Coursework II, GI07/M012

Klaudia Ludwisiak & Marika P. Paraskevopoulou

Department of Computer Science, University College London, UK

MSc. Computational Statistics and Machine Learning

January 15, 2017

1 Part 1: K- Means Clustering

1.1 Section 1

1.1.1 Overview and implementation of the K- Means Clustering Algorithm

K-Means Clustering is a method of classifying/grouping items into k pre-specified groups Hastie et al. (2001) Wikipedia (2016c) . The grouping is done by minimizing the sum of squared distances between data points and the k centroids. Where the centroids can be though of as the centers of mass of the data clusters. Centroids are initially chosen at random and updated at each iteration to find the global distance minimizing centroids, which define the final data clustering. K-Means belongs to the family of unsupervised learning algorithms as true data labels (clusters) are not known prior to the application of the algorithm. Even if knowledge of the true clusters exists, as is the case in this exercise, this information is not utilized by the algorithm. The K-Means algorithm was implemented as per the detailed description provided in the assignment handout. Thus ,this detail it will not be re-stated here. However, and overview of the potential shortcomings of K-means is provided as it is important to be aware of these when performing analysis. Further comments and the details of implementation can be found in the published code appended to this report (Section 4.1.1, 4.1.2 and 4.2 in Appendix)

- ***K-Means Weaknesses:***
  - For small datasets the initial centred choice will bear on the final clustering significantly. In other words, the global solution may be overlooked in favour a local distance minimising one.
  - The number of clusters, k, must be determined before K-Means is run. The algorithm cannot select the optimal amount of clusters itself. Thus, in real-life problems its may need to be tested for various instances of k before a feasible clustering is found.
  - All data points are assumed to have the same weight, so no information is passed about the relative importance of different data entries.
  - A mathematical drawback of the distance-minimisation approach means that results are sensitive to datapoint outliers. The existence of such outliers may significantly affect the position of the computes centroid. Another consequence of minimising based on distance is that the clusters produced will have rounded circular or spherical shapes, which may or may not accurately capture reality.

1.1.2 Testing the K- Means algorithm on random data

In this section we test our K-Means algorithm on data randomly sampled from three Normal distributions with distinct means and variance-covariance matrices. We will use 150 data points, 50 sampled from each of the distributions. The generated dataset naturally contains 3 clusters, each corresponding to data generated with a different Normal distribution. The knowledge of the true labels is used in conjunction with the labels predicted by the K-means algorithm to compute the optimistic clustering classification error (occe) as follows:

$$occe := min_{p \in P_k} \frac{|\{x | (x \in C_{p_1} \text{ and } x \notin S_1) \vee \cdots \vee (x \in C_{p_k} \text{ and } x \notin S_k)\}|}{\ell}$$

The true data as well as the clustered data are then plotted, and the figures obtained can be seen below (Figure 1 and Figure 2). It can be observed that the K- means algorithm has indeed worked and data was correctly assigned to 3 clusters. This was further tested by using the MATLAB in-built kmeans function, which produced similar results. However, upon visual examination of the true and predicted clustering, some miss-classification and overlap can be observed. Further inspection for different randomly generated data sets revealed that the outer data points between clusters 1 and 3 are often misclassified and sometimes cluster 2 also interferes with cluster 3. This inconsistencies are mirrored in by the OCCE error, found to have a mean of 0.095 and a standard deviation of 0.147. Indicating random data draws with relatively large OCCE misclassification errors of 0.25 or more could easily be observed

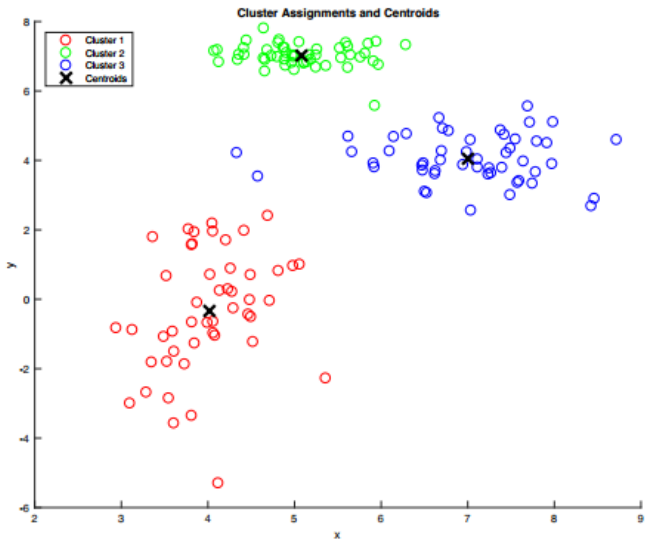

Figure 1: Clusters assignments and centroids.

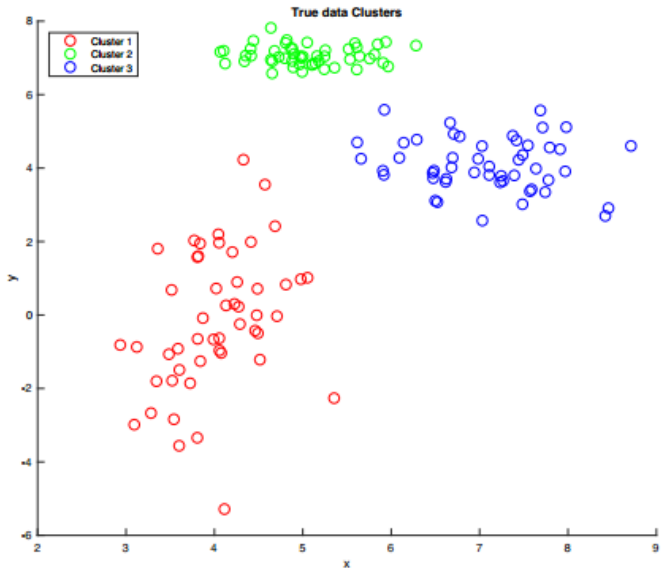

Figure 2: True data clusters.

1.1.3 Testing the K- Means algorithm on the Fisher's Iris data set

In the 1930's, botanists collected measurements on the sepal length, sepal width, peta length and petal width of 150 irises -50 from each of three species (setosa, versicolor, virginica). The measurements became known as Fisher's iris data and are widely known and often used to show the working of K-means. In this part of the question we will use to K-means algorithm to cluster the iris data and provide some statistical analysis of the results. First we perform a small exploratory analysis on the data set to get a feel for the expected results and dependencies between the different iris measurements. For this purpose Figure 3 is generated where it can be immediately observed that this data set lends itself to some clustering. However, data is mostly in 2 groups and it likely that only two distinct clusters will exist. Furthermore, we note that petal lengthened width are highly correlated.

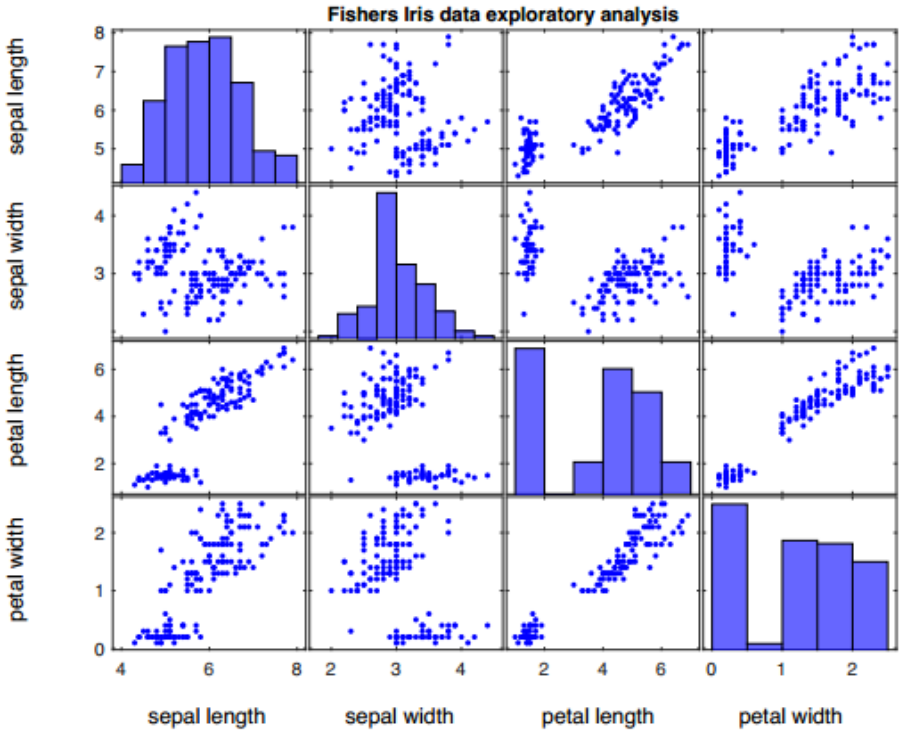

Figure 3: Scatter plot matrix.

When plotting it should be noted that the data is 4 dimensional (clustering performed in 4D), but can only be displayed in up to 3 dimensions. Therefore, two 3D plots are provided one omitting the petal length and another omitting the petal width (Figure 4). Viewing these in conjunction allows for full visual inspection the clustering results. As expected, one species of iris (setosa) forms a distinct cluster, whereas the remaining two clusters are artificial and could well be substituted by one larger cluster. Irises virginica and versicolor are not really separable without prior knowledge ofthe true data labels. This illustrates one of the previously outlined drawbacks of K-Means - the data set does not cluster into the pre-specified classes. After repeating the analysis a 100 times, the mean OCCE error on Iris data is 0.175 and the standard deviation of the OCCE error on Iris data is 0.130, this results are still satisfactory however, less so than for the randomly generated data in Part 1.1.2. Figure 5 displays the true Iris data classification based on known labels. Inspecting this confirms that the error arises predominantly due to the Iris virginica and versicolor neighboring clusters, where some overlap and miss-classification occurs. Improvements to this clustering can be achieved through the projection of Iris data is projected using principal component analysis (PCA), which is undertaken in Part 2

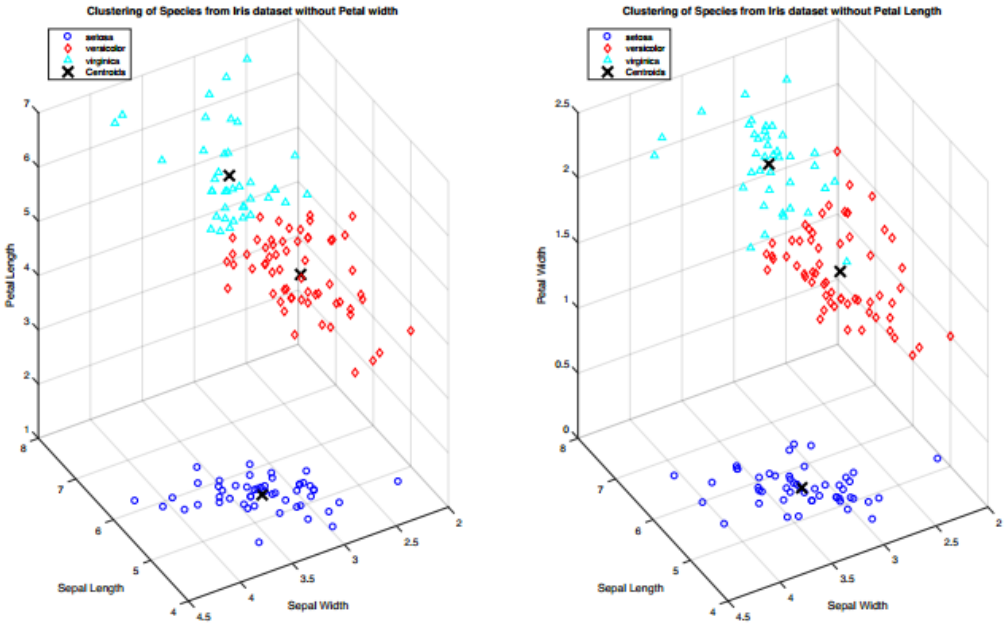

Figure 4: Clustering of species from Iris data set without petal width (left) and without petal length (right).

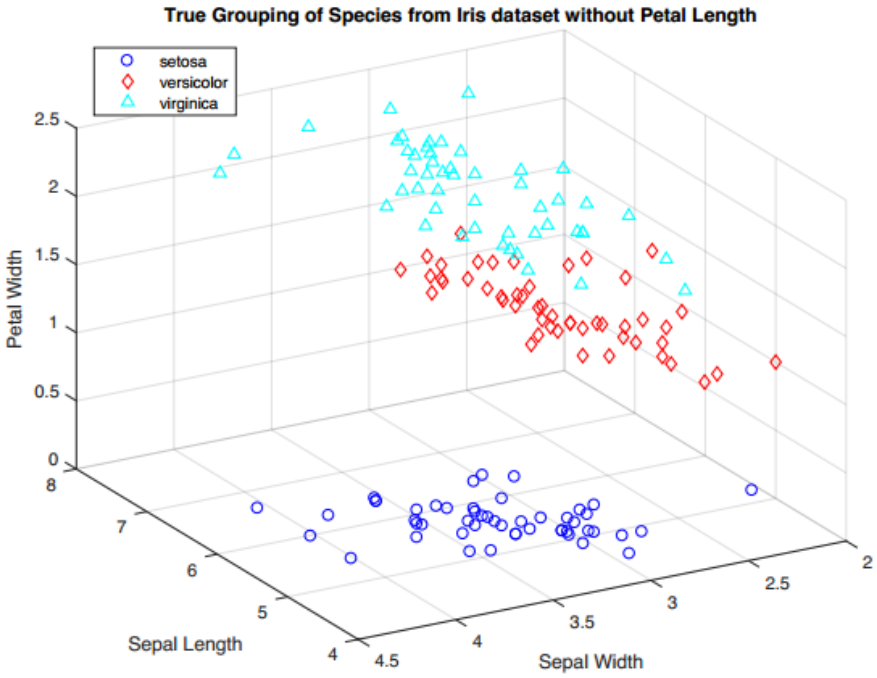

Figure 5: True grouping of species from iris data set without petal length.

PART 1 : K-MEANS

1.2 Question 1

let $c_j = \dfrac{\sum_{i=1}^{\ell} r_{ij} x_i}{\sum_{i=1}^{\ell} r_{ij}}$, $j = 1, \dots, k$

where $r_{ij} = \begin{cases} 1 & , \text{ if } j = \underset{1 \le s \le k}{\operatorname{argmin}} \|x_i - c_s\|^2 \\ 0 & , \text{ else} \end{cases}$

and $c_i = \underset{z \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{x \in C_i} \|z - x\|^2$. (1)

$$\frac{\partial}{\partial z} \sum_{z \in \mathbb{R}^n} \|z - x\|^2 = \sum_{x \in C_i} \frac{\partial}{\partial z} \|z - x\|^2 = \sum_{x \in C_i} 2(z - x)$$

set $\sum_{x \in C_i} 2(z - x) = 0 \Rightarrow \sum_{x \in C_i} z = \sum_{x \in C_i} x \Rightarrow \hat{z} = \dfrac{\sum_{x \in C_i} x}{\# C_i}$

Note: We have define $r_{ij} = 0, 1$ as an indicator function describing which cluster C_j the data point x_i belongs then in the Update step where we recompute the cluster mean to be the new centers of the clusters

$$c_j = \frac{\sum_{i=1}^{\ell} r_{ij} x_i}{\sum_{i=1}^{\ell} r_{ij}}$$

$\sum_{i=1}^{\ell} r_{ij} x_i$ ← # of x_n belongs in a specific cluster

$\sum_{i=1}^{\ell} r_{ij}$ ← # of 1 i.e: x_i belongs c_j.

$$\hat{z} = \frac{\sum_{x \in C_i} x}{\# C_i}$$ ← We sum over the clusters

Henceforth using the indicator r_{ij} we can write z as c_j. Thus $\hat{z} = c_j$, i.e: the centroid is the minimizer of the sum of Square distances (1).

PART 1: K-MEANS

1.2 Question 2

Overview: Suppose we have a data set $x_1, \dots, x_\ell$ our goal is then to give a disjoint partition into k sets $c_1, .., c_K$. Assume that we apriori know how many clusters the data should divide. Introduce prototypes $c_1, \dots, c_K$ where each c is a prototype associated with one of the k-clusters. The prototype represents the center of the cluster.

Goal: To find the $c_1, .., c_K$ centers that partition the data in such the sum of the Euclidean distances is minimal.

Non-Convex Optimization Problem: $\underset{c_1, .., c_K \; ; \; c_1, .., c_K}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{x \in c_i} \|x - c_i\|^2$

Method: first assign some initial random position to the centers $c_1, .., c_K$. Given the data set $x_1, .., x_\ell$ and a k-number of clusters, create a set of binary $r_{ij} = 1,0$ indicator variables describing which cluster c_j the data point x_i belongs to, i.e: if 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.

Assignment step: $r_{ij} = \begin{cases} 1, & \text{if } j = \underset{1 \leq s \leq k}{\operatorname{argmin}} \|x_i - c_s\|^2 \\ 0, & \text{else} \end{cases}$

Update Step: Recompute the cluster means to the new centers of the clusters $c_j = \dfrac{\sum_{i=1}^{\ell} r_{ij} x_i}{\sum_{i=1}^{\ell} r_{ij}} \; (j = 1, .., k)$

- Each iteration $(k\ell)$ operations

Proof of convergence:

- The k-means problem can be represented as the following non convex optimization problem;

$$\underset{C_1,\dots,C_k,c_1,\dots,c_k}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in C_i} \| x - c_i \|^2 \quad (1)$$

- We can consider the (1) as a cost function.

$$\text{cost}(c_1,\dots,c_k) = \underset{c_1,\dots,c_k}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in C_i} \| x - c_i \|^2$$

- From the definition of the k-means algorithm it is clear that each iteration decreases the cost function. Hence since this is done repeatidly. the loss is decreasing monotonically.

- Let $C_1^{old},\dots,C_k^{old}$ be the old clusters and $C_1^{new},\dots,C_k^{new}$ be the new clusters.

Hence $\text{cost}(C_1^{old},\dots,C_k^{old},\underset{\sim}{c}) \geq \min_{C_1^{old},\dots,C_k^{old}} (C_1^{old},\dots C_k^{old},\underset{\sim}{c}) = \text{cost}(C_1^{new},\dots,C_k^{new},\underset{\sim}{c})$

- Do the same for the centroids

Let $c_1^{old},\dots,c_k^{old}$ be the old centroids and $c_1^{new},\dots,c_k^{old}$ be the new centroids.

Hence $\text{cost}(c_1^{old},\dots,c_k^{old},\underset{\sim}{c}) \geq \min_{c_1^{old},\dots,c_k^{old}} (C_1^{old},\dots,C_k^{old},\underset{\sim}{c}) = \text{cost}(c_1^{new},\dots c_k^{new},\underset{\sim}{c})$

- Thus the cost is indeed monotonically decreases. Hence for n iterations (until convergence) the K-means will Convergence, to a local minimum. (Not Global). where $n < \infty$. finally $n < \infty$ because of the finite number of the updates.

2 Part 2: Principal Component Analysis (PCA)

2.1 Section 1

2.1.1 Overview and implementation of PCA

Principal component analysis (PCA) is a technique that applies an orthogonal transformation (feature map) to convert a correlated data set into a set of linearly uncorrelated principal components, of k dimensions (where k ¡ original data size). This is done by selecting the first k eigenvectors of the data, covariance matrix capturing the largest data variance in descending order. In practice, PCA can emphasize certain patterns in a data set and is often used to facilitate data exploration and visualization. A in-depth specification of the PCA algorithm is provided in the assignment handout and its implementation can be seen in the appended published code (Section 4.2.7 in Appendix). Thus, no further details will be given here, instead we will focus on result analysis and discussion. In this part of the question we not only coded the PCA but also tested on a random data set similar to the one in question 1.1.2 but with a higher dimensionally (150 by 6), generated with the function MYgenData3.m (Section 4.2.2 in Appendix) . PCA was used to reduce the dimensionally to 150 by 4 and results were then passed to the K-means algorithm for clustering. The results held a high occe error of 0.36 and the loss function has a value of over 700, signifying many points lie far from the centroid. The results are visualized below (Figure 6) and the source of the error becomes apparent, it is related to how the data was generated. Dimensionally was increased in such as way that the data lies on a plane and does not yield itself to orthogonal projection. However, the below results satisfy the aim of this exercise by proving that the PCA implementation was successful.

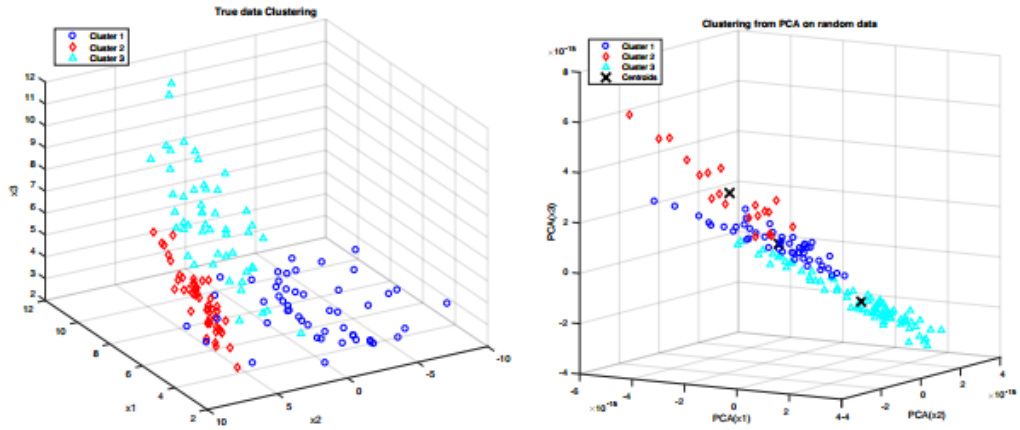

Figure 6: True data clustering (left), clustering from PCA on random data (right).

2.1.2 Re- implementation of K- Means algorithm

The question formulation of this section is slightly ambiguous as we have already implemented the K-means algorithm. Thus, it is our assumption that the discussion in Part 2.1 as well as in Question 1 addresses this task.

2.1.3 Error analysis performed on the Fisher's Iris data set

In this section of the exercise we turn to the Fishers Iris data set described in Part 1. This time we will attempt to cluster the Iris species after performing PCA on the data, while also testing for various values of the k parameter (k =1, 2, 3, 4). As outlined in Part 1 the occe error will be calculated for each instance. Additionally, we will inspect the value of the objective (loss) function, calculated as per assignment specification; as the norm square of the distance between x and the centroid summed over all

data in cluster and over all clusters. The process of applying PCA and fitting a K-means is also repeated a 100 times to obtain mean and standard deviation of the error measures.

- Part A: Find smallest 3 occe values and corresponding objective function. In asking for the 3 smallest occe values the question is ambiguous. We assume, it is asking for the 3 minimum non equivalent occe values (not vectors), in which case these can be found in the table below. If, instead the question is simply asking for the 3 lowest repeated vector values (containing the minimum error for each instance of parameter k) this will be 3 instances of the same min(occe) vector also included below for completeness.

	Occe	Objective
min1	0.08667	37.889
min2	0.10667	78.851
min3	0.11333	78.856

Table 1:

k=	0	1	2	3	4
Occe	0.1067	0.0867	0.1133	0.1067	0.1067
Objective	78.8514	37.8890	63.8199	75.3189	78.8514

Table 2:

The important conclusion from this part of the question is to examine which k parameters for PCA can yield improved K-means predictions. At first glance, the minimum clustering error is reached both in terms of min(occe) and the objective function for the PCA transformation of k=1. This is a trivial minimum as data is reduced to a 1D space and clustering (defined in 2D euclidean space) is meaningless. As the iris data set is 4D, at k=4 PCA is also redundant. Thus, it is good that the case where there is no PCA and the case where k=4 return the same errors, as this further validates our PCA procedure works correctly. The min(occe) for k=3 is the same as for the k=4 case, indicating no prediction improvement when reducing problem dimensionality from 4D to 3D. However, the corresponding objective function is slightly lower than for the k=4 case, thus after all there may be some improvement. Looking at min(occe) the k=2 case looks to be worse, but inspecting the value of the objective function indicates the opposite. It is likely that the k=2 case is in fact the best PCA transformation as will be elaborated in the sections that follow. The discrepancy between minimum occe and objective functions is alarming until one realises that this two approaches to measure error rely on fundamentally different concepts. The occe looks for the minimum of the difference of counts between true lables and all possible permutations of predicted labels. Whereas the objective function measures the minimum distance from the centroid of all points in a cluster. Therefore, it is not unreasonable that these two error measures give divergent indications especially when the minima are within less than one standard deviation away from the next best alternative. Discarding the k=1 solution it will thus be useful to further visualise results for k=2 and k=3 as is done in Part 2.1.4 below.

- Part B: The mean and standard deviation of the smallest 3 occe are: 0.102 and 0.014 respectively, whereas the corresponding mean and standard deviation of the objective function are: 65.199 and 23.651

- Part C: In this part of the exercise are asked to couple the occe and objective function results for each data instance and k value. These are then plotted as a bar chart displayed in Figure 7. It should be noted that although the objective function has been ranked as per question requirements, the bar plot command only admits the occe values and counts these (thus, we have a count of up to 150 on the x axis) assigning the rank therefore seams to have been redundant.

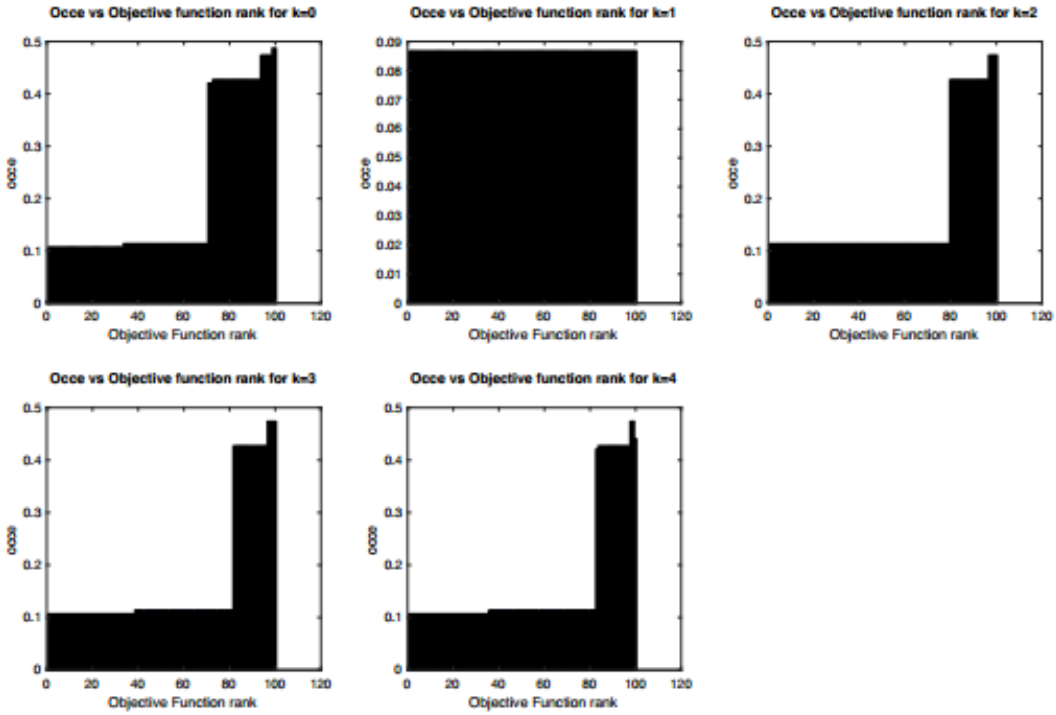

Figure 7: Occe vs objective function rank for k=0 (top left) until k=4 (bottom right).

2.1.4 Visualization and results analysis

PCA gives the relaxed solution of k-means clustering, thus it can provide improved results as comparing with the pure clustering case in Part 1. Considering the error analysis undertaken in prior sections, in this section we visualise results for PCA parameters k=2 and 3, which can be seen in Figures 8 and 9 below. K-Means captures spherically shaped clusters, hence for 2 clusters, a line connecting the two centroids will be the best 1-dimensional projection direction (1st PCA dimension). If we have 3 clusters, a 2-dimensional plane defined by the cluster centroids is the best 2-D projection, defining the first 2 PCA dimensions (K-means clustering, 2016). Therefore, it makes sense that for the 3 Iris clusters a principal component transformation into a 2D space produces the best data separation. This is also visible from figures below. By visual inspection, both PCA with k=2 and k=3 produce improved results to the case without PCA, however gong back to the error analysis in part 2.1.3 k=2 is marginally better. To conclude, PCA applied to the Iris data set improves K-Means clustering into 3 clusters marginally. However the Iris versicoloured and virginia species are still close together mainly because of the high correlation of petal width and length variables. One of the limitations of applying PCA is that it is sensitive to scale differences as it seeks to maximise variance. Therefore, if the Iris dataset contained some measurements in mm as well as the ones in cm the results could vary significantly. Similarly, results could be improved by standardising the data. Therefore room for rather improvement exists but it is deemed beyond the scope of this assignment.

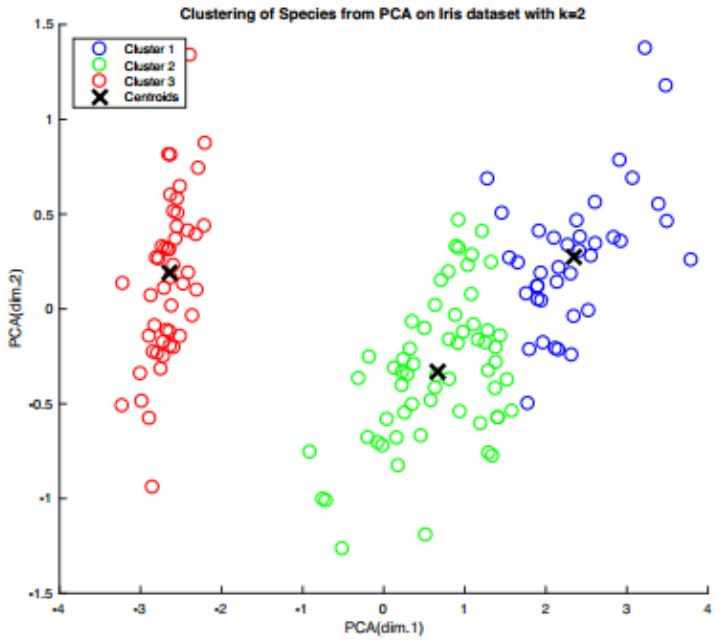

Figure 8: Clustering of species from PCA on iris data set with k=2 (Cluster 1 corresponds to iris virginica, Cluster 2 label depicts iris versicolor and Cluster 3 is iris setosa) .

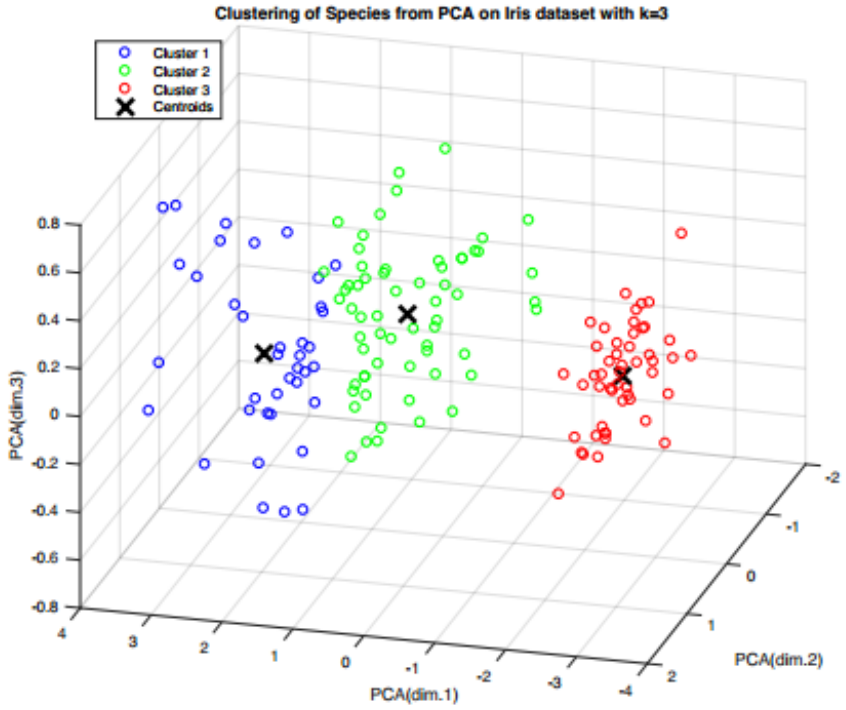

Figure 9: Clustering of species from PCA on iris data set with k=3 (Cluster 1 corresponds to iris virginica, Cluster 2 label depicts iris versicolor and Cluster 3 is iris setosa).

2.2 Section 2

- ***Computing the occe***

  The naive implementation of the optimistic clustering classification error (occe), such as implemented in questions 1 and 2 so far, has considerable computational expense. It requires k! iterations for k clusters, thus it is feasible to undertake for small datasets but does not scale well. In this section therefore, we will examine the possibility of improving the running time, giving an algorithm which can be run in polynomial time.

  This large time complexity arises predominantly because we are dealing with unsupervised learning problems, where the true data labels are unknown. Thus, requiring the cross examination of all possible label permutations before assigning the best configuration (minimising the occe). In order to propose a more scalable algorithm, the assumptions needs to be made that we are dealing with a supervised learning problem where true class labels are known. This has been the case in questions 1 and 2 so far, thus the assumption is not unreasonable. Furthermore, this is consistent with the definition of occe provided, which relies on the knowledge of true clustering classes (vector S). Thus, way we can turn the problem from a clustering to a classification problem, where a multitude of assignment (matching) algorithms exist for finding the loss minimising labels (Alt and Paul (1991)), (Hopcroft and Karp, 1973).

  Using concepts from graph theory, matching problems of this type can be framed as intending to find a maximum weight bipartite graph. A matching in a Bipartite Graph is a set of the edges chosen in such a way that no two edges share an endpoint. A bipartite graph: G=(V,E), describes a situation where the vertex set (V) can be partitioned into two disjoint sets of vertices (U1,U2), (here, representing the true and predicted labels, V=6) such that every edge (E) connects a vertex in one set (U1) to a vertex in the other set (U2) (Alt et al. 1991). Given the prefect matching condition, the number of edges will be half of the number of vertices.

  After obtaining a bipartite graph (matrix form) a variety of algorithms exist for optimising the weight matching, equivalent to finding the min(occe) class labels. Solving the assignment problem on a bipartite graph amounts to finding the maximum cardinality of this graph (i.e. number of non-trivial pairs) without having to search over all possible label combinations. Popular algorithms for solving the assignment problem include the Hungarian matching algorithm as well as adaptations of the Simplex and Auction algorithms (Assignment problem, 2016 Wikipedia (2016a)). However, we will adopt the Hopcroft Karp algorithm Wikipedia (2016b), which is known to be amongst the fastest in running time, and runs in $O(|V|^{2.5})$ for dense graphs Hopcroft and Karp (1973).

  Below we present a description of the Hopcroft-Karp algorithm based on an example of a set of 10 vertices split into two disjoint sets of 5 (as per Figure 10).

  Initialisation (see Figure 10 caption 1)

  To begin with, all vertices are free and no matches exist. In each iteration, the algorithm will look for an inclusion-maximal set of shortest vertex-disjunct augmenting paths. An augmenting path is a path that starts and ends in an unmatched node, alternating between edges outside and inside of the matching. Inclusion-maximal means that no other path can be added to the set, such that the paths don't share any vertices (i.e are vertex-disjunct) (TUM (2016)). The shortest augmenting paths can be found using Breadth-first (BFS) und Depth-first search (DFS). When no further augmenting paths can be found, the current matching is optimal and the algorithm terminates.

  - Phase 1: Initially, start with an arbitrary vertex and look for an inclusion-maximal set of shortest vertex- disjunct augmentation paths (here of length 1) Then, update matching using the augmentation path that has been found (Figure 10. Caption 2). Add unmatched edges of the augmentation path to the matching, and remove those contained in the matching. Those vertices that have been used in the current augmentation path, must no longer be used in any subsequent augmentation paths of the current phase. Thus, disregard them for the rest of the phase(Figure 10. Caption 3). Repeat this steps: looking for an augmentation path, matching, updating and excluding used vertices (Figure 10. Caption 4) until no further augmentation path can be found, and phase 1 ends (Figure 10. Caption 5) .
  - Phase 2: This phase is the repletion of the procedure described above, with the difference that now we will consider a shortest augmentation path of length 3. This should be vertex-disjunct with respect to the paths that were already found. In Figure 10. Caption 6 the bold path only

uses nodes that haven't been used previously and has minimal length. This is again used to update the matching and discard used vertices (Figure 10. Caption 7). The phase ends when no further vertex-disjunct augmentation paths of length 3 can be found (Figure 10. Caption 8).

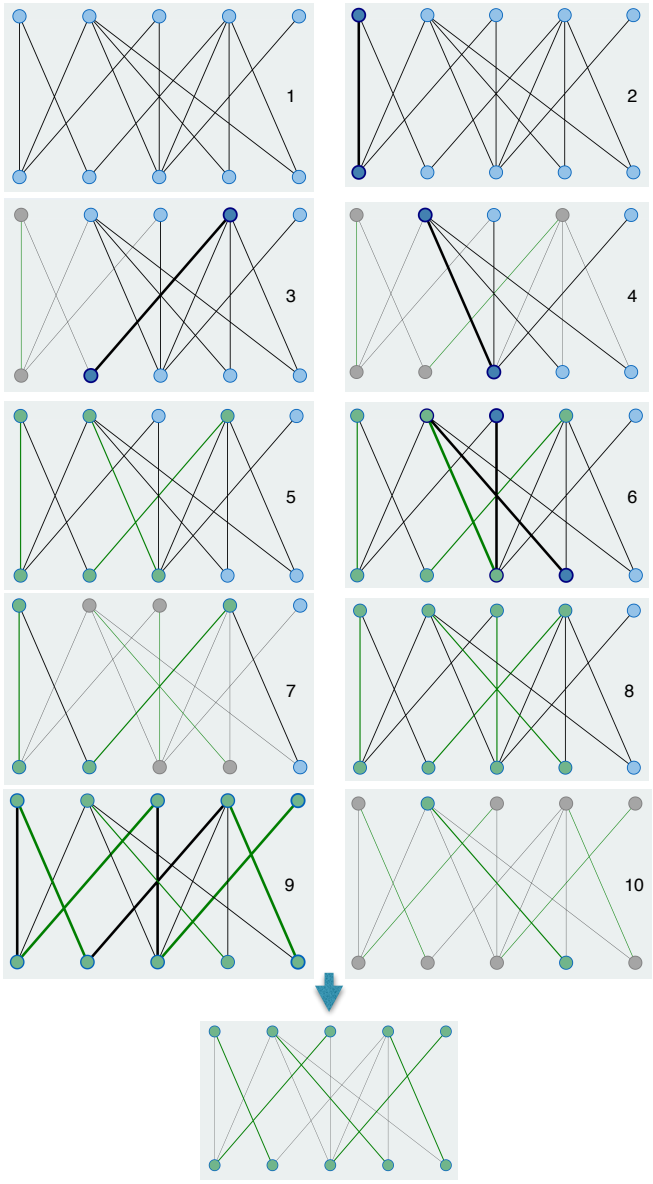

Figure 10: Visualization of the Hopcroft- Krap Algorithm.

- Phase 3: Repeat as per above the shortest augmentation path of length 7. Figure 10. Caption 9, depicts the situation where this augmentation path has been found, which is vertex-disjunct w.r.t the paths already found before. The next step (Figure 10. Caption 10), the used vertices are discarded and reveal the final augmentation path.
- Final result.

At this point, no further augmentation paths can be found and the current matching is optimal,

representing the final result of the algorithm.

Algorithm 1 Hopcroft and Karp

procedure HopcroftKarp algorithm (2016).
INPUT = Bipartite graph G($U_1 \cup U_2$, E)
BEGIN = Matching (M c E)
$M := \emptyset$
REPEAT (each iteration is a phase)
L:= Length of the shortest augmenting path (in given phase)
$P := \{P1, ..., Pk\}$*inclusion-maximal set of vertex-disjunct shortest augmenting paths of length L*
$M := M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$*every path found is used to enlarge M*
UNTIL $P = \emptyset$
RETURN M

notation: G($U_1 \cup U_2$, E) is a bipartite graph of vertices in sets U1 and U2 with edges E. M represents the matching between I=U1 and U2 at any point in time. P is the set of inclusionmaximal vertex-disjunct shortest augmenting paths of length L. Finally, k denotes the number ofpartitions in searching for the augmentation path (BFS/DFS) and is often called the number of layers. Above adapted from Hopcroft and Karp (1973), TUM (2016), and HopcroftKarp algorithm (2016).

3 Part 3: Kernel perceptron (Handwritten Digit Classification)

The perceptron algorithm is a class ofthe on- line algorithms for learning linear functions. The algorithm learns a thresholded linear function $sign < w, K(x) >$ in a kernel defined feature space in an on- line fashion making an update whenever a misclassified example is processed. The aim of the algorithm is to adapt his pattern function as quick as possible. This ability is also a measure of performance of the algorithm. Below we can see how this algorithm is summarized.

Algorithm 2 Two class kernel perceptron

procedure 2-k KernelPerceptron.
INPUT:$\{(x_1, y_1), \ldots, (x_n, y_n)\} \in (\Re^n, \{-1, 1\})^m$
initialization: $w_0 = 0, a_0 = 0$
prediction: Upon receiving the t^{th} *instance* x_t *predict* $\hat{y}_t = sign(\sum_{i=0}^{t-1} a_i K(x_i, x_t))$
UPDATE w,a as follows
if $\hat{y}_t = y_t$ **then**
    $a_t \leftarrow 0.$
else
    $a_t \leftarrow y_t.$
$w_{t+1}() \leftarrow w_t() + a_t K(x_t,).$

3.1 Section 1: Polynomial Kernel

3.1.1 Implementation of the Kernel Perceptron algorithm

We apply the Polynomial Kernel Perceptron algorithm, for classifying a data set consists of images of handwritten digits (0-9), so we have 10 classes one for each digit. Each image is a 16x16 pixel image. The data set consists on 7291 training images and 2007 testing images. As measure of performance we use the the accuracy of the prediction. An optimum algorithm will be an algorithm that it will perform all the errors and the earliest stages and then be much more accurate for the rest time online. For this scope we use the perceptron algorithm following similar philosophy with the assignment's description in an on-line fashion making an update whenever a misclassified example is processed. Henceforth for the given data in the following form $\{(x_1, y_1), \ldots, (x_n, y_n)\} \in (\Re^n, \{-1, 1\})^m$ if the weight vector is denoted by w_t after t updates then the update rule for the $(t+1)^{st}$ update when an example (x_t, y_t) is misclassified (wrong prediction) is given by $w_{t+1} = w_t + a_t K(x)$ and the corresponding dual update rule is simply $a_t = a_t + y_t$ and the weight vector is expressed as $w_t = \sum_{i=0}^{t-1} a_i K(x_i, x_t)$.

3.1.2 Indication of classifiers degree using the hold out method

In Table 3 we can see the test errors for $d = 2, \ldots, 7$, using the hold out method it is clear that d=4 is the optimum.

d:	TTE
2:	4.8011
3:	5.6241
4:	**4.5267**
5:	5.3498
6:	5.3498
7:	4.9383

Table 3: Total Test Errors (TTE) for different values of d, using a random split.

3.1.3 Test errors for the trained classifiers

In Table 4 we can see the test errors for the trained classifiers for $d = 2, \ldots, 7$, it is clear that d=4 is the optimum.

d:	TTE
2:	6.0082
3:	5.6790
4:	**5.0206**
5:	5.5967
6:	6.4198
7:	5.5144

Table 4: Total Test Errors (TTE) for different values of d, using a 2:1 split in the training set.

3.1.4 Digit recognition

- (a): easiest & hardest digit.

  In the Table 5 we represent the testing errors for each digit, based on this measure, we can assume that the most difficult to recognized digit is the 5, where it seems logical since it's shape can easily be confused by 3, 6 or even 8 and the easiest is the digit 1 (5 has the highest test error and 1 the smallest).

Digit	TE
0	0.3292
1	0.1646
2	0.4527
3	0.6173
4	0.7407
5	0.8642
6	0.4115
7	0.6173
8	0.4527
9	0.3704

Table 5: Test Errors (TTE) for each digit (for classifier's degree d=2).

- (b): most confused digit.

  Below we can see the confusion matrix of the predictions, Table 6. According to this table we can see the 5 and 3 are the most confused digits, one explanation for this could be their similar shapes.

0	1	2	3	4	5	6	7	8	9
$\surd$ 366	$\surd$ 344	$\surd$ 246	$\surd$ 275	$\surd$ 228	$\surd$ 207	$\surd$ 196	$\surd$ 187	$\surd$ 155	$\surd$ 165
5:1	4:2	3:1	0:1	2:2	0:2	4:3	2:3	5:2	4:2
6:1	8:1	5:2	2:2	7:4	**3:4**	5:3	3:1	6:1	7:3
7:2	9:1	6:1	**5:4**	8:2	8:2		4:1	7:1	
9:1			8:1	9:1			5:1		
			9:1						

Table 6: Confusion matrix, with -$\surd$ # of correct predictions- and -digit: # confused- (i.e; for the digit 0, second raw first column 5:1 the digit 0 is confused by 5, one time).

- (c): 5 most difficult to recognize digits.

  In Figure 11 we can see the five most difficult digits to recognize, their respective positions are 531, 530, 1780, 2390, 2142. For choosing them we based in a magnitude κ known as confidence which loosely corresponds to an estimate of a degree that n example is a "positive" example, henceforth $\kappa^i(x) = w^i(x)$ denotes the confidence of classifier w^i on pattern x.

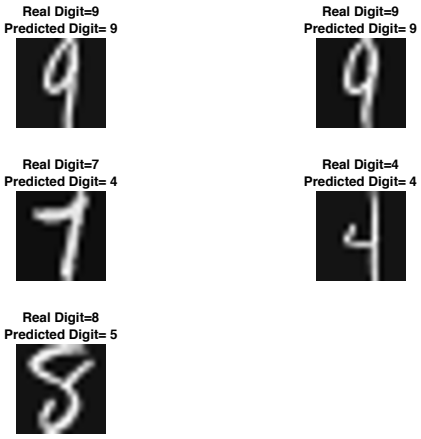

Figure 11: 5 most difficult to recognize scanned digits.

3.2 Section 2: Gauss Kernel

3.2.1 Indication of classifiers degree using the hold out method

In Table 7 we can see the test errors for $\tau = 0.01, \ldots, 0.07$, using the hold out method it is clear that $\tau = 0.02$ is the optimum.

τ:	TTE
0.01:	5.0754
0.02:	**3.9781**
0.03:	4.3896
0.04:	5.3498
0.07:	7.5446
0.08:	7.8189

Table 7: Total Test Errors (TTE) for different values of τ, using a random split.

3.2.2 Test errors for the trained classifiers

In Table 8 we can see the test errors for the trained classifiers for $\tau = 0.01, \ldots, 0.07$, it is clear that $\tau = 0.02$ is the optimum.

τ:	TTE
0.01:	5.4321
0.02:	**5.1852**
0.03:	6.6667
0.04:	8.0658
0.07:	8.3128
0.08:	9.5473

Table 8: Total Test Errors (TTE) for different values of τ, using a 66-33 split in the training set.

3.2.3 Digit recognition

- (a): easiest & hardest digit.

  In the Table 9 we represent the testing errors for each digit, based on this measure, we can assume that the most difficult to recognized digit is the 3, where it's shape can easily be confused by 5 or 8 and the easiest is the digit 1 (3 has the highest test error and 1 the smallest).

Digit	TE
0	0.4938
1	0.2058
2	0.5350
3	0.7407
4	0.6584
5	0.6584
6	0.2881
7	0.5761
8	0.5350
9	0.4938

Table 9: Test Errors (TTE) for each digit (for classifier's degree d=2).

- (b): most confused digit.

0	1	2	3	4	5	6	7	8	9
$\surd$ 364	$\surd$ 344	$\surd$ 247	$\surd$ 270	$\surd$ 227	$\surd$ 211	$\surd$ 195	$\surd$ 189	$\surd$ 156	$\surd$ 164
3:1	3:1	0:2	0:1	2:2	0:1	0:1	2:3	3:2	4:3
5:1	4:1	4:2	1:1	8:4	**3:6**	5:1	3:1	5:3	7:4
6:3	8:1	5:2	**5:2**	9:2			4:1	7:2	
8:1		6:1	8:2				5:1		
9:1			9:1				6:1		
							9:1		

Table 10: Confusion matrix, with -$\surd$ # of correct predictions- and -digit: # confused- (i.e; for the digit 0, second raw first column 3:1 the digit 0 is confused by 3, one time).

Below we can see the confusion matrix of the predictions, Table 10. According to this table we can see the 5 and 3 are the most confused digits, one explanation for this could be their similar shapes.

- (c): 5 most difficult to recognize digits.

  In Figure 12 we can see the five most difficult digits to recognize, their respective positions are 1356, 1780, 1399, 845, 428. For choosing them we based in a magnitude κ known as confidence which loosely corresponds to an estimate of a degree that n example is a "positive" example, henceforth $\kappa^i(x) = w^i(x)$ denotes the confidence of classifier w^i on pattern x.

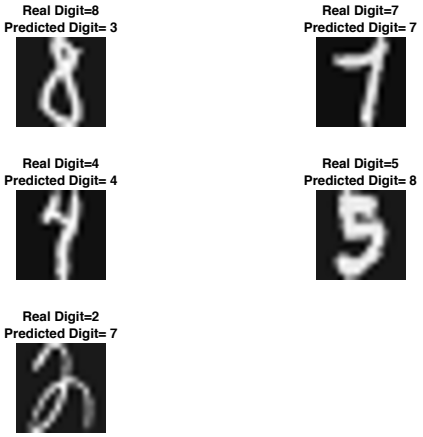

Figure 12: 5 most difficult to recognize scanned digits.

3.2.4 Comparison

In Section 1 and 2 of the Part 3 we apply the perceptron algorithm using two different kernels, a polynomial and a Gauss kernel. The main conclusion is that for the first kernel case kernel perceptron with degree around 4 is best for polynomial, where for the gaussian kernel, the precision parameter must be around $\tau = 0.02$. Although according to our analysis the two approaches are not 100% agree. We can also compare those two methods with the polynomial kernel of degree one which is the same as the normal perceptron. Testing accuracy at degree four is over 3.5-4 times better, where the test error for the normal perceptron is 15.2263, where in the case of the Gaussian kernel approach the testing accuracy is on average 5 times better, also between the polynomial and the gaussian kernel the performance of the polynomial kernel of degree 4 is slightly better. Finally one could also use other methods such us K-means for comparison.

4 Appendix

4.1 Matlab code

4.1.1 Question 1 Part 1.2

```
%% Question 1 Part 1.2 Test K means on specified dataset
clear all; close all;
%% Setup, initialise variables:
rng(7)   %ensures below function always generates same 100 ransom seeds
RNG=randperm(1000,100); %generate 100 random time seeds between 1 and 1000
OCCE100=zeros(100,1); %final occe error output vector for 100 iterations

%% Loop and plot:
for j=1:100

[data] = MYgenData2(RNG(1,j)); %ensures same 100 data sets always generated

k=3;
% note question does not explicitly state how many clusters we are after,
% I will assume 3 is this corresponds to the 3 data generating process

% generate augmented data matrix inc
D = zeros(size(data,1),k);
D(:,[1:2]) = data;
D([1:50],k) = 1;
D([51:100],k) = 2;
D([101:150],k) = 3;
S=D(:,k); %true labels vector

%fit kmenas model: c is the centroid and Y(:,end)=C labels vector
[Y c] = MyKmeansAd2(D(:,[1:2]),k);
C=Y(:,end); %predicted labels vector

%%%%%% Plot %%%%%%
% only plot for one data instance:
if j==1
    figure('position', [50, 50, 900, 550])
    hold on
    plot(D(Y(:,end)==1,1),D(Y(:,end)==1,2),'ro','MarkerSize',9)
    plot(D(Y(:,end)==2,1),D(Y(:,end)==2,2),'go','MarkerSize',9)
    plot(D(Y(:,end)==3,1),D(Y(:,end)==3,2),'bo','MarkerSize',9)
    plot(c(:,1),c(:,2),'kx',...
         'MarkerSize',15,'LineWidth',3)
    legend('Cluster 1','Cluster 2','Cluster 3','Centroids',...
           'Location','NW')
    title 'Cluster Assignments and Centroids'
    xlabel('x')
    ylabel('y')
    hold off
end
%% Calculate errors
    OCCE100(j,1)=MyOcceAd(S,C,k);
end

%answer:
aveOCCE=mean(OCCE100);
sdOCCE=std(OCCE100);
```

```
fprintf(' The mean OCCE error is %.3f\n',aveOCCE);
fprintf(' The standard deviation of the OCCE error is %.3f\n',sdOCCE);
```

4.1.2 Question 1 Part 1.3

```
%% Question 1 Part 1.3 Test K means on Iris dataset

clear all; close all;
%% Problem Setup
% load data:
load('fisheriris.mat');
% data already loads in desired format and a separate cell is loaded for
% species names/clustering
% load 1st 4 columns of data:
D = meas;
% convets speacies to numerical class
% from inspection, 1st 50 data points belong to the setosa group, the next 50
% to the versicolor and the last 50 to virginica. this simplifies
% assignment:
S = ones(150,1); % S is vector of true labels
S([51:100],1) = 2;
S([101:150],1) = 3;

%apply to results from part 1 and 2:
% as before:

% setup, initialise variables:
rng(7)   %ensures below function always generates same 100 ransom seeds
RNG = randperm(1000,100); %generate 100 random time seeds between 1 and 1000
OCCE100Ir = zeros(100,1); %final occe error output vector for 100 iterations
k = 3; % number of clusters

%% Loop and plot

for j = 1:100

%fit kmenas model: c is the centroid and Y(:,end)=C labels vector
[Y c] = MyKmeansAd2(D,k);
C = Y(:,end); %predicted lables vector


%%%%%% Plot %%%%%%
% only plot for one data instance:
if j == 1
    figure('position', [100, 1000, 1200, 700])
    %Improve plots to account for 4D data, make 2 times 3D plots
    symbol = {'bo','rd','c^'};
    subplot(1,2,1)
    for i = 1:k
        temp = find(C == i);
        plot3(D(temp,1),D(temp,2),D(temp,3),symbol{i});
        hold on
    end
    plot3(c(:,1),c(:,2),c(:,3),'kx','MarkerSize',15,'LineWidth',3)
    legend('setosa', 'versicolor', 'virginica','Centroids',...
        'Location','NW')
   hold off
   xlabel('Sepal Length');
```

```
   ylabel('Sepal Width');
   zlabel('Petal Length');
   title 'Clustering of Species from Iris dataset without Petal width'
   view(-120,30);
   grid on

   subplot(1,2,2)
    for i = 1:k
        temp = find(C == i);
        plot3(D(temp,1),D(temp,2),D(temp,4),symbol{i});
        hold on
    end
    plot3(c(:,1),c(:,2),c(:,4),'kx','MarkerSize',15,'LineWidth',3)
    legend('setosa', 'versicolor', 'virginica','Centroids',...
        'Location','NW')
   hold off
   xlabel('Sepal Length');
   ylabel('Sepal Width');
   zlabel('Petal Width');
   title 'Clustering of Species from Iris dataset without Petal Length'
   view(-120,30);
   grid on

end

%% Calculate error

%%%%%% Calculate errors %%%%%%

    OCCE100Ir(j,1)=MyOcceAd(S,C,k);
end

%answer:
aveOCCEIr = mean(OCCE100Ir);
sdOCCEIr = std(OCCE100Ir);

fprintf('The mean OCCE error on Iris data is %.3f\n',aveOCCEIr);
fprintf('The standard deviation of the OCCE error on Iris data is %.3f\n',sdOCCEIr);
```

4.1.3 Question 2 Part 1.1 and 1.2

```
%% Question 2 Part 1.1 & 1.2: Implement PCA and K means on random data
clear all; close all;
%% Setup
% take as input data generated by further modified MYgenData function
% this is done to test the PC algorithm (in function format below)
RNG = 7;  % time seed
k = 3;    % no of clusters/groups
kp = 4;   % PCA k parameter (no. of dimensions required after reduction)
N = 6;    % col dimension
M = 150;  % row dimension

[data] = MYgenData3(RNG,M,N,k); % ensures same data sets always generated

% generate augmented data matrix including labels (needed for part 2)
D = zeros(M,N+1);
D(:,[1:N]) = data;
```

```
D([1:50],N+1) = 1;
D([51:100],N+1) = 2;
D([101:150],N+1) = 3;
S = D(:,N+1); %true labels vector

% Center data:
% by taking mean of each column and substracting from column data values
dataC = data-mean(data);

%% Part 2.1.1 Implement PCA
% PCA implemented as a function (can be found at bottom of file)

% output function and test
dataPC = MyPCA(dataC,kp);

%% Part 2.1.2 Implement K-means
% using K-menas fucntion from Question 1 and the occe function (to get
% the objective/ loss function, which is teh minimum error)

[y,centr] = MyKmeansAd2(dataPC,k);
C = y(:,end); % last column of y contains the classification labels
occe = MyOcceAd(S,C,k);
% good but in question 2.1.3 objective function is defined as:
% EX(C1, . . . , Ck; c1, . . . , ck) = ? ? ?x ? ci?^2;
% (norm squear of differnece distance between x and centroid) sum over all
% x's in cluster and over all clusters.
% the following definition is implemneted through function MyLoss:

[EX mindist] = MyLoss(dataPC,centr,k);



fprintf('The objective function (minimum error)
for K-menas clustering on the PCA reduced data is
%.3f and the clustering vector can be found stored
under variable C \n',occe);
%% PCA function

function [dataPC] = MyPCA(dataC,kp)
% Purpose: perform Principle Component Analysis and return reduced data set
% Input: data, which needs to be centred and the number of new principal
% components (kp)
% Output: Reduced data set

    %%%% setup %%%%
    [M N] = size(dataC);
    %%%% test %%%%
    if round(mean(dataC)) ~= 0
        error('Error: need to input centered data');
        return;
    end
    %%%% Calculate covariance matrix %%%%
    COV = (1/M)*dataC'*dataC; %as per assignment hint
    %%%% Find eigensystem using Matlab command %%%%
    [vect, val] = eig(COV);
    % vector matrix in descending eigenvector order, such that need to extract
    %the last kp columns of this matrix, which will form PC feature map.
```

```
    %%%% Make feature map & reorder %%%%
    PCmap = vect(:, (end-kp+1):end);
    PCnew = zeros(size(PCmap));
        for i = 1:kp
            PCnew(:,i) = PCmap(:,kp+1-i);
        end
    PCmap = PCnew;
    %%%% Reduce data to PC form %%%%
    dataPC = dataC*PCmap;
end
```

4.1.4 Question 2 Part 1.3

```
%% Question 2 Part 1.3: Implement PCA and K means on Iris data
clear all; close all;
%% Setup
% Setup for Iris dataset:
% load data:
load('fisheriris.mat');
% data already loads in desired format and a separate cell is loaded for
% species names/clustering
% load 1st 4 columns of data:
D = meas;
% converts species strings to numerical class
% from inspection, 1st 50 data points belong to the setosa group, the next 50
% to the versicolor and the last 50 to virginica. this simplifies
% assignment:
S = ones(150,1); % S is vector of true labels
S([51:100],1) = 2;
S([101:150],1) = 3;

% Further problem setup:

% K-means setup:
k = 3; % no of clusters/groups

% PCA setup:
kp = [0:4]; % vector of PCA parameters (dim. reduction)
            % case when k=0 is when perform analysis on raw Iris data
            % without PCA conversion.

% center dataset:
dataC = D-mean(D);

% initiate occe and loss parameters
occe = zeros(100,length(kp));
EX = zeros(100,length(kp));

%% Part 2.1.3-loop 100 times:
% Loop to find occe and objective function:
for j=1:100
    %loop over the desired PC parameters (kp) calculating occe and objective
    %fun. for each iteration:
    for i=1:length(kp)

        if i==1 % case when kp=0,
            tempData = D; % set input argument for K-menas function to raw data
                          % function MyPCA below does not give error when
```

```
                        % kp=0 so this should work well.
        else
            % perform PCA with given kp parameter:
            tempData = MyPCA(dataC,(i-1));
        end

        % perform Kmeans for PCA or normal data with given k parameter:
        [y,centr] = MyKmeansAd2(tempData,k);
        C = y(:,end); % last column of y contains the classification labels

        % calculate occe error:
        occe(j,i) = MyOcceAd(S,C,k);
        EX(j,i) = MyLoss(tempData,centr,k);
    end
end
%% Part 2.1.3(a) Find smallest 3 occe values and corresponding objective fun.
%
% there are only 3 distinct values of the occe vector as the data has 3
% classes (k-menas k=3). Furthermore, this result is not surprising as we
% are looping over the same dataset and little variability is introduced.
% In asking for the 3 smallest occe values the question is ambiguous.
% I assume, it is  asking for the 3 minimum non equivalent occe values
% (not vectors), in which case these can be found in the vector below.
% If it is simply asking for the 3 lowest repeated values this will be 3
% instances of the first row of output below.

%Min3occe = svds(occe,3,'smallest');
Min3occe = sort(unique(min(occe)));
Min3occe = Min3occe(1:3)';

%corresponding corresponding objective function:
Min3Ex = zeros(3,1);
    for ii=1:3
        index = find(Min3occe(ii,1) == occe,1);
        Min3Ex(ii,1) = EX(index);

        clearvars index
    end
% store as matrix:
Min3vals=[ Min3occe,  Min3Ex];

% output as table:
Row = {'Min1';'Min2';'Min3'};
Occe = Min3occe;
Objective = Min3Ex;

disp('Three lowest occe values anc corresponding objective function:')
T = table(Occe,Objective,'RowNames',Row)

%% Part 2.1.3(b) Find corresponding mean and standard deviation.
meanMin3occe = mean(Min3occe);
sdMin3occe = std(Min3occe);

meanMin3Ex = mean(Min3Ex);
sdMin3Ex = std(Min3Ex);

fprintf('The mean and standard deviation of the smallest 3
occe are: %.3f and %.3f respectively, \nwhereas the
```

```
corresponding mean and standard deviation of the objective
function are: %.3f and %.3f \n',meanMin3occe,sdMin3Ex,meanMin3Ex,sdMin3Ex);

%% Part 2.1.3(c) Make chart of occe as a rank function.

SortOcce = zeros(size(occe));
SortEx = zeros(size(EX));

% below loop sorts and ranks occe and objective fun. as per question
% requirements:
for jj = 1:length(kp)

    % select corresponding columns of occe and objective function:
    tempEx = EX(:,jj);
    tempOcce = occe(:,jj);
    % concatenate:
    temp = [tempEx, tempOcce];
    % sort in ascending order:
    in = sortrows(temp);
    % store occe result:
    SortOcce(:,jj) = in(:,2);
    % converts objective fun. values to a rank and store result:
    uniqueEx = unique(in(:,1)); % vector of distinct loss function values
    for p=1:length(uniqueEx)
        index = find(in(:,1) == uniqueEx(p,1));
        SortEx(index,jj) = p; % overwrite loss fun.values with rank index
    end
    clearvars temp tempEx tempOcce in uniqueEx
end

% produce desired plot:
figure('position', [50, 500, 1000, 600]);
for pp = 1:length(kp)
    hold on
    subplot(2,3,pp)
    bar(SortOcce(:, pp));
    xlabel('Objective Function rank');
    ylabel('occe');
    title(sprintf('Occe vs Objective function rank %pp\n',pp));
end
% it should be noted that although the objective function has been ranked
% as per question requirements, the bar plot command only admits the occe
% values and counts these (thus, we have a count of up to 150 on the x axis)
% assigning the rank therefore seams to have been redundant.

%% Part 2.1.4 Visualise data for reduced dimensions
% in particular, when dimensions reduced to kp = 2 ,3.

% regenerate data:
D2 = MyPCA(dataC,2);
[Y2,centr2] = MyKmeansAd2(D2,k);
D3 = MyPCA(dataC,3);
[Y3,centr3] = MyKmeansAd2(D3,k);

% plot for 2D, k=2:
figure('position', [50, 500, 600, 500]);
hold on
plot(D2(Y2(:,end)==1,1),D2(Y2(:,end)==1,2),'ro','MarkerSize',9)
```

```
plot(D2(Y2(:,end)==2,1),D2(Y2(:,end)==2,2),'go','MarkerSize',9)
plot(D2(Y2(:,end)==3,1),D2(Y2(:,end)==3,2),'bo','MarkerSize',9)
plot(centr2(:,1),centr2(:,2),'kx',...
     'MarkerSize',15,'LineWidth',3)
legend('Cluster 1','Cluster 2','Centroids',...
       'Location','NW')
title 'Clustering of Species from PCA on Iris dataset with k=2'
xlabel('PCA(x1)')
ylabel('PCA(x2)')
hold off

% plot for 3D, k=3:

symbol = {'bo','rd','c^'};

figure('position', [50, 500, 600, 500])
    for i = 1:3
        temp = find(Y3(:,end) == i);
        plot3(D3(temp,1),D3(temp,2),D3(temp,3),symbol{i});
        hold on
    end
    plot3(centr3(:,1),centr3(:,2),centr3(:,3),'kx','MarkerSize',15,'LineWidth',3)
    legend('Cluster 1','Cluster 2', 'Cluster 3','Centroids',...
        'Location','NW')
   hold off
   xlabel('PCA(x1)');
   ylabel('PCA(x2)');
   zlabel('PCA(x3)');
   title 'Clustering of Species from PCA on Iris dataset with k=3'
   view(-120,30);
   grid on
```

4.1.5 Question 3 Part 1.4 and import data

```
close all
clear all
%% Question 3 Part 1.4 and data manipulation %%
% import the data
training_set = load('ziptrain.dat.gz');
test_set = load('ziptest.dat.gz');
% the y corresponds to the specific digits
% split y and x for the training and the test set
y_tr=training_set(:,1);
y_tst=test_set(:,1);
x_tr=training_set(:,[2:end]);
x_test=test_set(:,[2:end]);
% convert to -1,1
[digit] = unique(y_tr);
digit=digit';
y_train=(digit==y_tr);
y_train=double(y_train);
y_train(y_train==0)=-1;
[digit] = unique(y_tst);
digit=digit';
y_test=(digit==y_tst);
y_test=double(y_test);
y_test(y_test==0)=-1;
% split the training set into a train and a validation(test) set
```

```
n=round((2/3)*length(y_train));
% training data
X = x_tr(1:n, :);
y=y_tr(1:n,:);
yb = y_train(1:n, :);
% testing (validation) data
Xt = x_tr(n+1:end, :);
ytb = y_train(n+1:end, :);
ytb_digit=y_tr(n+1:end,:);
%d=0.02; %optimum for gauss kernel
d=4; %optimum for polynomial kernel
%type='gauss' %gauss type kernel
type='polynomial' %polynomial type kernel
[alpha] = perceptron_kernel(X, yb, d,20,type);
y_pred = -ones(length(Xt), 10);
conf_levels = zeros(length(Xt), 1);
K = kernel(X,Xt,d,type);
% predictions and confidence levels
for test= 1: length(Xt),
    weights= zeros(1,1);
    for train=1: length(X),
      weights = weights + alpha(train, :) * K(train, test);
    end
   [argmax, pos] = max(weights);
   conf_levels(test) = weights(pos);
   y_pred(test, pos) = 1;
end
% calculate the error as it described in HW
error = sum([ytb ~= y_pred])/length(ytb)*100;
% convert -1, 1 to digit
 a=(y_pred==1);
 a=double(a);
 digit_pred=[];
for j=1:size(a,1),
    for i=1:size(a,2),
        if a(j,i)==1
            digit_pred(j)=i;
        end
    end
end
digit_pred=digit_pred-1;
% find and plot the 5 most uncertain results
s=5;
[d, pos] = sort(conf_levels);
ytb_unc= ytb_digit(pos(1:5));
unc_pred = digit_pred(pos(1:5));
Xt_unc = Xt(pos(1:5), :);
d=16
figure;
hold on
for uncertain = 1:s,
subplot(3,2,uncertain)
colormap(gray(255));
train_x_reshape = (reshape(Xt_unc(uncertain, :), [d, d]))';
plot_image = imshow(imresize(train_x_reshape, d), []);
title({['Real Digit=', num2str(ytb_unc(uncertain))], ['Predicted Digit= ',
num2str(unc_pred(uncertain))]})
end
```

```
 axis square
 hold off
 ability_pr=confusionmat(ytb_digit,digit_pred);
 error
```

4.1.6 Question 3 Part 1.2,1.3 and 2.1

```
%Test error for each polynomial degree and/or value
%of parameter c on gauss Kernel and/or
%use the hold out method for choosing d and c
%choose d or c%
n=round((2/3)*length(y_train));
% training data
X = x_tr(1:n, :);
y=y_tr(1:n,:);
yb = y_train(1:n, :);
% testing (validation) data
Xt = x_tr(n+1:end, :);
ytb = y_train(n+1:end, :);
ytb_digit=y_tr(n+1:end,:);


%calculate the sum of errors- se for each d=2,3,..7
%type='gauss'
type='polynomial'
se=[];
;
for d=1:7 %for polynomial type kernel
%for d=0.01:0.01:0.06 %for gauss type kernel
[alpha] = perceptron_kernel(X, yb, d,20,type);

% Prediction part
% Initialise confidence and prediction vectors
y_pred = -ones(length(Xt), 10);
conf_levels = zeros(length(Xt), 1);

% Calculate kernel
K = kernel(X,Xt,d,type);

% predictions and confidence levels
for test= 1: length(Xt)
    weights= 0;
    for train=1: length(X)
      weights = weights + alpha(train, :) * K(train, test);
    end
   [argmax, pos] = max(weights);
   conf_levels(test) = weights(pos);
   y_pred(test, pos) = 1;
end

error = sum([ytb ~= y_pred])/length(ytb)*100;
se=sum(error)
end
%poly: 6.0082, 5.6790, 5.0206, 5.5967, 6.4198, 5.5144=>best d=4
%gauss; (0.01,0.02,..0.07) 5.4321, 5.1852, 6.6667, 8.0658, 8.3128,
%9.5473=>best=0.02
```

4.2 MATLAB functions

4.2.1 MYgenData2

```
function [ data ] = MYgenData2(RNG)

%contorl random number generation with seed
rng(RNG);
% generate data

A1=[0.5 0.2; 0 2];
u1=[4 0];

A2=[0.5 0.2; 0 0.3];
u2=[5 7];

A3=[0.8 0; 0 0.8];
u3=[7 4];

data = randn(150,2) ;
for i=1:50
data(i,:)    = u1' + A1 * data(i,:)' ;
end
for i=51:100
data(i,:)  = u2' + A2 * data(i,:)' ;
end
for i=101:150
data(i,:) = u3' + A3 * data(i,:)' ;
end
end
```

4.2.2 MYgenData3

```
function [ data ] = MYgenData3(RNG,M,N,k)
% generate random numbers
% Input:
% RNG
% M is row dimension of data
% N is column dimenstoon of data MUST BE EVEN
% k is number of clusters ie. distinct data sets to be produced
% k has to be smaller or equal 3


%contorl random number generation with seed
rng(RNG);
% generate data

A1=repmat([0.5 0.2; 0 2],N/2);
u1=repmat([4 0],1,(N/2));

A2=repmat([0.5 0.2; 0 0.3],N/2);
u2=repmat([5 7],1,(N/2));

A3=repmat([0.8 0; 0 0.8],N/2);
u3=repmat([7 4],1,(N/2));

bins=[round(M/k),2*round(M/k),3*round(M/k)];

data = randn(M,N) ;
```

```
for i=1:bins(1)
data(i,:)    = u1' + A1 * data(i,:)' ;
end
for i=(bins(1)+1):bins(2)
data(i,:)  = u2' + A2 * data(i,:)' ;
end
for i=(bins(2)+1):bins(3)
data(i,:) = u3' + A3 * data(i,:)' ;
end
end
```

4.2.3 MyDist2

```
function [posi] = MyDist2(X,c)
% Purpose: calculates distance between points in X and c, finds minimum and
% assigns the corresponding class label, retuning the clustering vector.
% Input:
% X is data matrix where rows are instances of observations
% c is matrix of centroids (where rows correspond to no of classes and
% columns to x,y coordinates)
% Output:
% Vector of cluster class


%setup:
d = zeros(size(c,1),1); %vector of distance to each of k clusters
posi = zeros(size(X,1),1);  %vector to corresponding class assignment

    for i=1:size(X,1) %loop over all data points
        for k=1:size(c,1) %loop over all clusters
            d(k,1)=sum((X(i,:)-c(k,:)).^2);
        end
        [~,pos]=min(d);
        posi(i,1)=pos;
    end

end
```

4.2.4 MyKmeansAd2

```
%% Question 1 Part 1.1 Implement K means Algorithm

function [y,centr] = MyKmeansAd2(X,k)
% Purpose: to classify vectors (data objects) into (k) classes by minimising
% the Euclidean distance between centroids and object points
% Input: data matrix X and cluster size k
% Output: original matrix augmented by the group classification column (at the end)

[N M] = size(X);

c = zeros(k,M); % initialise matrix of centroids

% chose initial values for centroid:

p = randperm(N,k);
% generate vector of random permutations with values from 1 to N, of length
% k. This will be use to index to data instances at random and initialise
% centroids.
```

```
for i = 1:k
    c(i,:) = X(p(i),:); %assign centroid values to random data points
end

while (1)  %statement terminates when break statement below is  reached

  % Assignment step- assign each data point to closest cluster centre
  % done using MyDist2 function:
  [clust] = MyDist2(X,c);         % calculate object-centroid distances
                                % take minimum and corresponding cluster
                                % thus make a new clustering.

  % Update step- compute the new centroids
    NEWc = zeros(k,M);
    for ii = 1:k
        temp = X(clust == ii, :); %select data in category
        NEWc(ii, :) = mean(temp);
    end
    c=NEWc;

  % Recalculate new cluster using new centroid values:
  NEWclust = MyDist2(X,c);

  % while loop termination condition:
   if clust == NEWclust
             % solution converges: end while loop /stop iterating
             % this only occurs when all distances are minimised
      break;
   end
end

%output is original matrix augmented by the group classification column:
y = [X, clust];
centr=c;

end
```

4.2.5 MyLoss

```
function [LossFun, minDist] = MyLoss(dataC,centr,k)
% function similar to MyDistance but now instead of returning cluster, min
% distance to cluster centroid is returned to calculate the value of the
% objective function.
% Input:
% dataC is data matrix where rows are instances of observations
% centr is matrix of centroids (where rows correspond to no of classes and
% columns to x,y coordinates)
% k is the number of clusters
% Output:
% Value of objective/loss function

d = zeros(k,1); % distance vector
minDist = zeros(size(dataC,1),1); % vector of min distances
LossFun=0; % initialise loss function variable

 for i=1:size(dataC,1) %loop over all data points
     for kk=1:k %loop over all clusters
         d(kk,1)=sum((dataC(i,:)-centr(kk,:)).^2);
```

```
    end
    [val, ~] = min(d);
    minDist(i) = val;
    LossFun=LossFun+val; %sum all min distances to get objective function
end
```

4.2.6 MyOcceAd

```
function [occe]=MyOcceAd(S,C,k)
% Function calculates the occe error for kmeans clustering algorithm
% it initalises a matrix of all possible cluster permitations based on k
% Input:
% S - True classification lables
% k - no of clusters
% Output: occe error

p=perms(1:k);%create matrix of possible lable permutations (6 by 3)

err=zeros(size(p,1),1);
temp=C;
len=size(C,1);

for i=1:size(p,1)
    temp(C==1)=p(i,1);
    temp(C==2)=p(i,2);
    temp(C==3)=p(i,3);
    err(i,1)=(sum(temp~=S))/len;
end
occe=min(err);
```

4.2.7 MyPCA

```
%% PCA function

function [dataPC] = MyPCA(dataC,kp)
% Purpose: perform Principle Component Analysis and return reduced data set
% Input: data, which needs to be centred and the number of new principal
% components (kp)
% Output: Reduced data set

    %%%% setup %%%%
    [M, ~] = size(dataC);
    %%%% test %%%%
    if round(mean(dataC)) ~= 0
        error('Error: need to input centered data');
        return;
    end
    %%%% Calculate covariance matrix %%%%
    COV = (1/M)*(dataC'*dataC); %as per assignment hint
    %%%% Find eigensystem using Matlab command %%%%
    [vect, ~] = eig(COV);
    % vector matrix in descending eigenvector order, such that need to extract
    %the last kp columns of this matrix, which will form PC feature map.

    %%%% Make feature map & reorder %%%%
    PCmap = vect(:, (end-kp+1):end);
    PCnew = zeros(size(PCmap));
        for i = 1:kp
            PCnew(:,i) = PCmap(:,kp+1-i);
```

```
        end
    PCmap = PCnew;
    %%%% Reduce data to PC form %%%%
    dataPC = dataC*PCmap;
end
```

4.2.8 Kernel

```
function [K] = kernel(X, Xt, d,type)
% KERNEL the kernel function
% usage: [K] = kernel(X, Xt, d, type)
% KERNEL type
switch type
 case 'polynomial'
% polynomilal
K = (X*Xt').^d;
  case 'gauss'
% gauss
  n= size(X,1);
  m = size(Xt,1);
  s = sum(X.^2,2);
  st = sum(Xt.^2,2);
  mtt = repmat(s,1,m);
  mttt = repmat(st',n,1);
  sq = mtt + mttt - 2*X*Xt';
  K = exp(-d*(sq));
end
```

4.2.9 perceptron_kernel

```
function [alpha] = perceptron_kernel(X, y, d,epoch,type)
% PERCEPTRON_KERNEL perceptron algorithm + kernel
% usage: [alpha] = perceptron_dual_kernel(X, y, d, epoch, type)
% ell: the number of training samples
% n: length of the input vector
  [ell n] = size(X);
% compute the Kernel matrix (using function kernel.m)
  K = kernel(X,X,d,type);
% the dual variables alpha
  alpha = zeros(ell,10);
% start repeat for loop
  a=0.2
% set a small possitive quantity for updating the alpha by
% setting zero if the prediction is correct and y ifelse
% we need a and a/2 for the logical operations for setting
% y in the case of not correct prediction
 for i= 1:epoch,
    for j = 1:ell,
        alpha(j,:) = alpha(j,:) + (y(j,:).*(a*((K(j,:)*(alpha))>0)-a/2)<0).*y(j,:);
      end
    end
end
```

References

Alt, H.; Blum, N. M. K. and Paul, M. (1991). *Computing a maximum cardinality matching in a bipartite graph in time.* Information Processing Letters, 37(4):237-240.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The elements of statistical learning: data mining, inference, and prediction.* Springer.

Hopcroft, J. and Karp, R. (1973). *An $n^{2.5}$ algorithm for maximum matchings in bipartite graphs.* SIAM J. Comput 2(4) 225-231.

TUM (2016). The hopcorft- karp algorithm. [Online; accessed 19-November-2016].

Wikipedia (2016a). Assignment problem — wikipedia, the free encyclopedia. [Online; accessed 27-October-2016].

Wikipedia (2016b). Hopcroftkarp algorithm — wikipedia, the free encyclopedia. [Online; accessed 19-November-2016].

Wikipedia (2016c). K-means clustering — wikipedia, the free encyclopedia. [Online; accessed 19-December-2016].