GI07/M012
(Due 16 January 2017)

You may work in groups of up to two[1] if desired, only one report need be given for your group. You should produce a report about your results (including all requested plots). You will not only be assessed on the correctness/quality of your code/experiments but also on clarity of presentation. This includes but is not limited to insuring that your code is *well commented* and that each plot has a descriptive title, and a legend if necessary. Please submit 1) the coursework to the department office on the due date at 12:00 noon with the following linked coversheet `http://www.cs.ucl.ac.uk/fileadmin/UCL-CS/students/documents/cwsheet.pdf` 2) on moodle please submit a zip file containing your coursework as well as your source code.

DRAFT Notice

In order to give you a chance to begin work on this coursework, I am presenting you in the current form. The current problems will remain unaltered in the final version but I may decide to an additional problem which lead to rebalancing of points in the assignment.

1 K-means

The aim of this exercise is to study K-means clustering.

Notation: Bold denotes a vector e.g., $\mathbf{x}$; where x_i denotes ith component of vector $\mathbf{x}$ and $\mathbf{x}_i$ denotes the ith vector.

***K*-means overview:** K-means clustering is an algorithm to partitions objects, based on their features, into k classes. Suppose we have a data set $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$ our goal is then to give a disjoint partition into k sets $C_1, C_2, \ldots, C_k$. We will assume that we know apriori into how many clusters the data should be divided. Intuitively, we might think of a cluster as consisting of a group of points whose inter-point distances are small compared with the distances to points outside of the cluster. We can formalise this notion by introducing a set of "prototypes" $\mathbf{c}_1, \ldots, \mathbf{c}_k$, where each $\mathbf{c}$ is a "prototype" associated with one of the k clusters. The "prototypes" represent the centres of the clusters. The goal of K-means clustering is to find the cluster centres $\mathbf{c}_1, \ldots, \mathbf{c}_k$ that partition the data in such a way that the sum of the squared-distances of each data point to its closest centre is minimal. This can be expressed as the following (nonconvex) optimisation problem:

$$\underset{C_1,\ldots,C_k;\, \mathbf{c}_1,\ldots,\mathbf{c}_k}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{c}_i\|^2 \tag{1}$$

an optimization over all centers and partitions.[2]

The following well-known algorithm may be used to find a local minimum of (1).

Inputs:	Data: $X := (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\ell) \subset \mathbb{R}^n$ Number of classes: k
Initialization:	Choose random centers $\mathbf{c}_1 \ldots, \mathbf{c}_k$
Algorithm:	1. **for** $i = 1, \ldots, k$ **do** $C_i = \{\mathbf{x} \in X \mid i = \operatorname{argmin}_{1 \le j \le k} \|\mathbf{c}_j - \mathbf{x}\|^2\}$ 2. **for** $i = 1, \ldots, k$ **do** $\mathbf{c}_i = \operatorname{argmin}_{\mathbf{z} \in R^n} \sum_{\mathbf{x} \in C_i} \|\mathbf{z} - \mathbf{x}\|^2$ 3. **While** not converged **go to** step 1.

Figure 1: k-means algorithm

For your convenience, we sketch an implementation in Matlab.

[1]Affiliates students should work alone.

[2]This could be written as strictly an optimization over only the centers or the partitions. Why?

Method for k-means: First, assign some initial random position to the centres $\mathbf{c}_1, \dots, \mathbf{c}_k$. A typical choice is to choose the initial centres to be "centered" on k of the data points chosen at random without replacement; please use this technique to choose your initial centres in your code. Given a data set $\mathbf{x}_1, \dots, \mathbf{x}_\ell$ and a k number of clusters, create a set of binary indicator variables $r_{ij} \in \{0, 1\}$ describing which cluster C_j the data point $\mathbf{x}_i$ belongs to. If $\mathbf{x}_i$ is assigned to cluster j, then $r_{ij} = 1$. Otherwise, $r_{is} = 0$ for $s \neq j$.

Next, the k-means algorithm proceeds by alternating between two steps: assignment and update. In the assignment step, we assign each data point to the closest cluster centre. More formally, this can be expressed as:

$$r_{ij} = \begin{cases} 1 & \text{if } j = \operatorname{argmin}_{1 \le s \le k} \|\mathbf{x}_i - \mathbf{c}_s\|^2 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

In the update step, we recompute the cluster means to be the new centres of the clusters:

$$\mathbf{c}_j = \frac{\sum_{i=1}^{\ell} r_{ij}\mathbf{x}_i}{\sum_{i=1}^{\ell} r_{ij}}, \qquad (j = 1, \dots, k) \tag{3}$$

The denominator is equal to the number of points assigned to cluster j and so $\mathbf{c}_j$ is the centroid (Why?) of all the data points that belong to that cluster. The assignment and update procedures are repeated until there is no further change in the assignments of the data points to the clusters.

1.1 Practical [25 pts intermediate]

1. Implement the k-means clustering algorithm. Your function will take as the input a series of data points and the number of clusters.

2. Test your algorithm on the 2-dimensional data set $S = s_1, \dots, s_\ell$, where $s \sim N(\boldsymbol{\mu}, \Sigma)$. Produce three sets of data consisting of 50 points each, where each data set has a different μ and σ^2. Your data sets S_1, S_2 and S_3 should have the following normal distributions : $S_1 \sim N((4,0), \left(\begin{smallmatrix} .29 & .4 \\ .4 & 4 \end{smallmatrix}\right))$, $S_2 \sim N((5,7), \left(\begin{smallmatrix} .29 & .06 \\ .06 & .09 \end{smallmatrix}\right))$ and $S_3 \sim N((7,4), \left(\begin{smallmatrix} .64 & 0 \\ 0 & .64 \end{smallmatrix}\right))$. *[code is provided in the file `http://www0.cs.ucl.ac.uk/staff/M.Herbster/GI07/week3/genData2.m`]* Plot the clustered data using a different colour for the points in each cluster. Use additional larger markers to indicate the centres of the clusters. Add a legend to your final plot. The legend should explain that the larger markers are cluster centres and the smaller markers are data points. Hint: when plotting the data, you may find the command *scatter* useful.

   Assess the accuracy of the clustering algorithm, and calculate the **optimistic clustering classification error** `[occe]`. Assume the output of your algorithm is a partitioning $C_1, \dots, C_k$ and the true partitioning is $S_1, \dots, S_k$. We define the **simple error** as follows:

   $$e := \frac{|\{\mathbf{x} | (\mathbf{x} \in C_1 \text{ and } \mathbf{x} \notin S_1) \vee \dots \vee (\mathbf{x} \in C_k \text{ and } \mathbf{x} \notin S_k)\}|}{\ell}, \tag{4}$$

   and then the `occe` is computed by minimising over all permutations of the indices of C thus

   $$\texttt{occe} := \min_{\mathbf{p} \in \mathcal{P}_k} \frac{|\{\mathbf{x} | (\mathbf{x} \in C_{p_1} \text{ and } \mathbf{x} \notin S_1) \vee \dots \vee (\mathbf{x} \in C_{p_k} \text{ and } \mathbf{x} \notin S_k)\}|}{\ell}. \tag{5}$$

   Thus when $k = 3$ we have that $\mathcal{P}_k = \{(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)\}$ and thus the `occe` may be computed by calculating the simple error six times each time with a different permutation of the cluster indices. Repeat the clustering 100 times and calculate the `occe` each time. Calculate the average and standard deviation of the `occe` for 100 runs. Submit a script with your measurements as well as the code you used to obtain the measurements.

3. The iris data set (available at: `http://archive.ics.uci.edu/ml/datasets/Iris`) is one of the best known data sets in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. Each data point has 4 attributes: sepal length, sepal width, petal length and petal width. (Note that the fifth column of the data set contains the class of the iris. You will need to remove this column before running the experiment.). Perform k-means clustering on this data set. Using the fifth column "class" of the original data set calculate the error. Repeat the clustering 100 times and calculate the mean and standard deviatiation of the `occe` error. Report your measurements.

1.2 Questions [15 pts, intermediate]

1. Argue that the centroid (3) is the minimizer of the sum of square distances (Figure 1 (Algorithm Step 2)).
2. Prove that k-Means converges in a *finite* number of steps.

1.3 Extensions [25 pts, hard]

1. (p,k)**-means:** Define the p-discrepancy $d_p : \Re^n \times \Re^n \to [0,\infty)$ for $p \in (1,\infty)$ as

$$d_p(\mathbf{c},\mathbf{x}) := \sum_{i=1}^{n} |c_i|^p - |x_i|^p - p(c_i - x_i)\,\mathrm{sign}(x_i)|x_i|^{p-1} \tag{6}$$

observe that when $p = 2$ the squared euclidean distance is recovered. Design a "(p,k)-means" algorithm for the objective

$$\underset{C_1,\ldots,C_k;\,\mathbf{c}_1,\ldots,\mathbf{c}_k}{\operatorname{argmin}} \sum_{i=1}^{k}\sum_{\mathbf{x}\in C_i} d_p(\mathbf{c}_i,\mathbf{x}) \tag{7}$$

you should i) argue that your algorithm converges to a local minima of (7), ii) produce an appropriate alternative to the update step (3), iii) for ease in algebra you may optionally assume that the components of the input are nonnegative i.e, $\mathbf{x} \in [0,\infty)^n$.[3]

2. k**-means segmentation:** Give an efficient (polynomial-time in k, n, and ℓ) algorithm to *segment* a sequence $\{\mathbf{x}_1,\ldots,\mathbf{x}_\ell\} \subset \mathbb{R}^n$ into k segments

$$\{i_0 + 1 = \overbrace{1,2,3,\ldots,i_1}^{\text{"segment 1"}}, \overbrace{i_1+1,\ldots,i_2}^{\text{"segment 2"}}, \overbrace{i_2+1,\ldots,i_k}^{\text{"segments \ldots"}} = \ell\}\,,$$

by finding the *global minima* of the following optimization problem,

$$\underset{i_1,i_2,\ldots,i_{k-1};\,\mathbf{c}_1,\ldots,\mathbf{c}_k}{\operatorname{argmin}} \sum_{j=1}^{k}\sum_{p=i_{j-1}+1}^{i_j} \|\mathbf{x}_p - \mathbf{c}_j\|^2 \tag{8}$$

which is an optimization over centers $\mathbf{c}_1,\ldots,\mathbf{c}_k \in \mathbb{R}^n$ and all segmentations where a vector $\mathbf{i} \in \mathbb{N}_\ell^{k-1}$ with $0 < i_1 < i_2 \cdots < i_{k-1} < \ell$ determines a segmentation. This differs from the usual k-means which optimizes over all *partitions.* Give an argument that your algorithm is both correct and polynomial. No credit will be given for algorithms without an argument.

2 Principal Components Analysis

The aim of this exercise is to implement Principal Components Analysis or PCA.

PCA overview: PCA is a method of finding of finding an "optimal" linear feature map $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^k$. relative to a set of data $X = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m \subset \mathbb{R}^n$ by projecting it into the subspace spanned by the first k eigenvectors of the covariance matrix of the data for some $k < n$, where n indicates the dimensionality of the data. The new coordinates are known as the *principal coordinates* with the eigenvectors referred to as the *principal axes.* Assuming that the data is centred, PCA can be expressed as the following optimisation problem:

$$\underset{P}{\operatorname{argmin}} \sum_{t=1}^{m} \|\mathbf{x}_t - P\mathbf{x}_t\|^2, \quad \text{where } P \text{ is an } n \times n \text{ rank } k \text{ projection matrix} \tag{9}$$

PCA Algorithm: For simplicity we assume that the data is *centered.* That is each column of the $m \times n$ data matrix X has a mean of zero. This may be accomplished by computing the mean of each column and subtracting the column mean from each element in the column so that now $X'\mathbf{1} = \mathbf{0}$.
After the data has been centred, the PCA algorithm proceeds as follows:

We will not prove, but observe that the projection matrix[4] P in equation (9) is the sum of the outer products of the top k eigenvectors $P = \sum_{i=1}^{k} \mathbf{u}_i\mathbf{u}_i'$.

[3]For an unmarked extension, now consider the limiting case as $p \to 1$.
[4]A square matrix is a projection matrix iff $P = P^2$.

Inputs:	Centered Data ($m \times n$): $X := (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m) \subset \mathbb{R}^n$ Number of reduced dimensions: k
Algorithm:	1. Compute covariance matrix $C = \frac{1}{m} X' X$ 2. Compute Eigensystem of C, find $(\mathbf{u}_1; \lambda_1), \ldots, (\mathbf{u}_n; \lambda_n)$ such that $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_n$ and $C\mathbf{u} = \lambda \mathbf{u}$. 3. Compute the feature map ($k \times n$ matrix) $\boldsymbol{\phi}_k = \begin{pmatrix} \mathbf{u}_1' \\ \vdots \\ \mathbf{u}_k' \end{pmatrix}$
Return:	Transformed data $\tilde{X} = X \boldsymbol{\phi}_k'$

Figure 2: PCA algorithm

2.1 Exercises [40 pts, intermediate]

1. Implement PCA. Your function will take as the input a series of data points and the number of new principal components.

2. Implement the K-means algorithm. Your function should take as the input the data and the number of clusters, and then return the value of the objective function and a "clustering."

3. The iris data set (available at: http://archive.ics.uci.edu/ml/datasets/Iris) is one of the best known data sets in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. Each data point has 4 attributes: sepal length, sepal width, petal length and petal width.

   Apply PCA to the iris data set when $k = 1, 2, 3, 4$. Perform K-means clustering on the thus obtained 4 datasets as well as the untransformed dataset. Using the fifth column "class" of the original data set assess the accuracy of K-means clustering in each of the 5 data sets. In each case, calculate the `occe` and the value of the objective function $E_X(C_1, \ldots, C_k;\ \mathbf{c}_1, \ldots, \mathbf{c}_k)$. Recall that the objective is:

$$E_X(C_1, \ldots, C_k;\ \mathbf{c}_1, \ldots, \mathbf{c}_k) = \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{c}_i\|^2 .$$

   Repeat the clustering 100 times for *each of the 5 cases* discussed above then report on the following.

   For each of the five cases:

   (a) Give the 3 smallest "`occes`" (along with computed value of the objective)

   (b) Give the mean and standard deviation of the "`occes`" and the "objectives."

   (c) Give a bar chart, where the `occes` is plotted as a function of the *rank* of the objective function. The rank is the "sorted" value of the "objectives." For example, if you run the experiments 100 times you will have pairs of the the following form,

$$(128.5, .0.2), (77.02, .01), (105.33, .25), \ldots$$

   where the first coordinate is the value of the objective and the second is the `occe`. To plot versus rank we sort via the objective in ascending order thus if we had only 3 runs we would then have the following data

$$(3, .0.2), (1, .01), (2, .25)$$

   i.e. we have replaced the value of the objective with the rank of the objective. Now produce a bar chart by plotting a bar for each run with height proportional to the run's `occe`.

4. **Visualization:** Visualize the clustered data when the dimensionality is reduced to 2 dimensions and to 3 dimensions. Use a different marker for each of the clusters. Create one plot for the 2-dimensional case and one for the 3-dimensional case. Use a different, larger marker for the cluster centres.

2.2 Computing the `occe` [35 pts, challenge]

An apparent drawback of the `occe` is that compute naively requires $k!$ time for k classes. Design an a polynomial time algorithm to compute the `occe`. Give an argument that your algorithm is both correct and polynomial. No credit will be given for algorithms without an argument.

3 Kernel perceptron (Handwritten Digit Classification)

Introduction: In this exercise you will train a classifier to recognize hand written digits. The task is quasi-realistic and you will (perhaps) encounter difficulties in working with a moderately large dataset which you would not encounter on a "toy" problem.

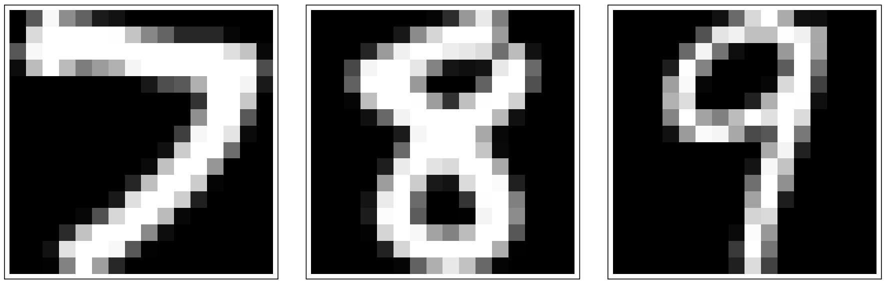

Figure 3: Scanned Digits

You may already be familiar with the perceptron, this exercise generalizes the perceptron in two ways, first we generalize the perceptron to use *kernel* functions so that we may generate a nonlinear separating surface and second, we generalize the perceptron into a majority network of perceptrons so that instead of separating only two classes we may separate k classes.
Adding a kernel: The *kernel* allows one to map the data to a higher dimensional space as we did with basis functions so that class of functions learned is larger than simply linear functions. We will consider a single type of kernel, the polynomial $K_d(\mathbf{p}, \mathbf{q}) = (\mathbf{p} \cdot \mathbf{q})^d$ which is parameterized by a positive integer d controlling the dimension of the polynomial.
Training and testing the kernel perceptron: The algorithm is *online* that is the algorithms operate on a single example $(\mathbf{x}_t, y_t)$ at a time. As may be observed from the update equation a single kernel function $K(\mathbf{x}_t, \cdot)$ is added for each example scaled by the term α_t (may be zero). In online training we repeatedly cycle through the training set; each cycle is known as an *epoch*. When the classifier is no longer changing when we cycle thru the training set, we say that it has converged. It may be the case for some datasets that the classifier never converges or it may be the case that the classifier will *generalize* better if not trained to convergence, for this exercise I leave the choice to you to decide how many epochs to train a particular classifier. The algorithm given in the table correctly describes training for a single pass thru the data (*1st epoch*). The algorithm is still correct for multiple epochs, however, explicit notation is not given. Rather, latter epochs (additional passes thru the data) is represented by repeating the dataset with the $\mathbf{x}_i$'s renumbered. I.e., suppose we have a 40 element training set $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_{40}, y_{40})\}$ to model additional epochs simply extend the data by duplication, hence an m epoch dataset is

$$\underbrace{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{40}, y_{40})}_{\text{epoch 1}}, \underbrace{(\mathbf{x}_{41}, y_{41}), \ldots, (\mathbf{x}_{80}, y_{80})}_{\text{epoch 2}}, \ldots, \underbrace{(\mathbf{x}_{(m-1)\times 40+1}, y_{(m-1)\times 40+1}), \ldots, (\mathbf{x}_{(m-1)\times 40+40}, y_{(m-1)\times 40+40})}_{\text{epoch m}}$$

where $\mathbf{x}_1 = \mathbf{x}_{41} = \mathbf{x}_{81} = \ldots = \mathbf{x}_{(m-1)\times 40+1}$, etc. Testing is performed as follows, once we have trained a classifier $\mathbf{w}$ on the training set, we simply use the trained classifier with only the *prediction* step for each example in test set. It is a mistake when ever the prediction $\hat{y}_t$ does not match the desired output y_t, thus the test error is simply the number of mistakes divided by test set size. Remember in testing the *update* step is never performed.

	Two Class Kernel Perceptron (training)
Input:	$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\} \in (\mathbb{R}^n, \{-1, +1\})^m$
Initialization:	$\mathbf{w}_0 = \mathbf{0}$ $(\alpha_0 = 0)$
Prediction:	Upon receiving the tth instance $\mathbf{x}_t$, predict $\hat{y}_t = \text{sign}(\mathbf{w}_t(\mathbf{x}_t)) = \text{sign}(\sum_{i=0}^{t-1} \alpha_i K(\mathbf{x}_i, \mathbf{x}_t))$
Update:	`if` $\hat{y}_t = y_t$ `then` $\alpha_t = 0$ `else` $\alpha_t = y_t$ $\mathbf{w}_{t+1}(\cdot) = \mathbf{w}_t(\cdot) + \alpha_t K(\mathbf{x}_t, \cdot)$

Generalizing to k classes: We may generalize a classification algorithm for two classes to k as follows.[5] Suppose we have a two-class classifier (2-classifier) algorithm which in addition to predicting a class also gives a magnitude (κ) called a confidence which loosely correponds to an estimate of a degree of belief that an example is a "positive" example. We proceed to build the k-classifier as follows,

1. Train k 2-classifiers ($\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(k)}$).
   - Each 2-classifier is trained as follows, for classifier $\mathbf{w}^{(i)}$, all of the examples with desired ouput class i are labeled as positive examples and other examples are labeled as negative examples.
   - For example, suppose we had a dataset with 100 item dataset spread evenly over 10 classes, then each classifier $\mathbf{w}^{(i)}$ will be a trained on a 100 item dataset with 10 positive examples and 90 negative examples.
2. We may identify the confidence κ of the classifier $\mathbf{w}$ on pattern $\mathbf{x}$ with $\mathbf{w}(\mathbf{x})$.
3. Let $\kappa^{(i)}(\mathbf{x}) = \mathbf{w}^{(i)}(\mathbf{x})$ denote the confidence of classifier $\mathbf{w}^{(i)}$ on pattern $\mathbf{x}$.
4. Predict class
   $$\operatorname*{argmax}_{1 \le i \le k} \kappa^{(i)}(\mathbf{x}), \tag{10}$$
   i.e., we simply predict with the classifier that is most confident in its prediction.
5. Hence our k-classifier is simply our k 2-classifiers used in conjunction with equation (10).

Choosing parameters (model selection): The general issue of choosing parameters for an algorithm is known as *model selection.* For this exercise *model selection* is relatively simple in that only the dimension of the kernel[6] (d) need be chosen. A straightforward strategy would be to search through the set of parameters (for this problem there is only d) and to choose those parameters which lead to minimum training error. However, as seen in earlier exercises this might lead to *overfitting*, i.e, the training error is much smaller than the test error. To solve this problem we may subdivide the training set into sets T and V. For a given of parameters (d) we may train on the set T then measure our performance on the validation set V. We then choose as our parameters the parameters corresponding to the classifer which had the best performance (fewest mistakes) on V. Then you may retrain your classifier on the dataset $T + V$ with the chosen parameters. This is known as *hold out* method. For this exercise, set T to be 2/3 of the training set and let V be the remaining third. **Optional:** You may devise your own approach to choosing the parameter d, such as using *cross-validation*, however, do not *cheat* by looking at the performance on the test set.

3.1 Exercises [40pts intermediate]

I'm providing you with mathematica code for a 3-classifier and a demonstration on a small subset of the data. First, however, my mathematica implementation is flawed and is relatively inefficient for large datasets. One part of your goal is improve my code so that it can work on a larger datasets, the mathematical logic of the algorithm will not change, however either and/or the program logic and data structures will need to change. Also, I suspect that it will be considerable easier to implement sufficiently fast code in Matlab rather than Mathematica. Second, you will use *hold out* to decide which degree polynomial should be chosen for your "test" classifier, you will report your error with this classifier.
Files: From `http://www0.cs.ucl.ac.uk/staff/M.Herbster/GI07/week5/`, you will find files relevant to this assignment. These are,

`hw5dig.nb`	demo mathematica code
`dtrain123.dat`	mini training set with only digits 1,2,3 (329 records)
`dtest123.dat`	mini testing set with only digits 1,2,3 (456 records)
`ziptrain.dat`	full training set with all digits (7291 records)
`ziptest.dat`	full testing set with all digits (2007 records)
`ziptrain.dat.gz`	compressed `ziptrain.dat`
`ziptest.dat.gz`	compressed `ziptest.dat`.

each of the data files consists of records each record contains 257 values, the first value is the digit, the remaining 256 values represent a 16×16 matrix of grey values scaled between -1 and 1. In attempting to understand the algorithms you may find it valuable to study the mathematica code. However, remember the demo code is partial (it does not address model selection, and though less efficient implementations are possible it is not particularily efficient.) Improving the code may

[5]This is an area which still has much active research. There are many known methods to do accomplish this conversion, here we give only one.

[6]I am ignoring the issue of the implicit parameter which is method for deciding how many epochs to train the classifier. You may formally include the number epochs to train a classifier as part of the model selection process. However, the process I suggest is to simply train a classifier to convergence or until the online training error is no longer decreasing or some such criteria.

require thought and observation of behavior on the given data, there are many distinct types of implementations for the kernel perceptron.

Assessment: In the report for this section You will not only be assessed on the correctness/quality of your experiment (e.g., sound methods for choosing parameters, reasonable final test error $< 9\%$) ii) but also on the clarity of presentation and the insightfulness of your observations. The report should contain at a minimum,

1. At least one paragraph discussing your implementation of the algorithms. In particular how the sum $\mathbf{w}(\cdot) = \sum_{i=0}^{m} \alpha_i K(\mathbf{x}_i, \cdot)$ was i) represented, ii) evaluated and iii) how new terms are added to the sum during training.
2. An indication of what degree classifier (d) was chosen by the *hold out* method.
3. A table of test errors (use `ziptest.dat`) for the trained classifiers (use `ziptrain.dat`) for $d = 2, \ldots, 7$.
4. Answers to the following questions (When giving a reason "why" include a *quantitative* justification).
   (a) What is the easiest digit (as a class) to recognize? Why? Hardest? Why?
   (b) What are the two digits (as classes) that are most often confused? Why?
   (c) In the test set what are the five most difficult to recognize (relative to your algorithm) scanned digits (as specific scanned digits). Why? Also give the "record numbers' and "print-outs" of these five digits.

3.2 Extension [20pts intermediate]

1. Repeat the above the experiments with a gaussian kernel, i.e.,

$$K(\mathbf{p}, \mathbf{q}) = e^{-c\|\mathbf{p}-\mathbf{q}\|^2},$$

   c the width of the kernel is now a parameter which must be optimized. Report on training and test errors.
2. Choose two additional algorithms to compare to the kernel perceptron. You do not need to implement these algorithms from scratch you may choose to use a package if desired. Compare the results between the three methods as "fairly" as possible for example, do **not** choose the parameters by hand for one method, then use a validation set on the other and cross-validation on the other. Discuss your results systematically.