Russel Stuart Daries
UCABRSD

STUDENT

FIRST LETTER OF SURNAME D

PERSONAL TUTOR

Coursework Coversheet

This document is the assessed coursework coversheet for all Computer Science modules conducted at UCL for this academic year. Please do the following when submitting coursework:

- Staple a completed and signed copy of this form to every piece of assessed coursework you submit for modules in the Department of Computer Science.

- avoid the use of document containers such as cardboard or plastic covers, document wallets, ring binders or folders (unless otherwise stated by the lecturer concerned).

We do not wish to discourage students from discussing their work with fellow students and collaborating in solving problems. However you should avoid allowing the collaborative phase to approach too close to a final solution which might make it impossible for you to make your own distinctive intellectual contribution. The key point is that you must not present the results of another person's work "as though they were your own". http://www.ucl.ac.uk/current-students/study/plagiarism/

UCL has now signed up to use a sophisticated detection system (JISC Turn-It-In) to scan work for evidence of plagiarism, and the Department intends to use this for assessed coursework. This system gives access to billions of sources worldwide, including websites and journals, as well as work previously submitted to the Department, UCL and other universities

The collection point for "returned marked" coursework will be in a public area. If you wish to have your coursework returned privately. You MUST collect your coursework as soon as you are informed. □

Submission Details

Please ensure that the details you give are accurate and completed to the best of your knowledge.

Coursework Title	Lecturer
Coursework 2	Dr. Herbster

Module Code	Module Name	Deadline
COMPGI07	Mathematical Programming & Mathematical Models	13 / 02 / 2017

Declaration

I have read and understood the UCL and Departmental statements and guidelines concerning plagiarism.
I declare that:

1. This submission is entirely my own unaided work.
2. Wherever published, unpublished, printed, electronic or other information sources have been used as a contribution or component of this work, these are explicitly, clearly and individually acknowledged by appropriate use of quotation marks, citations, references and statements in the text.
3. In preparing this coursework, I discussed the general approach to take with the person(s) named below, however the content of the submission is my own work alone:

Person(s) with whom I have discussed this coursework: Hugo Philion

Signature: R.Daries

For Departmental Office Use

Date and Time Received	Initials or Stamp

COMPGI07: Programming & Mathematical Methods for ML Coursework 2

Due on Monday, January 16, 2017

Dr. Mark Herbster

Russel Daries: UCABRSD

Hugo Philion: UCABHPO

February 15, 2017

Contents

List of Figures

List of Tables

1 K-Means

(1.1.1)

The K-Means algorithm was implemented in MATLAB, as a function called *kMeans*, which took a series of data points and the number of clusters as data points. The implementation of K-Means algorithm can be found in the file called *kMeans.m* which was attached with this assignment. The function call for this algorithm can be seen in more detail as shown below.

```
%% K-Means Algorithm
function [centroids, indicator_variable] = kMeans(X,k)
```

Where:

- X: Input data points.
- k: Number of clusters
- indicator_variable: The indicator matrix $\boldsymbol{r}$.
- centroids: The centroids calculated $(\boldsymbol{c}_1, ...\boldsymbol{c}_k)$ given an input data series.

The implementation of K-Means can be seen in Appendix A.

(1.1.2)

The K-Mean algorithm created in 1.1.1 was then tested on a series of data sets namely S_1, S_2 and S_3 using the script *genData2.m*. Each one of the data sets had the following distributions:

1. S_1: $S_1 \sim N((4,0), (\begin{smallmatrix} 0.29 & 0.4 \\ 0.4 & 4 \end{smallmatrix}))$
2. S_2: $S_2 \sim N((5,7), (\begin{smallmatrix} 0.29 & 0.06 \\ 0.06 & 0.09 \end{smallmatrix}))$
3. S_3: $S_3 \sim N((7,4), (\begin{smallmatrix} 0.64 & 0 \\ 0 & 0.64 \end{smallmatrix}))$

The data S without the class labels can be seen in Figure 1. The plot of the original data sets namely S_1, S_2 and S_3 can also be seen in Figure 2 on the following page .

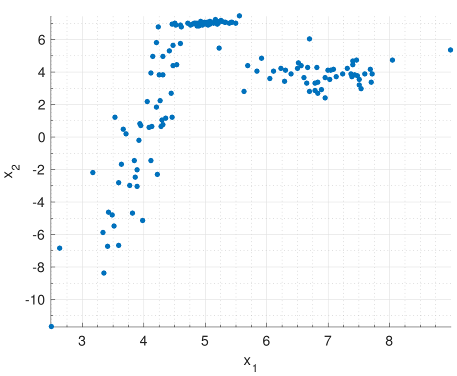


Figure 1: Unlabelled data from S.

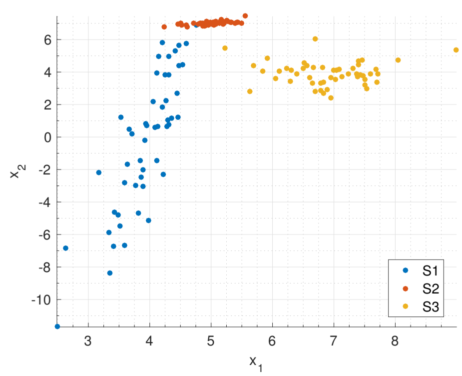


Figure 2: Labelled data sets from S.

The K-Means algorithm was then applied as can be seen in MATLAB *Q1_1.m* file found in Appendix A to cluster the input data and find the cluster centres. The resultant classification of the data can be seen in Figure 3 shown below. It can be seen when comparing Figure 2 and

1 K-Means [(1.1.2)] continued on next page...

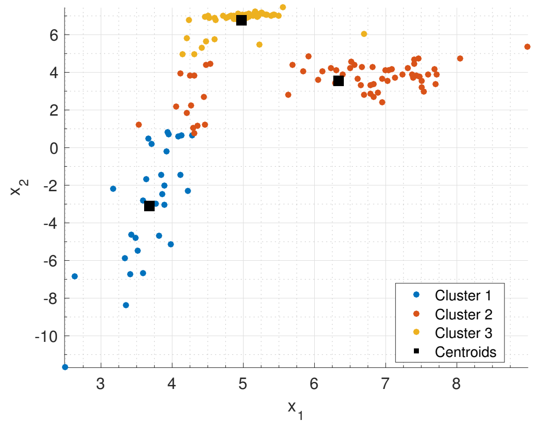


Figure 3: Data after K-Means algorithm applied to S.

3, there were some data points that were misclassified. However, overall the performance was sufficient.
Using the equations for the simple error e and optimistic clustering classification error *occe* as defined in the question page, 100 and 250 trials were computed in order to determine the mean *occe* and standard deviation.

$$e := \frac{|\{\boldsymbol{x}|(\boldsymbol{x} \in C_1 \, and \, \boldsymbol{x} \notin S_1) \vee \cdots \vee (\boldsymbol{x} \in C_k \, and \, \boldsymbol{x} \notin S_k)\}|}{l} \tag{1}$$

$$occe := \min_{\boldsymbol{p} \in P_k} \frac{|\{\boldsymbol{x}|(\boldsymbol{x} \in C_{p_1} \, and \, \boldsymbol{x} \notin S_1) \vee \cdots \vee (\boldsymbol{x} \in C_{p_k} \, and \, \boldsymbol{x} \notin S_k)\}|}{l} \tag{2}$$

The figures demonstrating the 100 and 250 trials completed can be seen below.

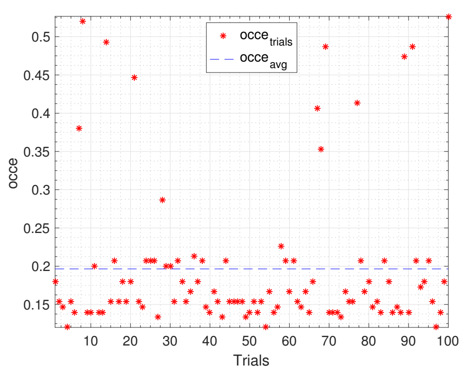


Figure 4: occe for 100 trials.

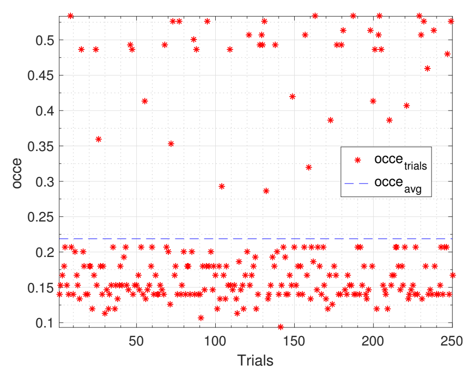


Figure 5: occe for 250 trials.

The large fluctuations in the *occe* from trial to trial shown in Figures 4 and 5 above can be associated with the randomness when initializing the centroids centres at the start of the algorithm.

1 K-Means [(1.1.2)] continued on next page...

Table 1 shown below demonstrates the mean and standard deviation of the *occe*.

No. of Trails	Average occe	Std Dev. occe
100	0.1965	0.0966
250	0.2187	0.1281

Table 1: Comparison of occe for 100 and 250 trials.

It can be seen from the results in Figure 4 and 5 as well as Table 1 that the K-Means algorithm achieves a low mean *occe* for both the case of 100 and 250 trials of approximately 20%.

The code created in order to obtain these results can be seen in Appendix A (*Q1_1.m*).

(1.1.3)

The Iris dataset was then obtained to perform K-Means on it and evaluate its performance. The Iris data set has a similar structure to that of S generated in 1.1.2, with three distinct classes related to a type of iris plant. Furthermore, each data point contained 4 features such as sepal length, sepal width, petal length and petal width. The data was preprocessed such that the class labels (5-th) column was removed. The resultant data was then saved as a *mat* file as , *iris_dataset.mat*.

Following this, K-Means clustering was performed on the Iris data for 100 and 250 independent trials. The resultant plots for each experiment can be seen in Figures 6 and 7 below.

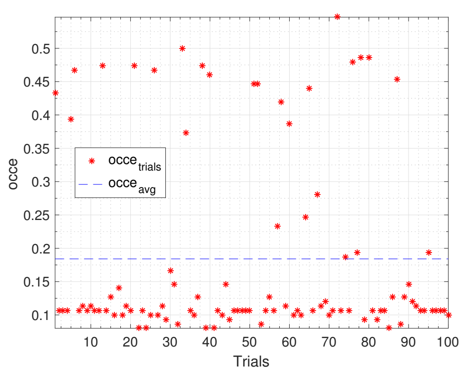


Figure 6: occe for 100 trials on Iris data set.

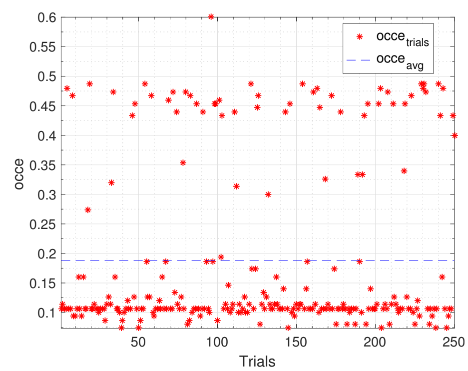


Figure 7: occe for 250 trials on Iris data set.

The mean and standard deviation associated with the *occe* can be seen in Table 2 shown below.

No. of Trails	Average occe	Std Dev. occe
100	0.1841	0.1410
250	0.1878	0.1438

Table 2: Comparison of occe for 100 and 250 trials on Iris data set.

It can be seen from Table 2 that the resultant *occe* for 100 and 250 trial are rather similar. The K-Mean clustering algorithm performed slightly better on the Iris data set when comparing Table 1 and 2.

The code utilized in order to obtain the results using the Iris data set can be found in Appendix A.

(1.2.1)

The centroid can be shown to be the minimizer of the sum of the squared distances (SSD) in the following manner.

$$SSD = \sum_{i=1}^{k} \sum_{x \epsilon C_i} dist(\boldsymbol{c}_i, \boldsymbol{x}) = \sum_{i=1}^{k} \sum_{x \epsilon C_i} ||\boldsymbol{c}_i - \boldsymbol{x}||^2 \tag{3}$$

Equation 3 is minimizes when

$$\frac{\partial}{\partial \boldsymbol{c}_j}(SSD) = \frac{\partial}{\partial \boldsymbol{c}_j}\left(\sum_{i=1}^{k} \sum_{x \epsilon C_i} dist(\boldsymbol{c}_i, \boldsymbol{x})\right) = 0$$

$$\frac{\partial}{\partial \boldsymbol{c}_j}(SSD) = \frac{\partial}{\partial \boldsymbol{c}_j}\left(\sum_{i=1}^{k} \sum_{x \epsilon C_i} ||\boldsymbol{c}_i - \boldsymbol{x}||^2\right)$$

$$\frac{\partial}{\partial \boldsymbol{c}_j}(SSD) = \left(\sum_{i=1}^{k} \sum_{x \epsilon C_i} \frac{\partial}{\partial \boldsymbol{c}_j} ||\boldsymbol{c}_i - \boldsymbol{x}||^2\right)$$

$$\frac{\partial}{\partial \boldsymbol{c}_j}(SSD) = \left(\sum_{i=1}^{k} \sum_{x \epsilon C_i} \frac{\partial}{\partial \boldsymbol{c}_j}\left(\sum_{l=1}^{n} (\boldsymbol{c}_i - \boldsymbol{x}_l)^2\right)\right)$$

$$\frac{\partial}{\partial \boldsymbol{c}_j}(SSD) = \sum_{x \epsilon C_j}\left(2(\boldsymbol{c}_j - \boldsymbol{x}_j)\right) \tag{4}$$

Now, we are able to equate te result dervied in Equation 4 shown above to zero.

$$\frac{\partial}{\partial \boldsymbol{c}_j}(SSD) = 0$$

$$\sum_{x \epsilon C_j}\left(2(\boldsymbol{c}_j - \boldsymbol{x}_j)\right)\Bigg) = 0$$

We now, attempt to make $\boldsymbol{c}_j$ the subject of the equation as can be seen below.

$$\sum_{x \epsilon C_i} \boldsymbol{c}_j - \sum_{x \epsilon C_j} \boldsymbol{x}_j = 0$$

$$\sum_{x \epsilon C_i} \boldsymbol{c}_j = \sum_{x \epsilon C_j} \boldsymbol{x}_j$$

$$m_j \boldsymbol{c}_j = \sum_{x \epsilon C_j} \boldsymbol{x}_j$$

$$\boldsymbol{c}_j = \frac{\sum_{x \epsilon C_j} \boldsymbol{x}_j}{m_j} \tag{5}$$

Thus, it can be seen from Equation 5 shown above that Equation 3 is minimized by the centroid as required.

1 K-Means [(1.2.1)] continued on next page...

Equation 5 can be rewritten using the indicator matrix $\boldsymbol{r}$ as shown below in Equation 6.

$$\boldsymbol{c}_j = \frac{\sum_{i=1}^{l} \boldsymbol{r}_{ij}\boldsymbol{x}_i}{\sum_{i=1}^{l} \boldsymbol{r}_{ij}} \tag{6}$$

Therefore, proving the fact that the centroid is the minimizer of the sum of the squared distances.

1 K-Means continued on next page...

(1.2.2)

The proof that K-means converges within a finite number of iterations can be done by showing the SSD decreseas with each iterations due to the fact that the SSD is monotonically decreasing.

Formally, given a finite set of data denoted as S that is a subset of $\Re^n$ ($S \subset \Re^n$) and k-cluster ($k\epsilon\boldsymbol{Z}$) are decided to partition the data. It follows that there are k^l ways to partition the data given l data points ($\boldsymbol{x}_1, ..., \boldsymbol{x}_l$). It should also be noted that the centroids $c\epsilon\Re^n$.

Thus, based on every iteration of the algorithm, the updated clustering assignment done in Step 2 denoted in Equation 7 only depends on the previous clustering assignment.

$$\boldsymbol{r}_{ij} = \begin{cases} 1 & if\ j = argmin_{1\leq s\leq k}||\boldsymbol{x}_i - \boldsymbol{c}_s||^2 \\ 0 & otherwise \end{cases} \tag{7}$$

The centroids $\boldsymbol{c}_1, ..., \boldsymbol{c}_k$ are then computed as shown in Step 3 which is shown in Equation 6 earlier. In a more formal manner [1]:

- ${\boldsymbol{c}_1}^{(t)}, {\boldsymbol{c}_2}^{(t)}, ..., {\boldsymbol{c}_k}^{(t)}$: Denotes the centroids on the t-th iteration.
- ${C_1}^{(t)}, {C_2}^{(t)}, ..., {C_k}^{(t)}$: Denotes the clusters on the t-th iteration.

Using Equation 3, we consider the case when $t = 1$:

$$SSD(C_{1:k}, \boldsymbol{c}_{1:k}) = \sum_{i=1}^{k} \sum_{x\epsilon C_i} ||\boldsymbol{c}_i - \boldsymbol{x}||^2$$

$$SSD({C_{1:k}}^{(1)}, {\boldsymbol{c}_{1:k}}^{(1)})$$

Therefore, after the t-th iteration ($t > 1$) the following inequality holds true:

$$SSD({C_{1:k}}^{(t)}, {\boldsymbol{c}_{1:k}}^{(t)}) \leq SSD({C_{1:k}}^{(1)}, {\boldsymbol{c}_{1:k}}^{(1)})$$

On the $t + 1$-th iteration in Step 2 of Equation 7, the cluster assignment update results in the following inequality:

$$SSD({C_{1:k}}^{(t+1)}, {\boldsymbol{c}_{1:k}}^{(t)}) \leq SSD({C_{1:k}}^{(t)}, {\boldsymbol{c}_{1:k}}^{(t)})$$

The centroids are then recalculated in Step 3 shown in Equation 6 resulting in a further reduction in the inequality:

$$SSD({C_{1:k}}^{(t+1)}, {\boldsymbol{c}_{1:k}}^{(t+1)}) \leq SSD({C_{1:k}}^{(t+1)}, {\boldsymbol{c}_{1:k}}^{(t)}) \tag{8}$$

It can be seen from from the mathematical relationships shown earlier, there are three observations that can be made:

1. The new clustering can be different from previous but it will always be equal to the old SSD or less as seen in Equation 8.
2. The algorithm can enter a cyclic stage such that when the update in Step 2 for a new clustering is made, it is the same as the previous clustering assignment. Thus K-Means has converged as the clustering remains unchanged [2].
3. Since there is a finite data set S, the algorithm must converge.

1 K-Means [(1.2.2)] continued on next page...

Therefore, after t^* iterations the algorithm converges after a finite number of steps.

$$SSD({C_{1:k}}^{(t^*)}, {\boldsymbol{c}_{1:k}}^{(t^*)}) = SSD({C_{1:k}}^{(t^*-1)}, {\boldsymbol{c}_{1:k}}^{(t^*-1)})$$

In saying this, given the finite data set and the fact that the SSD is monotonically decreasing the algorithm will converge.

(1.3.1)

(1.3.2)

2 PCA

(2.1.1)

The PCA algorithm was implemented in MATLAB, as a function called *pca_algorithm*, which took a series of data points and the number of new principal components. The implementation of PCA algorithm can be found in the file called *pca_algorithm.m* shown in Appendix B which is attached with this assignment. The function call for this algorithm can be seen in more detail as shown below.

```
%% PCA Algorithm
function X_hat = pca_algorithm(X,k)
```

Where:

- X: Input data points.
- k: Number of principal components.
- X_hat: Transformed data.

(2.1.2)

The K-Means algorithm was implemented to take the input data and the number of cluster similiar to that created in 1.1.1 and return the value of the objective function as well as the "clustering". The implementation of this algorithm can be seen in the MATLAB file, *k_MeansQ2.m* which can be found in Appendix B. The function declaration can be seen below.

```
%% Modified K-Means Algorithm
[centroids, indicator_variable,objective_result] = kMeans_Q2(X,k)
```

The objective function was established to be the following:

$$E_X(C_1, ..., C_k; \boldsymbol{c}_1, ..., \boldsymbol{c}_k) = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} ||\boldsymbol{x} - \boldsymbol{c}_i||^2 \tag{9}$$

(2.1.3)

Using the algorithms created earlier such as K-Means and PCA, the Iris data set was used as a test bed in order to evaluate the performance of the K-Means algorithm was the dimension of the transformed data was varied from $k = 1$ principal components up until $k = 4$, as well as evaluate the performance on the untransformed data set.

For each of the five cases, the following information will be presented:

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

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

(c) Plots the occes plotted as a function of the rank of the objective function.

Case: k = 1

(a) The smallest 3 ”occes” along with their computed value of the objective can be seen in Table 3 shown below for $k = 1$.

Index	occe	obj. func
1	0.0533	39.8389
2	0.0533	39.8389
3	0.0533	39.8389

Table 3: Smallest three occes for $k = 1$.

(b) The resultant mean and standard deviation of the ”occes” and the ”objectives” can be seen in Table 4 shown below.

mean occe	std dev. occe	mean obj. func	std dev. obj. func
0.1450	0.1185	55.1489	53.3739

Table 4: Smallest three occes for $k = 1$.

It can be seen from Table 4 that when the dimensionality was reduced to $k = 1$ with PCA only resulted in an average 14.50% error in clustering. Considering, the data compression of reducing k, $4 \rightarrow 1$ in dimension while maintaining a 14.50% *occe* is impressive. It should be noted that the objective function has a high variance which can be attributed to the initialization of the clustering algorithm centroids.

The bar chart with the ”occes” plotted as a function of the rank of the objective function can be seen below for the case when $k = 1$.

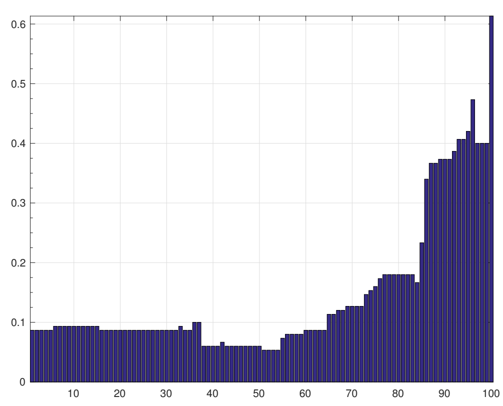


Figure 8: Plot for $k = 1$ of occe as a function of rank of objective function.

It can be seen from Figure 8 shown above that as the objective function ranking increases the corresponding *occe* value increases which is to be expected as the objective equation calculates the sum of the squared euclidean distances between the data points $\boldsymbol{x} \in C_i$ and the corresponding centroids $\boldsymbol{c}_i$ as $i = 1, ...k$ (Equation 9). Thus if the objective function is large, the classification error would be large as more points were misclassified.

Case: k = 2

(a) The smallest 3 ”occes” along with their computed value of the objective can be seen in Table 5 shown below for $k = 2$.

Index	**occe**	**obj. func**
1	0.0733	57.5610
2	0.0800	55.8189
3	0.0867	56.4502

Table 5: Smallest three occes for $k = 2$.

(b) The resultant mean and standard deviation of the ”occes” and the ”objectives” can be seen in Table 6 shown below.

mean occe	**std dev. occe**	**mean obj. func**	**std dev. obj. func**
0.1997	0.1454	71.7835	27.8529

Table 6: Smallest three occes for $k = 2$.

It can be seen when comparing Table 4 and 6 that the standard deviation for the case when $k = 2$ is far lower than $k = 1$, however the mean objective was higher.

(c.) The bar chart with the ”occes” plotted as a function of the rank of the objective function can be seen below for the case when $k = 2$.

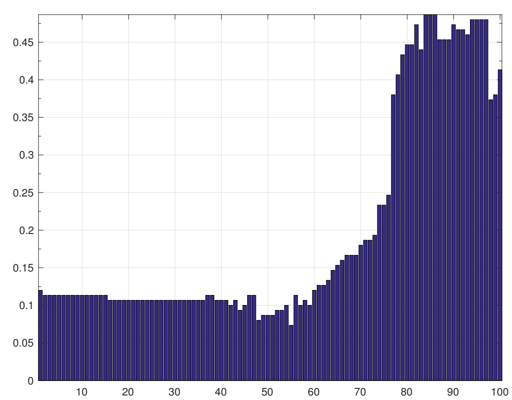


Figure 9: Plot for $k = 2$ of occe as a function of rank of objective function.

The same pattern present in Figure 8 can also be seen for Figure 9. As the rank of the objective function increases, correspondingly so does the *occe*.

Case: k = 3

(a) The smallest 3 ”occes” along with their computed value of the objective can be seen in Table 5 shown below for $k = 3$.

Index	**occe**	**obj. func**
1	0.0733	56.9389
2	0.0733	58.1221
3	0.0800	56.4247

Table 7: Smallest three occes for $k = 3$.

(b) The resultant mean and standard deviation of the ”occes” and the ”objectives” can be seen in Table 8 shown below.

mean occe	**std dev. occe**	**mean obj. func**	**std dev. obj. func**
0.2147	0.1651	78.3798	58.5448

Table 8: Smallest three occes for $k = 3$.

The bar chart with the ”occes” plotted as a function of the rank of the objective function can be seen below for the case when $k = 3$.

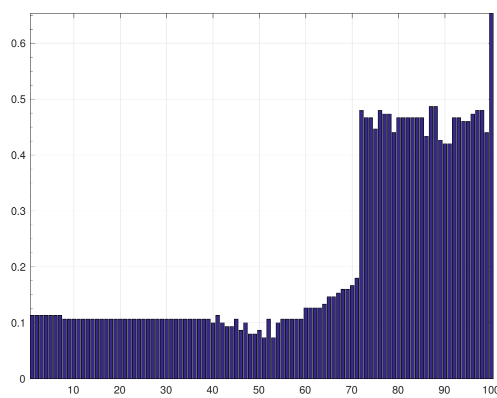


Figure 10: Plot for $k = 3$ of occe as a function of rank of objective function.

Case: k = 4

(a) The smallest 3 "occes" along with their computed value of the objective can be seen in Table 9 shown below for $k = 4$.

Index	**occe**	**obj. func**
1	0.0333	62.4443
2	0.0333	62.4443
3	0.0733	56.2085

Table 9: Smallest three occes for $k = 4$.

(b) The resultant mean and standard deviation of the "occes" and the "objectives" can be seen in Table 10 shown below.

mean occe	**std dev. occe**	**mean obj. func**	**std dev. obj. func**
0.1825	0.1401	69.7301	25.3975

Table 10: Smallest three occes for $k = 4$.

(c.) The bar chart with the "occes" plotted as a function of the rank of the objective function can be seen below for the case when $k = 4$.

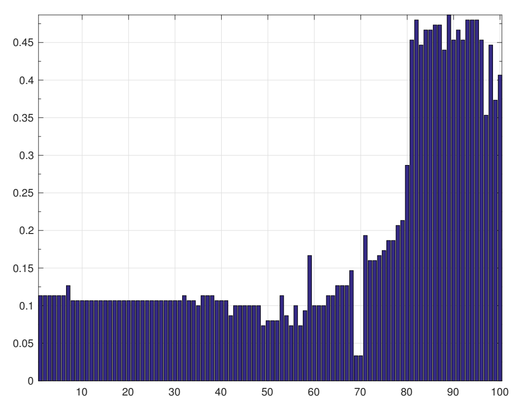


Figure 11: Plot for $k = 4$ of occe as a function of rank of objective function.

Comparing each of the scenarios when PCA was applied with $k = 1, 2, 3, 4$, the relevant mean *occe* remained approximately 20% with a standard deviation of $\approx 15\%$. Thus demonstrating, the preservation of information throughout the dimensionality reduction process of PCA.

Case: Untransformed Data

(a) The smallest 3 "occes" along with their computed value of the objective can be seen in Table 11 shown below for the untransformed data.

Index	**occe**	**obj. func**
1	0.0333	67.6534
2	0.0733	59.6037
3	0.0800	58.6102

Table 11: Smallest three occes for the untransformed data.

(b) The resultant mean and standard deviation of the "occes" and the "objectives" can be seen in Table 12 shown below.

mean occe	**std dev. occe**	**mean obj. func**	**std dev. obj. func**
0.1981	0.1479	81.0257	62.3956

Table 12: Smallest three occes for the untransformed data.

(c.) The bar chart with the "occes" plotted as a function of the rank of the objective function can be seen below for the case when the data is untransformed.

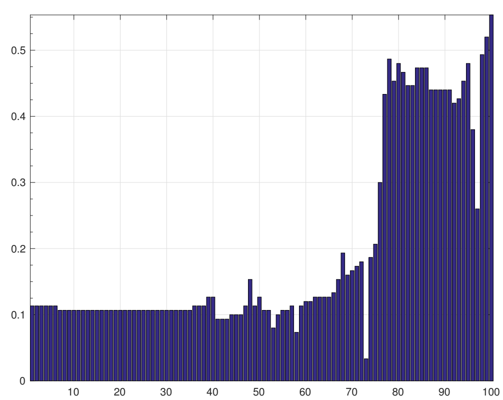


Figure 12: Plot for untransformed data of occe as a function of rank of objective function.

The relevant error bar plots associated with $k = 1, 2, 3, 4$ and the original data for the "occes" and their relevant objective functions can be seen below can be seen in Figure 13 and 14 shown on the following page.

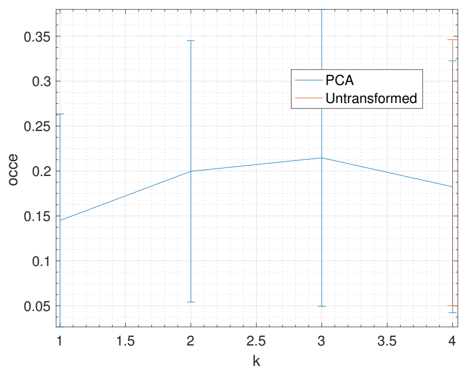


Figure 13: occe for 100 trials.

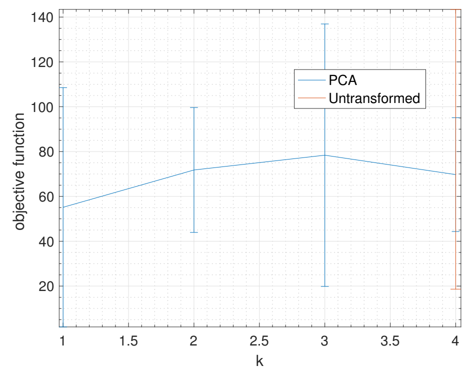


Figure 14: occe for 250 trials.

The resultant plots for the *occe* and objective function for 100 trials are shown in Figure 13 and 14 demonstrate the large variation in the objective function values as presented in the five earlier cases. Furthermore, It can be seen from inspection of Figure 13 that for the cases when $k = 2$ and $k = 3$, the average *occe* remains the same but decreasing for the case when $k = 1$ and $k = 4$. Even in the case for the untransformed data shown in Figure 13, there are large variations in the *occe* computed. The relationship between the *occe* and objective function is further evident when comparing the behaviour of Figure 13 and 14 seen above.

The code created in order to generate the results shown above can be found in Appendix B (*Q2_1.m*).

(2.1.4)

When the data set is in a higher dimension space such as $k = 4$ it is not possible to have a visual interpretation of the distribution of the data. In saying this, through the use of Principal Component Analysis (PCA), the dimensions of the original data $\boldsymbol{X}$ could be reduced to that of $\hat{\boldsymbol{X}}$ which could be 2-dimensional or 3-dimensional.

The K-Means algorithm was then applied to the transformed data which can be seen in Figure 15 shown below with the corresponding centroids. The code for this visualisation can be seen in Appendix B (*Q2_1.m*).

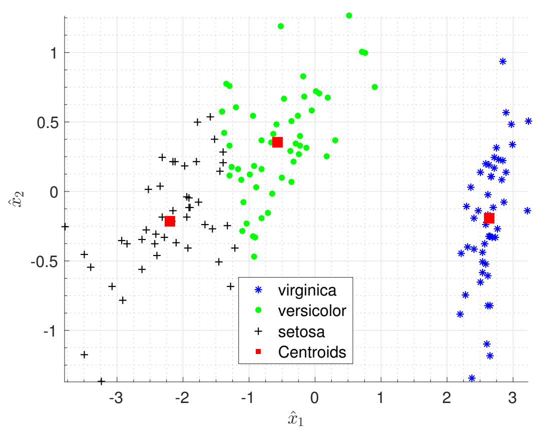


Figure 15: Plot of transformed Iris data set $\hat{\boldsymbol{X}}$ using PCA in 2D with clustering applied.

The data after K-Means clustering was applied can be seen in Figure 16 below with their corresponding centroids in 3D.

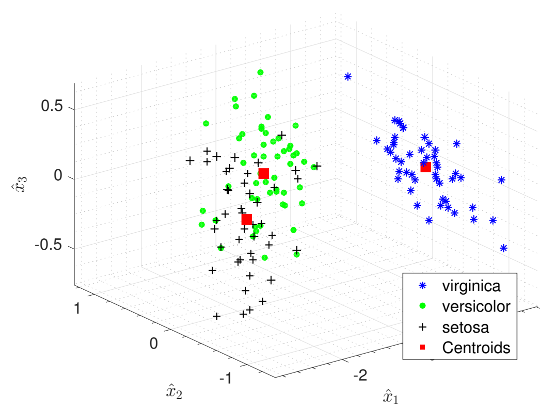


Figure 16: Plot of transformed Iris data set $\hat{\boldsymbol{X}}$ using PCA in 3D with clustering applied.

It can be seen when comparing Figure 15, that the K-Means algorithm performed well in the 2D case as the $occe = 0.0933$. The centroids were centered on the main clusters as required. Likewise, the same pattern was evident when analysing Figure 16 as the $occe = 0.1067$, thus PCA was able to preserve important information while reducing the input data dimensions to a new feature space.

(2.2)

Given partitioning $C_1, ..., C_k$ and true partitioning $S_1, ..., S_k$, the simple error as defined in Equation 1 as well as the *occe* defined in Equation 2. Although the *occe* computes the correct clustering error is it computationally expensive as it involed the minimization of errors over all possible permutations of C as can be seen in Equation 2. It is computed naively requiring $k!$ time, which for small dimensions is irrelevant but once the clustering $k \to \infty$ the computational time becomes unrealistic to use.

This can be attributed to the fact that all the possible permutations for the clusters in C need to be tested. Thus for the case when $k = 3$, the resultant permutation matrix P takes the following form:

$$P_{k!,k} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \\ 3 & 2 & 1 \end{pmatrix} \tag{10}$$

Each column of P are the possible assignments/mapping of C_1, C_2, C_3 to the true clusters S_1, S_2, S_3.

$$C = \begin{pmatrix} C_1 & C_2 & C_3 \end{pmatrix} \tag{11}$$

However, in order to overcome the computational complexity associated with *occe* formula that is $k!$ time an imporved polynomial time algorithm is presented.

We will first construct a relevant simple error matrix denoted as A, which is the cost associated with assigning each cluster $C_i \to S_j$ and the error associated with it.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,k} \end{pmatrix} \tag{12}$$

A resultant binary matrix $B \in \{0, 1\}$ is also created which will return indicated the index of optimal clustering combination such that the resultant *occe* is minimized.

$$B = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,k} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k,1} & b_{k,2} & \cdots & b_{k,k} \end{pmatrix} \tag{13}$$

Therefore, the resultant new computation of *occe* denoted as $\hat{occe}$ can be expressed in the following manner.

$$\hat{occe} = \sum_{i=1}^{k} \sum_{j=1}^{k} A_{i,j} B_{i,j} \tag{14}$$

If we consider the case when $k = 3$ as shown in question 1 of this assignment, the error matrix A takes the following form.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \tag{15}$$

The binary matrix we initialise to an all zero matrix as shown below.

$$B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{16}$$

Once the error matrix A has been formed using Equation 1, the problem shown above thus becomes a combinatorial optimization task similar to that proposed by Kuhn and Munkres [3], [4]. Thus the optimal selection of cluster mapping for error matrix can be solved using the following steps [5]:

1. **Row Subtraction**: Search for the smallest error associated with each row in A ($\hat{a} = min(A_{(i,\cdot)})$). Where $A_{(i,\cdot)}$ denotes the i-th row of A. Then subtract the error in each row with the minimum error found in each row to form A^*.

2. **Column Subtraction**: Search for the smallest error associated with each column in A^* ($\ddot{a} = min({A^*}_{(\cdot,j)})$). Where ${A^*}_{(\cdot,j)}$ denotes the j-th column of A^*. Then subtract the error in each column with the minimum error found in each column of A^*.

3. **RC Elimination**: Eliminate rows and columns by drawing lines through appropriate rows and columns of A^* such that all the zero entries in the new error matrix A^* and the minimum number of lines l were used in the elimination process.

4. **Optimality Condition**:

   (a) Check if number of lines l used in RC Elimination is greater than or equal to k. If true, optimality condition true.

   (b) Otherwise, $l < k$, optimality condition false.

5. **Smallest Uncovered Value**: Find the smallest entry not covered by lines l done in Step 3 for A^*. Then subtract this minimum value found from all uncovered rows in A^* as well as add it to all covered columns. Return to Step 3 (RC Elimination).

The pseudocode describing the algorithm described above, can be seen in Algorithm 1 shown below.

Algorithm 1 *$o\hat{\check{c}}ce$* polynomial algorithm

Input: Error matrix A
$optimal \leftarrow 0$
$count \leftarrow 0$
while $optimal \neq 1$ **do**
  if$(count \neq 0)$

    for $i = 1 : k$ **% Step 1**
      $\hat{a}(i) \leftarrow min(A(i, \cdot))$
    end for
    $A^* \leftarrow A - \hat{a}$

    for $j = 1 : k$ **% Step 2**
      $\ddot{a}(i) \leftarrow min(A^*(\cdot, j))$
    end for
    $A^* \leftarrow A^* - \ddot{a}$
  end if

  $[l, rows_{uncovered}, cols_{uncovered}] \leftarrow$ *rc_elimnation(*A^***)** **% Step 3**

  if$(l \geq k)$ **% Step 4**
    $optimal \leftarrow 1$
    $B \leftarrow$ *optimal_index_assignment(*$A^*, l, rows_{uncovered}, cols_{uncovered}$*)*
  else
    $count \leftarrow 1$
  end if

  if$(optimal \neq 1)$ **% Step 5**
    $d \leftarrow$ *min_remaining(*$A^*, rows_{uncovered}, cols_{uncovered}$*)*
    $A^* \leftarrow A^*(rows_{uncovered}) - d$
    $cols_{covered} \leftarrow$*covered_cols(*$A^*, cols_{uncovered}$*)*
    $A^* \leftarrow A^*(cols_{covered}) + d$
  end if
end while
return B

Therefore, once the binary matrix B is returned from the algorithm we are able to compute the minimized error $o\hat{c}ce$ using Equation 14.

In order solidify the $o\hat{c}ce$ polynomial algorithm further, we will test it on a simple $k = 3$ error matrix with arbitrary errors in each element. Thus the error matrix can be seen below.

$$A = \begin{pmatrix} 52 & 63 & 69 \\ 77 & 73 & 52 \\ 11 & 69 & 5 \end{pmatrix}$$

Therefore, it can be seen from the derivation done thus far and the pseudocode presented in Algorithm 1, the optimization problem can be solved as follows.

$$\begin{pmatrix} 52 & 63 & 69 \\ 77 & 73 & 52 \\ 11 & 69 & 5 \end{pmatrix}^{Input} \sim \begin{pmatrix} 0 & 11 & 17 \\ 25 & 21 & 0 \\ 6 & 64 & 0 \end{pmatrix}^{Step\ 1} \sim \begin{pmatrix} 0 & 0 & 17 \\ 25 & 10 & 0 \\ 6 & 53 & 0 \end{pmatrix}^{Step\ 2}$$

$$\begin{pmatrix} 0 & 0 & 17 \\ 25 & 10 & 0 \\ 6 & 53 & 0 \end{pmatrix}^{Step\ 3} \sim \begin{pmatrix} 0 & 0 & 23 \\ 19 & 4 & 0 \\ 0 & 47 & 0 \end{pmatrix}^{Step\ 5} \sim \begin{pmatrix} 0 & 0 & 23 \\ 19 & 4 & 0 \\ 0 & 47 & 0 \end{pmatrix}^{Step\ 3}$$

After Step 3 in each instance the conditional statement for Step 4 was evaluated before proceeding to Step 5 or terminating the algorithm. Therefore the adjusted resultant matrix A^* takes the following form:

$$A^* = \begin{pmatrix} 0 & 0 & 23 \\ 19 & 4 & 0 \\ 0 & 47 & 0 \end{pmatrix}$$

The resultant binary matrix B returned can be seen below.

$$B = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Using Equation 14 we can compute the $o\hat{c}ce$ as an be seen below:

$$o\hat{c}ce = \sum_{i=1}^{k}\sum_{j=1}^{k} A_{i,j}B_{i,j} = (63 \times 1) + (52 \times 1) + (11 \times 1) = 127$$

2 PCA [(2.2)] continued on next page...

It is now important to prove that the algorithm suggested in the workings above is able to compute in polynomial time.

- **Step 1**: It can seen from Algorithm 1 on lines $7 \rightarrow 9$ that the search or the minimum element in each row, goes through each column associated with a given row. Thus, the complexity for this calculation is $O(k^2)$.
- **Step 2**: In the same manner as Step 1, the search through all rows for a given column that varies across A^* has a complexity of $O(k^2)$ as can be seen by the for-loop on lines $7 \rightarrow 9$ of Algorithm 1.
- **Step 3**: The search through A^* to eliminate the optimal number of rows and columns contributes to the highest complexity associated with this algorithm. This is largely due to the fact that one must keep track of the number of zeros eliminated as we go through each row, and likewise columns of A^*. Thus, by having to check if each element in A^* and wheter or not it has been covered by a column or row elimination results in a nested for-loop which would be executed in the function *rc_elimination(·)* on line 18 of Algorithm 1. Therefore, the complexity associated with such a computation is $O(k^4)$.
- **Step 4**: This is simply a conditional statement check and does not contribute the majority of the computational time $O(1)$.
- **Step 5**: The search for the minimum number in the uncovered set of A^*, subtracting it from uncovered rows values as well as adding it to covered columns results in a complexity of $O(k^2)$.

This result, proves that the algorithm works and computes in polynomial time namely $O(k^4)$ opposed to $O(k!)$ as originally suggested when calculating the *occe* in 2.1 [6].

3 Perceptron

(3.1.1)

Implementation

The implementation of the algorithm is shown in full in the code section. Efficiency is improved beyond the supplied sample code through the prediction and update functions which calculate the kernel, store only the non-zero alphas and the x vectors associated with non zero alphas. The prediction and update functions are:

$$\hat{y}_t = \sum_{i \in m(t-1)}^{t} y_i \kappa(x_i, x_t)^d$$

Where m is the set of indicies of non-zero alphas. Non-zeros alphas are equivalent to the y's at each index. The polynomial kernel summation is then calculated as:

$$k_{class} = X_{class} * x_t^T$$

Where X is the matrix of previously stored training examples for the class under consideration associated with the non-zero alphas for that class and x_t^T is the current training example.
Then k_{class} is dot multiplied by itself d times where d is the polynomial index.
A prediction when not training is made simply by inputting x_t^T that corresponds to the example under consideration.
The summation is then made more efficient using the dot product:

$$\hat{y}_t = \alpha_{class} * k_{class}$$

Where α_{class} is the vector of stored alphas.

Once the class prediction $\hat{y}_t$ is compared to class of y_t the update algorithm if there is no match is then an appendment as follows:

$$\alpha_{class} = [\alpha_{class} \quad \hat{y}_t]$$

$$X_{class} = [X_{class} \ ; x_t]$$

The two algorithm's are entirely analagous to the algorithm presented in the coursework document.

3 Perceptron [(3.1.1)] continued on next page...

(3.1.2)

Table: Validation Set Error

Kernel Polynomial	2	3	4	5	6	7
Validation Error %	2.14	3.25	3.95	5.64	6.46	7.12

Each polynomial kernel is trained over 10 epochs of the modified training set. 10 epochs is selected as in trials this no led to the lowest training set error.
The validation set error is calculated as a simple percentage over 2431 examples split from the original training set.
The results would pick polynomial kernel order 2 as the optimum kernel.

(3.1.3)

Table: Test Set Error

Kernel Polynomial	2	3	4	5	6	7
Test Set Error %	5.43	5.33	6.48	8.12	9.22	9.87

The test set error is calculated as a simple percentage over the 2007 examples from the test set. Polynomial kernel order 3 produces the lowest test error. This result is surprising given that the validation set would suggest that the lowest test error would be produced polynomial kernel order 2. One explanation is that the limited examples in the validation set are randomly more suitable to the specific updates performed on kernel perceptron of order 2 in the training set. This can be addressed by increasing the effective size of the validation set using k=fold validation.

(3.1.4)

i)
The word recognize is potentially ambiguous so the author has where necessary provided several explanations based on possible interpretations.
If recognize could be read to mean which classes are most and least correctly classified (recognized) from the model defined by the training and parameter optimization applied to the test set the following table shows the errors per class as a percentage of the number of examples of each class present in the test set. The optimized model applied is the kernel perceptron order 3.

Table: No of errors per class over test set of optimized model

Class	% errors
0	1.67
1	2.65
2	7.56
3	10.24
4	7.00
5	7.50
6	4.12
7	4.08
8	9.04
9	4.52

Class zero has the least percentage error and is thus most easily recognized by the model. Class 3 has the most percentage errors and is thus the hardest for the model to recognize.
For such varied examples for each class (the way people draw their numbers) there will be substantial overlap between the numerical descriptions of one example and the numerical descriptions of another another - the data is non-linearly separable further however is that whole classes may have features that are similar and the model works to separate the data as best as it can. However if the perceptron can be thought of as converging roughly to the average of a class the ambiguities of a particular example can cause a misclassification.
The class with the least percentage error is the class with the most examples as a percentage of the test set that fits within the separating boundary described by the polynomial kernel trained on the examples in the training set. Conversely the class with the greatest percentage error is the class with the least examples as a percentage of the test set that fits within the separating boundary.

An alternative interpretation of the question could be to try to understand how certain the model is about each prediction by taking normalized value of a correct prediction by class and then averaging for each class.

(3.1.4)

When making each prediction we take: $\underset{1\le i\le k}{\arg\max}\ k^{(i)}(x)$

If the k vector is normalised it can be thought of as a list of probabilities of the example belonging to each class:

$$k_{norm} = \left| \frac{\underset{1\le i\le k}{\arg\max}\ k^{(i)}(x)}{\sum_{i}^{k} k^{(i)}(x)} \right|$$

This can then be averaged for each class and the class with the highest average probability would be the easiest to recognize and the class with the lowest average probability would be the hardest.

Table: Average Normalized Class Probability

Class	Average Normalised Probability
0	0.1293
1	0.1069
2	0.1278
3	0.1166
4	0.1261
5	0.1173
6	0.1353
7	0.1387
8	0.0929
9	0.0991

Thus by this measure the easiest class to recognize is 7 and the hardest class to recognize is 8.

(3.1.4)

ii)

<u>Table: Confusion Matrix</u>

		Prediction									
	Class	0	1	2	3	4	5	6	7	8	9
Actual	0	0	0	1	0	0	0	2	3	0	0
	1	0	0	0	0	4	0	3	0	0	0
	2	3	2	0	1	3	0	1	1	4	0
	3	2	0	1	0	0	9	0	2	2	1
	4	0	3	2	0	0	2	1	2	1	3
	5	4	2	0	1	1	0	1	1	1	1
	6	2	0	1	0	1	1	0	1	1	0
	7	0	1	1	1	2	0	0	0	0	1
	8	3	2	1	4	0	2	0	1	0	2
	9	2	0	1	0	4	0	0	1	0	0

The table above shows that over the test set classes 1 & 4 are most frequently confused for each other.
Quantitatively this is due to the examples for 1 & 4 in the training set having the most similar numerical features meaning that the separating boundary sits close to the middle of the distribution of the numerical features for these two classes and thus small changes in the test set mean that the separating boundary classifies these classes in an alternate fashion.

In absolute terms the actual class 3 is most frequently predicted as a 5. This will likely be because the examples in the test set of actual 3's most closely correspond numerically to the examples of 5's in the training set.

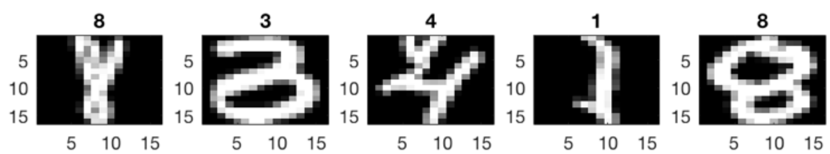
(3.1.4)

iii) The 5 hardest to recognize correctly predicted samples in the test set is computed using the normalised probability measure as described in part i calculated for each sample.

Table: 5 Hardest Samples

	1	2	3	4	5
Normalised Probability of argmax	9E-05	1E-04	5E-04	6E-04	7E-04
"record no"	1029	648	63	728	1812
Value	8	3	4	1	8

Print Out of Hardest Samples



Reading from the left three of the five hardest samples are indeed hard to recognize even for a human. The first 8 does not resemble anything, the three is reminiscent of a curling stone and the four looks like a poodle on its hind legs - in other words they do not resemble numbers. It is important to remember that all these numbers have been correctly classified but it is easy to see why the model having trained on more representative examples would not have a high degree of confidence in it's prediction. The 1 and the second 8 whilst still correctly classified are harder to understand as to the naked eye these are fairly good examples of each no. Remembering that a perceptron unlike the SVM does not maximize the width around the separation boundary - in the case of non seperable data - the min error width - thus the 1 and 8 whilst still good examples may be leaning towards a 7 and a 0 in the models estimation (or some other number).

Appendices

A: k-Means

In the this section and all of the following sections of the code we have used external functions at multiple places. Please refer Library Functions section at the end in the appendix for their implementation.

Q1_1.m

```
%------------------------------------------------------------------------%
% Module: GI07 - Mathematical Methods for Machine Learning
% Assignment : Coursework 2
% Author : Russel Daries, Hugo Philion
% Student ID: 16079408, 14102040
% Question: 1
% Description: K-Means
% ------------------------------------------------------------------------%

% clearing memory
clear all
close all
clc

%% 1.1.1 - Implement K-Means Algorithm

% Can be seen in function kMeans

%% 1.1.2 - Testing Algorithm

dataset = genData2;
k = 3;

cluster_1_cor = dataset(1:50,:);
cluster_2_cor = dataset(51:100,:);
cluster_3_cor = dataset(101:150,:);

% Plot of data with clustering
figure;
scatter(cluster_1_cor(:,1),cluster_1_cor(:,2),'filled')
hold on
scatter(cluster_2_cor(:,1),cluster_2_cor(:,2),'filled')
hold on
scatter(cluster_3_cor(:,1),cluster_3_cor(:,2),'filled')
% hold on
% scatter(centroids(:,1),centroids(:,2),'rs');
xlabel('x_1','FontSize',15)
ylabel('x_2','FontSize',15)
set(gcf, 'Color', 'w');
leg=legend('S1','S2','S3','Location','Best');
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
set(gca,'XMinorTick','on')
```

```
set(gca,'FontSize',15)
grid on;
grid minor;
axis tight;
print('q1_1_class_data_correct','-depsc')
close all;

[centroids, indicator_variable] = kMeans(dataset,k);

cluster_1 = dataset((indicator_variable(:,1)~=0),:);
cluster_2 = dataset((indicator_variable(:,2)~=0),:);
cluster_3 = dataset((indicator_variable(:,3)~=0),:);

% Plot of Original Data that is unclustered
figure;
scatter(dataset(:,1),dataset(:,2),'filled')
xlabel('x_1','FontSize',15)
ylabel('x_2','FontSize',15)
set(gcf, 'Color', 'w');
set(gca,'YMinorTick','on')
set(gca,'XMinorTick','on')
set(gca,'FontSize',15)
grid on;
grid minor;
axis tight;
print('q1_1_org_data','-depsc')
close all;

% Plot of data with clustering
figure;
scatter(cluster_1(:,1),cluster_1(:,2),'filled')
hold on
scatter(cluster_2(:,1),cluster_2(:,2),'filled')
hold on
scatter(cluster_3(:,1),cluster_3(:,2),'filled')
hold on
scatter(centroids(:,1),centroids(:,2),150,'ks','filled');
xlabel('x_1','FontSize',15)
ylabel('x_2','FontSize',15)
set(gcf, 'Color', 'w');
leg=legend('Cluster 1','Cluster 2','Cluster 3','Centroids','Location','Best');
set(gca,'YMinorTick','on')
set(gca,'XMinorTick','on')
set(gca,'FontSize',15)
grid on;
grid minor;
axis tight;
print('q1_1_class_data_unlabelled','-depsc')
close all;
```

```

% Resultant plot for err calculation

% [err_logic , err_counts] = simple_error(dataset,k,indicator_variable)
err_counts = simple_error(dataset,k,indicator_variable)

% % Resultant plot for occe calculation
%
[occe_err , perm_vec_1] = occe_error(dataset,k,indicator_variable)
[cluster1_str,cluster2_str,cluster3_str] = cluster_mapping_q1(perm_vec_1)

cluster_1 = dataset((indicator_variable(:,perm_vec_1(1))~=0),:);
cluster_2 = dataset((indicator_variable(:,perm_vec_1(2))~=0),:);
cluster_3 = dataset((indicator_variable(:,perm_vec_1(3))~=0),:);

% Plot of data with clustering
figure;
scatter(cluster_1(:,1),cluster_1(:,2),'filled')
hold on
scatter(cluster_2(:,1),cluster_2(:,2),'filled')
hold on
scatter(cluster_3(:,1),cluster_3(:,2),'filled')
hold on
scatter(centroids(:,1),centroids(:,2),150,'ks','filled');
xlabel('x_1','FontSize',15)
ylabel('x_2','FontSize',15)
set(gcf, 'Color', 'w');
leg=legend(cluster1_str,cluster2_str,cluster3_str,'Centroids','Location','Best
    ');
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
set(gca,'XMinorTick','on')
set(gca,'FontSize',15)
grid on;
grid minor;
axis tight;
print('q1_1_class_data_labelled','-depsc')
close all;

% Resultant plots for average and std deviation of occe for 100 trials
occe_err_100 = zeros(100,1);
k_m = 3;
for m=1:100
    dataset_m = dataset;
    [centroids_m, indicator_variable_m] = kMeans(dataset_m,k_m);
    occe_err_100(m) = occe_error(dataset_m,k_m,indicator_variable_m);
end

occe_err_100_avg = mean(occe_err_100)
```

```
occe_err_100_std = std(occe_err_100)

figure;
plot(occe_err_100,'r*')
hold on
plot(occe_err_100_avg*ones(length(occe_err_100),1),'b--')
xlabel('Trials','FontSize',15)
ylabel('occe','FontSize',15)
set(gcf, 'Color', 'w');
leg=legend('occe_{trials}','occe_{avg}','Location','Best');
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
set(gca,'XMinorTick','on')
set(gca,'FontSize',15)
grid on;
grid minor;
axis tight;
print('q1_2_occe_100_trials','-depsc')
close all;

% Resultant plots for average and std deviation of occe for 250 trials
occe_err_250 = zeros(250,1);
k_n = 3;

for n=1:250
    dataset_n = dataset;
    [centroids_n, indicator_variable_n] = kMeans(dataset_n,k_n);
    occe_err_250(n) = occe_error(dataset_n,k_n,indicator_variable_n);
end

occe_err_250_avg = mean(occe_err_250)
occe_err_250_std = std(occe_err_250)

figure;
plot(occe_err_250,'r*')
hold on
plot(occe_err_250_avg*ones(length(occe_err_250),1),'b--')
xlabel('Trials','FontSize',15)
ylabel('occe','FontSize',15)
set(gcf, 'Color', 'w');
leg=legend('occe_{trials}','occe_{avg}','Location','Best');
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
set(gca,'XMinorTick','on')
set(gca,'FontSize',15)
grid on;
grid minor;
axis tight;
print('q1_2_occe_250_trials','-depsc')
```

```
close all;


%% 1.1.3 - Testing on Iris Dataset

k_iris = 3;

% Read in Iris Data from file

load('iris_dataset.mat');
dataset_iris = iris_data;

% Perform k-means on Iris data

[centroids_iris, indicator_variable_iris] = kMeans(dataset_iris,k_iris);

cluster_1 = dataset_iris((indicator_variable_iris(:,1)~=0),:);
cluster_2 = dataset_iris((indicator_variable_iris(:,2)~=0),:);
cluster_3 = dataset_iris((indicator_variable_iris(:,3)~=0),:);

% Resultant plot for err calculation

err_iris = simple_error(dataset_iris,k_iris,indicator_variable_iris);

% Resultant plot for occe calculation

occe_err_iris = occe_error(dataset_iris,k_iris,indicator_variable_iris);

% Resultant plots for average and std deviation of occe for 100 trials
occe_err_100_iris = zeros(100,1);
k_k=3;
for k=1:100
    [centroids_k, indicator_variable_k] = kMeans(dataset_iris,k_k);
    occe_err_100_iris(k) = occe_error(dataset_iris,k_k,indicator_variable_k);
end

occe_err_100_iris_avg = mean(occe_err_100_iris)
occe_err_100_iris_std = std(occe_err_100_iris)

figure;
plot(occe_err_100_iris,'r*')
hold on
plot(occe_err_100_iris_avg*ones(length(occe_err_100_iris),1),'b--')
xlabel('Trials','FontSize',15)
ylabel('occe','FontSize',15)
set(gcf, 'Color', 'w');
leg=legend('occe_{trials}','occe_{avg}','Location','Best');
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
```

```
set(gca,'XMinorTick','on')
set(gca,'FontSize',15)
grid on;
grid minor;
axis tight;
print('q1_2_occe_iris_100_trials','-depsc')
close all;

% Resultant plots for average and std deviation of occe for 250 trials

occe_err_250_iris = zeros(250,1);
k_l=3;
for l=1:250
    [centroids_l, indicator_variable_l] = kMeans(dataset_iris,k_l);
    occe_err_250_iris(l) = occe_error(dataset_iris,k_l,indicator_variable_l);
end

occe_err_250_iris_avg = mean(occe_err_250_iris)
occe_err_250_iris_std = std(occe_err_250_iris)

figure;
plot(occe_err_250_iris,'r*')
hold on
plot(occe_err_250_iris_avg*ones(length(occe_err_250_iris),1),'b--')
xlabel('Trials','FontSize',15)
ylabel('occe','FontSize',15)
set(gcf, 'Color', 'w');
leg=legend('occe_{trials}','occe_{avg}','Location','Best');
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
set(gca,'XMinorTick','on')
set(gca,'FontSize',15)
grid on;
grid minor;
axis tight;
print('q1_2_occe_iris_250_trials','-depsc')
close all;
```

genData2.m

```
function [ data ] = genData2
% generate data function provided by Dr. Mark Herbster

A1=[0.29 0.4; 0.4 4];
u1=[4 0];

A2=[0.29 0.06; 0.06 0.09];
u2=[5 7];

A3=[0.64 0; 0 0.64];
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
```

kMeans.m

```
function [centroids, indicator_variable] = kMeans(X,k)
% Function to implement K-Mean Algorithm

% Initialize centroid positions on k of data points selected randomly
indicator_variable = zeros(size(X,1),k);
rand_indices = randsample(size(X,1),k);
prev_centroids = zeros(k,size(X,2));
centroids = X(rand_indices,:);
max_iterations = 1000;
iteration_count = 0;

while(iteration_count<max_iterations) % Max iteration counter
    iteration_count = iteration_count+1;
    while(prev_centroids~=centroids) % Check thatk-means has converged
        % Assignment Step 1
        for i=1:size(X,1) % For-loop for all data set points x_i
            min_dist = Inf;
            for j=1:k % For-loop for all centroids
                min_dist_current = (sum((X(i,:) - centroids(j,:)) .^ 2));
                if (min_dist_current<min_dist) % Check min distance currently

                    indicator_variable(i,:) = 0; % Reassign entire row to zero
                    indicator_variable(i,j) = 1; % Assign closest data point to 1 for cluster
                    min_dist = min_dist_current; % Update minimum distance for data point

                else

                end
            end
        end
        %Centroid Update Step 2
        prev_centroids = centroids;
        for n=1:k
            denominator = sum(indicator_variable(:,n)); % Calculate denomiator
            numerator = zeros(1,size(X,2)); % Initialise variable to row of zeros
            for m=1:size(X,1)
                numerator = numerator + (indicator_variable(m,n)*X(m,:)); % Calculate numerator
            end
            centroids(n,:) = numerator/denominator; %Update kth centroid position
        end

    end
```

```
end
```

simple_error.m

```
function [err_counts] = simple_error(dataset,k,indicator_variable)
% Function to implement simple errors
% function [err_logic , err_counts] = simple_error(dataset,k,indicator_variable
    )

% Data clusters
cluster_1 = dataset((indicator_variable(:,1)~=0),:);
cluster_2 = dataset((indicator_variable(:,2)~=0),:);
cluster_3 = dataset((indicator_variable(:,3)~=0),:);

S1 = dataset(1:round(size(dataset,1)/k),:);
S2 = dataset((round(size(dataset,1)/k)+1):(round(size(dataset,1)/k)*2),:);
S3 = dataset(((round(size(dataset,1)/k)*2)+1):end,:);

% Checking data members of each cluster
Lia_c1 = ismember(cluster_1,S1,'rows');
Lia_c2 = ismember(cluster_2,S2,'rows');
Lia_c3 = ismember(cluster_3,S3,'rows');

err_numerator = sum(ones(size(Lia_c1,1),1)-Lia_c1)+sum(ones(size(Lia_c2,1),1)-
    Lia_c2)+sum(ones(size(Lia_c3,1),1)-Lia_c3);

l = size(dataset,1);

err_counts = err_numerator/l;

% Logic Calculation

% S1_vec = zeros(size(dataset,1),1);
% vec_1 = ones(50,1);
% S1_vec = [vec_1; S1_vec(51:150)];
%
% S2_vec = zeros(size(dataset,1),1);
% vec_1 = ones(50,1);
% S2_vec = [S2_vec(1:50);vec_1; S2_vec(101:150)];
%
% S3_vec = zeros(size(dataset,1),1);
% vec_1 = ones(50,1);
% S3_vec = [S3_vec(1:100);vec_1];
%
% S = [S1_vec S2_vec S3_vec];
%
% err_numerator_logic = 0;
%
% for i=1:k
%     err_numerator_logic = err_numerator_logic + sum(xor(indicator_variable
    (:,i),S(:,i)));
```

```
% end
%
%
% err_logic = err_numerator_logic/l;
```

occe_error.m

```
function [occe_err, perm_vec] = occe_error(dataset,k,indicator_variable)
% Function for implementation of occe calculation
l = size(dataset,1);

permutation_matrix = perms(1:k);

S1 = dataset(1:round(size(dataset,1)/k),:);
S2 = dataset((round(size(dataset,1)/k)+1):(round(size(dataset,1)/k)*2),:);
S3 = dataset(((round(size(dataset,1)/k)*2)+1):end,:);

err_numerator = zeros(size(permutation_matrix,1),1);
occe_err_temp = zeros(size(permutation_matrix,1),1);

for i=1:size(permutation_matrix,1)

    % Data clusters
    cluster_1 = dataset((indicator_variable(:,permutation_matrix(i,1))~=0),:);
    cluster_2 = dataset((indicator_variable(:,permutation_matrix(i,2))~=0),:);
    cluster_3 = dataset((indicator_variable(:,permutation_matrix(i,3))~=0),:);

    % Checking data members of each cluster
    Lia_c1 = ismember(cluster_1,S1,'rows');
    Lia_c2 = ismember(cluster_2,S2,'rows');
    Lia_c3 = ismember(cluster_3,S3,'rows');

    err_numerator(i) = sum(ones(size(Lia_c1,1),1)-Lia_c1)+sum(ones(size(Lia_c2,1),1)-Lia_c2)+sum(ones(size(Lia_c3,1),1)-Lia_c3);

    occe_err_temp(i) = err_numerator(i)/l;

end

[err_val_min,perm_indx] = min(occe_err_temp);

perm_vec = permutation_matrix(perm_indx,:);
occe_err = min(err_val_min);
```

cluster_mapping_q1.m

```
function [cluster1_str,cluster2_str,cluster3_str] = cluster_mapping_q1(
    perm_vec)
% Mapping cluster assignmetn function for highest combination

if(isequal(perm_vec,[1 2 3]))
    cluster1_str = 'S1';
    cluster2_str = 'S2';
    cluster3_str = 'S3';
elseif(isequal(perm_vec,[1 3 2]))
    cluster1_str = 'S1';
    cluster2_str = 'S3';
    cluster3_str = 'S2';
elseif(isequal(perm_vec,[2 1 3]))
    cluster1_str = 'S2';
    cluster2_str = 'S1';
    cluster3_str = 'S3';
elseif(isequal(perm_vec,[2 3 1]))
    cluster1_str = 'S2';
    cluster2_str = 'S3';
    cluster3_str = 'S1';
elseif(isequal(perm_vec,[3 1 2]))
    cluster1_str = 'S3';
    cluster2_str = 'S1';
    cluster3_str = 'S2';
elseif(isequal(perm_vec,[3 2 1]))
    cluster1_str = 'S3';
    cluster2_str = 'S2';
    cluster3_str = 'S1';
end
```

B: PCA

Q2_1.m

```
%------------------------------------------------------------------------%
% Module: GI07 - Mathematical Methods for Machine Learning
% Assignment : Coursework 2
% Author : Russel Daries, Hugo Philion
% Student ID: 16079408, 14102040
% Question: 2
% Description: Principal Component Analysis
% -----------------------------------------------------------------------%

% clearing memory
clear all
close all
clc

%% 2.1.1 PCA Algorithm

% Done in function pca_algrithm

%% 2.1.2 Implement K-Means Algorithm

% Done in function

%% 2.1.3 Implement on K-Means with PCA on Iris data

load('iris_dataset.mat');
dataset_iris = iris_data;
k_dimension_vec = 1:4;
k_iris = 3;
number_of_trials = 100;

% [centroids_iris, indicator_variable_iris, objective_result] = kMeans_Q2(dataset_iris,k_iris);

% Resultant plot for occe calculation

% occe_err_iris_org = occe_error(dataset_iris,k_iris,indicator_variable_iris);
occe_err_iris_org = zeros(number_of_trials,1);
occe_err_iris_pca = zeros(number_of_trials,size(k_dimension_vec,2));
objective_iris = zeros(number_of_trials,1);
objective_iris_pca = zeros(number_of_trials,size(k_dimension_vec,2));

% 100 Trials
for j=1:number_of_trials
    % Perform K-Means on Original Data
    [centroids_iris, indicator_variable_iris, objective_result] = kMeans_Q2(dataset_iris,k_iris);
```

```
    %objective
    objective_iris(j) = objective_result;
    %occe
    occe_err_iris_org(j) = occe_error(dataset_iris,k_iris,
        indicator_variable_iris);
    % Compute Objective Function
    for i=1:size(k_dimension_vec,2)
        % Perform PCA
        X_hat = pca_algorithm(dataset_iris,k_dimension_vec(i));
        % Perform K-Means on Reduced Data
        [centroids_iris_pca, indicator_variable_iris_pca, objective_result_pca
            ] = kMeans_Q2(X_hat,k_iris);
        % Compute Objective Function
        objective_iris_pca(j,i) = objective_result_pca;
        % Compute occe error
        occe_err_iris_pca(j,i) = occe_error(X_hat,k_iris,
            indicator_variable_iris_pca);
    end
end

occe_err_complete = [occe_err_iris_pca occe_err_iris_org];
objective_complete = [objective_iris_pca objective_iris];
objective_values_occe = zeros(3,1);

%% 2.1.3(a) Three Smallest occe's and value of computed objective

for n=1:5
    [rows,col] = size(occe_err_complete);
    [occe_values,occe_indx] = sort(occe_err_complete(:,n),'ascend');
    lowes_3_occe = occe_values(1:3)
    row_indx = occe_indx(1:3);
    col_indx = repmat(n,3,1);
    for m=1:3
        objective_values_occe(m) = objective_complete(row_indx(m),col_indx(m))
    end
    objective_values_occe = zeros(3,1);
end

%% 2.1.3(b) Mean and standard deviation of occe's and the objectives

occe_mean_pca = mean(occe_err_iris_pca)
occe_mean_org = mean(occe_err_iris_org)
occe_std_pca = std(occe_err_iris_pca)
occe_std_org = std(occe_err_iris_org)

obj_func_pca_mean = mean(objective_iris_pca)
obj_func_org_mean = mean(objective_iris)
obj_func_pca_std = std(objective_iris_pca)
obj_func_org_std = std(objective_iris)
```

```

%User Error bars
figure;
errorbar(k_dimension_vec,occe_mean_pca,occe_std_pca)
hold on
errorbar(k_dimension_vec(end),occe_mean_org,occe_std_org)
xlabel('k','FontSize',15)
ylabel('occe','FontSize',15)
set(gcf, 'Color', 'w');
leg=legend('PCA','Untransformed','Location','Best');
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
set(gca,'XMinorTick','on')
set(gca,'FontSize',15)
grid on;
grid minor;
axis tight;
print('q2_1b_occe','-depsc')
close all;

figure;
errorbar(k_dimension_vec,obj_func_pca_mean,obj_func_pca_std)
hold on
errorbar(k_dimension_vec(end),obj_func_org_mean,obj_func_org_std)
xlabel('k','FontSize',15)
ylabel('objective function','FontSize',15)
set(gcf, 'Color', 'w');
leg=legend('PCA','Untransformed','Location','Best');
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
set(gca,'XMinorTick','on')
set(gca,'FontSize',15)
grid on;
grid minor;
axis tight;
print('q2_1b_obj','-depsc')
close all;

%% 2.1.3(c) Bar Chart Plots
occe_correct_values = zeros(100,5);
% occe_sorted_values = zeros(100,5);

rank_vec = 1:100;

for nn=1:5

    [rows_obj,col_obj] = size(objective_complete);
    [obj_values,obj_indx] = sort(objective_complete(:,nn),'ascend');
    obj_values_sorted = obj_values;
```

```
    occe_correct_values(:,nn) = occe_err_complete(obj_indx,nn);
    occe_sorted_values = sortrows([objective_complete(:,nn) occe_err_complete
        (:,nn)],1);

    formatSpec = 'q2_1b_bar_%d';
    A1 = nn;
    str = sprintf(formatSpec,A1);

    figure;
    bar(occe_sorted_values(:,2))
    xlabel('','FontSize',15)
    ylabel('','FontSize',15)
    set(gcf, 'Color', 'w');
    set(gca,'YMinorTick','on')
    grid on;
    axis tight;
    print(str,'-depsc')
    close all;
end

%% 2.1.4 Visualisation

% Resultant output plots
k_vis=3;
X_hat_2 = pca_algorithm(dataset_iris,2);
% Perform K-Means on Reduced Data
[centroids_iris_pca_2, indicator_variable_iris_pca_2, objective_result_pca_2]
    = kMeans_Q2(X_hat_2,k_vis);

[occe_err_cluster,perm_vec_2] = occe_error(X_hat_2,k_vis,
    indicator_variable_iris_pca_2);

% Mapping Clusters

[cluster1_str,cluster2_str,cluster3_str] = cluster_mapping_iris(perm_vec_2);

cluster_1 = X_hat_2((indicator_variable_iris_pca_2(:,perm_vec_2(1))~=0),:);
cluster_2 = X_hat_2((indicator_variable_iris_pca_2(:,perm_vec_2(2))~=0),:);
cluster_3 = X_hat_2((indicator_variable_iris_pca_2(:,perm_vec_2(3))~=0),:);

% Plot of data with clustering in 2D
figure;
scatter(cluster_1(:,1),cluster_1(:,2),'b*')
hold on
scatter(cluster_2(:,1),cluster_2(:,2),'go','filled')
hold on
scatter(cluster_3(:,1),cluster_3(:,2),'k+')
hold on
```

```
scatter(centroids_iris_pca_2(:,1),centroids_iris_pca_2(:,2),150,'rs','filled');
hl1=xlabel('${\hat{x}}_1$','FontSize',15)
set(hl1, 'Interpreter', 'latex');
hl2= ylabel('${\hat{x}}_2$','FontSize',15)
set(hl2, 'Interpreter', 'latex');
set(gcf, 'Color', 'w');
leg=legend(cluster1_str,cluster2_str,cluster3_str,'Centroids','Location','Best');
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
set(gca,'XMinorTick','on')
set(gca,'FontSize',15)
grid on;
grid minor;
axis tight;
print('q2_4_2d','-depsc')
close all;

X_hat_3 = pca_algorithm(dataset_iris,3);
% Perform K-Means on Reduced Data
[centroids_iris_pca_3, indicator_variable_iris_pca_3, objective_result_pca_3] = kMeans_Q2(X_hat_3,k_vis);

[occe_err_cluster,perm_vec_3] = occe_error(X_hat_3,k_vis, indicator_variable_iris_pca_3);
[cluster1_str,cluster2_str,cluster3_str] = cluster_mapping_iris(perm_vec_3);

cluster_1 = X_hat_3((indicator_variable_iris_pca_3(:,perm_vec_3(1))~=0),:);
cluster_2 = X_hat_3((indicator_variable_iris_pca_3(:,perm_vec_3(2))~=0),:);
cluster_3 = X_hat_3((indicator_variable_iris_pca_3(:,perm_vec_3(3))~=0),:);

% Plot of data with clustering in 3D
figure;
scatter3(cluster_1(:,1),cluster_1(:,2),cluster_1(:,3),'b*')
hold on
scatter3(cluster_2(:,1),cluster_2(:,2),cluster_2(:,3),'go','filled')
hold on
scatter3(cluster_3(:,1),cluster_3(:,2),cluster_3(:,3),'k+')
hold on
scatter3(centroids_iris_pca_3(:,1),centroids_iris_pca_3(:,2), centroids_iris_pca_3(:,3),150,'rs','filled');
hl1=xlabel('${\hat{x}}_1$','FontSize',15)
set(hl1, 'Interpreter', 'latex');
hl2= ylabel('${\hat{x}}_2$','FontSize',15)
set(hl2, 'Interpreter', 'latex');
hl3 = zlabel('${\hat{x}}_3$','FontSize',15)
set(hl3, 'Interpreter', 'latex');
set(gcf, 'Color', 'w');
```

```
leg=legend(cluster1_str,cluster2_str,cluster3_str,'Centroids','Location','Best');
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
set(gca,'XMinorTick','on')
set(gca,'FontSize',15)
grid minor;
grid on;
axis tight;
print('q2_4_3d','-depsc')
close all;
```

PCA Algorithm.m

```
function X_hat = pca_algorithm(X,k)
% Implementation of PCA algorithm
X = mean_norm_data(X);

[dim_row,dim_col] = size(X);

Sigma = (1/dim_row)* (X'*X);

[U,S,V] = svd(Sigma);

U_reduced = U(:,1:k);

phi_map = U_reduced;

X_hat = X*phi_map;
```

kMeans-Q2.m

```
function [centroids, indicator_variable,objective_result] = kMeans_Q2(X,k)
% Function to implement K-Mean Algorithm with call to objective function

% Initialize centroid positions on k of data points selected randomly
indicator_variable = zeros(size(X,1),k);
rand_indices = randsample(size(X,1),k);
prev_centroids = zeros(k,size(X,2));
centroids = X(rand_indices,:);
max_iterations = 1000;
iteration_count = 0;

while(iteration_count<max_iterations) % Max iteration counter
    iteration_count = iteration_count+1;
    while(prev_centroids~=centroids) % Check thatk-means has converged
        % Assignment Step 1
        for i=1:size(X,1) % For-loop for all data set points x_i
            min_dist = Inf;
            for j=1:k % For-loop for all centroidss
                min_dist_current = (sum((X(i,:) - centroids(j,:)) .^ 2));
                if (min_dist_current<min_dist) % Check min distance currently

                    indicator_variable(i,:) = 0; % Reassign entire row to zero
                    indicator_variable(i,j) = 1; % Assign closest data point to 1 for cluster
                    min_dist = min_dist_current; % Update minimum distance for data point

                else

                end
            end
        end
        %Centroid Update Step 2
        prev_centroids = centroids;
        for n=1:k
            denominator = sum(indicator_variable(:,n)); % Calculate denomiator
            numerator = zeros(1,size(X,2)); % Initialise variable to row of zeros
            for m=1:size(X,1)
                numerator = numerator + (indicator_variable(m,n)*X(m,:)); % Calculate numerator
            end
            centroids(n,:) = numerator/denominator; %Update kth centroid position
        end

    end
```

```
end

objective_result = objective_function(X, centroids, indicator_variable);
```

mean_norm_data.m

```
function X = mean_norm_data(X)

X_mean = mean(X);
X_mean_rep = repmat(X_mean,size(X,1),1);

X = X - X_mean_rep;
```

objective_function.m

```
function objective_result = objective_function(X, centroids, indicator_variable)
% Function to calculate the objective function
cluster_1 = X((indicator_variable(:,1)~=0),:);
cluster_2 = X((indicator_variable(:,2)~=0),:);
cluster_3 = X((indicator_variable(:,3)~=0),:);

temp = 0;

for i=1:size(centroids,1)
%     temp = temp + sum((X((indicator_variable(:,i)~=0),:) - repmat(centroids(i,:),size(X((indicator_variable(:,i)~=0),:),1),1)) .^ 2);
    temp = temp + (norm(X((indicator_variable(:,i)~=0),:) - repmat(centroids(i,:),size(X((indicator_variable(:,i)~=0),:),1),1)))^2;
%        temp= temp+pdist2(X((indicator_variable(:,i)~=0),:),repmat(centroids(i,:),size(X((indicator_variable(:,i)~=0),:),1),1),'squaredeuclidean');
end

objective_result = temp;
```

cluster_mapping_iris.m

```
function [cluster1_str,cluster2_str,cluster3_str] = cluster_mapping_iris(
    perm_vec)

if(isequal(perm_vec,[1 2 3]))
    cluster1_str = 'setosa';
    cluster2_str = 'versicolor';
    cluster3_str = 'virginica';
elseif(isequal(perm_vec,[1 3 2]))
    cluster1_str = 'setosa';
    cluster2_str = 'virginica';
    cluster3_str = 'versicolor';
elseif(isequal(perm_vec,[2 1 3]))
    cluster1_str = 'versicolor';
    cluster2_str = 'setosa';
    cluster3_str = 'versicolor';
elseif(isequal(perm_vec,[2 3 1]))
    cluster1_str = 'versicolor';
    cluster2_str = 'virginica';
    cluster3_str = 'setosa';
elseif(isequal(perm_vec,[3 1 2]))
    cluster1_str = 'virginica';
    cluster2_str = 'setosa';
    cluster3_str = 'versicolor';
elseif(isequal(perm_vec,[3 2 1]))
    cluster1_str = 'virginica';
    cluster2_str = 'versicolor';
    cluster3_str = 'setosa';
end
```

C: Kernel Perceptron

Q3_1.m

```
close all;
clear all;

clc

load('train.txt')
load('test.txt')

% Epoch indices
epoch_start = 1;
epoch_stop = 10;

% Kernel Degrees
kernel_degree_max = 2;
kernel_degree_min=2;

% Digit ranges
min_dig=0;
max_dig=9;

% Kernel Type
type = 'poly';

%Initialisation for first for loop

for i=min_dig+1:max_dig+1

    alpha{i}=0;

    X_train{i}=train(1,2:end);
end


%for1 kernel_degree
for k_deg=kernel_degree_min:kernel_degree_max
    %for2 digits
    for epo_current= epoch_start:epoch_stop
        %for3 epochs
        for digit=min_dig+1:max_dig+1
            %for4 training samples
            for train_sample=2432:7291
                % Prediction
                current_sample = train(train_sample,2:end);
                y_prediction = kernel_map(alpha{digit},X_train{digit},
                    current_sample,k_deg,type);
```

```
%                   y_prediction = kernel_map(alpha{},X_train{},X_train{},k_deg,
    type)

                  % Signed output
                  y_signed = signed_output(y_prediction);
                  %comparison
                  y_true=train(train_sample,1);
                       if y_true==digit-1
                           y_comp=1;
                       else
                           y_comp=-1;
                       end

                       if y_signed ~= y_comp
                           %trigger update

                           %Perform paramater update awhen mistake made
                           %Update alpha
                           alpha{digit} = [alpha{digit} y_comp];

                           %Update X's
                           X_train{digit} = [X_train{digit};  train(train_sample
                               ,2:end)];

                           %Update Parameters
                         %alpha, X_train  = update_parameters(alpha{digit,
                             train_sample},y_true,X_train{digit,train_sample},
                             X_train{digit,train_sample+1})
                       end

               end



          end
     end

end
count_correct=0;
prediction_vec = zeros(10,1);
length_validation = 2431;
length_train = 4860;

for i=1:2431
    pred_sample=train(i,2:end);
    y_true_actual = train(i,1);
    for digit_j=1:10
        prediction_vec(digit_j) = kernel_map(alpha{digit_j},X_train{digit_j},
            pred_sample,k_deg,type);
```

```
    end

    [value, index] = max(prediction_vec);
    y_pred_actual = index -1;
    if(y_true_actual == y_pred_actual)
        count_correct = count_correct+1;
    end
    %training_accuracy = count_correct/7291

end


training_accuracy_final = count_correct/length_validation
```

Q3_1b.m

```
function y_pred = kernel_map(alphas,x_previous,x_t,power,type)

% Polynomials Basis
if type == 'poly'
    x_correlate = x_previous*x_t';
    for dim=1:power
        x_correlate = x_correlate.*x_correlate;
    end
    y_pred = alphas* x_correlate;

% Gaussian Basis
elseif type == 'gaussian'
    % gaussian kernel

% Default Basis
else

end


end
```

References

[1] S. Dasgupta, "Algorithms for k-means clustering," 2013 (accessed January 9, 2017). Lecture 3, UC San Diego.

[2] Y. Saw, "Clustering and the k-means algorithm," 2013 (accessed January 10, 2017). Slides of lecture, MIT.

[3] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–9, 1955.

[4] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society of Industrial and Applied Mathematics*, vol. 5, pp. 32–38, March 1957.

[5] D. Bruff, *The Assignment Problem and the Hungarian Method*, 2005 (accessed January 8, 2017).

[6] A. Mills-Tettey, *The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs*, 2007 (accessed January 15, 2017).