}$, where the i-th row of matrix A corresponds to point x_i.

You rcompute the k rank approximation of A, i.e., the left singular vectors $u_1, \ldots, u_k$, the right singular vectors $v_1, \ldots, v_k$ and the singular values $\sigma_1, \ldots, \sigma_k$. Explain how you will use the above to do the followings tasks when you are given a new point y.

1. Project y on the space spanned by the left singular vectors $u_1, \ldots, u_k$, $k < r$, where r is the rank of the matrix A.

2. Project y on the space spanned by the left singular vectors $u_{k+1}, \ldots, u_r$, $k < r$, where r is the rank of the matrix A. Observe that your solution however must use only the vectors $u_1, \ldots, u_k$. In other words, even if you want to project y on the space spanned by $u_{k+1}, \ldots, u_r$ you can only use $u_1, \ldots, u_k$.

   Hint: Use the orthogonality properties of the singular vectors.

2.4 A favorite inequality for Extra Credit [7 points]

Let A, B be $n \times n$ symmetric matrices. Let λ_i, μ_i be the i-the eigenvalue of A, B respectively, both ordered in increasing order, $i = 1, \ldots, n$. Prove the following inequality:

$$\sum_{i=1}^{n} (\lambda_i - \mu_i)^2 \leq ||A - B||_F^2 \tag{3}$$

where the Frobenius norm of a matrix is given by the following equation: $||A||_F^2 = \sum_i \sum_j a_{ij}^2$
Remark: Only complete answers will receive credit.

d (P)CMU, 2009f, GGordon, HW4, pr. 2 (eigenfaces - 3 preproc steps:

1. basic analysis

2. svd

3. dim. reduction

[4. Naive Bayes])

2 Hot Eigenfaces for Gaussian Naive Bayes [65 Points]

In this problem you will have the opportunity to apply PCA to a moderately sized collection of images scraped from the *Hot-or-Not*[1] website and then use a Naive Bayes classifier to learn to classify facial attractiveness. For this problem we have selected only female faces because they are more consistently labeled and are typically more amenable to basic classifiers. The data for this problem is available at http://www.cs.cmu.edu/~ggordon/10601/hws/hw4/faces.mat. The *faces.mat* Matlab file contains the following data:

h: The height of each image.

w: The width of each image.

tr and te: These structs contain the training and test data respectively. The fields in these structs are:

n: The number of images.

hot: A vector with `n` elements. The entry `hot(i)` is 1 if the i^{th} image is hot and 0 otherwise.

images: An `n` by `w * h` matrix where each row corresponds to a separate image. To view the j^{th} image in matlab you would use the following code:

```
load('faces.mat');
figure(1);
j = 3;
% Reshape the row vector into an 86 by 86 pixel image
img = reshape(tr.images(j, :), h, w);
% Show the image
imagesc(img);
% Set the plot to gray-scale
colormap('gray');
```

★ **SOLUTION:** The solution code for this problem can be downloaded from: http://www.cs.cmu.edu/~ggordon/10601/hws/hw4/hw4_code.tar.gz

Use only the data in the `tr` struct for all training and validation. The data in `te` will only be used in the last part to evaluate your classifier. We will explicitly say when to use `te`. Don't use `te` until we tell you.

2.1 Basic Analysis [15 Points]

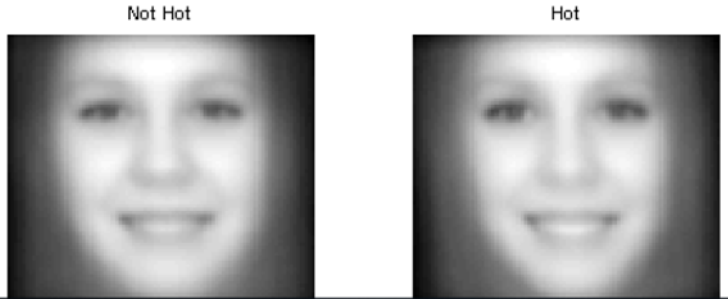
We will begin with some basic data analysis to become more familiar with the data.

1. What fraction of the faces are hot?

   ★ **SOLUTION:** 0.499

2. Using `subplot(1,2,i)` plot both the average hot and not hot faces side by side. To compute the average face simply use the `mean` function on the respective class of face vectors (i.e., `mean(tr.images(tr.hot,:))`).

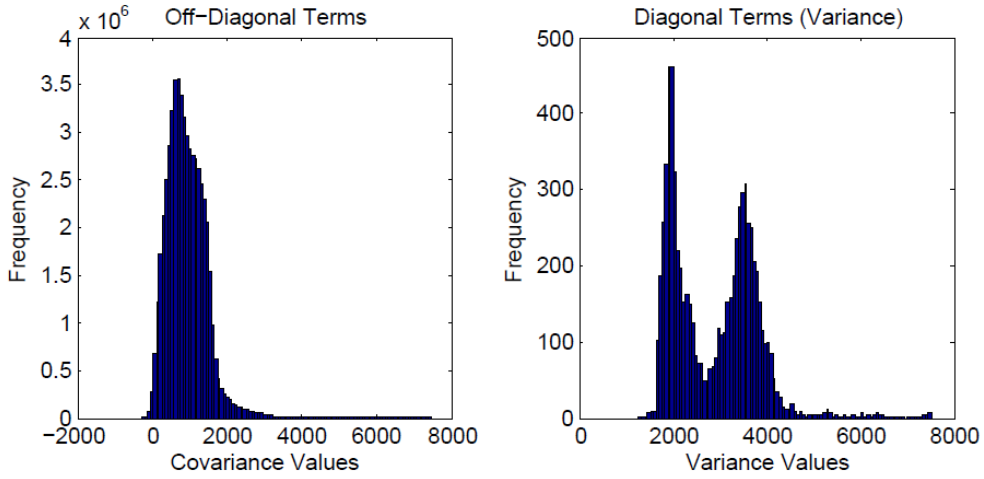
   ★ **SOLUTION:**



3. Based on the average faces, does it seem like their may be sufficient difference to build a classifier for facial attractiveness? Provide a *very* brief explanation.

   ★ **SOLUTION:** Yes. There are visually distinguishable differences between the average hot and average not hot face.

4. Compute the covariance between the pixels (i.e., `cov(tr.images)`). Plot the distribution of the values for the off-diagonal terms. Plot the distribution of the diagonal (variance) terms. (That is, make histograms of the given sets of numbers.) Solution



We can see that the off diagonal terms are nonzero and have magnitude similar to the diagonal terms. This indicates strongly correlated features.

5. Based on the above observations, what can you say about the dependence/independence of pixels in these images? Give a *brief* justification why neighboring pixels might be correlated.

★ **SOLUTION:** Pixel values are not independent. Intuitively, neighboring pixels are likely to share similar values since face images are generally smooth. Furthermore, the two bumps in the histogram are likely be due to strongly correlated regions like the hair, cheeks and background.

2.2 Singular Value Decomposition [15 Points]

For this question you will need to use the matlab `svds` function to compute the top 100 eigenvectors. Please attach the code you write for this section to your solution set.

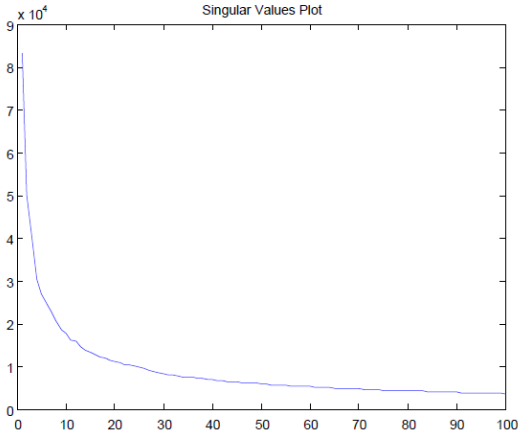
1. Compute the average face `xbar` using the `mean` function. You will need `xbar` throughout the rest of this problem. Center the faces data by subtracting `xbar` from each row of `tr.images` storing the result in `Xcenter`. Run `svds` on `Xcenter` (i.e., `[U, svalues, efaces] = svds(Xcenter, 100);`). The columns of `efaces` are the eigenfaces and the diagonal entries of `svalues` are the singular values. Please include the code you used for the past few steps.

★ **SOLUTION:**

.....

2. Plot the singular values (i.e., `plot(diag(svalues))`).

★ **SOLUTION:** Notice how the singular values decrease quickly.



3. Do the singular values decrease rapidly?

   ★ **SOLUTION:** Yes. (See above plot.)

4. What does this suggest about the approximation quality obtained by representing faces as a linear combination of the top few eigenfaces?

   ★ **SOLUTION:** Because the singular values decrease rapidly the top few eigenfaces will likely provide a reasonable approximation.

5. Plot the top 10 eigenfaces (preferably using `subplot(2,5,i)` to save paper).

   ★ **SOLUTION:**

6. For two of the eigenfaces provide a *brief* interpretation of what aspect of the images they might encode (i.e., "lighting", "eyes", ...).

   ★ **SOLUTION:** Along the first row from left to right, face 1 likely encodes variations in eye shape and face width. Face 2 encodes variation in face width. Face 3 encodes lighting from the left. Face 4 encodes deviations in lip and eyelashes. Face 5 encodes hair and teeth deviations.

2.3 Dimensionality Reduction [15 Points]

We will now use the eigenfaces computed in the previous part to do dimensionality reduction. We will see how we can transform 7396 dimensional vectors into 100 dimensional vectors while preserving most of the original information. You should not call `svds` again (this will be very slow); instead, use your existing eigenfaces from the previous question.

1. Create a function to project a collection of faces onto a fixed number (`dim`) of eigenfaces. To project the matrix `X` of images (as row vectors) into the low dimensional linear space spanned by the eigenfaces use the following:

   ```
   %>  Z = (X - repmat(xbar, n, 1)) * efaces(1:dim,:)';
   ```

   where `efaces` are the eigenfaces as row vectors. The `n` by `dim` matrix `Z` contains the low dimensional representation of each image. Provide the code for this function.

.....

2. Create a function to map points in the low dimensional space back to the original space. You may want to use the following piece of code:

```
%>   Xapprox = (Z * efaces(1:dim, :)) + repmat(xbar, n, 1);
```

Provide the code for this function.

.....

3. Project the first 10 faces onto the first 10, 50, and 100 eigenfaces and then compute `Xapprox`. Using a 4 row by 10 column grid (use `\subplot(4,10,i)`), plot:

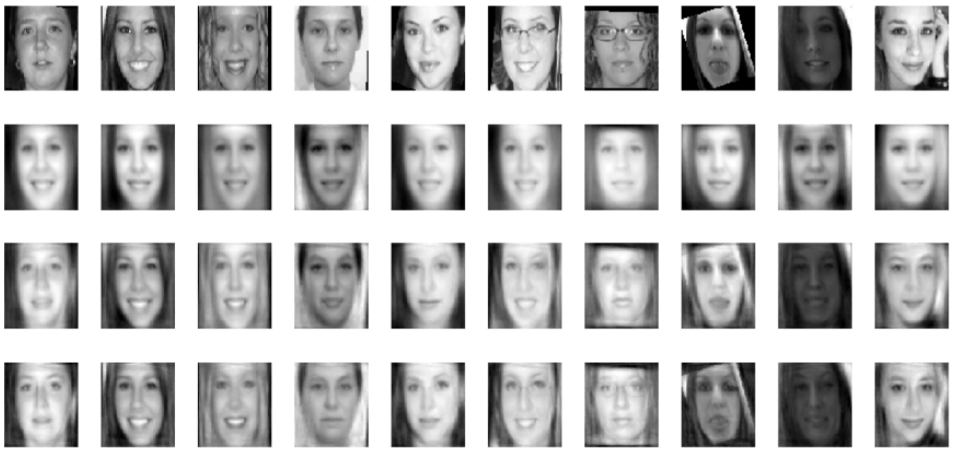
Row 1: the original images.

Row 2: the `Xapprox` approximation using the first `dim=100` eigenfaces.

Row 3: the `Xapprox` approximation using the first `dim=50` eigenfaces.

Row 4: the `Xapprox` approximation using the first `dim=10` eigenfaces.

★ **SOLUTION:**

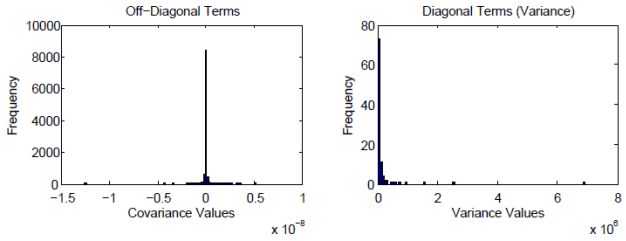


4. From the above plot identify characteristics or features that are lost in the low dimensional representations. For example, can you express glasses or mouth gestures in the low dimensional subspaces?

★ **SOLUTION:** Features like glasses are quickly removed in the low dimensional representation. In addition more interesting properties like smile shape (tongue sticking out), jewelery and even facial obstruction by hair are lost in the low dimensional representation.

1. Project all the training images into the `dim=100` linear space using the above function. Compute the covariance matrix for the `dim=100` dimensional representation `Z` and plot the distribution of the diagonal and off diagonal terms.

★ **SOLUTION:**



2.4 Gaussian Naive Bayes [20 Points]

Ultimately, beauty is in the eye of the beholder. In the case of this question the beholder is a simple Gaussian Naive Bayes classifier. Here we will use a Gaussian Naive Bayes classifier with separate variance terms for each feature and class. Thus, the joint probability can be written as:

$$\mathbf{P}\left(Z_1, \ldots, Z_{\text{dim}}, Y \mid \theta, \mu, \sigma\right) \propto \theta^Y (1-\theta)^{(1-Y)} \prod_{i=1}^{\text{dim}} \exp\left(-\frac{(Z_i - \mu_{i,Y})^2}{2\sigma_{i,Y}^2}\right)$$

★ **SOLUTION:** There was a minor bug in the above equation that several people noted. The normalizing constant $1/\sigma_{i,Y}$ was dropped. The correct form of the equation should have been:

$$\mathbf{P}\left(Z_1, \ldots, Z_{\text{dim}}, Y \mid \theta, \mu, \sigma\right) \propto \theta^Y (1-\theta)^{(1-Y)} \prod_{i=1}^{\text{dim}} \frac{1}{\sigma_{i,Y}} \exp\left(-\frac{(Z_i - \mu_{i,Y})^2}{2\sigma_{i,Y}^2}\right)$$

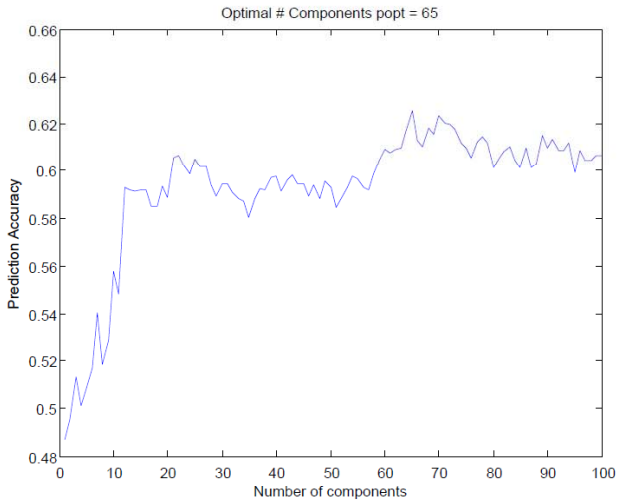
2. If you assume that the rows of `Z` are Gaussian, what can you say about the independence of the "features" of `Z`? How might this be helpful when using a Gaussian Naive Bayes classifier?

★ **SOLUTION:** The definition of SVD ensures that individual pairs of eigenfaces have minimal covariance on the training data. Thus by projecting faces onto the eigenfaces we are minimizing the covariance in the transformed feature space. These transformed features are less "dependent" and therefore closer to the Naive Bayes assumption. Since the original features are unlikely to be Gaussian and we have not addressed higher order dependencies and so it is unreasonable to claim complete independence.

3. Write a function that given the data `Z` and `Y = hot` computes the parameters θ and the `2` by `dim` parameter matrices μ and σ for the Gaussian Naive Bayes classifier. Provide the code in your solution.

6. Plot the average cross validation prediction accuracy as a function of the number of principal components.

★ **SOLUTION:**



7. What number of components maximizes the average cross validation prediction accuracy?

★ **SOLUTION:** 65

8. You may now use the test data `te`. Using the optimal number of components project the faces from `te` into the low dimensional space. Then using the Gaussian Naive Bayes model trained on all of `tr` compute the prediction accuracy. Compare the prediction accuracy for the Gaussian Naive Bayes classifier to the prediction accuracy for the simple classifier that believes all faces are hot.

★ **SOLUTION:** The test error is 0.65 which is better than the classifier that finds all faces hot which would get an accuracy of 0.50.

LDA (2) – practice

p CMU, s.10-701, HW5, pr. 4 (extension of linear discriminant analysis to multi-class case and

nonlinear decision boundaries. + questions?)

4 Linear Discriminant Analysis (45%)

In this problem we will consider an extension of linear discriminant analysis to multi-class case and nonlinear decision boundaries. You will need to download the data file from `http://www.cs.cmu.edu/~epxing/Class/10701/hw5/lda.mat`. The relevant variables in it are X, 300 2D points, and Y, the class labels (one of three classes).

Let C be the number of classes; we will assume that the prior probability (estimated as the relative frequency) for each class is uniform, $1/C$. We will start with the assumption that the covariance matrices are identical for all the classes, i.e., $\forall k = 1, \ldots, C, \Sigma_k = \Sigma$. The means $\mu_1, \ldots, \mu_C$ are of course distinct.

1. Show that the optimal decision rule is based on calculating a set of C linear discriminant functions

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k$$

   and selecting $C = \text{argmax}_c \delta_c(x)$. Apply this method to the data, using ML estimate of means and covariance, and plot the resulting linear decision boundaries along with the data. Report the classification error you obtain.

2. Now assume that the covariances are no longer required to be identical. Derive the (no longer linear) discriminant function for this case. Apply the resulting decision rule to the data, plot the decision boundaries and report classification error. Briefly discuss your conclusions, including the reasons for difference in performance between the two models.

p Radford, Spring 2007, (7), linear + quadratic distriminant on two datasets

STA 414/2104, Spring 2007 — Assignment #2

Due at start of class on March 9. Note that this assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own.

In this assignment you will apply several linear and quadratic discriminant methods to two synthetic datasets, and assess how well the different methods work on these problems.

For both datasets, each case has values for a binary class variable and for six real-valued input variables. The first dataset has 250 training cases and 3000 test cases. The second dataset has 2000 training cases and 5000 test cases. The training and test data can be downloaded from the course webpage, in eight files, with the first data set identified by 'a2a' and the second by 'a2b', with 't' for class variables (targets) and 'x' for input variables.

You should try out four classification methods on these datasets, a follows:

- A linear discriminant classifier, found from the class means and the pooled covariance matrix. You should set the threshold (determined by w_0) so as to minimize the error rate on training cases.
- A quadratic discriminant classifier, found from the class means and the covariance matrices for each class. Again, you should set the threshold (determined by w_0) so as to minimize the error rate on training cases.
- Maximum likelihood logistic regression with just the original inputs as predictors (plus an intercept).
- Maximum likelihood logistic regresson with the original inputs, the squares of inptus, and all the pairwise products of two different inputs as predictors (plus an intercept).

You should write R functions that implement the first two methods (not using any builtin R functions except general facilities such as matrix inverse). You should use R's builtin `glm` function (with the `family="binomial"` and `maxit=1000` options) to find the maximum likelihood estimates for logistic regression, but you should make predictions for test cases using this maximum likelihood estimate without using any builtin functions related to `glm`.

You should report the error rate on test cases from each of these four methods applied to each of the two datasets. You should also discuss why the results are as they are. As a basis for this discussion, you should look at scatterplots of pairs of input variables (with class indicated by colour or type of point drawn), so as to see whether the data has the characteristics that one would expect are necessary for each method to work well. You may also want to look at the estimated coefficients for the discriminants, comparing what was found by the two linear or the two quadratic methods.

You should hand in a listing of your program, with suitable but not excessive comments, the error rates you found, your discussion, and whatever (non-excessive) plots or other output is needed to support your discussion.

p Radford, Spring 2006, (9), LDA on an artificial dataset

STA 414/2104, Spring 2006 — Assignment #2

*Due at **start** of class on March 9. Note that this assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own.*

In this assignment you will try out logistic regression and linear discriminant analysis on an artificial dataset that I have created. This dataset resembles data obtained from spectrometry methods (eg, mass spectrometry or NMR spectrometry), which are commonly used to analyse complex mixtures of components, such as occur in blood. The aim is to assign spectra to one of two classes, which might in a real application correspond, for example, to blood from patients who do or do not have a certain disease.

Each spectrum consists of 200 non-negative data points, which are ordered (eg, by mass for mass spectrometry). Nearby points in the spectrum are often correlated, since a single component of the mixture analysed may show up as a peek that is spread over a range of masses. We hope that which peeks are present in the spectrum, or their magnitude, is related to the class, but there may also be variation unrelated to the class. Each point is also subject to measurement noise.

From the course web page, you can obtain a training set of 60 cases, with the inputs for each case consisting of the 200 points of the measured spectrum. You also are given the class (0 or 1) for these 60 cases. You are to test the methods you try on a test set of 1000 cases, also available from the web page, for which you initially assume that you have only the 200 inputs, from which you try to predict the class. The classes for these test cases are also available, however, so that you can see how well each method did (in terms of error rate). But the methods you test should not look at these, of course!

Rather than try to apply logistic regression and LDA directly to the 200 inputs, you should apply them after reducing dimensionality to 10 by one of two methods:

- Simply average the inputs in blocks of 20. That is, the first average is of inputs 1–20, the next of inputs 21–40, etc.
- Find the first ten principle components.

For each of these new data sets (with only 10 inputs), you should try out both logistic regression, with parameters found by maximum likelihood, and LDA. (You will therefore try a total of four methods.)

You should write your own R or Matlab functions for finding principle components and for LDA — you should **not** use or consult the source code for any predefined functions in R or Matlab that do PCA or LDA. However, you **may** use a predefined function for finding maximum likelihood estimates for logistic regression, and information on doing this in R and Matlab will be put on the course web page soon. However, you should **not** use a predefined function for using the maximum likelihood estimates to make predictions. You should instead write your own code to make predictions for test cases using the estimated regression coefficients.

Once you have the results of the four methods, you should try to make sense of them — for example, by plotting the original data and the principle components, examining the details of the discriminant functions, etc. You should hand in a listing of the functions and scripts you wrote, the results you obtained (summaries only, not results for every test case), and your discussion of the results, including whatever plots or other information you think helps explain these results. Finally, you should suggest methods that might be better than any of the four you tried (but you're not expected to actually try them).

MIT, 2004f, HW2, pr. 2 (Fisher lin disr; intro + proof!!!)

Problem 3: Fisher linear discriminat vs. logistic regression

For this problem we have provided a MATLAB data file `data.mat` that contains 4 datasets, all binary classification tasks. Each dataset consists of a training set (`train`i) and a test set (`test`i):

`train1, train1_2, test1` Data generated from two bivariate Gaussians with identical covariances

`train2, test2` Data generated from two bivariate Gaussians with different covariances

`train3, test3` A digit classification task. The vectors representing digits are projected to a plane defined by two dimensions that capture most of the variability.

`train4, test4` The same digit classificaction task as in the 3rd dataset, but now digit images are 64-dimensional vectors of 1's or 0's — pixels in a 8×8 bitmap. The 3rd dataset is derived from this representation by projecting onto a plane.

The `.X` field of each variable is the representation of the points (each row is one datapoint), while the `y` field contains the class labels (0 or 1).

We have provided the following MATLAB functions that implement both Fisher discriminant classification and logistic regression:

`plotdata(train`i`)` For 2D data, plots the data and associated labels.

`w = fisherdiscriminant(train`i`.X, train`i`.y)` Trains a Fisher discriminant linear classifiers and returns its parameters.

`w = logisticreg(train`i`.X, train`i`.y)` Trains a logistic regression classifier by maximizing likelihood with the Newton's method, and returns its parameters.

`boundary([w1 w2 ...], test`i`)` For 2D data, plots the data and the decisions boundaries of several logistic regression or Fisher discriminant sets of parameters in a single figure.

`errorrate(w, test`i`)` Computes the error rate of a Fisher discriminant or logistic regression model.

1. *(5 points)* Train logistic regression and Fisher discriminant classifiers on each of the first 3 training sets and report the test error on the corresponding test set (6 numbers). For each dataset plot on the same graph the test data and the decision boundaries corresponding to logistic regression and Fisher discriminant methods (3 plots).

2. *(5 points)* On the first data set (`test1.dat`) the performance of the two classifiers is similar. Would you expect the performances to be similar for all datasets sampled from class conditional distributions that are Gaussians with equal covariances?

3. *(5 points)* Training set `train1_2` is identical to `train1` except that it has an extra training point. Train the logistic regression and the Fisher discriminant on `train1_2` and compare the error rates on `test1` with those achieved by models trained on `train1`. For each method explain why the error rates change or not change with the addition of a single training point.

4. *(5 points)* Train logistic regression and the Fisher discriminant on the full 64 dimensional representation of the digits and report the error rates (`train4` and `test4`). Compare the error rates with those achieved on the reduced 2D representation (3rd dataset). Is the multivariate Gaussian assumption reasonable for the full representation of the digits? How sensitive logistic regression and Fisher discriminant classification are to the Gaussian assumption?

Problem 3: Linear Discriminant vs. Logistic Regression

1. **Solution:**

	Set 1	**Set 2**	**Set 3**
logistic	0.067	0.17	0.2125
Fisher disc	0.07	0.1650	0.22

For the plots see Figure 3 at the end. ■

2. **Solution:** Yes, the performance of logistic regression and classification with the Fisher linear discriminant should be similar if data truly comes from Gaussian classes

of equal covariance. In this situation in the limit of infinite training data both logistic regression and the Fisher discriminant converge to the decision-theoretical optimal boundary, thus the only differences should arise because of the randomness of the finite training sample. ■

3. **Solution:**

	Trained on `train1`	**Trained on** `train1_2`
logistic	0.067	0.067
Fisher disc	0.07	0.1510

We observe that the addition of the outlier does not affect logistic regression, but has a strong negative impact on the performance of the Fisher linear discriminant classifier.

Since the new point is correctly classified and far from the boundary, the logistic model assigns to the outlier a $P(y|\mathbf{x})$ probability exponentially close to 1. Adding this probability to the likelihood has almost not effect on the criterion, because the probability is already almost maximum at the point. Thus logistic regression is not affected.

On the other hand the Fisher discriminant sees the data as if each class is Gaussian, and the addition of a single point very far from the current mean can greatly affect the estimate of the mean and variance of that Gaussian. We can distinguish two effects on the decision boundary:

- a translation of the decision boundary because the location of the mean of one class shifts with the addition of the outlier
- a rotation of the decision boundary because the variance in one direction increases by a large amount, while the variance in the perpendicular direction remains small. Thus the projection performed by the Fisher discriminant has to be rotated to keep the variance small.

■

4. **Solution:**

	64 features	**2D projection**
logistic	0.1425	0.2125
Fisher disc	0.23	0.22

We observe that Fisher discrimination and logistic regression achieve similar classification performance on the reduced 2D representation, but while the performance of logistic regression improves significantly on the full set of features, that of Fisher discrimination remains at best the same (if not even worse than that on 2D features).

Several properties of the given representation of digits violate the Gaussian assumption. Think about averaging together all digit images in one class as if they were

printed on transparent playing cards and placed in a deck. If digits were Gaussian, the average image should be well defined (the mean of the multivariate Gaussian), and the variability around that mean image should be distributed in all directions as if its noise. In reality:

- some digits, like 4 and 7, are commonly written in more than one way, so when we look through the transparent deck of cards we see more than one defined outline
- the same basic outline can be transformed by slight rotations, shear, scaling, or translation, and while the digit remains the same, the Gaussianity is violated by such operations

Logistic regression is not that sensitive to the Gaussian assumption. If fact, in logistic regression we only model $P(y|\mathbf{x})$ as if it is the decision boundary between Gaussians, but we do not assume that $P(\mathbf{x})$ is a Gaussian distribution.

The Fisher discriminant is more sensitive to the Gaussian assumption because it is derived by modeling directly the means and covariances of the training data as if classes were Gaussian. If data is not Gaussian the means and covariances alone do not fully capture data structure. ■

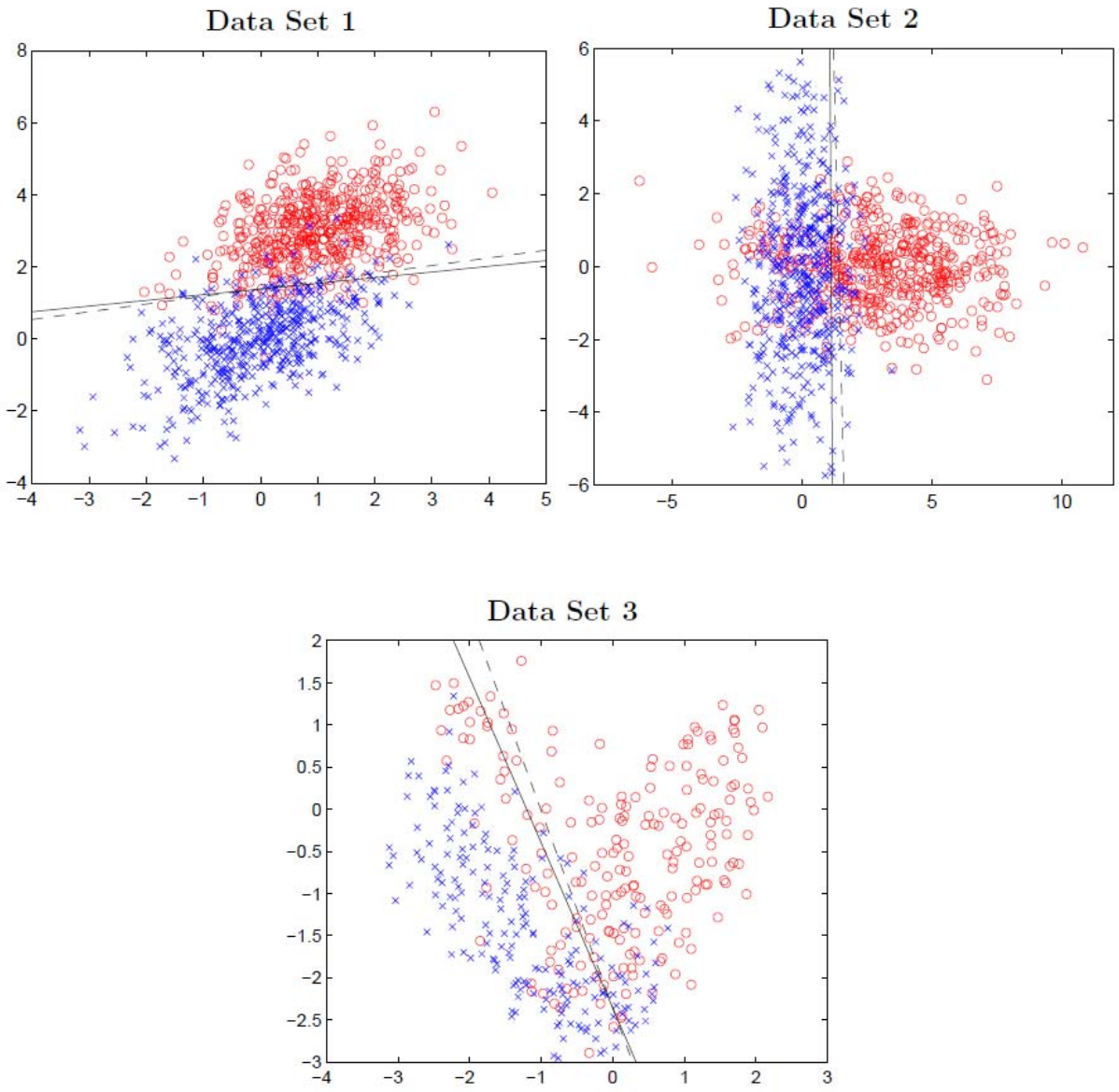


Figure 3: Plots for Problem 3 Part 1

Spectral Clustering (1) - practice

p2 CMU, 2010f, ASingh, HW5, pr. 3 (intro; 1.questions on a datasset;2.implementation;3.theory)

3 Spectral Clustering [Leman, 25 points]

There is a class of clustering algorithms, called spectral clustering algorithms, which has recently become quite popular. Many of these algorithms are quite easy to implement and perform well on certain clustering problems compared to more traditional methods like k-means. In this problem, we will try to develop some intuition about why these approaches make sense and implement one of these algorithms.

Before beginning, we will review a few basic linear algebra concepts you may find useful for some of the problems.

- If A is a matrix, it has an eigenvector v with eigenvalue λ if $Av = \lambda v$.
- For any $m \times m$ symmetric matrix A, the *Singular Value Decomposition* of A yields a factorization of A into

  $$A = USU^T$$

  where U is an $m \times m$ orthogonal matrix (meaning that the columns are pairwise orthogonal) and $S = diag(|\lambda_1|, |\lambda_2|, \ldots, |\lambda_m|)$ where the λ_i are the eigenvalues of A.

Given a set of m data points $x_1, \ldots, x_m$, the input to a spectral clustering algorithm typically consists of a matrix, A, of pairwise similarities between datapoints. A is often called the *affinity matrix*. The choice of how to measure similarity between points is one which is often left to the practitioner. A very simple affinity matrix can be constructed as follows:

$$A(i,j) = A(j,i) = \begin{cases} 1 & \text{if} d(x_i, x_j) < \Theta \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $d(x_i, x_j)$ denotes Euclidean distance between points x_i and x_j.

The general idea of spectral clustering is to construct a mapping of the datapoints to an eigenspace of A with the hope that points are well separated in this eigenspace so that something simple like k-means applied to these new points will perform well.

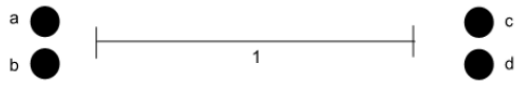


Figure 3: Simple dataset.

As an example, consider forming the affinity matrix for the dataset in Figure 3 using Equation 1 with $\Theta = 1$. Then we get the affinity matrix in Figure 4(a).

$A =$

	a	b	c	d
a	1	1	0	0
b	1	1	0	0
c	0	0	1	1
d	0	0	1	1

(a)

$\tilde{A} =$

	a	c	b	d
a	1	0	1	0
c	0	1	0	1
b	1	0	1	0
d	0	1	0	1

(b)

Figure 4: Affinity matrices of Figure 3 with $\Theta = 1$.

Now for this particular example, the clusters $\{a, b\}$ and $\{c, d\}$ show up as nonzero blocks in the affinity matrix. This is, of course, artificial, since we could have constructed the matrix A using any ordering of $\{a, b, c, d\}$. For example, another possibile affinity matrix for A could have been as in Figure 4(b).

The key insight here is that the eigenvectors of matrices A and $\tilde{A}$ have the same entries (just permuted). The eigenvectors with nonzero eigenvalue of A are: $e_1 = (.7, .7, 0, 0)^T$, $e_2 = (0, 0, .7, .7)^T$. And the nonzero eigenvectors of $\tilde{A}$ are: $e_1 = (.7, 0, .7, 0)^T$, $e_2 = (0, .7, 0, .7)^T$. Spectral clustering embeds the original data points in a new space by using the coordinates of these eigenvectors. Specifically, it maps the point x_i to the point $(e_1(i), e_2(i), \ldots, e_k(i))$ where $e_1, \ldots, e_k$ are the top k eigenvectors of A. We refer to this mapping as the spectral embedding. See Figure 5 for an example.

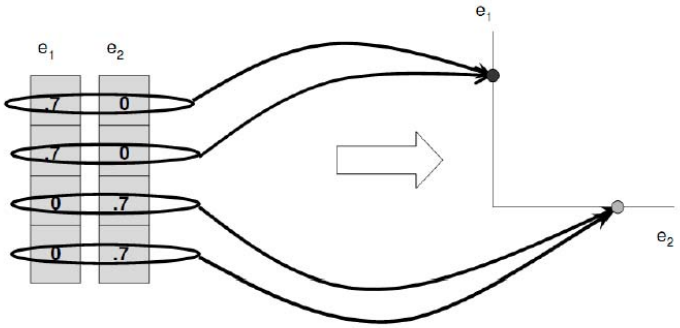


Figure 5: Using the eigenvectors of A to embed the data points. Notice that the points $\{a, b, c, d\}$ are tightly clustered in this space.

3.1 Another Simple Dataset

In this problem we will analyze the operation of one of the variants of spectral clustering methods on another simple dataset shown in Figure 6.

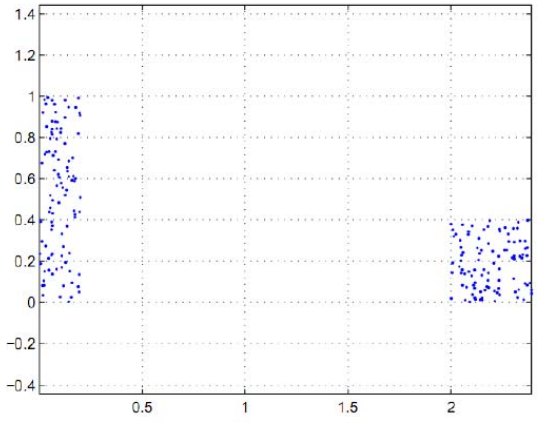


Figure 6: Dataset with rectangles.

1. **[3 points]** For the dataset in Figure 6, assume that the first cluster has m_1 points and the second one has m_2 points. If we use Equation 1 to compute affinity matrix A, what Θ value would you choose and why?

   SOLUTION: In general, we want to choose such a parameter that the similarity between points in different clusters is 0 (or close to it), while the similarity between neighboring points in the same cluster is close to 1. The answers to this question are based purely on eye-balling. So, in this case, Θ in between about 1.04 and 1.75 will result in a block diagonal affinity matrix which is the ideal.

2. [**4 points**] The second step is to compute first k dominant eigenvectors of the affinity matrix, where k is the number of clusters we want to have. For the dataset in Figure 6 and the affinity matrix defined by Equation 1, is there a value of Θ for which you can analytically compute the first two eigenvalues and eigenvectors? If not, explain why not. If yes, compute and write these eigenvalues and eigenvectors down. What are the other eigenvalues? Explain briefly.

SOLUTION: For $1.04 \leq \Theta < 1.75$, we get $A = \begin{pmatrix} 1_{m_1 \times m_1} & 0 \\ 0 & 1_{m_2 \times m_2} \end{pmatrix}$, where $1_{m_1 \times m_1}$ is a block of ones of size m_1 by m_1. Such a matrix has the first two unit length eigenvectors $\frac{1}{\sqrt{m_1}}(1, \ldots, 1, 0, \ldots, 0)^T$ and $\frac{1}{\sqrt{m_2}}(0, \ldots, 0, 1, \ldots, 1)^T$, with eigenvalues m_1 and m_2, respectively. These are in fact the only positive eigenvalues of A, since $rank(A)$ is clearly 2. Therefore, all the other eigenvalues are zero.

3. [**2 points**] As in Figure 5 we can now compute the spectral embedding of the data points using the k top eigenvectors. For the dataset in Figure 6 write down your best guess for the coordinates of the $k = 2$ cluster centers using the Θ that you picked in the first part.

SOLUTION:

$$\left(\frac{1}{\sqrt{m_1}}, 0\right), \left(0, \frac{1}{\sqrt{m_2}}\right)$$

3.3 Theoretic analysis

In the above algorithm we make use of the matrix $N = D^{-1/2}AD^{-1/2}$. Remember that A is an affinity matrix with $a_{ij} = a_{ji}$ being a non-negative distance between points x_i and x_j. D is a dioganal matrix whose i^{th} diagonal element, d_{ii}, is the sum of A's i^{th} row. In the following you will prove several properties related to spectral clustering.

7. [**3 points**] Show that a vector $v_1 = [\sqrt{d_{11}}\sqrt{d_{22}} \ldots \sqrt{d_{nn}}]^T$ is an eigenvector of N with an eigenvalue $\lambda_1 = 1$.

SOLUTION:

We need to show $Nv_1 = v_1$.

$$\begin{aligned} D^{-1/2}AD^{-1/2}v_1 &= D^{-1/2}AD^{-1/2}[\sqrt{d_{11}}\sqrt{d_{22}} \ldots \sqrt{d_{nn}}]^T \\ &= D^{-1/2}A[11 \ldots 1]^T = \left[\frac{\sum_j a_{ij}}{\sqrt{d_{11}}} \ldots \frac{\sum_j a_{nj}}{\sqrt{d_{nn}}}\right]^T \\ &= [\sqrt{d_{11}} \ldots \sqrt{d_{nn}}]^T = v_1. \end{aligned}$$

For the following proof, you might find the following property useful: $\lambda_1 = 1$ is in fact the largest eigenvalue of N and all the other eigenvectors (that are orthogonal to v_1) have an eigenvalue strictly smaller than 1, that is, $|\lambda_i| < 1$ for $\forall i > 1$.

Now consider $P = D^{-1}A$, where $p_{ij} = a_{ij}/d_{ii}$, which is 'kind of' the probability of transitioning from point i to point j. In other words, we normalize each row of P, so that it sums up to 1 and therefore is a valid probability transition matrix. Hence, P^t is a matrix whose $\{i, j\}^{th}$ element shows the probability of being at vertex j if started at vertex i, after t number of steps.

8. [**4 points**] Show that $P^\infty = D^{-1/2}v_1v_1^T D^{1/2}$.

This property shows that if points are viewed as vertices in a Markov graph with transition probabilities proportional to distances between points (elements of A), then v_1 is the only eigenvector needed to compute the probability distribution over states matrix P^∞.

SOLUTION:

$$P^{\infty} = D^{-1/2} N^{\infty} D^{1/2} = D^{-1/2} (\lambda_1 v_1 v_1^T + \lambda_2 v_2 v_2^T + \ldots)^{\infty} D^{1/2}$$

Since the eigenvectors of a symmetric matrix N can always be made orthonormal, the dot-product of any two distinct eigenvectors will be 0 and the dot-product of same eigenvectors will be 1. Hence, $(\lambda_1 v_1 v_1^T + \lambda_2 v_2 v_2^T + \ldots)^{\infty} = (\lambda_1^{\infty} v_1 v_1^T + \lambda_2^{\infty} v_2 v_2^T + \ldots)$. Finally, since $\lambda_1 = 1$ and $|\lambda_i| < 1$ for $\forall i > 1$, $P^{\infty} = D^{-1/2} v_1 v_1^T D^{1/2}$.

p MIT, 2002f, HW6, pr. 1 (+questions!!!)

Problem 1: Clustering

In this problem we will experiment with the spectral clustering algorithm, and explore its properties and the influence of its parameters. A Matlab implementation of spectral clustering is provided in `spectral.m`. You can call it as

```
labels = spectral(X,k,beta);
```

where `X` is the 2D data, `k` is the number of neighbors to use in the neighborhood graph, and `beta` is the weight falloff parameter. The default values for `k` is 3, for `beta` 1. Use `spectral` to experiment with the algorithm, and to test your hypotheses regarding the questions.

You may also find use for the little function

```
X = mkdata(m,n);
```

that generates random data points, from `m` Gaussian distributions, with `n` points from each. Such data typically has `m` sets of points that form natural clusters.

The code is written to perform binary clustering; let us first see how the algorithm deals with the data where there are indeed two clusters.

1. [**5pt**] What will typically happen if only a single nearest neighbor is used to create the graph (i.e. `k=1`)? Explain why that happens.

   Answer: The neighborhood graph will typically not be connected. Therefore, the basic operating assumption that motivates spectral clustering (higher probability of paths within a cluster than paths across the clusters in the random walk) breaks down. As a result the cluster assignments look "chaotic" and bear little connection to the underlying structure of the data (one would expect a reasonable clustering result to somewhat reflect the two Gaussian components).

2. [**5pt**] What happens if `beta` is set to a very large value (say, 100), when `k` is reasonable - say, 3? Explain.

   Answer: The clusters are assigned spuriously, with little relation to the underlying structure.

   The value of β influences the slope of the decaying weight on an edge as a function of the distance. Very low (close to zero) β means the weight is almost constant, and depends very little on the distance; very high β, on the other hand, means the weight falls off rapidly as the distance increases. With $\beta = 100$, the probability of transition from $\mathbf{x}$ to any of its neighbors is negligibly small (assuming we include self-transition as a possibility), and, more importantly, the relative difference between these probabilities for different points and for different neighbors of the same point are vast (many orders of magnitude). Such weights carry little information about the arrangement of the points to be clustered.

Now assume that there are in fact three clusters in the data. In the form given to you, the clustering algorithm of course can not discover the three clusters (since it only looks for two).

3. [**5pt**] How will the algorithm, as given to you, behave on the data with three clusters? Explain.

> **Answer:** There are two main possibilities. To simplify our discussion, let us denote the three components by A, B and C. One possibility is to have all the points assigned to a single cluster. This will happen when each component has

sufficiently many points nearby the other components thus creating significant transitions across the components in the random walk.

In the second typical case the data from two of the Gaussian components (say, A and B) are assigned to one cluster and the remaining component C is a cluster of its own. This happens when there are significant probabilities of transition between the points in A and B, but not between A and C or B and C.

Of course, one will occasionally encounter other scenarios. For instance, when the number of the data points drawn from each Gaussian is low, the neighborhood graph may be disconnected, often leading to a separate connected component per Gaussian. However, to use this the algorithm would have to include either a simple preprocessing stage, which the current implementation does not do.

4. [**10pt**] Modify the algorithm in order to find the three clusters. Note that deciding how many clusters are in fact present in the data is an important problem in clustering; you do not have to solve it, however - the modified algorithm should simply assume there are three clusters. Explain your modifications, and turn in your code.

Answer: One could think of a few ways to handle the multiple cluster situation. Below we mention two of those; any sensible proposal was given the credit. Note that we are not concerned here with the problem of finding the number of clusters: we assume that the data is known to contain 3 clusters, and we only need to find them.

One way is to implement a hierarchical clustering scheme, like the one mentioned in lecture. Namely, once the original clustering algorithm finds the two clusters, we can apply the clustering again within each cluster to search for further subdivisions. This does not require any change in the code. Note that unless some prior information is available on the size of the clusters, we do not know which cluster should be subdivided. We can apply the algorithm on both clusters, and the "right" one will produce a reasonable number of points assigned to different clusters.

An alternative is to use the 2 eigenvectors $\mathbf{v}, \mathbf{u}$ corresponding to the second and third largest eigenvalues. The pair (v_i, u_i) defines a 2D mapping of the data points. Typically the points are easily clustered in this 2D plane (using some simple clustering method, such as k-means).

p MIT, 2003f, HW5, pr. 3 (kmeans+questions!!!)

Problem 3: Spectral clustering

In this problem we will experiment with k-means and a spectral clustering algorithm. We have provided you with a skeleton for a k-means clustering routine, as well as two routines for data generation and plotting:

`X = mkdata(m,n)` generates random 2D data points, from `m` Gaussian distributions, with `n` points from each.

`X = build` allows you to create a data set of manually (graphically) entered points.

`plotclust(X,y)` plots the 2D points specified in the rows of X, and colors each point according to the cluster assignments specified in the vector y (one integer value per point).

The file `clustdata.mat` also contains three data sets `X1`, `X2` and `X3`.

The file `clustdata.mat` also contains three data sets `X1`, `X2` and `X3`.

(3-1) (10pts) Complete the provided skeleton for k-means clustering. The provided skeleton initializes the centers to a random subset of the points, and imposes a constant number (100) of k-means iterations.

Use your k-means implementation to cluster the three provided data sets into $k = 2, 3, 4, 5$ clusters. For each data set and each number of clusters, try running the routine several times. You can also generate additional data sets and test the behavior of k-means on them.

(3-2) (5pts) Why do different executions with the same setting for k yield different clusterings? How would you choose, in an automated way, between these various clusterings? Write a matlab routine that runs k-means five times and chooses the best of the five clusterings (it might be useful to add some calculations, or output variables, to the k-means routine).

(3-3) (5pts) Provided that you have only limited computation time, do you think it makes sense to run k-means several times separately and choose a clustering based on the criterion you suggested, instead of running k-means a single time for more iterations? Why?

(3-4) (5pts) Could the same criterion be used in order to choose between different number of clusters? Why?

Although you should be able to cluster data set `X1`, as well as most data sets generated by `mkdata`, using k-means clustering, the clustering produced for data sets `X2` and `X3` do not seem 'natural'.

(3-5) (5pts) We saw in lecture how k-means can be seen as a 'hard' version of Gaussian mixture density estimation. Suggest a clustering procedure based on Gaussian mixture density estimation that would be able to cluster data set two into natural clusters. You do not need to implement this method. (Hint: would a method based on equal-variance spherical Gaussian be able to capture the clusters?).

We will now see how spectral clustering can be used to cluster data such as `X2` and `X3`.

A Matlab implementation of the spectral clustering method presented in lecture is provided in `spectral.m`. You can call it as

```
labels = spectral(X,2,r,beta);
```

where `X` is an $n \times 2$ data matrix, with a 2D data vector in each row, `r` is the number of neighbors to use in the neighborhood graph, and `beta` is the parameter determining the exponential decay of the edge weights as a function of the distance. The default value for `r` is 3 and the default for `beta` is 1. You might want to refer to Fall 2002 Problem Set 6 (solutions available online; link from the course website) for a discussion on the effect of various settings of r and β.

Our routine only implements clustering into two clusters, as presented in lecture.

A possibly way of extending the spectral clustering framework to more than two clusters is as follows: instead of considering only the eigenvector with the second largest eigenvalue, consider the k eigenvectors $v_1, \ldots, v_k$ with the k largest eigenvalues. The ith point in the data set can now be represented by the k-dimensional vector $(v_1[i], v_2[i], \ldots, v_k[i])$. We can subsequently use k-means in order to cluster the points in this new k-dimensional representation (k-means and k-dimensional can refer to different k's).

(3-6) (5pts) Given an $n \times n$ transition probability matrix P show that the leading eigenvector (the eigenvector with largest eigenvalue) can always be taken to be the vector of all ones.

(3-7) (10pts) Modify our spectral clustering routine to cluster the points into k clusters using this method.

Run the spectral clustering routine, with $k = 2, 3, 4, 5$ on the three data sets, and possibly on other data sets you generated, and experiment with various settings of r and β. Try explaining to yourself the values of r and β that result in the 'natural' clustering, and what happens, e.g., when r is too small, or β is too small or too large.

Another advantage of spectral clustering is that all the operations can be done purely in terms of distances between the points. This means that the points to be clustered need not be points in any R^d Euclidean space, and all that is needed are the distances between the points, or even just the distances between close-by points.

(3-8) (5pts) Why is it not enough to specify only inter-point distances for k-means? What other operation needs to be performed on the input points in k-means?

Problem 3: Clustering

(3-1) (10pts) Here is the modified code `kmeans.m`:

.....

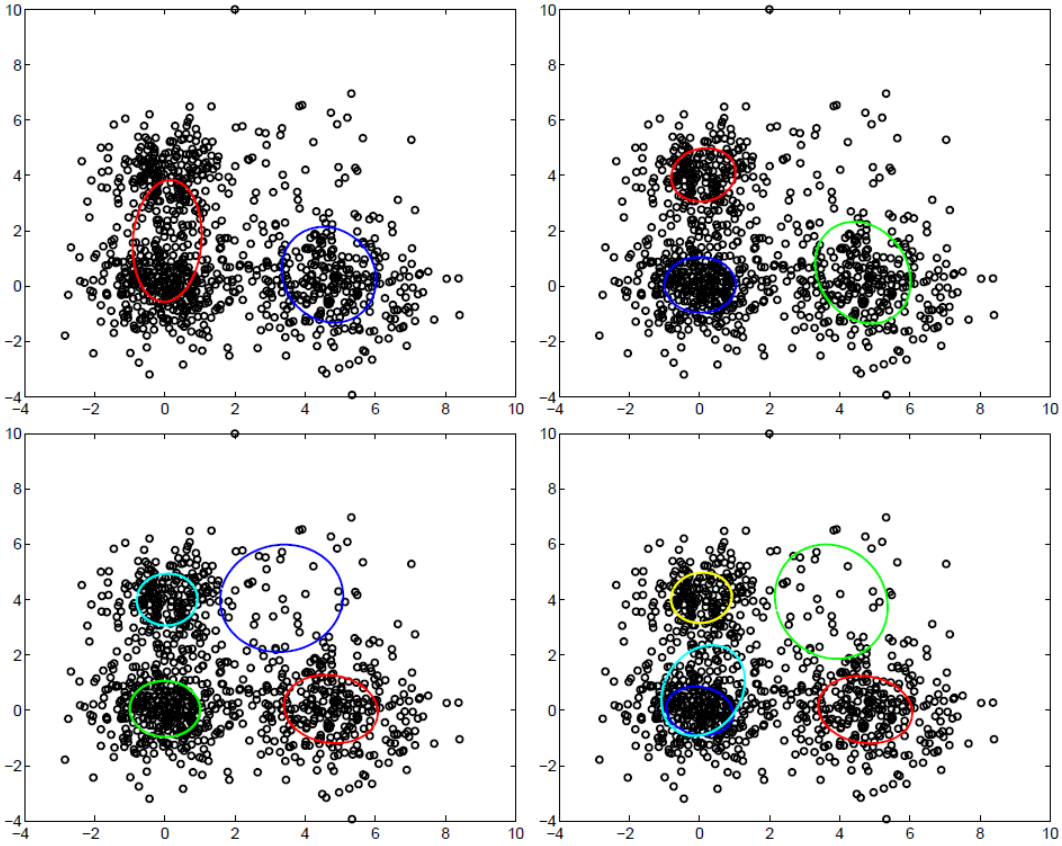


Figure 3: Plots for (2-4) of EM estimated Gaussian mixture models for $m = 2, 3, 4, 5$ (resp. top-left, top-right, bottom-left and bottom-right).

Here is our script `hw5prob3_1.m`:

.....

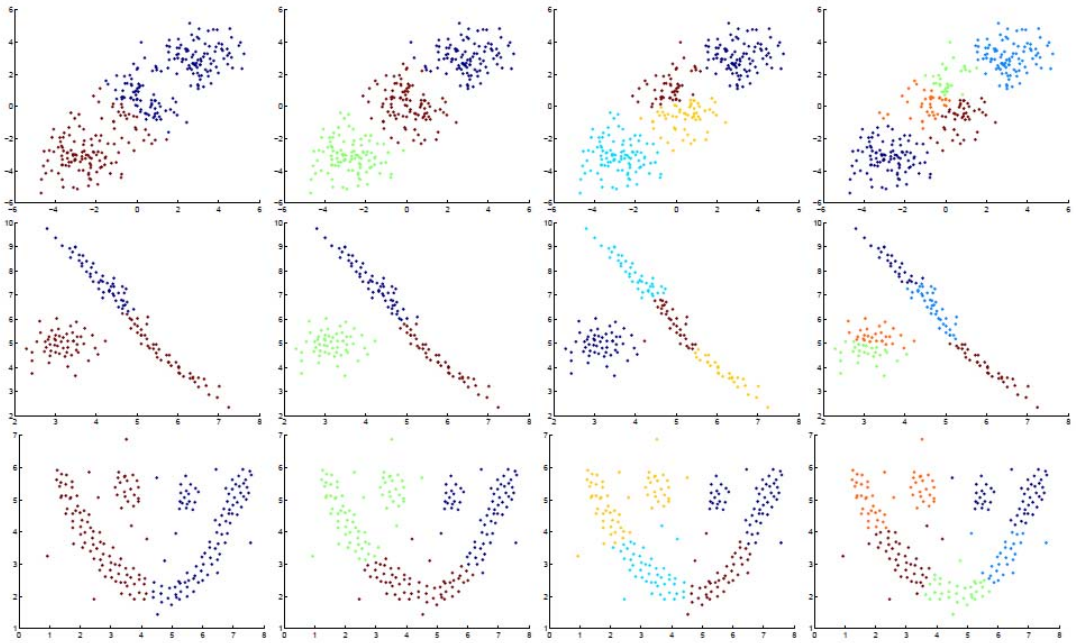


Figure 4: Plots for problem (3-1) for $k = 2, 3, 4, 5$ (left to right) and $X1, X2, X3$ (top to bottom).

The plots generated by this script are shown in Figure 4.

(3-2) (5pts) To choose between the various possible stable points of the k-means algorithm, we should select the clustering which minimizes the average squared distance of each point from its associated mean:

$$J(a, \mu) = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - \mu_{a(i)}\|^2 \tag{56}$$

Where μ_j for $j = 1, \ldots, k$ are the cluster means and $a : \{1, \ldots, n\} \to \{1, \ldots, k\}$ assigns samples to clusters. Note that the "e-step" of k-means assigns each point i to cluster $a(i)$ so as to minize $J(a, \mu)$ for fixed means μ while the "m-step" of k-means resets the means μ so as to minimize $J(a, \mu)$ subject to fixed associations a.

Here is our code to compute this metric:

.....

Here is our script for this problem:

.....

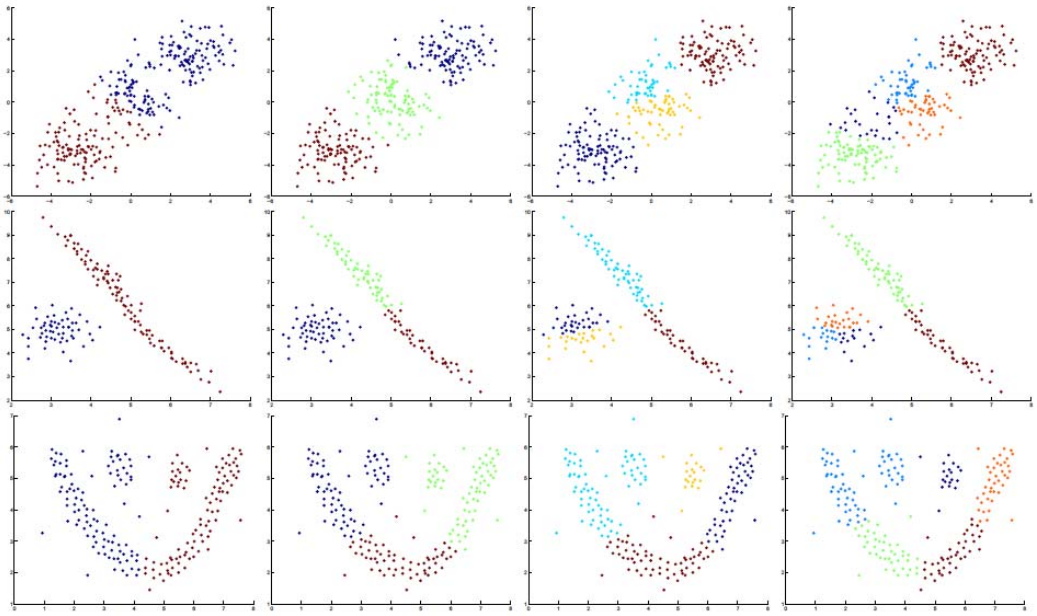


Figure 5: Plots for problem (3-2) for $k = 2, 3, 4, 5$ (left to right) and $X1, X2, X3$ (top to bottom).

The plots generated by this script are shown in Figure 5.

(3-3) (5pts) It probably makes more sense to run k-means many times (for fewer iterations) rather then fewer times (for many iterations) since this should give us a coarse estimate of the global minimum rather than a precise estimate of a local minimum.

(3-4) (5pts) No, this metric would always favor higher values of k as we can always decrease J by adding more clusters. For instance, let $k = n$ and set $\mu_j = \mathbf{x}_j$ for $j = 1, \ldots, n$ so that $J = 0$. This would certainly tend to overfit the data.

(3-5) (5pts) Run EM to generate a joint Gaussian mixture model for (x, y) with $y = 1, \ldots, k$. Then, for each sample x, estimate $\hat{y}(x) = \arg\max_y P(y|x)$. These estimates then produce a clustering of the input samples.

(3-6) (5pts) First, we show that the vector $\pi = (1, \ldots, 1)'$ is an eigenvector of the transittion probability matrix P, with entries $P_{ij} = P(j|i)$, and has eigenvalue $\lambda = 1$, i.e. $P \cdot \pi = \pi$.

$$
\begin{aligned}
(P \cdot \pi)_i &= \sum_i P_{ij}\pi_j && (57)\\
&= \sum_i P_{ij}1 && (58)\\
&= \sum_i P(j|i) && (59)\\
&= 1 && (60)\\
&= \pi_i && (61)
\end{aligned}
$$

Hence, $P \cdot \pi = \lambda\pi$ with $\lambda = 1$ as was to be shown.

Now, we argue that $\lambda = 1$ must be the largest eigenvalue. Suppose there existed a vector π s.t. $P \cdot \pi = \lambda\pi$ with $\lambda > 1.0$. Then, the t-step transition probability matrix P^t must have some eigenvalues going to infinity as t becomes large. But this violates P^t being a transition probability matrix (with entries between 0 and 1). Hence, $\lambda = 1$ is the maximum eigenvalue of P.

Moreover, the symmetric matrix considered in spectral clustering is similar to P and hence has the same eigenvalues as P (with maximum eigenvalue 1 as claimed in lecture).

(3-7) (10pts) Here is the code we modified in `spectral.m`:

.....

I got good results by using $r = 4$ nearest neighbors and setting $\beta = 0.1$ (roughly the inter-point distance between nearest neighbors). Here is our automated script for this problem:

.....

The plots generated by this script are shown in Figure 6.

(3-8) (5pts) We need to recompute the means $\mu_j = \frac{1}{n_j}\sum_{i:a(i)=j} \mathbf{x}_i$ where $n_j = |\{x_i : a(i) = j\}|$ at each iteration of the k-means algorithm. We can't recover these from just the inter-point distances $D_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$ for all i, j.

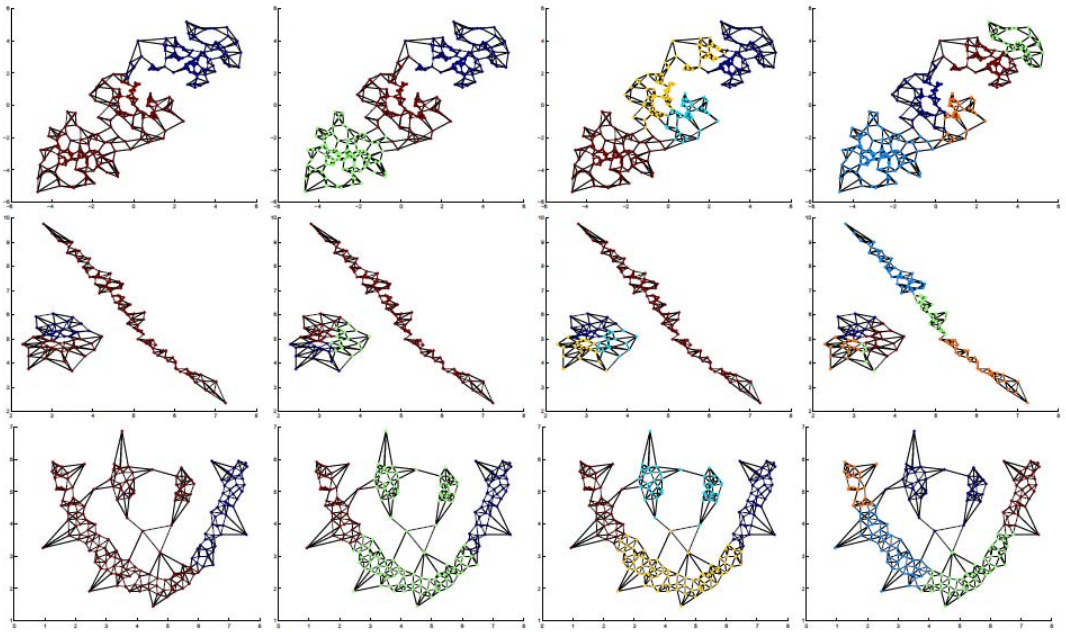


Figure 6: Plots for problem (3-7) for $k = 2, 3, 4, 5$ (left to right) and $X1, X2, X3$ (top to bottom).

p MIT, 2004f, HW5, pr. 3 (kmeans+questions!!!)

Problem 3: Spectral Clustering

In this problem, we experiment with two types of clustering methods—k-means and spectral clustering—and compare their performance on two different data sets. We have provided you with Matlab code to perform k-means clustering (`kmeans.m`) and spectral clustering (`spectral.m`), as well as two 2-dimensional data sets `X1.mat` and `X2.mat`. The function `plotclust(X,y)` will help you visualise the clusters.

1. (*10 points*) Run $k-$means clustering on both data sets, with $k = 2$. Since the initialisation of the 2 means is random, different runs of the algorithm may produce different results. Turn in a typical plot of the clusters obtained. Compare the results in each case with what you expect to be the *correct* answer (when searching for 2 clusters).

2. (*10 points*) Perform spectral clustering on both data sets, with $r = 5$ and $r = 20$ (where r is the number of nearest neighbours considered when building the graph). Turn in plots of the clusters in these cases. Compare the results in each case with what you expect to be the *correct* answer (when searching for 2 clusters), and with the result obtained from $k-$means clustering. Explain the behaviour of spectral clustering in each case.

Problem 3

1. **Solution:** For the data in `X1.mat`, the typical k-means solution is shown in Figure 4. This clearly does not correspond to the intuitive notion of clustering, where the clusters should be the inner and outer rings. The reason for this is that the clusters found by k-means are a partition (tessellation) of space into polygonal regions separated by straight line-segments. For $k = 2$, the two clusters must be separated by a straight line.

   For `X2.mat`, the typical k-means solution is shown in Figure 5. However this is not the only solution possible, since the initialisation of the means is random. Other solutions, such as those shown in Figure 6 occur about once in seven trials.

   The typical solution here does correspond to the intuitive notion of clusters: groups of closely spaced points, where distance between groups are typically larger than distances within the group. The two point in the top-left part of the space would be considered outliers, since there are many more points in the 'clusters' in the bottom-right part. The k-means algorithm assigns these two points to one or the other cluster seemingly randomly.

   ■

2. **Solution:** See Figures 7 and 8 for clustering results with data in `X1.mat`, and Figures 9 and 10 for corresponding results with `X2.mat`.

The spectral clustering algorithm discussed in class has two parameters: the number of nearest neighbours to which each point is connected, r, and the exponential decay parameter, β.

First let us consider the data in `X1.mat`. When $r = 5$, the corresponding graph is not connected, but consists of two subgraphs (each of which is connected). In this case, the largest eigenvalue calculated in the spectral clustering solution is not unique, but has multiplicity two. Thus, the first and second eigenvectors are equivalent, and both of them appear in the expression for the asymptotic transition probability matrix, P^{∞}. Further, neither of these vectors has all elements equal, nor does either correspond to the first order correction term to P^{∞}. So looking at the sign of the elements of the second eigenvector is practically meaningless. In particular, the first two eigenvectors have zero elements corresponding to points in one of the two clusters whenever the graph is disconnected. This can be seen from Figure 11 in the present case. Thus, the resulting clusters depend on how Matlab chooses to interpret the zero elements, as a consequence of finite precision arithmetic. For instance, running the same code with $r = 4$ gives different results, with the inner ring containing both red and blue points (see Figure 12). With $r = 2$, all the points—both inner and outer rings—are considered to be belonging to the same cluster.

However, there is a second way of interpreting the eigenvectors of the normalised affinity matrix. When the affinities (or distances) within each cluster are all constant, and the distances between points belonging to different clusters are also (some other) constant, the two eigenvectors are piecewise constant (i.e. the elements of the eigenvectors are constant for all indices corresponding to a distinct cluster). Hence, clustering the elements of these two eigenvectors (using k-means clustering) provides us the indices of the points belonging to clusters in the original space.

Even if the affinity matrix is not exactly block constant, the eigenvector elements are still approximately piecewise constant as long as the cluster structure is clearly present in the data. This can be seen from both the first and second eigenvectors in the present case. For instance, the elements of the latter tend to cluster around 0 and -0.16, and can easily be separated by k-means clustering.

When $r = 20$, the graph becomes connected. Thus, the spectral clustering solution is a valid one. However, for the default value of the exponential decay parameter ($\beta = 1$), we find that the clustering does not separate the rings. This is because there are now many connections between the inner and outer rings, and the solution found is essentially the same as that using k-means clustering on the data.

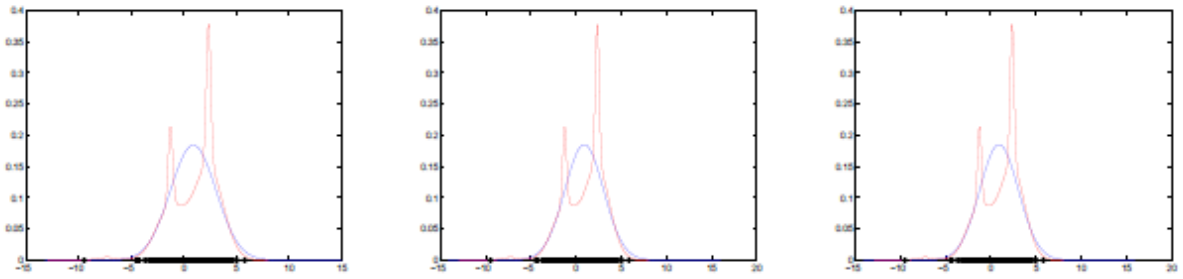
The 'correct' solution—the one separating the inner and outer rings—can be obtained by increasing β to 5, which makes longer distance transitions in the random walk more unlikely (see Figure 13).

Next, consider the data in `X2.mat`. Here, when $r = 5$, the corresponding graph is connected, by virtue of the two outlying points. However, even though connections within each cluster are strong, there are no direct connections between points in the

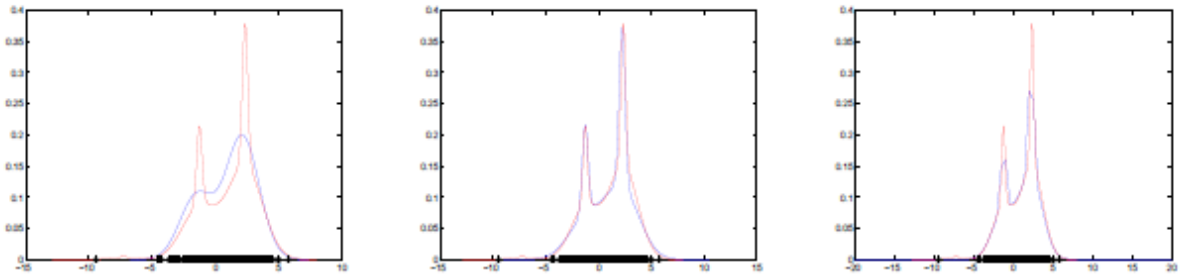
two clusters. Thus, the spectral clustering solution is the one we expect best explains the underlying data.

We have plotted the second eigenvector for this spectral clustering solution in Figure 14. As explained above, there are two clearly identifiable clusters of eigenvector elements (around 0.1 and -0.1), due to tight clustering evident from the data. However, note the two eigenvector elements with values near 0. These indicate the possible presence of a third cluster, and correspond to the two outlying points.

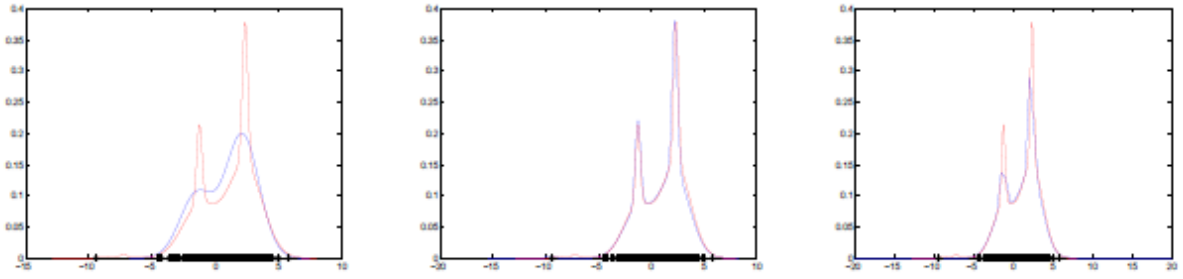
When $r = 20$, we find new connections between points in distinct clusters. Now, the random walk can start at a point in one cluster and end at a point in the other without having to transition through the outlying points. In fact, since the outlying points lie far away from the two clusters, they are connected to the latter by edges with very small weights. In contrast, the weights associated with the edges directly connecting the two underlying clusters are very large. Thus, spectral clustering effectively groups the two clusters into one, and labels the two outlying points as the other cluster. ■



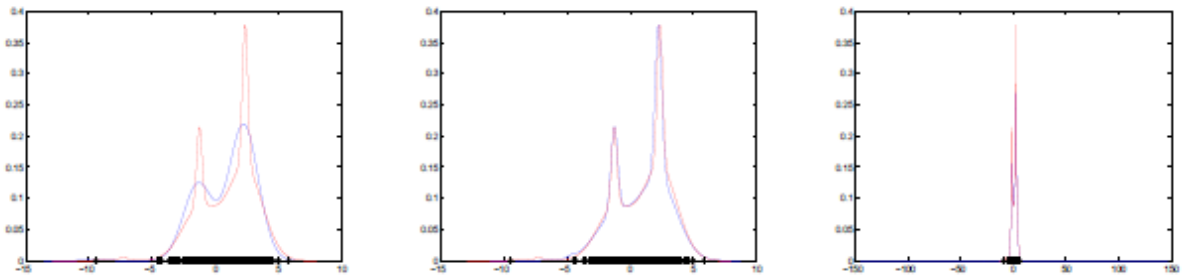
m=1, l=1 m=1, l=2 m=1, l=3



m=2, l=1 m=2, l=2 m=2, l=3



m=3, l=1 m=3, l=2 m=3, l=3



m=4, l=1 m=4, l=2 m=4, l=3

Figure 3: Samples, estimated densities and true densities, for different values of l and m

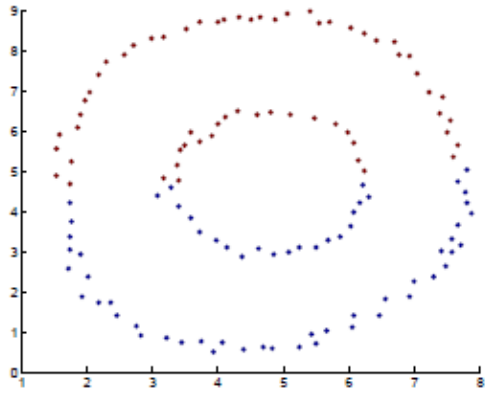


Figure 4: Typical clusters found by k-means ($k = 2$) for data in `X1.mat`

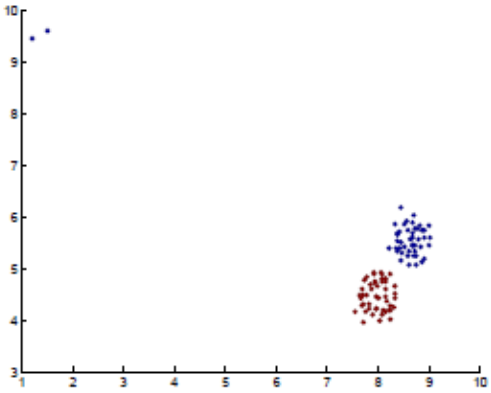


Figure 5: Typical clusters found by k-means ($k = 2$) for data in `X2.mat`

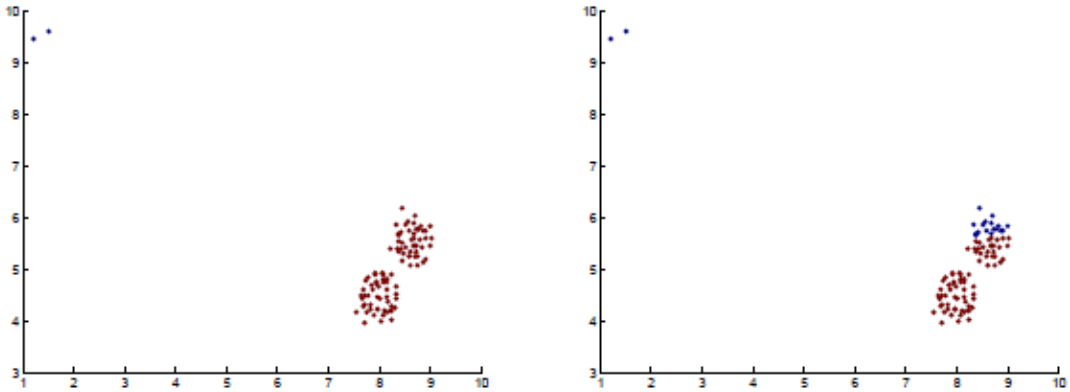


Figure 6: Other (less typical) clustering solutions found by k-means ($k = 2$) for `X2.mat`

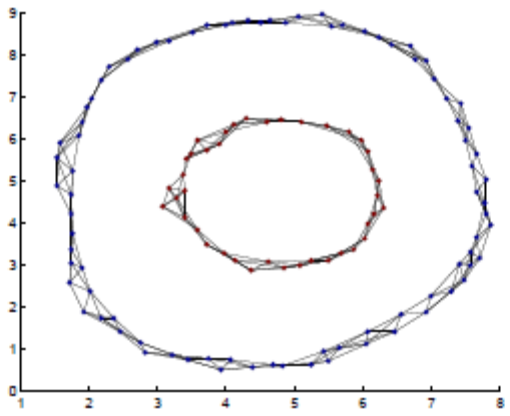


Figure 7: Neighbourhood graph and clusters for data in `X1.mat`, with $r = 5$

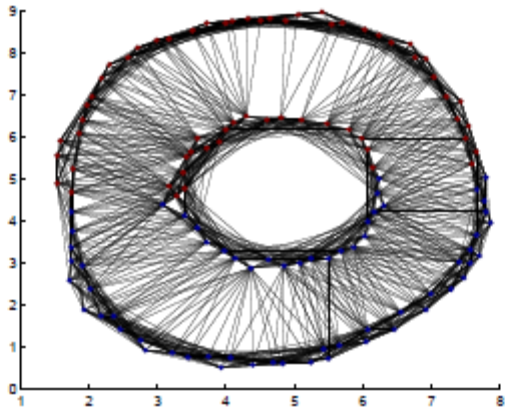


Figure 8: Neighbourhood graph and clusters for data in `X1.mat`, with $r = 20$

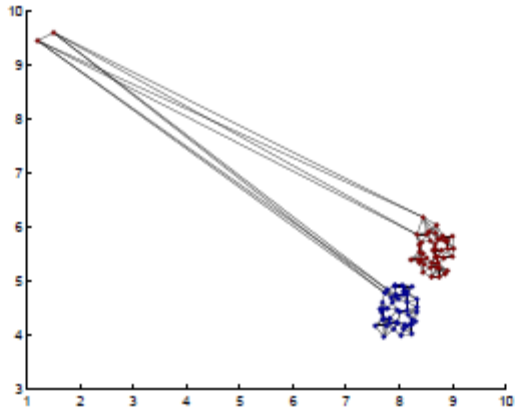


Figure 9: Neighbourhood graph and clusters for `X2.mat`, with $r = 5$

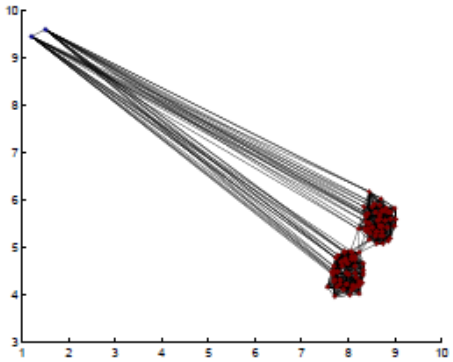


Figure 10: Neighbourhood graph and clusters for `X2.mat`, with $r = 20$

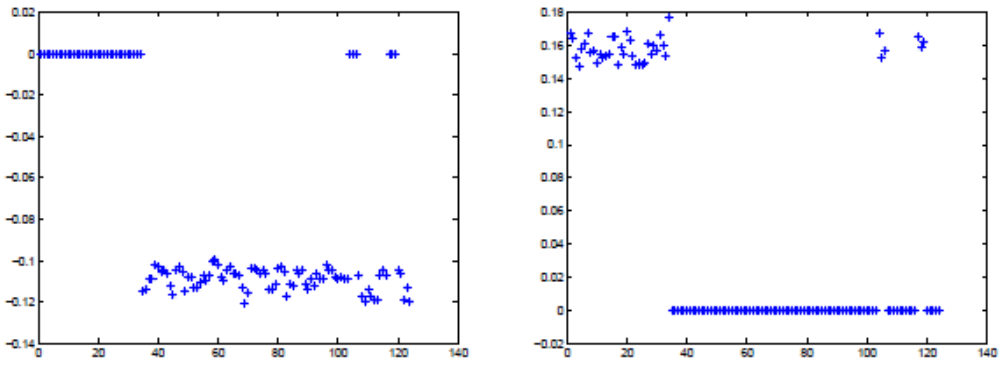


Figure 11: First two eigenvectors (plotted elementwise), with $r = 5$ for data in `X1.mat`

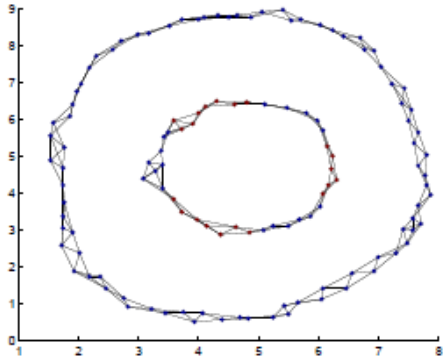


Figure 12: Neighbourhood graph and clusters for `X1.mat`, with $r = 4$

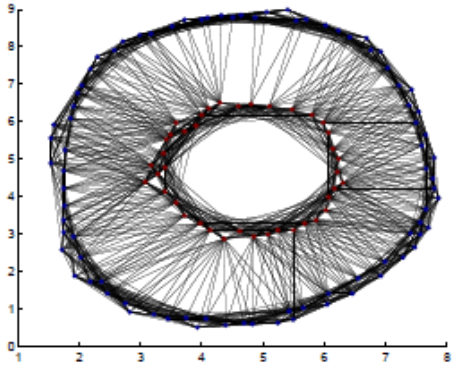


Figure 13: Neighbourhood graph and clusters for `X1.mat`, with $r = 20$ and $\beta = 5$. Notice that the 'correct' clustering is now obtained.

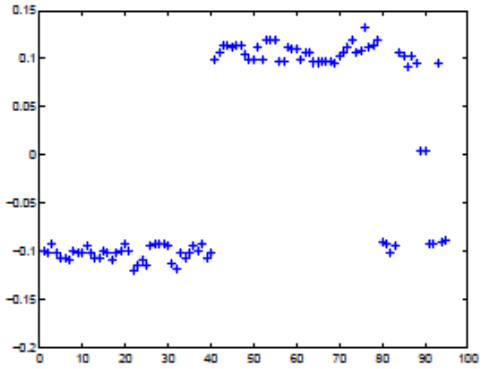


Figure 14: Second eigenvector (plotted elementwise), with $r = 5$ for data in `X2.mat`

FA (1) - practice

p d CMU, 2014f, EXing, BPoczos, HW3, pr. 1.4 (3vPCA, ASI, FA: implementation + questions!)

1.4 Experiment

Here we will compare the above three methods on two data sets. For ASI and FA we will take $\Psi = I_d$.

A common preprocessing step before applying PCA is to subtract the mean. As we will see, without this preprocessing just taking the SVD of X will give very bad results. For the purposes of this problem we will call these two variants "demeaned PCA" and "buggy PCA". Sometimes, after subtracting the mean we also apply a diagonal scaling so that each dimension has unit variance. We will call this normalized PCA.

One way to study how well the low dimensional representation captures the linear structure in our data is to project it back to D dimensions and look at the reconstruction error. For PCA, if we mapped it to d dimensions via $z = Vx$ then the reconstruction is $V^\top z$. For the preprocessed versions, we first do this and then reverse the preprocessing steps too. For ASI we just compute $Az + b$. For FA, we will use the posterior mean $\mathbb{E}[z|x]$ as the lower dimensional representation and $Az + b$ as the reconstruction. We will compare all four methods by the reconstruction error on the datasets.

Please implement code for the five methods: Buggy PCA (just take the SVD of X) , Demeaned PCA, Normalized PCA, ASI, FA. In all cases your function should take in an $n \times d$ data matrix and d as an argument. It should return the the d dimensional representations, the estimated parameters, and the reconstructions of these representations in D dimensions. For FA, use the values obtained from ASI as initializations for A. Set η based on the reconstruction errors of ASI. Use 10 iterations of EM.

You are given two datasets: A two Dimensional dataset with 50 points `data2D.mat` and a thousand dimensional dataset with 500 points `data1000D.mat`.

For the $2D$ dataset use $d = 1$. For the $1000D$ dataset, you need to choose d. For this, observe the singular values in ASI and see if there is a clear "knee point" in the spectrum. Attach any figures/ Statistics you computed for this to justify your choice.

For the $2D$ dataset you need to attach the a plot comparing the orignal points with the reconstructed points for all five methods. For both datasets you should also report the reconstruction errors. The given starter code does all of this for you so you need to just attach the results.

These were our errors for the $2D$ dataset. If your answers do not tally, please check with the TAs.

```
>> q14
Reconstruction Errors:
Buggy PCA: 0.365284
Demeaned PCA: 0.008448
Normalized PCA: 0.008454
ASI: 0.008448
FA: 0.008526
```

Questions

1. Look at the results for Buggy PCA. The reconstruction error is bad and the reconstructed points don't seem to well represent the original points. Why is this ?
   Hint: Which subspace is Buggy PCA trying to project the points onto ?
2. In both demeaned PCA and ASI the errors are identical. But the Z values are not. You can check this via the command `norm(Z2-Z4, 'fro')`. Explain why ?.
3. The error criterion we are using is the average squared error between the original points and the reconstructed points. In both examples ASI (and demeaned PCA) achieves the lowest error among all methods. Is this surprising ? Why ?

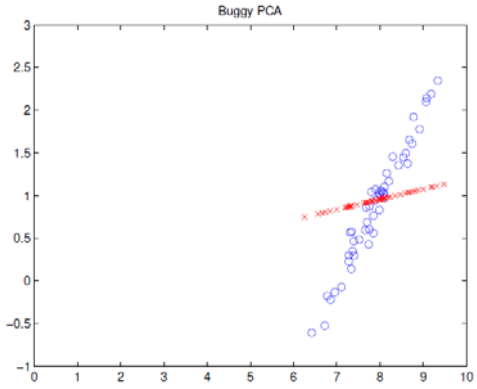
Please submit your code along with your results.

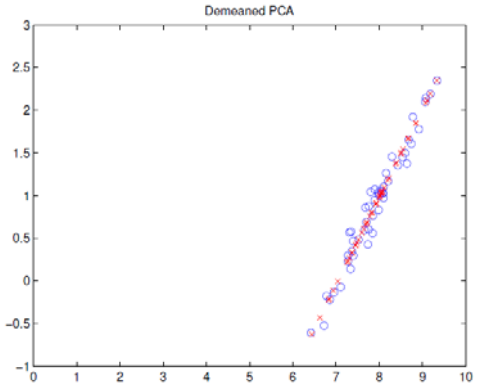
Point Allocation

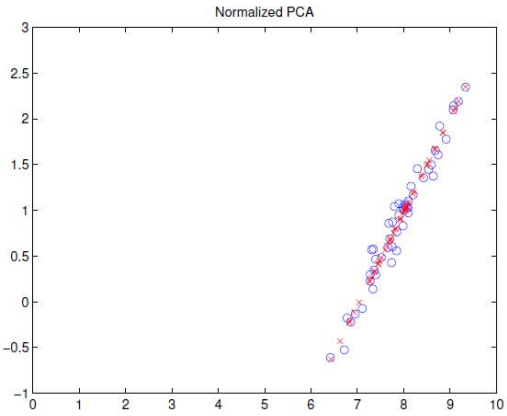
- Implementation of all five methods: **(6 Points)**
- Results - Figures and errors **(3 Points)**
- Choice of d for $1000D$ dataset and appropriate justification: **(1 Point)**
- Questions **(3 Points)**

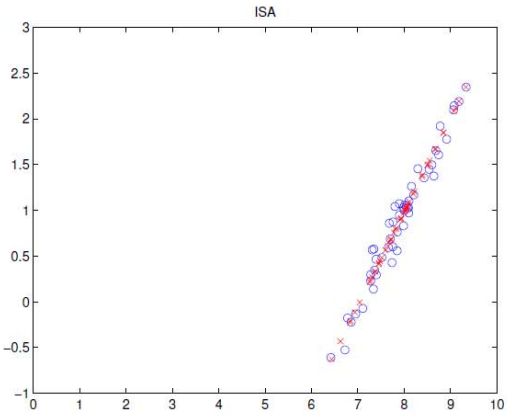
1.4 Experiment

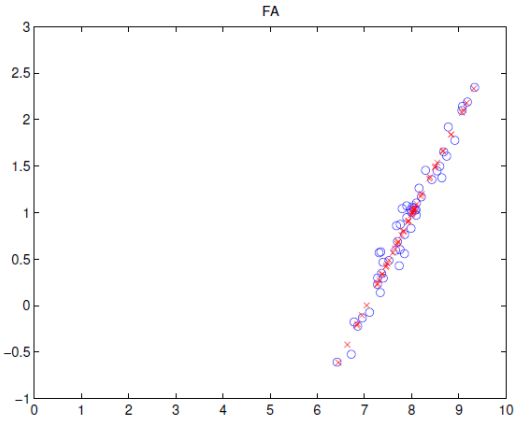
Results for the 2D dataset





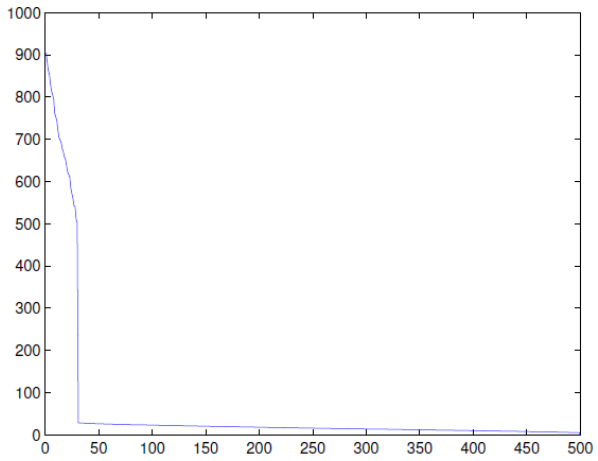






Results for the 1000D dataset

We used $d = 30$ since the singular values fall off sharply after this. See figure below:



This was the output.

```
>> q14
Reconstruction Errors:
Buggy PCA: 777.871447
Demeaned PCA: 272.546928
Normalized PCA: 273.140905
ASI: 272.546928
FA: 272.549605
```

Answers to Questions

1. When you SVD without demeaning on the dataset, you are find the best linear (as opposed to affine) subspace. The first principal component then will then be from the origin towards the data and other principal components will be orthogonal to this.
2. This is because the reconstructions are identical in both cases even if the representations are not.
3. No. Since in ASI we are directly minimizing this error criterion.

This is our implementation.

.....

p d CMU, s.10-701, HW5, pr. 2 (PCA and FA + questions?)

2 Principal Component Analysis and Factor Analysis (extra credit points: 20%)

Generate 200 three-dimensional points (X_1, X_2, X_3) according to

$$X_1 \sim Z_1 \tag{1}$$

$$X_2 = X_1 + 0.001Z_2 \tag{2}$$

$$X_3 = 10Z_3 \tag{3}$$

where Z_1, Z_2, Z_3 are independent, standard normal random variables ($Z_1, Z_2, Z_3 \sim \mathcal{N}(0, 1)$).

1. Compute the leading principal component and factor analysis directions.
2. Show that the leading principal component aligns itself in the maximal variance direction X_3; while the leading factor essentially ignores the uncorrelated component X_3 and picks up the correlated component $X_2 + X_1$.

You do not need to implement PCA or FA by yourself. Instead, use the matlab functions `princomp` and `factoran` in the statistics toolbox. Use the matlab function `biplot` to visualize the results and hand in the plots.

p Radford, Fall 2009, (4), PCA + FA on the same 2 datasets?

STA 437/1005, Fall 2009 — Assignment #3

Due at the start of the lecture on November 30. (Note that due to lack of time, I've had to abandon the idea of an assignment with a two-part solution/critique form that I discussed earlier.)

Please hand this assignment in on 8 1/2 by 11 inch paper, stapled in the upper-left corner, without any folder or other packaging around it.

This assignment is worth 12% of the course grade. It is to be done by each student individually. You may discuss this assignment in general terms with other students, but the work you hand in should be your own. In particular, you should not leave any discussion of this assignment with any written notes or other recordings, nor receive any written or other material from anyone else by other means such as email.

For this assignment, you will look again at the two data sets you used for the second assignment. This time, you will apply principal component analysis (PCA) and factor analysis (FA) to the data, comparing the results you obtain with these methods, and seeing the effect of reducing dimensionality on use of the data for regression or classification. Another purpose of the assignment is to give general insight into properties of PCA and FA.

This handout, the data sets, some hints about useful R commands, and a solution to the second assignment are (or will shortly be) available from the course web page, at `http://www.utstat.utoronto.ca/~radford/sta437/`

Data set 1:

I have provided a version of this data set with nine observations deleted, because they have values for one or more of the variables that are of doubtful accuracy. This version also omits the "density" variable (retaining "pcfat") and the "age" variable. You should use this version for this assignment, and not remove any additional observations as outliers. You should read this data with the "head=TRUE" option. The data frame will contain column names that are the indexes of the observations in the original data set (some indexes are therefore skipped).

You should perform PCA using only the 12 variables other than "pcfat". You should try PCA with and without scaling these variables to all have standard deviation one, and discuss whether scaling or not scaling (or something else) seems most appropriate. You should look at scree plots of the variances in successive principal components, and on this basis comment on how many components should (perhaps) be used.

You should also perform factor analysis on this data, using the 12 variables other than "pcfat". You should try models with one and with two factors.

You should try to interpret the coefficients found by FA and PCA in terms of the meaning of the variables, using your common sense knowledge of the variability of human body proportions.

You should compare the two-factor model with the first two principal components (both with scaling and without scaling). For this, you should look at the coefficients of the linear combinations of the original 12 variables (after centering) that are used to project onto the components found with PCA, and the linear combinations that are used to predict the values of the factors with the FA model. You should take into account the non-uniqueness of the FA solution when doing this, as well as the non-uniqueness of the sign for the principal components.

You should also look at predicting "pcfat" from the other 12 variables, and compare linear regression on all 12 variables with linear regression on a smaller number of variables found using PCA (projections on the components) and with a smaller number of variables found using FA (from the regression estimates of the unobserved factors). You can judge how well "pcfat" can be predicted using the adjusted R-squared value output by the "summary" command applied to the output of "lm". You can also consider adding BMI (weight divided by height squared) as an additional predictor in the regression, and see whether this helps.

Data set 2:

For this data set, you should use the same data file as for the second assignment. It should be read with the "head=TRUE" option. You should not remove any observations as outliers.

You should perform PCA using observations in all classes, and using the 36 variables in this data set with the class variable omitted. You should try PCA with and without scaling these variables to all have standard deviation one, and discuss whether scaling or not scaling (or something else) seems most appropriate. You should also look at scree plots of the variances in successive principal component directions, and on this basis comment on how many components should (perhaps) be used.

You should also perform factor analysis on this data (again, for all classes, 36 variables), using two factors. You should compare the results of PCA and FA. For this, you should look at the coefficients of the linear combinations of the original 36 variables (after centering) that are used to project onto the components found with PCA, and the linear combinations that are used to predict the values of the factors with the FA model. You should take into account the non-uniqueness of the FA solution when doing this, as well as the non-uniqueness of the sign for the principal components.

You should try to interpret the coefficients found by FA and PCA (including how many components seem to be needed) in terms of the meaning of the 36 variables, as observations on 4 spectral bands for 9 pixels, and of the context that observations come from three classes.

You should also look at how effective reducing the 36 variables to only 2 variable with PCA or FA would be if the goal is to classify future observations into one of these three classes. You can do this informally by just looking at scatterplots with the class indicated by colour.

ICA (1) – practice

p d CMU, 2014f, EXing, BPoczos, HW3, pr. 2.2 (ICA; use an API and implemement some steps + question)

2.2 Independent Components Analysis

1. **(3 Points)** You are given starter code that generates some signals in Matlab and then mixes them. Use any ICA library (FastICA is a popular package, http://research.ics.aalto.fi/ica/fastica/) and reconstruct the signals. Attach your code and the unmixed signals.
2. **(1 Point)** Explain why the unmixed components may be scaled versions of the original inputs.

2.2 Independent Components Analysis

Solutions are straightforward.

p CMU, 2014s, BPoczos, ASingh, HW4, pr. 2 (ICA: apply on 2 sets; non-sound; sound - cocktail party pb)

ICA (Prashant; 10 Points)

The purpose of this problem is to experiment with ICA tool boxes and to see how powerful ICA can be. Firstly, you will need to download an ICA package. I recommend FastICA which you can get here : *http : //research.ics.aalto.fi/ica/fastica*. It is available in several languages and you can pick your favorite one.

- We are going to warm up with some synthetic data. Go ahead and make two signals, one sine wave and one sawtooth wave. The code to do this in Matlab would look something like

```
signal1 = sin(linspace(0,50, 1000));
signal2 = sawtooth(linspace(0,37, 1000));
```

Start out by plotting your two signals. Now, generate two mixtures with random coefficients. An example set of mixtures would be

```
mix1 = signal1 - 2*signal2;
mix2 = 1.73*signal1 + 3.41*signal2;
```

Plot both of these mixtures. Now, use the FastICA package to get back the original signals. Plot the two components. Along with your plots, turn in the code you used to generate them (probably won't be longer than a few lines).

Explain why the recovered components might be scaled differently.

- Now we are going to take a look at the cocktail party problem. When in a party, we hear a mixture of sounds coming from the many sources near us. ICA can help break this signal up into its components. To generate back 3 of the source signals, 3 separate inputs must be observed.

  For the homework, we are going to combine two mono samples of sound and then try to recover the original samples. Go ahead and download the zip file along with the homework with two wav files in it.

  Using the programming language of your choice, load the two wav files. Generate two random mixes of these samples just like we did for the first part. Listen to the mixes if you can. These two mixes will simulate the inputs from which we want to recover the signals. Think of it as two different microphones placed in the party.

  Now recover the two signals using the FastICA package. Since they may be scaled differently, divide each signal by the max value in that signal. Listen to the output and make sure it sounds alright. Each signal may have a whisper of the other signal but it should, on the whole, sound like the original.

  Plot the two original wav files, plot the mixtures you generated, and plot the output signals after normalization.

  Note that in practice, this won't work nearly as well. ICA assumes that the two signals mix with the same delay in each input. However, usually, each signal will reach each microphone at different times.

1. The original sin and sawtooth functions are shown in Figure 2
2. The two mixed sin and sawtooth functions are shown in Figure 3
3. The two separated sin and sawtooth functions are shown in Figure 4
4. Let $\mathbf{x} = \mathbf{As}$, where $\mathbf{s} = [s_1, \ldots, s_d]^T$ contains the original hidden independent components. Since for any invertible diagonal matrix $\mathbf{D}$, we have $\mathbf{x} = \mathbf{AD}^{-1}\mathbf{Ds}$, and $\mathbf{Ds}$ also has independent components, therefore ICA algorithms can never know if the original sources were $\mathbf{s}$, or $\mathbf{Ds}$.
5. The original wav files are shown in Figure 5.
6. The mixed wav files are shown in Figure 6.
7. The estimated independent wav files after normalization are shown in Figure 7.

Grading guidelines:

1. Rubric: **1 point** for plotting the two mixed sin and sawtooth functions.
2. Rubric: **3 points** for plotting the two separated sin and sawtooth functions.
3. Rubric: **1 point** for the explanation why the recovered components might be scaled differently.
4. Rubric: **1 point** for plotting the original wav files.
5. Rubric: **1 point** for plotting the mixed wav files.
6. Rubric: **3 points** for plotting the estimated independent wav files after normalization.

p d Stanford, 2007f, ANg, HW4, pr. 3 (implement! PCA and ICA (with advice)!!! and apply them on natural images)

3. PCA and ICA for Natural Images

In this problem we'll apply Principal Component Analysis and Independent Component Analysis to images patches collected from "natural" image scenes (pictures of leaves, grass, etc). This is one of the classical applications of the ICA algorithm, and sparked a great deal of interest in the algorithm; it was observed that the bases recovered by ICA closely resemble image filters present in the first layer of the visual cortex.

The `q3/` directory contains the data and several useful pieces of code for this problem. The raw images are stored in the *images/* subdirectory, though you will not need to work with these directly, since we provide code for loading and normalizing the images.

Calling the function `[X_ica, X_pca] = load_images;` will load the images, break them into 16x16 images patches, and place all these patches into the columns of the matrices `X_ica` and `X_pca`. We create two different data sets for PCA and ICA because the algorithms require slightly different methods of preprocessing the data.[1]

For this problem you'll implement the `ica.m` and `pca.m` functions, using the PCA and ICA algorithms described in the class notes. While the PCA implementation should be straightforward, getting a good implementation of ICA can be a bit trickier. Here is some general advice to getting a good implementation on this data set:

[1] Recall that the first step of performing PCA is to subtract the mean and normalize the variance of the features. For the image data we're using, the preprocessing step for the ICA algorithm is slightly different, though the precise mechanism and justification is not imporant for the sake of this problem. Those who are curious about the details should read Bell and Sejnowki's paper "The 'Independent Components' of Natural Scenes are Edge Filters," which provided the basis for the implementation we use in this problem.

- Picking a good learning rate is important. In our experiments we used $\alpha = 0.0005$ on this data set.
- Batch gradient descent doesn't work well for ICA (this has to do with the fact that ICA objective function is not concave), but the pure stochastic gradient described in the notes can be slow (There are about 20,000 16x16 images patches in the data set, so one pass over the data using the stochastic gradient rule described in the notes requires inverting the 256x256 W matrix 20,000 times). Instead, a good compromise is to use a hybrid stochastic/batch gradient descent where we calculate the gradient with respect to several examples at a time (100 worked well for us), and use this to update W. Our implementation makes 10 total passes over the entire data set.
- It is a good idea to randomize the order of the examples presented to stochastic gradient descent before each pass over the data.
- Vectorize your Matlab code as much as possible. For general examples of how to do this, look at the Matlab review session.

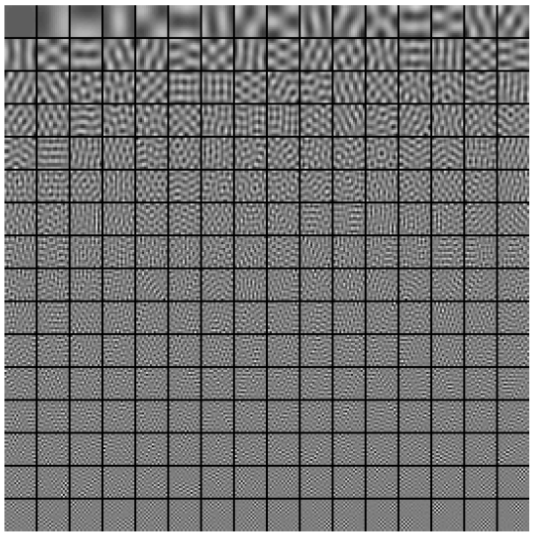
For reference, computing the ICA W matrix for the entire set of image patches takes about 5 minutes on a 1.6 Ghz laptop using our implementation.

After you've learned the U matrix for PCA (the columns of U should contain the principal components of the data) and the W matrix of ICA, you can plot the basis functions using the `plot_ica_bases(W);` and `plot_pca_bases(U);` functions we have provide. Comment briefly on the difference between the two sets of basis functions.

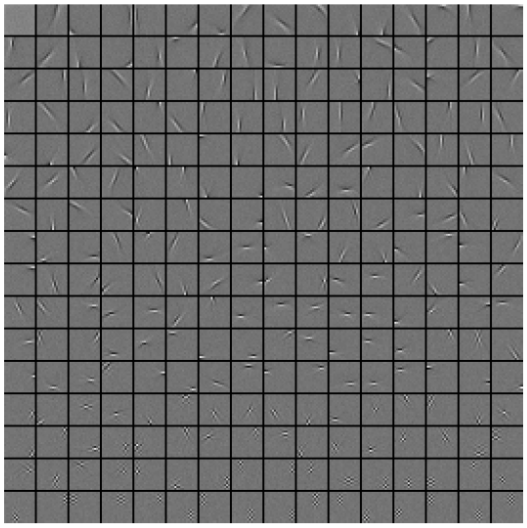
Answer: The following are our implementations of `pca.m` and `ica.m`:

.....

PCA produces the following bases:



while ICA produces the following bases



The PCA bases capture global features of the images, while the ICA bases capture more local features.

p Stanford, ANg, 2009f, HW4, pr. 4 (ICA: implement it)

Others (1) - practice

p d CMU, 2014f, EXing, BPoczos, HW3, pr. 1.4 (3vPCA, ASI, FA: implementation + questions!)

1.4 Experiment

Here we will compare the above three methods on two data sets. For ASI and FA we will take $\Psi = I_d$.

A common preprocessing step before applying PCA is to subtract the mean. As we will see, without this preprocessing just taking the SVD of X will give very bad results. For the purposes of this problem we will call these two variants "demeaned PCA" and "buggy PCA". Sometimes, after subtracting the mean we also apply a diagonal scaling so that each dimension has unit variance. We will call this normalized PCA.

One way to study how well the low dimensional representation captures the linear structure in our data is to project it back to D dimensions and look at the reconstruction error. For PCA, if we mapped it to d dimensions via $z = Vx$ then the reconstruction is $V^\top z$. For the preprocessed versions, we first do this and then reverse the preprocessing steps too. For ASI we just compute $Az + b$. For FA, we will use the posterior mean $\mathbb{E}[z|x]$ as the lower dimensional representation and $Az + b$ as the reconstruction. We will compare all four methods by the reconstruction error on the datasets.

Please implement code for the five methods: Buggy PCA (just take the SVD of X) , Demeaned PCA, Normalized PCA, ASI, FA. In all cases your function should take in an $n \times d$ data matrix and d as an argument. It should return the the d dimensional representations, the estimated parameters, and the reconstructions of these representations in D dimensions. For FA, use the values obtained from ASI as initializations for A. Set η based on the reconstruction errors of ASI. Use 10 iterations of EM.

You are given two datasets: A two Dimensional dataset with 50 points `data2D.mat` and a thousand dimensional dataset with 500 points `data1000D.mat`.

For the $2D$ dataset use $d = 1$. For the $1000D$ dataset, you need to choose d. For this, observe the singular values in ASI and see if there is a clear "knee point" in the spectrum. Attach any figures/ Statistics you computed for this to justify your choice.

For the $2D$ dataset you need to attach the a plot comparing the orignal points with the reconstructed points for all five methods. For both datasets you should also report the reconstruction errors. The given starter code does all of this for you so you need to just attach the results.

These were our errors for the $2D$ dataset. If your answers do not tally, please check with the TAs.

```
>> q14
Reconstruction Errors:
Buggy PCA: 0.365284
Demeaned PCA: 0.008448
Normalized PCA: 0.008454
ASI: 0.008448
FA: 0.008526
```

Questions

1. Look at the results for Buggy PCA. The reconstruction error is bad and the reconstructed points don't seem to well represent the original points. Why is this ?
   Hint: Which subspace is Buggy PCA trying to project the points onto ?
2. In both demeaned PCA and ASI the errors are identical. But the Z values are not. You can check this via the command `norm(Z2-Z4, 'fro')`. Explain why ?.
3. The error criterion we are using is the average squared error between the original points and the reconstructed points. In both examples ASI (and demeaned PCA) achieves the lowest error among all methods. Is this surprising ? Why ?

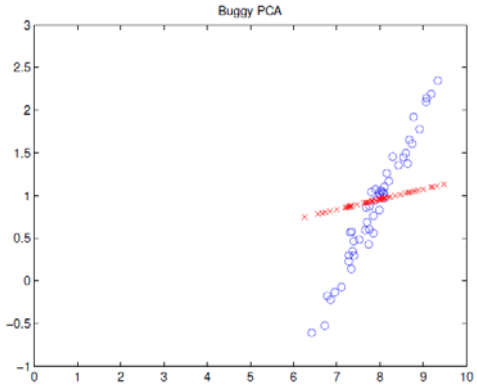
Please submit your code along with your results.

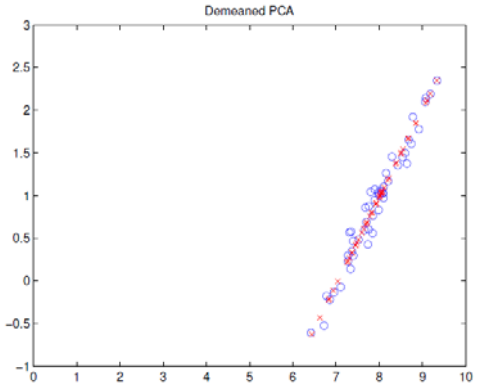
Point Allocation

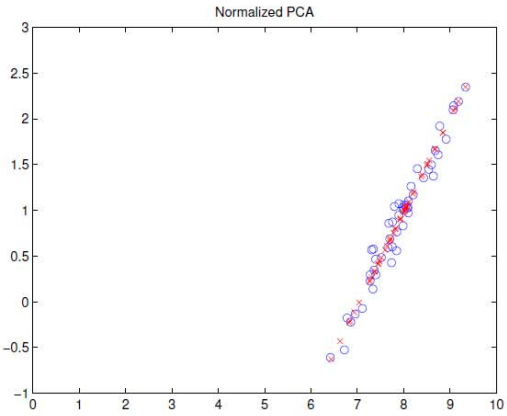
- Implementation of all five methods: **(6 Points)**
- Results - Figures and errors **(3 Points)**
- Choice of d for $1000D$ dataset and appropriate justification: **(1 Point)**
- Questions **(3 Points)**

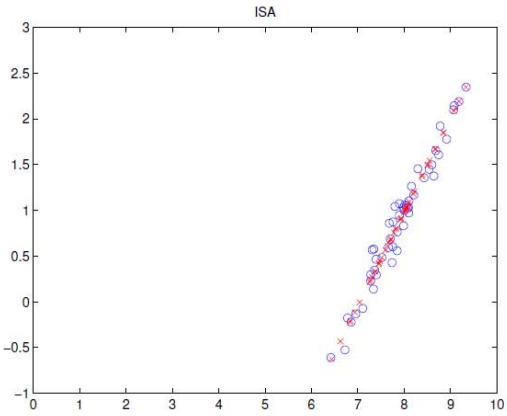
1.4 Experiment

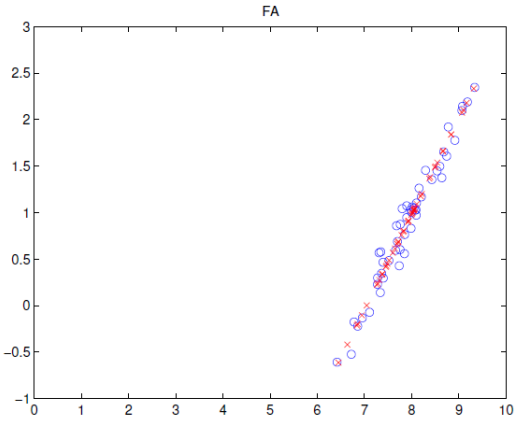
Results for the 2D dataset





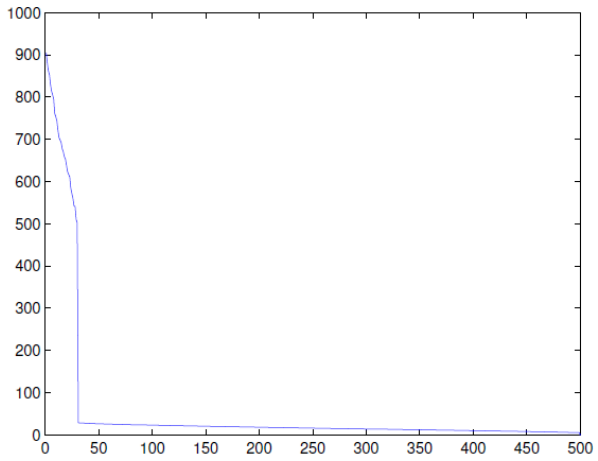






Results for the 1000D dataset

We used $d = 30$ since the singular values fall off sharply after this. See figure below:



This was the output.

```
>> q14
Reconstruction Errors:
Buggy PCA: 777.871447
Demeaned PCA: 272.546928
Normalized PCA: 273.140905
ASI: 272.546928
FA: 272.549605
```

Answers to Questions

1. When you SVD without demeaning on the dataset, you are find the best linear (as opposed to affine) subspace. The first principal component then will then be from the origin towards the data and other principal components will be orthogonal to this.
2. This is because the reconstructions are identical in both cases even if the representations are not.
3. No. Since in ASI we are directly minimizing this error criterion.

This is our implementation.

.....