Deep Clustering Using Pseudolabels and Centroids

Nikolaos Papadopoulos

Diploma thesis

Supervisor: Aristidis Likas

Ioannina, October 2022

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

ACKNOWLEDGEMENTS

I would like to thank my teacher and supervisor, Professor Aristidis Likas, who gave me the opportunity to work on this very interesting research subject in the field of Machine Learning, which is a field that I have become enthusiastic about during my undergraduate studies. He guided me through the whole process, helping me understand the deep clustering ideas he wanted me to work on and offering unconditional support. He helped me get used to the previously foreign for me process of research and offered suggestions all the way. The knowledge and inspiration I got from him are invaluable.

Moreover, I would like to thank his PhD student, Georgios Vardakas, whose guidance was of great importance, from the bottom of my heart. Georgios was always available for help and oversaw my thesis step-by-step. He answered all my questions regarding practical matters, like the use of Python libraries, the mathematical and algorithmic basis of many aspects of this research and the theory of the clustering ideas he proposed, which are the subject of this thesis. He conveyed to me his own enthusiasm for Machine Learning and Deep Clustering and was like a teacher for me during the whole process.

Last but not least, this effort would not have been possible had it not been for my family, who supported me greatly during all my undergraduate years, and all my close friends, who motivated me and made this journey more fun.

October 2022

Papadopoulos Nikolaos

ABSTRACT

Deep Learning is a very important subset of the field of Machine Learning. It focuses on deep neural networks which are structured in such a way that they can mimic the way that our brain attains knowledge using its network of neurons. It takes advantage of statistics and predictive modelling. Lately, Deep Learning has been used extensively in Clustering. Clustering belongs to unsupervised learning, which is practically the procedure in which a computer learns about a subject without being trained with the help of a set of examples accompanied by their true answers\labels. The aim of clustering is to group objects in clusters. The items belonging to the same cluster need to be similar to each other and differ sufficiently from items of other clusters. Clustering inspects the data and tries to uncover (hidden) structures in them.

In this undergraduate thesis, we study Deep Clustering. Deep Clustering uses traditional clustering algorithms, such as k-means and then exploits the information provided by these algorithms for the training of neural networks. These networks then provide us with representations that have a great capacity in aiding us with the clustering problems.

The introduction of this thesis focuses on laying the foundation knowledge for our clustering techniques. We discuss briefly about neural networks, such as Multilayer Perceptrons and Autoencoders and their architecture parameters, we mention different kinds of traditional clustering algorithms and provide some metrics that help us rate their results. We then discuss the methodology of our approach, mainly how we use the traditional clustering algorithms and then exploit their results with three different methods. The first method extracts representations from a Multilayer Perceptron and the other two from an Autoencoder. The latter use two different techniques, one training the Autoencoder with the centroids acquired from a traditional k-means algorithm run and the other not using them. We then run a clustering algorithm on the latent space of the autoencoder. Our main focus is on the centroids-training method, because we want to prove and emphasize the improvement on clustering performance that this offers. We also explain a criterion that we use in the aid of picking a certain (new) number of clusters for the clustering purposes of each dataset.

The experiments section cites all the datasets and experiments that took place, the graphs that we extracted regarding the clustering performance metrics and the differences in the results of each method. It also rates the suitability of the criterion we propose for the choice of the number of clusters.

Last but not least, we summarize our work in the final section of the thesis and draw conclusions. We also put forward ideas for further research on the matter.

Keywords: machine learning, clustering, deep learning, neural networks, criterion, boxplot, representation, latent space, deep clustering, transformed space

ΠΕΡΙΛΗΨΗ

Η Βαθιά Μάθηση αποτελεί πολύ σημαντικό κομμάτι του τομέα της Μηχανικής Μάθησης. Επικεντρώνεται στα βαθιά νευρωνικά δίκτυα τα οποία είναι δομημένα με τέτοιο τρόπο ώστε να μιμούνται τον τρόπο με τον οποίο ο εγκέφαλός μας αποκτά γνώση χρησιμοποιώντας το δίκτυο των νευρώνων του. Εκμεταλλεύεται τη στατιστική και την προγνωστική μοντελοποίηση. Τον τελευταίο καιρό, η βαθιά μάθηση έχει χρησιμοποιηθεί εκτενώς στην ομαδοποίηση. Η ομαδοποίηση ανήκει στα προβλήματα μάθησης χωρίς επίβλεψη. Είναι πρακτικά η διαδικασία κατά την οποία ένας υπολογιστής μαθαίνει για ένα θέμα χωρίς να έχει εκπαιδευτεί με τη βοήθεια ενός συνόλου παραδειγμάτων που συνοδεύονται από τις πραγματικές απαντήσεις τους (ετικέτες). Στόχος της συσταδοποίησης είναι η ομαδοποίηση αντικειμένων σε ομάδες. Τα αντικείμενα που ανήκουν στην ίδια συστάδα πρέπει να είναι παρόμοια μεταξύ τους και να διαφέρουν αρκετά από τα αντικείμενα άλλων συστάδων. Η συσταδοποίηση εξετάζει τα δεδομένα και προσπαθεί να αποκαλύψει (κρυφές) δομές σε αυτά.

Σε αυτή την προπτυχιακή διπλωματική εργασία, μελετάμε τη βαθιά συσταδοποίηση (Deep Clustering). Η βαθιά συσταδοποίηση χρησιμοποιεί παραδοσιακούς αλγορίθμους συσταδοποίησης, όπως ο k-means και στη συνέχεια εκμεταλλεύεται τις πληροφορίες που παρέχουν αυτοί οι αλγόριθμοι για την εκπαίδευση νευρωνικών δικτύων. Αυτά τα δίκτυα στη συνέχεια μας παρέχουν αναπαραστάσεις ικανές να μας βοηθούν στα προβλήματα ομαδοποίησης.

Η εισαγωγή αυτής της διατριβής επικεντρώνεται στη παρουσίαση των βασικών γνώσεων για τις τεχνικές ομαδοποίησης που εφαρμόζουμε. Συζητάμε εν συντομία για τα νευρωνικά δίκτυα, όπως τα Multilayer Perceptrons και οι Autoencoders και τις παραμέτρους της αρχιτεκτονικής τους, αναφέρουμε διάφορα είδη παραδοσιακών αλγορίθμων ομαδοποίησης και παρέχουμε ορισμένες μετρικές που μας βοηθούν να αξιολογήσουμε τα αποτελέσματά τους. Στη συνέχεια εξηγούμε τη μεθοδολογία της προσέγγισής μας, κυρίως πώς χρησιμοποιούμε τους παραδοσιακούς αλγορίθμους ομαδοποίησης και στη συνέχεια αξιοποιούμε τα αποτελέσματά τους με τρεις διαφορετικές μεθόδους. Η πρώτη μέθοδος εξάγει αναπαραστάσεις από ένα Multilayer Perceptron και οι άλλες δύο από έναν Autoencoder. Οι τελευταίες χρησιμοποιούν δύο διαφορετικές τεχνικές, η μία εκπαιδεύει τον Autoencoder με τα κεντροειδή που αποκτήθηκαν από την εκτέλεση ενός παραδοσιακού αλγορίθμου k-means και η άλλη δεν τα χρησιμοποιεί. Στη συνέχεια εκτελούμε έναν αλγόριθμο ομαδοποίησης στον συμπιεσμένο χώρο του autoencoder. Εστιάζουμε κυρίως στη μέθοδο εκπαίδευσης με κεντροειδή, επειδή θέλουμε να αποδείξουμε και να τονίσουμε τη βελτίωση στην απόδοση της συσταδοποίησης

που αυτή προσφέρει. Προσδιορίζουμε, επίσης, ένα κριτήριο που χρησιμοποιούμε για την επιλογής ενός συγκεκριμένου (νέου) αριθμού συστάδων για τους σκοπούς της συσταδοποίησης κάθε συνόλου δεδομένων.

Στην ενότητα "Πειραματική μελέτη" παραθέτουμε όλα τα σύνολα δεδομένων και τα πειράματα που πραγματοποιήθηκαν, τα γραφήματα που εξήγαμε σχετικά με τις μετρικές απόδοσης της συσταδοποίησης και τις διαφορές στα αποτελέσματα κάθε μεθόδου. Αξιολογεί επίσης την καταλληλότητα του κριτηρίου που προτείνουμε για την επιλογή του αριθμού των συστάδων.

Τέλος, στην τελευταία ενότητα της διατριβής συνοψίζουμε την εργασία μας και εξάγουμε συμπεράσματα. Επίσης, διατυπώνουμε ιδέες για περαιτέρω έρευνα επί του θέματος.

Λέξεις-κλειδιά: μηχανική μάθηση, ομαδοποίηση, βαθιά μάθηση, νευρωνικά δίκτυα, κριτήριο, θηκόγραμμα, αναπαράσταση, λανθάνων χώρος, βαθιά ομαδοποίηση, μετασχηματισμένος χώρος.

Contents

1. Introduction 11

1.1 Machine Learning 11

1.2 Clustering 11

1.2.1 Hierarchical clustering 12

1.2.2 Agglomerative clustering 13

1.2.3 K-means clustering 15

1.3 Neural networks 16

1.3.1 Convolutional Neural Network 16

1.3.2 Multilayer Perceptron (MLP) 19

1.3.3 Autoencoder 23

1.4 Deep Clustering 25

1.4.1 Introduction 25

1.4.2 Essential components of Deep Clustering 25

1.4.3 Learning Deep Representations 25

1.4.4 Deep Clustering Loss Functions 26

1.4.5 Deep Neural Network Architecture 27

1.5 Methods of assessing datapoint spaces and clustering results 27

1.5.1 t-distributed stochastic neighbour embedding (t-SNE): 27

1.5.2 Normalised Mutual Info Score (NMI) 28

1.5.3 Silhouette score/coefficient 29

1.5.4 Adjusted rand score (ARI) 30

1.5.5 Accuracy score 31

1.6 Data scalers 31

1.7 Model parameters and optimisers 32

1.7.1 Adam optimiser 32

1.7.2 Cross entropy loss 34

1.7.3 Softmax activation function 35

1.7.4 Leaky ReLU 35

1.8 Thesis contribution 36

1.9 Thesis outline 37

2. The proposed method 38

2.1 Deep Clustering using pseudolabels 38

2.2 Deep Clustering using cluster centroids 40

2.3 The proposed overclustering criterion based on silhouette score 42

3. Experimental study 45

3.1 Datasets description 45

3.2 Boxplots and overclustering criterion for autoencoder-trained-with-centroids method 49

3.3 Boxplots and overclustering criterion for MLP-transformed space method 83

3.4 Boxplots and overclustering criterion for autoencoder-trained-without centroids method 102

3.5 Boxplots and overclustering criterion for CNN-deep-network clustering method 112

3.5.1 Training the CNN-autoencoder model without cluster centroids (proof of concept) 123

3.6 Table of metrics for each clustering method 134

3.7 t-SNE representation of initial and latent spaces 138

4. Conclusions and future work 148

4.1 Conclusion 148

4.2 Suggestions for future work 149

Bibliography 150

Appendix 153

Table of figures

Figure 1: Hierarchical clustering dendogram 12

Figure 2: The joining of clusters visualised in hierarchical clustering 13

Figure 3: Linkage criteria, image found in [4], which also mentions: “Inspired by Figure 17.3 of Manning et al. (2008)” 15

Figure 4: ReLU graph 18

Figure 5: Convolution operation, image from [5], pp. 12 19

Figure 6: The perceptron structure, as shown in the source [18]. 20

Figure 7: Example of a Multilayer Perceptron 21

Figure 8: Autoencoder, copied from chapter 14 of [1] 24

Figure 9: Leaky ReLU, a: slope coefficient 36
Figure 10: Two-dimensional dataset containing two clusters 38
Figure 11: Clustering output of k-means with k = 10 39
Figure 12: Basic algorithm steps of the proposed method 40
Figure 13: The gaussian rings dataset 46
Figure 14: The squeezed gaussian blobs dataset 46
Figure 15: The moons dataset 47
Figure 16: MNIST dataset sample, taken from the Wikipedia article in [20] 48
Figure 17: Fashion-MNIST dataset examples 49
Figure 18: 10x_73k dataset initial space t-SNE representation 138
Figure 19: 10x_73k dataset latent space t-SNE representation 139
Figure 20: Pendigits dataset initial space t-SNE representation 139
Figure 21: Pendigits dataset latent space t-SNE representation 140
Figure 22: Gaussian rings dataset initial space t-SNE representation 140
Figure 23: Gaussian rings dataset latent space t-SNE representation 141
Figure 24: Squeezed gaussian blobs dataset initial space t-SNE representation 141
Figure 25: Squeezed gaussian blobs dataset latent space t-SNE representation 142
Figure 26: Moons dataset initial space t-SNE representation 142
Figure 27: Moons dataset latent space t-SNE representation 143
Figure 28: Iris dataset initial space t-SNE representation 143
Figure 29: Iris dataset latent space t-SNE representation 144
Figure 30: Australian dataset initial space t-SNE representation 144
Figure 31: Australian dataset latent space t-SNE representation 145
Figure 32: MNIST dataset initial space t-SNE representation 145
Figure 33: MNIST dataset latent space t-SNE representation 146
Figure 34: Fashion-MNIST dataset initial space t-SNE representation 146
Figure 35: Fashion-MNIST dataset latent space t-SNE representation 147

Table of algorithms

Algorithm 1: Agglomerative clustering 13
Algorithm 2: K-means clustering 16
Algorithm 3: Overclustering criterion based on silhouette coefficient algorithm 43

List of tables

Autoencoder-trained-with-centroids method parameters for 10x_73k dataset 50
Autoencoder-trained-with-centroids method parameters for pendigits dataset 55
Autoencoder-trained-with-centroids method parameters for gaussian rings dataset 60
Autoencoder-trained-with-centroids method parameters for gaussian squeezed blobs dataset 65
Autoencoder-trained-with-centroids method parameters for moons dataset 69
Autoencoder-trained-with-centroids method parameters for Australian dataset 74
Autoencoder-trained-with-centroids method parameters for iris dataset 79
MLP-representations method parameters for 10x_73k dataset 84
MLP-representations method parameters for pendigits dataset 86
MLP-representations method parameters for gaussian rings dataset 88
MLP-representations method parameters for gaussian squeezed blobs dataset 90
MLP-representations method parameters for moons dataset 92
MLP-representations method parameters for australian dataset 94
MLP-representations method parameters for iris dataset 96
MLP-representations method parameters for MNIST dataset 98
MLP-representations method parameters for Fashion-MNIST dataset 100
Autoencoder-trained-without-centroids method parameters for 10x_73k dataset 103
Autoencoder-trained-without-centroids method parameters for pendigits dataset 104
Autoencoder-trained-without-centroids method parameters for gaussian rings dataset 105
Autoencoder-trained-without-centroids method parameters for squeezed gaussian blobs dataset 107
Autoencoder-trained-without-centroids method parameters for moons dataset 108
Autoencoder-trained-without-centroids method parameters for australian dataset 110
Autoencoder without centroids method parameters for iris dataset 111
CNN-deep-network clustering method parameters for MNIST dataset 113
CNN-deep-network clustering method parameters for fashion-MNIST dataset 118
CNN-deep-network-without-centroids clustering method parameters for MNIST dataset 124
CNN-deep-network-without-centroids clustering method parameters for fashion-MNIST dataset . 129
Table of k-means clustering metrics for each dataset 136
Table of agglomerative clustering metrics for each dataset 138

1. Introduction

1.1 Machine Learning

Machine learning is the process in which a computer program learns how to perform a task from given data. In this process, the program adjusts to the needs of the given goal and improves its performance by experience [1]. The problems solved by machine learning are generally too difficult to solve with fixed programs. The main characteristic of this domain is programming something, for example a model, to learn to do the task by itself, rather than instructing it specifically on how to attain the goal. We usually use examples for the model to process and learn from. In mathematical terms, we define an example as a vector $x \epsilon R^n$ where each entry x_i of the vector is a feature.

Machine learning algorithms are split into two categories, unsupervised and supervised. Unsupervised algorithms learn from the features of each datapoint and try to find hidden patterns and information. An example of unsupervised learning is data clustering, which aims to create clusters of data points, in a way that the data points grouped together are similar to each other and the ones in different clusters are not. On the other hand, in supervised learning we associate each datapoint with a label. During training, we inspect many different examples of random vectors x whose label is a value y and we gradually learn to predict the corresponding label y given the input x. It is important to note that unsupervised and supervised learning are not completely formal concepts, but they do help us make distinctions between different kinds of machine learning problems we may encounter in the world. In mathematical terms, supervised algorithms use a training set in the form of D = $\{(x_i , y_i)\}$, where x_i is the input datapoint and y_i is the known datapoint label. The model acquires its knowledge from the matching of each training example to a known label. Then, its performance is tested in a test set, which contains unlabelled data. We evaluate how well the model predicts the labels of the examples in the test set. However, unsupervised algorithms use a training set in the form of D = $\{(x_i)\}$ (unlabelled data).

1.2 Clustering

Clustering is the process of assigning labels to unlabelled elements. In essence, the purpose of any clustering algorithm is to group data points together in clusters. The components of the same cluster ought to be similar with each other and dissimilar to the components of the

other clusters. The algorithm tries to uncover unknown properties of the dataset. Clustering can be characterized by the use of resemblance or dissemblance measures between the datapoints [2].

1.2.1 Hierarchical clustering

Hierarchical clustering [3] is a procedure that transforms a collection of items into a sequence of nested clusters. In mathematical terms:

$X = \{x_1, x_2, \ldots, x_n\}$: the initial set of items to be clustered by the algorithm

C_i : a subset of a previous partition. If we split a set X into two partitions, C_i and C_j, then the two partitions must satisfy the following: $C_i \cup C_j = X$ and $C_i \cap C_j = \Phi$, where f denotes the empty set. A partition is nested inside a bigger partition, if each element of it is a subset of the bigger one. In a hierarchical clustering, every partition of the sequence is nested into the next in the sequence. A convenient way to represent a hierarchical structure is a dendrogram. A dendrogram is a tree structure that consists of levels of nodes, each node representing a cluster of the partition. Traversing a vertical line in this tree will give us a series of clusters that are nested inside on another. In addition, a horizontal line in the tree contains the groups of a certain clustering.

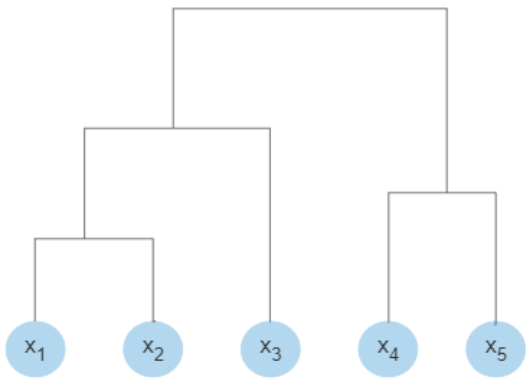


Figure 1: Hierarchical clustering dendogram

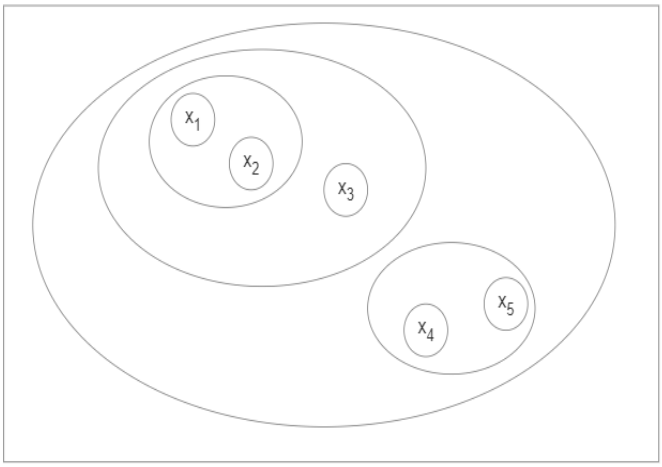


Figure 2: The joining of clusters visualised in hierarchical clustering

1.2.2 Agglomerative clustering

Agglomerative clustering is a type of hierarchical clustering. The algorithm maintains an "active set" of clusters [4] and in every step decides which two clusters need to be merged. The merging of the two clusters results in the addition of their union in the active set and in their own removal from it. This procedure goes on until only one cluster exists in the active set.

The pseudocode for this algorithm, as mentioned in paper [4]:

Algorithm 1: Agglomerative clustering

1: **Input**: Data vectors $\{x_n\}_{n=1}^{N}$, group-wise distance $DIST(G, G')$

2: A ← ∅ // Active set starts out empty.

3: **for** n ← 1 . . . N **do** // Loop over the data.

4: A ← A ∪ $\{\{x_n\}\}$ // Add each datum as its own cluster.

5: **end for**

6: T ← A // Store the tree as a sequence of merges. In practice, pointers.

7: **while** |A| > 1 **do** // Loop until the active set only has one item.

8: $G_1^*, G_2^* \leftarrow argmin_{G_1,G_2 \in A} DIST(G_1, G_2)$ // Choose pair in A with

// best distance.

9: A ← (A\ $\{G_1^*\}$) \ $\{G_2^*\}$ // Remove each from active set.

10: A ← A ∪ $\{G_1^* \cup G_2^*\}$ // Add union to active set.

11: T ← T ∪ $\{G_1^* \cup G_2^*\}$ // Add union to tree.

12: **end while**

13: **Return**: Tree T

It becomes apparent that the distance metric in the algorithm is of great importance. First of all, we need to count distances between **groups** of data points. Here are some choices, where we consider two groups G = $\{x_n\}_{n=1}^N$ and G' = $\{y_m\}_{m=1}^M$, where N and M are not necessarily equal.

1) **The Single-Linkage Criterion**: This criterion merges the two groups based on the shortest distance over all existing pairs between them. In mathematical terms, DIST-SINGLELINK($\{x_n\}_{n=1}^N$, $\{y_m\}_{m=1}^M$) = $min_{n,m}||x_n - y_m||$.
2) **The Complete-Linkage Criterion**: According to this criterion, we need to merge the groups in terms of the maximum distance between all possible pairs i.e., DIST-COMPLETELINK($\{x_n\}_{n=1}^N$, $\{y_m\}_{m=1}^M$) = $max_{n,m}||x_n - y_m||$.
3) **The Average-Linkage Criterion**: We compute the distance between the groups by averaging the distances between all possible pairs : DIST-AVERAGE($\{x_n\}_{n=1}^N$, $\{y_m\}_{m=1}^M$) = $\frac{1}{NM}\sum_{n=1}^N \sum_{m=1}^M ||x_n - y_m||$.
4) **The Centroid Criterion**: The final solution is to compute the distance between clusters by calculating the difference between their centroids: DIST-CENTROID($\{x_n\}_{n=1}^N$, $\{y_m\}_{m=1}^M$) = $||(\frac{1}{N}\sum_{n=1}^N x_n) - (\frac{1}{M}\sum_{m=1}^M y_m)||$.

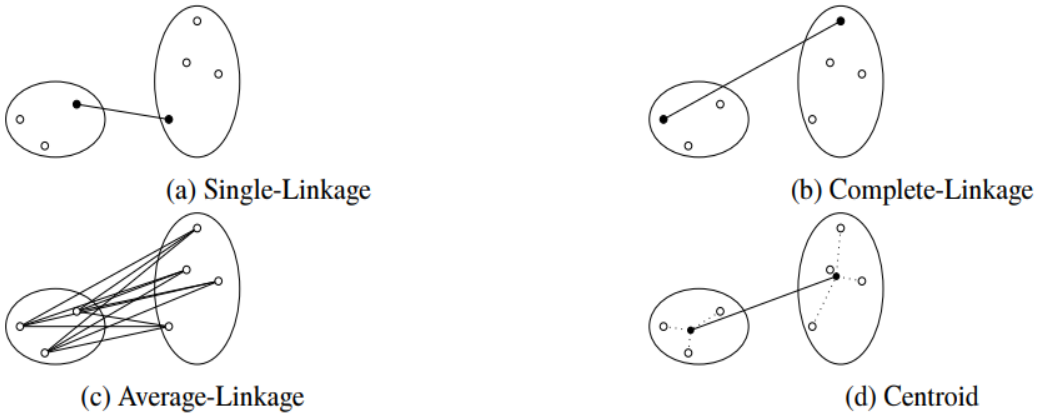


Figure 3: Linkage criteria, image found in [4], which also mentions: “Inspired by Figure 17.3 of Manning et al. (2008)”

1.2.3 K-means clustering

The k-means clustering algorithm [10] is a widely used method for clustering data in many fields. The algorithm needs a fixed number of clusters k in order for it to work. At first, the k cluster centers (w_1, w_2, … , w_k) are initialised to one of the n input vectors (i_1, i_2, … , i_n). Generally, C_j is the symbol of the j^{th} cluster. The Euclidean distance is generally used in order to determine the quality of the clustering. Therefore, the error is calculated using the following formula (from [10]):

$$E = \sum_{j=1}^{k} \sum_{i_l \in C_j} | \, i_l - w_j \, |^2$$

An effective choice of k is of great importance for the quality of the clusters, as is the random initialisation of the cluster centers. As a result, the algorithm doesn’t guarantee that it will converge on the optimal solution when it comes to the error mentioned above. We can tackle the second problem of unpredictability by running the algorithm many times and eventually keeping the best solution, which is the solution that minimises the error above. The k-means clustering algorithm is as follows:

Algorithm 2: K-means clustering

Input: Data vectors $\{x_n\}_{n=1}^{N}$, $x_n = (x_{n1}, x_{n2}, \ldots, x_{nd})$, where d is the dimension of the data and N is the count of datapoints in the dataset. We also need K, which is the number of clusters.

1: Random initialiasation of M centroids, with each centroid O_j being a vector $\mu_j = (\mu_{j1}, \mu_{j2}, \ldots, \mu_{jd})$.

2: t ← 1 // Initialise number of repetitions

3: **while true do**

4: **for** each x_n in the dataset:

5: Compute the Euclidean distance $d(x_n, \mu_m)$ of the vector from every centroid μ_m. Assign x_n to the cluster of its closest centroid.

6: **end for**

5: Compute the new centroid $\mu_j(t + 1)$ as the means of the data points of the cluster it represents

6: **If** any of the centroids changed **then**

7: t ← t + 1

8: **else**

9: Return the centroids μ_j

10: **end if**

11: **end while**

1.3 Neural networks

1.3.1 Convolutional Neural Network

The Convolutional Neural Network (CNN) [5] is a very efficient architecture in many a machine learning and computer vision application. It performs especially well in image-related tasks.

An important aspect of CNN models to discuss is the definition of a tensors. Tensors are essentially higher-than-second order matrices. A helpful way to visualise tensors is a number

of channels, with each channel containing a matrix. For example, $x \epsilon R^{HxWxD}$ is an order 3 tensor, which is equal to D channels of HxW matrices. Each element in a tensor can be indexed by a triplet in the form of (i, j, k). We can also note that scalar values are zero-order tensors, vectors are 1-order tensors and matrices are 2-order tensors. The input and intermediate representations in CNN models are always tensors.

As far as the CNN architecture is concerned, the input is usually an order 3 tensor, as we have already mentioned, which is a HxW image containing 3 color channels. The types of layers a CNN model might have are convolution layers, pooling layers, normalization layers, fully connected layers, loss layers, etc.

The loss layer is the last layer. It is used in order to compute how close the prediction of the network is to the ground truth of the input. A simple loss function would be z $z = \frac{1}{2}\left|\left|t - x^L\right|\right|^2$, where t is the real value and x^L is the prediction. Of course, more complex forms of error are often used. The error is then used to optimise the parameters of the network by using the Stochastic Gradient Descent algorithm (SGD) and also error back propagation.

Another crucial part to explain is the way intermediate layers pass information to one another (from the previous the next, which happens in forward-backpropagation). The input of every intermediate layer is essentially also a 3-dimensional tensor, but the mini-batch strategy is used, which means that the tensors become 4-dimensional, as we need an extra dimension for the batch size. The layer will transform this input x to an output y, which will be passed on to the next layer.

The Rectified Linear Unit (ReLU) layer is a kind of layer which doesn't change the size of the input. Its formula is $y_{i,j,d} = max\{0, x^l_{i,j,d}\}$. There are no parameters inside a ReLU layer. Its purpose is to increase the nonlinearity of the CNN. This helps us if we take into consideration the non-linear aspect pixels in day-to-day images. The sigmoid function has also been tried in CNNs, with worse results. Note: $\sigma(x) = \frac{1}{1 + exp(x)}$

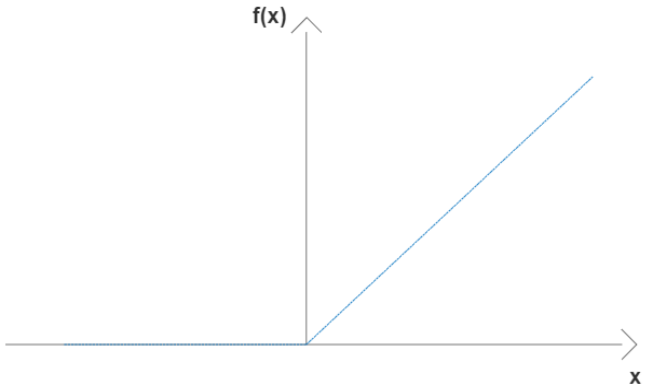


Figure 4: ReLU graph

The Convolutional Layer is the one which is of the greatest importance in CNNs, hence their name. We must first define what convolution is. Let's suppose we have a convolution kernel of a certain size and an input image. Starting from the top left corner of the image, we overlap the kernel on top of the image, calculating the products of the numbers in the same position and summing all these together, in order to get the number of the output in the corresponding location. This process is repeated all over the input image, until we reach the bottom right corner. A stride s makes it skip s-1 pixels, both horizontally and vertically. The kernel is also a 3-dimensional tensor, and the convolution process is repeated in each channel. We sum the products to get the result.

Let f be an order 4 tensor in $R^{HxWxD^{l}xD}$, where HxW is the size of the input image-matrix, D is the number of channels and l is the layer number. Let the stride be 1 and no padding is used. Hence, we have y (or x^{l+1}) in $R^{H^{l+1}\,x\,W^{l+1}\,x\,D^{l+1}}$, with $H^{l+1} = H - H + 1^{l}$, $W^{l+1} = W^{l} - W + 1^{l}$, and $D^{l+1} = D$. In mathematical terms, the convolution process can be expressed as an equation: $y_{i^{l+1},\,j^{l+1},d^{l}} = \sum_{i=0}^{H}\sum_{j=0}^{W}\sum_{d^{l}=0}^{D^{l}} f_{i,j,d^{l},d} \times x^{l}_{i^{l+1}+\,i,\,j^{l+1}+\,j,d^{l}}$.

This equation is repeated for any spatial location. Usually, a bias term is added to the output of each level.

The benefits of convolution is that it helps us detect edges, which helps us extract certain features from the image. This helps the next layers to activate only for specific patterns, for example collections of edges that form certain figures. The next layers in the architecture take advantage of these, so that meaningful object parts can be identified. In addition, the same convolution kernel is used in all locations, which reduces the number of parameters that must be trained in the network.

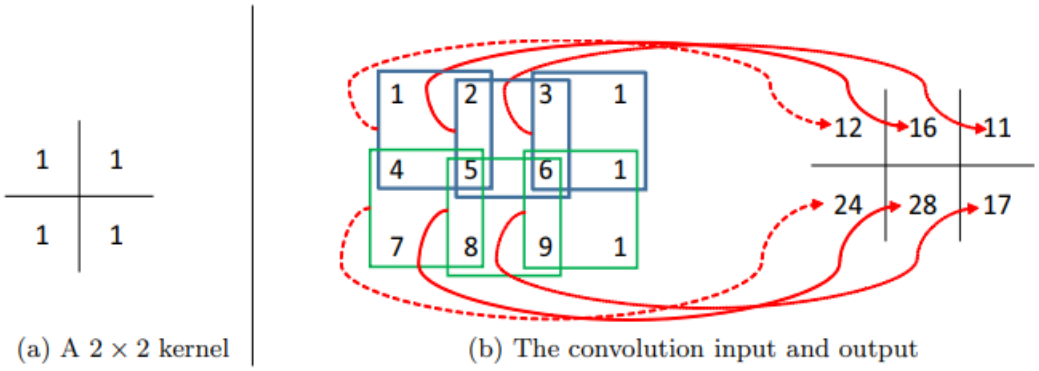


Figure 5: Convolution operation, image from [5], pp. 12

Fully connected layers are also worth mentioning. They are useful towards the end of a CNN model. In these layers, each element requires the values of the whole layer input for its computational purposes.

Last but not least, we will discuss about the pooling layer, using the same notations that we used in the convolution layer. There are no weights/parameters in the pooling operation. Assuming that H divides H^l and W divides W^l , the output of the pooling operation will be a third order tensor of size $H^{l+1} \times W^{l+1} \times D^{l+1}$ with $H^{l+1} = \frac{H^l}{H}$, $W^{l+1} = \frac{W^l}{W}$, $D^{l+1} = D$. Essentially, the layer, acting in each channel seperately, divides the $H^l \times W^l$ input matrix into $H^{l+1} \times W^{l+1}$ subregions, which are $H \times W$ in size. Then, an operation takes place in order to map each subregion into a single number. There are two kinds of such operations:

1) Max pooling: The operation maps each partition to the maximum pixel value in it.
2) Average pooling: As suggested by its name, the operation sums all the pixel values in the subregion, computes the average and then maps the partition to the result.

1.3.2 Multilayer Perceptron (MLP)

The main component of a Multilayer Perceptron is Rosenblatt's Perceptron [18]. Frank Rosenblatt was a psychologist who introduced this model which mimics the way our brain works. Its name was inspired by the piece of technology used to recreate it at the time. A Perceptron consists of a main element that has n different inputs x_i, i = 1, …, n. Each input is multiplied by a certain weight that can only have the value of +1 or -1 and then all these results

are added together. Then, if this sum is more than a certain threshold θ, then the output of the Perceptron is 1, otherwise it is 0.

The perceptron was intended for the transmission of signals. It consists of if sensory, association, output and response units. The receptor of the perceptron resembles the function of the eye and its photocells. The units that simulate the photocells pass the input on to the next level of the network, the association level. This part of the perceptron acts exactly like we described in the paragraph above. Then the outputs of each unit are multiplied with their respective weight and the sum of all this is calculated. It is obvious that the output is binary.

If the output of the whole network is not correct in regard to the prediction it is supposed to make, then the weight will be adjusted. The adjustment of weights is a very interesting subject and one can understand sufficiently how it's done in the paragraphs that follow, which will go into detail about how Multilayer Perceptrons work.

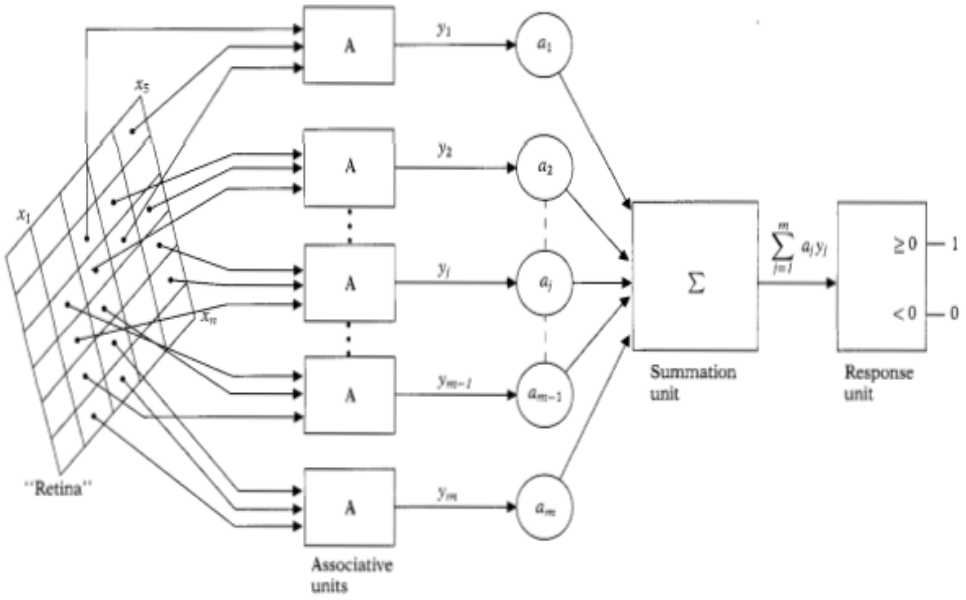


Figure 6: The perceptron structure, as shown in the source [18].

Multilayer Perceptrons [6] have become very important networks for solving machine learning problems over the past years. There has been a lot of research done on these networks, yet it's still possible to make improvements on the way they work, since the choice of their parameters is a bit random and experience-based.

MLPs consist of neurons arranged in layers. In a fully-connected network, each neuron connects to every neuron of the next layer. These networks consist of a minimum of three layers: one is the input layer, through which we input the data of the real world problem we need to solve, then we have the minimum of one hidden layer and then the output layer, which produces the predictions of the model. Layers have activation functions. Their purpose is to pass the output of a neuron through a function that transforms it. Input and output layers use linear activation functions, whereas hidden layers use nonlinear functions. One crucial part of every MLP is the weights of the connections. Each connection is associated with a certain weight, which is randomly-created in the beginning and updated afterwards in each stage of the training procedure. The output of each neuron is multiplied by the weight of the connection and the result is fed to the corresponding neuron of the next layer. Optionally, each node might have a bias term associated with it. In this case, the bias value is added to the weight-output product. However, in our mathematics discussion we will not take bias terms into account.

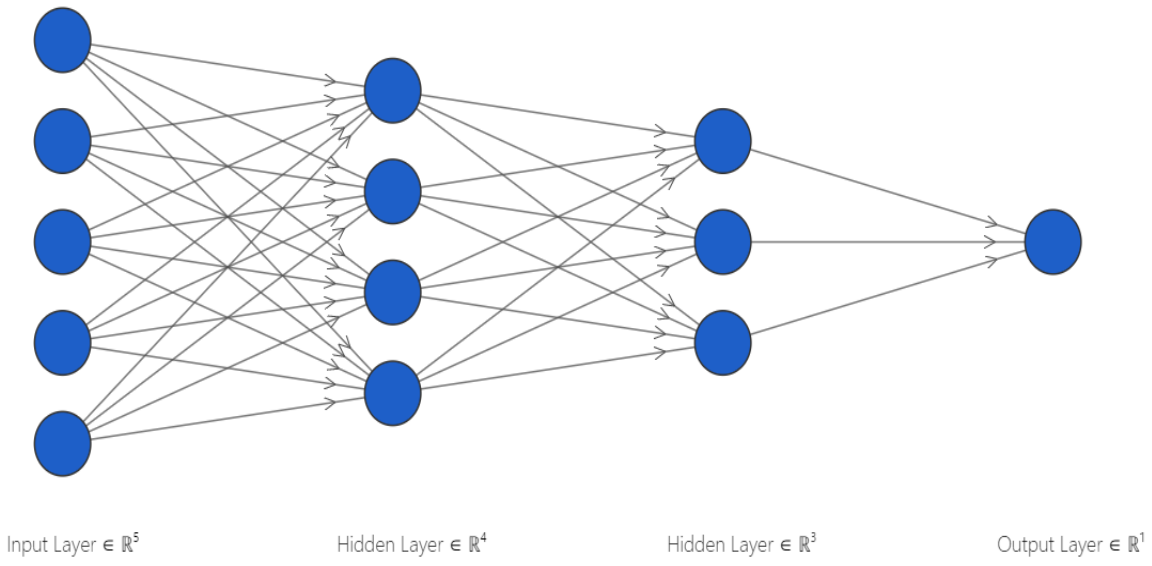


Figure 7: Example of a Multilayer Perceptron

Let's suppose we have a training set of N training tuples (x_p, t_p), where p represents the tuple index in the set. x_p represents the N-dimensional input vector of the pth tuple and y_p represents the M-dimensional output of our network when we train it on the features of x_p. For ease of explanation and mathematical definition, the bias terms are handled by appending a

1 to the vector x_p, which is consequently denoted by $x_p(N+1)$. The input to the j[th] hidden unit, netp(j), is given by the following formula:

$net_p(j) = \sum_{k=1}^{N+1} w_{hi}(j,k) \cdot x_p(k), 1 \le j \le N_h,$

with the output for the p[th] training tuple, $O_p(j)$, being expressed by $O_p(j) = f(net_p(j))$.

We usually choose the sigmoidal function as the nonlinear activation function: $f(x) = \frac{1}{1+e^{-x}}$, but another choice could be the hyperbolic tangent function: $f(x) = \frac{e^{ax} - e^{-ax}}{e^{ax} + e^{-ax}}$.

A good way to measure the performance of a Multilayer Perceptron is the Mean Square Error (MSE) metric. Its formula is:

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} E_p = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_p(i)]^2$$

In essence, E_p gives us the error of the p[th] item in the current training set. By taking advantage of this, we can calculate the mapping error for the i[th] output unit, using the formula:

$$E_i = \frac{1}{N_v} \sum_{p=1}^{N_v} [t_p(i) - y_p(i)]$$

while the i[th] output for the p[th] training tuple can be expressed by:

$$y_p(i) = \sum_{k=1}^{N+1} w_{oi}(i,k) \cdot x_p(k) + \sum_{j=1}^{N_h} w_{oh}(i,j) \cdot O_p(j)$$

Note: $w_{oi}(i,k)$ represents the weights from the input to the output neurons, whereas $w_{oh}(i,j)$ represents the weights from the hidden to the output neurons.

The values that the weights attain during the training process is the “knowledge” of the MLP. They represent the way the network solves the problem it was trained to solve. As a result, the training process focuses on adjusting the weights of the connections in such a way that

they minimise the error of the network, in terms of the expected output for a given input of the training set. The procedure consists of two main parts:

1) **Forward pass:** During this stage, the input layer takes the next training vector and passes its features through the connections to the next layers, multiplied by the corresponding connection weights. In each of the next layers, the neurons calculate the output with the methods we have already discussed, and pass their output to the next layer and so on. Once the values are in the output layer, they are passed through the activation function and we eventually have the output y_p for the current training vector. In the next stage, we will compare it to the target t_p, in order to asses the network's performance.

2) **Backpropagation:** In this stage, all the weights and biases are updated using the formula $w(j,i) \leftarrow w(j,i) + Z \cdot \frac{-\partial E_p}{\partial w(j,i)}$,
where Z is a constant that is called learning factor. We can use the chain rule to express the gradients:

$\frac{\partial E_p}{\partial w(j,i)} = -\delta_p(j) \cdot O_p(i)$,

where $\delta_p(j) = \frac{-\partial E_p}{\partial net_p(j)}$ is called the delta function. Its calculation differs for the output and hidden neurons, with it respectively being:

$$\delta_p(k) = f'(net_p(j)) \cdot (t_p(k) - O_p(k))$$

$$\delta_p(j) = f'(net_p(j)) \sum_n \delta_p(n) w(n,j),$$

where f' is the first derivative of the activation function and n is the index of units in subsequent layers that are connected to the jth unit in the previous layer.

1.3.3 Autoencoder

Autoencoders (chapter 14 of [1]) are neural networks whose purpose is to try to copy their input to their output, but their usefulness lies in not being able to exactly copy everything in their input, rather the more important features of it. The network has two main components: an encoder function h = f(x) and a decoder function that reconstructs this r = g(h).

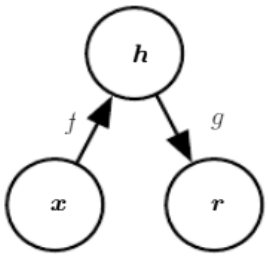


Figure 8: Autoencoder, copied from chapter 14 of [1]

It is of great importance that an autoencoder doesn't copy the inputs to the letter. There are restrictions that force these networks to copy their input only in approximation and also discard data that is not similar to the data it was trained on. These models usually memorise useful properties of the data.

Autoencoders have been studied and researched on for decades. Their more historical usage lies in dimensionality reduction and feature learning. However, these days these networks can act like feedforward models.

In the architectural side of things, the encoder part of an autoencoder is actually a fully-connected neural network, as is the decoder part. Often, this parts can actually be single-layer parts, but it has been proven that adding more layers to these networks can not only reduce the computational cost of estimating some functions, but also decrease the number of training examples needed. Moreover, it has been documented that deeper autoencoders provide us with better compression rates. Right in the middle of the encoder and the decoder lies a single layer of a few neurons. It represents what we call the latent space and is where the input's information is compressed. It is also called the bottleneck. This part feeds the data to the decoder part of the autoencoder, which tries to uncompress the data and output a representation of the input. A common choice for the decoder is a mirror of the network of the encoder.

1.4 Deep Clustering

1.4.1 Introduction

Deep Clustering [29] is an impactful field which has been aided by the improvement in computational capabilities of today's computers. It has impacted research and applications in the Machine Learning, Artificial Intelligence and Computer Vision domains, among many others. When it comes to Machine Learning, its contribution has been extended to unsupervised tasks, taking advantage of deep networks trained during a preliminary unsupervised stage. These networks offer representations that have been known to significantly improve the results of algorithms, especially when it comes to Clustering. There are many clustering techniques that have taken advantage of the representations of deep networks in order to enhance clustering results.

1.4.2 Essential components of Deep Clustering

Before we present deep clustering approaches, we mention the essential components for good clustering. The first one is good representation or features and the second is a good cost function which defines what exactly a good representation is. Having this in mind, we present the following building blocks of any successful deep learning algorithm:

• Deep representation models: These are unsupervised pretraining models that produce new deep/latent representations or embeddings, which offer clearer and more solid detail of the data [29].

• Loss functions: These are the loss functions that aid in the training of the above deep representation models and the extraction of clusters from the data [29]. We discuss these two components extensively in the next subsections.

1.4.3 Learning Deep Representations

Using deep neural networks, it is possible to learn a high-level representation of the features of the input data. These representations help the clustering algorithms. Typically, we extract them from one layer of the network, but is it also possible to extract them from a concatenation of layers.

The representations are acquired after training the network using unsupervised techniques, including denoising auto encoders.

1.4.4 Deep Clustering Loss Functions

There are various types of clustering loss functions that are widely used. In our research, we focus on the two following ones:

- **Autoencoder reconstruction loss:** As we have discussed in subsection 1.3.3, autoencoders are networks that serve the purpose of encoding the input information to a hidden layer in a latent space Z, while making sure all the most important information is retained. Either side of the latent space there is an encoder and a decoder. When the training process is over, the decoder part ceases to be used and the input information can be mapped to the latent space using the encoder part of the network. Autoencoders are useful for denoising and compressing information. The autoencoder's reconstruction loss is the distance between the input x_i to the autoencoder and the corresponding reconstruction of the input $f(x_i)$, calculated in accordance with a certain distance metric. A typical choice is the Euclidean distance:

$$L = \sum_i || x_i - f_{autoencoder}(x)||^2$$

- **Clustering loss:** The purpose of clustering algorithms, given a dataset of examples $\{x_i\}_{i=1,\dots,N}$, is to group the samples in K clusters, in such a way that samples belonging to the same cluster are similar with each other. The clustering process aims to minimise the following loss function:

$$\min_{\boldsymbol{M}\in\mathbb{R}^{M\times K},\{\boldsymbol{s}_i\in\mathbb{R}^K\}} \sum_{i=1}^{N} \|\boldsymbol{x}_i - \boldsymbol{M}\boldsymbol{s}_i\|_2^2$$
$$\text{s.t. } s_{j,i} \in \{0,1\},\ \mathbf{1}^T\boldsymbol{s}_i = 1\ \ \forall i, j,$$

  where, as source [29] mentions: " s_i is the cluster membership assignment vector of data point i which has only one nonzero element for the cluster nearest to the data point; $s_{j,i}$ is the j^{th} element of s_i and represents the membership assignment in the j^{th} cluster (1 if this cluster is the nearest cluster to the data point and 0 otherwise); and the k^{th} column of M, m_k, is the centroid of the k^{th} cluster."

1.4.5 Deep Neural Network Architecture

A commonly used technique in deep learning is training a deep network to transform the input into representations more suitable for clustering. The following types of deep networks that serve the aforementioned purpose are used in our study:

- **Multilayer Perceptron (MLP):** A multilayer perceptron is a feedforward artificial neural network, that consists of at least one hidden layer of neurons with no linear activation function.
- **Convolutional Neural Network (CNN):** A convolutional neural network is a class of artificial neural network, most commonly used to analyse images.

1.5 Methods of assessing datapoint spaces and clustering results

Generally, in the clustering problem, what we strive for is creating clusters in such a way that elements of the same cluster are very similar to one another and substantially different to elements of other clusters. A lot of metrics have been implemented over the years for the purpose of evaluating clustering results and we have used some of them in our research.

1.5.1 t-distributed stochastic neighbour embedding (t-SNE):

The t-SNE (t-distributed stochastic neighbour embedding) method is actually useful in visualising high-dimensional data (with which we are mainly concerned over the course of our research) by matching each data point to a location in a new data space (map) of 2 or 3 dimensions. It's a linear dimensionality reduction method. The way in which the data points are mapped to the lower-dimension space ensures that points that are close to each other in the initial space are also close to each other in the new space. In contrast, we aim to keep the distant elements of the initial space far apart in the 2 or 3 dimensional space. T-SNE is capable of retaining the structure of the dataset while also revealing to us some of the structures that take place in it, information that might not be so apparent in the previous form of the data [7].

When it comes to the mathematical description of the procedure, the first step is converting the Euclidean distances between data points into conditional probabilities. The similarity between two data points x_i and x_j is the conditional probability $p_{j|i}$ that x_i would pick x_j as its

neighbour, were the neighbors to be picked in accordance with the probability density under a Gaussian centered at x_i. The mathematical formula is this:

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}$$

, where σ_i is the variance of the Gaussian that is centered on datapoint x_i [7]. How we calculate σ_i is discussed later on. We can also say that:

$$\sum_j p_{j|i} = 1$$

We can also construct a formula for the similarity of y_i and y_j , which represent the corresponding coordinates of the initial points in the eventual space :

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l\neq k}(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$$

If the new points represent the similarity of the two initial points x_i and x_j , the conditional probabilities $p_{j|i}$ and $q_{j|i}$ will be equal. Taking this into consideration, the procedure then turns into a minimisation of the mismatch between $p_{j|i}$ and $q_{j|i}$ problem. The algorithm selects to minimise the Kullback-Leibler divergence, which is actually a measure of how one probability distribution P differs from a second one. This is a distance that has its roots in Information Theory:

$$\mathrm{KL}\,(P \parallel Q) = \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x\in\mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right).$$

1.5.2 Normalised Mutual Info Score (NMI)

This is a metric that assesses how well the results of our clustering algorithm match the true labels of the element in the dataset. The known labels represent a probability distribution which matches each datapoint to the value 1 for each true label and the value 0 for all the others. Let this probability distribution be U. On the other hand, the predictions represent a similar probability distribution, let's name it V. The mathematical formulation of the NMI score, which comes from source [8], is as follows:

The entropy of a probability distribution is practically its amount of uncertainty and its calculation formula is this: $H(U) = -\sum_{i=1}^{|U|} P(i)\ log\ (P(i))$, where $P(i) = \frac{|Ui|}{N}$ is the probability

that an object picked at random from U falls into class Ui. The same goes for distribution V: $H(V) = -\sum_{j=1}^{|V|} P'(j)\ log\ (P'(j))$, where $P'(j) = \frac{|Vj|}{N}$.

Taking these into account, the mutual information (MI) score between U and V is calculated by:

$$\mathrm{MI}(U,V) = \sum_{i=1}^{|U|}\sum_{j=1}^{|V|} P(i,j)\log\left(\frac{P(i,j)}{P(i)P'(j)}\right)$$

Now, onto the normalised version of this formula (NMI):

$$\mathrm{NMI}(U,V) = \frac{\mathrm{MI}(U,V)}{\mathrm{mean}(H(U),H(V))}$$

One thing to note about the mutual information and also its normalised counterpart is that it is not adjusted for chance and will invariably increase if the number of different labels in our problem increases [8].

1.5.3 Silhouette score/coefficient

The silhouette score [9] is an appropriate clustering metric, as it does not necessitate the existence of a training set in our data. The mathematical formula for this metric, which is worth further analysing, is this for a certain point x_i in our data:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max\{b(x_i),\ a(x_i)\}}.$$

In this formula, we assume that x_i is an element of cluster π_k, $a(x_i)$ is the average distance of x_i to all other elements in the same cluster [9], and

$$b(x_i) = \min\ \{d_l(x_i)\},\ \text{among all clusters } l \neq k.$$

where $d_l(xi)$ is the average distance from xi to all points in cluster π_l for $l \neq k$ (between dissimilarity). The silhouette score can take values in the space [-1,1].

The silhouette coefficient of a clustering partition is defined as the mean value of the silhouette coefficients of all datapoints in the partition.

1.5.4 Adjusted rand score (ARI)

Adjusted rand score (ARI) [14] is a method that calculates the agreement between two different clustering partitions, assuming that every data point is assigned to only one cluster in each partition.

Let's suppose we have a set of n datapoints $S = \{O_1, O_2, \dots, O_n\}$ and two different partitions of those datapoints $U = \{u_1, u_2, \dots, u_n\}$ and $V = \{v_1, v_2, \dots, v_n\}$, such that each data point in the initial set S has been assigned to a single cluster of each partition. Let's also assume that U is our external criterion and V is the clustering result. In mathematical terms:

α: The number of pairs of datapoints that are placed in the same class in U and in the same cluster in V.

b: The number of pairs of datapoints in the same class in U but not in the same cluster in V.

c: The number of pairs of datapoints in the same cluster in U but not in the same class in V.

d: The number of pairs of datapoints in different classes and clusters in both partitions.

We can describe the counts a and d as agreements between the two clustering results, while the quantities b and c can be referred to as disagreements. The Rand index is the fraction $\frac{a+d}{a+b+c+d}$. It results in a value between 0 and 1.

The issue with the Rand index is that it doesn't take a constant value for two random partitions, which is why the Adjusted Rand Index was introduced. The adjusted Rand index that was proposed by [Hubert and Arabie, 1985] and assumes the generalised hypergeometric distribution as the model of randomness, i.e., the partitions are picked at random such that the number of objects in the classes and clusters are fixed.

In order to define the Adjusted Rand Index, we define $n_{i.}$ as the number of datapoints in class u_i and $n_{.j}$ as the count of datapoints in cluster v_j, while n_{ij} is defined as the number of items in both. Generally, the formula of an index that has a constant expected value is $\frac{index - expected\ index}{maximum\ index - expected\ index}$. By using the aforementioned generalised hypergeometric distribution, we can prove the following mathematical formula:

$$E\left[\sum_{i,j}\binom{n_{ij}}{2}\right] = \left[\sum_{i}\binom{n_{i.}}{2}\sum_{j}\binom{n_{.j}}{2}\right] / \binom{n}{2}$$

Using some simple algebra, the Adjusted Rand Index can be expressed as the following formula:

$$\frac{\sum_{i,j}\binom{n_{ij}}{2} - \left[\sum_i \binom{n_{i.}}{2}\sum_j \binom{n_{.j}}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{n_{i.}}{2} + \sum_j \binom{n_{.j}}{2}\right] - \left[\sum_i \binom{n_{i.}}{2}\sum_j \binom{n_{.j}}{2}\right] / \binom{n}{2}}$$

1.5.5 Accuracy score

We can evaluate a clustering/classification result by recording which datapoints where correctly/incorrectly classified as members of a class. We can define the following terminology for such purposes:

True positives (tp): The examples that belong in the positive class and where classified as such.

True negatives (tn): The examples that belong in the negative class and where classified as such.

False positives (fp): The datapoints which are members of the negative class but where classified as being positive.

False negatives (fn): The datapoints that are members of the positive class but where classified in the negative class.

Based on the above, we can define the accuracy score [15], which is the most common measure used to assess the results of machine learning algorithms:

$$accuracy = \frac{tp + tn}{tp + fp + fn + tn}$$

The accuracy score does not take into account the count of correct labels of different classes.

1.6 Data scalers

Scaling is an important step in the pre-processing of our data, before we use the data in any sort of machine learning model. Scaling helps the model understand the data better and learn it easier. Next, we present some of the most common data scalers, which we have also taken advantage of during our research:

1) **Robust scaler:** This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile). In simpler terms, when using the IQR, we remove the median from each datapoint and then divide by the difference of the median of the second half of our dataset from the median of the first half of the dataset. This scaler proves to be especially useful when the initial data have outliers, that is, data points which differ massively from the average datapoint.
2) **Standard scaler:** Useful scaler when our data are distributed in a similar way with the normal distribution. This method scales the data in such a way that they eventually have a mean value of 0 and a standard deviation of 1, which are properties of the normal distribution. This scaler does not perform well on datasets that have many outliers. In essence, it removes the mean from each datapoint and divides the result by the standard deviation.
3) **MinMax scaler:** Scales the features in such a way that they all receive values between 0 and 1. This scaler potentially works well for data that don't follow a distribution similar to the normal one, but has bad performance when it comes to data with outliers. For each feature, we simply subtract the minimum value of the distribution from it and then divide the result with the difference of the maximum from the minimum value of the distribution.

1.7 Model parameters and optimisers

1.7.1 Adam optimiser

Adam optimiser serves as an improvement on Stochastic Gradient Descent. It is a frequently used optimisation method in a lot of machine learning and artificial intelligence problems. Adam was first introduced in 2014, at a deep learning conference called ICLR 2015. Nowadays, people use it as an alternative to SGD. The method's title comes from adaptive moment estimation and is not an acronym. The name was inspired by the fact that it uses estimations of the first and second moments of the gradient, in order to adapt the learning rate for every weight of the neural network. Before the appearance of the Adam optimiser, there were other techniques used, for example AdaGrad and RMSP. These methods often offer good performance, but sometimes they generalise performance, with it being worse than that of the SGD. Adam, on the other hand, doesn't have this drawback. In addition, this handy optimiser doesn't require much parameterisation effort. This doesn't, however, mean that Adam is a panacea. It has been noted that there are datasets and occurrences in which it

doesn't converge in the optimal solution, whereas SGD does. The details of the algorithm's implementation are as follows:

Momentum: This is actually an acceleration of SGD and takes advantage of the 'exponentially weighted average'. The weights are updated in the following way:

$w(t+1) = w(t) – a * m(t)$

where $m(t) = \beta\, m(t-1) + (1-\beta)\frac{\partial L}{\partial W(t)}$

m_t = aggregate of gradients at time t [current] (initially, $m_t = 0$)
m_{t-1} = aggregate of gradients at time t-1 [previous]
W_t = weights at time t
W_{t+1} = weights at time t+1
α_t = learning rate at time t
∂L = derivative of Loss Function
∂W_t = derivative of weights at time t
β = Moving average parameter (constant, 0.9)

Root Mean Square Propagation (RMSP):

RMSP is an adaptive optimization algorithm. It faces the problems of momentum. In RMSP we take the 'exponential average' of squared gradients.

$$w(t+1) = w(t) - \frac{a(t)}{\sqrt{v(t)}+\varepsilon}\frac{\partial L}{\partial W(t)}$$

$$v(t) = \beta\, v(t-1) + (1-\beta)\left(\frac{\partial L}{\partial W(t)}\right)^2$$

W_t = weights at time t
W_{t+1} = weights at time t+1
α_t = learning rate at time t
∂L = derivative of Loss Function
∂W_t = derivative of weights at time t
V_t = sum of square of past gradients. [i.e sum($\partial L/\partial W t-1$)] (initially, $V_t = 0$)
β = Moving average parameter (constant, 0.9)
ϵ = A small positive constant (10^{-8})

Adam optimizer algorithm:

The eventual algorithm takes advantage of the above two equations:

$$m_t = \beta_1\, m_t + (1-\beta_1)\frac{\partial L}{\partial W(t)}$$

$v_t = \beta_2 v_t + (1- \beta_2) \frac{\partial L}{\partial W(t)}^2$

Both m_t and v_t are initialised to 0. Also, both are more biased towards 0 as β_1 and β_2 are equal to 1. The method solves this issue with the help of computing bias-corrected m'_t and v'_t , in this way:

$m'_t = m_t \div (1 - \beta_1^t)$

$v'_t = v_t \div (1 - \beta_2^t)$

By interchanging the old parameters with the new ones, we get the formula:

$$w_t = w(t-1) - \alpha * (\hat{m}_t / \sqrt{(\hat{v}_t)} + e)$$

Here are the steps of the algorithm in greater detail, as mentioned in [11]:

while *w(t) not converged* **do**

$t = t + 1.$

$m_t = \beta_1 * m_t + (1 - \beta_1) * (\delta L / \delta w_t)$

$v_t = \beta_2 * v_t + (1 - \beta_2) * (\delta L / \delta w_t)^2$

$\hat{m}_t = m_t \div (1 - \beta_1^t)$

$\hat{v}_t = v_t \div (1 - \beta_2^t)$

$w_t = w(t-1) - \alpha * (\hat{m}_t / \sqrt{(\hat{v}_t)} + e)$

end

return *w(t)*

1.7.2 Cross entropy loss

Cross entropy loss is a metric that comes from Information Theory and is based on entropy and generally calculates the difference between two probability distributions.

Formula: Let P be the the target distribution and Q the estimation of the target distribution. If we know the probabilities of every point in the distributions, then H(P,Q) = - $\sum$ P(x) log(Q(x)).

This calculation is used in discreet probability distributions. The returned value of the calculation is measured in bits.

In the training process of our neural network, we use the known labels, which can be represented with a discreet probability distribution which takes the value of 1 for each label-

target and 0 for all other labels. Our neural network calculates the probability for the input vector to be categorized as a certain label and does that for all labels, giving us the estimation probability distribution. Next, we calculate the cross entropy loss metric between the target-probability distribution and the estimation probability distribution. This gives us a value which reveals how close the estimation is to reality. In the process of the neural network evaluation, we calculate the average of the cross entropy losses of all training examples. The calculation obviously returns zero when the network perfectly predicts the label of each example, equating the estimation probability distribution to the target probability distribution.

1.7.3 Softmax activation function

Softmax activation [12] is used in the output layer of neural networks and serves as a normalisation of the output values, converting them from weighted sum values into probabilities values whose sum is 1. The softmax output of each output neuron's value represents the probability that the input vector of the neural network belongs to the class that corresponds to this certain neuron. The fact that the added sum of the softmax outputs amounts to 1 is a very desirable trait in classification problems with q classes and also enables the use of cross-entropy error function.

If we represent the vector of the output layer as $y = (y_1, y_2, \ldots, y_n)$, then the softmax value for each output neuron is calculated with the formula:

$$softmax(y_i) = \frac{e^{y_i}}{\sum_{k=1}^{n} e^{y_k}}$$

1.7.4 Leaky ReLU

Leaky Rectified Linear Unit [13], also named as Leaky ReLU**,** is an activation function based on the normal ReLU, but it has a small slope for the negative values instead of a flat slope. The slope coefficient is fixed before the training process takes place and remains unchanged. As our source [13] references, "This type of activation function is popular in tasks where we we may suffer from sparse gradients, for example training generative adversarial networks".

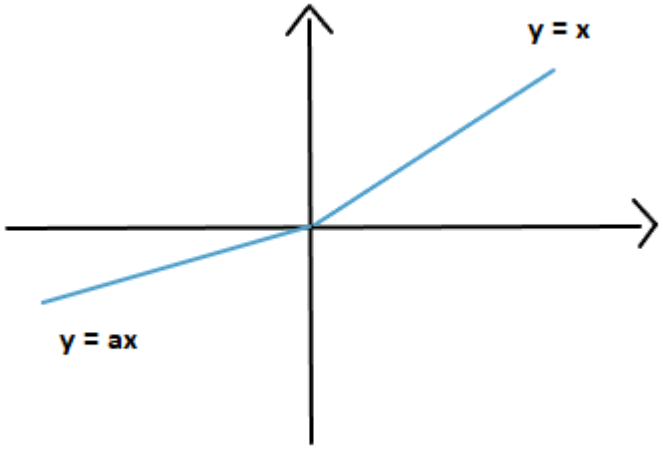


Figure 9: Leaky ReLU, a: slope coefficient

1.8 Thesis contribution

Deep Clustering is a recently developed area of clustering algorithms with great potential. The core of Deep Clustering is training Deep Neural Networks and taking advantage of the representations' capabilities. One of the most common strategies is to execute clustering algorithms on these representations. In this study, our main focus is using Deep Clustering methods and comparing their results to those of traditional clustering algorithms. We also propose new Deep Clustering methodology. The first approach takes advantage of the representations acquired from hidden layers in Multilayer Perceptrons that were trained using pseudolabels. The pseudolabels are more in number than the real number of labels. They are new labels, different from the real ones, that are acquired by executing the k-means algorithm for more than the real number of clusters. Essentially, they are the labels of the clusters that are formed in the partition of this k-means execution where K is greater than the real number of labels. We also test the capabilities of the latent space of autoencoders. An innovative idea is proposed for the training of the autoencoders, which takes advantage of clusters centroids acquired by k-means algorithm where K is greater than the real number of labels of the datasets. We also try out CNN deep networks for image datasets. Moreover, we introduce a method that estimates the number of K clusters the initial dataspace should be split into, using the k-means algorithm, for optimal clustering results in the latent space. Eventually, we compare the results of each Deep Clustering approach.

1.9 Thesis outline

The following is a brief description of the content of the next chapters:

- Chapter 2: The deep clustering methods we propose and the introduction of the criterion we use to choose the number of clusters.
- Chapter 3: The experimental study. Details about the architectures and datasets we used, boxplots of the clustering metrics for each method, evaluation of the criterion we propose and an extensive table that compares the best results of every method.
- Chapter 4: The conclusions of our study and an idea for future work on the subject of Deep Clustering.

2. The proposed method

2.1 Deep Clustering using pseudolabels

Traditional clustering algorithms have their advantages and drawbacks. As we have discussed, the k-means algorithm is widely used and works well in many cases, but struggles in others. The same can be argued about hierarchical clustering algorithms, for example agglomerative clustering. Sometimes, the problem lies in the peculiarities of the data space.

The main focus of our research is on improving the efficiency of clustering algorithms by transforming the initial dataspace to a cluster-friendly space that will enhance our performance. Many problems arise from data points that have different real labels but are positioned is such a way in the initial data space, that it is hard for the k-means algorithm to output a good quality clustering solution. For example, vectors that have a different true label might be too close to each other, as defined by distance metric such as the Euclidean distance. Moreover, datapoints from different clusters might overlap.

The idea that we propose in this thesis is splitting the data into many clusters, more than the actual number of clusters in the data, using the k-means algorithm, in order to create a clearer and more discernible data space for the traditional clustering algorithms to work on. We call this assignment clustering with pseudolabels. In order to acquire the labels of the smaller partitions, we run a k-means algorithm with its number of clusters K being a number greater than true data clusters. Then, the datapoints of the dataset are labelled by these pseudolabels, while their features remain the same.

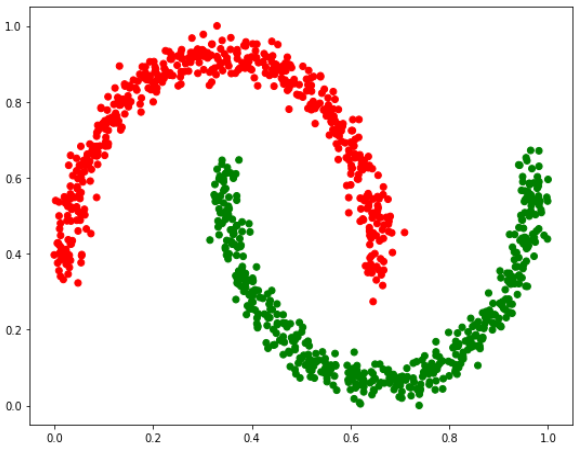


Figure 10: Two-dimensional dataset containing two clusters

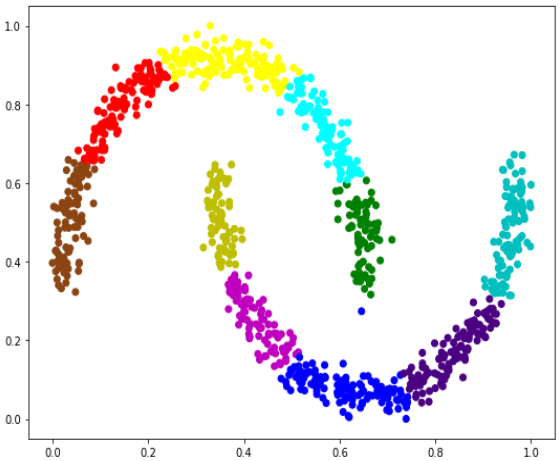


Figure 11: Clustering output of k-means with k = 10

In figures 10 and 11, we present a dataset containing two clusters. In figure 2 the datapoints are colored according to the cluster they actually belong to, whereas in figure 11 we grouped the data using the k-means algorithm with K = 10. The k-means algorithm with K = 2 fails to cluster the data correctly into 2 clusters due to the complex data structure. On the other hand, in figure 11 k-means clustered the data better but K is much greater than the actual clusters of the data. We will take advantage of this overclustering ability of k-means and will treat this partition as pseudolabels for our proposed method.

We then take advantage of the pseudolabels, by training a Multilayer Perceptron. The MLP is trained with the following dataset: $D = \{x_i, y_i\}$, where x_i is the datapoint i and y_i is the pseudolabel of x_i. We can produce the latent representations of the data from the hidden layers of the trained network. The vectors of these representations provide us with a latent space in which the points will potentially be cluster-friendly.

In the next stage of the process, we use the k-means algorithm with K being the real number of clusters in the data. The input data to the k-means algorithm are the representations of the latent space.

Figure 12: Basic algorithm steps of the proposed method

Depending on each dataset, scaling the initial data, using an option from the ones we have discussed in subsection 1.6, may prove to be helpful for the traditional clustering algorithms, like k-means. This is tested in each dataset experimentally. Moreover, we explore the possibility of scaling the data in the transformed data space, in case it helps the stats on this space as well. We mention the presence or absence of normalisation in the tables for each experiment in the experiments section, in addition to the parameters we used in our neural network models and the number of data points we took from each dataset. Multilayer Perceptrons are not the only type of neural networks we tested for our methods.

2.2 Deep Clustering using cluster centroids

A different - and more efficient, as the experiment results will show later on - approach uses the results of the k-means clustering algorithm on the initial space in a different way. The idea is to run the k-means algorithm for more than the real number of clusters and store the cluster centroids and labels that the algorithm outputs. Then, we use the centroids during the training process of an autoencoder network. Before we explain the training process of the autoencoder, we should underline that, as we have already described, autoencoders compress the input they are given through an encoder subnetwork and store the compressed information in a layer called latent space. Then this information is passed on to the decoder subnetwork which aims to reconstruct the input. In particular, the encoder of the autoencoder networks we use have 2 hidden layers after the input layer. The second hidden layer is connected to the latent space, and the latent space feeds its information to the decoder part. The decoder is the mirror network of the encoder, meaning that its first layer is the same as the second hidden layer of the encoder, its second layer is the same as the first hidden layer of the encoder and its output layer is the same as the input layer of the encoder. Normally, the training process of an autoencoder aims to minimise the reconstruction error, which is defined as follows:

$E = \sum_i ||x_i - x'_i||^2 = \sum_i ||x_i - \mathrm{D}(\mathrm{E}(x_i))||^2$, where x_i refers to the datapoint with index i and x'_i refers to the reconstruction of the datapoint x_i.

We adjust the training process, in order to minimise a slightly different measure of error, which we call centroid loss. Assuming c_i refers to the centroid of the cluster i to which datapoint x_i has been mapped, the centroid loss is defined as follows:

$$E = \sum_i ||c_i - x'_i||^2 = \sum_i ||c_i - \mathrm{D}(\mathrm{E}(x_i))||^2$$

What this adjustment achieves is compressing the initial information in the autoencoder's bottleneck in such a way that the datapoints of each cluster are pulled closer to its cluster's centroid in the latent space. We assume (and later experimentally show) that this training technique will achieve better results than the use of the traditional autoencoder training to compress the data and create a cluster-friendly latent space.

At this stage, it is vital to note that for each dataset and method we used, we run the initial k-means algorithm that gives us the labels and centroids with 100 initialisations using the k-means++ algorithm and return the solution with the lowest cluster error. This is done so as to extract the best partition possible. We then repeat the following process ten times: we train our neural network using the centroids, pass the datapoints through the network and then execute the k-means and agglomerative clustering algorithm on the latent space data. The reason why we repeat this procedure is because it has a degree of randomness. The metrics of the partitions created by the k-means and agglomerative clustering vary. This is because the initialisation of the autoencoder network's weights is random. As a result, an autoencoder trained using the same training set and clusters centroids two times will potentially give different representations in the latent space and the traditional clustering algorithms will give different results in metrics when run on different representations. The results of all the 10 experiments help us produce boxplots of the values of each metric. Each box in the boxplots corresponds to the 10 different executions of the above process for a certain number of K clusters used in the k-means execution that splits the initial dataspace. We also keep the mean value and standard deviation of the metrics in tables.

The last part of our study focuses on datasets containing images, which were handled using Deep Convolutional autoencoders.

2.3 The proposed overclustering criterion based on silhouette score

Using the traditional k-means clustering algorithm on the latent space of a trained autoencoder yields promising results, yet the number of pseudolabels is a crucial hyperparameter. It is not clear from the beginning what choice for this parameter would be ideal, therefore we need to come up with a mathematical criterion that helps us estimate the number of clusters. In this thesis, we propose an overclustering criterion based on the silhouette coefficient metric, in order to estimate the number of clusters.

When we train the autoencoder using the centroids and labels of a clustering partition with k clusters, we get a certain representation in the latent space. Each choice of number of clusters k in the initial k-means run can be associated with the representation we extract from the latent space of the autoencoder network that was trained using the centroids of the initial k-means run for k clusters.

In this paragraph we present the proposed overclustering criterion that is based on the silhouette coefficient. The first step is to extract one representation from each choice of number of clusters k we want to try, $k \in \{k_1, \dots, k_n\}$, and then test its performance in clustering. In the way we have discussed in the paragraph above, we extract one representation r_i, i = 1, …, n for each clustering solution k_i we get from a k-means run for a number of clusters same as the number in position i of the list $\{k_1, \dots, k_n\}$.

The idea is not only to test every representation r_i in clustering with the number of clusters to which it corresponds, but also in clustering with every number of k clusters available in our choices. This will logically help us choose a well-rounded contender. To do this, for every representation r_i, we run the k-means algorithm one time for each of the choices for number of clusters in $\{k_1, \dots, k_n\}$, calculate the silhouette coefficient for every partition and then store the maximum silhouette score:

$$\hat{S}(r) = argmax_k \, sil(r, k)$$

The next step is to calculate the silhouette score of the clustering partition that a k-means run for each representation provides with the real number of clusters M. We then normalise this result by dividing it with the value $\hat{S}(r_i)$ that corresponds to representation r_i. We choose the representation r* that has the highest normalised value as the best representation and the number of clusters k*, for which the k-means algorithm was run to extract this representation, as the best choice for the number of pseudo-labels. In mathematical terms:

$$\hat{S}(r, M) = \frac{sil(r, M)}{\hat{S}(r)}, M: real\ number\ of\ clusters$$

$$r^* = argmax_r\ \hat{S}(r, M)$$

Analytically, the steps of the algorithm are as follows:

Algorithm 3: Overclustering criterion based on silhouette coefficient algorithm

---

Input:

- Data vectors $\{x_n\}_{n=1}^{N}$, $x_n = (x_{n1}, x_{n2}, \ldots, x_{nd})$, where d is the dimension of the data and N is the count of datapoints in the dataset.
- List of the choices for the number of clusters for the initial k-means run $[k_1, k_2, \ldots, k_m]$.
- M: The real number of clusters of our dataset.

1: i = 1, representations = [] // initialise representations list

2: **For each** k in $[k_1, k_2, \ldots, k_m]$:

3: cluster_centroids, cluster_labels = k-means(k, x_n) // run k-means on data vectors

4: autoencoder = Autoencoder() // initialise autoencoder model

5: autoencoder.train(cluster_centroids, cluster_labels) // train autoencoder

6: autoencoder.encoder(x_n) // pass datapoints through the encoder part

7: r_i = autoencoder.latent_space // extract representation from the latent space

8: representations.append(r_i)

9: i = i + 1 // increase the representation index

10: **End for**

11: max_silhouette = [] // maximum silhouette score for representation r_i is in position i of this list

12: **For each** r in representations:

13: silhouette_scores = [] // holds the silhouette scores of the partitions produced by r

14: **For each** k in [k_1, k_2, …, k_m]: // test every representation in clustering for all options of k

15: silhouette_score = k-means(k, r) // get the silhouette score of the partition

16: silhouette_scores.append(silhouette_score)

17: **End for**

18: max_silhouette.append(max(silhouette_scores)) // keep the maximum for each r

19: **End for**

20: normalised_silhouette_scores = [] // initialise list

21: **For each** r in representations:

index = max_silhouette.index_of(r) // get the maximum silhouette of r from the list

22: silhouette_score = k_means(M, r) // run k-means on r for the real number of clusters

23: $\text{normalised\_silhouette} = \frac{silhouette\_score}{\text{max\_silhouette (index)}}$ // normalise score by dividing with max of r

24: normalised_silhouette_scores.append(normalised_silhouette)

25: **End for**

26: **Return** the k that corresponds to the representation that has the max normalised silhouette score.

3. Experimental study

3.1 Datasets description

The datasets we use to test our clustering methods are popular datasets extensively used in machine learning and similar fields research:

10x73k dataset: A single-cell RNA-sequencing dataset [19] that was generated by 10x genomics' software tools. It consists of xenograft cell line and primary tumour data that was taken from patients.

The dataset has 8 different target labels (1-8) and 720 features for each datapoint.

Pendigits dataset: The pendigits dataset [16] is a dataset that consists of hand-written digits. It was created by asking several writers to write digits using a stylus on an LCD display. The information of the digits was stored with the help of the pressure levels that the pen produced. The first few tries of each participant were not taken into consideration, without their knowledge, so as to allow them to get accustomed to the stylus and screen. If a participant was unhappy with their writing, they always had the option to erase it and try again.

All features are integers in the range 0…100, whereas the last attribute is the class label 0…9. In total, there are 16 attributes.

Gaussian rings dataset: This is a synthetic dataset made with the help of the python scikit-learn library "make_circles". This function produces random data points that belong in two circles which have the same center. By parameterizing it properly, we produce thick circles that can be categorized as rings. We create two rings of datapoints. The dataset consists of two clusters, the first is the outer ring and the second is the inner ring.

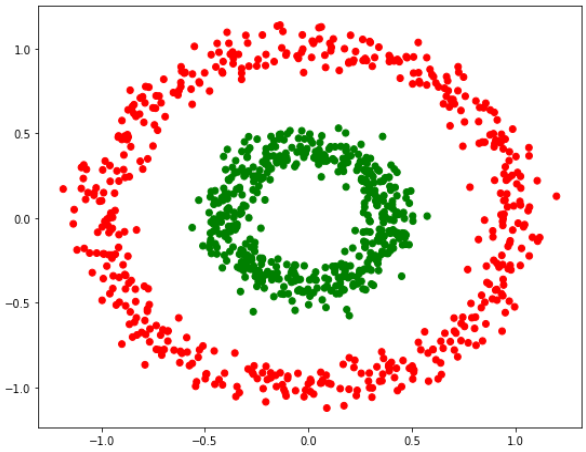


Figure 13: The gaussian rings dataset

Squeezed gaussian blobs dataset: This is a synthetic dataset made with the help of the python scikit-learn library “make_blobs”. This function helps us produce random data points that belong in a blob. By parameterizing it properly, giving it a bigger standard deviation along the y-axis than the x-axis, we produce blobs that are elongated on the y-axis, giving them a squeezed look, hence the name of the dataset. We create two blobs of datapoints. The dataset consists of two clusters, the first is the left blob and the second is the right blob.

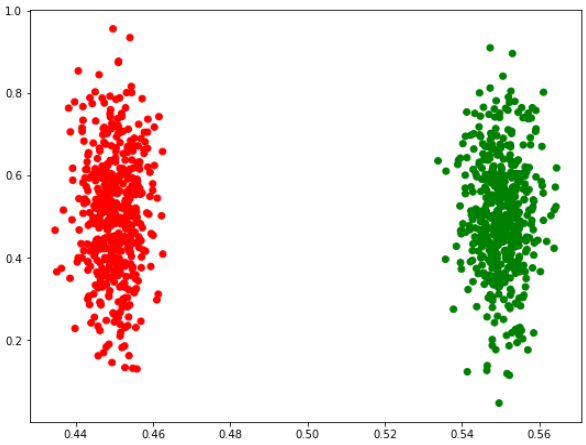


Figure 14: The squeezed gaussian blobs dataset

Moons dataset: This is a synthetic dataset made with the help of the python scikit-learn library “make_moons”. This function helps us produce random data points that belong in a moon shape. We create two moons of datapoints, with one edge of each moon getting close to the center of the other moon. Each moon is mapped to a different label. We also add some noise.

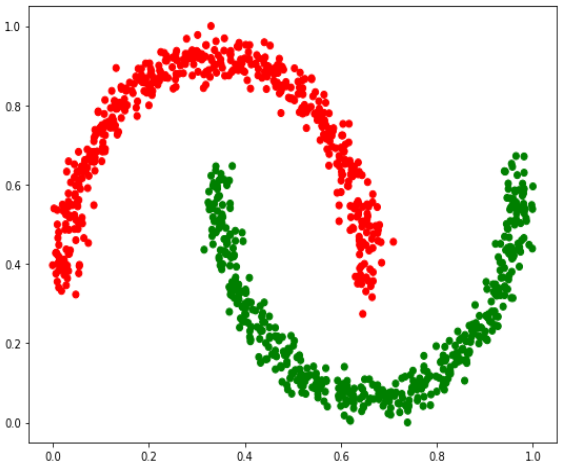


Figure 15: The moons dataset

Australian dataset: The Australian Credit Approval dataset [17] is a dataset that refers to credit card information. As the source suggests: “all attribute names and values have been changed to meaningless symbols to protect confidentiality of the data”.

Source [17] also goes on to mention this about the dataset’s attributes:

“There are 6 numerical and 8 categorical attributes. The labels have been changed for the convenience of the statistical algorithms. For example, attribute 4 originally had 3 labels p, g, gg, and these have been changed to labels 1,2,3.

A1: 0, 1 CATEGORICAL (formerly: a, b)

A2: continuous.

A3: continuous.

A4: 1, 2, 3 CATEGORICAL (formerly: p, g, gg)

A5: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 CATEGORICAL (formerly: ff, d, i, k, j, aa, m, c, w, e, q, r, cc, x)

A6: 1, 2, 3, 4, 5, 6, 7, 8, 9 CATEGORICAL (formerly: ff, dd, j, bb, v, n, o, h, z)

A7: continuous.

A8: 1, 0 CATEGORICAL (formerly: t, f)

A9: 1, 0 CATEGORICAL (formerly: t, f)

A10: continuous.

A11: 1, 0 CATEGORICAL (formerly t, f)

A12: 1, 2, 3 CATEGORICAL (formerly: s, g, p)

A13: continuous.

A14: continuous.

A15: 1,2 class attribute (formerly: +,-)”

Iris dataset: The iris dataset [18] is a very popular dataset in machine learning applications. The dataset is a small one and contains 3 classes of 50 instances each. The classes have to do with types of iris plant. The data have 4 features (real) and 1 class. It is a very simple data set.

Source [18] mentions this about the features:

“1. sepal length in cm

2. sepal width in cm

3. petal length in cm

4. petal width in cm

5. class:

-- Iris Setosa

-- Iris Versicolour

-- Iris Virginica”

MNIST dataset: This is a large dataset of handwritten digits [20] that is widely used in image processing and machine learning applications. The images have an 28x28 dimension and are grayscale. The database contains 60,000 training images and 10,000 testing images.

Figure 16: MNIST dataset sample, taken from the Wikipedia article in [20]

Fashion-MNIST dataset: This is a large dataset of images of ten different clothing categories [21] that is widely used in image processing and machine learning applications. The images have an 28x28 dimension and are grayscale. The database contains 60,000 training images and 10,000 testing images. These specifications are the same as the MNIST dataset, as the fashion-MNIST dataset was created as its replacement for benchmarking machine learning algorithms.

Figure 17: Fashion-MNIST dataset examples

3.2 Boxplots and overclustering criterion for autoencoder-trained-with-centroids method

In this section, we will cite every dataset we test the autoencoder with centroids method on, provide the details of the architecture used, the number of datapoints selected and their scaling (or lack thereof), the boxplots and the overclustering criterion graph and in the end comment on the data.

10x73k dataset:

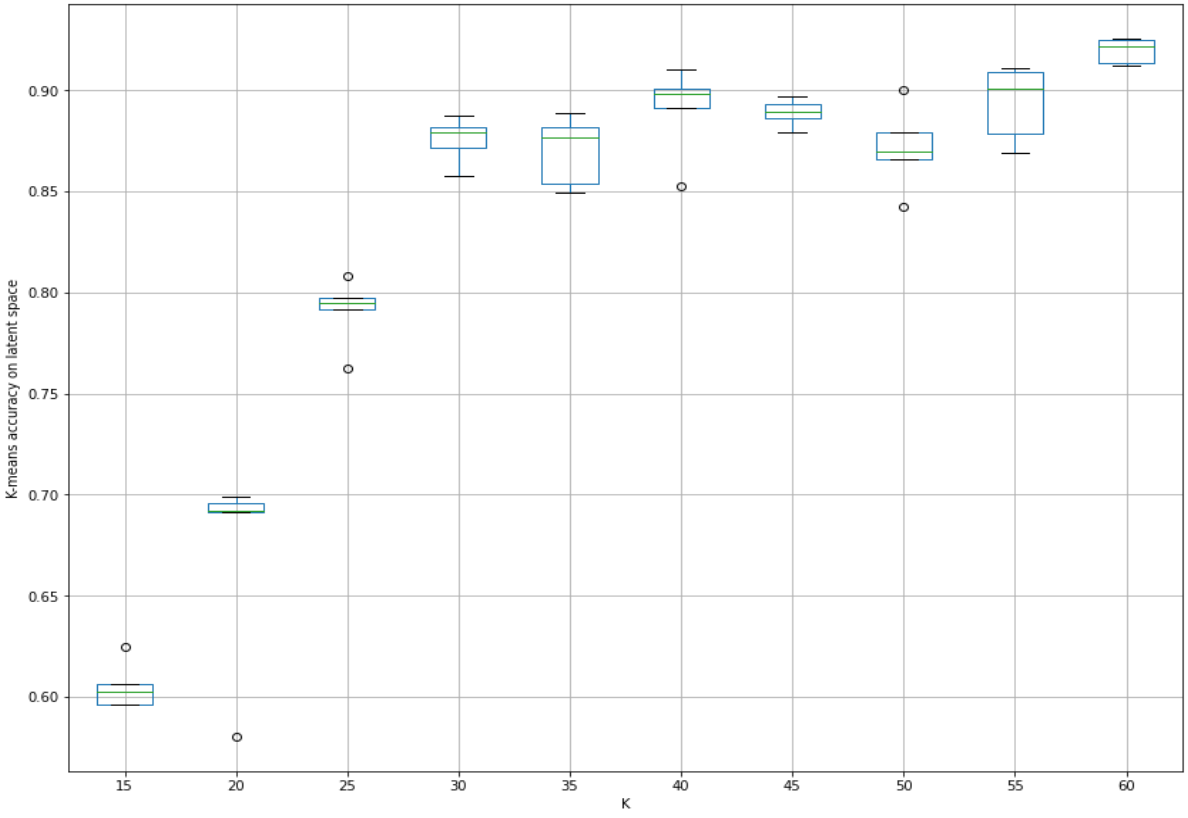
Autoencoder architecture and data processing details:

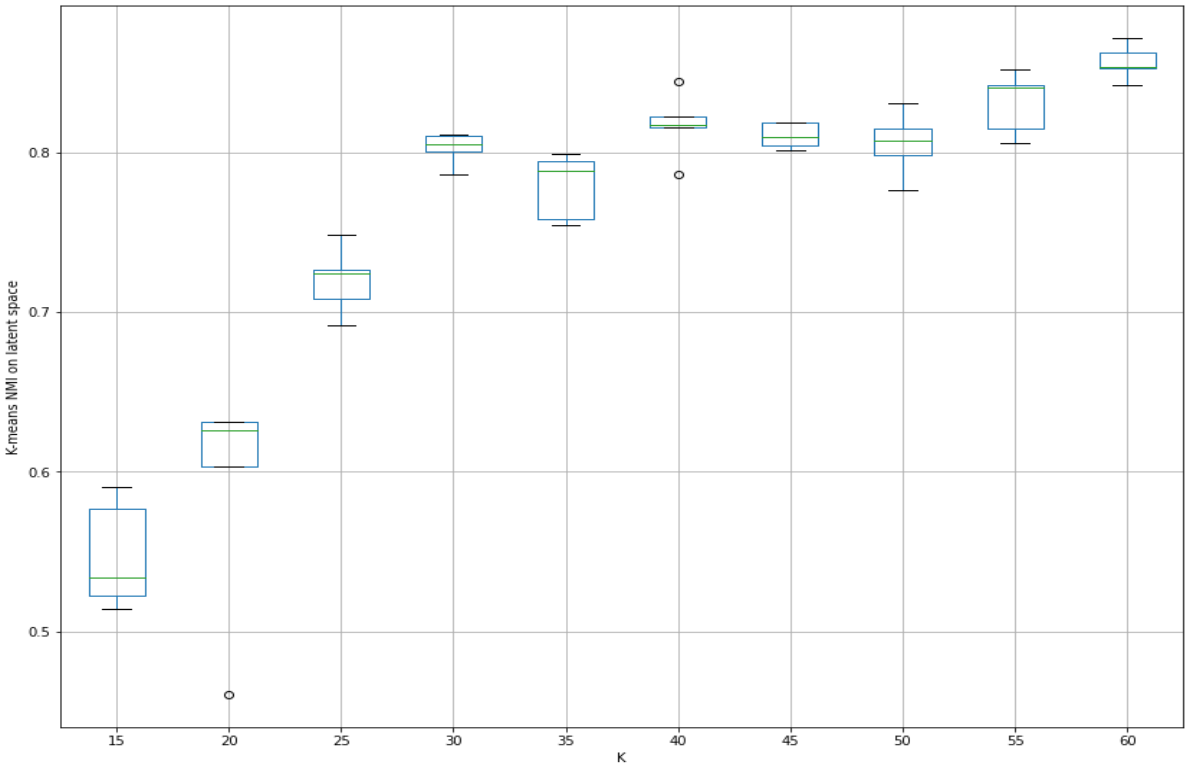
Hidden layer 1 neurons	800
Hidden layer 2 neurons	300

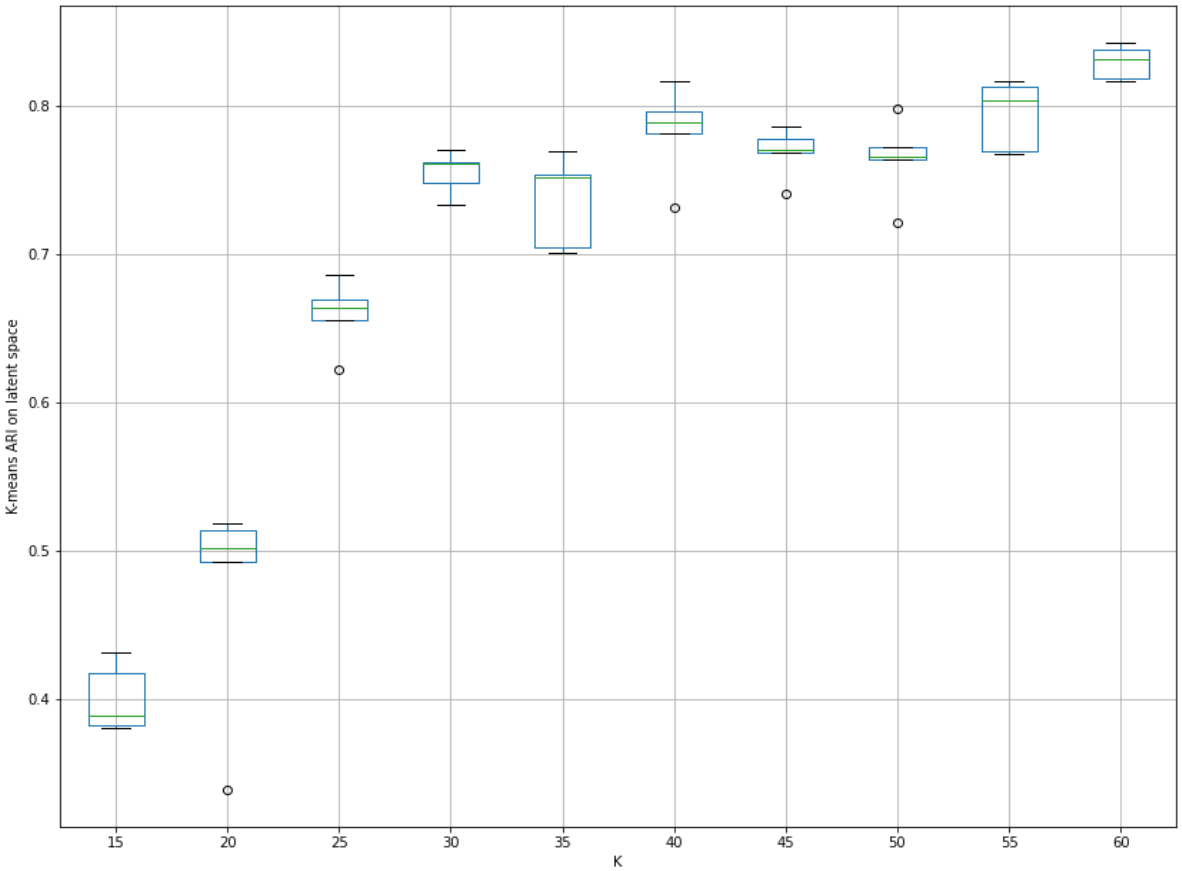
Latent dimension	8
Connections	Fully Connected (in all layers)
Activation function	Sigmoid (in all layers)
Training epochs	50
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	64
Datapoints	10000
Scaling in the initial data space	Divide everything by the max value
Scaling in the latent space	None

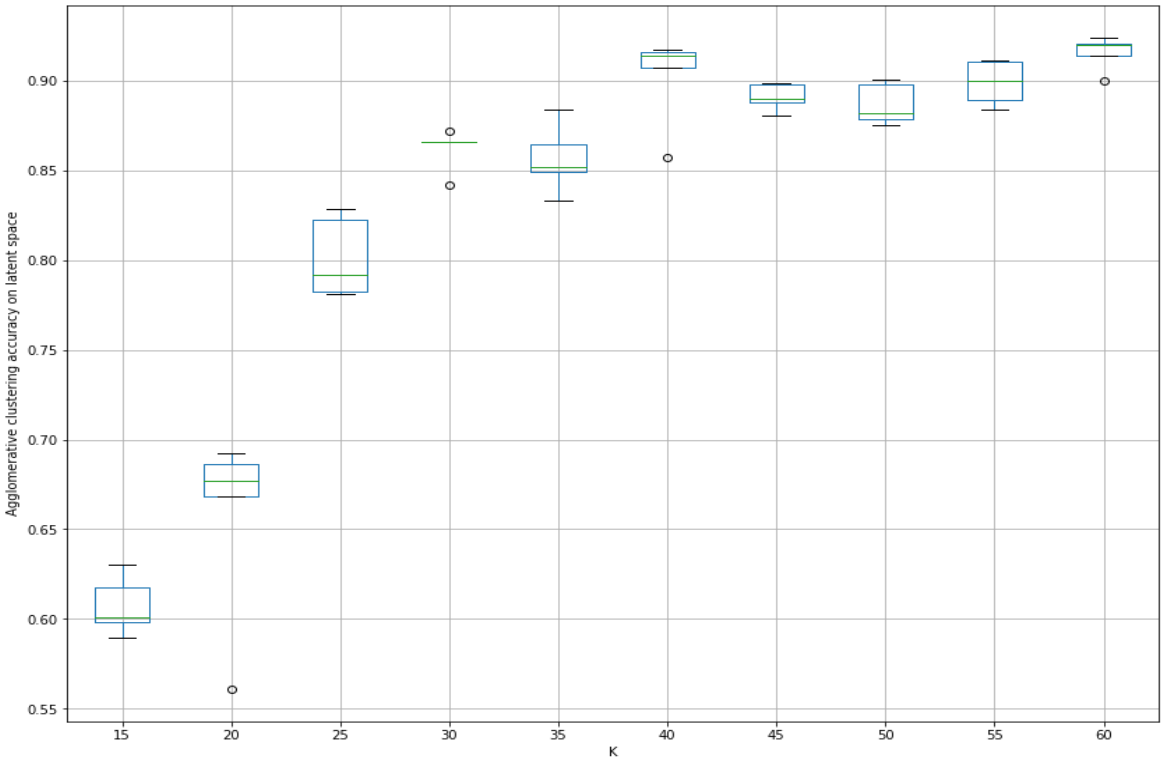
Autoencoder-trained-with-centroids method parameters for 10x_73k dataset

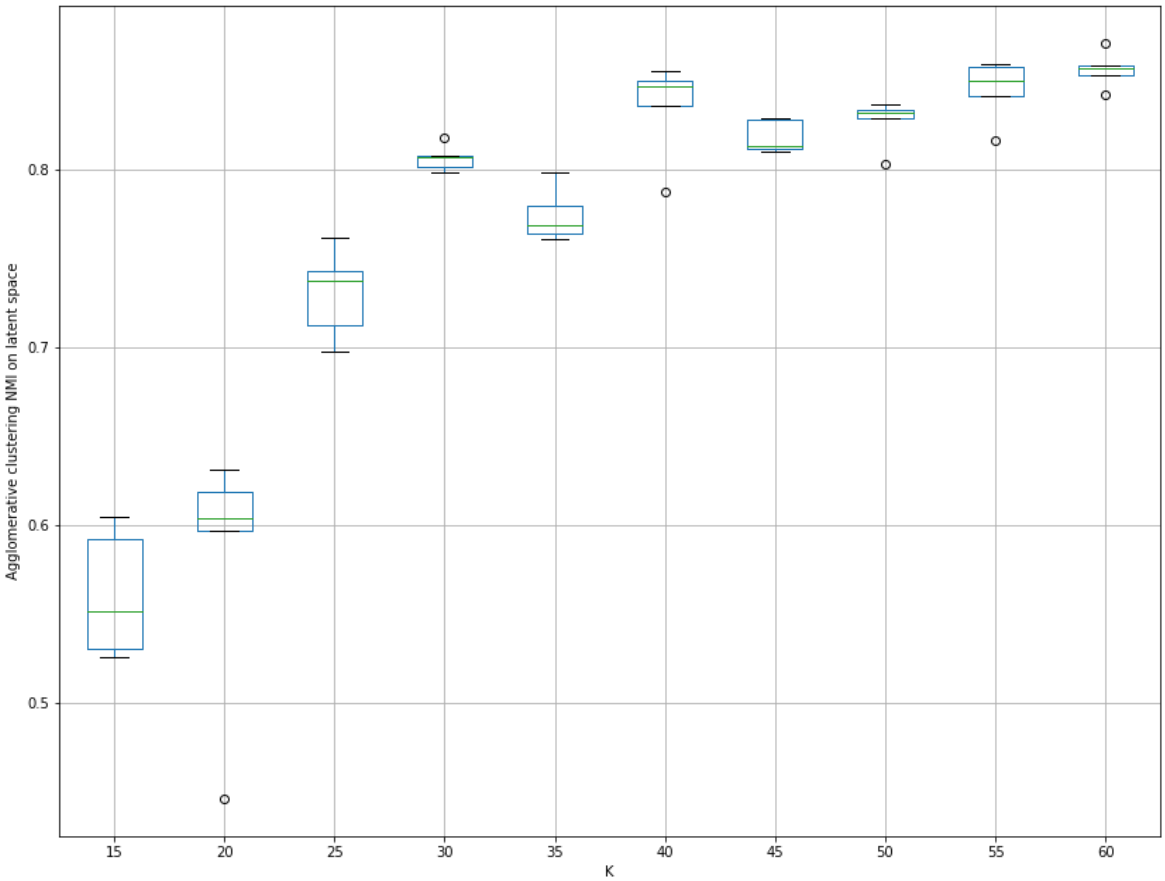
Boxplots:

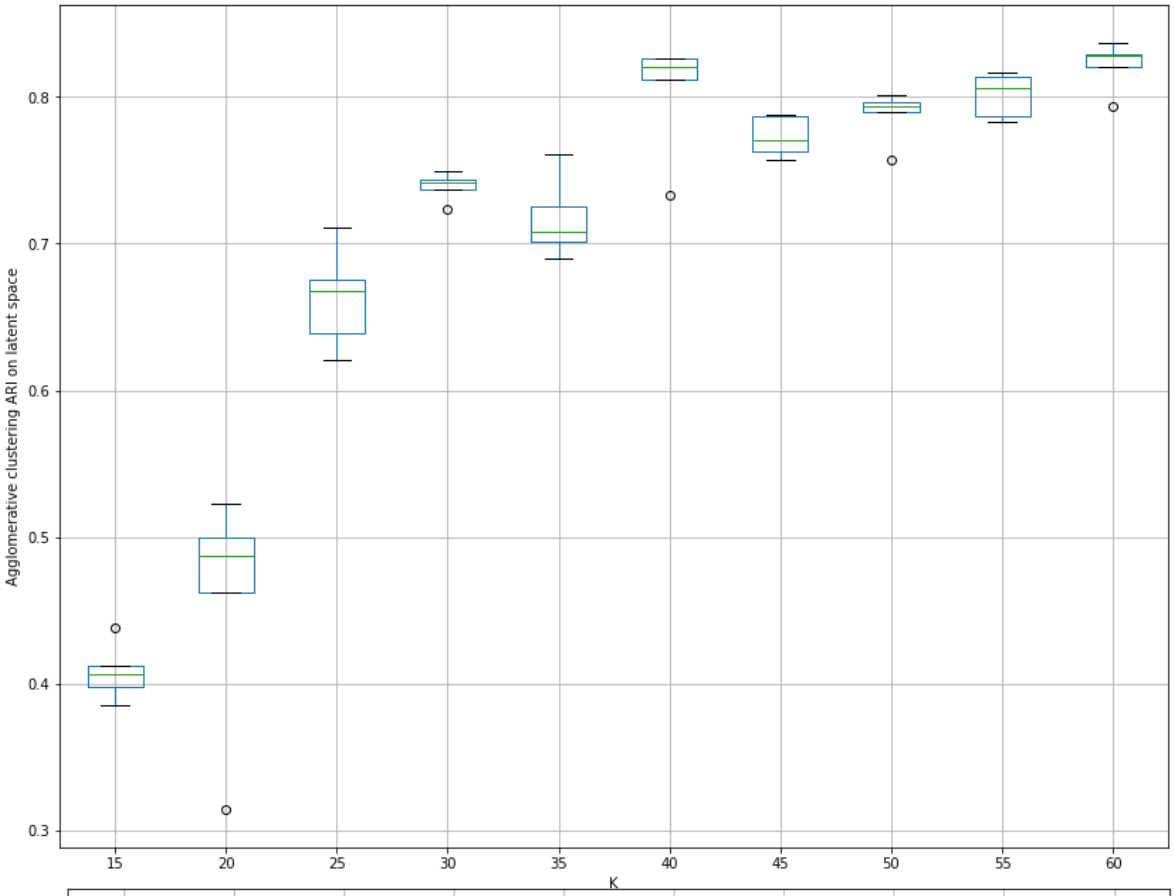


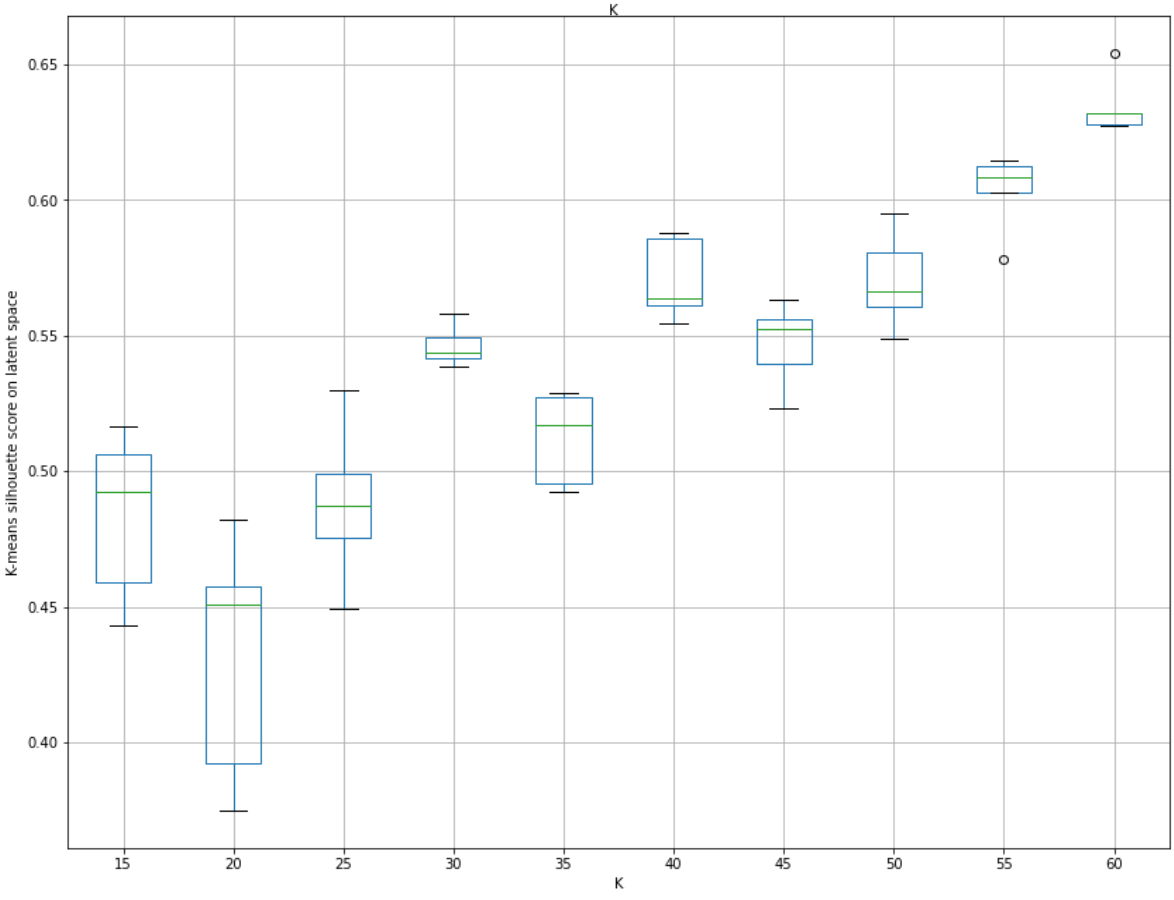

K-means NMI on latent space
0.8
0.7
0.6
0.5
15
20
25
30
35
40
45
50
55
60
K


K-means ARI on latent space
0.8
0.7
0.6
0.5
0.4
15
20
25
30
35
40
45
50
55
60
K

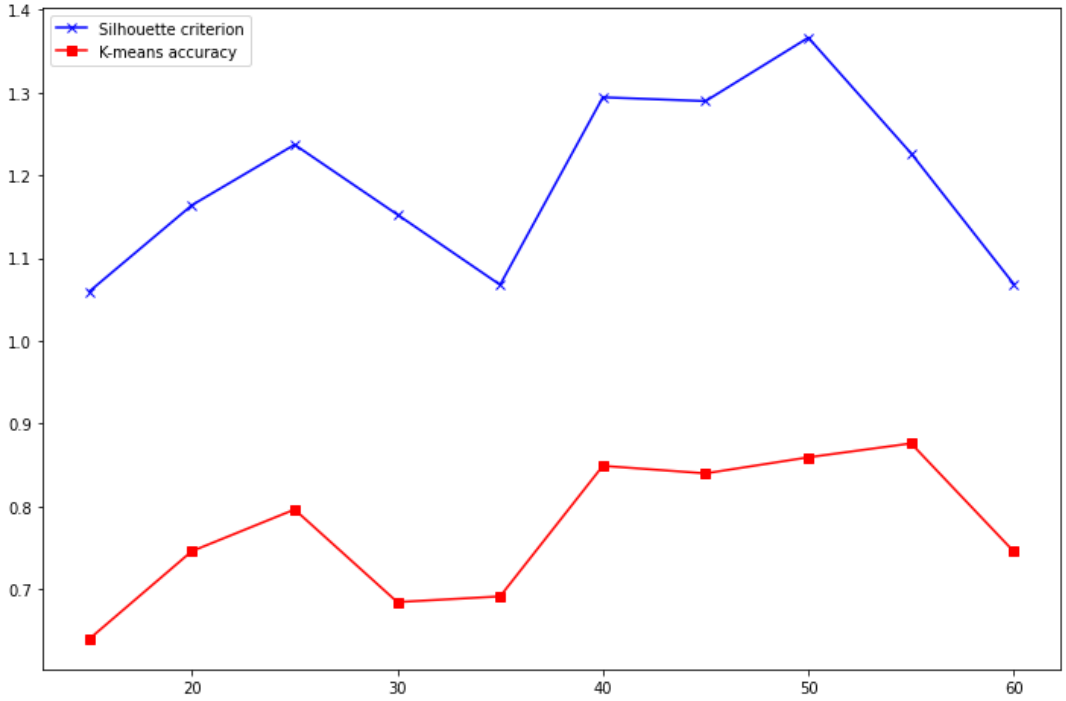

Agglomerative clustering accuracy on latent space
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering NMI on latent space
0.8
0.7
0.6
0.5
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering ARI on latent space
0.8
0.7
0.6
0.5
0.4
0.3
15
20
25
30
35
40
45
50
55
60
K


K-means silhouette score on latent space
0.65
0.60
0.55
0.50
0.45
0.40
15
20
25
30
35
40
45
50
55
60
K

10x73k overclustering criterion:



Comments:

In this dataset, we notice impressive improvements in clustering performance, compared to the traditional use of the algorithms in the initial space. The boxplots show us that k = 60 is the best-performing option for the initial number of clusters and the use of the overclustering criterion gives us a global maximum for the option k = 50, which produces close to the same results. In other words, by choosing the option k = 50 using this criterion, we get close to optimal results in our metrics.

Pendigits dataset:

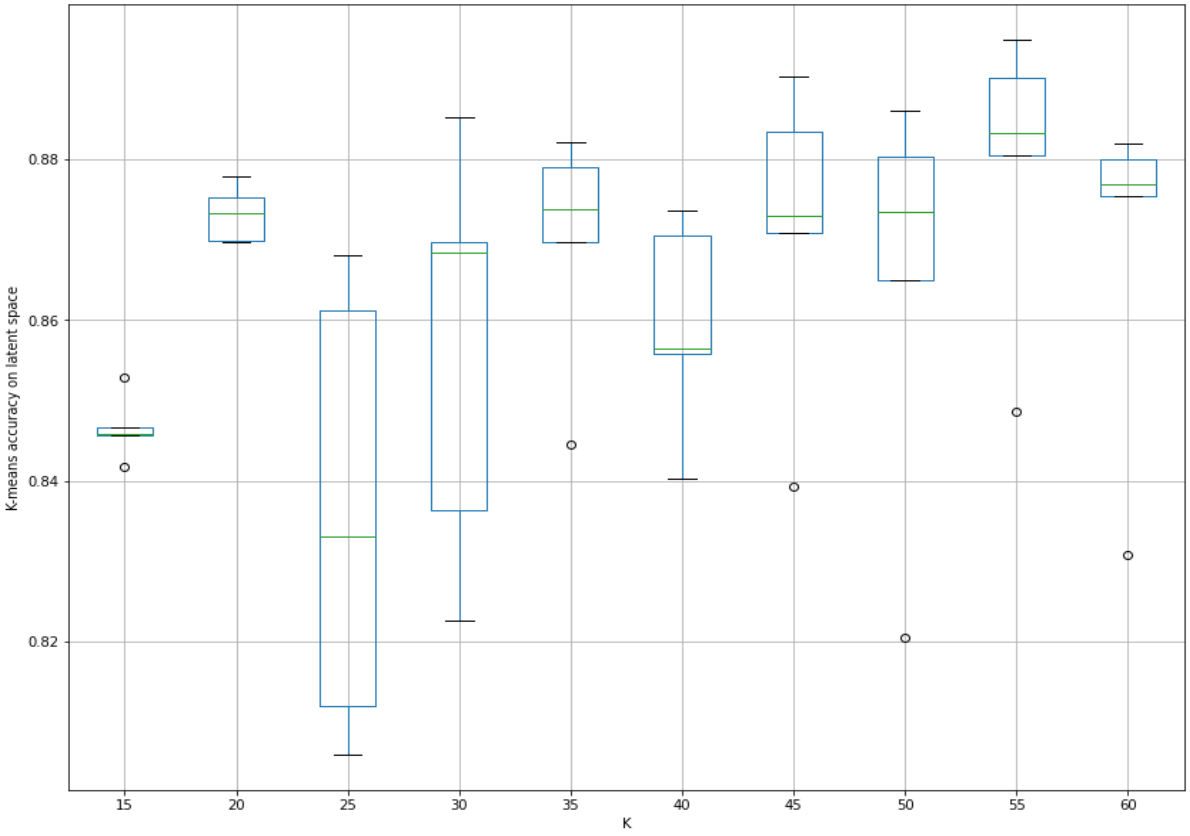
Autoencoder architecture details and data processing details:

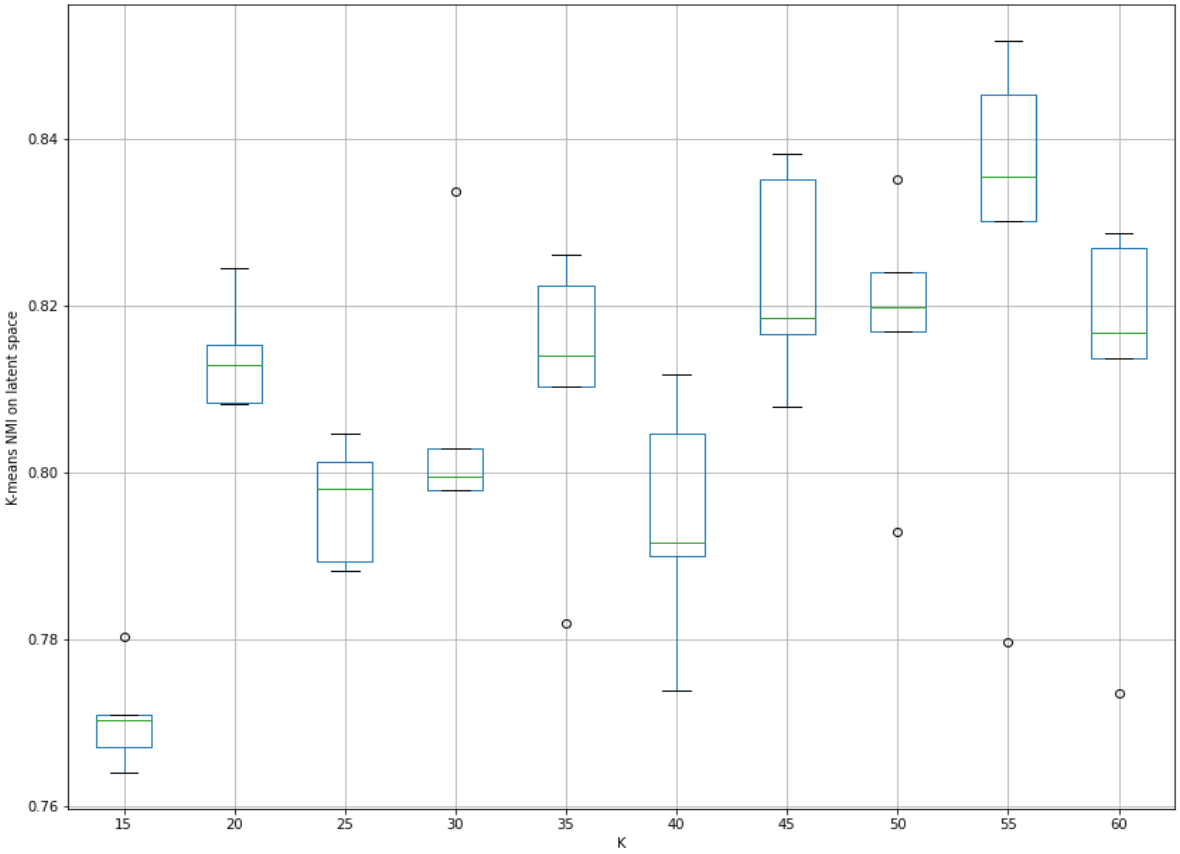
Hidden layer 1 neurons	800
Hidden layer 2 neurons	300
Latent dimension	8
Connections	Fully Connected (in all layers)
Activation function	Sigmoid (in all layers)

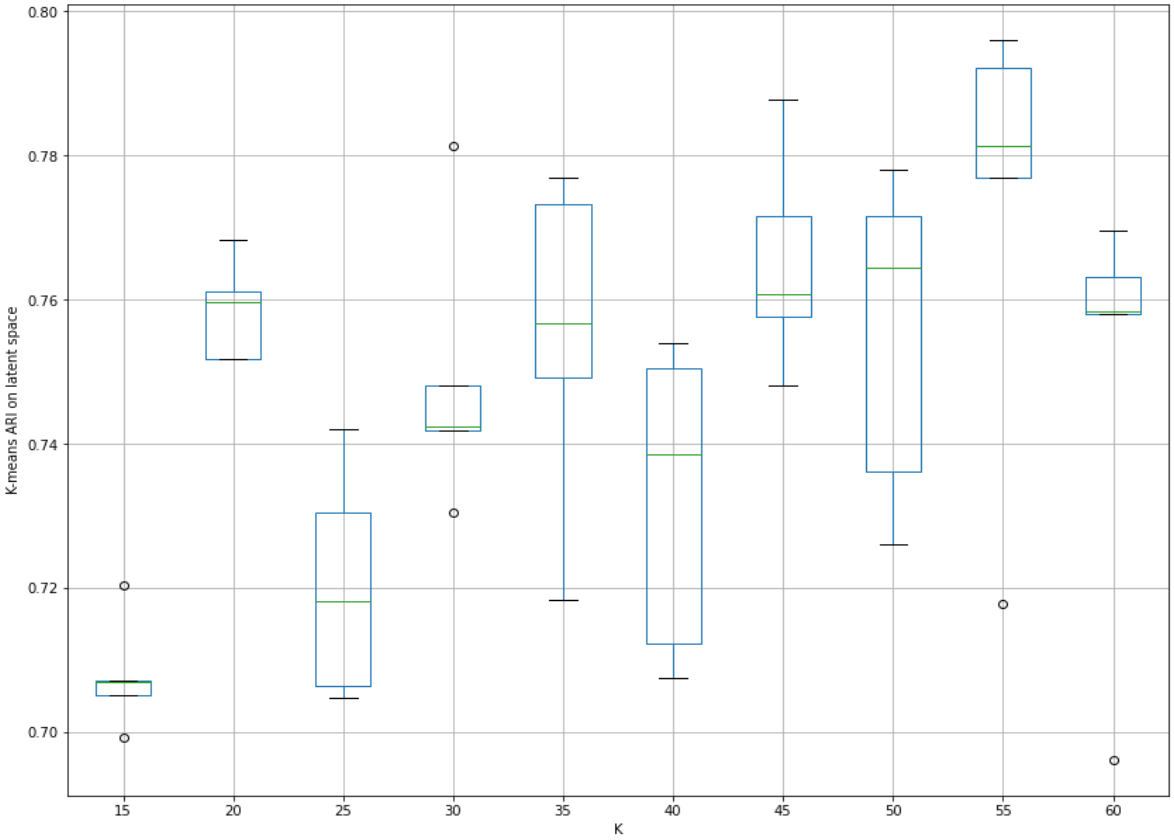
Training epochs	50
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	128
Datapoints	7494
Scaling in the initial data space	MinMax
Scaling in the latent space	None

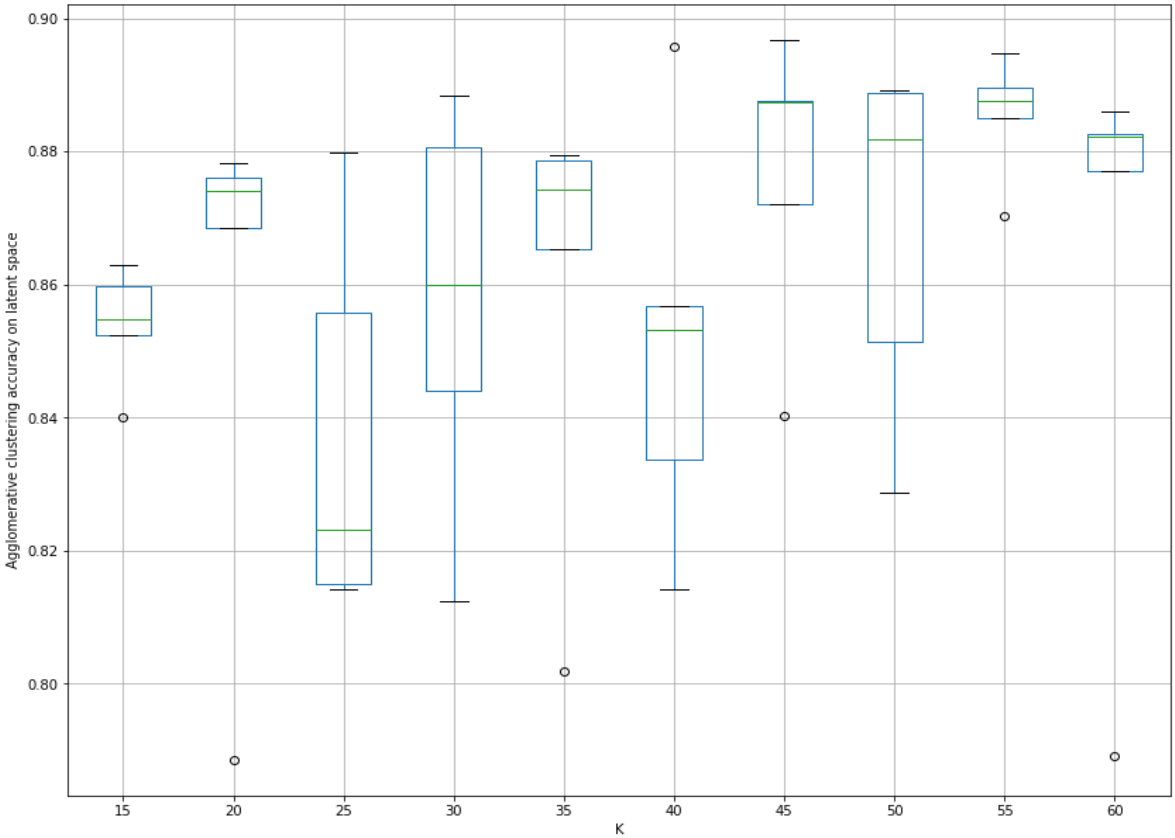
Autoencoder-trained-with-centroids method parameters for pendigits dataset

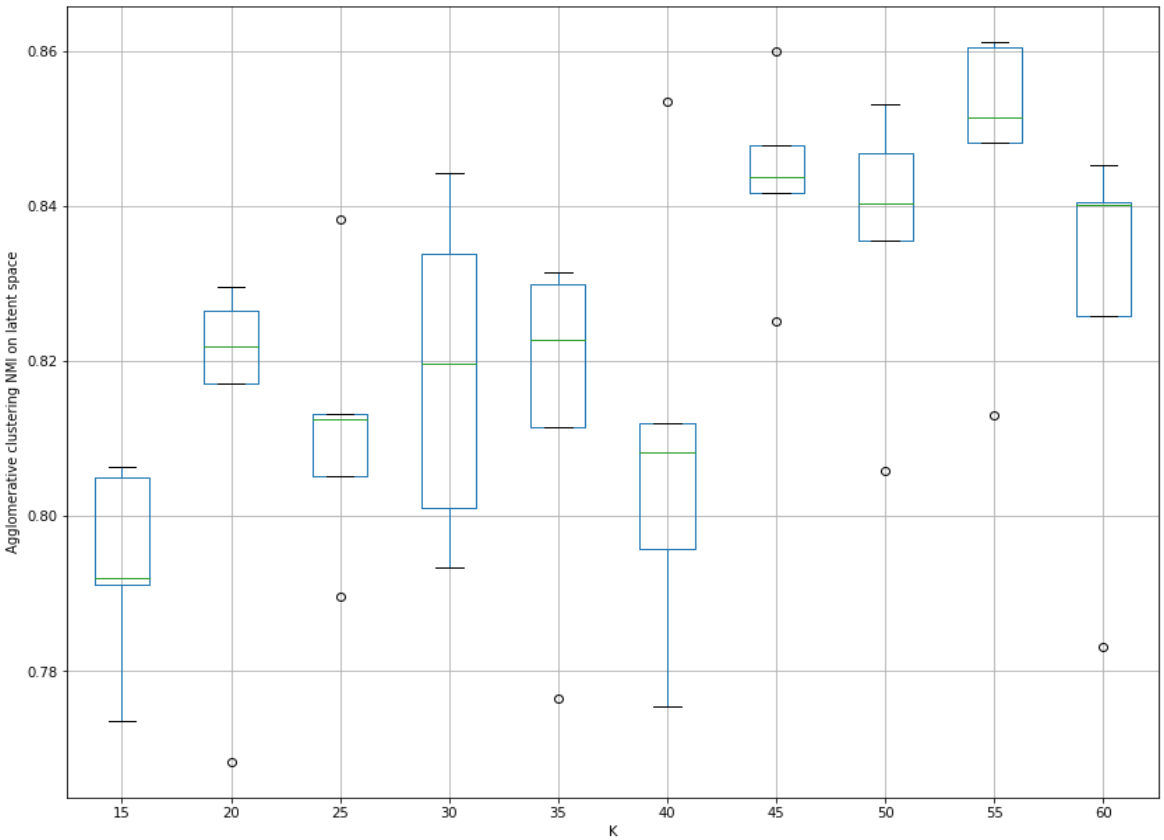
Boxplots:

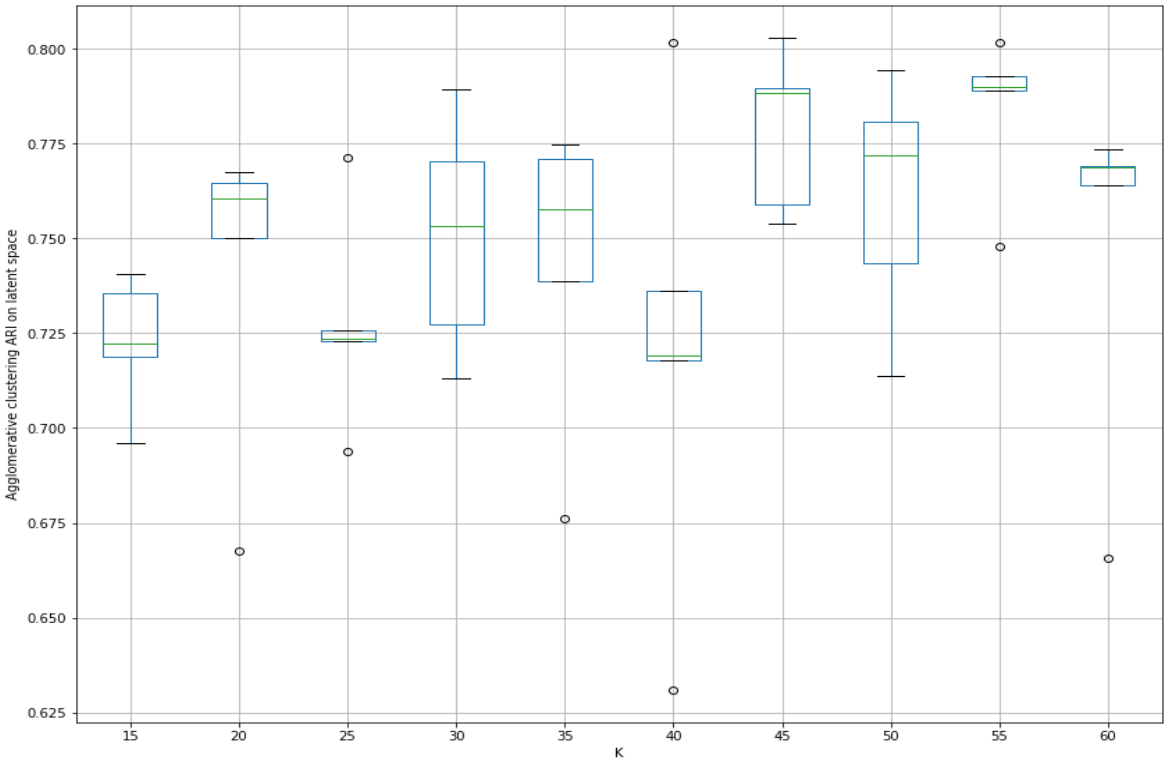


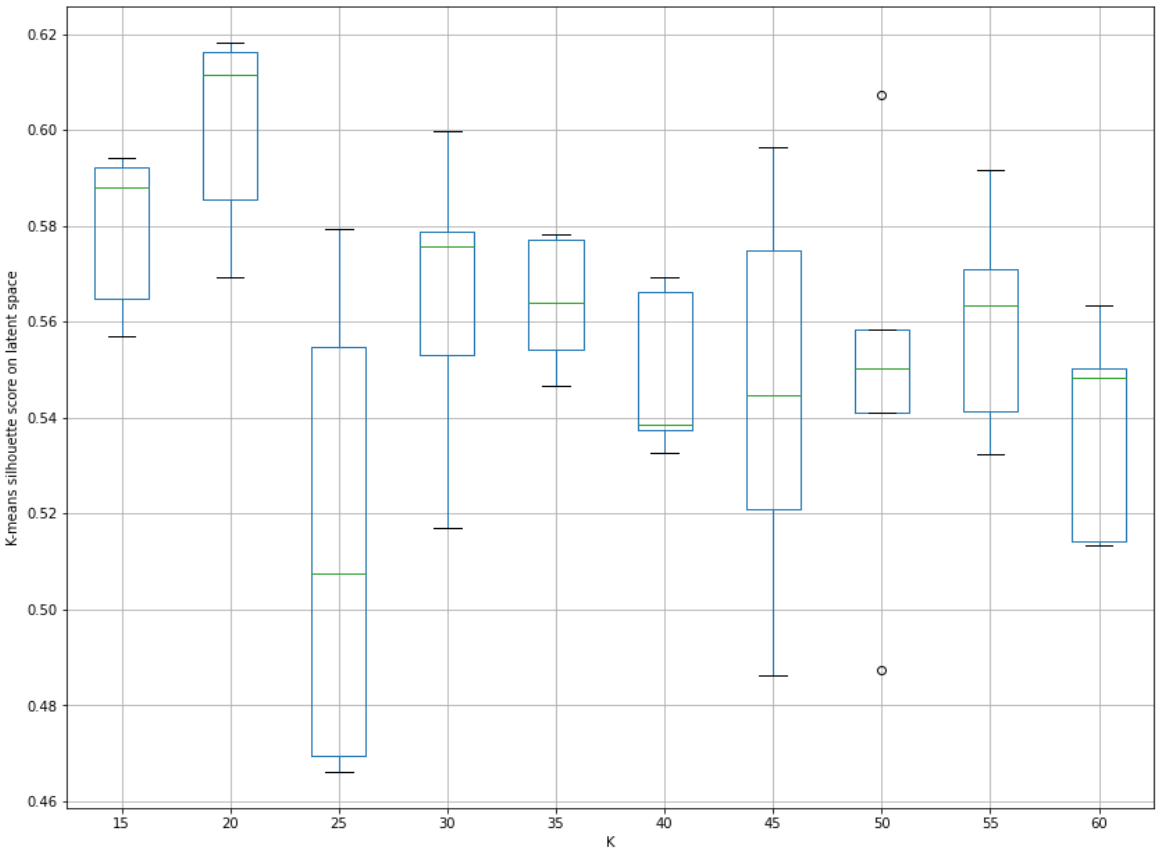

K-means NMI on latent space
0.84
0.82
0.80
0.78
0.76
15
20
25
30
35
40
45
50
55
60
K


K-means ARI on latent space
0.80
0.78
0.76
0.74
0.72
0.70
15
20
25
30
35
40
45
50
55
60
K

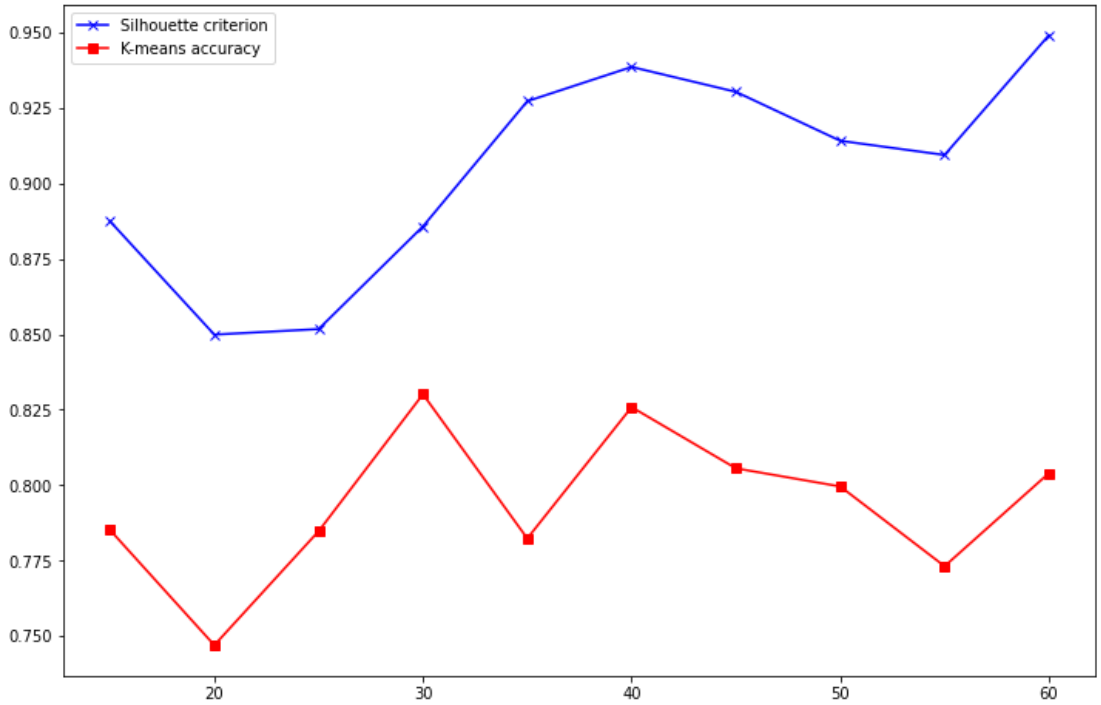

Agglomerative clustering accuracy on latent space
0.90
0.88
0.86
0.84
0.82
0.80
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering NMI on latent space
0.86
0.84
0.82
0.80
0.78
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering ARI on latent space
0.800
0.775
0.750
0.725
0.700
0.675
0.650
0.625
15
20
25
30
35
40
45
50
55
60
K


K-means silhouette score on latent space
0.62
0.60
0.58
0.56
0.54
0.52
0.50
0.48
0.46
15
20
25
30
35
40
45
50
55
60
K

Pendigits overclustering criterion:



Comments:

Taking the global maximum of the overclustering criterion into consideration, we would make the choice k = 60 for this dataset. This results in performance pretty close to the actual best performing choice of k = 55, according to the boxplots, which means the criterion is working quite well in this case. It's worth noting that k = 55 provides us with the optimal solution for all measures, apart from the k-means silhouette score one.

Gaussian rings dataset:

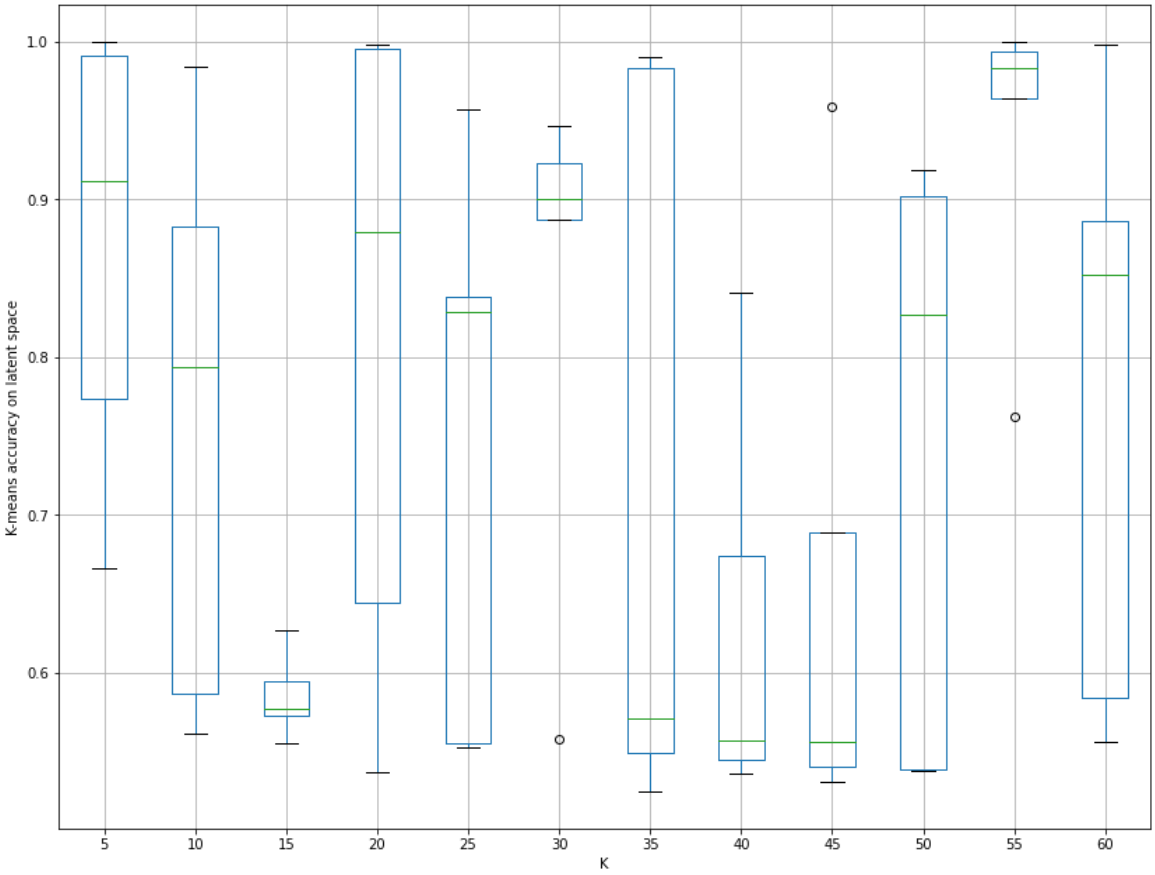
Autoencoder architecture and data processing details:

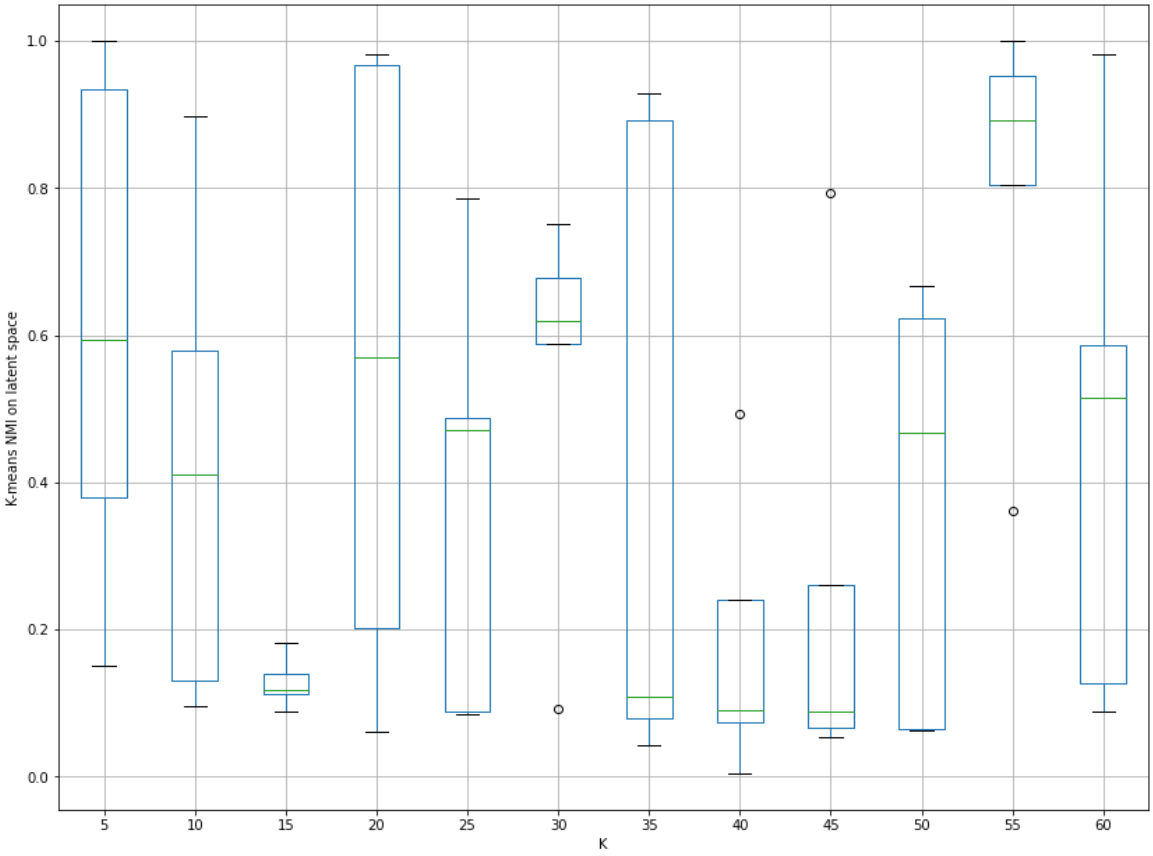
Hidden layer 1 neurons	800
Hidden layer 2 neurons	300
Latent dimension	1
Connections	Fully Connected (in all layers)
Activation function	ReLU (in all layers)
Training epochs	50
Loss function	MSE
Optimiser	Adam optimiser

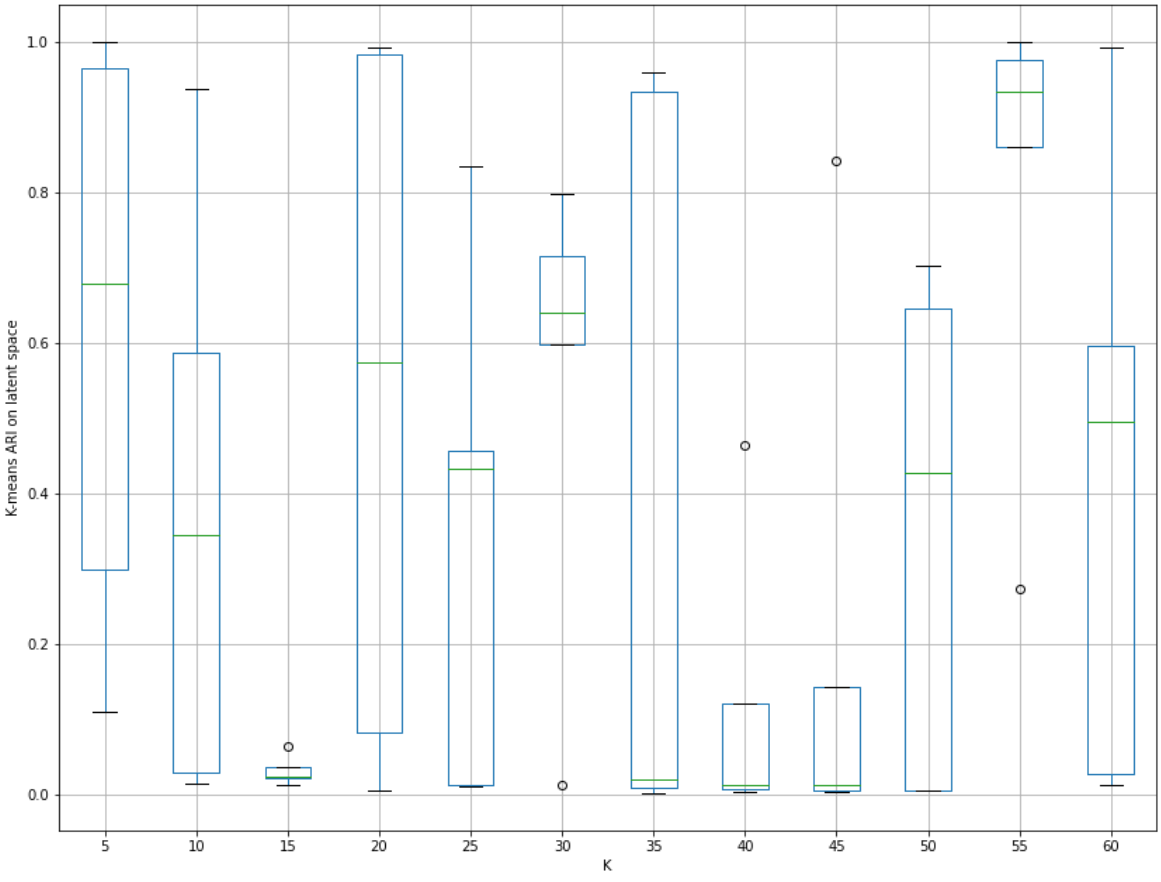
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	100
Datapoints	1000
Scaling in the initial data space	MinMax
Scaling in the latent space	None

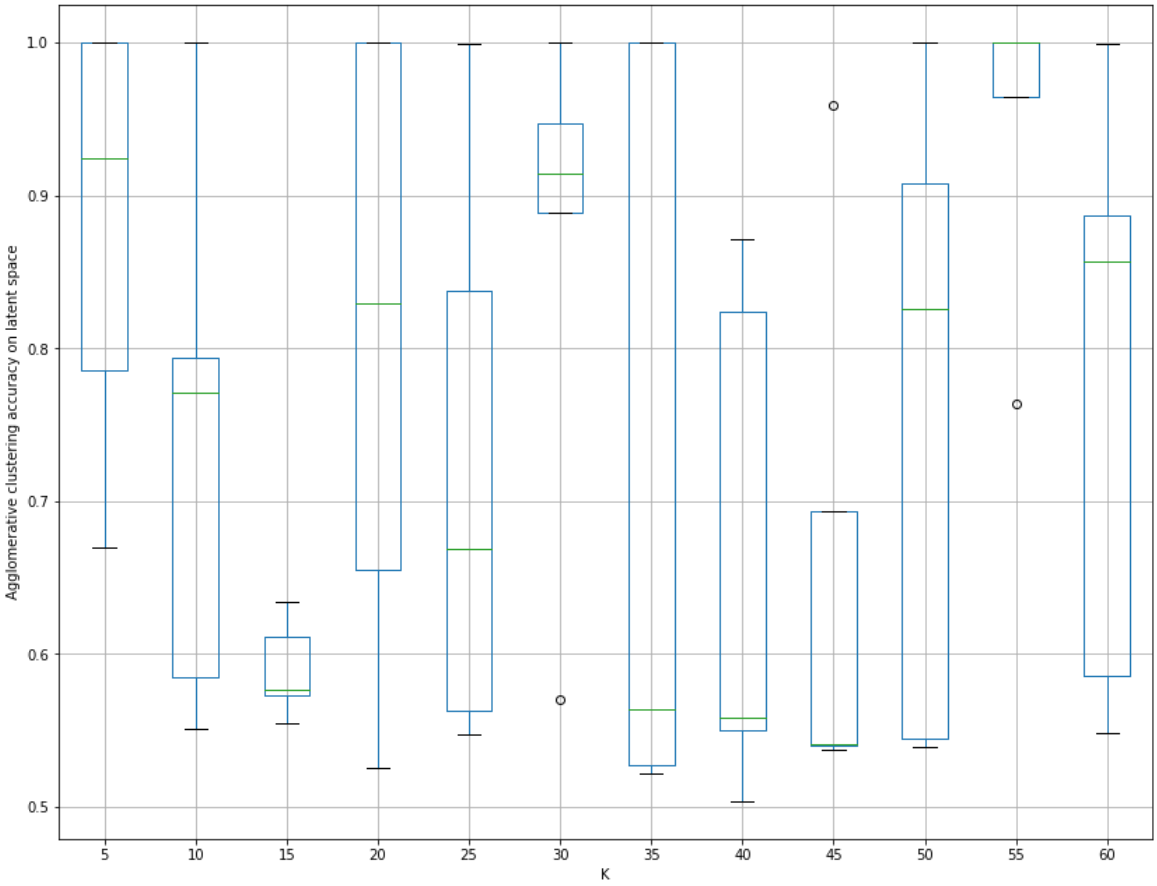
Autoencoder-trained-with-centroids method parameters for gaussian rings dataset

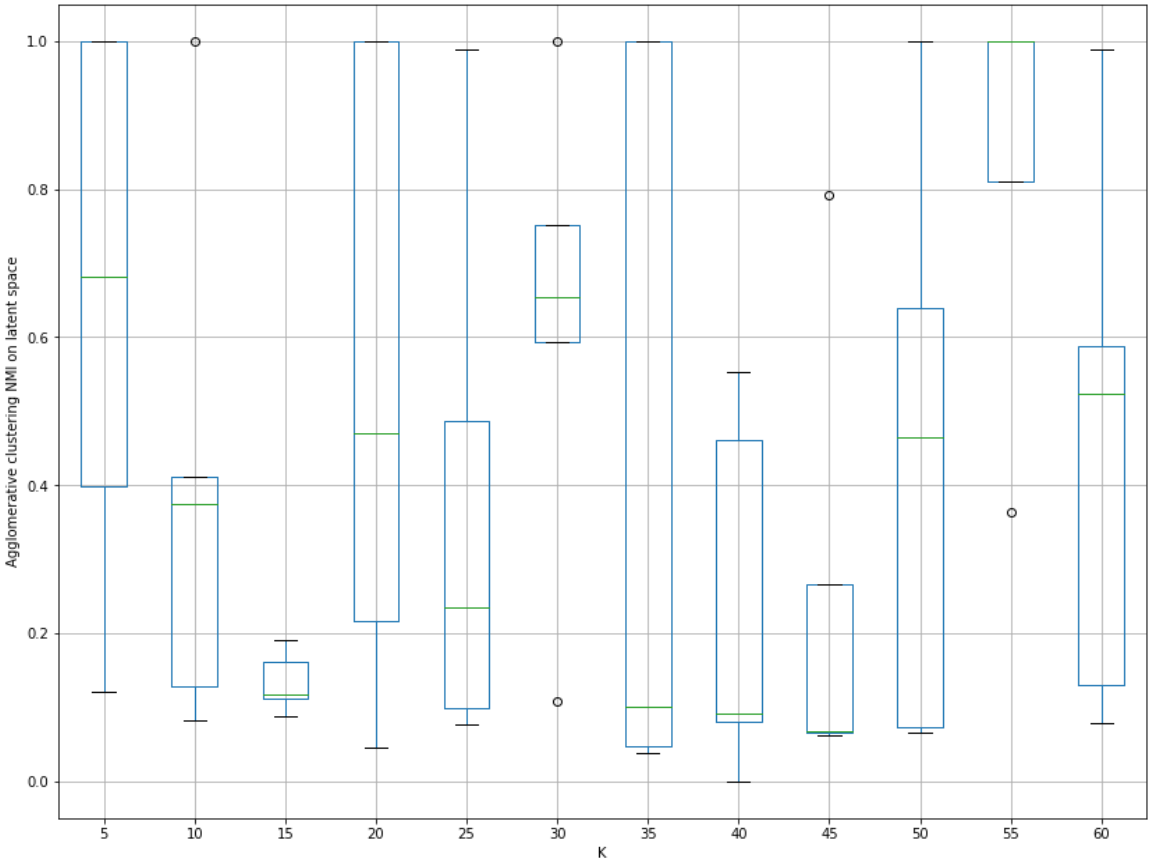
Boxplots:

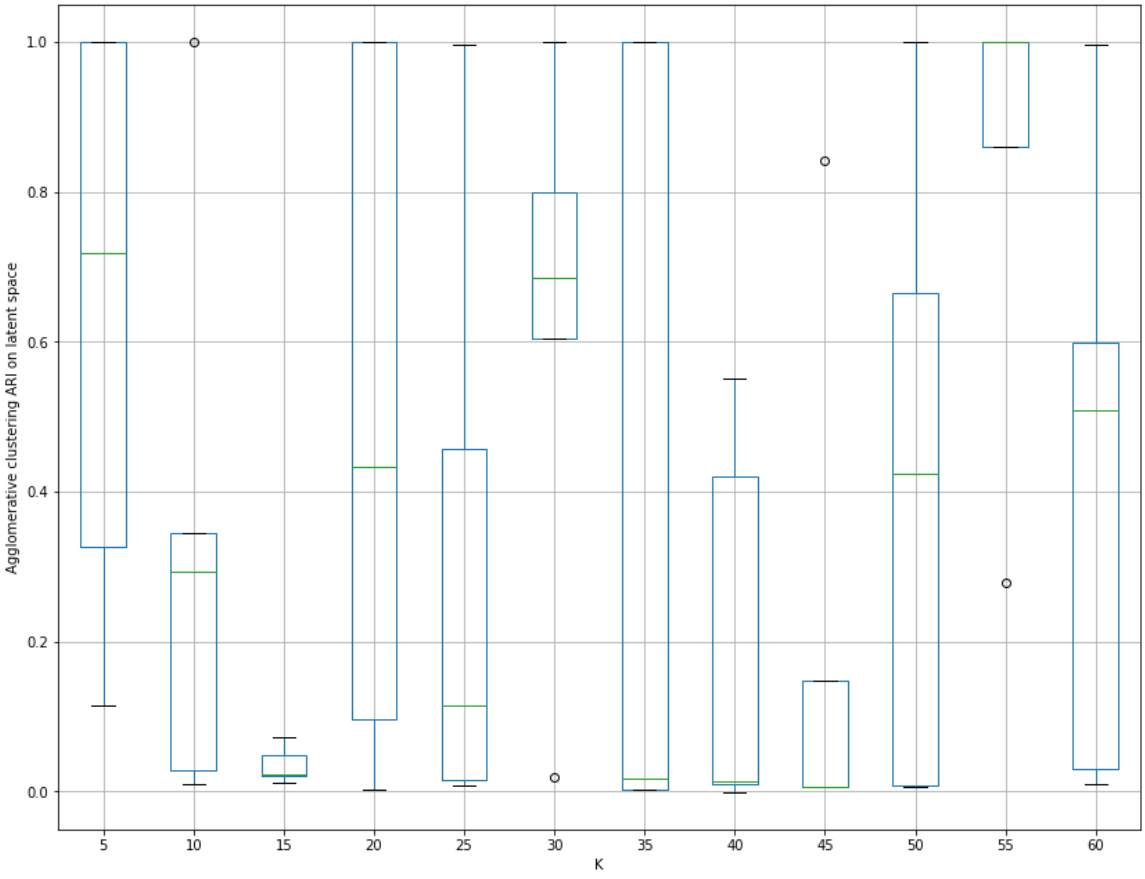


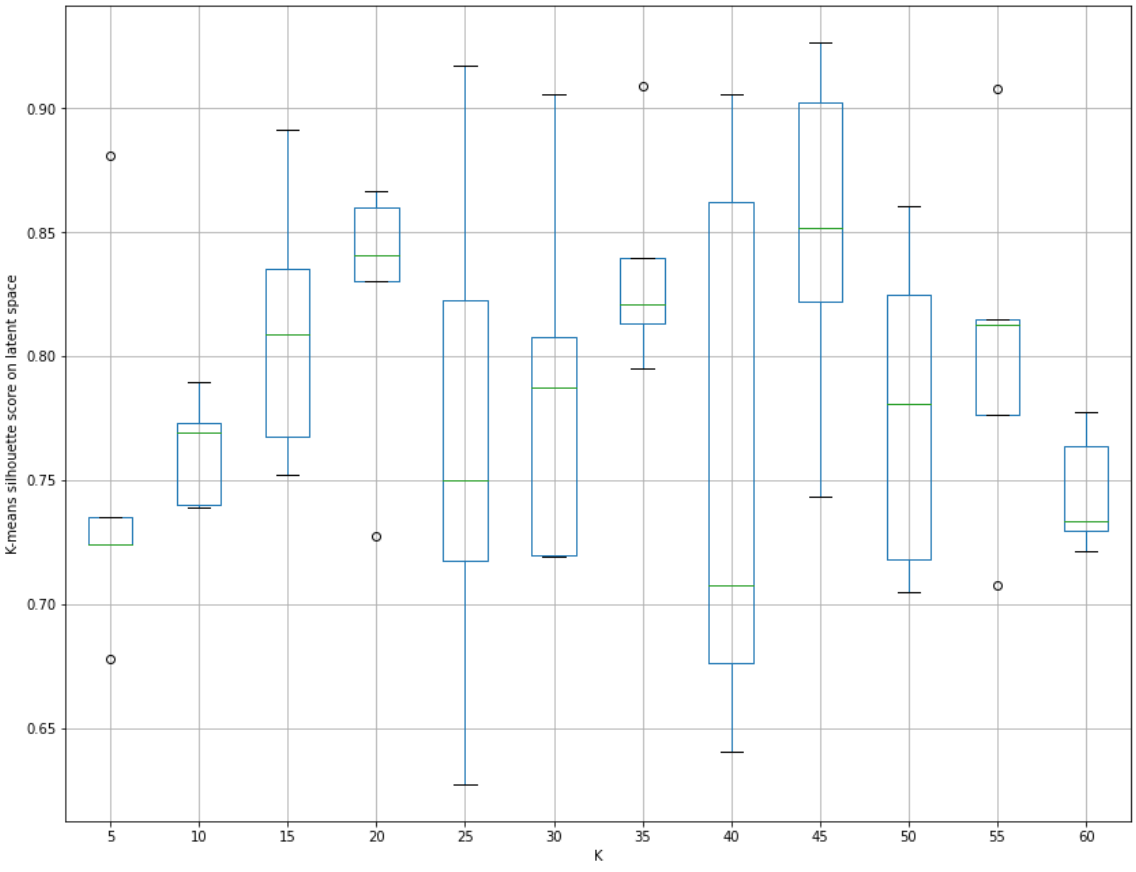

K-means NMI on latent space
1.0
0.8
0.6
0.4
0.2
0.0
5
10
15
20
25
30
35
40
45
50
55
60
K


K-means ARI on latent space
1.0
0.8
0.6
0.4
0.2
0.0
5
10
15
20
25
30
35
40
45
50
55
60
K

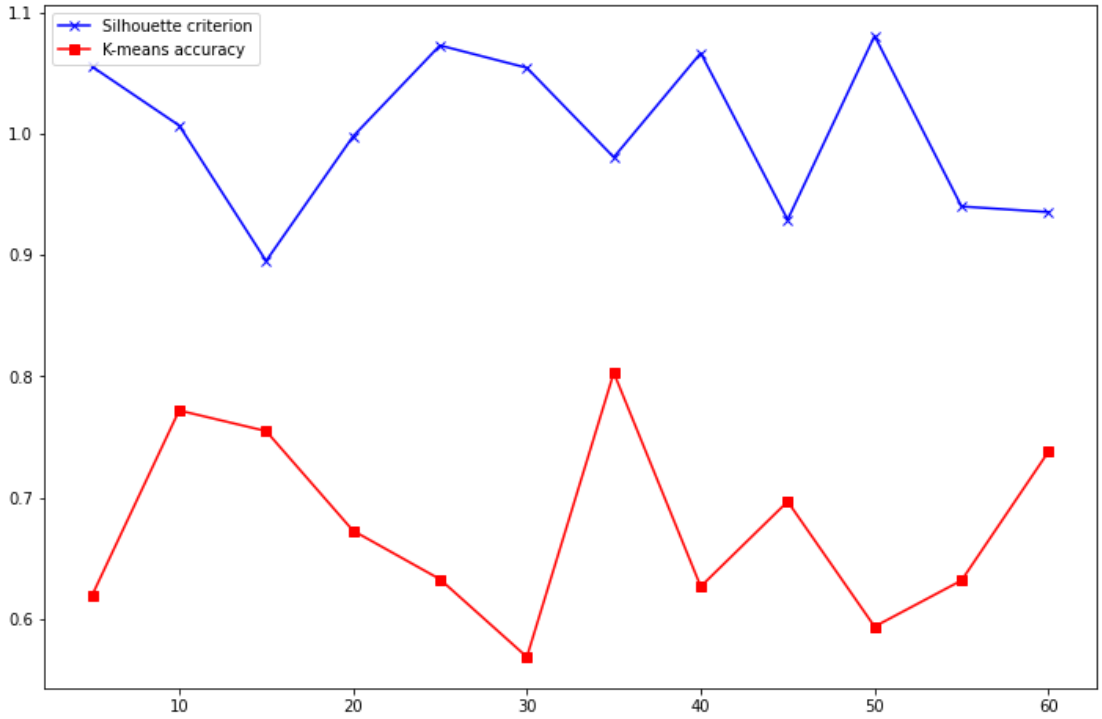

Agglomerative clustering accuracy on latent space
1.0
0.9
0.8
0.7
0.6
0.5
5
10
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering NMI on latent space
1.0
0.8
0.6
0.4
0.2
0.0
5
10
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering ARI on latent space
1.0
0.8
0.6
0.4
0.2
0.0
5
10
15
20
25
30
35
40
45
50
55
60
K


K-means silhouette score on latent space
0.90
0.85
0.80
0.75
0.70
0.65
5
10
15
20
25
30
35
40
45
50
55
60
K

Gaussian rings overclustering criterion:



Comments:

Taking the global maximum of the overclustering criterion into consideration, we would make the choice k = 50 for this dataset. The performance with this choice varies a lot and is an average choice, as it doesn't reach the heights of other choices. It definitely isn't one of the worst choices either. However, other choices sometimes achieve the value 1 for some measures, while the choice k = 50 does not achieve this.

Gaussian squeezed blobs:

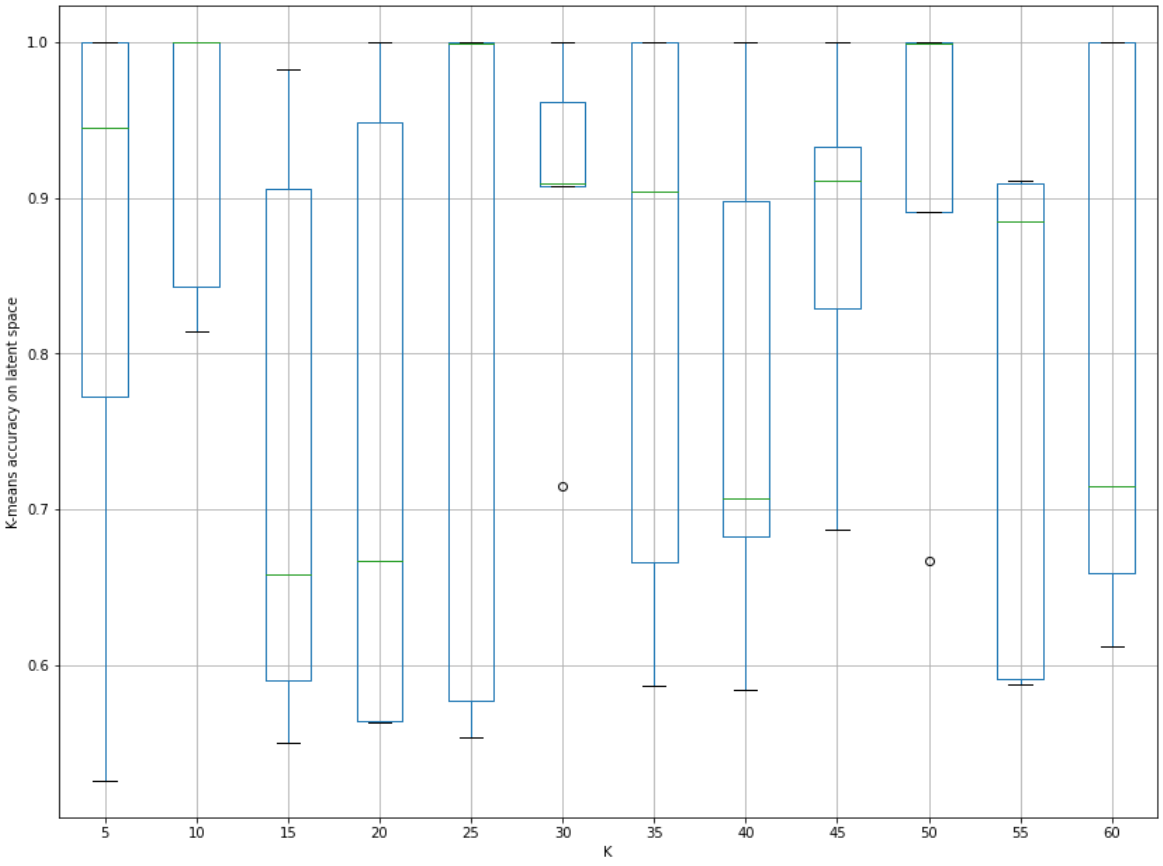
Autoencoder architecture details and data processing details:

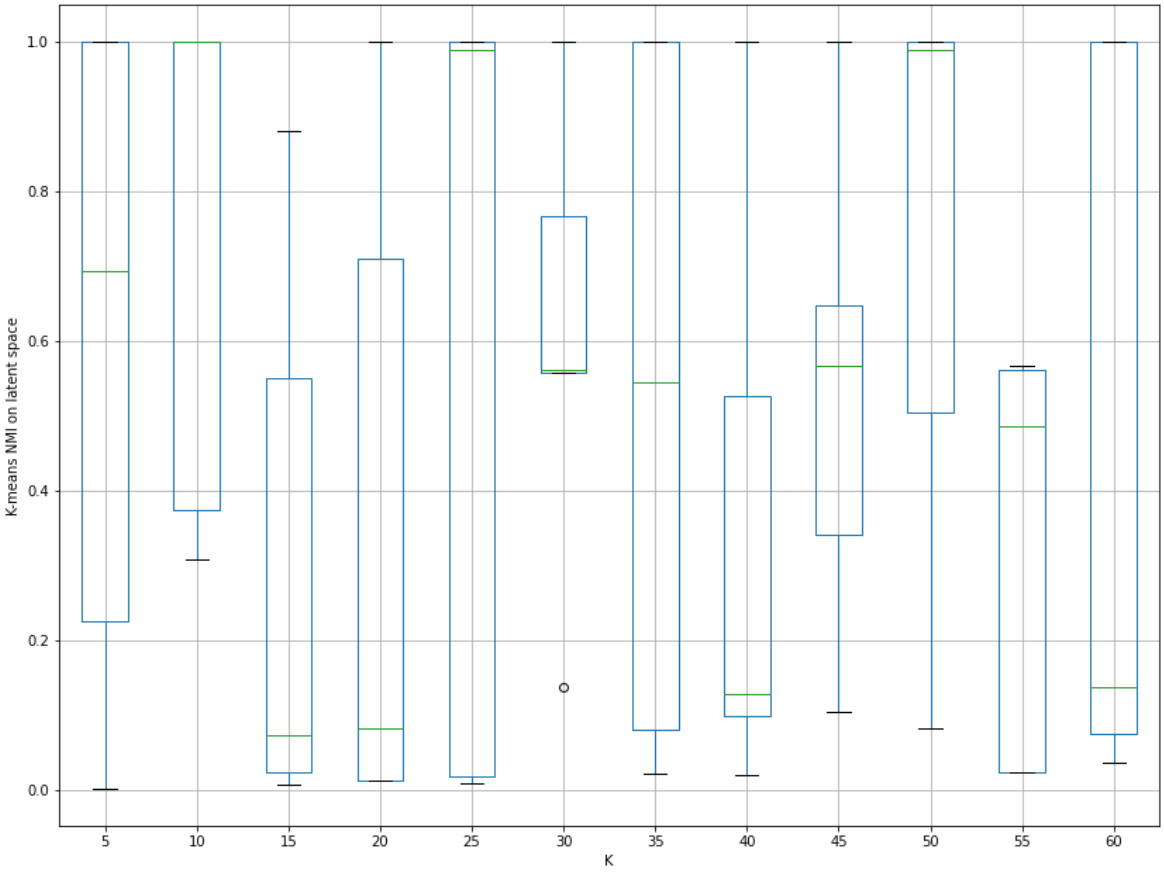
Hidden layer 1 neurons	800
Hidden layer 2 neurons	300
Latent dimension	1
Connections	Fully Connected (in all layers)
Activation function	Sigmoid (in all layers)
Training epochs	50
Loss function	MSE

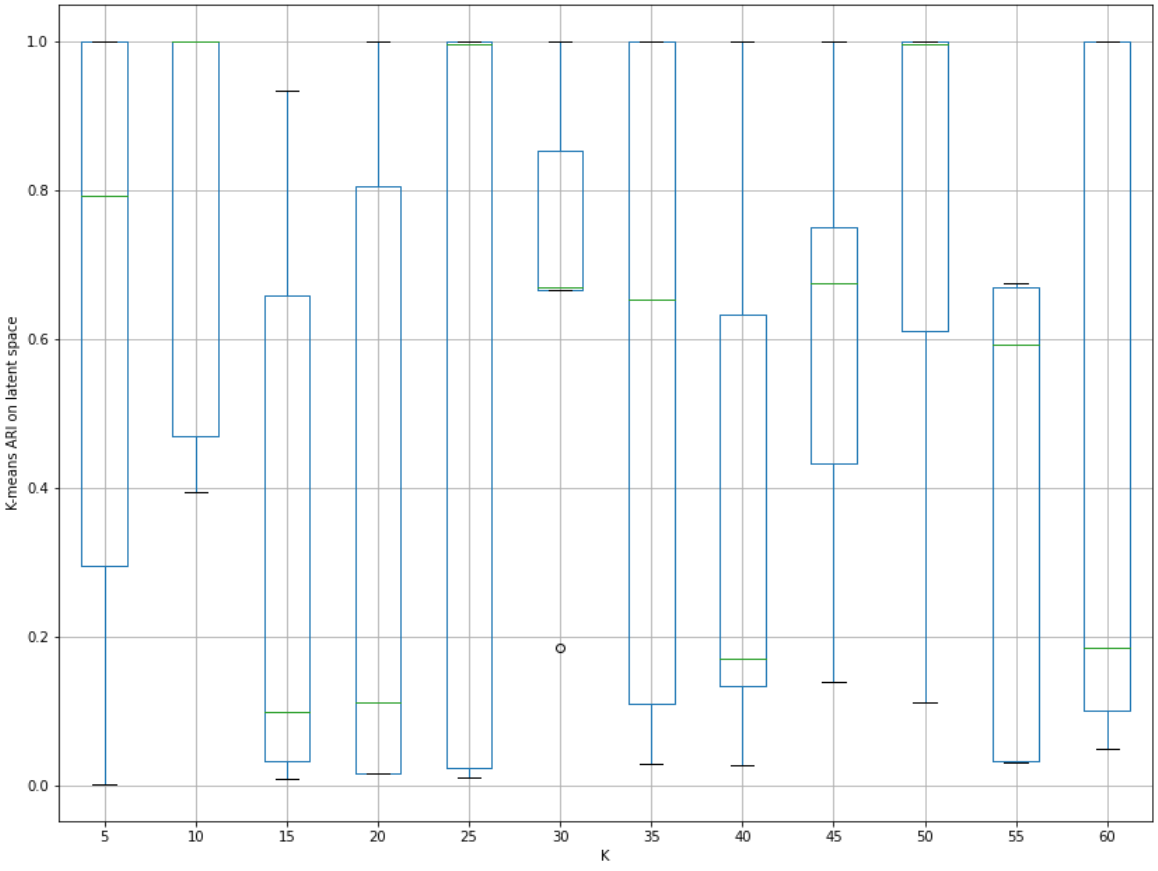
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	100
Datapoints	1000
Scaling in the initial data space	None
Scaling in the latent space	None

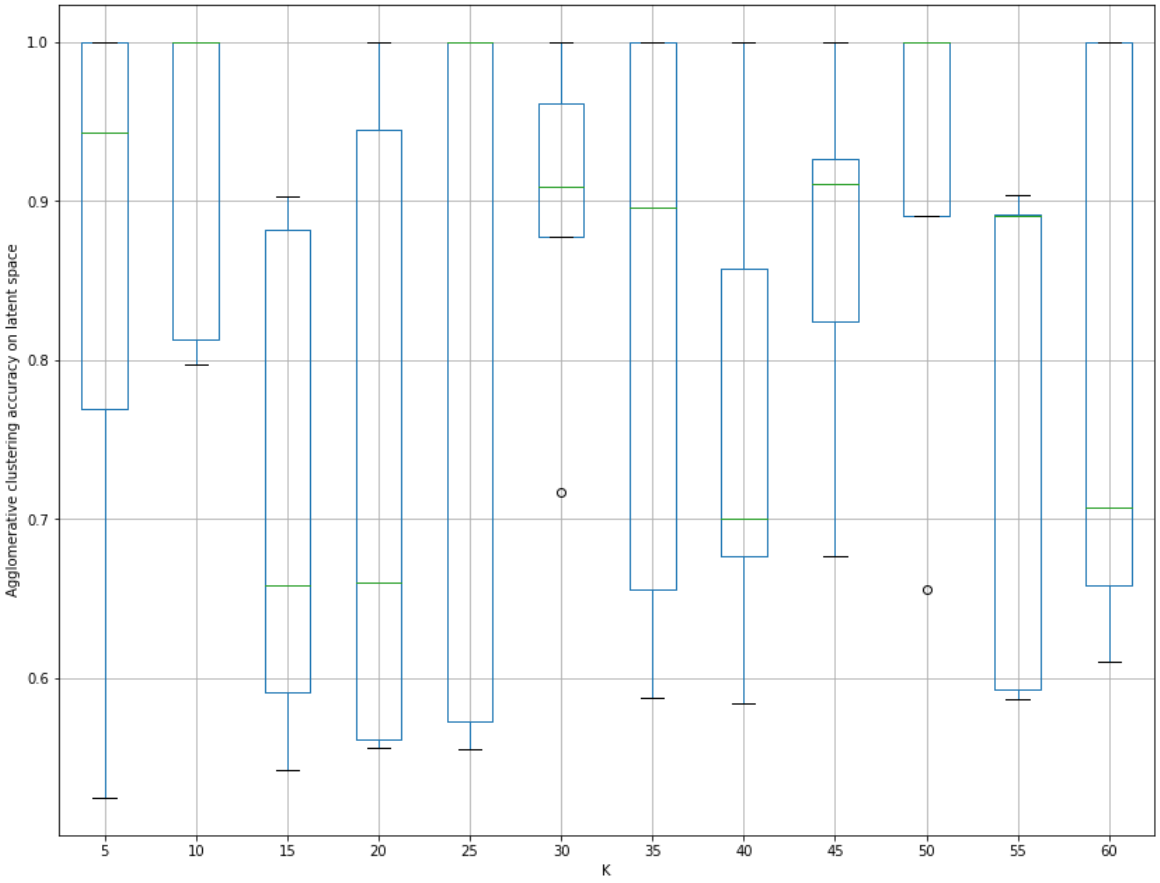
Autoencoder-trained-with-centroids method parameters for gaussian squeezed blobs dataset

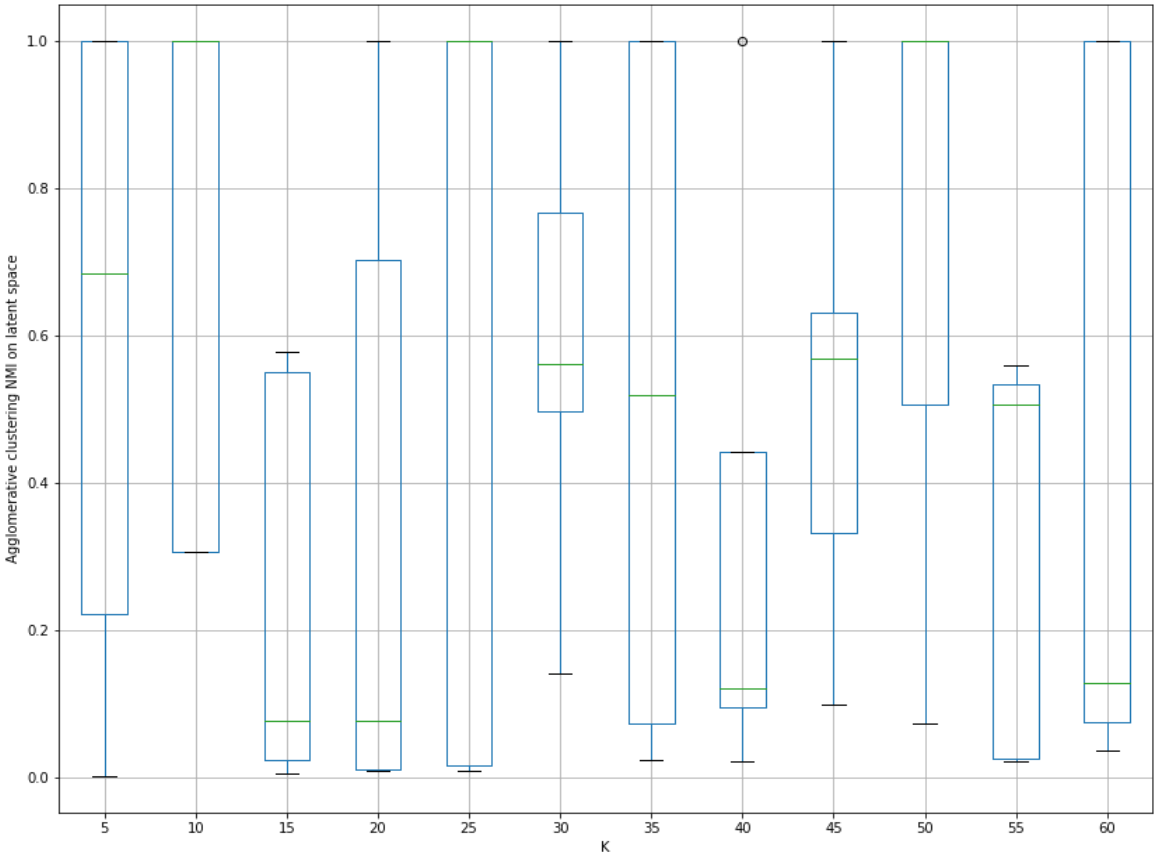
Boxplots:

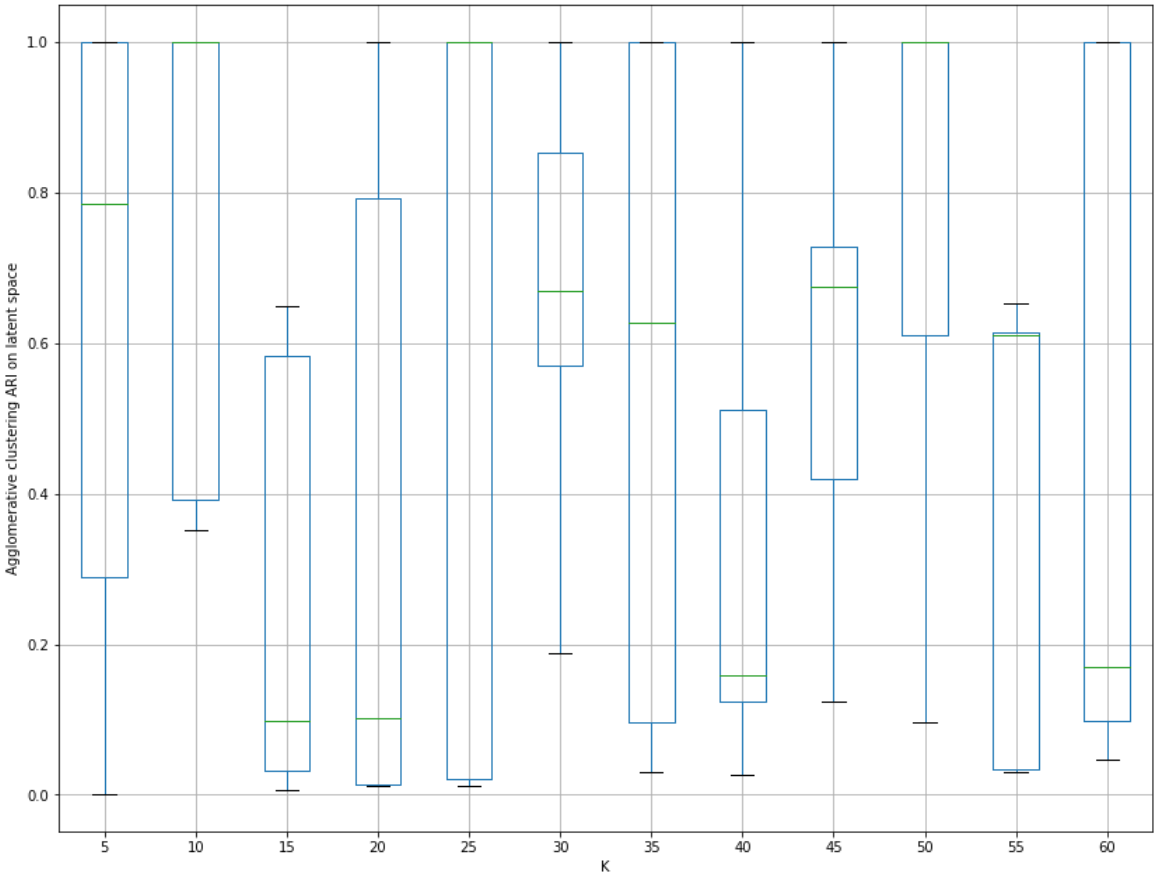



K-means NMI on latent space
1.0
0.8
0.6
0.4
0.2
0.0
5
10
15
20
25
30
35
40
45
50
55
60
K

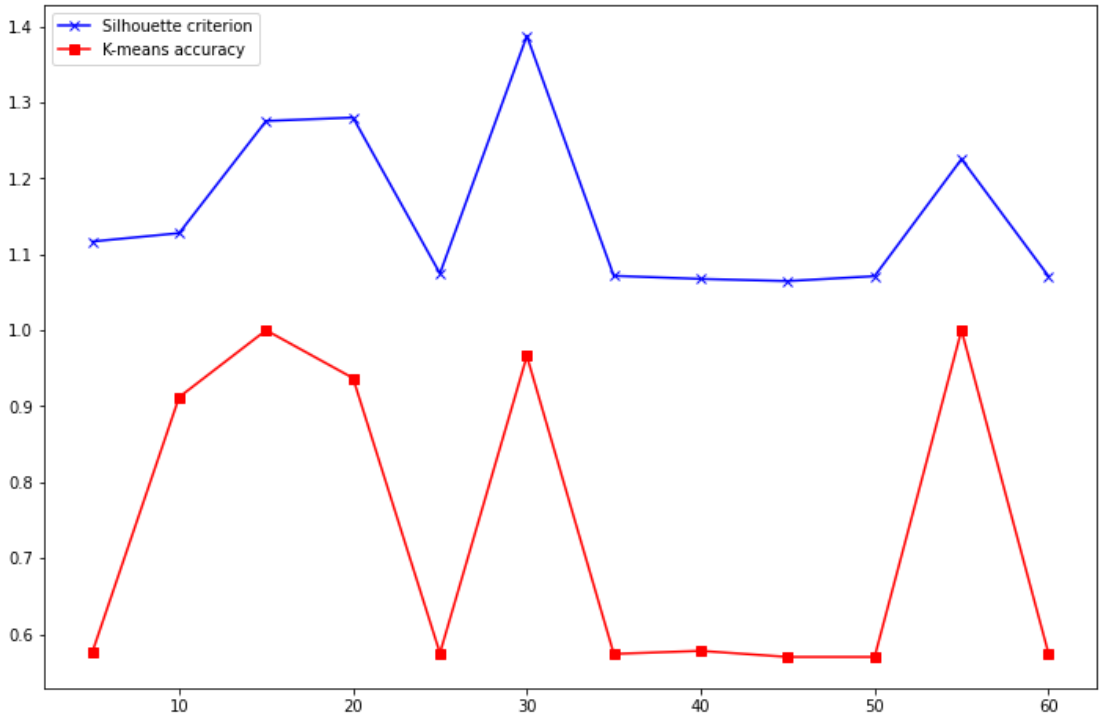

K-means ARI on latent space
1.0
0.8
0.6
0.4
0.2
0.0
5
10
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering accuracy on latent space
1.0
0.9
0.8
0.7
0.6
5
10
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering NMI on latent space
1.0
0.8
0.6
0.4
0.2
0.0
5
10
15
20
25
30
35
40
45
50
55
60
K



Gaussian squeezed blobs overclustering criterion:



Comments:

Taking the global maximum of the overclustering criterion into consideration, we would make the choice k = 30 for this dataset. Looking at the boxplots, this option achieves average performance when compared to the others, yet it has the lowest standard deviation of all. The overclustering criterion line seems to be following the k-means accuracy line perfectly. Generally, k = 50 is the best number of initial clusters for this dataset. It has a low standard deviation and often achieves scores close to 1, which is a massive improvement on the initial space clustering results. It must be noted that the overclustering criterion doesn't provide us with the optimal option, albeit a good one.

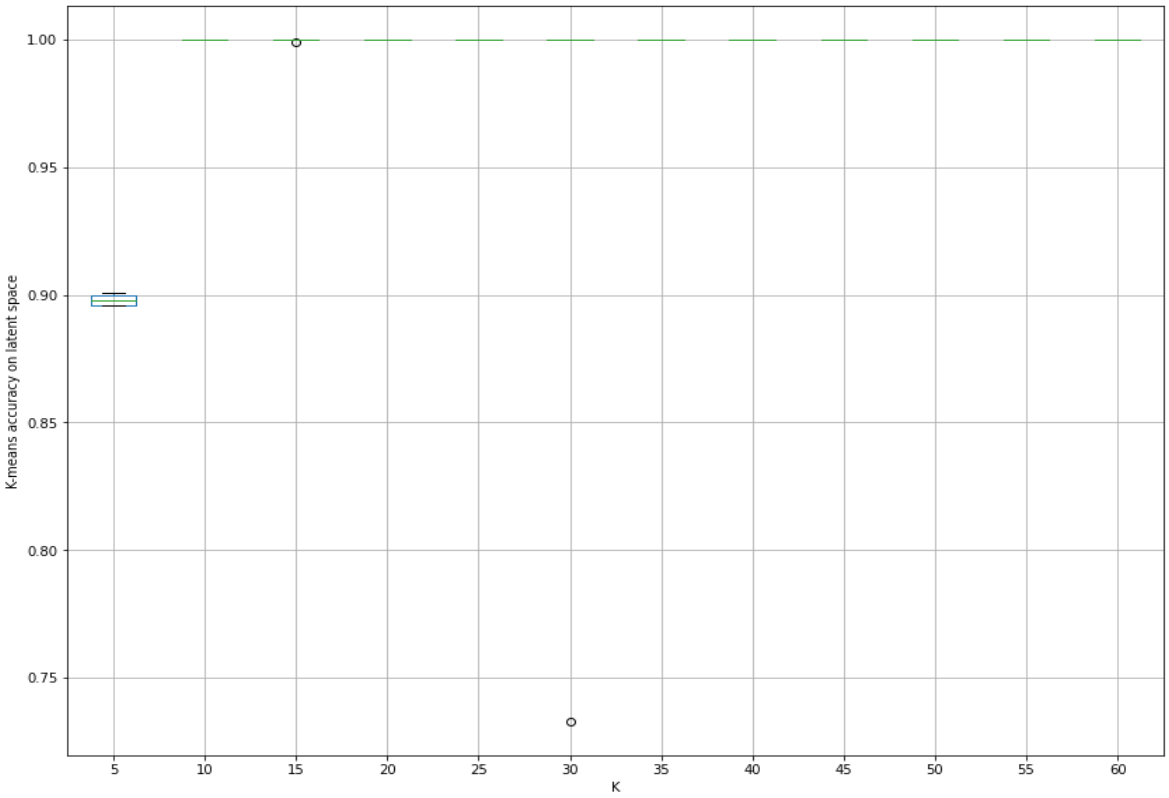
Moons dataset:

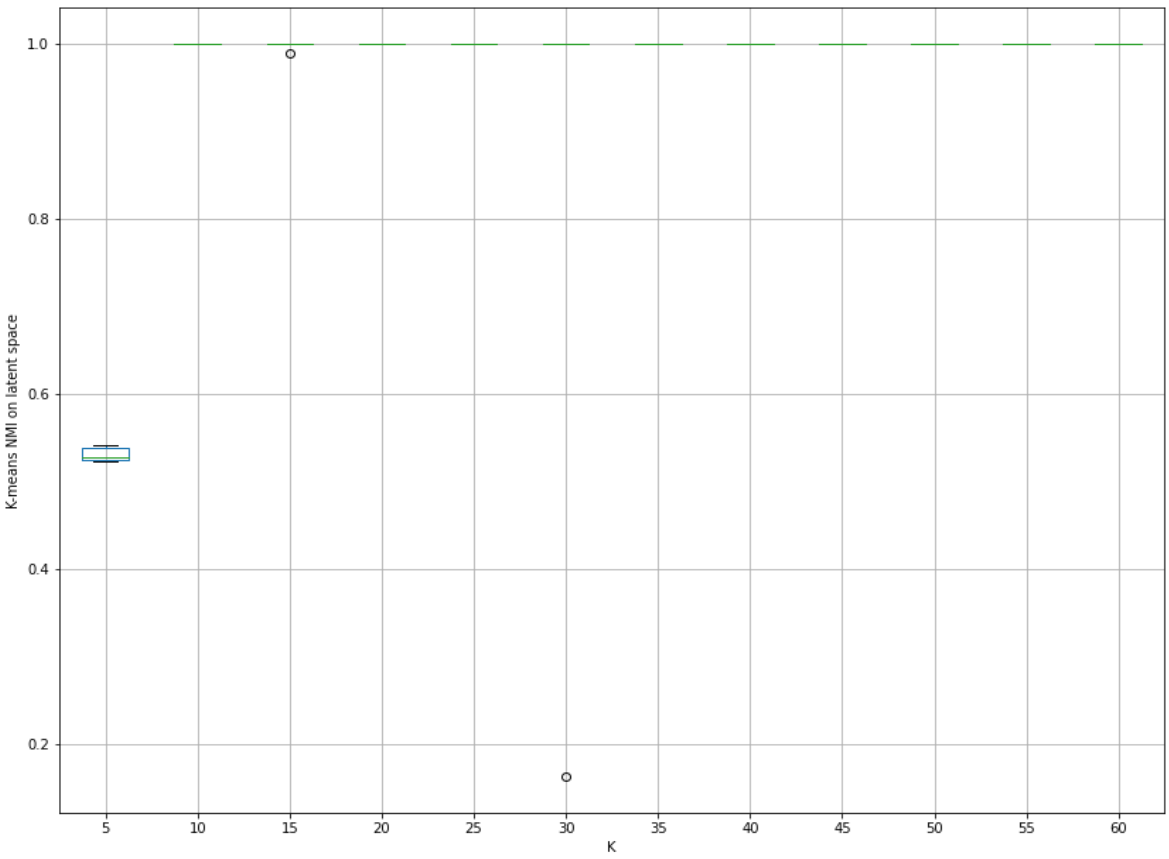
Autoencoder architecture and data processing details:

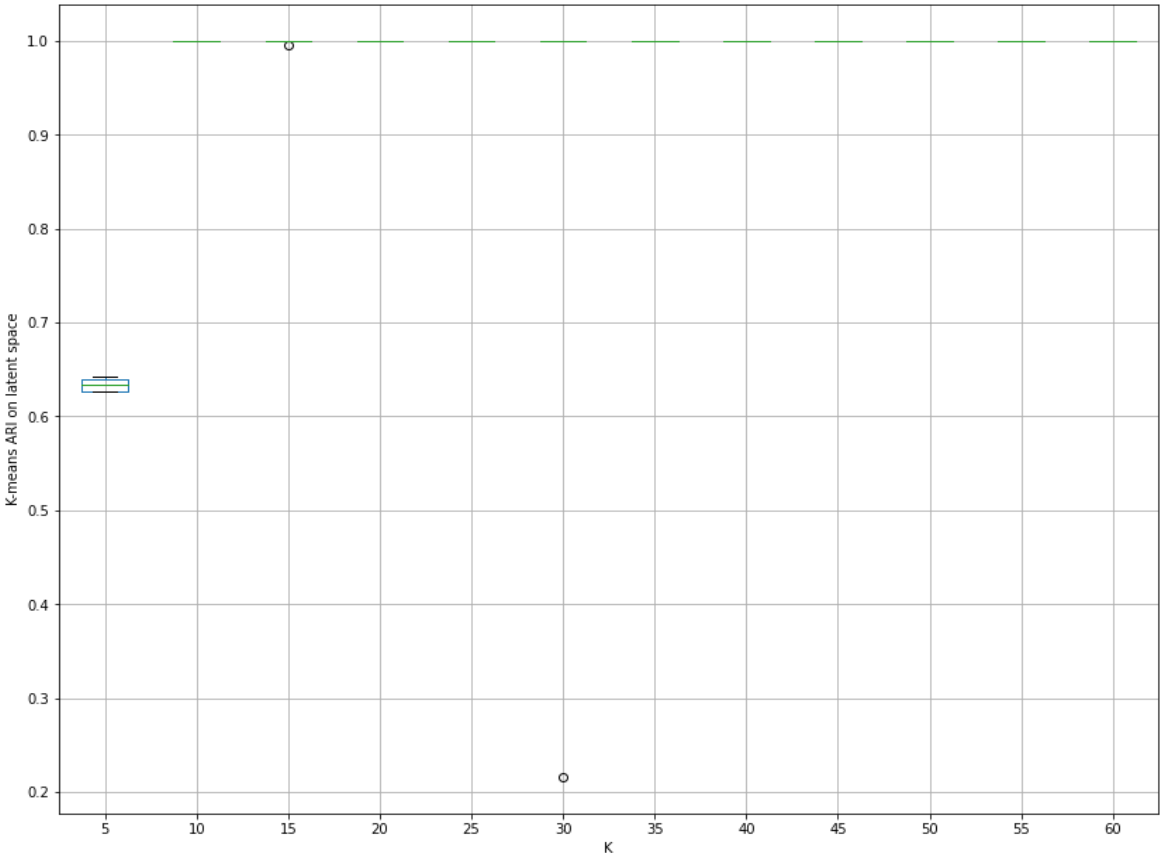
Hidden layer 1 neurons	400
Hidden layer 2 neurons	200
Latent dimension	1
Connections	Fully Connected (in all layers)
Activation function	Sigmoid (in all layers)
Training epochs	50
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	100
Datapoints	1000
Scaling in the initial data space	MinMax
Scaling in the latent space	None

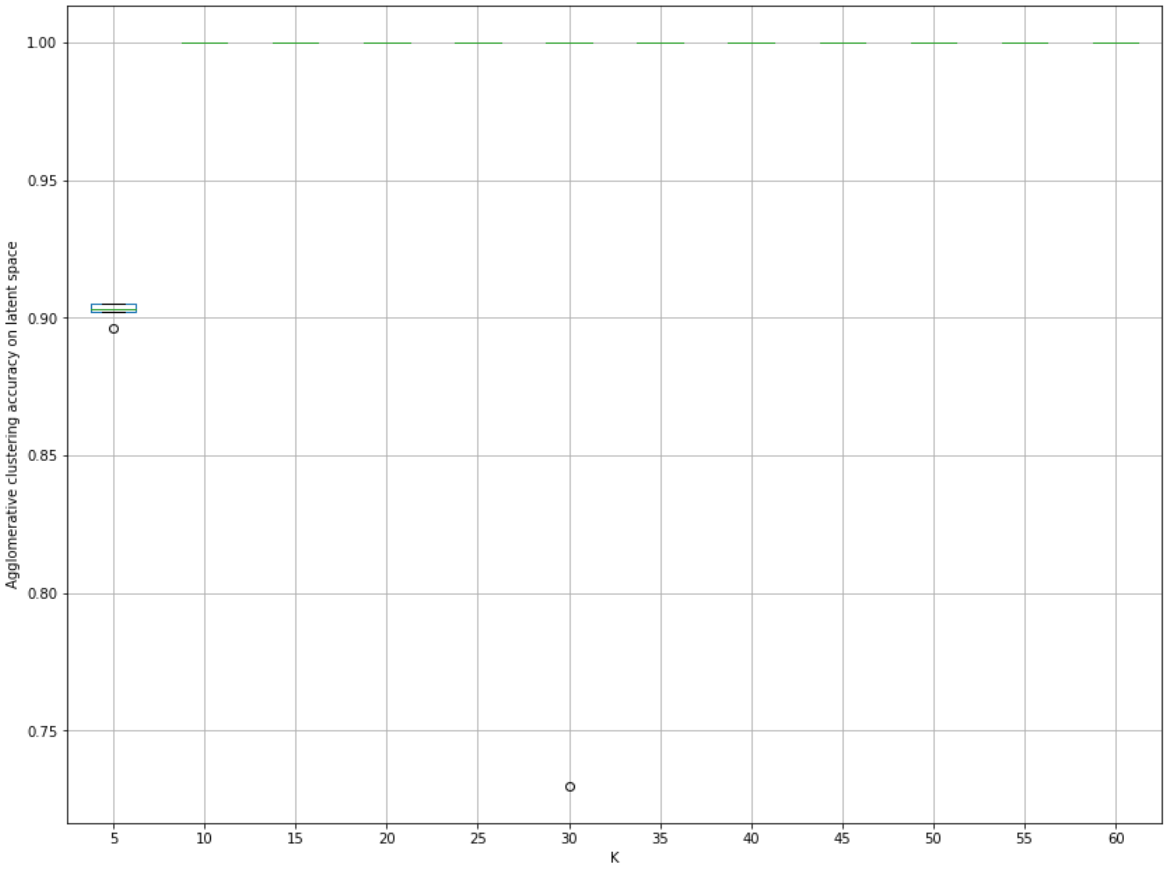
Autoencoder-trained-with-centroids method parameters for moons dataset

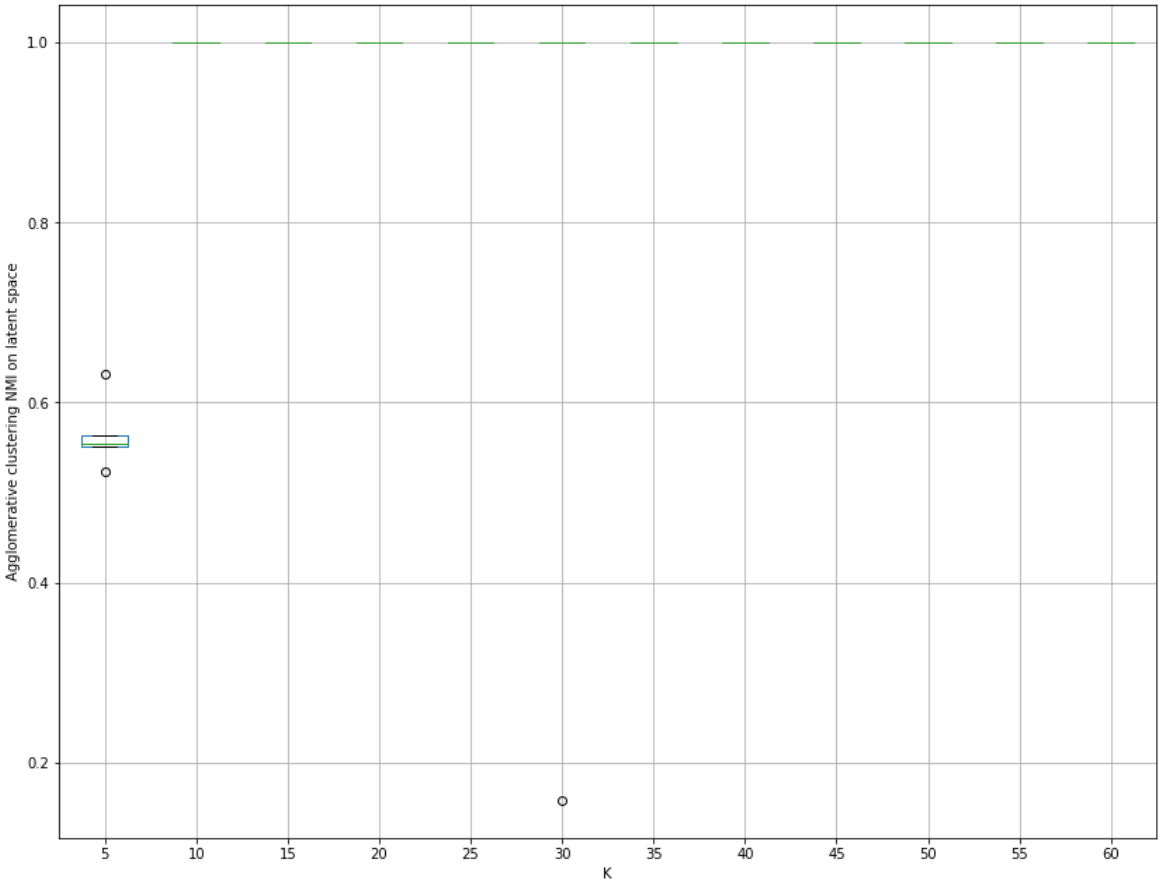
Boxplots:

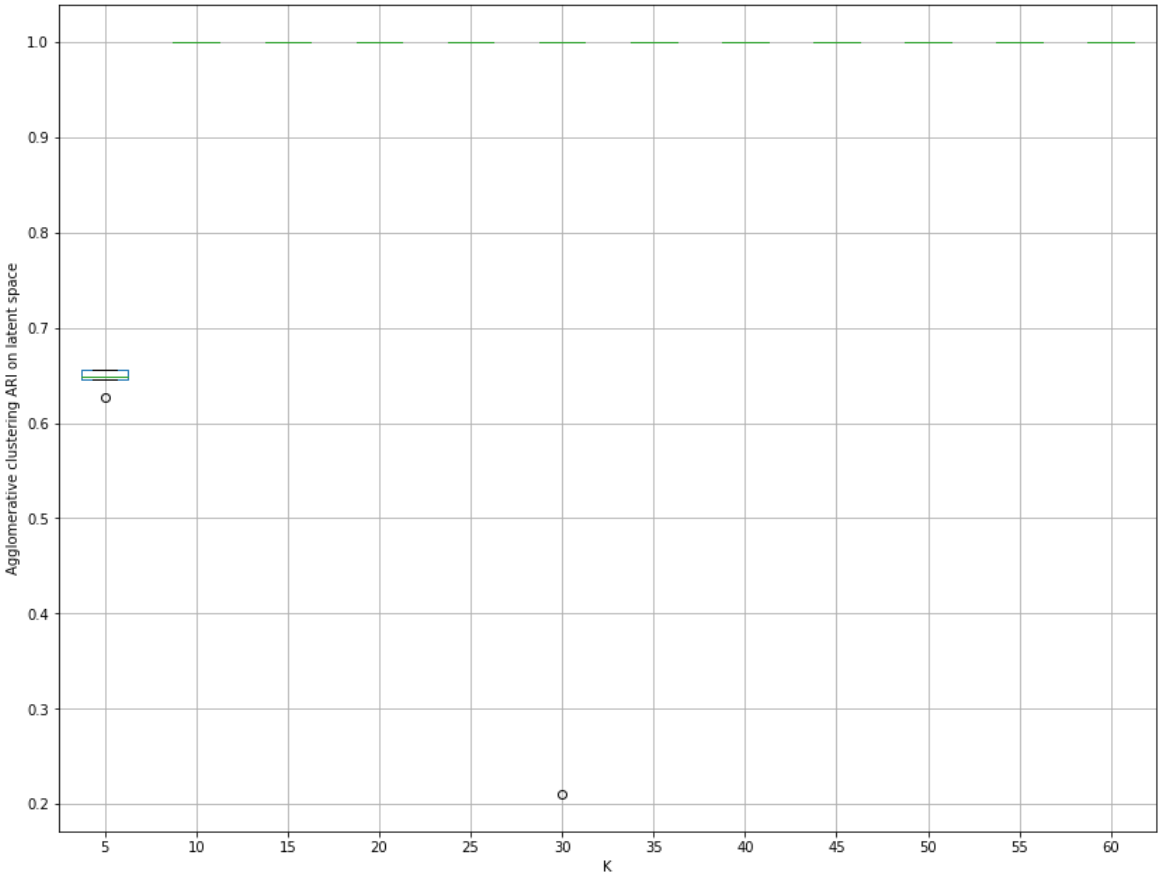

1.00
0.95
0.90
0.85
0.80
0.75
K-means accuracy on latent space
5
10
15
20
25
30
35
40
45
50
55
60
K

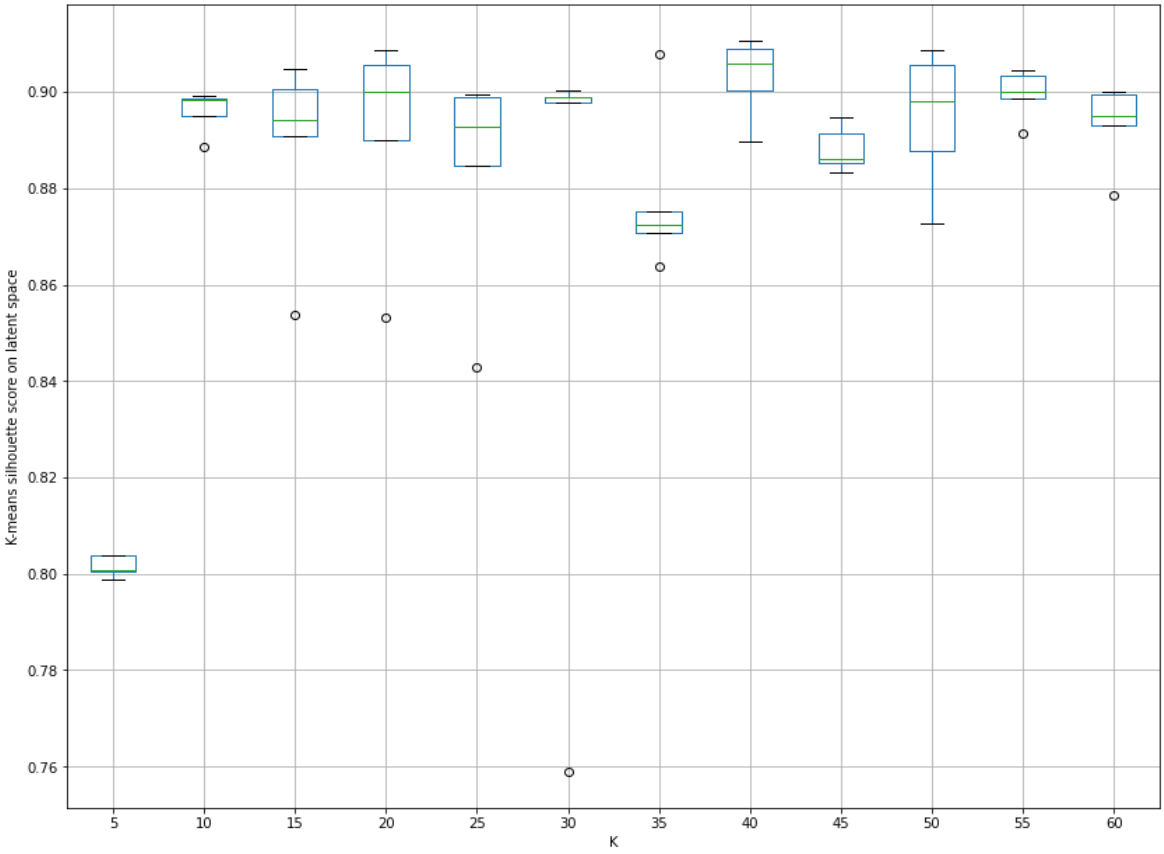

1.0
0.8
0.6
0.4
0.2
K-means NMI on latent space
5
10
15
20
25
30
35
40
45
50
55
60
K


K-means ARI on latent space
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
5
10
15
20
25
30
35
40
45
50
55
60
K

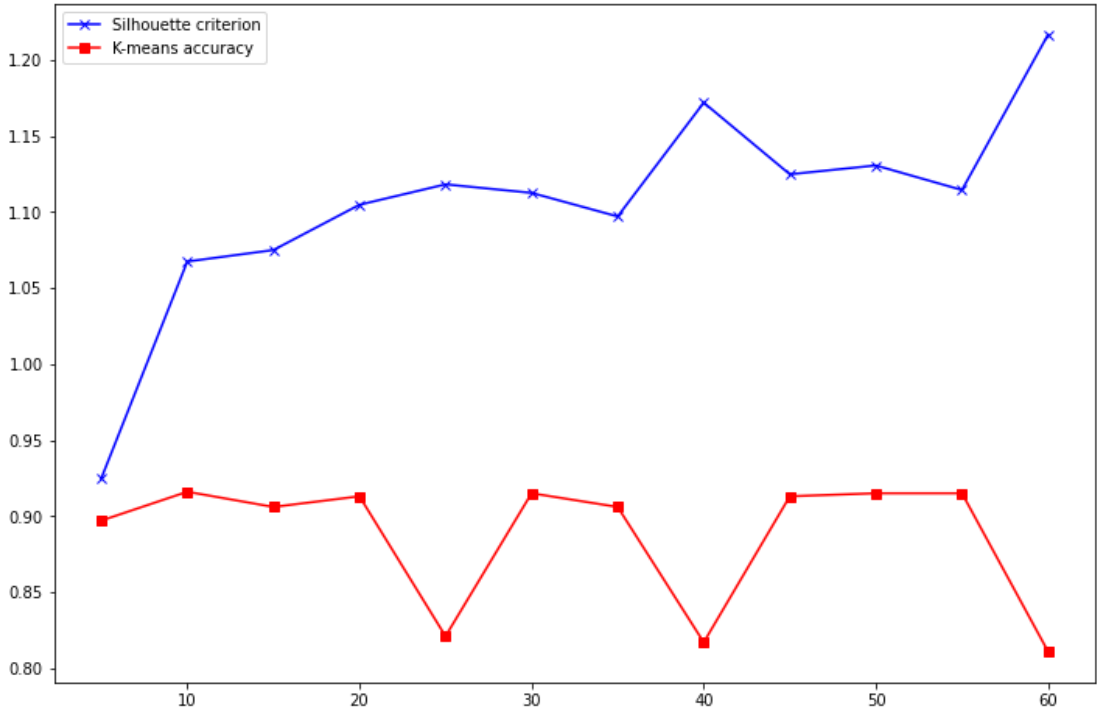

Agglomerative clustering accuracy on latent space
1.00
0.95
0.90
0.85
0.80
0.75
5
10
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering NMI on latent space
1.0
0.8
0.6
0.4
0.2
5
10
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering ARI on latent space
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
5
10
15
20
25
30
35
40
45
50
55
60
K



Moons overclustering criterion:



Comments:

The results of this method on the moons dataset are arguably the most impressive of all, because the value of all metrics, apart from the silhouette score, are steadily 1 for choices $k \geq 10$. This is also a considerable improvement on the stats of the initial space. Taking the global maximum of the overclustering criterion into consideration, we would make the choice k = 60 for this dataset. As mentioned above, this value gives perfect results, but this can also be said about all the other choices bar k = 5. As a consequence, the choice in this dataset is not of great importance. In addition, this dataset is one of the few datasets in which the silhouette coefficient line doesn't follow the trajectory of the k-means accuracy line. Despite the similarity of all the results, k = 40 edges it as the best choice purely because of its slightly better k-means silhouette score.

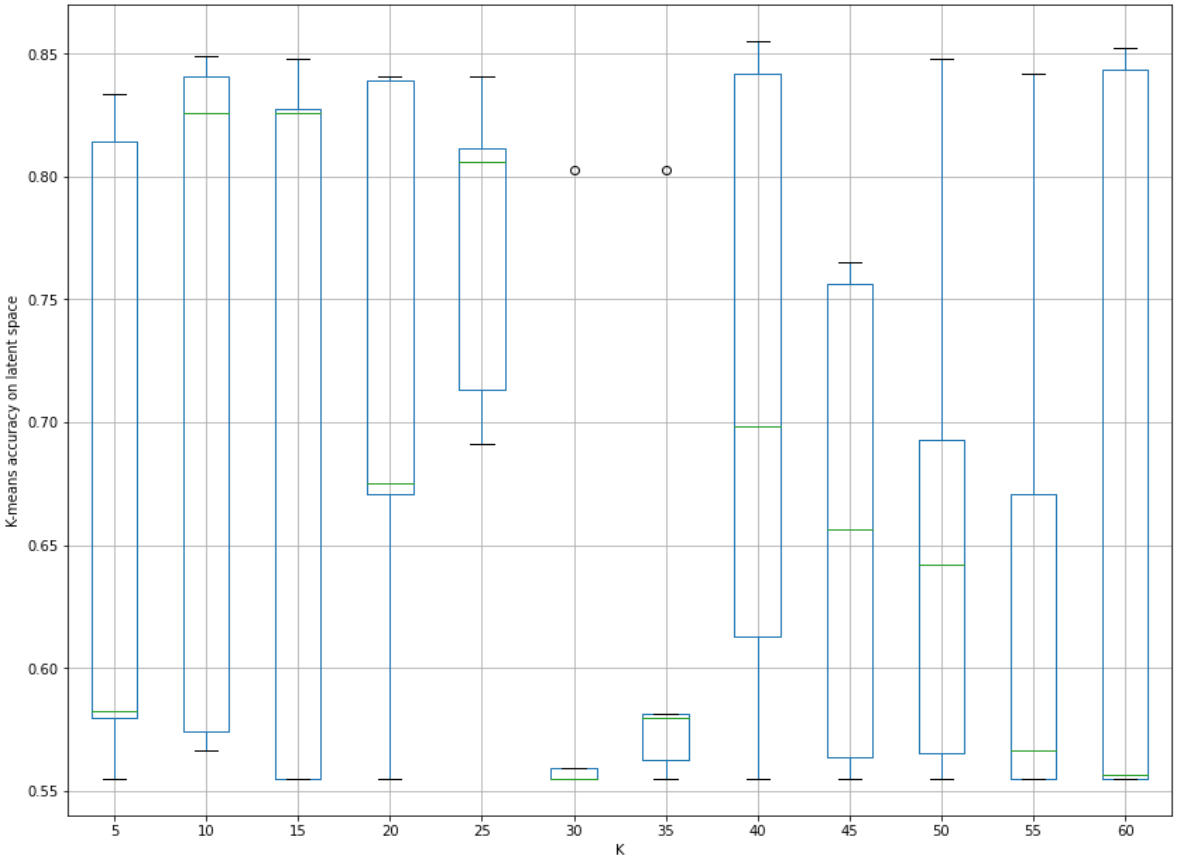
Australian dataset:

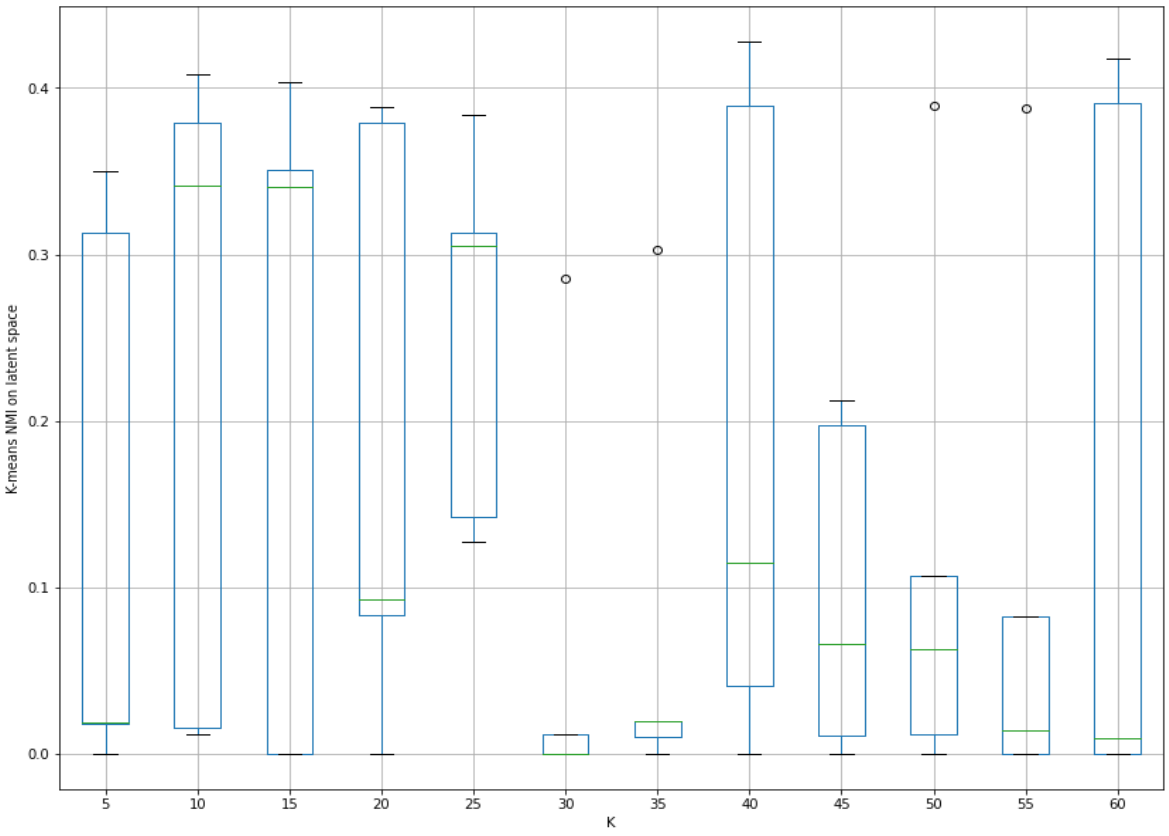
Autoencoder architecture and data processing details:

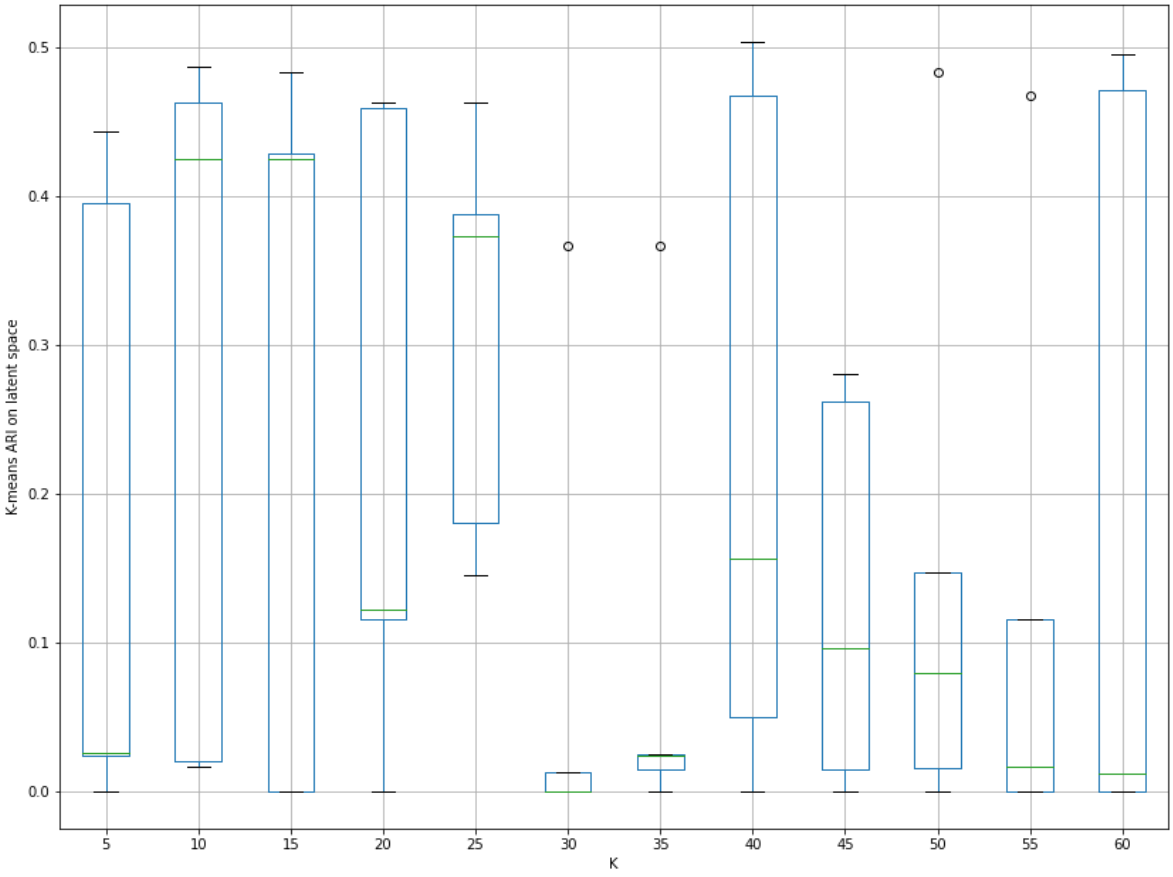
Hidden layer 1 neurons	500
Hidden layer 2 neurons	300
Latent dimension	5
Connections	Fully Connected (in all layers)
Activation function	Sigmoid (in all layers)
Training epochs	50
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	69
Datapoints	690
Scaling in the initial data space	MinMax
Scaling in the latent space	None

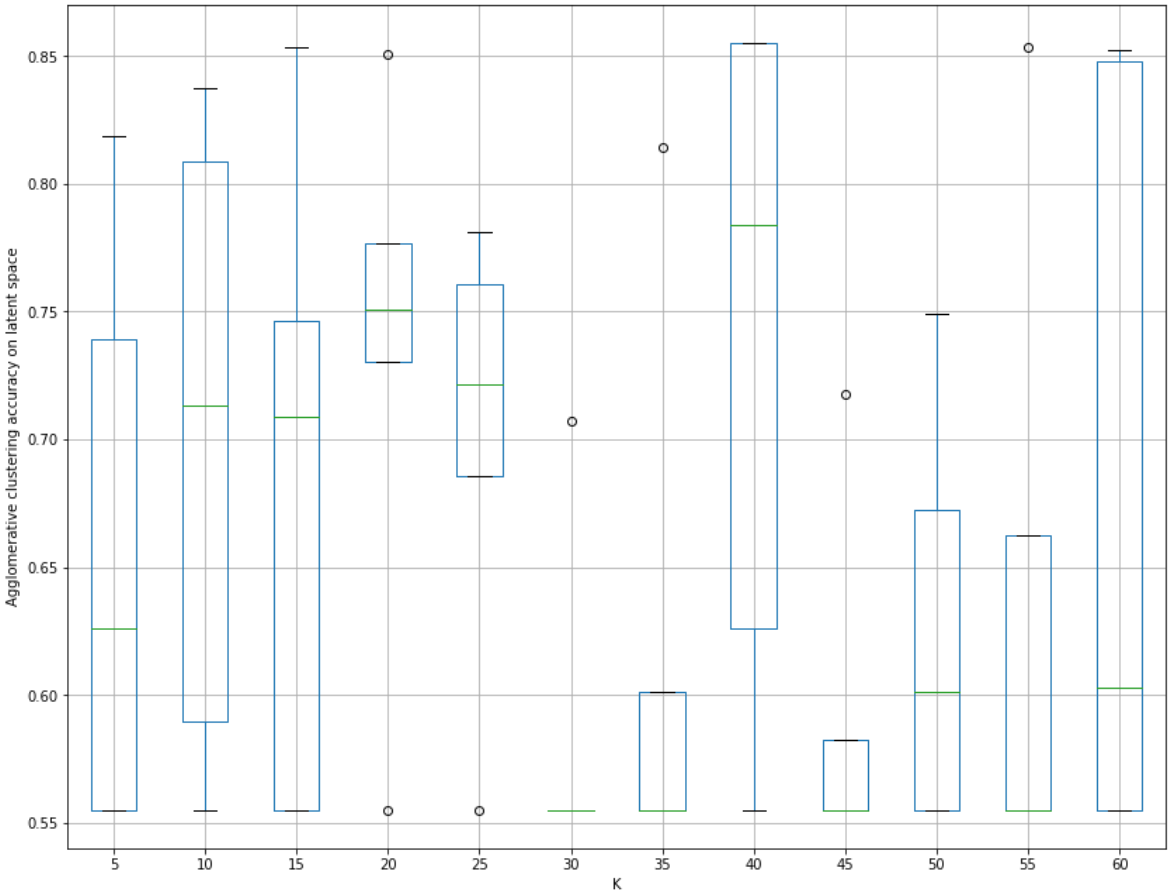
Autoencoder-trained-with-centroids method parameters for Australian dataset

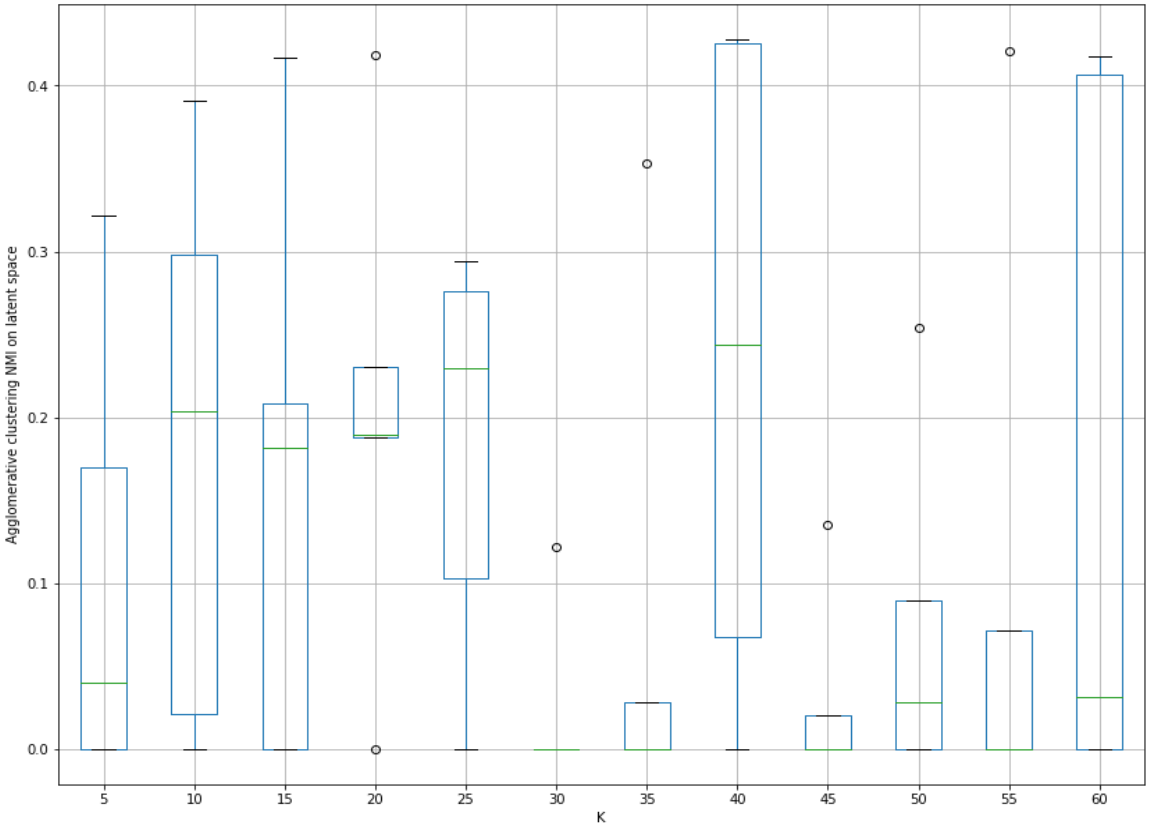
Boxplots:

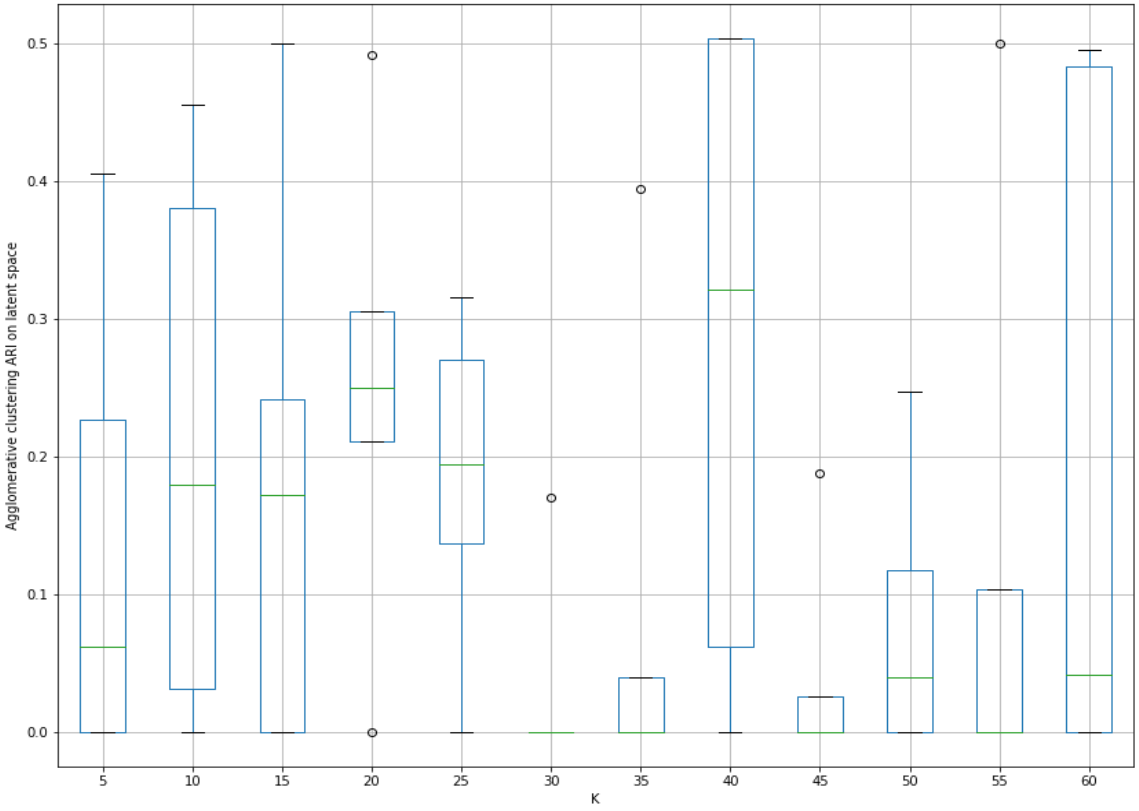

K-means accuracy on latent space
0.85
0.80
0.75
0.70
0.65
0.60
0.55
5
10
15
20
25
30
35
40
45
50
55
60
K


K-means NMI on latent space
0.4
0.3
0.2
0.1
0.0
5
10
15
20
25
30
35
40
45
50
55
60
K

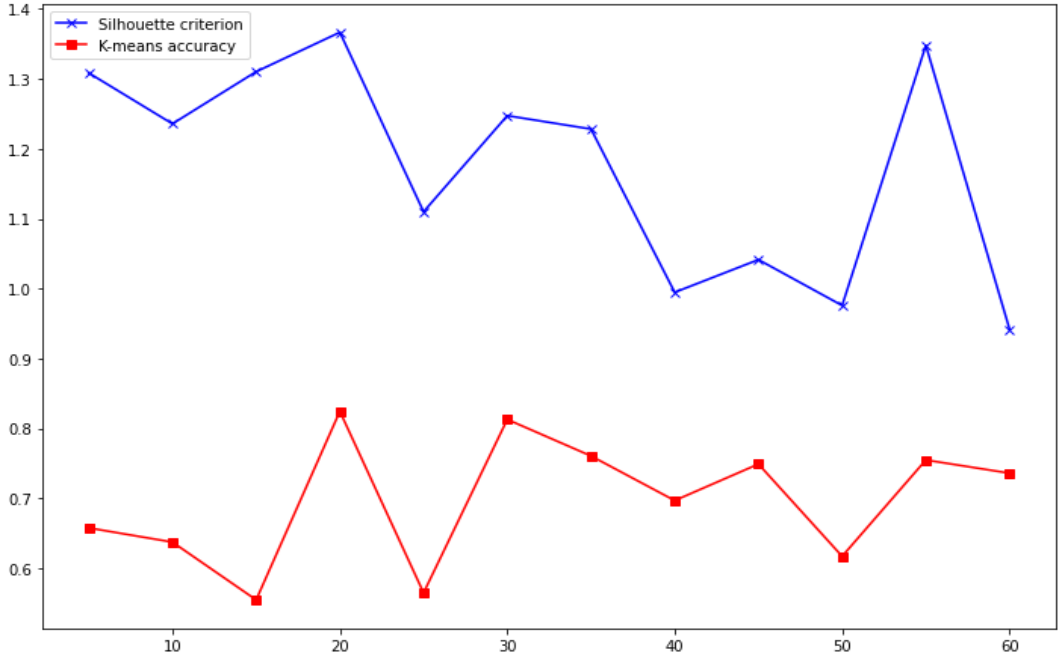

K-means ARI on latent space
0.0
0.1
0.2
0.3
0.4
0.5
5
10
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering accuracy on latent space
0.55
0.60
0.65
0.70
0.75
0.80
0.85
5
10
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering NMI on latent space
0.0
0.1
0.2
0.3
0.4
5
10
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering ARI on latent space
0.0
0.1
0.2
0.3
0.4
0.5
5
10
15
20
25
30
35
40
45
50
55
60
K

Australian dataset overclustering criterion:



Comments:

This dataset is an example of our clustering methods not improving in the metrics, when compared to the traditional algorithms. The results are pretty close to the initial. However, we have to comment on the use of the overclustering criterion. In this case, the silhouette line follows the ups and downs of the k-means accuracy line. Where the former has a local maximum, the latter also has one. The same goes for the local minima. Taking the global maximum of the overclustering criterion into consideration, we would make the choice of k = 20 initial clusters. If we take a look into the boxplots, we will come to the conclusion that this choice gives us close to the best results in each metric. It also has a low standard deviation, compared to other choices. Therefore, the overclustering criterion worked pretty well in this case.

Iris dataset:

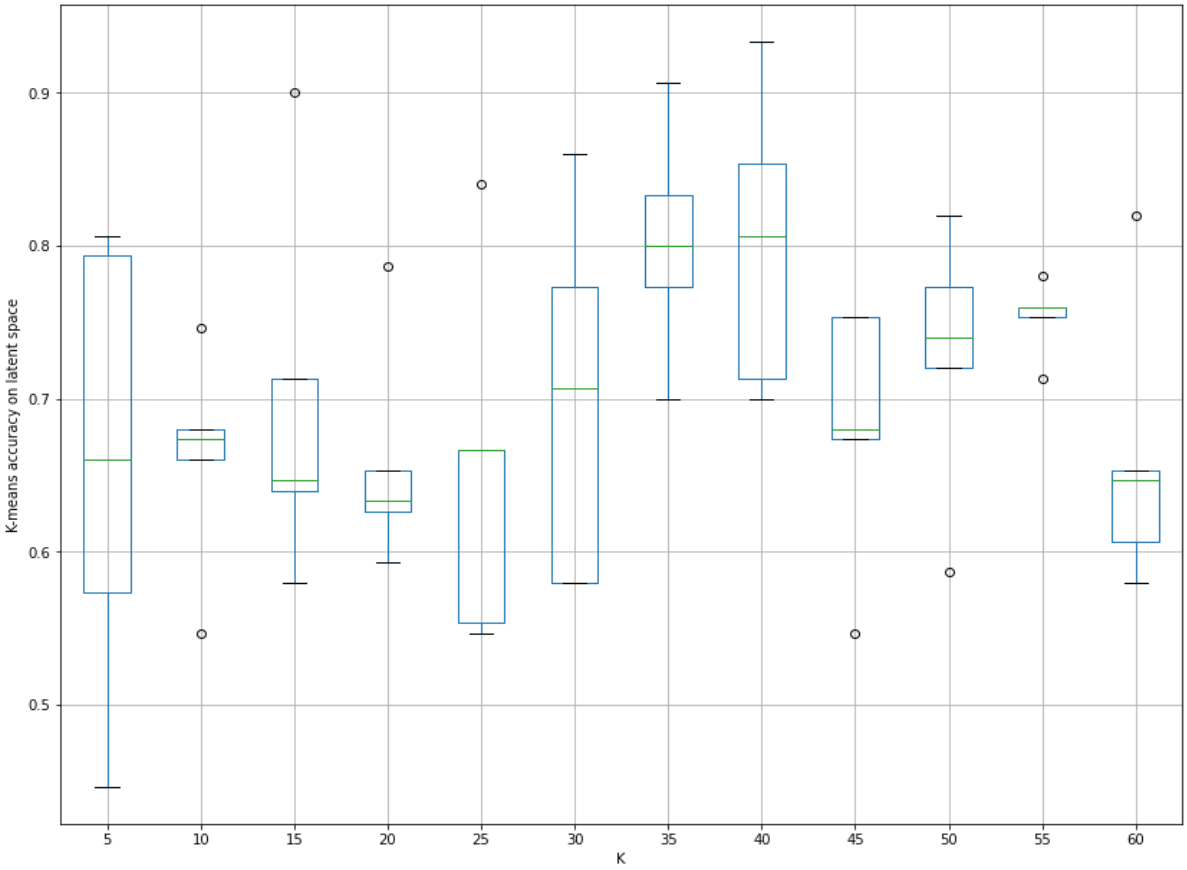
Autoencoder architecture and data processing details:

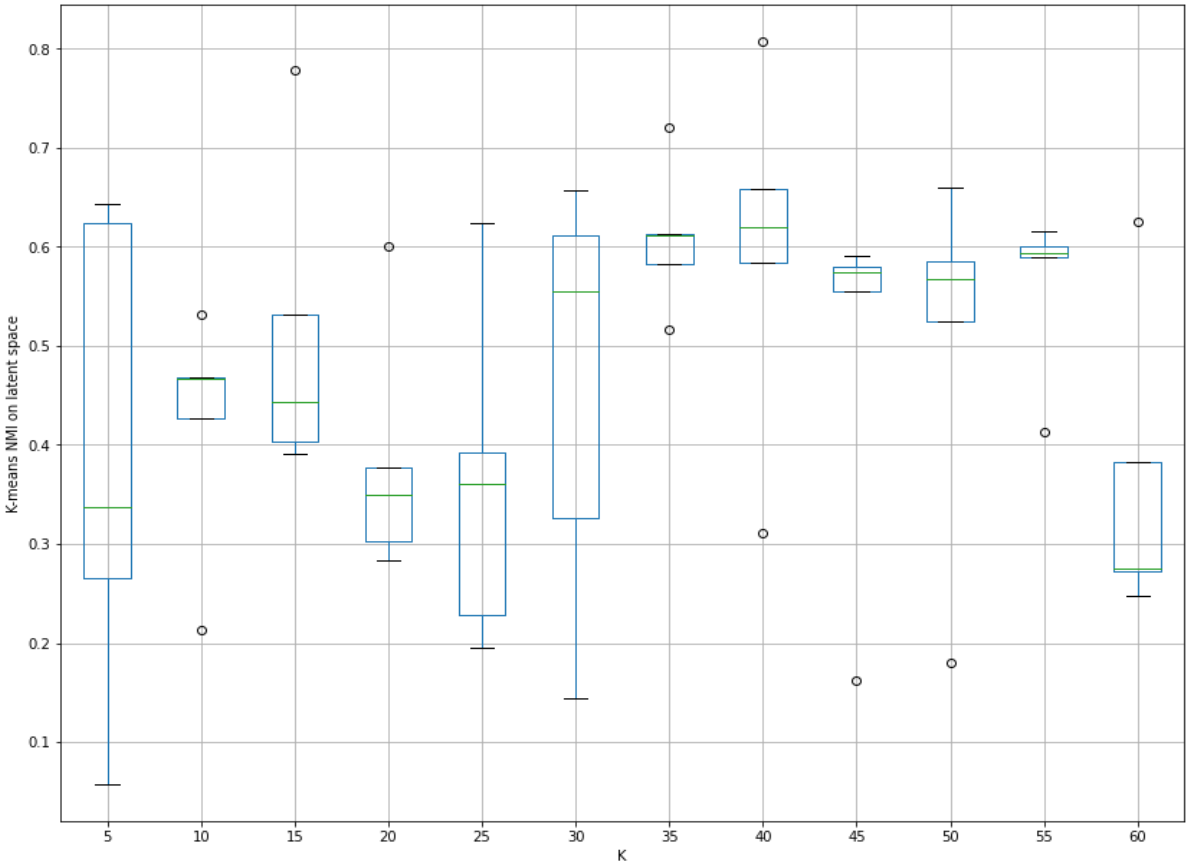
Hidden layer 1 neurons	500
Hidden layer 2 neurons	200
Latent dimension	3
Connections	Fully Connected (in all layers)

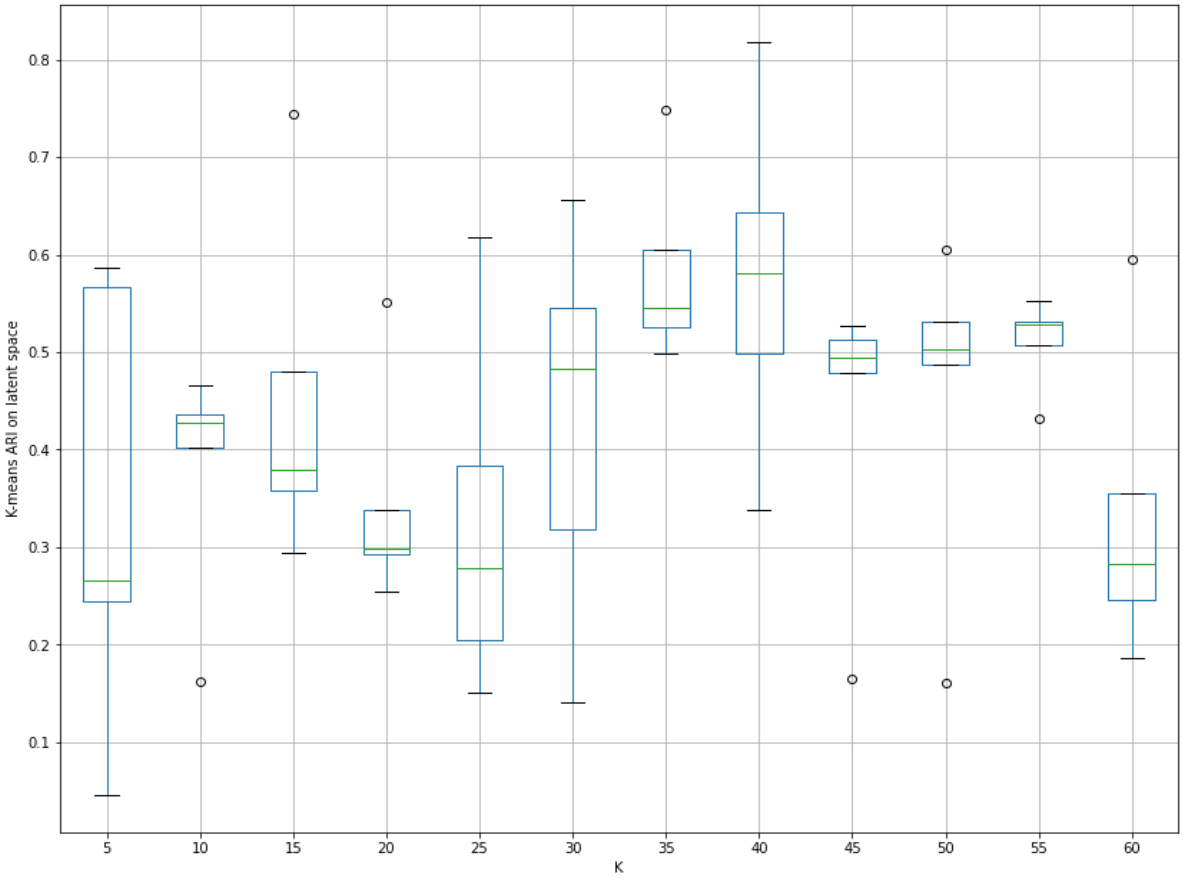
Activation function	Sigmoid (in all layers)
Training epochs	50
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	50
Datapoints	150
Scaling in the initial data space	MinMax
Scaling in the latent space	None

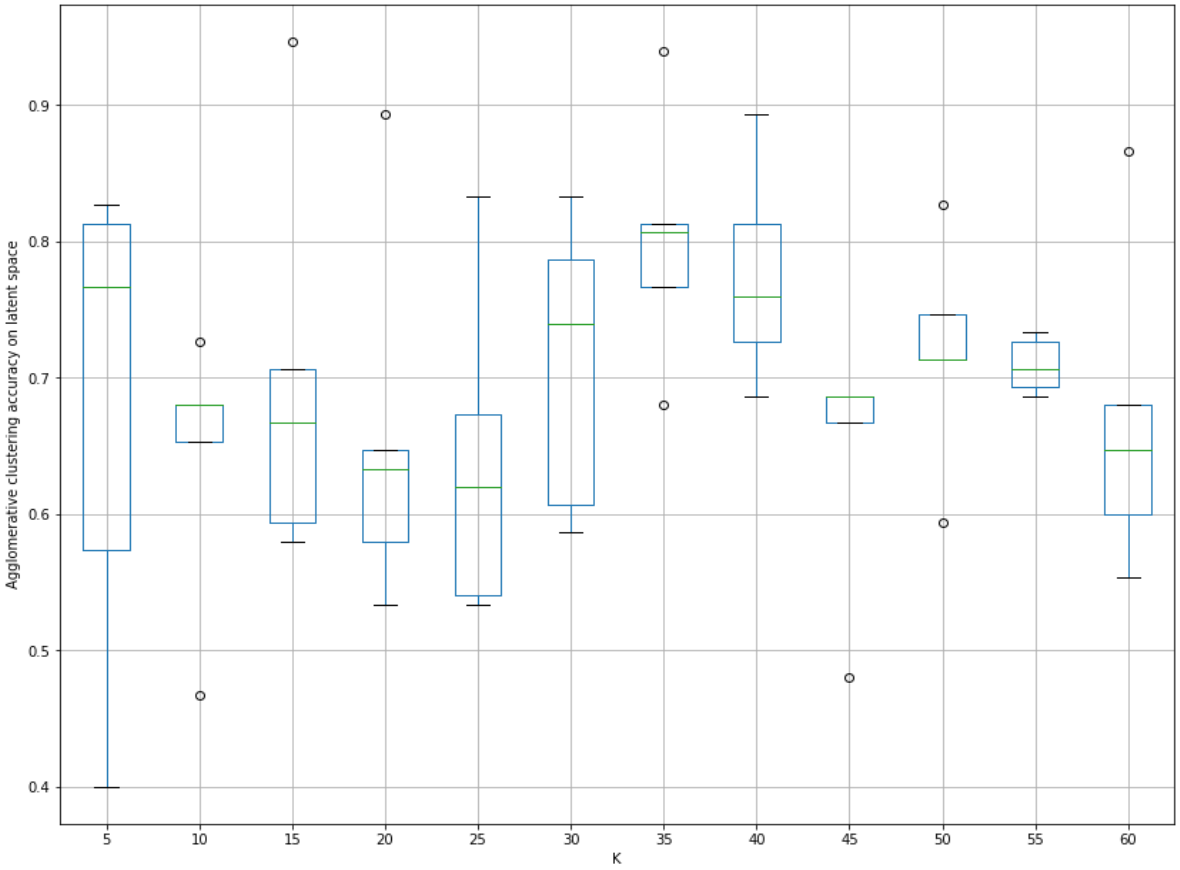
Autoencoder-trained-with-centroids method parameters for iris dataset

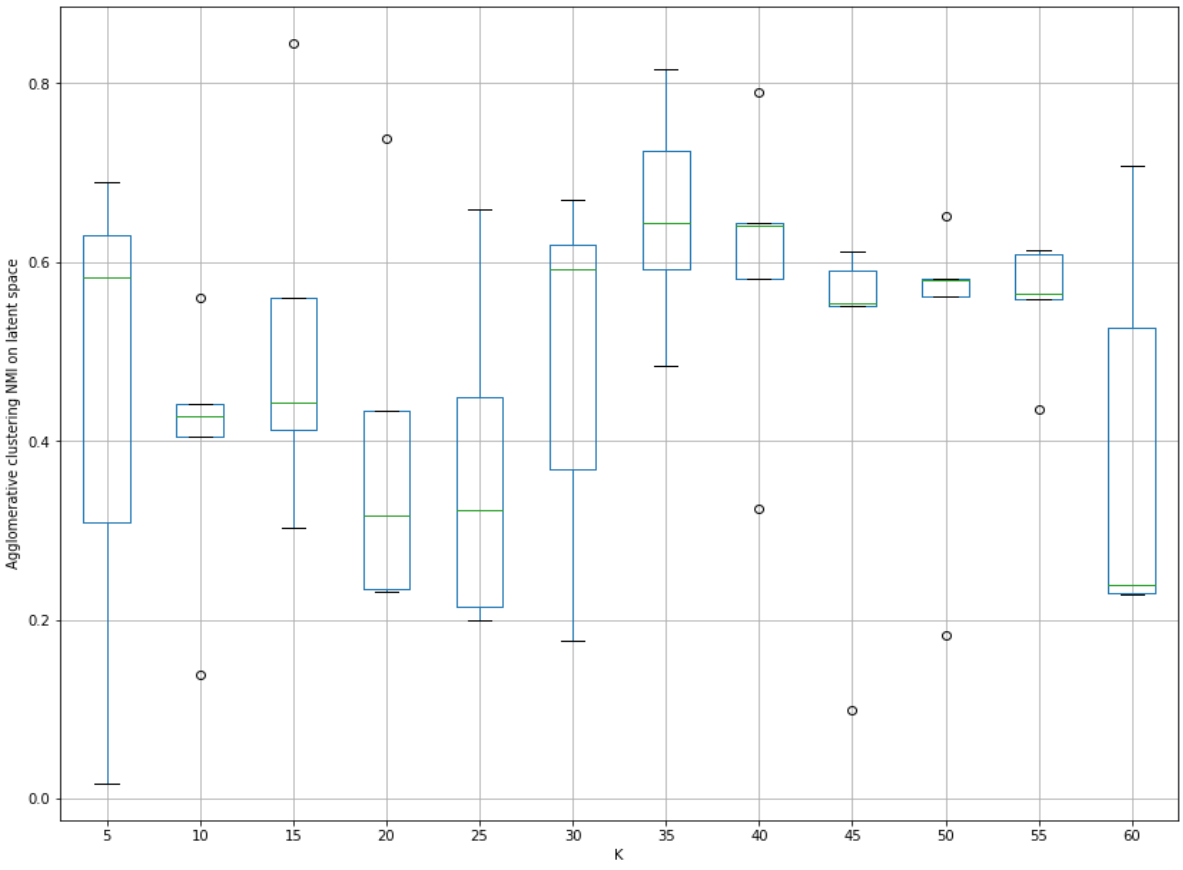
Boxplots:

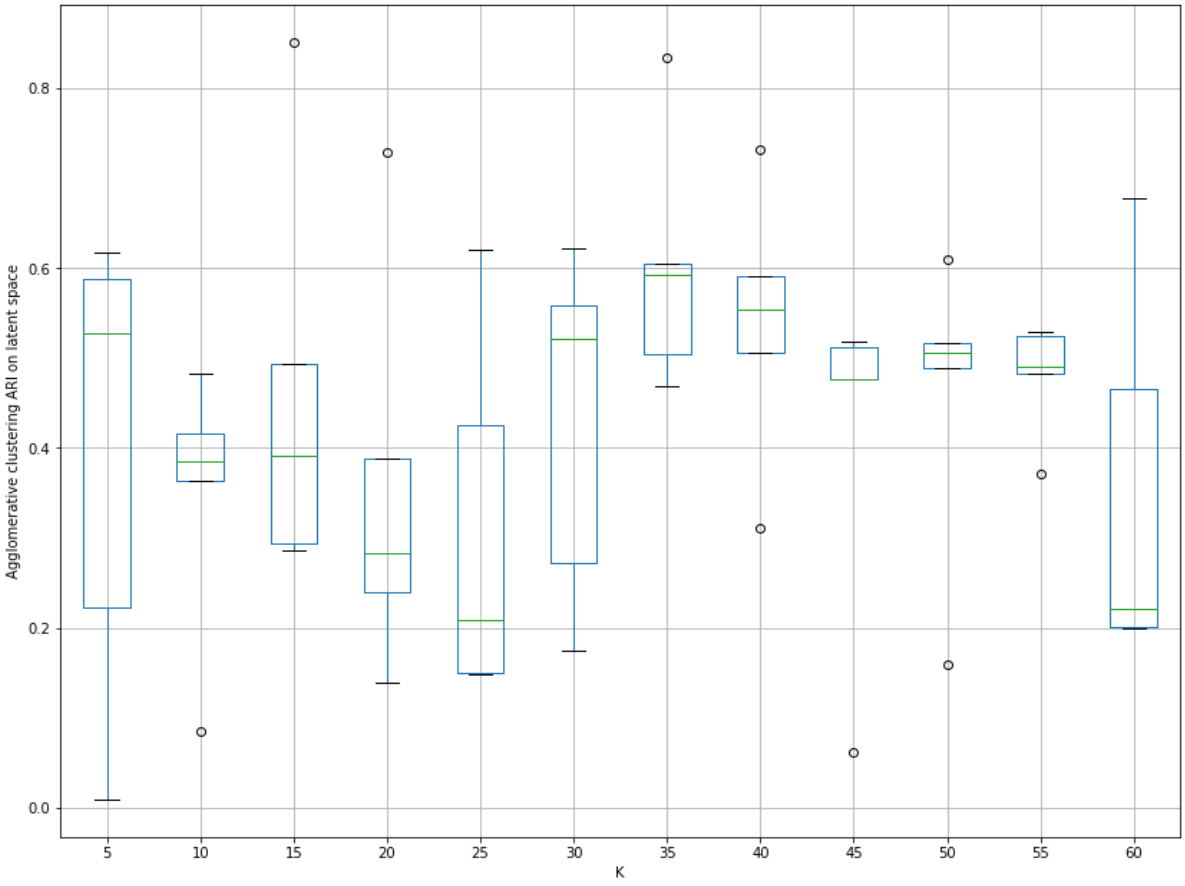


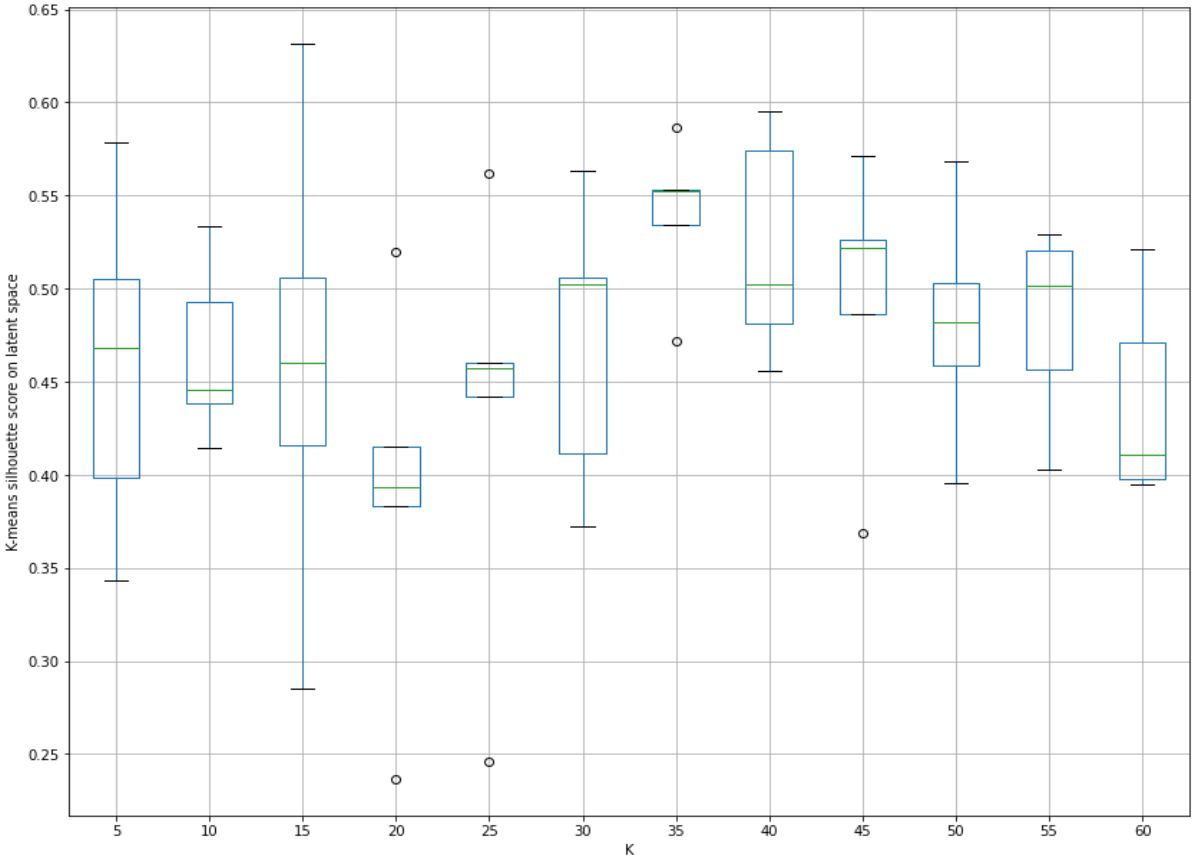

K-means NMI on latent space
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
5
10
15
20
25
30
35
40
45
50
55
60
K


K-means ARI on latent space
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
5
10
15
20
25
30
35
40
45
50
55
60
K

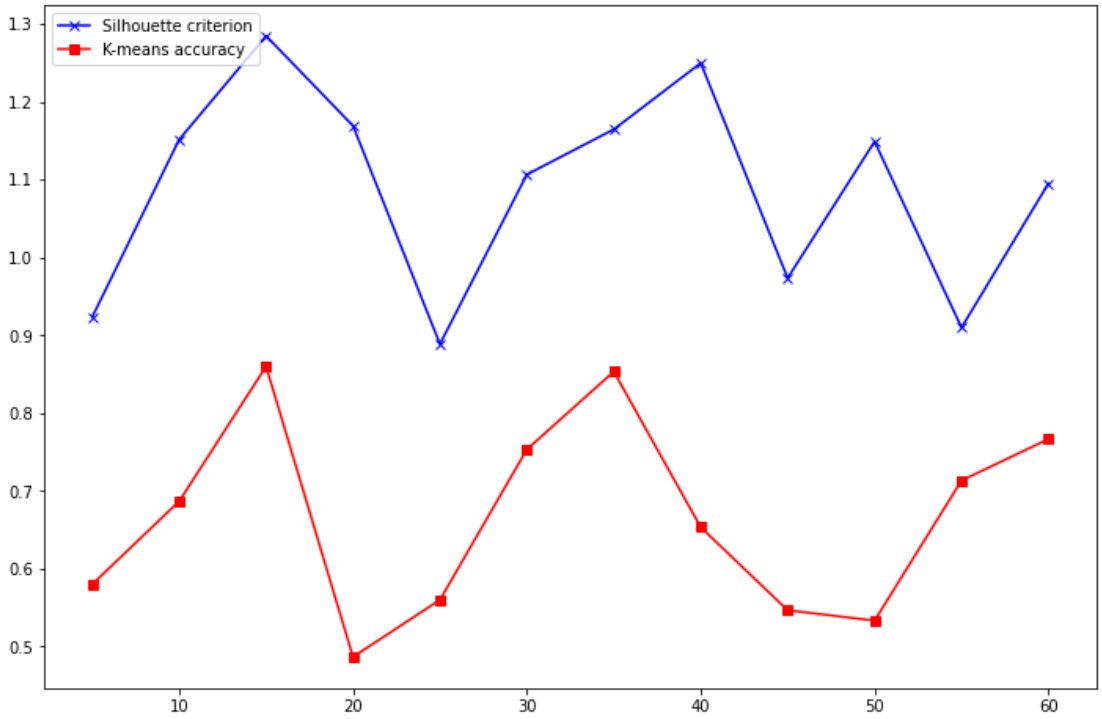

Agglomerative clustering accuracy on latent space
0.9
0.8
0.7
0.6
0.5
0.4
5
10
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering NMI on latent space
0.8
0.6
0.4
0.2
0.0
5
10
15
20
25
30
35
40
45
50
55
60
K


Agglomerative clustering ARI on latent space
0.0
0.2
0.4
0.6
0.8
5
10
15
20
25
30
35
40
45
50
55
60
K


K-means silhouette score on latent space
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65
5
10
15
20
25
30
35
40
45
50
55
60
K

Iris dataset overclustering criterion:



Comments:

Our clustering algorithm using the autoencoder training provides us with worse results than the traditional clustering algorithms on the initial space. Taking the global maximum of the overclustering criterion into consideration, we would make the choice k = 15 for this dataset. The performance with this choice varies a lot and is an average choice, as it doesn't reach the heights of other choices. It definitely isn't one of the worst choices either. However, the overclustering criterion line seems to be ascending when the k-means accuracy line is ascending and descending when the k-means accuracy line is descending. The actual best option, according to the boxplots, for the initial number of clusters is k = 40.

3.3 Boxplots and overclustering criterion for MLP-transformed space method

In this section we will provide only the k-means accuracy boxplots in the MLP-transformed space for every dataset for the report not to get too long. We also look at the overclustering criterion graph, since it is a crucial part of our research. The information about the performance of the best choice of k initial clusters in every metric is provided in an extensive table in the

next section. This should be enough to give us a complete idea of the differences in performance using each clustering method.

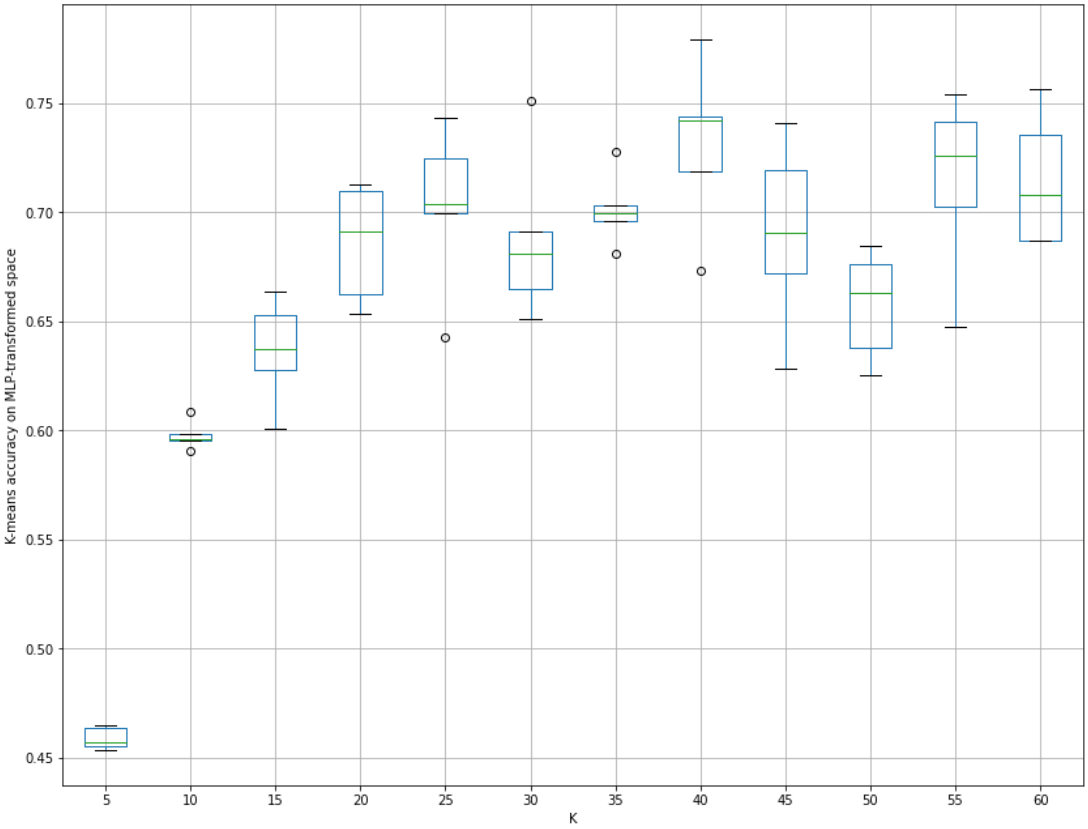
10x_73k dataset:

MLP architecture and data processing details:

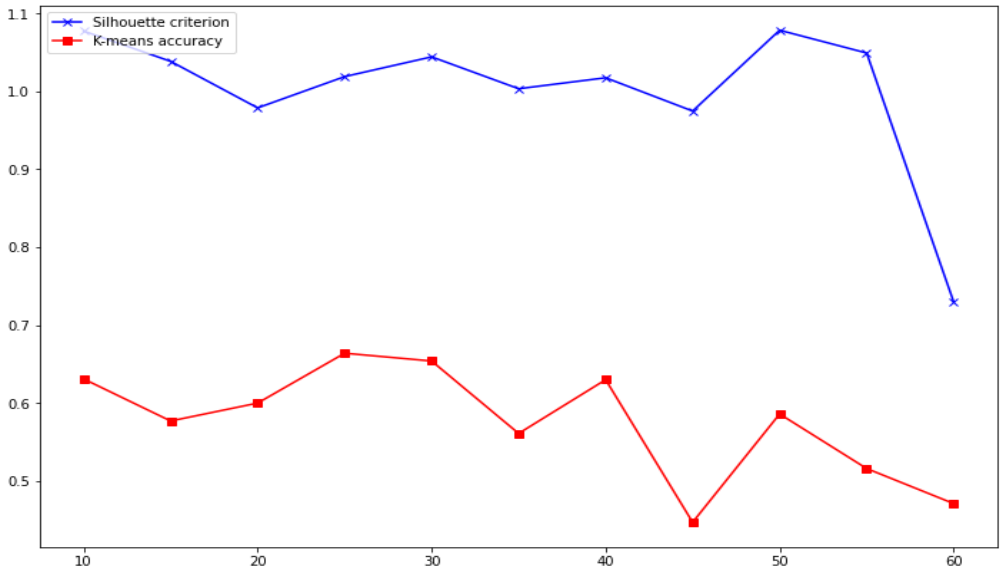
Hidden layer 1 neurons	400
Hidden layer 2 neurons	20
Connections	Fully Connected (in all layers)
Activation function in intermediate layers	Leaky ReLU (negative slope = 0.2)
Output activation function	Softmax
Optimiser	Adam
Learning rate	10^{-3}
Weight decay	0
Loss criterion	Cross-entropy
Training epochs	10
Datapoints	10000
Batch size	64
Scaling in the initial data space	Division by maximum datapoint
Scaling in the latent space	None

MLP-representations method parameters for 10x_73k dataset

Boxplot:



Overclustering criterion:



Comments:

From the boxplots we have collected (despite only putting the k-means accuracy boxplot on the) we come to the conclusion that the best choice of k for the initial clusters to create the

MLP-transformed space is k = 30, which is the best performing option for the k-means statistics and good for the agglomerative clustering part as well. The overclustering criterion has a high score for this option, but it chooses k = 50, which is also good. Therefore, it performs well in this situation, giving us a good image in advance.

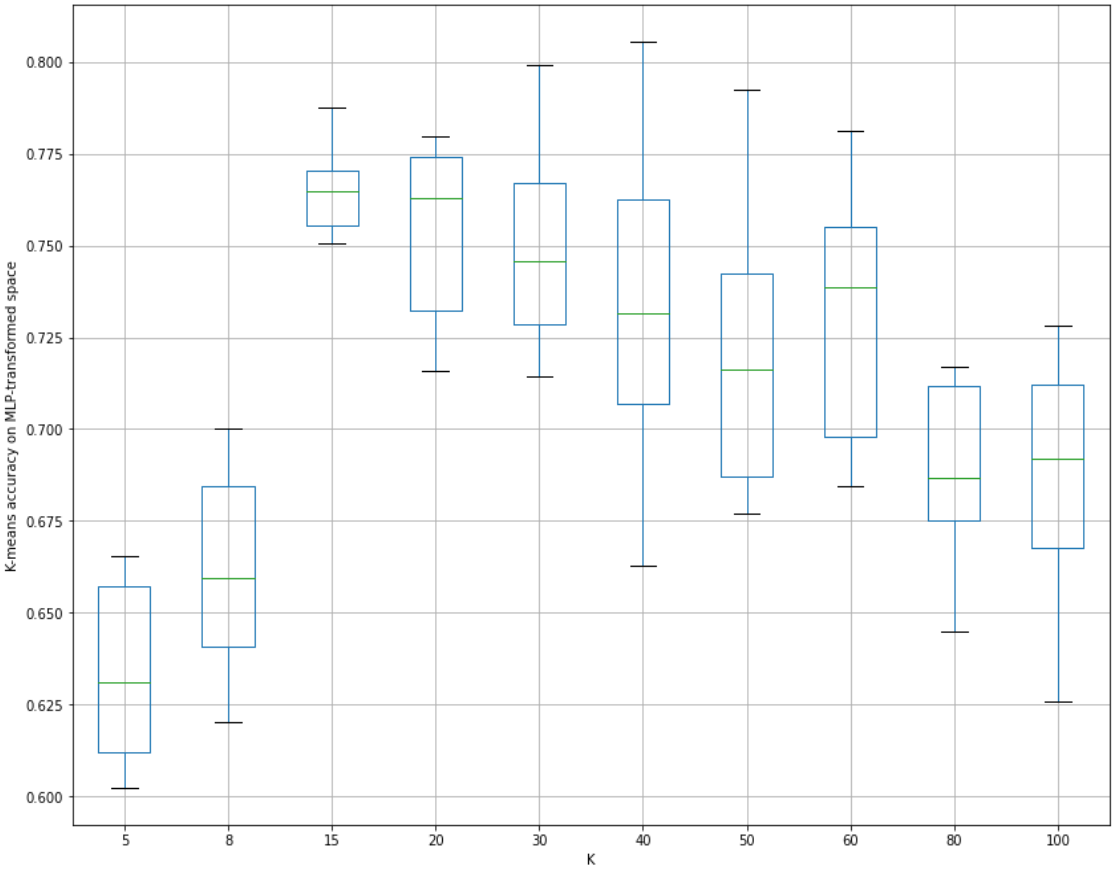
Pendigits dataset:

MLP architecture and data processing details:

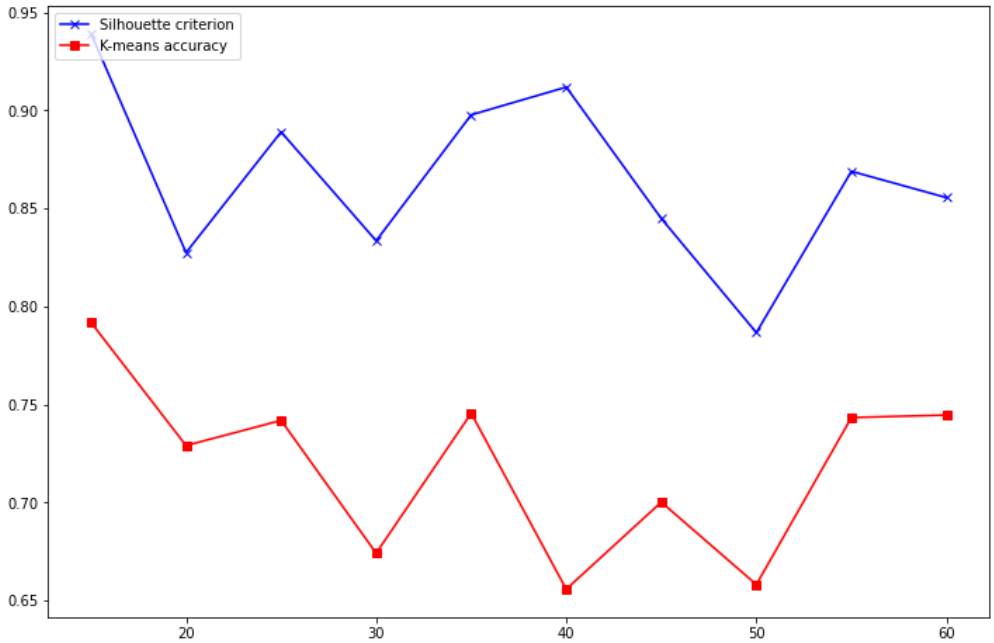
Hidden layer 1 neurons	20
Hidden layer 2 neurons	50
Connections	Fully Connected (in all layers)
Activation function in intermediate layers	Leaky ReLU (negative slope = 0.2)
Output activation function	Softmax
Optimiser	Adam
Learning rate	10^{-3}
Weight decay	0
Loss criterion	Cross-entropy
Training epochs	10
Datapoints	7494
Batch size	128
Scaling in the initial data space	MinMax
Scaling in the latent space	None

MLP-representations method parameters for pendigits dataset

Boxplot:



Overclustering criterion:



Comments:

From the boxplots we have collected (despite only putting the k-means accuracy boxplot on the report) we come to the conclusion that the best choice of k for the initial clusters to create the MLP-transformed space is k = 15, which is the best performing option for all the stats. The overclustering criterion has a high score for values close to this option. Therefore, it performs well in this situation, giving us a good image in advance.

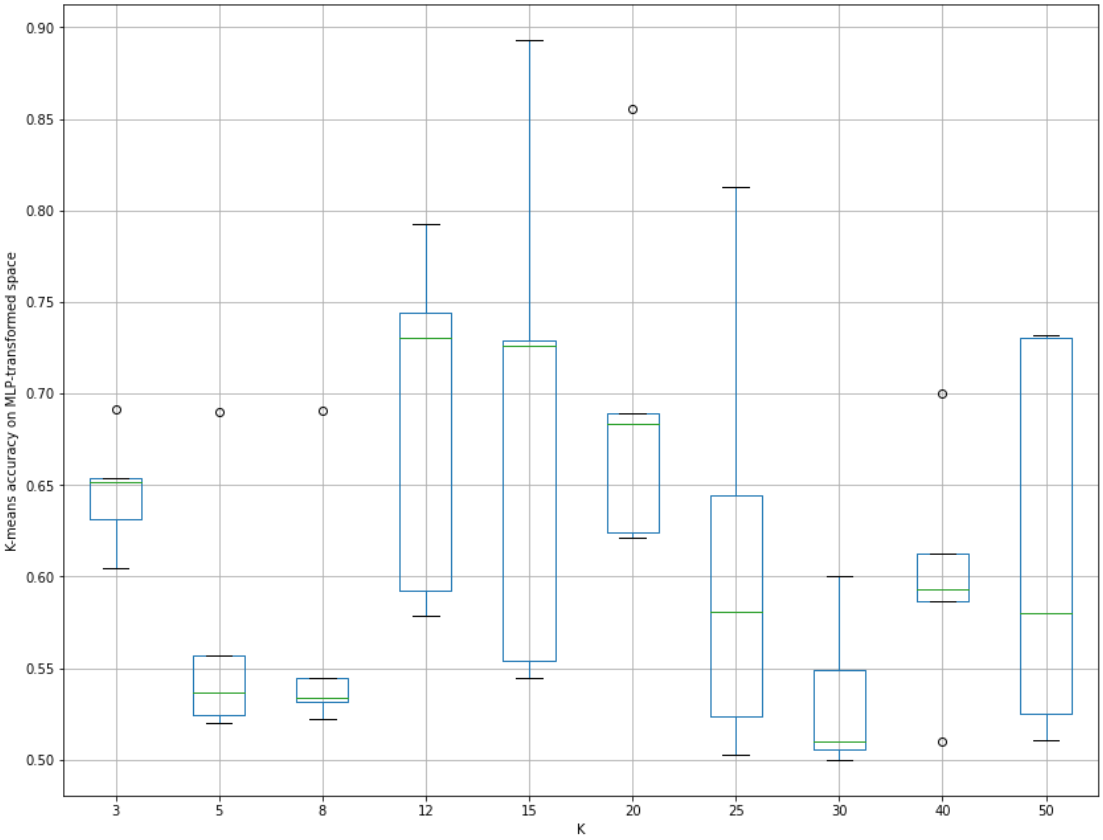
Gaussian rings dataset:

MLP architecture and data processing details:

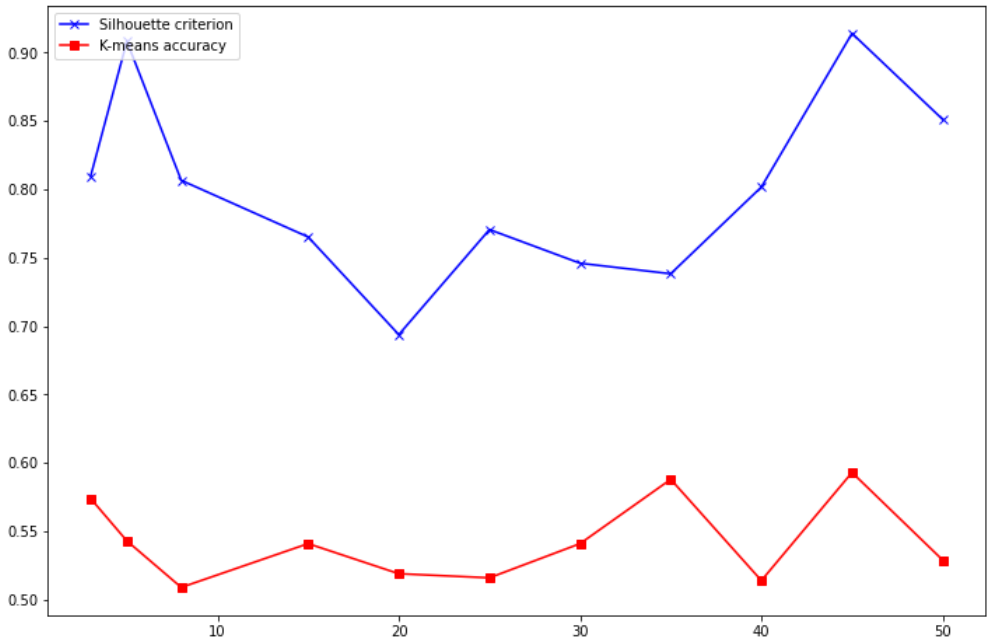
Hidden layer 1 neurons	20
Hidden layer 2 neurons	10
Connections	Fully Connected (in all layers)
Activation function in intermediate layers	Leaky ReLU (negative slope = 0.2)
Output activation function	Softmax
Optimiser	Adam
Learning rate	10^{-3}
Weight decay	0
Loss criterion	Cross-entropy
Training epochs	10
Datapoints	1000
Batch size	50
Scaling in the initial data space	MinMax
Scaling in the latent space	None

MLP-representations method parameters for gaussian rings dataset

Boxplot:



Overclustering criterion:



Comments:

Taking all boxplots into consideration, it appears that k = 12 is the best option in this dataset, yet it offers a high standard deviation. The overclustering criterion seems to suggest a much higher option, therefore it doesn't quite work well in this case.

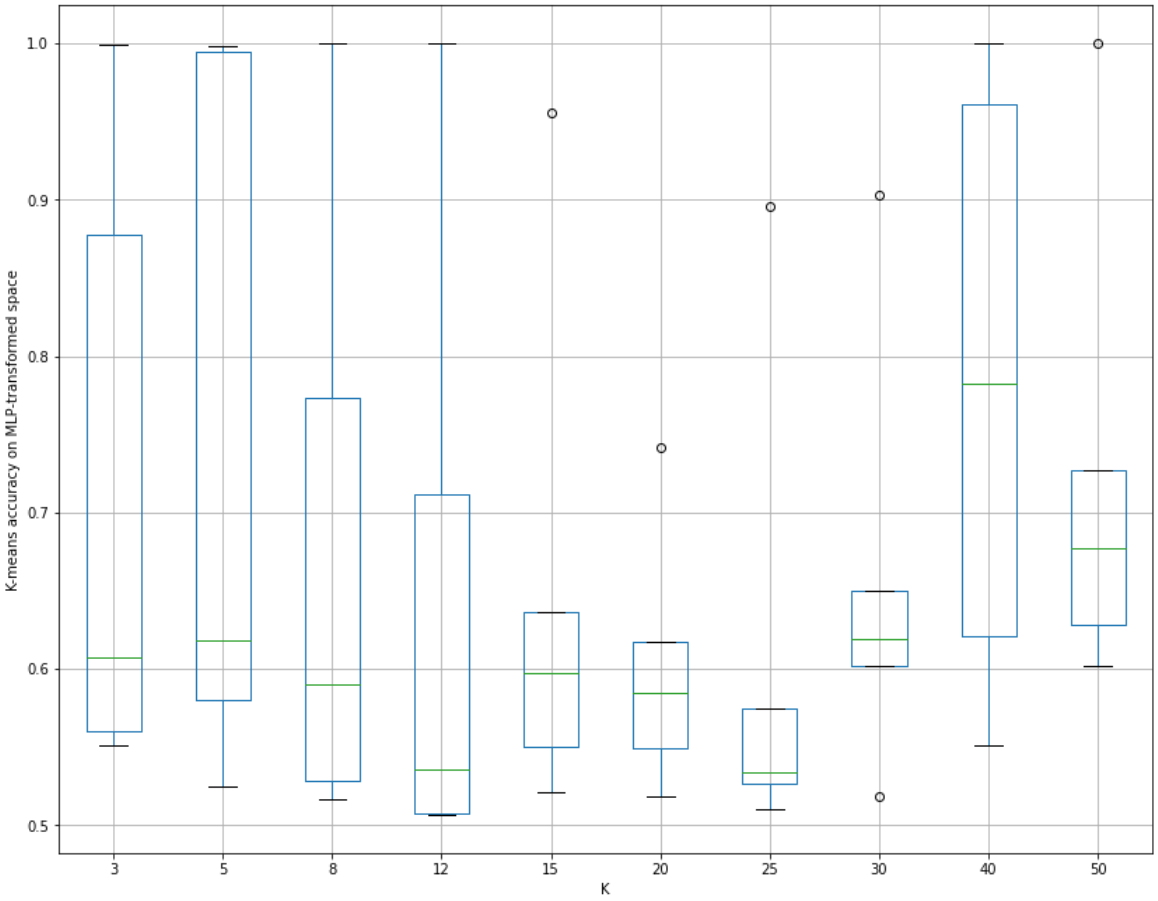
Gaussian squeezed blobs dataset:

MLP architecture and data processing details:

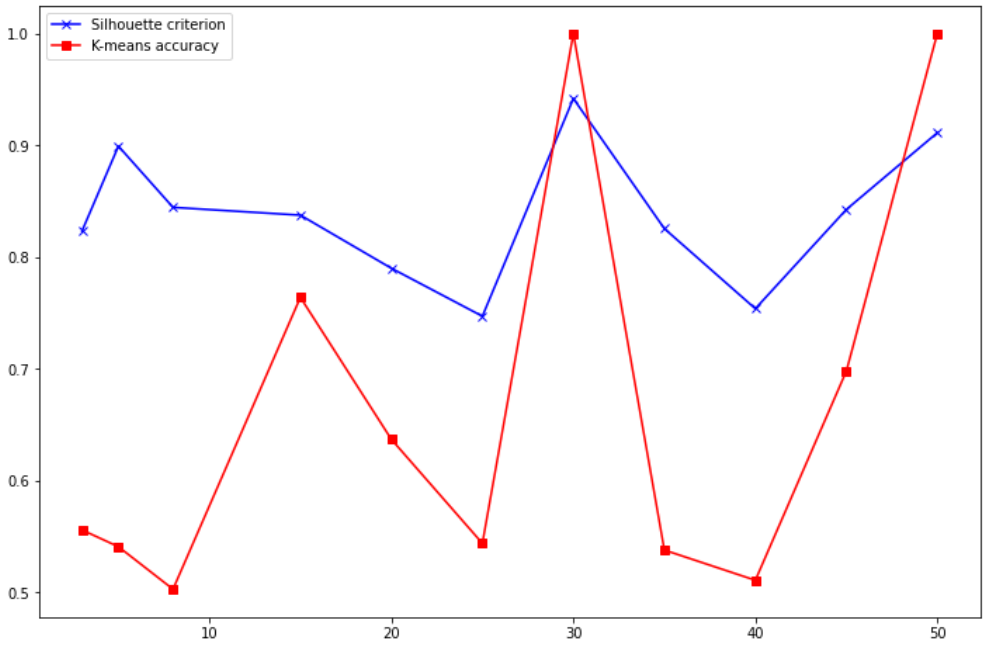
Hidden layer 1 neurons	20
Hidden layer 2 neurons	10
Connections	Fully Connected (in all layers)
Activation function in intermediate layers	Leaky ReLU (negative slope = 0.2)
Output activation function	Softmax
Optimiser	Adam
Learning rate	10^{-3}
Weight decay	0
Loss criterion	Cross-entropy
Training epochs	10
Datapoints	1000
Batch size	50
Scaling in the initial data space	None
Scaling in the latent space	None

MLP-representations method parameters for gaussian squeezed blobs dataset

Boxplot:



Overclustering criterion:



Comments:

Taking all boxplots into consideration, it appears that k = 20 is the best option in this dataset when it comes to the agglomerative clustering stats and it also works well for k-means clustering, despite not being the best at it. The overclustering criterion seems to suggest two much higher options, therefore it doesn't quite work well in this case. The overclustering criterion has many more higher options for other choices. It would actually pick k = 30, which is close to 20, but 20 doesn't achieve such a high score.

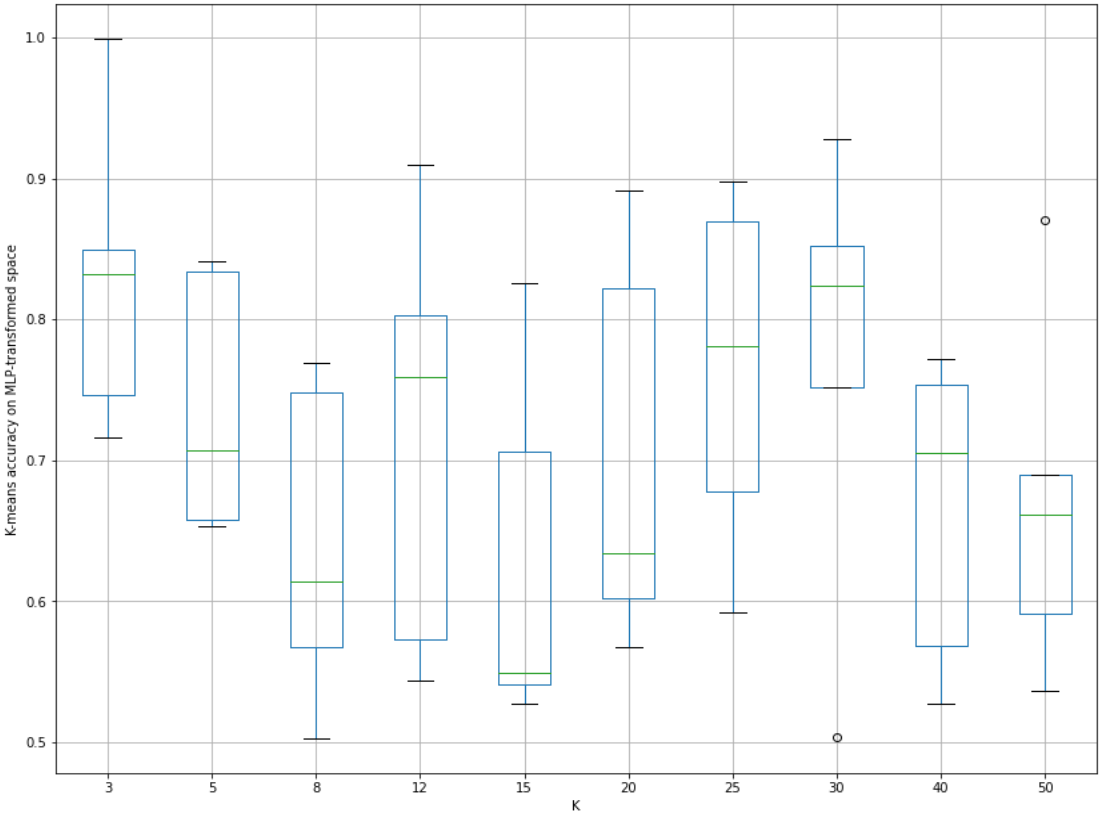
Moons dataset:

MLP architecture and data processing details:

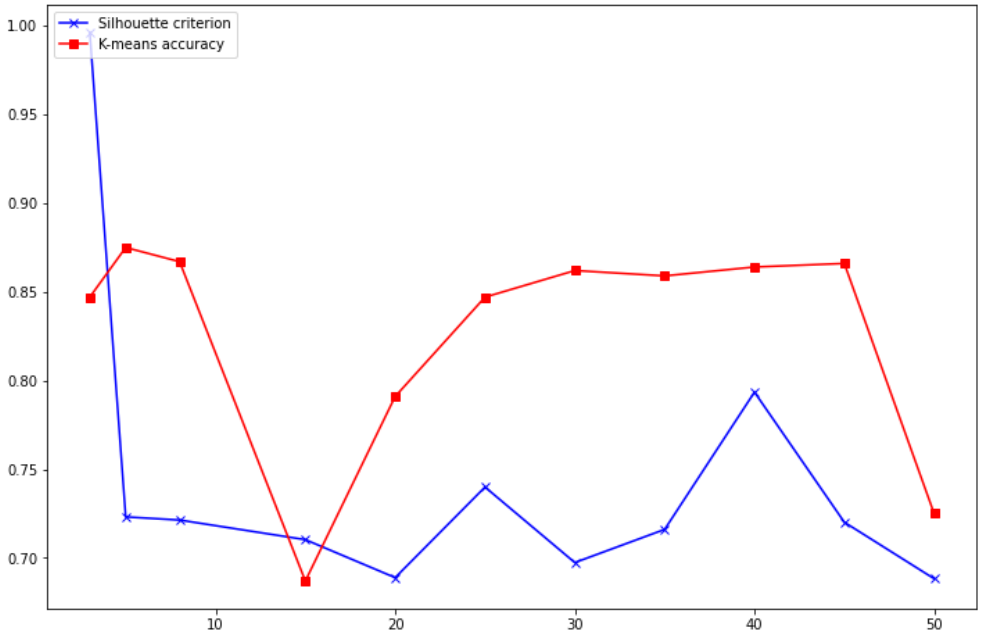
Hidden layer 1 neurons	20
Hidden layer 2 neurons	10
Connections	Fully Connected (in all layers)
Activation function in intermediate layers	Leaky ReLU (negative slope = 0.2)
Output activation function	Softmax
Optimiser	Adam
Learning rate	10^{-3}
Weight decay	0
Loss criterion	Cross-entropy
Training epochs	10
Datapoints	1000
Batch size	50
Scaling in the initial data space	MinMax
Scaling in the latent space	None

MLP-representations method parameters for moons dataset

Boxplot:



Overclustering criterion:



Comments:

In this dataset, k = 3 performs really well and the overclustering criterion does a great job in picking it. Its second choice is k = 40, which is also a relatively good option in relation to others.

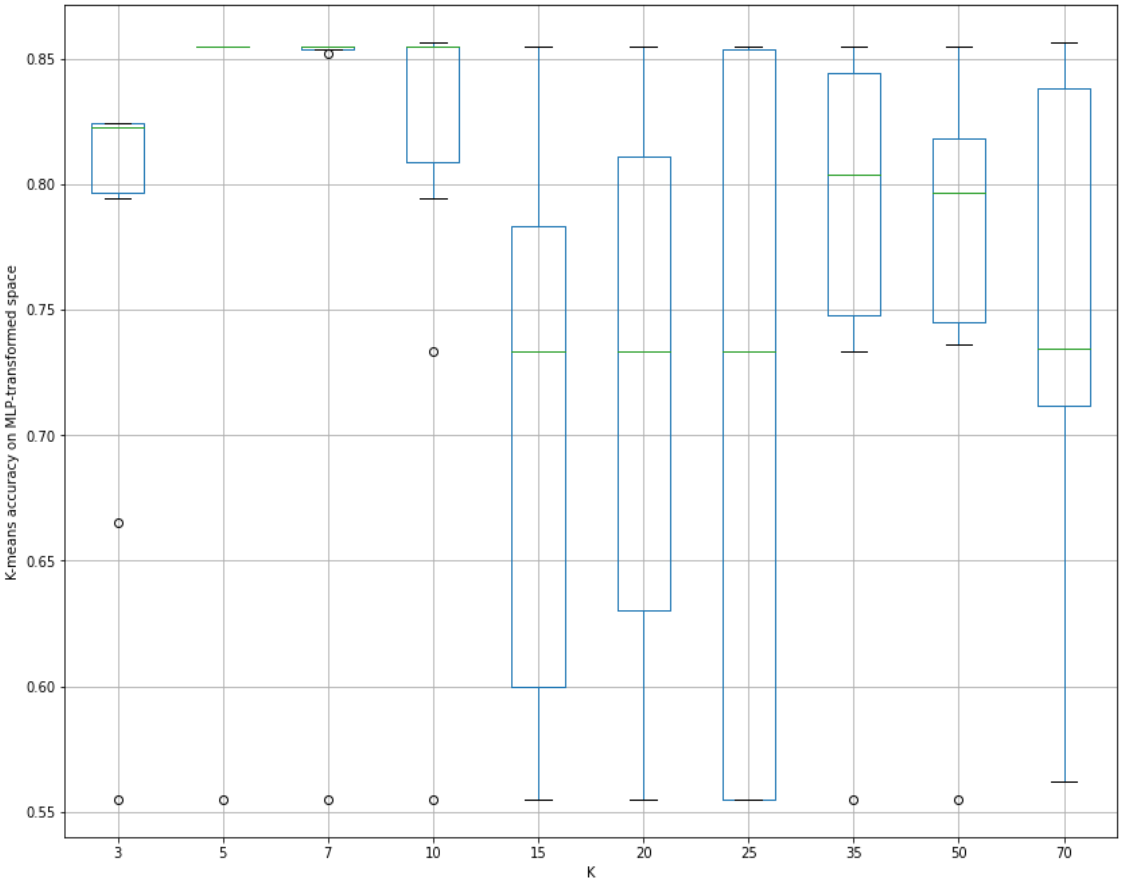
Australian dataset:

MLP architecture and data processing details:

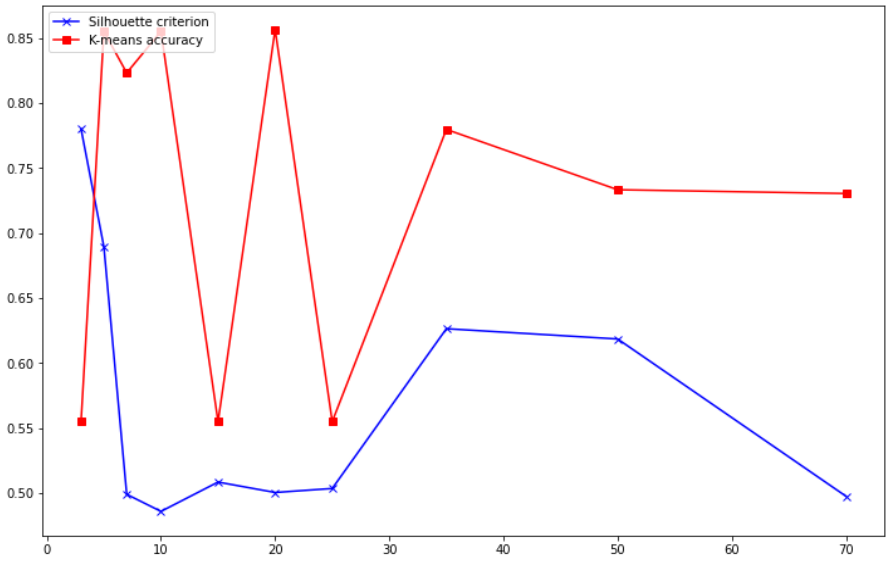
Hidden layer 1 neurons	50
Hidden layer 2 neurons	60
Connections	Fully Connected (in all layers)
Activation function in intermediate layers	Leaky ReLU (negative slope = 0.2)
Output activation function	Softmax
Optimiser	Adam
Learning rate	10^{-3}
Weight decay	0
Loss criterion	Cross-entropy
Training epochs	20
Datapoints	690
Batch size	69
Scaling in the initial data space	MinMax
Scaling in the latent space	Standard Scaler

MLP-representations method parameters for australian dataset

Boxplot:



Overclustering criterion:



Comments:

The performance is not improved but the overclustering criterion makes close to the best choice for the initial number of clusters, as it picks the number of 3 clusters, which follows the performance of the 5 clusters closely.

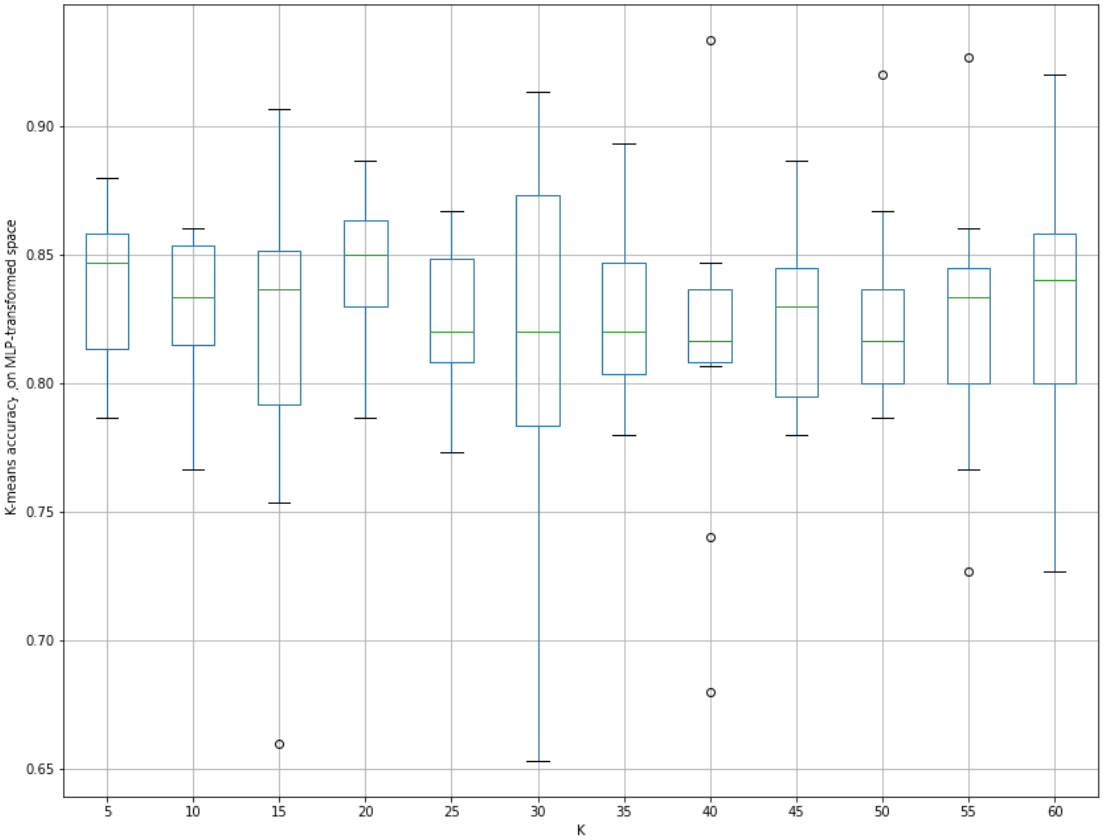
Iris dataset:

MLP architecture and data processing details:

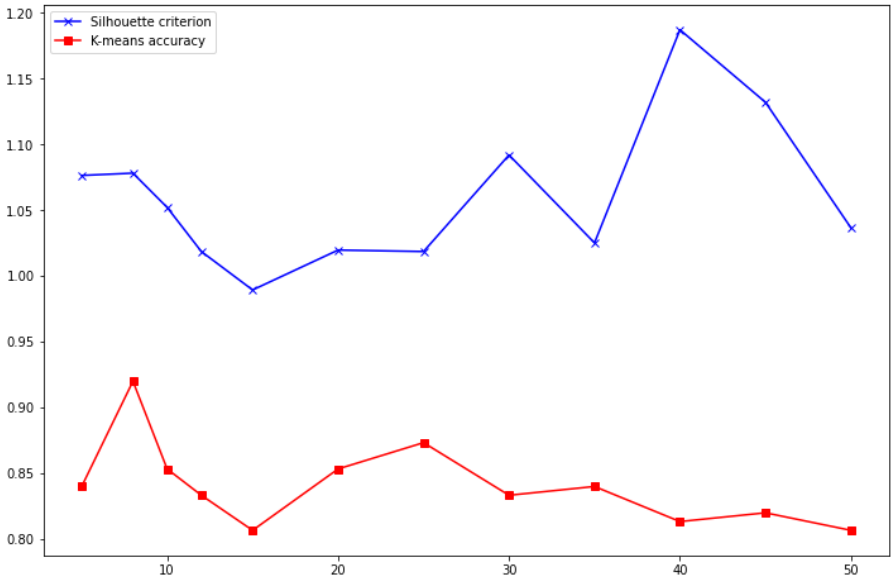
Hidden layer 1 neurons	100
Hidden layer 2 neurons	100
Connections	Fully Connected (in all layers)
Activation function in intermediate layers	Leaky ReLU (negative slope = 0.2)
Output activation function	Softmax
Optimiser	Adam
Learning rate	10^{-3}
Weight decay	0
Loss criterion	Cross-entropy
Training epochs	10
Datapoints	150
Batch size	50
Scaling in the initial data space	MinMax
Scaling in the latent space	Standard Scaler

MLP-representations method parameters for iris dataset

Boxplot:



Overclustering criterion:



Comments:

In this dataset, using the MLP clustering method we found out that option k = 20 offer the greatest results in terms of k-means clustering metrics and it also does well for agglomerative clustering's purposes. Generally, though, we have not improved from the initial space clustering. The overclustering criterion picks option 40, which is very far and doesn't perform as well.

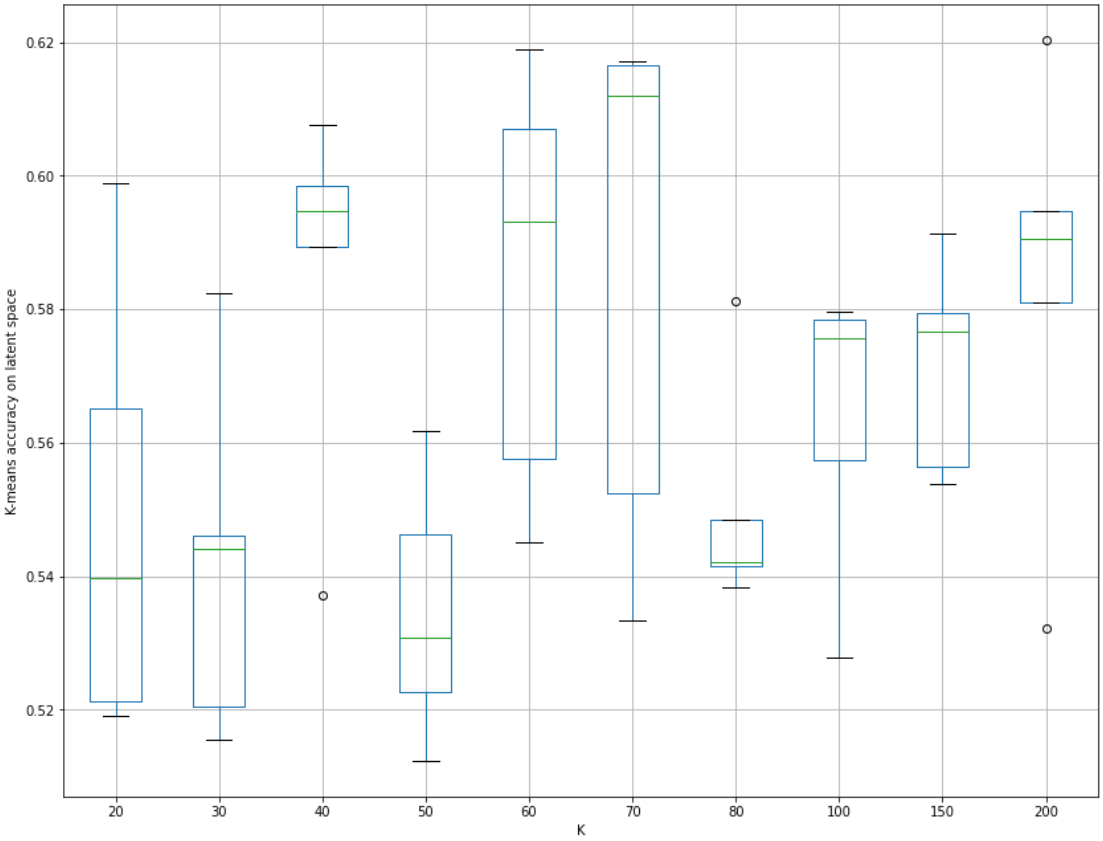
MNIST dataset:

MLP architecture and data processing details:

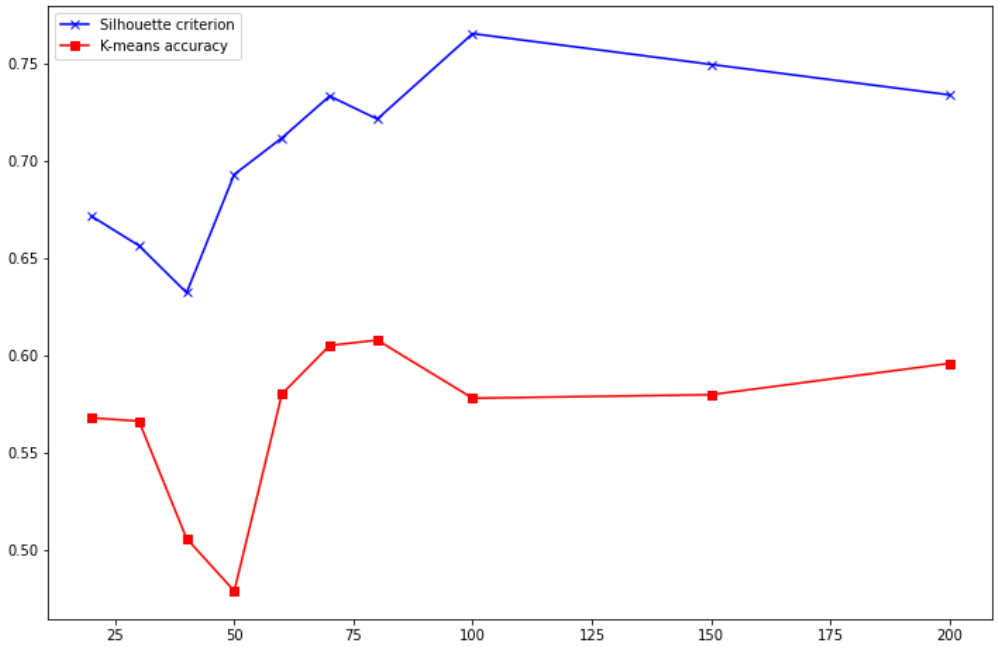
Hidden layer 1 neurons	40
Hidden layer 2 neurons	20
Connections	Fully Connected (in all layers)
Activation function in intermediate layers	Sigmoid
Output activation function	Softmax
Optimiser	Adam
Learning rate	10^{-3}
Weight decay	0
Loss criterion	Cross-entropy
Training epochs	10
Datapoints	10000 (1000 for each digit)
Batch size	50
Scaling in the initial data space	None
Scaling in the latent space	None

MLP-representations method parameters for MNIST dataset

Boxplot:



Overclustering criterion:



Comments:

Using MLP representations on reshaped MNIST images (to vectors) is not the most effective way to improve clustering performance in this situation, yet we still manage little improvements with some choices for the initial number of clusters. The overclustering criterion chooses the option k = 100, which gives us an average performance among its competitor choices.

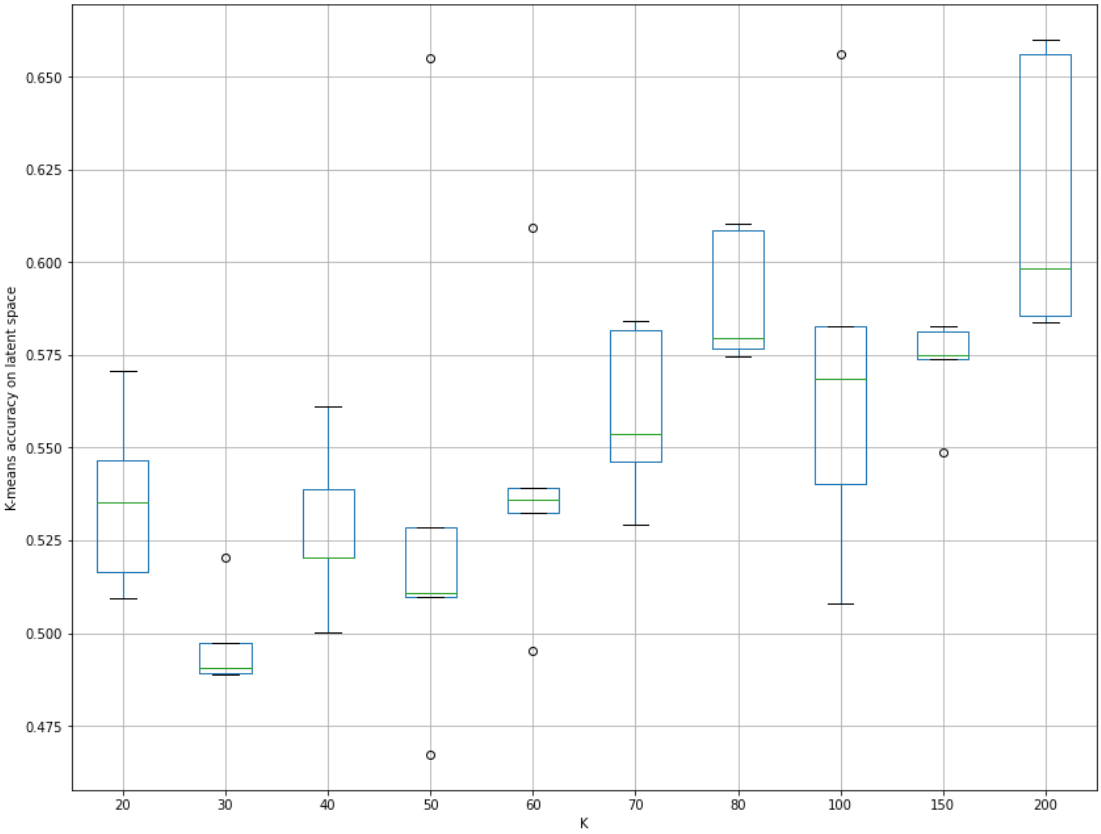
Fashion-MNIST dataset:

MLP architecture and data processing details:

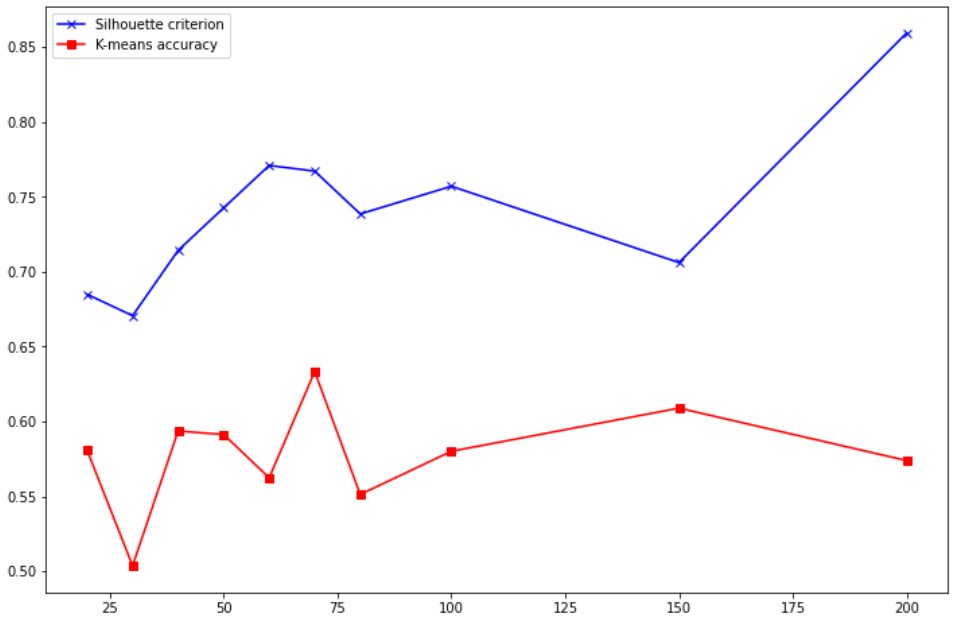
Hidden layer 1 neurons	40
Hidden layer 2 neurons	20
Connections	Fully Connected (in all layers)
Activation function in intermediate layers	Sigmoid
Output activation function	Softmax
Optimiser	Adam
Learning rate	10^{-3}
Weight decay	0
Loss criterion	Cross-entropy
Training epochs	10
Datapoints	10000 (1000 for each clothing category)
Batch size	50
Scaling in the initial data space	None
Scaling in the latent space	None

MLP-representations method parameters for Fashion-MNIST dataset

Boxplot:



Overclustering criterion:



Comments:

Using MLP representations on reshaped Fashion-MNIST images (to vectors) doesn't improve clustering performance. The best performing option from the ones we tried is k = 200, which is also the one chosen from the overclustering criterion, so the criterion works well in this case.

3.4 Boxplots and overclustering criterion for autoencoder-trained-without centroids method

In this section we will provide only the k-means accuracy boxplots in the autoencoder-without-centroids-transformed space for every dataset for the report not to get too long. We also look at the overclustering criterion graph, since it is a crucial part of our research. The information about the performance of the best choice of k initial clusters in every metric is provided in an extensive table in the next section. This should be enough to give us a complete idea of the differences in performance using each clustering method. It will become obvious that the concept of training the autoencoder using the centroids and labels of the initial k-means run, as we described in the methodology section, improves the results massively.

10x_73k dataset:

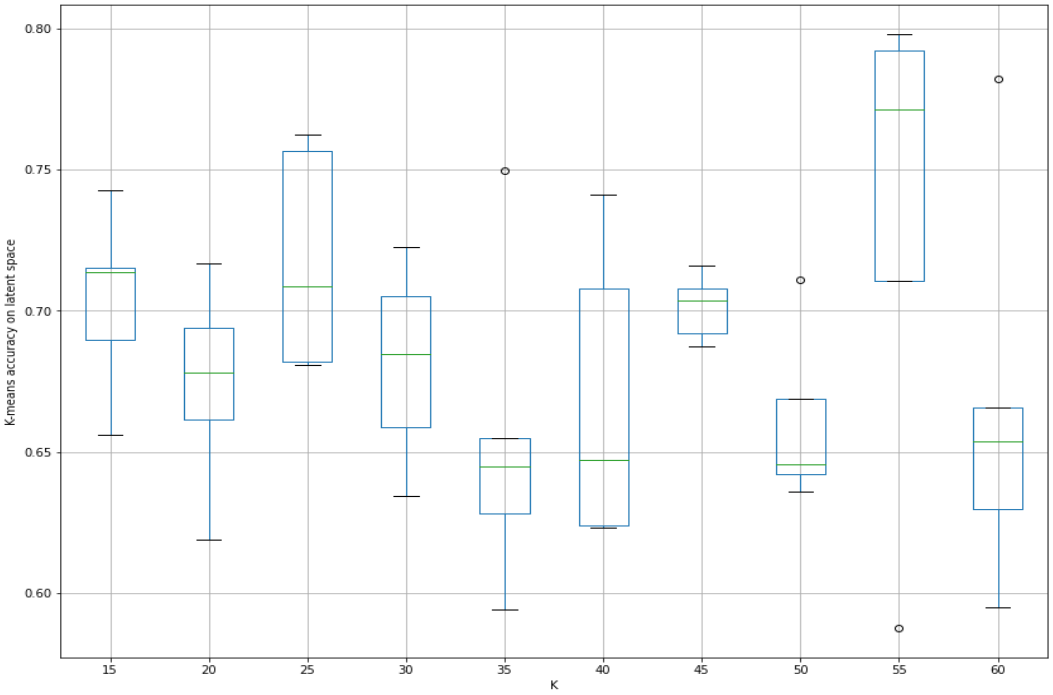
Autoencoder and data processing details:

Hidden layer 1 neurons	800
Hidden layer 2 neurons	300
Latent dimension	8
Connections	Fully Connected (in all layers)
Activation function	Sigmoid (in all layers)
Training epochs	50
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	64

Datapoints	10000
Scaling in the initial data space	Divide by max datapoint
Scaling in the latent space	None

Autoencoder-trained-without-centroids method parameters for 10x_73k dataset

Boxplot:



Comments:

The difference in performance with the method of training the autoencoder using the centroids is significant and is portrayed in the boxplots. By looking at the other graphs other than the k-means accuracy graph which we have given above, we have come to the conclusion that k = 55 is the best option for the initial number of clusters when it comes to the performance of all clustering measures in the latent space.

Pendigits dataset:

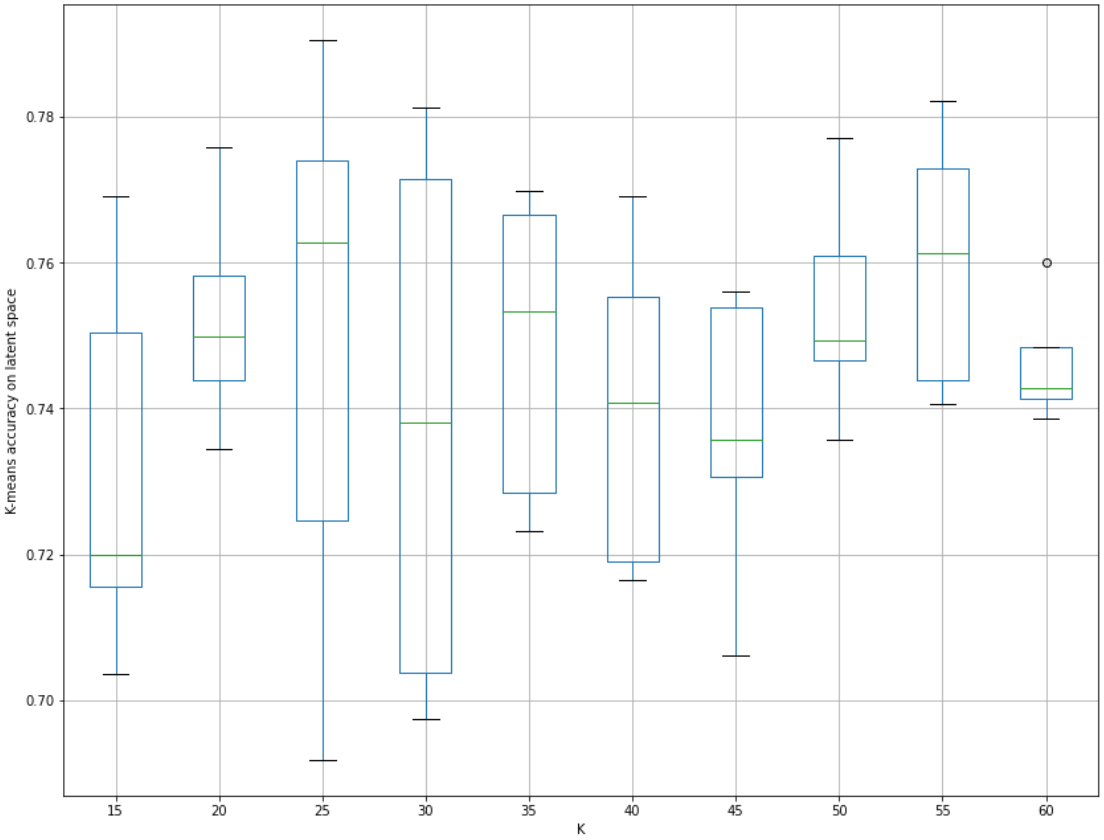
Autoencoder and data processing details:

Hidden layer 1 neurons	800
Hidden layer 2 neurons	300

Latent dimension	8
Connections	Fully Connected (in all layers)
Activation function	Sigmoid (in all layers)
Training epochs	50
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	128
Datapoints	7494
Scaling in the initial data space	Minmax
Scaling in the latent space	None

Autoencoder-trained-without-centroids method parameters for pendigits dataset

Boxplots:



Comments:

The difference in performance with the method of training the autoencoder using the centroids is significant and is portrayed in the boxplots. By looking at the other graphs other than the k-means accuracy graph which we have given above, we have come to the conclusion that k = 55 is the best option for the initial number of clusters when it comes to the performance of most clustering measures in the latent space.

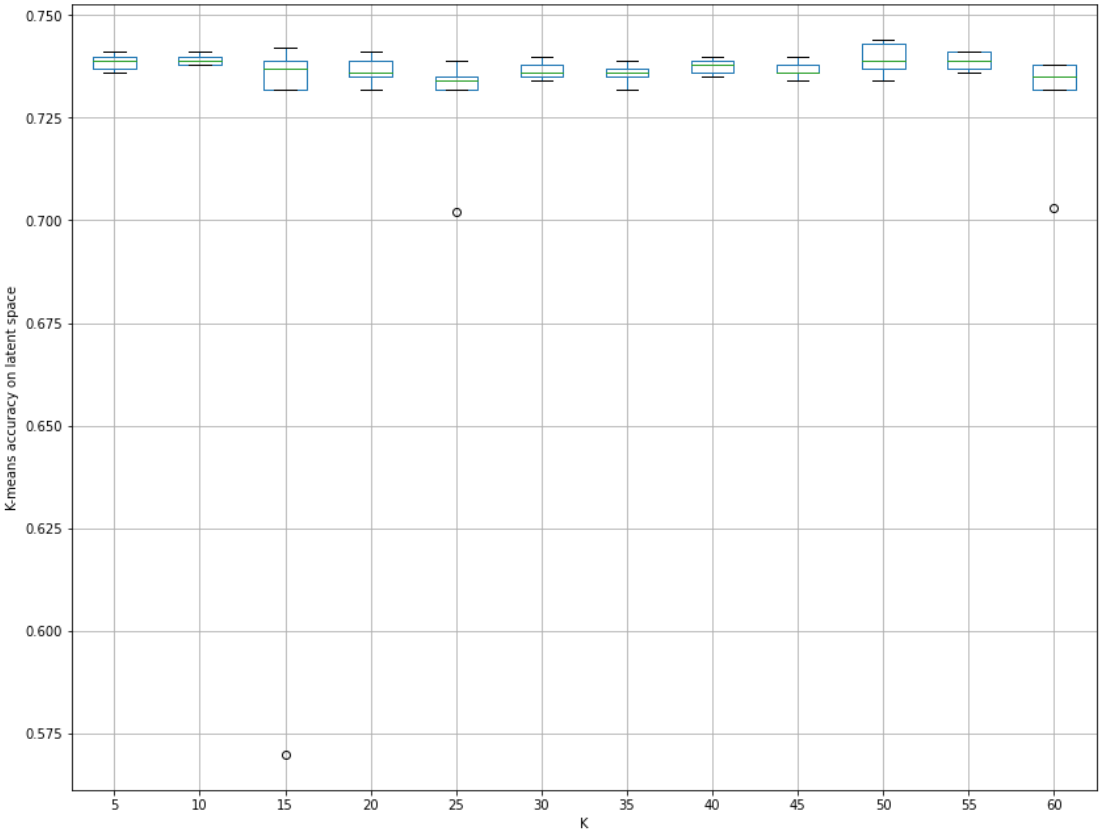
Gaussian rings dataset:

Autoencoder and data processing details:

Hidden layer 1 neurons	800
Hidden layer 2 neurons	300
Latent dimension	1
Connections	Fully Connected (in all layers)
Activation function	ReLU (in all layers)
Training epochs	50
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	100
Datapoints	1000
Scaling in the initial data space	Minmax
Scaling in the latent space	None

Autoencoder-trained-without-centroids method parameters for gaussian rings dataset

Boxplots:



Comments:

The choice of number of clusters doesn't make much difference in this dataset, as all choices offer close to the same improvement from the initial space. However, option k = 50 offers slightly better results in most metrics. What is worth noting is the very small standard deviation that comes from all choices.

Squeezed gaussian blobs dataset:

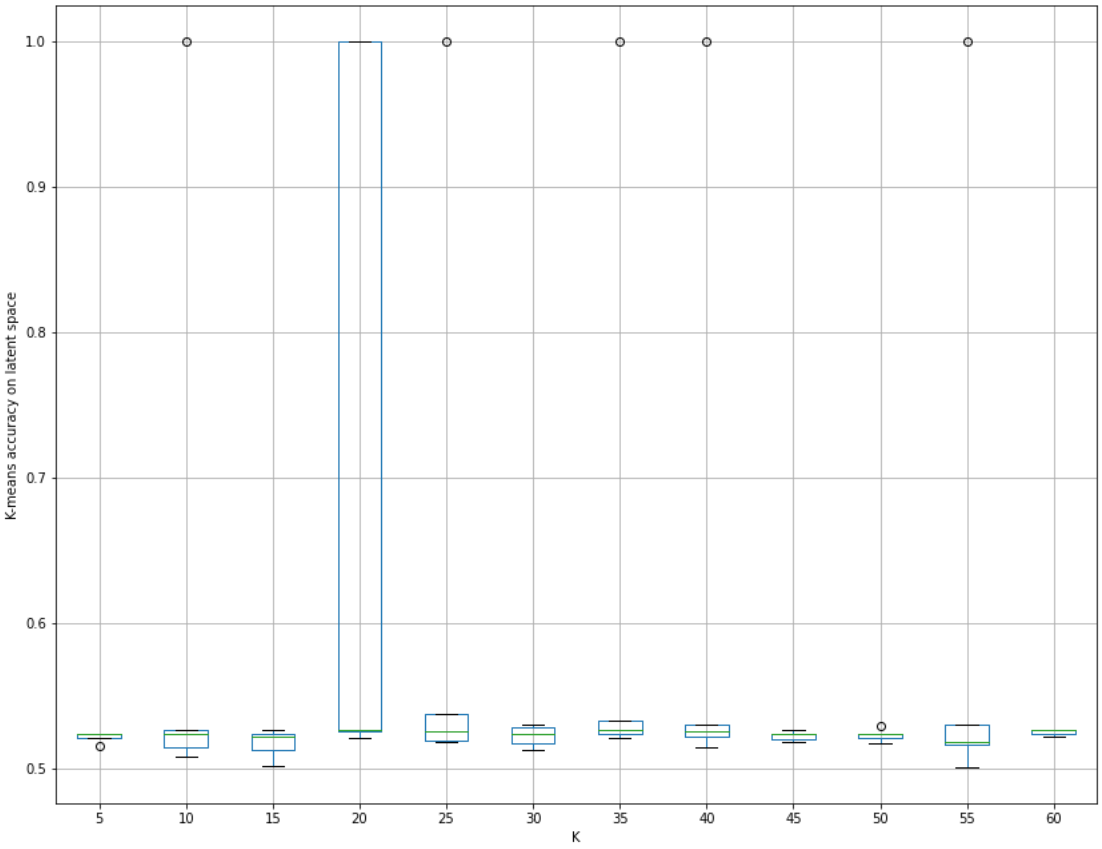
Autoencoder and data processing details:

Hidden layer 1 neurons	800
Hidden layer 2 neurons	300
Latent dimension	1
Connections	Fully Connected (in all layers)
Activation function	Sigmoid (in all layers)
Training epochs	50

Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	100
Datapoints	1000
Scaling in the initial data space	None
Scaling in the latent space	None

Autoencoder-trained-without-centroids method parameters for squeezed gaussian blobs dataset

Boxplots:



Comments:

We had no improvement in this dataset using the autoencoder training without the centroids method, in stark contrast with the almost perfect results using the centroids. Option k = 20 has

huge standard deviation, due to some rare representation cases that offer improvement, and a slightly better mean value than all the other choices.

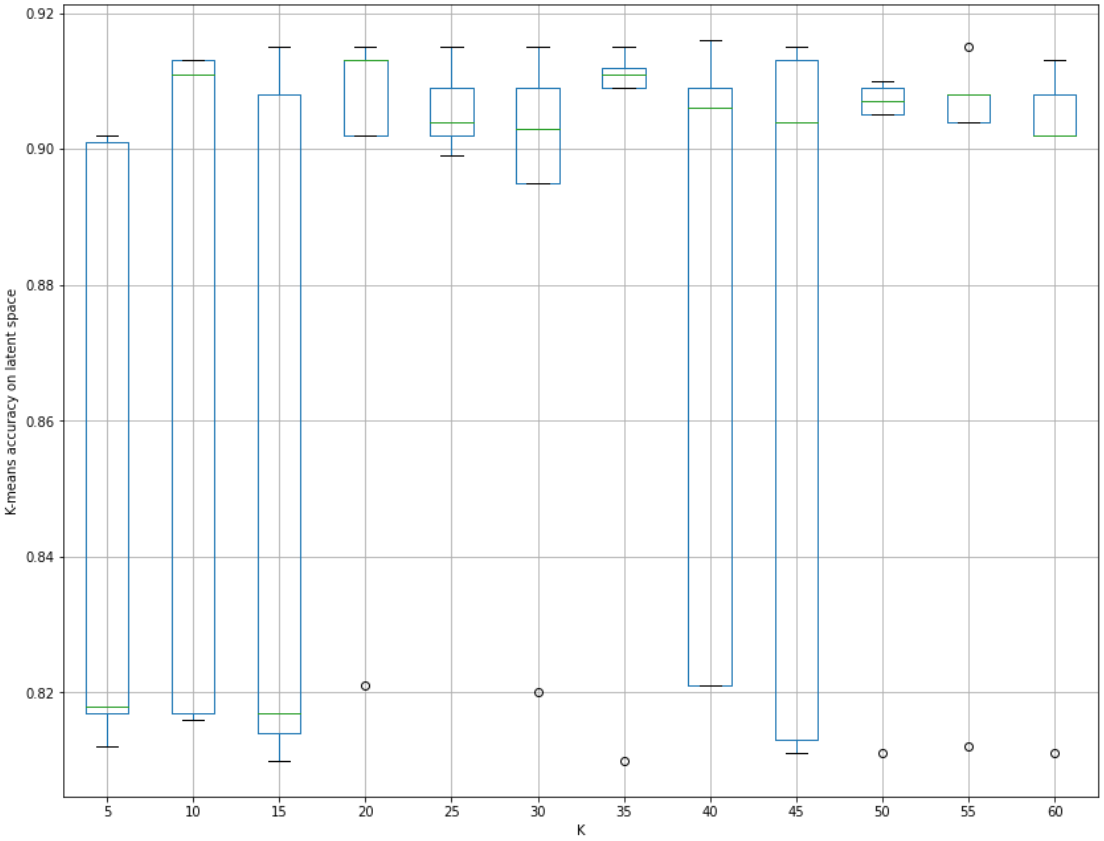
Moons dataset:

Autoencoder and data processing details:

Hidden layer 1 neurons	400
Hidden layer 2 neurons	200
Latent dimension	1
Connections	Fully Connected (in all layers)
Activation function	Sigmoid (in all layers)
Training epochs	50
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	100
Datapoints	1000
Scaling in the initial data space	MinMax
Scaling in the latent space	None

Autoencoder-trained-without-centroids method parameters for moons dataset

Boxplots:



Comments:

The clustering of the moons dataset is improved using this method, but not as much as the training an autoencoder using the centroids method. Option k = 20 is the best in best in most metrics and is obviously very efficient when it comes to k-means clustering accuracy, judging by the above diagram.

Australian dataset:

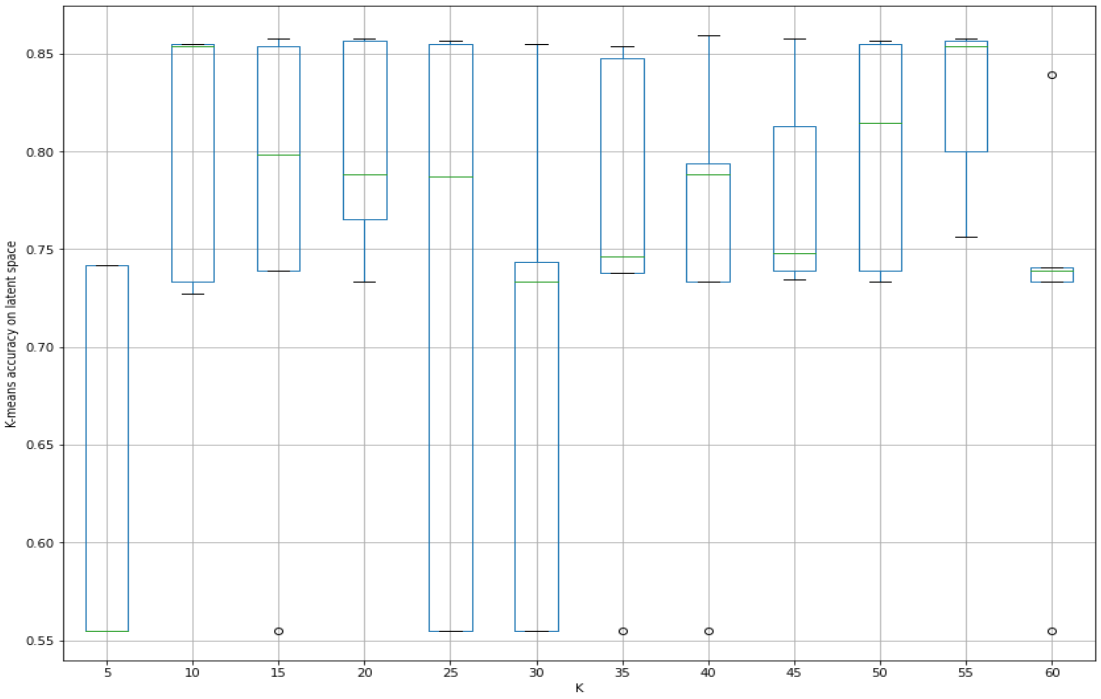
Autoencoder and data processing details:

Hidden layer 1 neurons	500
Hidden layer 2 neurons	300
Latent dimension	5
Connections	Fully Connected (in all layers)
Activation function	Sigmoid (in all layers)
Training epochs	50
Loss function	MSE
Optimiser	Adam optimiser

Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	69
Datapoints	690
Scaling in the initial data space	MinMax
Scaling in the latent space	None

Autoencoder-trained-without-centroids method parameters for australian dataset

Boxplots:



Comments:

Yet another method that fails to make an improvement in the Australian dataset. However, it has to be said that option k = 55 offers the best results in all metrics and comes with low standard deviation, which is always desirable.

Iris dataset:

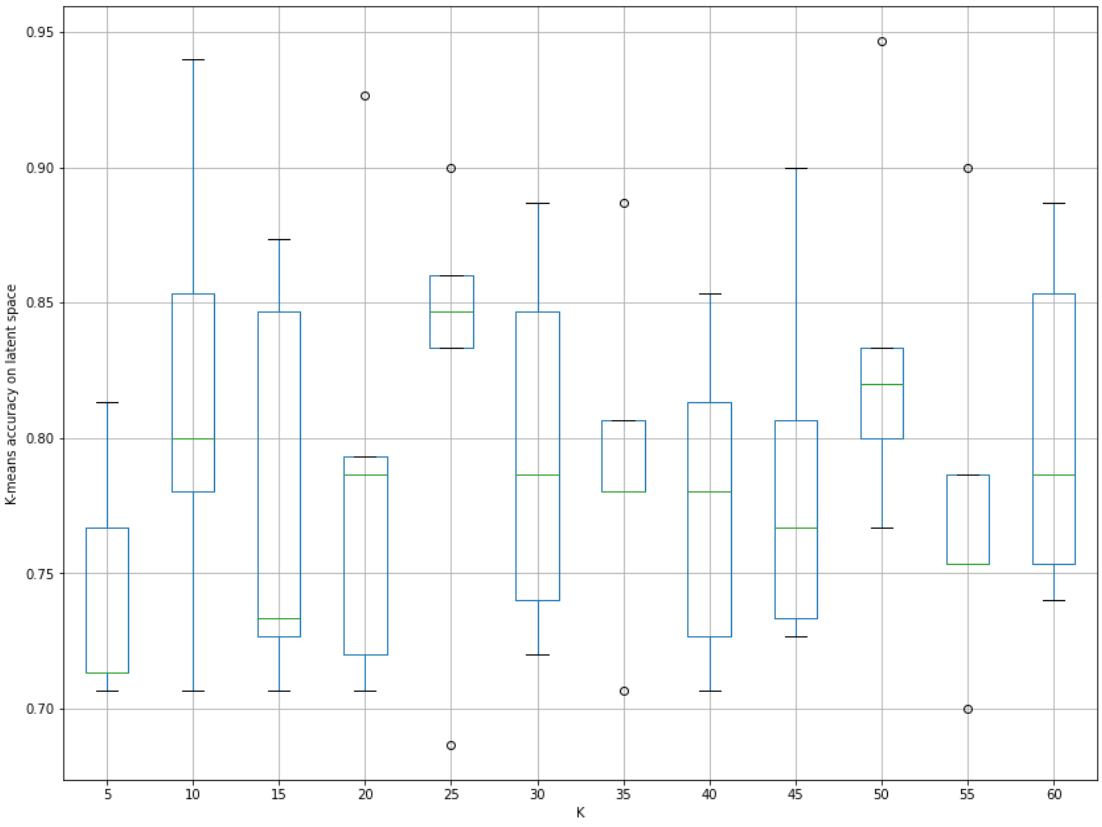
Autoencoder and data processing details:

Hidden layer 1 neurons	500

Hidden layer 2 neurons	200
Latent dimension	3
Connections	Fully Connected (in all layers)
Activation function	Sigmoid (in all layers)
Training epochs	50
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	50
Datapoints	150
Scaling in the initial data space	MinMax
Scaling in the latent space	None

Autoencoder without centroids method parameters for iris dataset

Boxplots:



Comments:

Yet another method that fails to make an improvement in the Iris dataset. However, it has to be said that option k = 25 offers the best results in most metrics and comes with low standard deviation, which is always desirable.

3.5 Boxplots and overclustering criterion for CNN-deep-network clustering method

We have used a specific CNN-autoencoder architecture on the MNIST and Fashion-MNIST datasets, which consist of images that make it the preferable choice for clustering. The architecture has a bottleneck from which we will draw the datapoints for the k-means and agglomerative clustering algorithms run, after training. The difference with the other autoencoder methods is that this time, instead of an MLP network before the latent space and a mirror MLP network of the first after the latent space, we have a CNN and its mirror. The model is trained using the centroids of the initial k-means algorithm.

We present the parameters of the experiments, the boxplots and the overclustering criterion for each of the two datasets.

MNIST dataset:

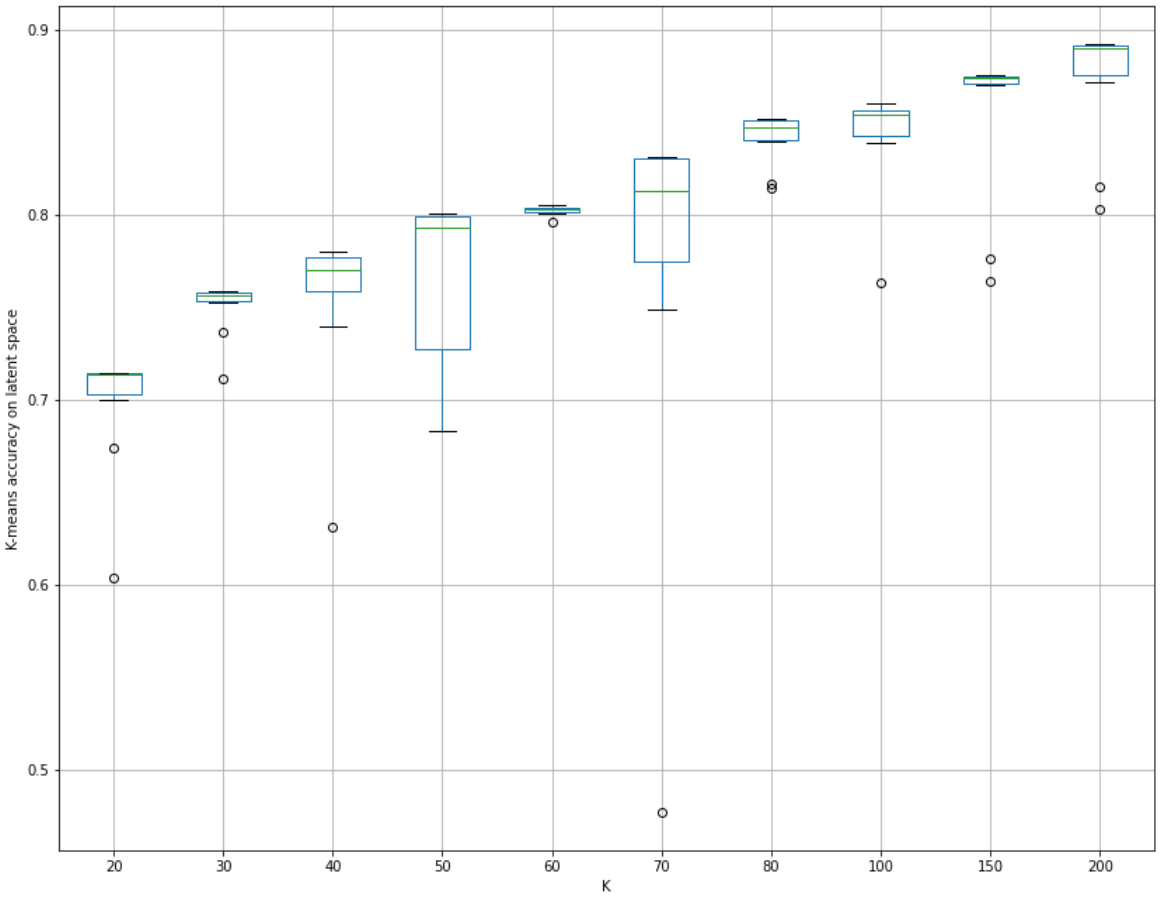
Autoencoder and data processing details:

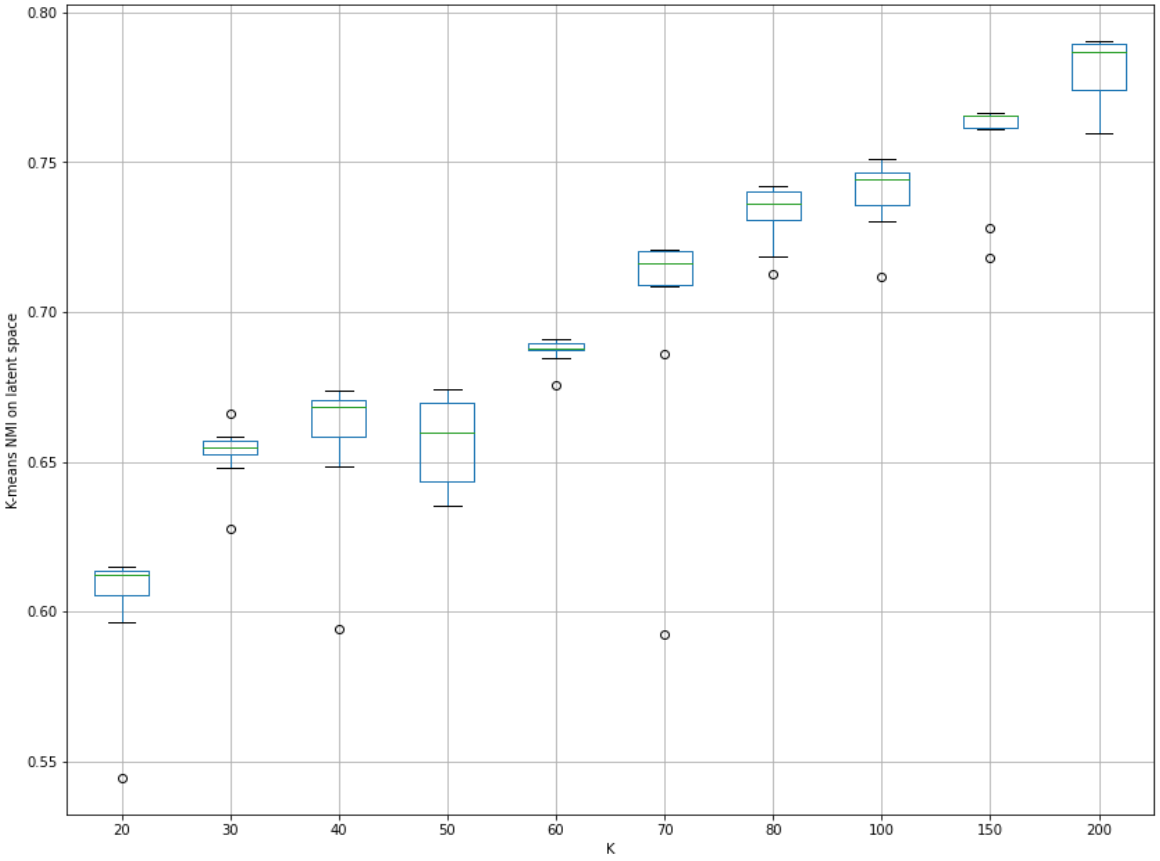
Convolution layer 1 neurons	32
Convolution layer 1 parameters	kernel_size = 5, stride = 2, padding = 2
Convolution layer 2 neurons	64
Convolution layer 2 parameters	kernel_size = 5, stride = 2, padding = 2
Convolution layer 3 neurons	128
Convolution layer 3 parameters	kernel_size = 3, stride = 2, padding = 0
Latent dimension	10
Connections	Fully Connected (in all layers)
Activation function	ReLU (in all layers)

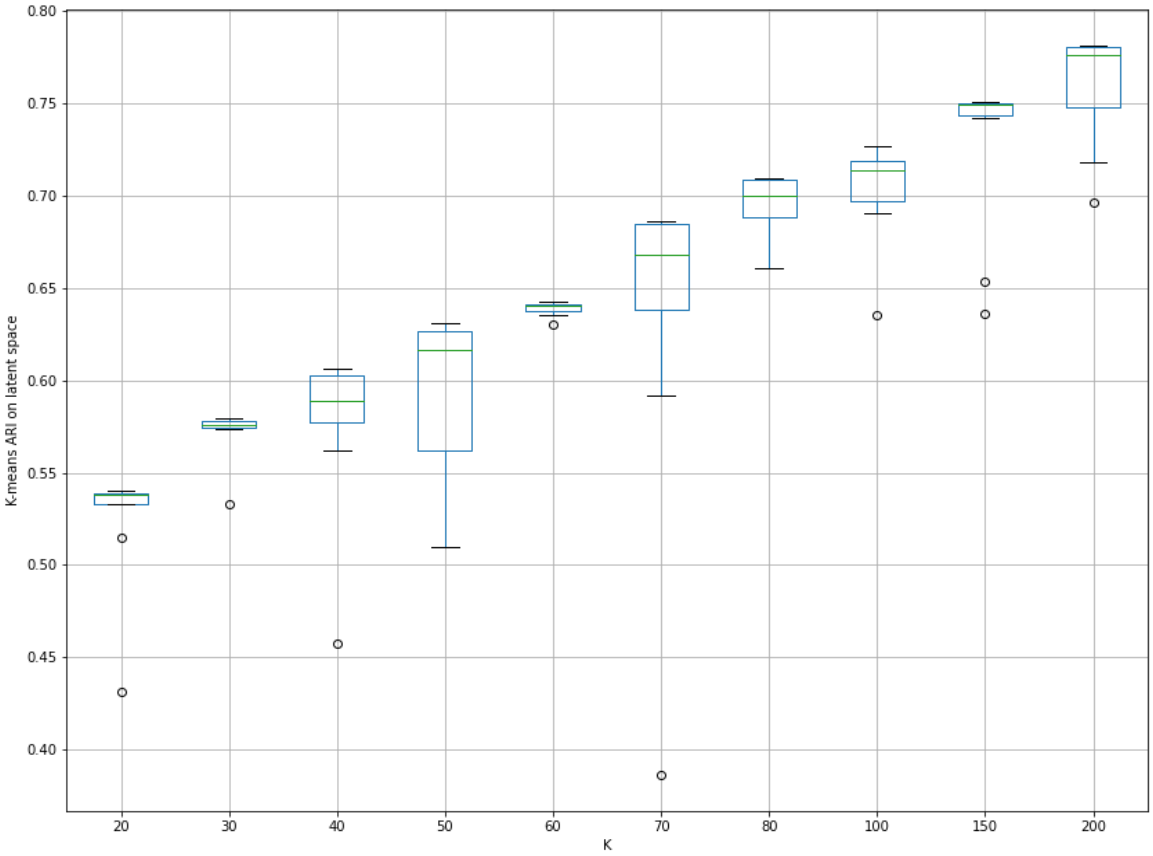
Training epochs	100
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	200
Datapoints	10000 (1000 for each digit)
Scaling in the initial data space	None
Scaling in the latent space	None

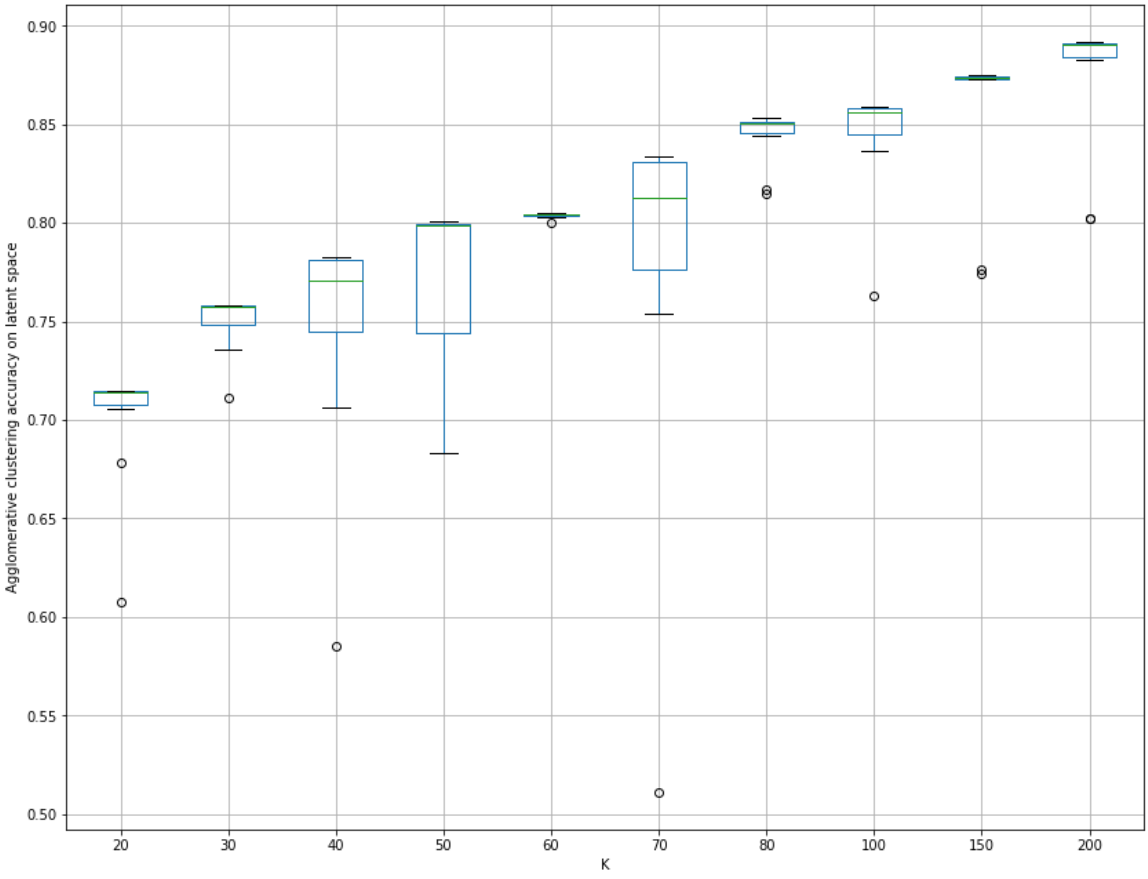
CNN-deep-network clustering method parameters for MNIST dataset

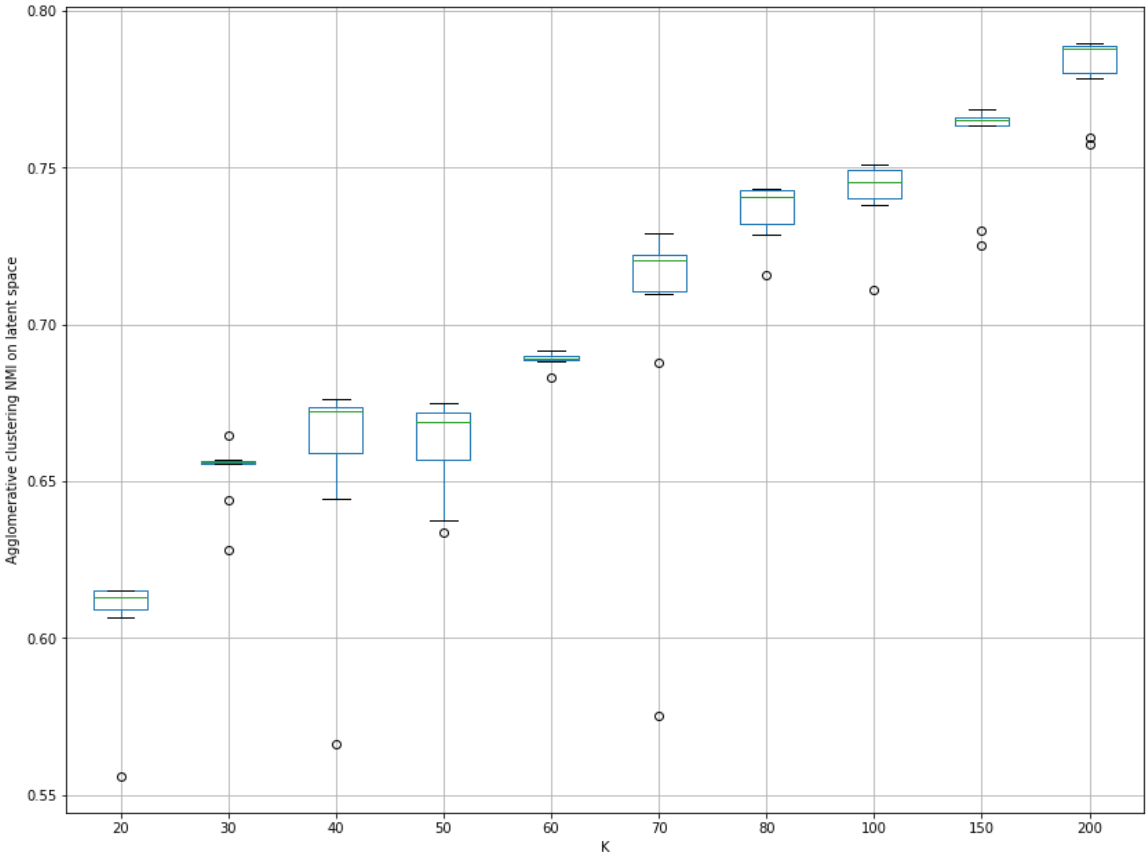
Boxplots:

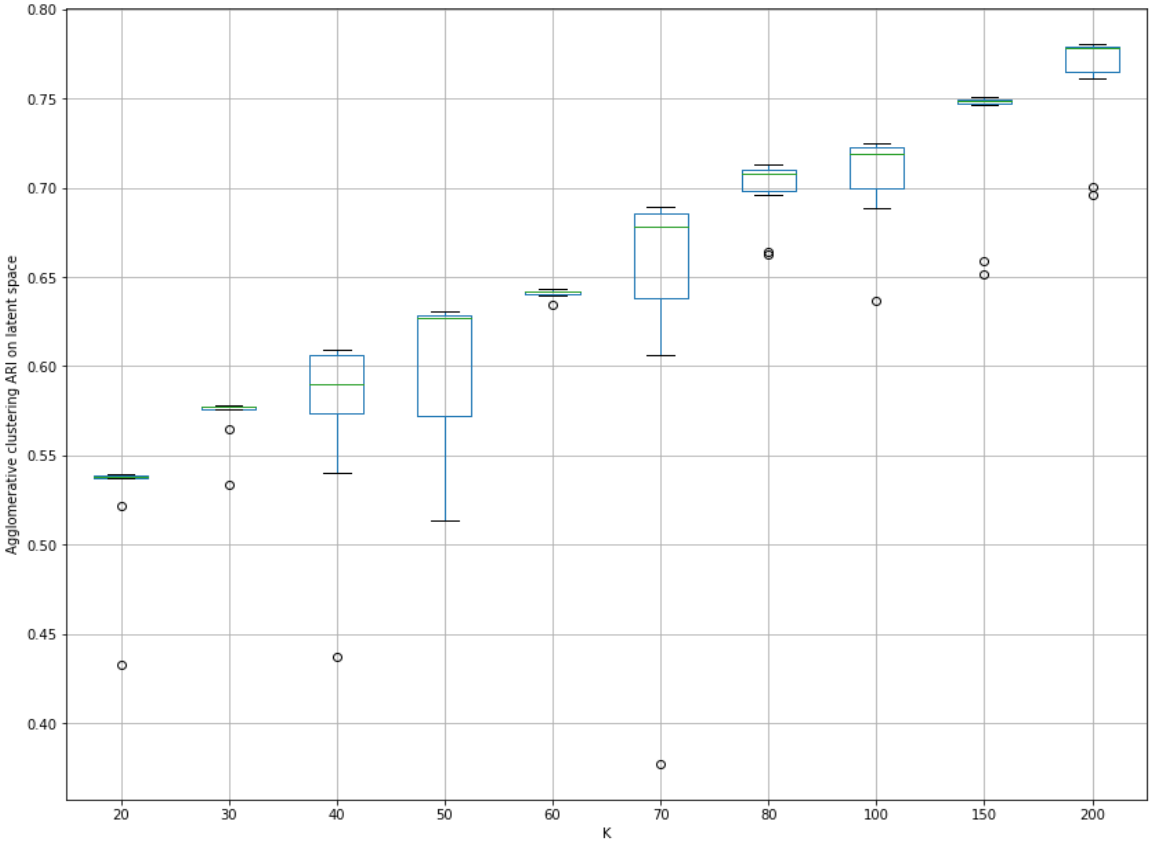


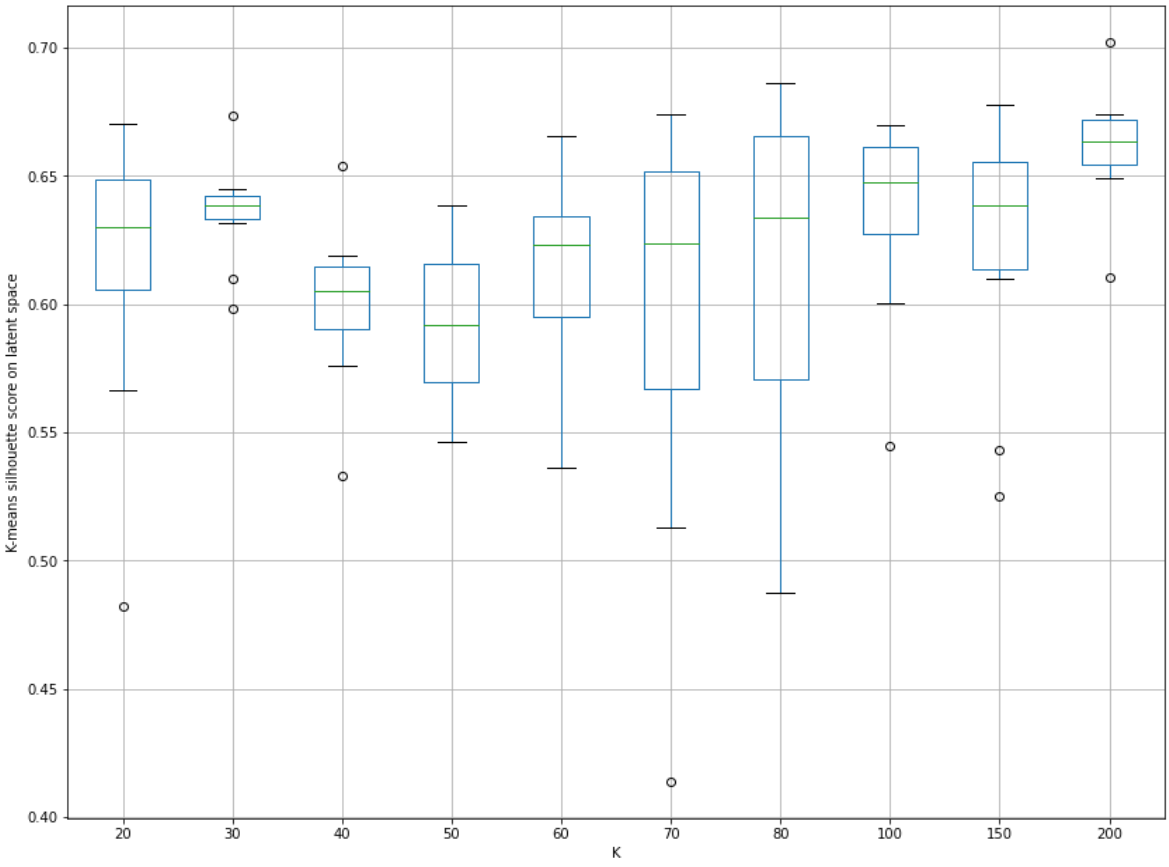

K-means NMI on latent space
0.80
0.75
0.70
0.65
0.60
0.55
20
30
40
50
60
70
80
100
150
200
K

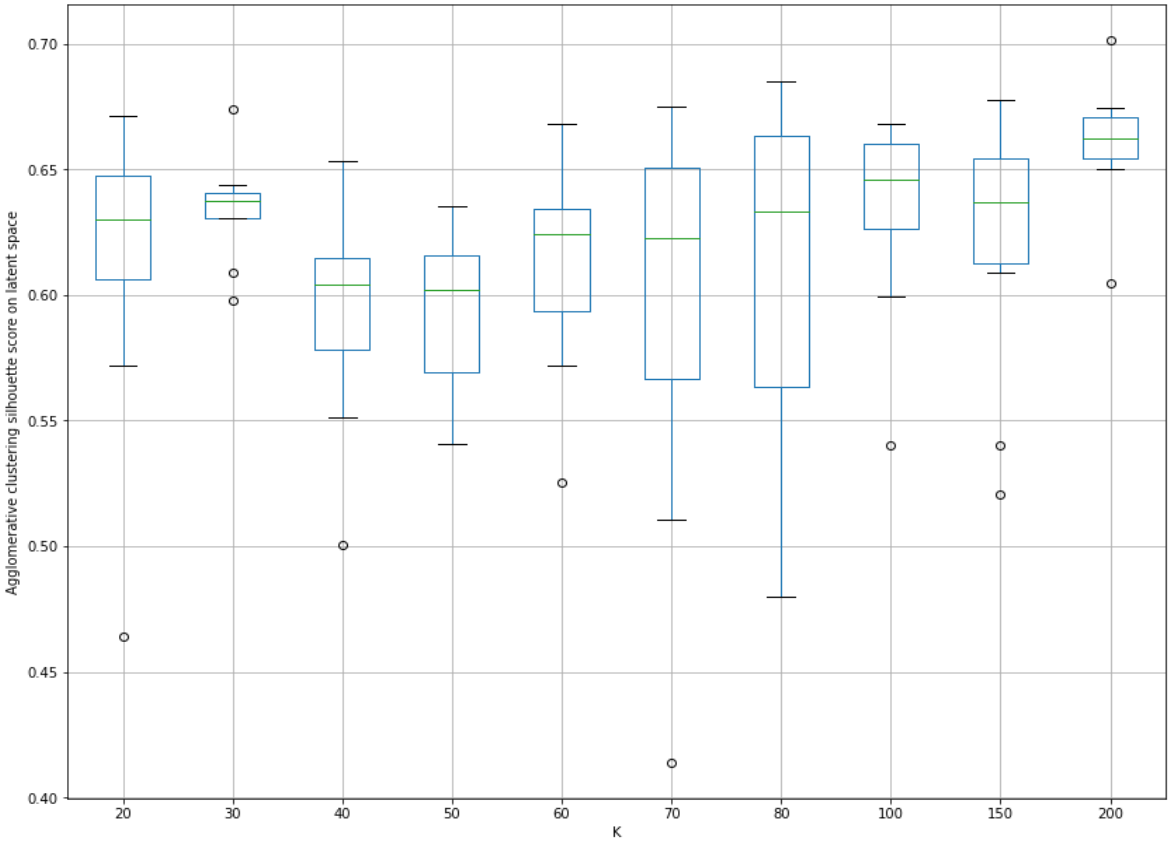

K-means ARI on latent space
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
20
30
40
50
60
70
80
100
150
200
K


Agglomerative clustering accuracy on latent space
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
20
30
40
50
60
70
80
100
150
200
K

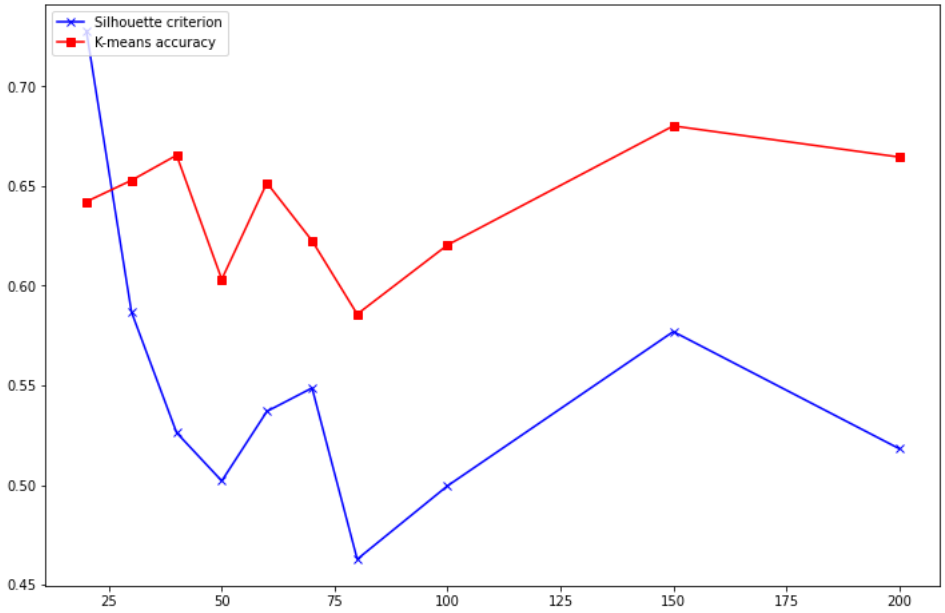

Agglomerative clustering NMI on latent space
0.80
0.75
0.70
0.65
0.60
0.55
20
30
40
50
60
70
80
100
150
200
K


Agglomerative clustering ARI on latent space
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
20
30
40
50
60
70
80
100
150
200
K


K-means silhouette score on latent space
0.70
0.65
0.60
0.55
0.50
0.45
0.40
20
30
40
50
60
70
80
100
150
200
K



Overclustering criterion:



Comments:

In most metrics, our model's improvement in performance increases linearly by increasing the number of clusters used to created the pseudolabels. In this dataset, we have achieved some of the best results of our research, outstandingly increasing the results.

The overclustering criterion doesn't work well in this particular case, as it chooses the least amount of clusters that we have tried, whereas the best choice is to maximise the number of clusters. However, after the first few datapoints in the graph, the silhouette line follows the accuracy line to good effect.

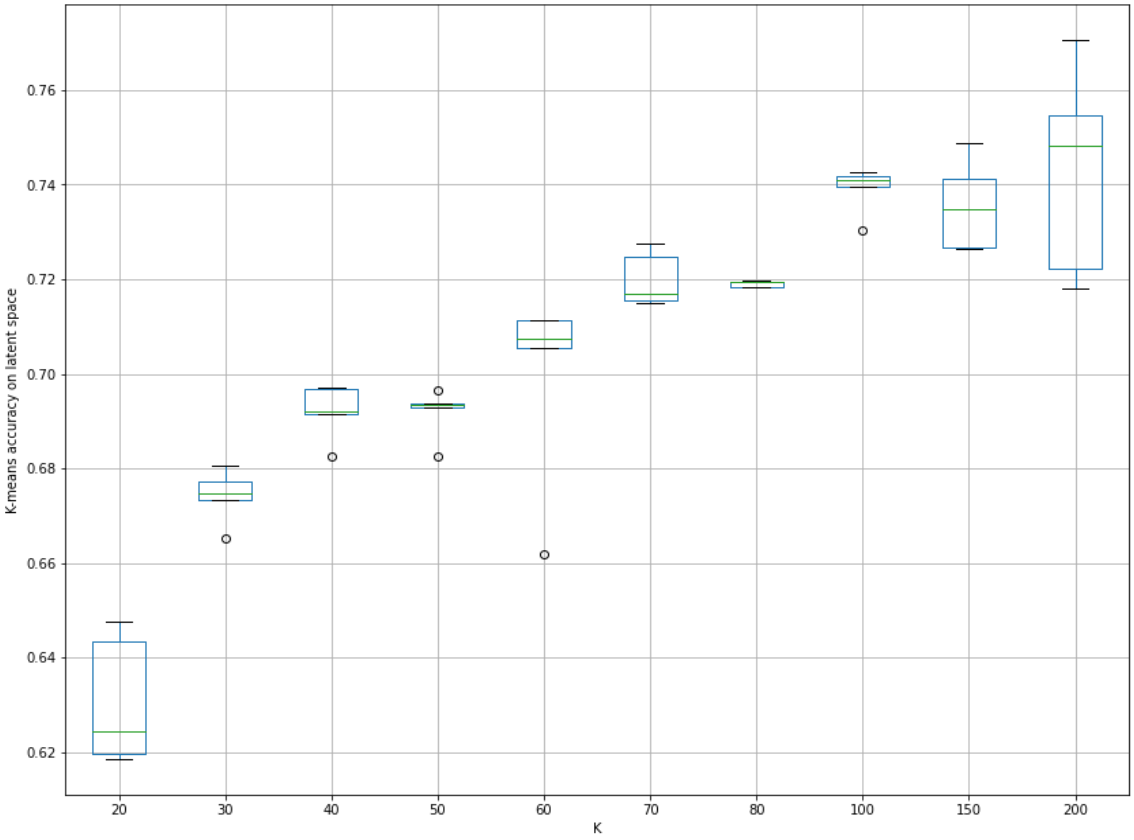
Fashion-MNIST dataset:

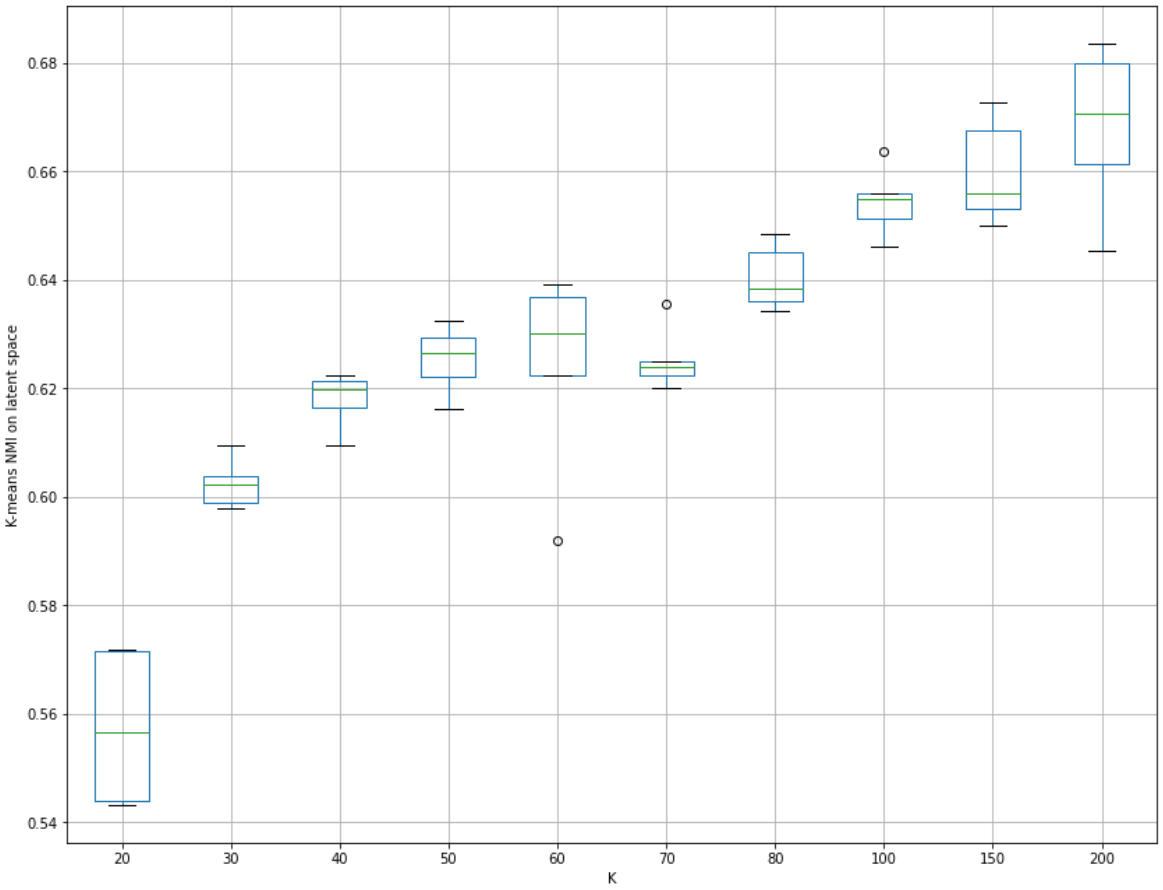
Autoencoder and data processing details:

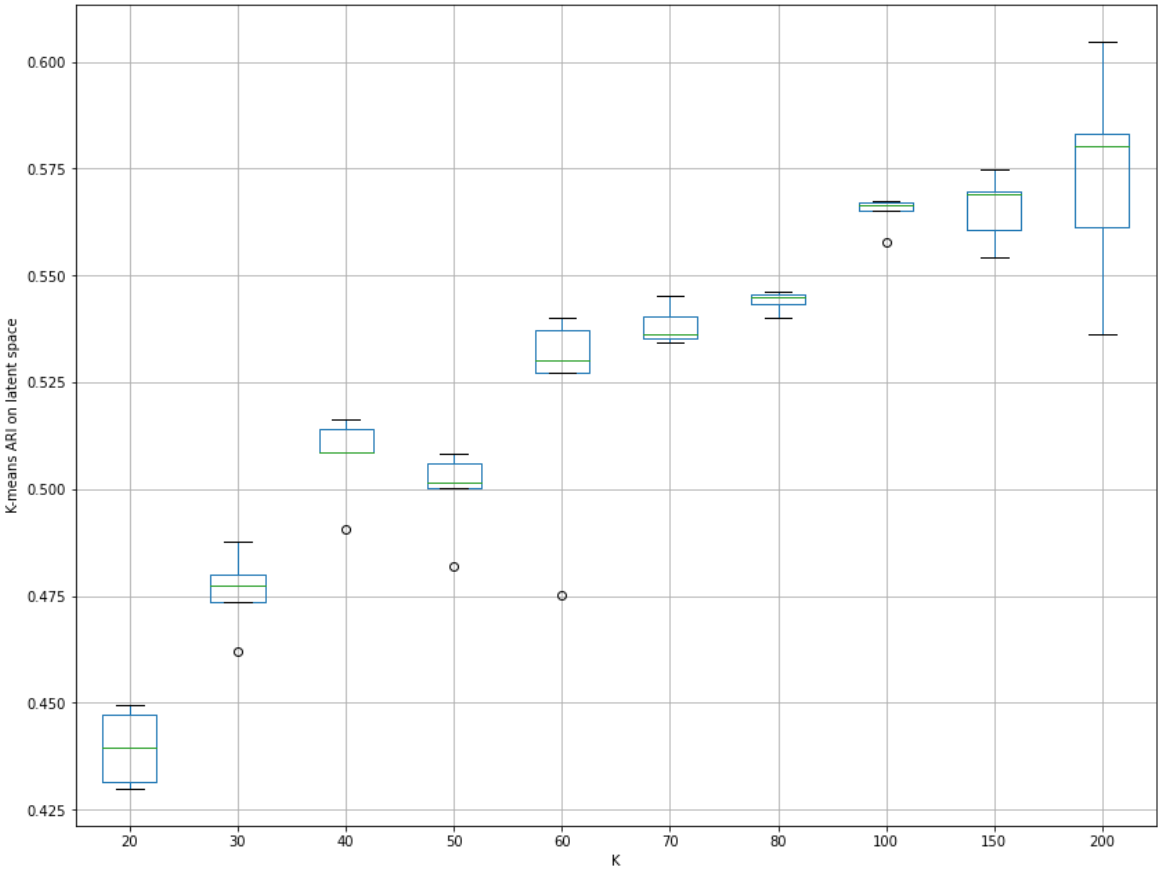
Convolution layer 1 neurons	32
Convolution layer 1 parameters	kernel_size = 5, stride = 2, padding = 2
Convolution layer 2 neurons	64
Convolution layer 2 parameters	kernel_size = 5, stride = 2, padding = 2
Convolution layer 3 neurons	128
Convolution layer 3 parameters	kernel_size = 3, stride = 2, padding = 0
Latent dimension	10
Connections	Fully Connected (in all layers)
Activation function	ReLU (in all layers)
Training epochs	100
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	50
Datapoints	10000 (1000 for each clothing category)
Scaling in the initial data space	None
Scaling in the latent space	None

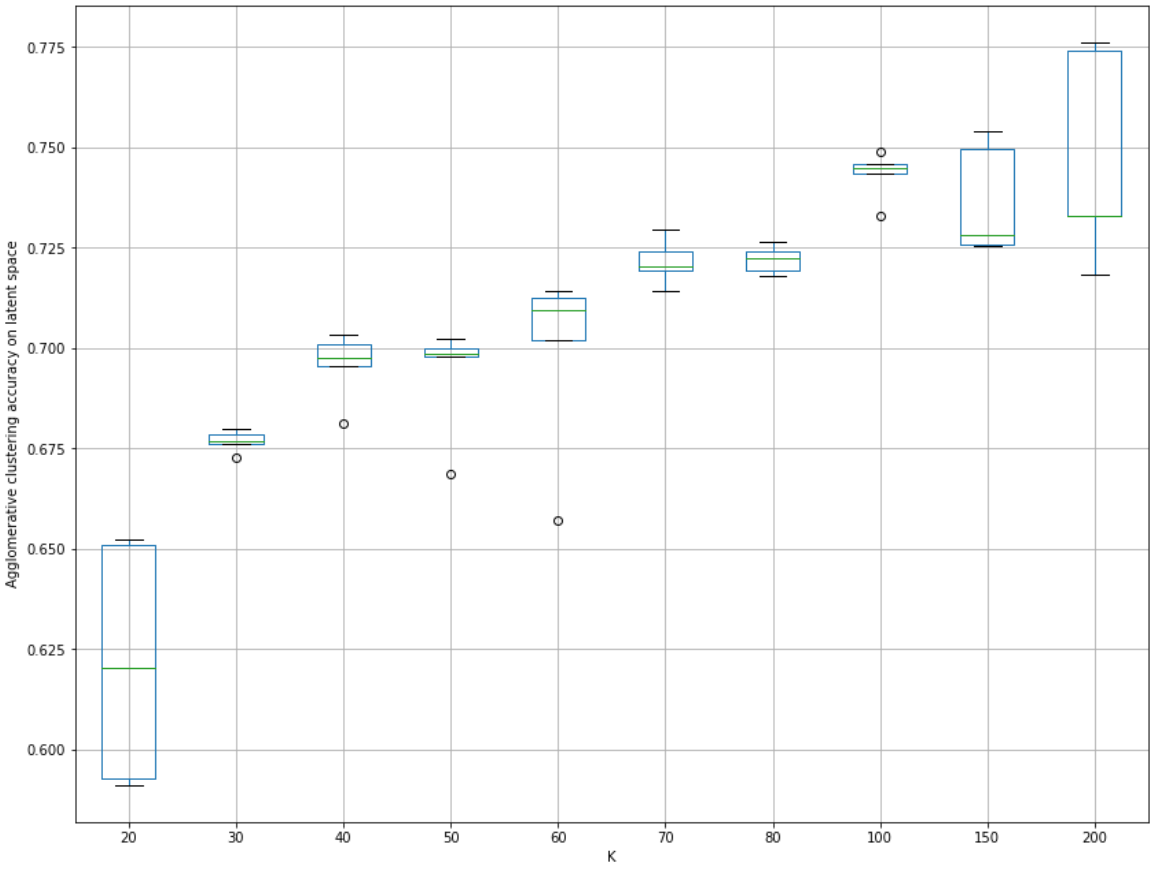
CNN-deep-network clustering method parameters for fashion-MNIST dataset

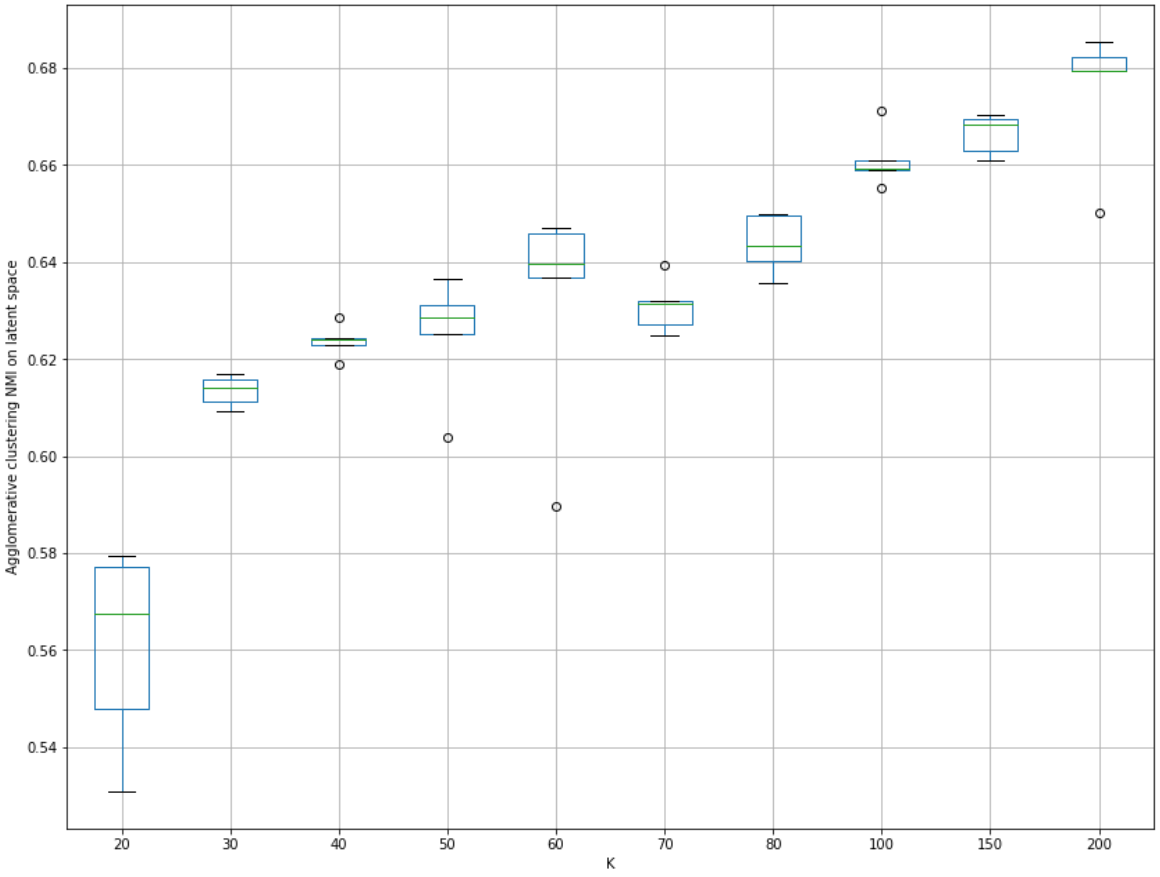
Boxplots:

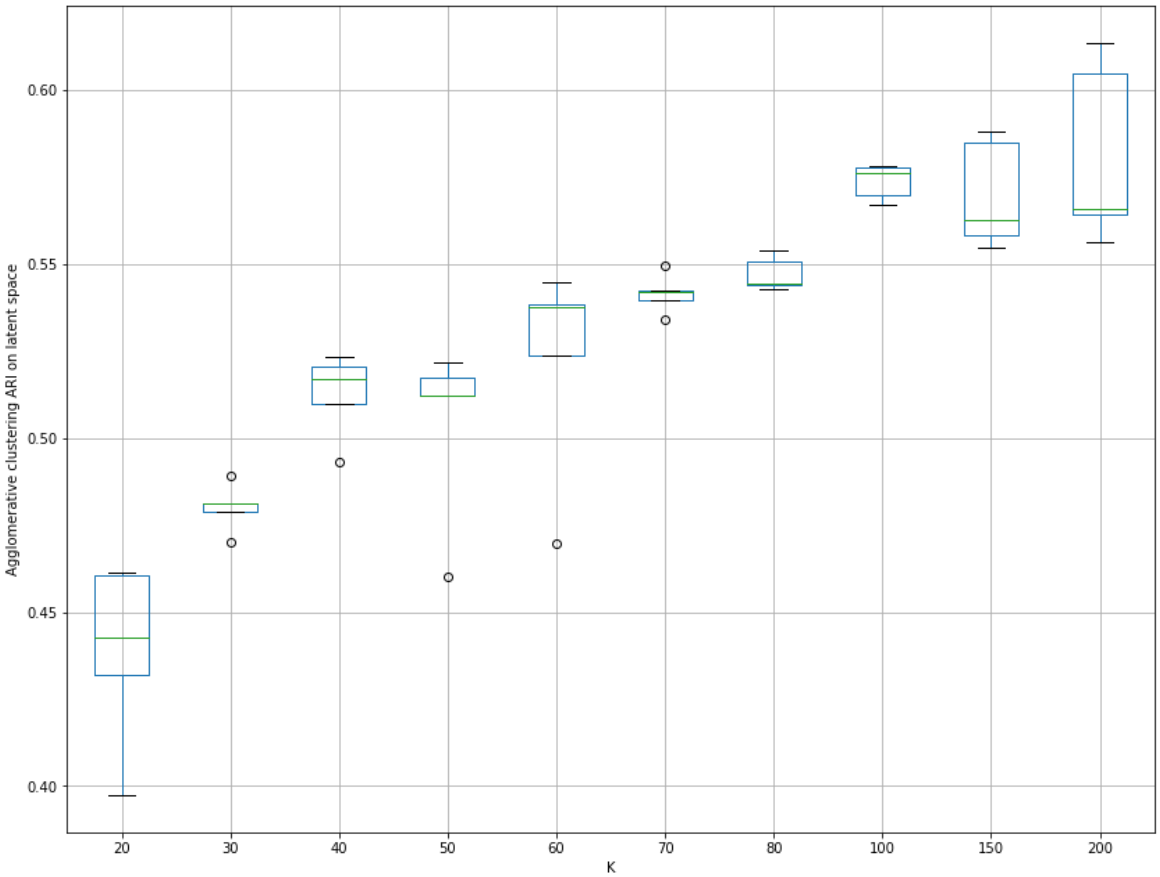

K-means accuracy on latent space
0.76
0.74
0.72
0.70
0.68
0.66
0.64
0.62
20
30
40
50
60
70
80
100
150
200
K

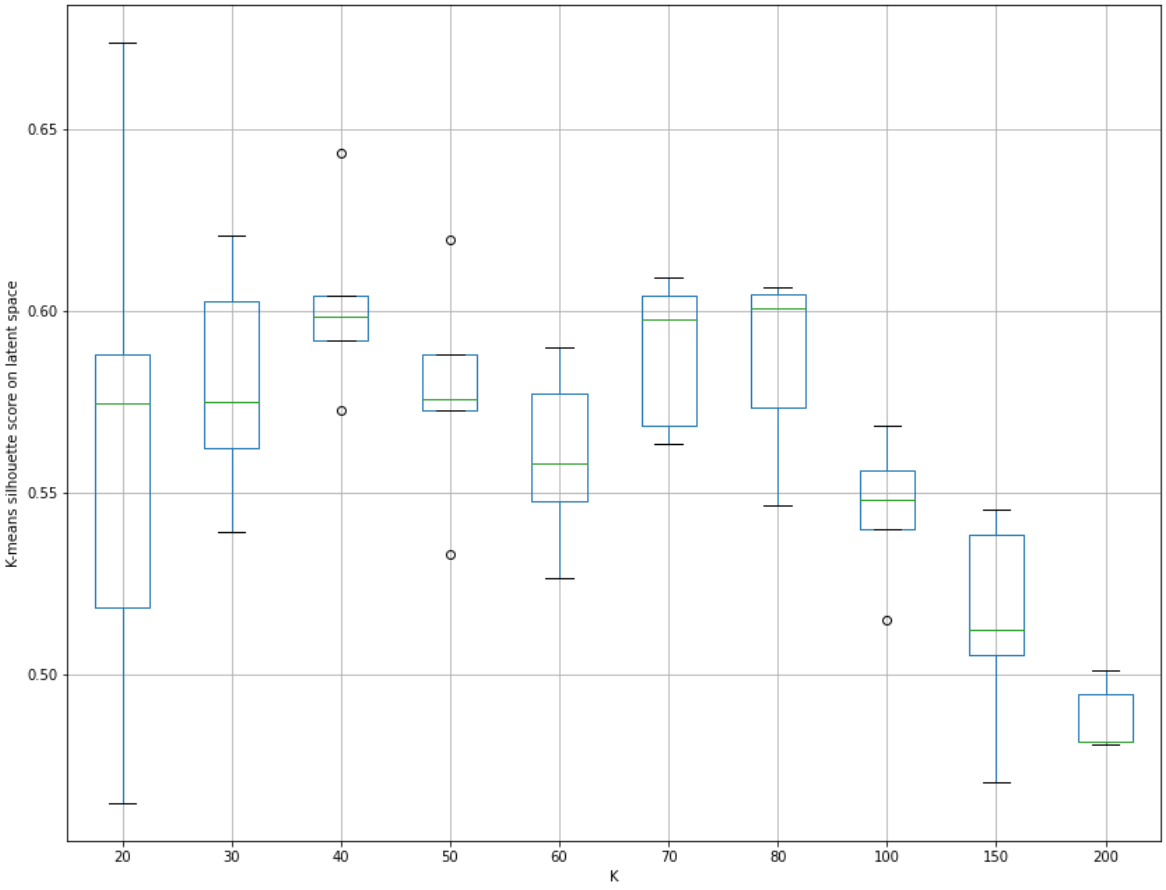

K-means NMI on latent space
0.68
0.66
0.64
0.62
0.60
0.58
0.56
0.54
20
30
40
50
60
70
80
100
150
200
K

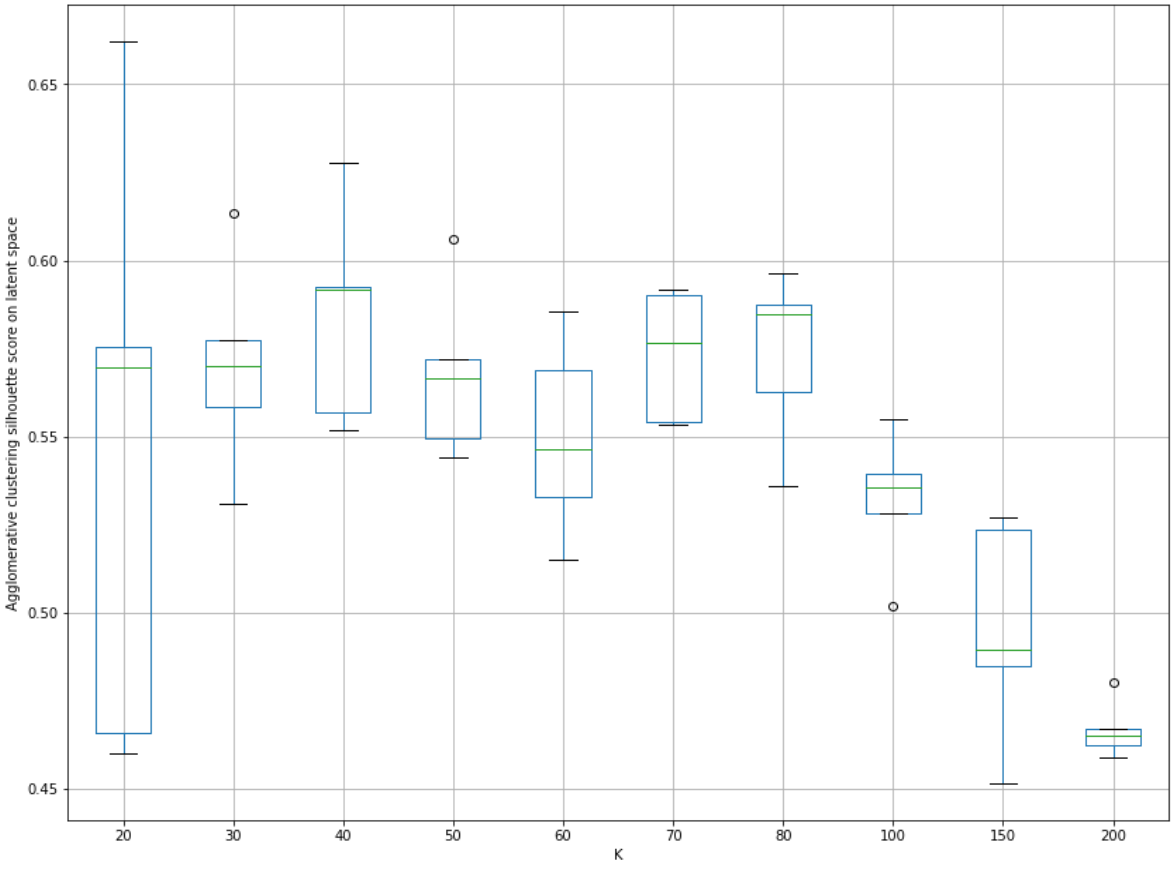

K-means ARI on latent space
0.600
0.575
0.550
0.525
0.500
0.475
0.450
0.425
20
30
40
50
60
70
80
100
150
200
K


Agglomerative clustering accuracy on latent space
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.600
20
30
40
50
60
70
80
100
150
200
K

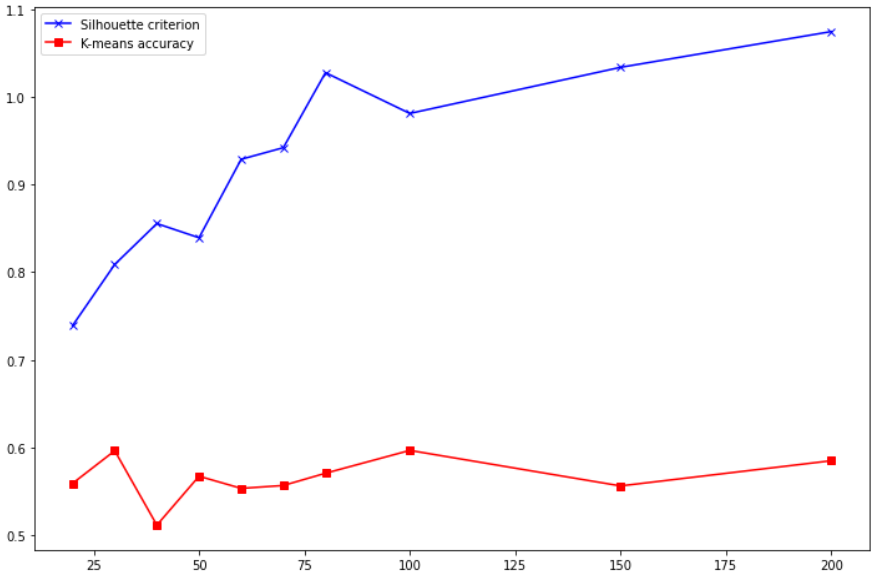

Agglomerative clustering NMI on latent space
0.68
0.66
0.64
0.62
0.60
0.58
0.56
0.54
20
30
40
50
60
70
80
100
150
200
K


Agglomerative clustering ARI on latent space
0.60
0.55
0.50
0.45
0.40
20
30
40
50
60
70
80
100
150
200
K


K-means silhouette score on latent space
0.65
0.60
0.55
0.50
20
30
40
50
60
70
80
100
150
200
K


Agglomerative clustering silhouette score on latent space
0.65
0.60
0.55
0.50
0.45
20
30
40
50
60
70
80
100
150
200
K

Overclustering criterion:



Comments:

In this dataset, using the CNN mode, the results are not as impressive as the results in the MNIST dataset, and this might be attributed to the peculiarities of the clothing images. However, the improvement in performance is clearly noticeable and follows a linearly increasing trend, like in the MNIST dataset. The overclustering criterion works perfectly, as it suggests we choose the maximum number of clusters. According to the boxplots, this is the best choice.

3.5.1 Training the CNN-autoencoder model without cluster centroids (proof of concept)

Previously, the CNN-autoencoder model we tried used the centroids of the k-means algorithm for training. We tested the performance of the exact same model, but training it without the centroids, in order to prove that the clustering performance of its latent space will deteriorate.

MNIST dataset:

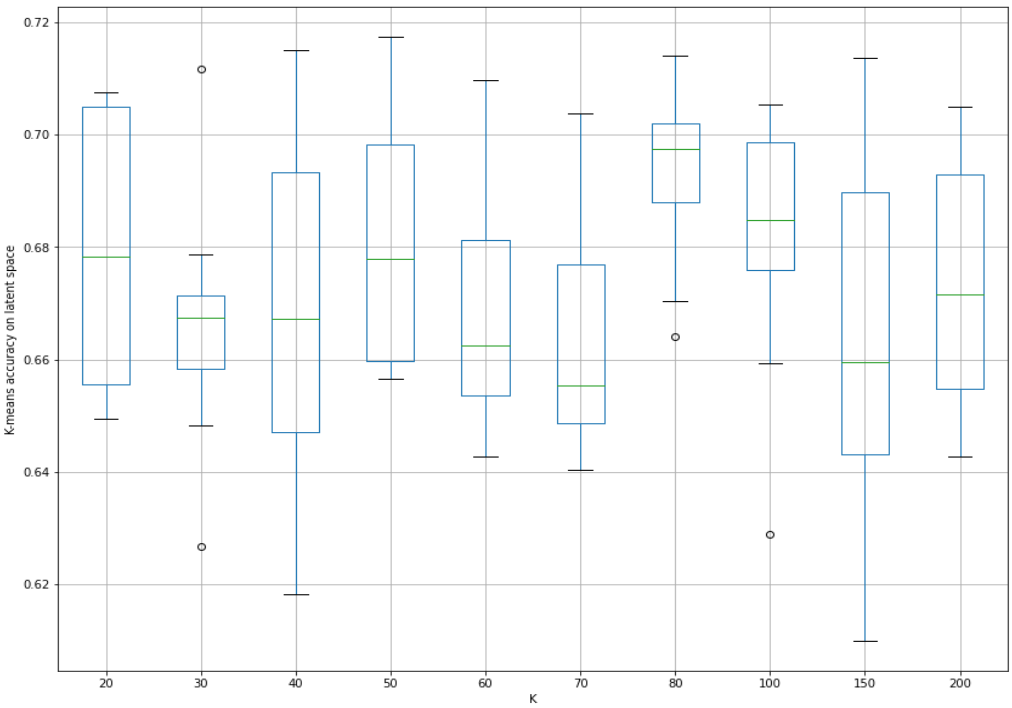
Autoencoder and data processing details:

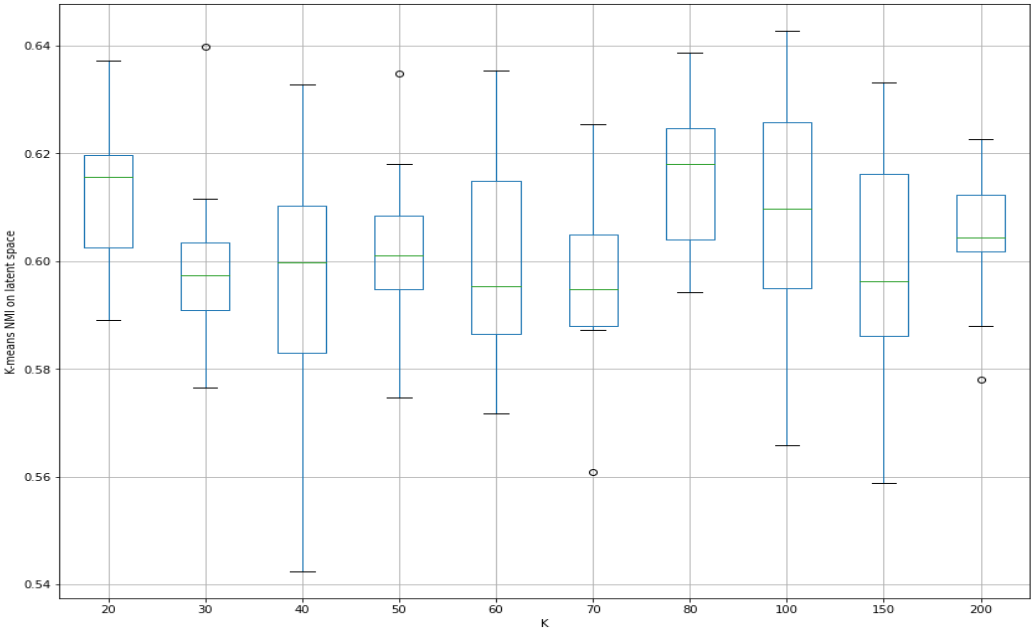
Convolution layer 1 neurons	32
Convolution layer 1 parameters	kernel_size = 5, stride = 2, padding = 2

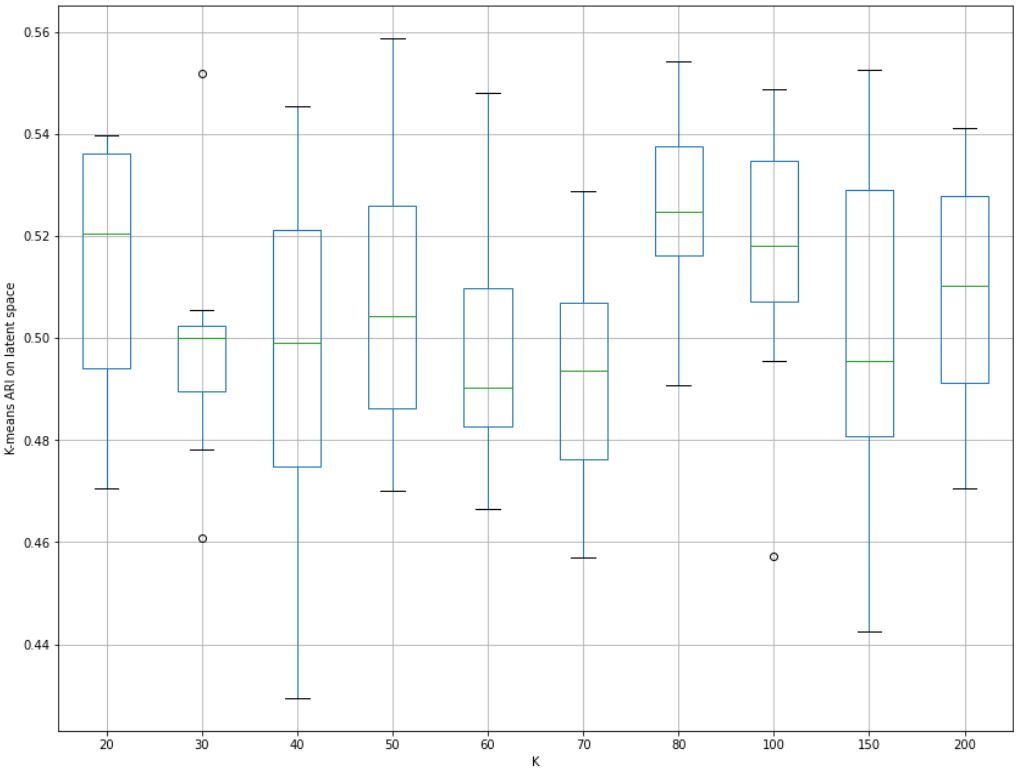
Convolution layer 2 neurons	64
Convolution layer 2 parameters	kernel_size = 5, stride = 2, padding = 2
Convolution layer 3 neurons	128
Convolution layer 3 parameters	kernel_size = 3, stride = 2, padding = 0
Latent dimension	10
Connections	Fully Connected (in all layers)
Activation function	ReLU (in all layers)
Training epochs	100
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	200
Datapoints	10000 (1000 for each digit)
Scaling in the initial data space	None
Scaling in the latent space	None

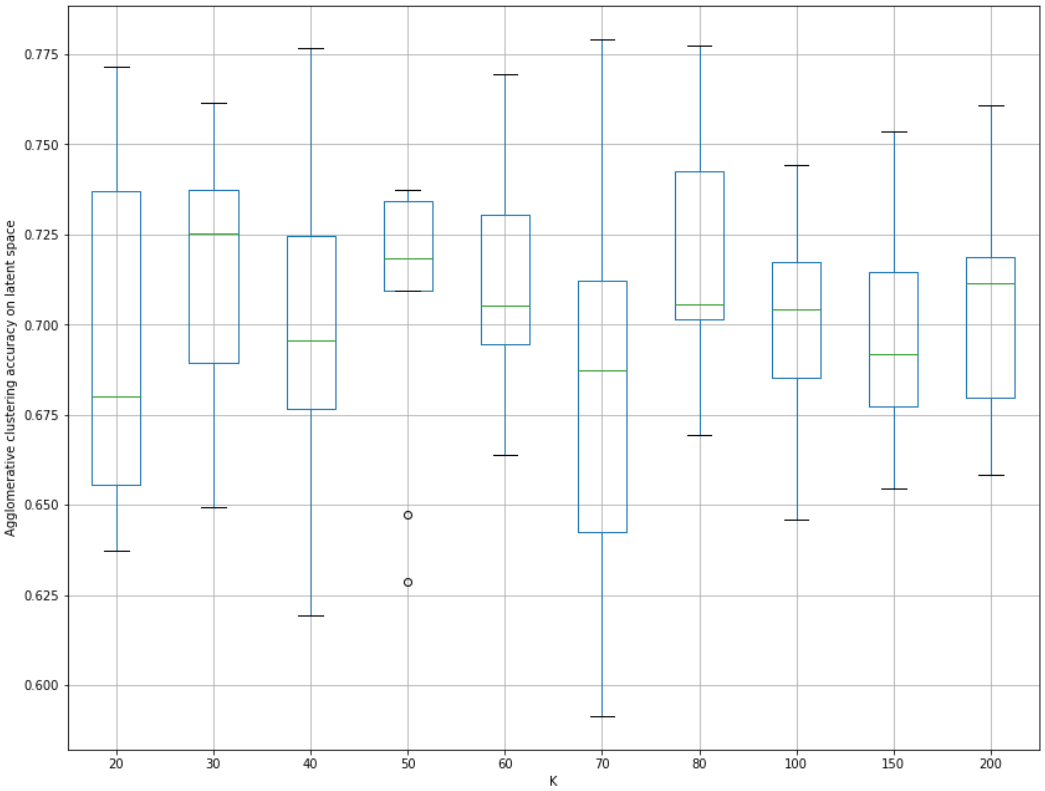
CNN-deep-network-without-centroids clustering method parameters for MNIST dataset

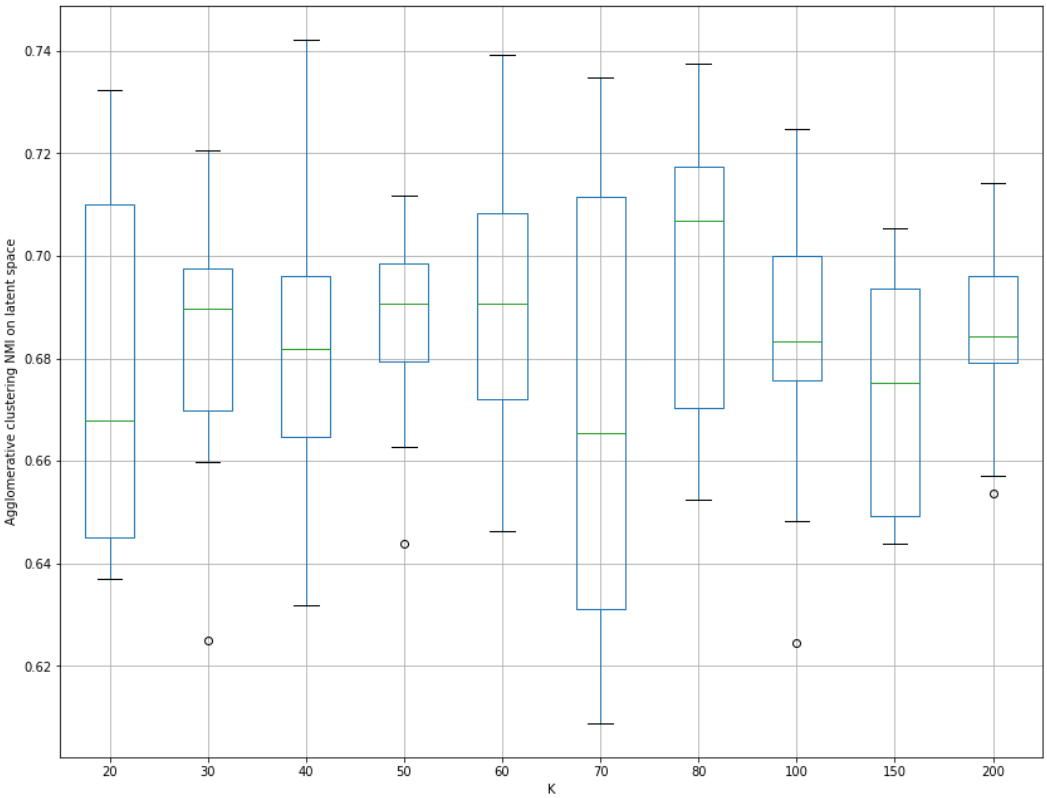
Boxplots:

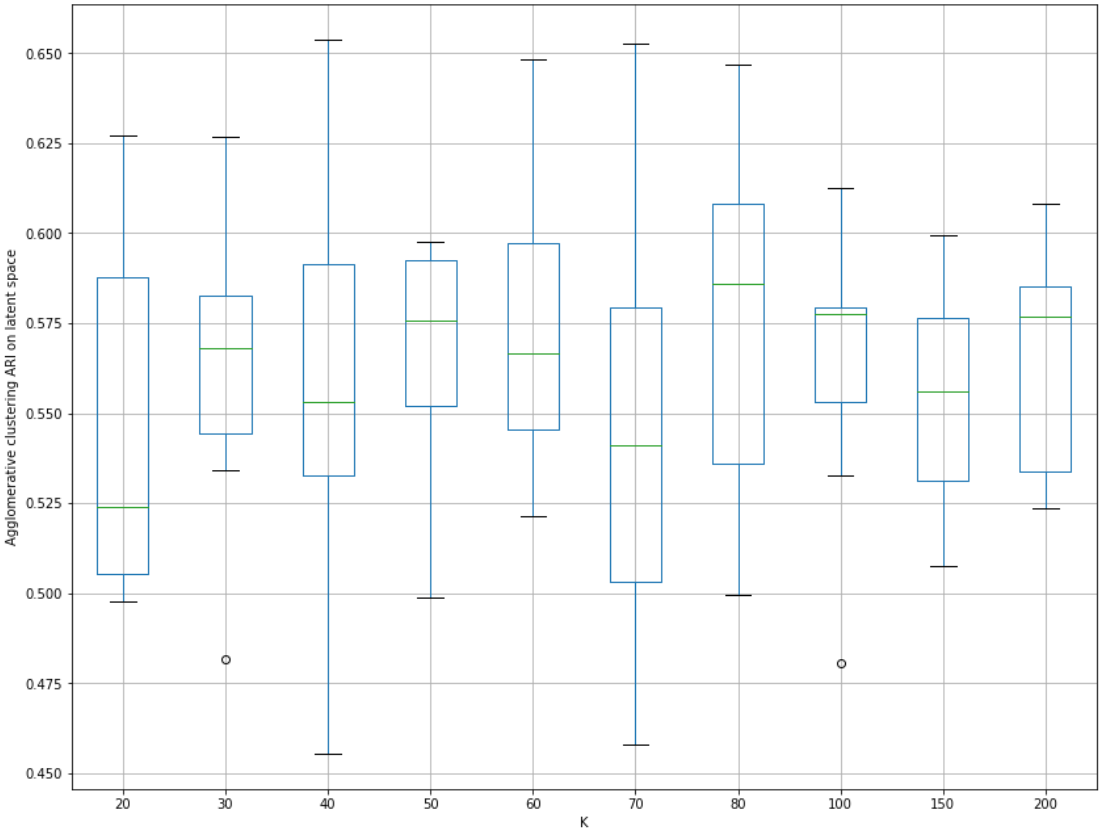


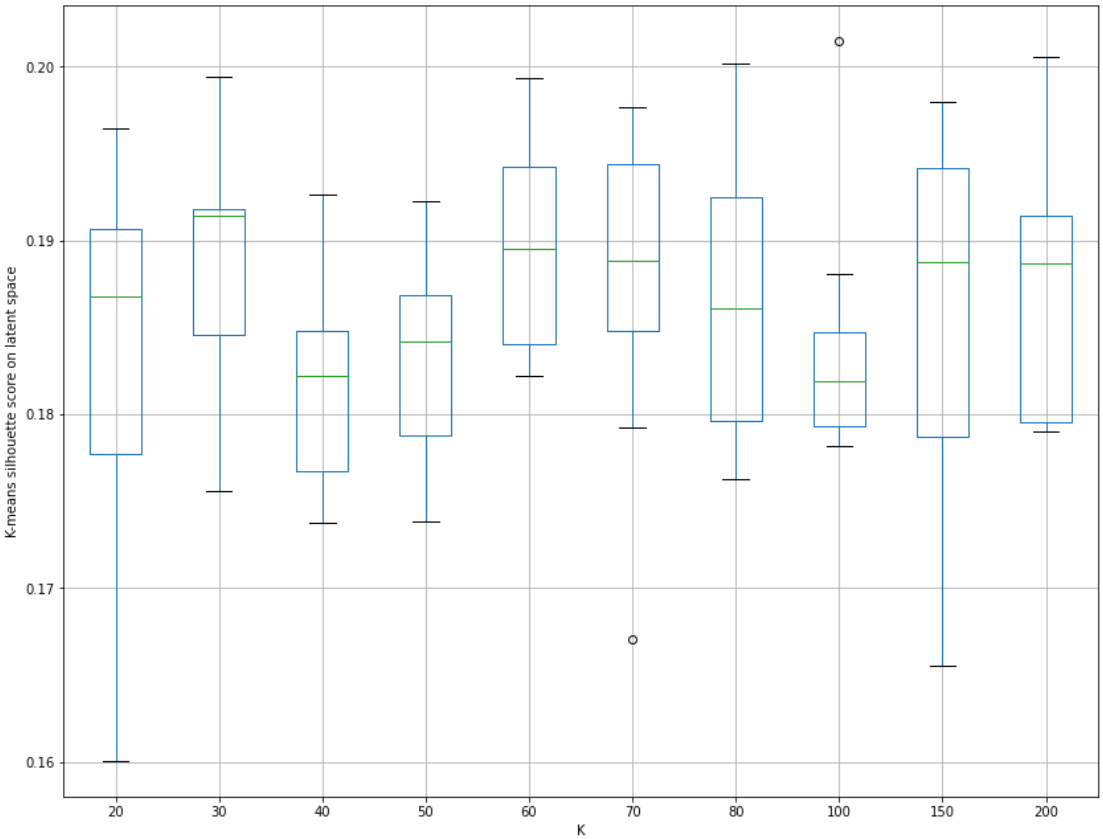

K-means NMI on latent space
0.64
0.62
0.60
0.58
0.56
0.54
20
30
40
50
60
70
80
100
150
200
K

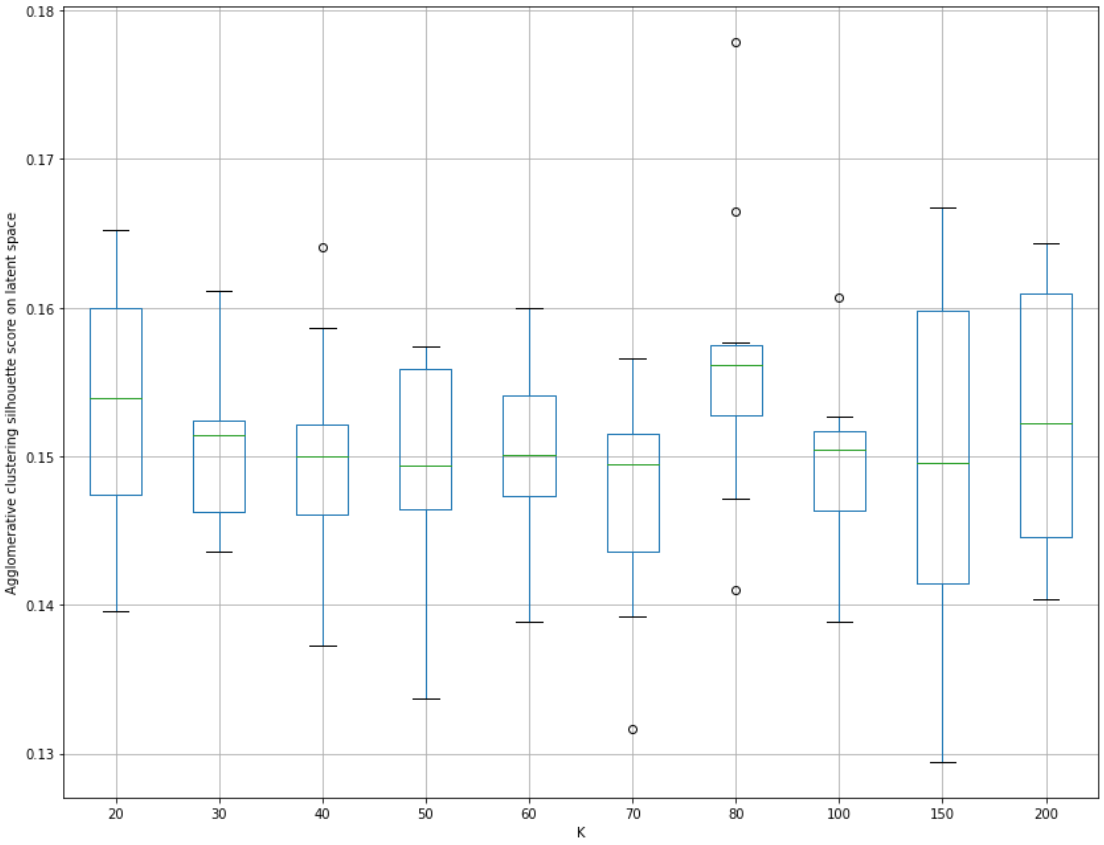

K-means ARI on latent space
0.56
0.54
0.52
0.50
0.48
0.46
0.44
20
30
40
50
60
70
80
100
150
200
K


Agglomerative clustering accuracy on latent space
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.600
20
30
40
50
60
70
80
100
150
200
K

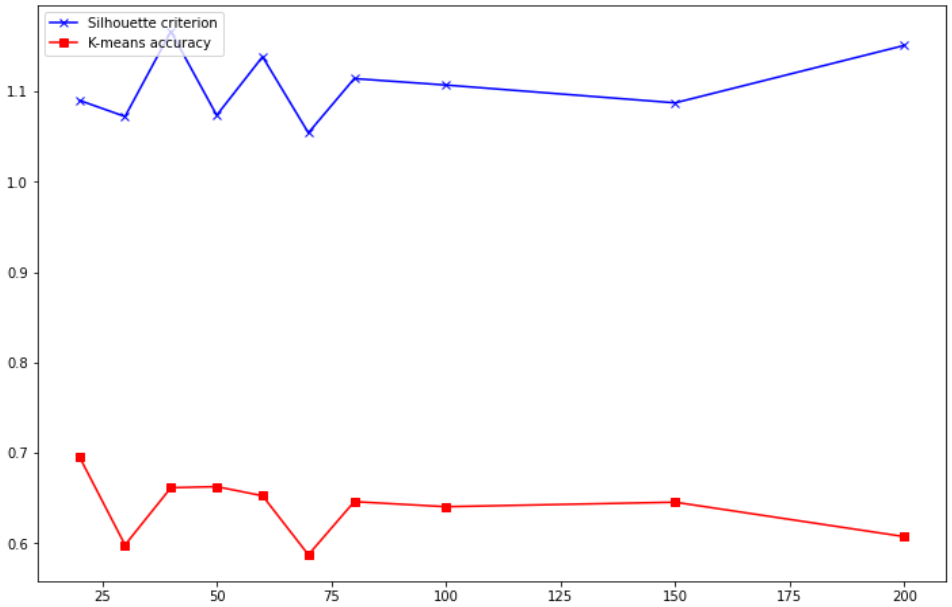

Agglomerative clustering NMI on latent space
0.74
0.72
0.70
0.68
0.66
0.64
0.62
20
30
40
50
60
70
80
100
150
200
K


Agglomerative clustering ARI on latent space
0.650
0.625
0.600
0.575
0.550
0.525
0.500
0.475
0.450
20
30
40
50
60
70
80
100
150
200
K


K-means silhouette score on latent space
0.20
0.19
0.18
0.17
0.16
20
30
40
50
60
70
80
100
150
200
K



Overclustering criterion:



Comments:

It is obvious that without using the centroids during the training of the model, performance deteriorates. The large choices for K don't perform even close to as well as during the use of

the centroids-method. However, there is still improvement to a certain level. The overclustering criterion chooses option k = 40, which is not the best, but generally in this dataset there is little difference between all the option, so it is difficult for the criterion to work optimally.

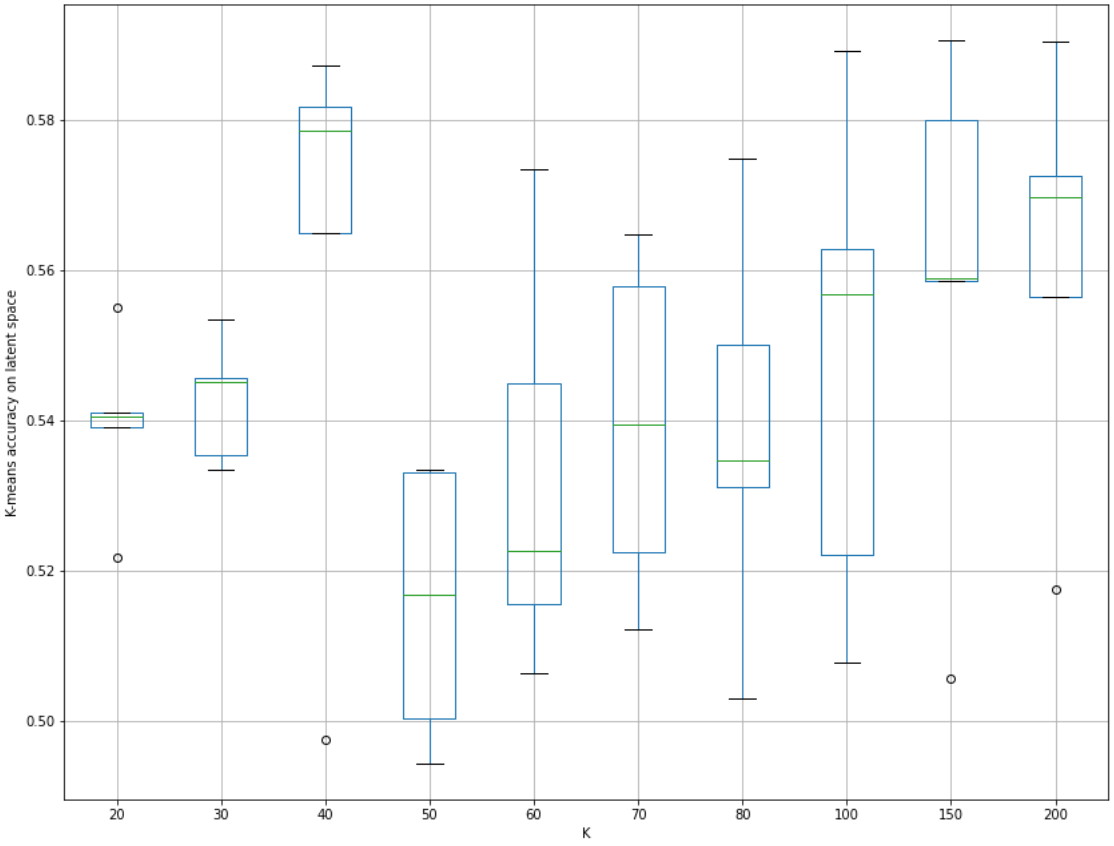
Fashion-MNIST dataset:

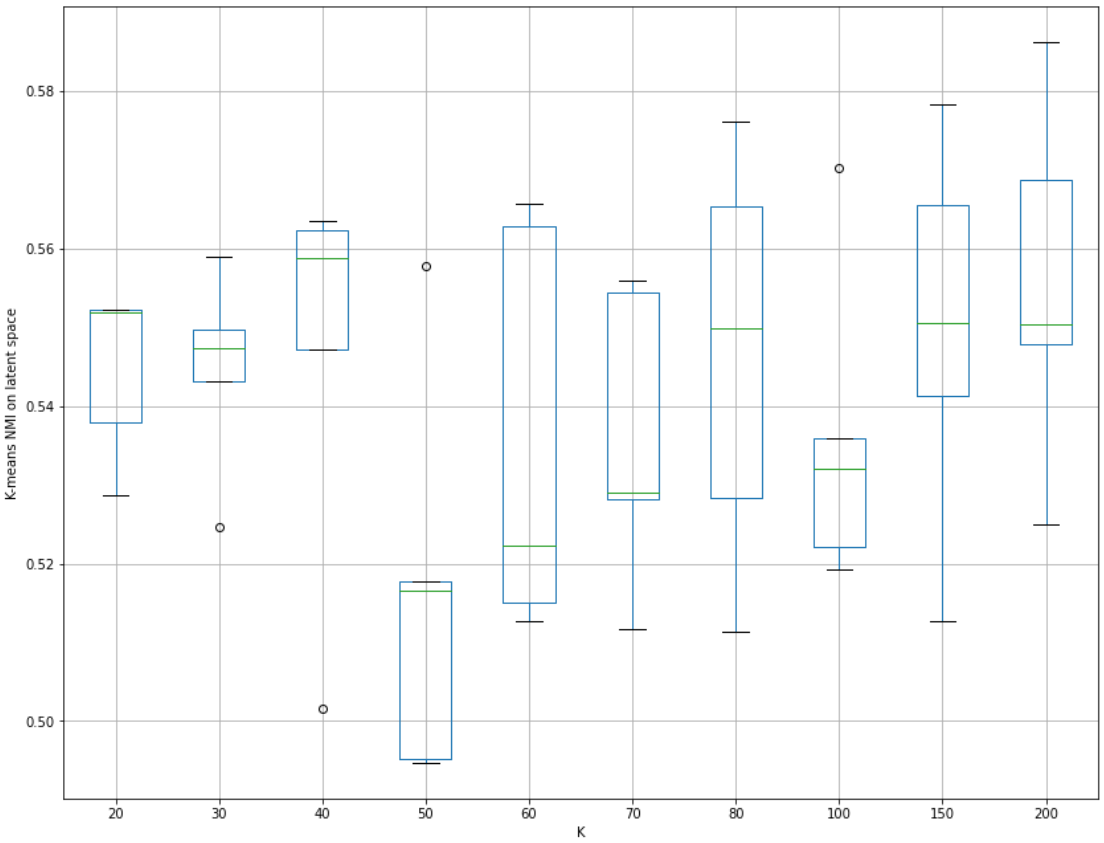
Autoencoder and data processing details:

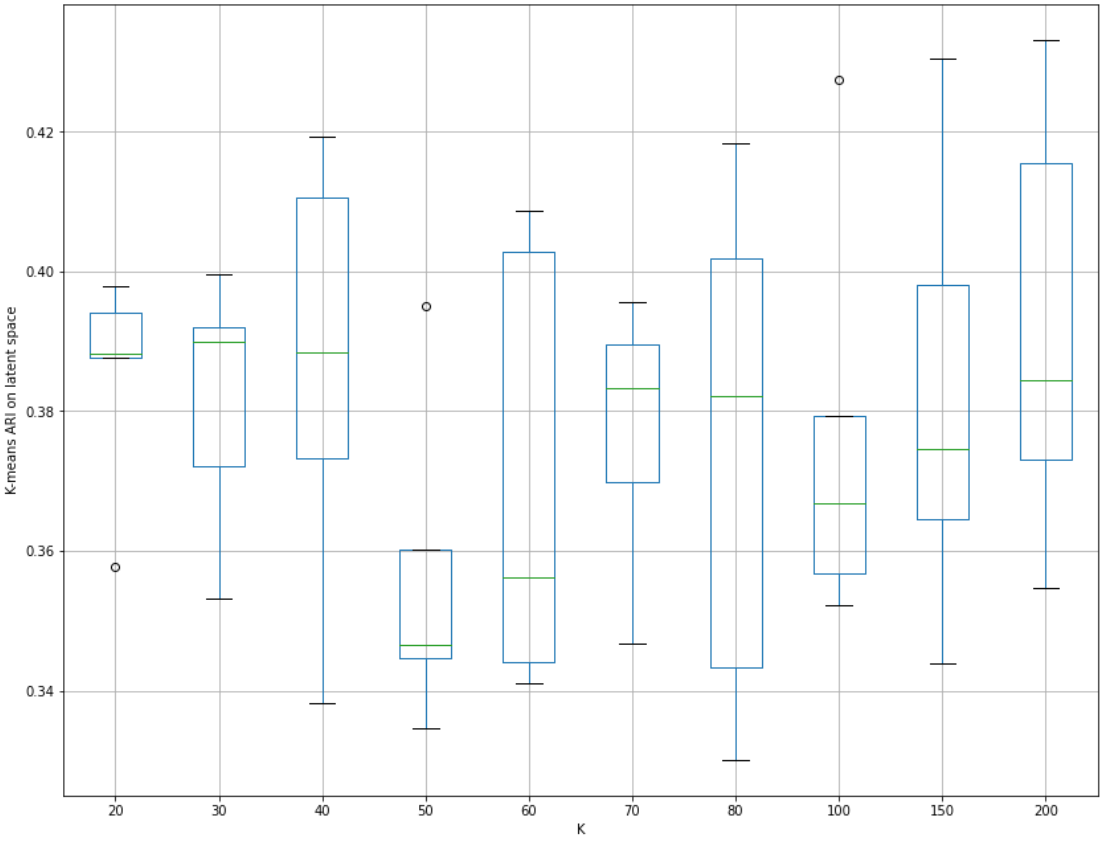
Convolution layer 1 neurons	32
Convolution layer 1 parameters	kernel_size = 5, stride = 2, padding = 2
Convolution layer 2 neurons	64
Convolution layer 2 parameters	kernel_size = 5, stride = 2, padding = 2
Convolution layer 3 neurons	128
Convolution layer 3 parameters	kernel_size = 3, stride = 2, padding = 0
Latent dimension	10
Connections	Fully Connected (in all layers)
Activation function	ReLU (in all layers)
Training epochs	100
Loss function	MSE
Optimiser	Adam optimiser
Learning rate	10^{-4}
Weight decay	10^{-5}
Batch size	50
Datapoints	10000 (1000 for each clothing category)
Scaling in the initial data space	None
Scaling in the latent space	None

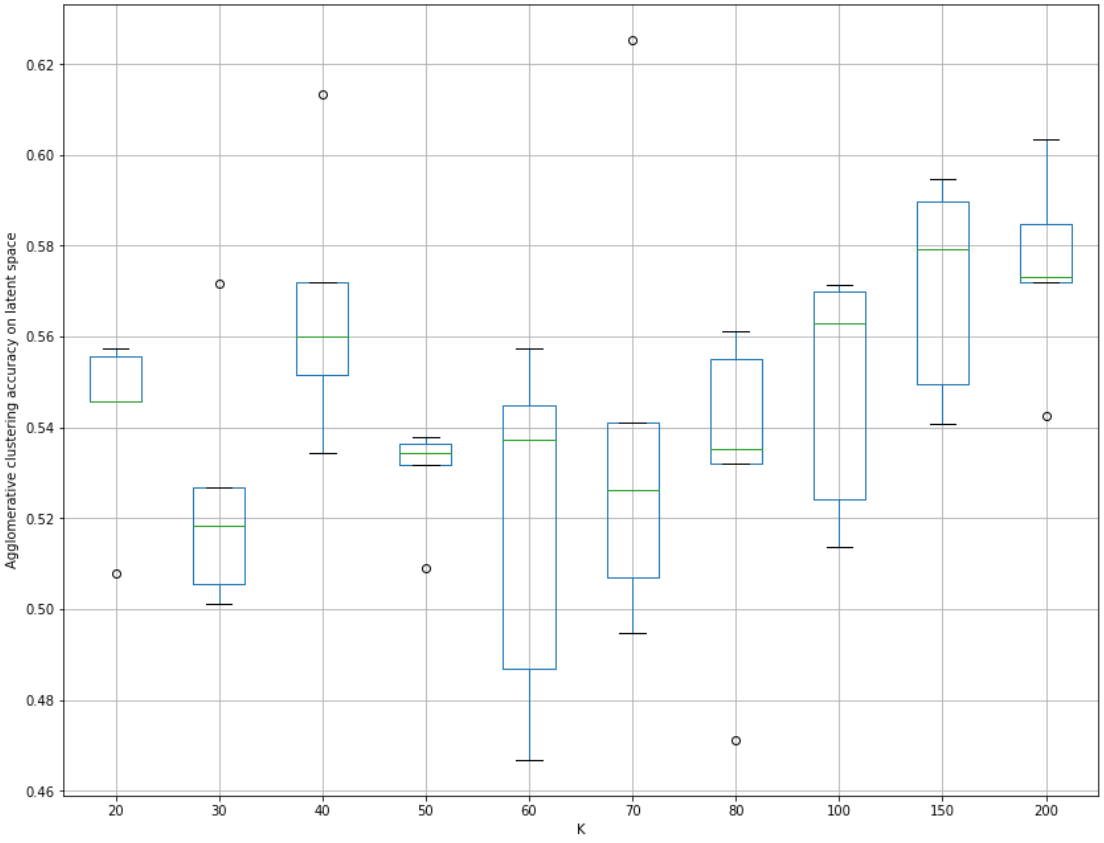
CNN-deep-network-without-centroids clustering method parameters for fashion-MNIST dataset

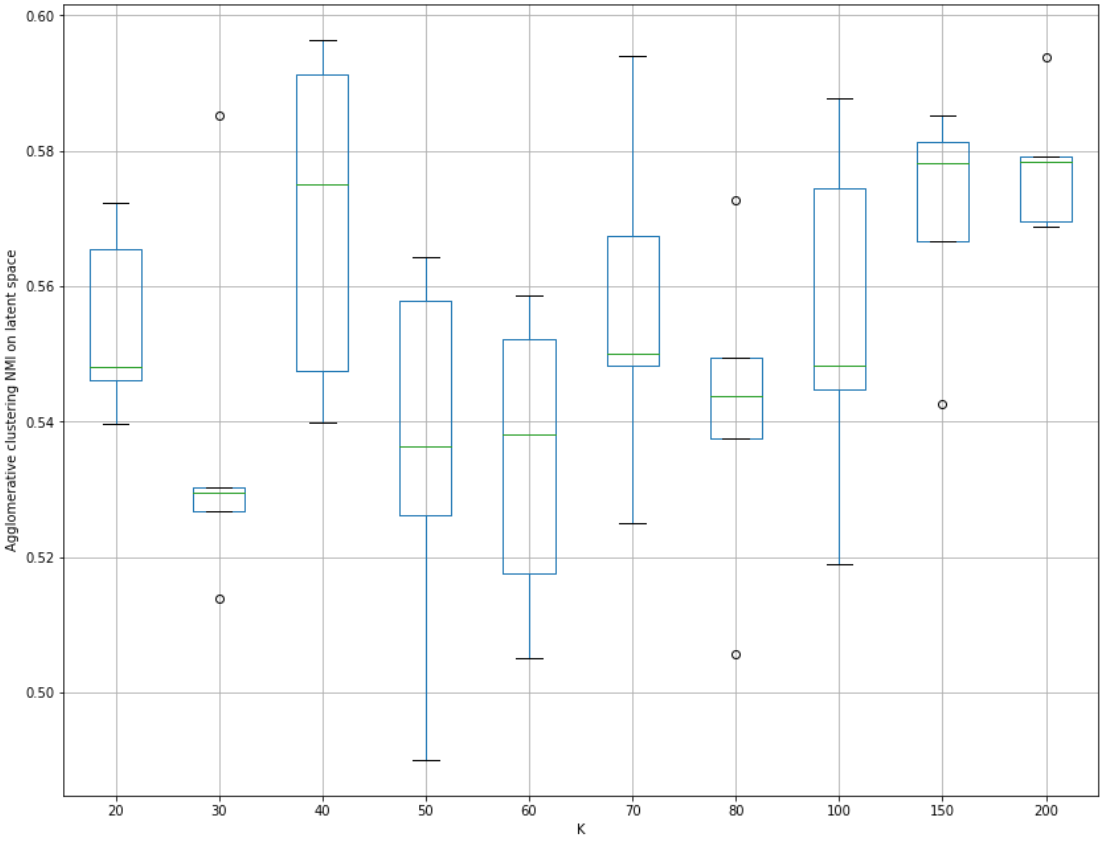
Boxplots:

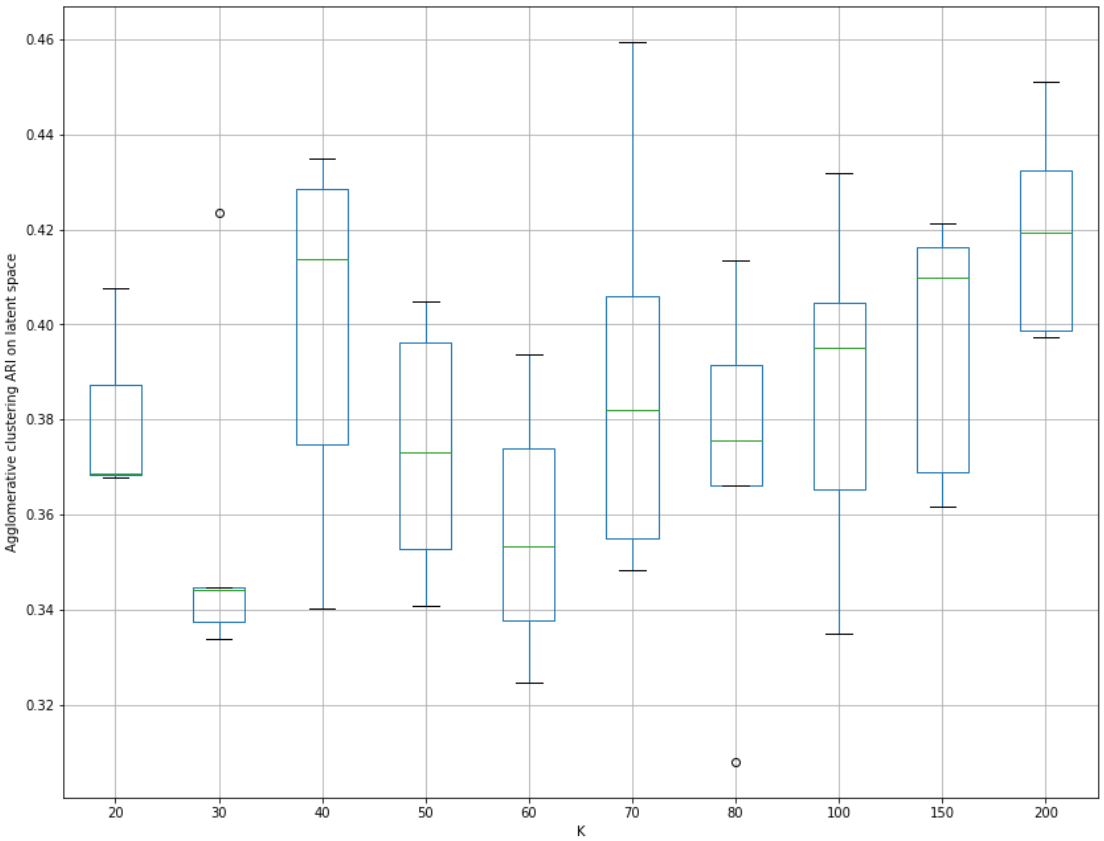

K-means accuracy on latent space
0.58
0.56
0.54
0.52
0.50
20
30
40
50
60
70
80
100
150
200
K

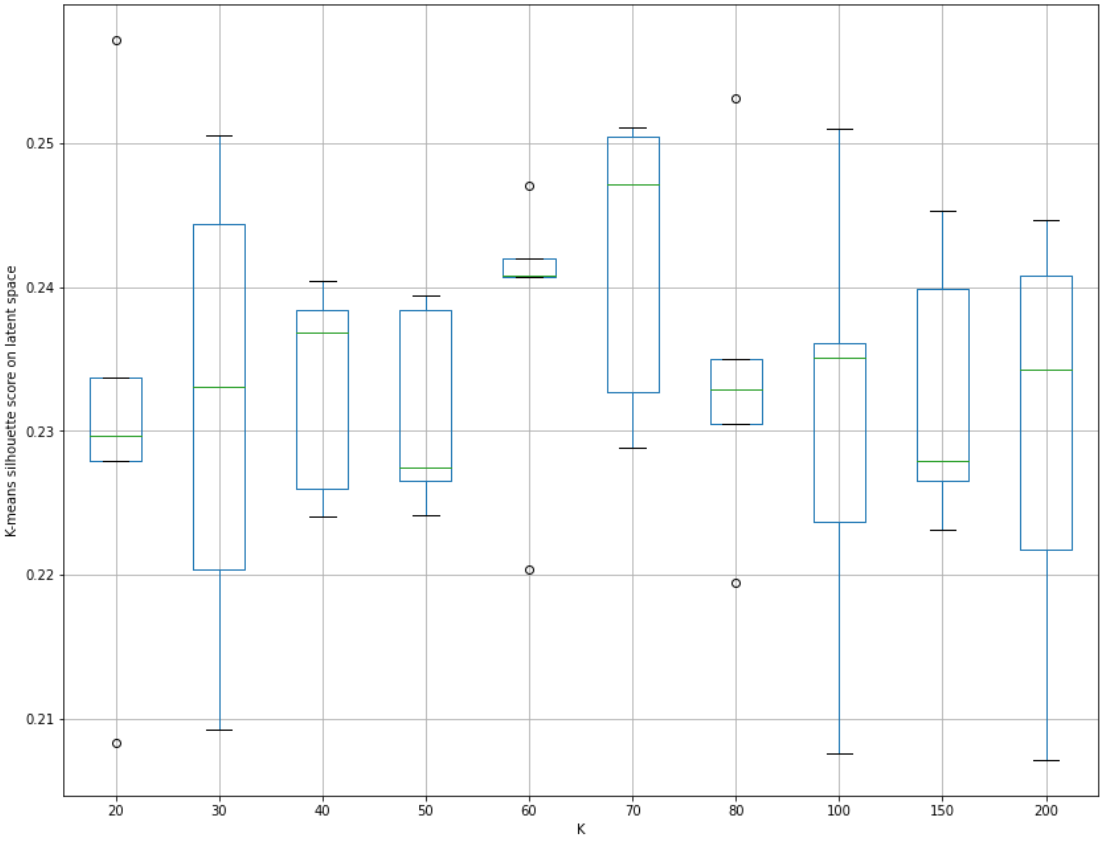

K-means NMI on latent space
0.58
0.56
0.54
0.52
0.50
20
30
40
50
60
70
80
100
150
200
K

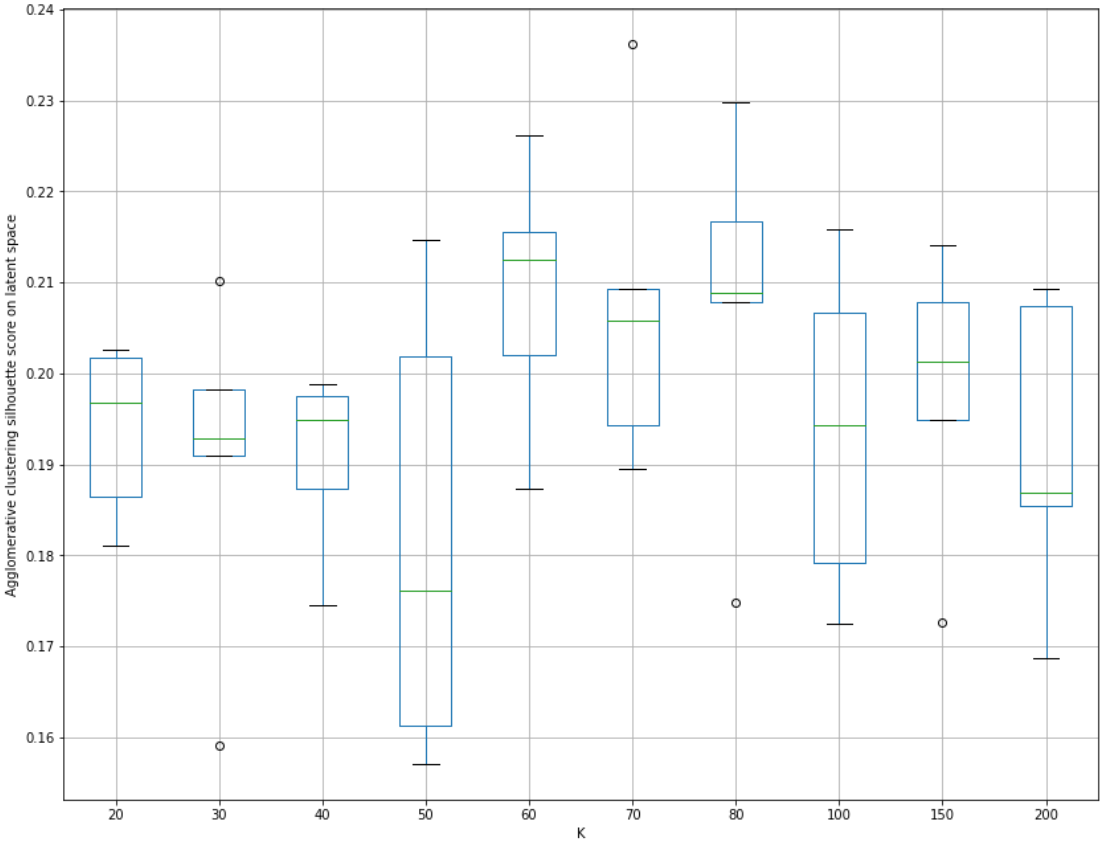

K-means ARI on latent space
0.42
0.40
0.38
0.36
0.34
20
30
40
50
60
70
80
100
150
200
K


Agglomerative clustering accuracy on latent space
0.62
0.60
0.58
0.56
0.54
0.52
0.50
0.48
0.46
20
30
40
50
60
70
80
100
150
200
K

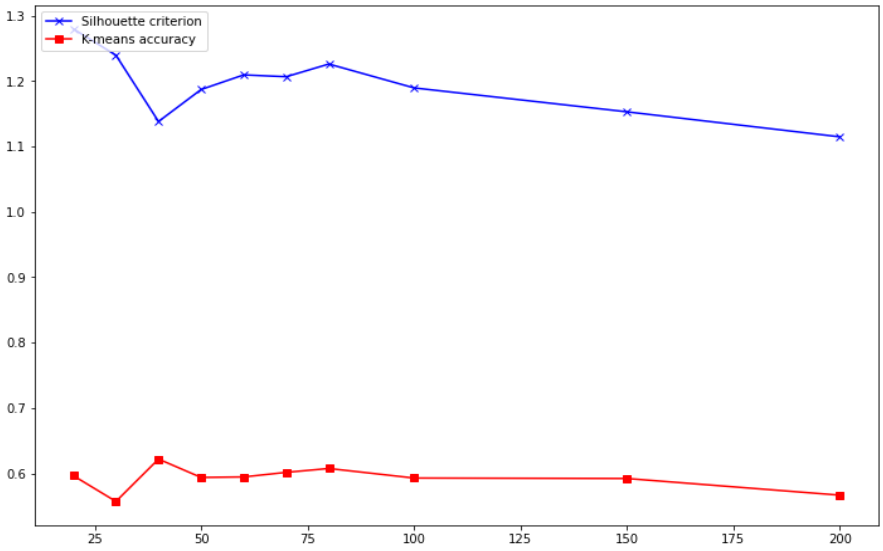

Agglomerative clustering NMI on latent space
0.60
0.58
0.56
0.54
0.52
0.50
20
30
40
50
60
70
80
100
150
200
K


Agglomerative clustering ARI on latent space
0.46
0.44
0.42
0.40
0.38
0.36
0.34
0.32
20
30
40
50
60
70
80
100
150
200
K


K-means silhouette score on latent space
0.25
0.24
0.23
0.22
0.21
20
30
40
50
60
70
80
100
150
200
K


Agglomerative clustering silhouette score on latent space
0.24
0.23
0.22
0.21
0.20
0.19
0.18
0.17
0.16
20
30
40
50
60
70
80
100
150
200
K

Overclustering criterion:



Comments: It is obvious that not using the centroids during the training of the model doesn't improve clustering performance. This is another case where using the centroids during training makes the difference. The overclustering criterion chooses option k = 20, which is average in performance amongst its competitors.

3.6 Table of metrics for each clustering method

In this section, we have concentrated the mean and standard deviation values for each clustering evaluation metric and for every clustering method we used (apart from the CNN-autoencoder method which applies to images) on the datasets. We compare the results on the various datasets and mark with bold the results for the best deep-clustering method. The results are presented in the form of <mean value±standard deviation>.

Dataset	Method	K-means accuracy	K-means NMI	K-means ARI	K-means silhouette coefficient
10x_73k	Initial space	0.60	0.55	0.37	0.05
	MLP-transformed space	0.74±0.02	0.63±0.01	0.54±0.02	0.20±0.01
	Autoencoder without centroids	0.75±0.04	0.71±0.01	0.58±0.03	0.36±0.01
	Autoencoder with centroids	**0.92±0.02**	**0.85±0.01**	**0.85±0.03**	**0.62±0.01**
	Initial space	**0.86**	0.42	**0.52**	**0.50**

Australian	MLP-transformed space	0.86±0	**0.43±0**	0.52±0	0.33±0.02
	Autoencoder without centroids	0.84±0.04	0.37±0.09	0.45±0.08	0.30±0.02
	Autoencoder with centroids	0.73±0.11	0.22±0.22	0.23±0.22	0.37±0.08
Iris	Initial space	**0.89**	**0.74**	**0.72**	**0.50**
	MLP-transformed space	0.85±0.02	0.66±0.04	0.64±0.04	0.35±0.30
	Autoencoder without centroids	0.85±0.02	0.68±0.04	0.64±0.04	0.48±0.02
	Autoencoder with centroids	0.80±0.11	0.62±0.52	0.58±0.11	0.50±0.07
Squeezed gaussian blobs	Initial space	0.50	0	0	0.48
	MLP-transformed space	0.58±0.06	0.45±0.43	0.44±0.42	0.44±0.07
	Autoencoder without centroids	0.73±0.25	0.40±0.33	0.40±0.33	0.70±0.13
	Autoencoder with centroids	**0.96±0.04**	**0.80±0.19**	**0.85±0.11**	**0.85±0.09**
Gaussian rings	Initial space	0.50	0	0	0.30
	MLP-transformed space	0.68±0.11	0.30±0.15	0.23±0.19	0.60±0.02
	Autoencoder without centroids	0.74±0.01	0.32±0.01	0.23±0.09	0.60±0.02
	Autoencoder with centroids	**0.98±0.02**	**0.90±0.08**	**0.91±0.08**	**0.82±0.03**
Moons	Initial space	0.86	0.42	0.52	0.50
	MLP-transformed space	0.71±0.17	0.39±0.17	0.40±0.12	0.44±0.07
	Autoencoder without centroids	0.91±0.01	0.56±0.02	0.67±0.03	0.63±0.01
	Autoencoder with centroids	**1±0**	**1±0**	**1±0**	**0.91±0.01**
Pendigits	Initial space	0.71	0.69	0.55	0.31
	MLP-transformed space	0.76±0.01	0.71±0.01	0.63±0.02	0.35±0.02
	Autoencoder without centroids	0.76±0.02	0.69±0.01	0.60±0.02	0.31±0.02
	Autoencoder with centroids	**0.89±0.08**	**0.83±0.07**	**0.79±0.07**	**0.57±0.02**

MNIST	Initial space	0.56	0.49	0.36	0.07
	MLP-transformed space (by reshaping the images into vectors)	0.59±0.01	0.52±0.01	0.41±0.01	0.12±0.01
	CNN-autoencoder without centroids	0.69±0.02	0.61±0.02	0.53±0.02	0.19±0.01
	CNN-autoencoder with centroids	**0.88±0.01**	**0.77±0.01**	**0.76±0.02**	**0.66±0.01**
Fashion-MNIST	Initial space	0.55	0.53	0.40	0.14
	MLP-transformed space (by reshaping the images into vectors)	0.62±0.03	0.56±0.02	0.42±0.03	0.16±0.01
	CNN-autoencoder without centroids	0.57±0.01	0.55±0.02	0.39±0.02	0.23±0.01
	CNN-autoencoder with centroids	**0.74±0.02**	**0.67±0.02**	**0.58±0.01**	**0.48±0.01**

Table of k-means clustering metrics for each dataset

Dataset	Method	Agglomerative clustering accuracy	Agglomerative clustering NMI	Agglomerative clustering ARI	Agglomerative clustering silhouette coefficient
10x_73k	Initial space	0.63	0.61	0.50	0.04
	MLP-transformed space	0.75±0.30	0.68±0.15	0.55±0.52	0.17±0.01
	Autoencoder without centroids	0.73±0.02	0.74±0.02	0.61±0.04	0.34±0.01
	Autoencoder with centroids	**0.92±0.07**	**0.85±0.07**	**0.85±0.07**	**0.62±0.01**
Australian	Initial space	0.85	**0.43**	0.43	0.24
	MLP-transformed space	**0.86±0**	0.43±0	**0.50±0**	0.30±0.02
	Autoencoder without centroids	0.84±0.02	0.41±0.03	0.48±0.05	0.30±0.04
	Autoencoder with centroids	0.75±0.04	0.20±0.02	0.25±0.08	**0.35±0.07**
	Initial space	**0.89**	**0.78**	**0.72**	**0.50**
	MLP-transformed space	0.84±0.06	0.67±0.06	0.63±0.07	0.33±0.02

Iris	Autoencoder without centroids	0.77±0.03	0.65±0.03	0.56±0.02	0.4±0.04
	Autoencoder with centroids	0.75±0.07	0.62±0.04	0.55±0.06	0.48±0.08
Squeezed gaussian blobs	Initial space	0.55	0.01	0.01	0.47
	MLP-transformed space	0.82±0.16	0.55±0.33	0.57±0.39	0.43±0.08
	Autoencoder without centroids	0.69±0.21	0.4±0.35	0.4±0.35	0.59±0.10
	Autoencoder with centroids	**0.95±0.05**	**0.84±0.15**	**0.86±0.13**	**0.85±0.07**
Gaussian rings	Initial space	0.71	0.29	0.18	0.36
	MLP-transformed space	0.71±0.06	0.30±0.11	0.20±0.11	0.62±0.03
	Autoencoder without centroids	0.73±0.01	0.32±0.01	0.22±0.01	0.59±0.01
	Autoencoder with centroids	**0.98±0.01**	**0.9±0.08**	**0.92±0.07**	**0.82±0.02**
Moons	Initial space	0.82	0.45	0.41	0.42
	MLP-transformed space	0.85±0.12	0.50±0.225	0.46±0.26	0.43±0.08
	Autoencoder without centroids	0.89±0.01	0.57±0.01	0.6±0.004	0.61±0.01
	Autoencoder with centroids	**1±0**	**1±0**	**1±0**	**0.92±0.01**
Pendigits	Initial space	0.75	0.75	0.61	0.30
	MLP-transformed space	0.78±0.01	0.77±0.02	0.67±0.02	0.33±0.02
	Autoencoder without centroids	0.77±0.03	0.75±0.01	0.65±0.04	0.27±0.01
	Autoencoder with centroids	**0.89±0.01**	**0.85±0.08**	**0.79±0.01**	**0.58±0.02**
MNIST	Initial space	0.66	0.66	0.49	0.04
	MLP-transformed space (by reshaping the images into vectors)	0.65±0.01	0.61±0.01	0.49±0.02	0.1±0.01
	CNN-autoencoder without centroids	0.72±0.03	0.7±0.02	0.57±0.03	0.15±0.01
	CNN-autoencoder with centroids	**0.88±0.01**	**0.78±0.01**	**0.78±0.01**	**0.66±0.01**

Fashion-MNIST	Initial space	0.66	0.62	0.49	0.12
	MLP-transformed space (by reshaping the images into vectors)	0.57±0.01	0.56±0.02	0.4±0.02	0.13±0.01
	CNN-autoencoder without centroids	0.58±0.01	0.57±0.01	0.42±0.01	0.19±0.01
	CNN-autoencoder with centroids	**0.75±0.02**	**0.68±0.01**	**0.58±0.02**	**0.47±0.01**

Table of agglomerative clustering metrics for each dataset

3.7 t-SNE representation of initial and latent spaces

As mentioned in the introduction, t-SNE can produce 2-dimensional representations of high dimensional data. We use it to validate if the data calculated in the 2D space are cluster friendly.

In our experiments, for each dataset, we have extracted a t-SNE representation of the datapoints in the initial space and the datapoints in the latent space. It is interesting to note how the t-SNE representation in the latent space for datasets in which our method has improved clustering results appears to be much more cluster friendly, whereas in datasets where our method hasn't performed well, the inverse is true.

10x_73k dataset:

1) **Initial space**

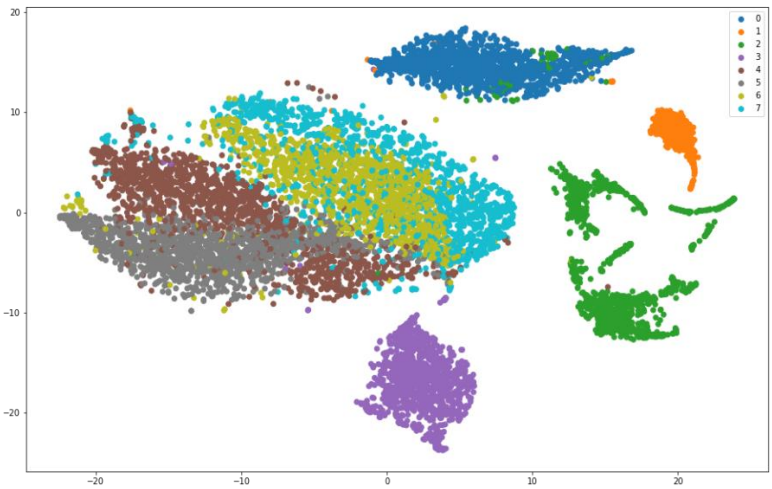


Figure 18: 10x_73k dataset initial space t-SNE representation

2) Latent space

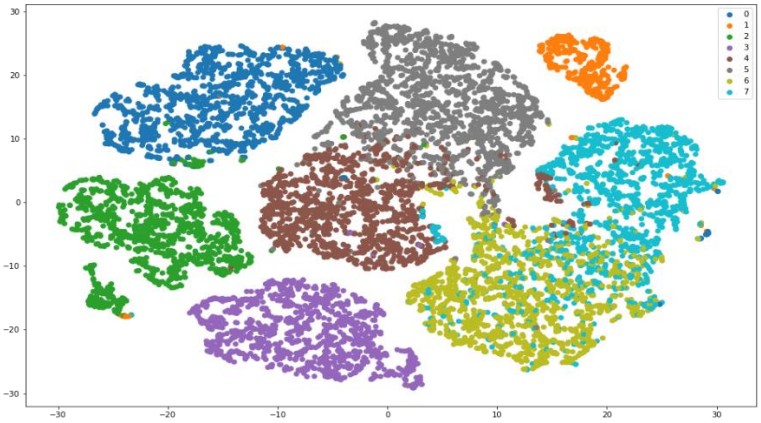


Figure 19: 10x_73k dataset latent space t-SNE representation

The t-SNE representation in the latent space seems to suggest that datapoints colored grey, brown, light blue and olive oil, which were clattered together in the initial space, are uncluttered in the latent space, forming almost distinct clusters. The green data points have also come closer together. This helps visualise the improvement in the data space and explains the increase in all clustering metrics.

Pendigits dataset:

1) Initial space

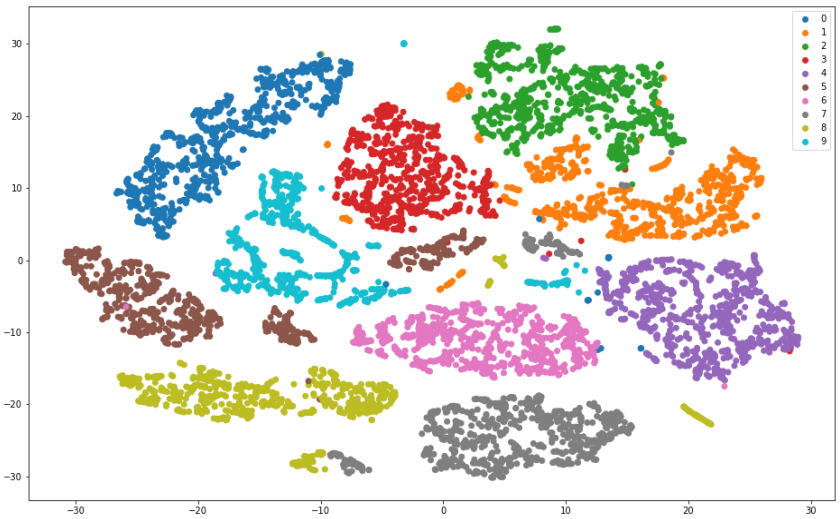


Figure 20: Pendigits dataset initial space t-SNE representation

2) Latent space

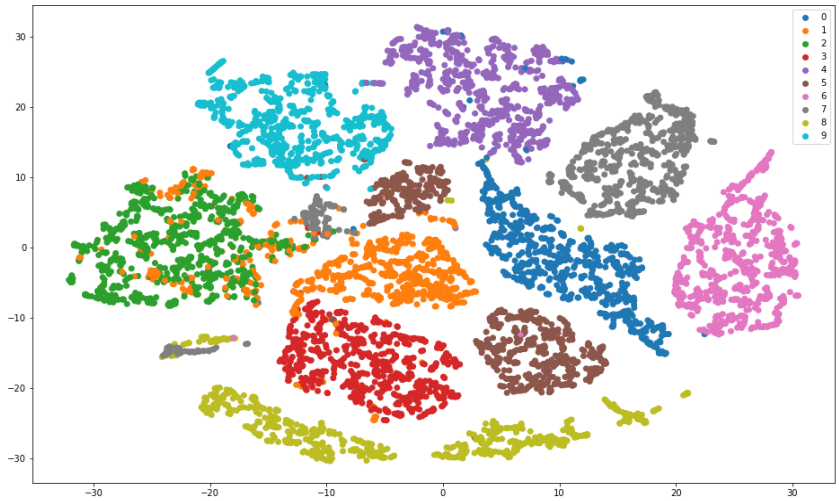


Figure 21: Pendigits dataset latent space t-SNE representation

The difference here is not obvious, but it can be argued that each coloured cluster of data points has become a little bit denser in the latent space. Two subclusters of brown have merged. The light blue cluster has pulled its datapoints closer.

Gaussian rings dataset:

1) Initial space

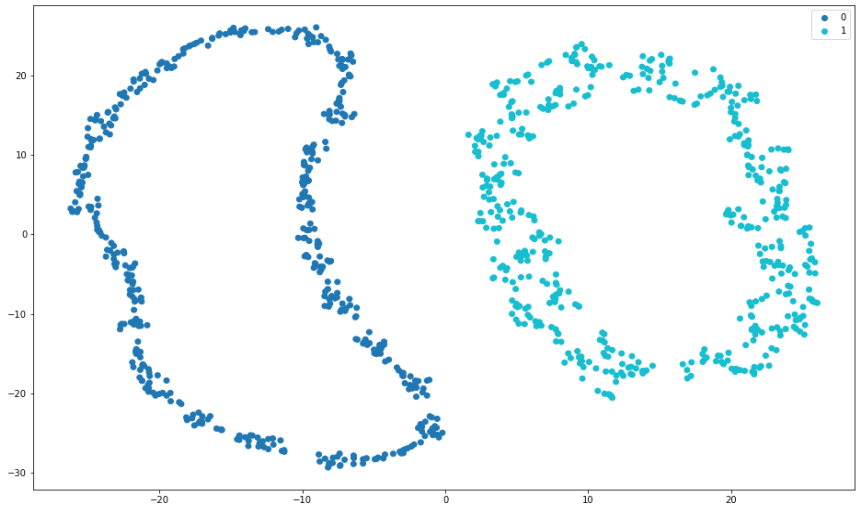


Figure 22: Gaussian rings dataset initial space t-SNE representation

2) Latent space

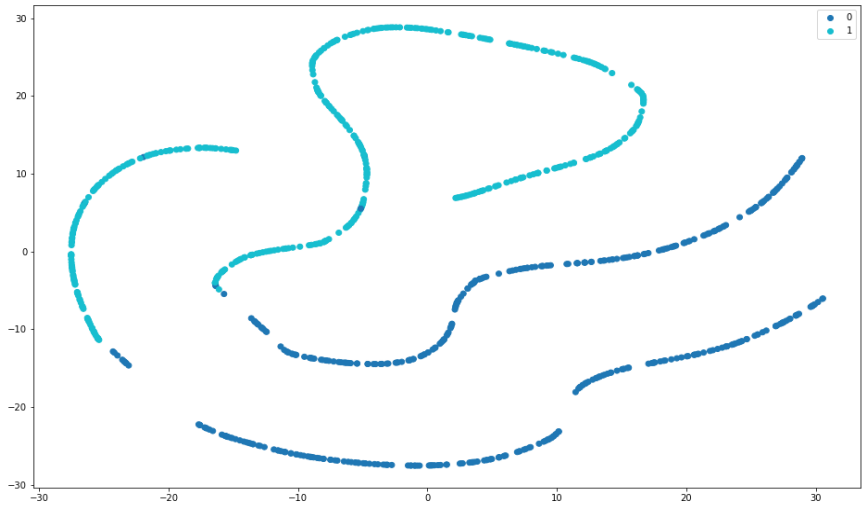


Figure 23: Gaussian rings dataset latent space t-SNE representation

What is interesting in latent space is that the rings have become curved lines with no difference in width. Also, the latent space seems to be much friendlier to a k-means algorithm run.

Squeezed gaussian blobs dataset:

1) Initial space

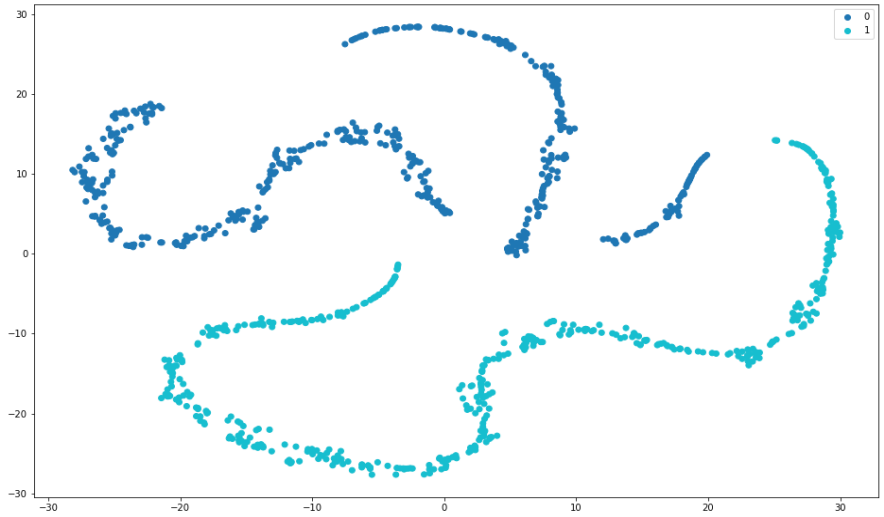


Figure 24: Squeezed gaussian blobs dataset initial space t-SNE representation

2) Latent space

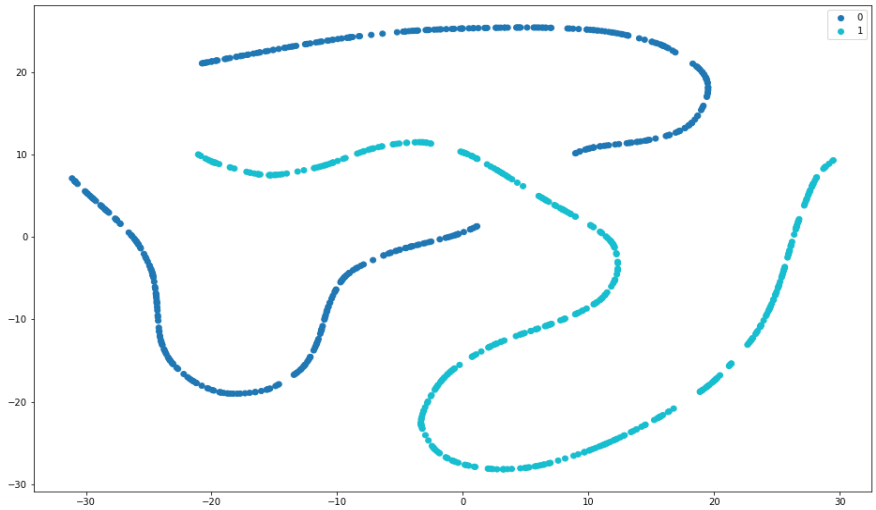


Figure 25: Squeezed gaussian blobs dataset latent space t-SNE representation

What is interesting in latent space is that the lines now have almost zero width, as the datapoints are behind one another. Also, the latent space seems to be much friendlier to a k-means algorithm run

Moons dataset:

1) Initial space

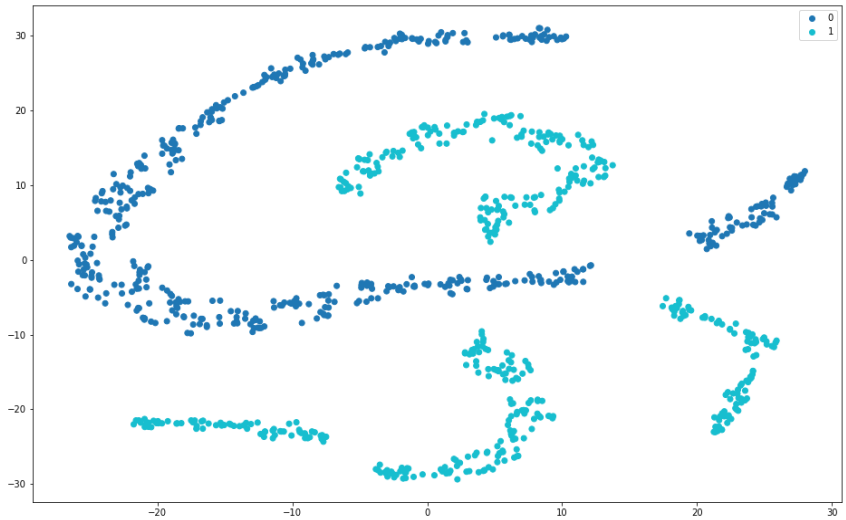


Figure 26: Moons dataset initial space t-SNE representation

2) Latent space

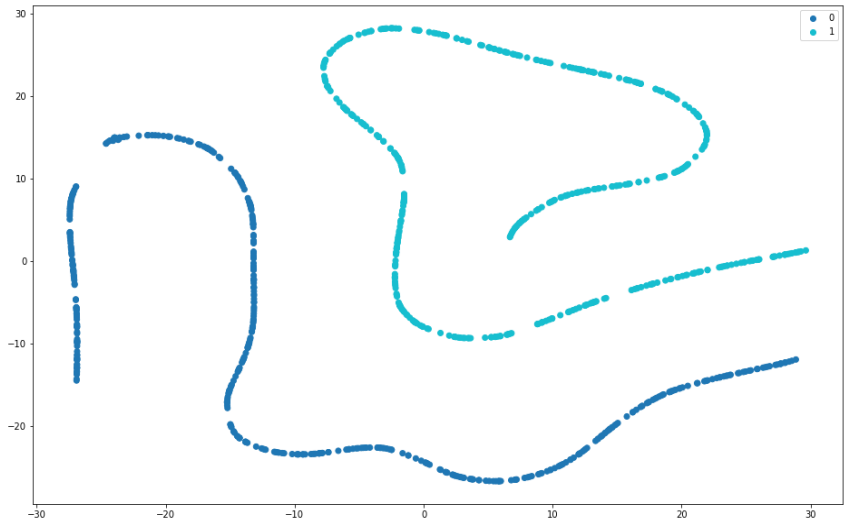


Figure 27: Moons dataset latent space t-SNE representation

The datapoints of each label have come closer together in the latent space and formed distinct clusters. Moreover, we notice the same width removal in the lines that we noticed in the gaussian rings and squeezed gaussian blobs dataset.

Iris dataset:

1) Initial space

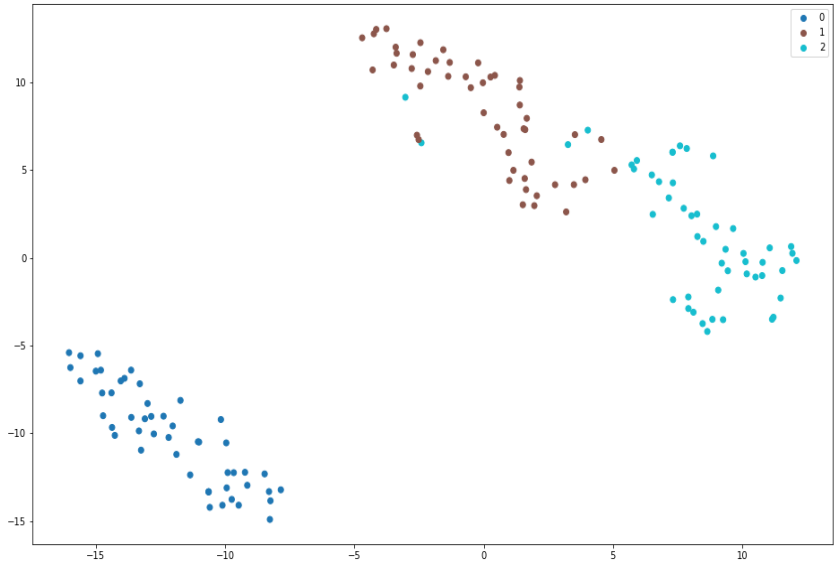


Figure 28: Iris dataset initial space t-SNE representation

2) Latent space

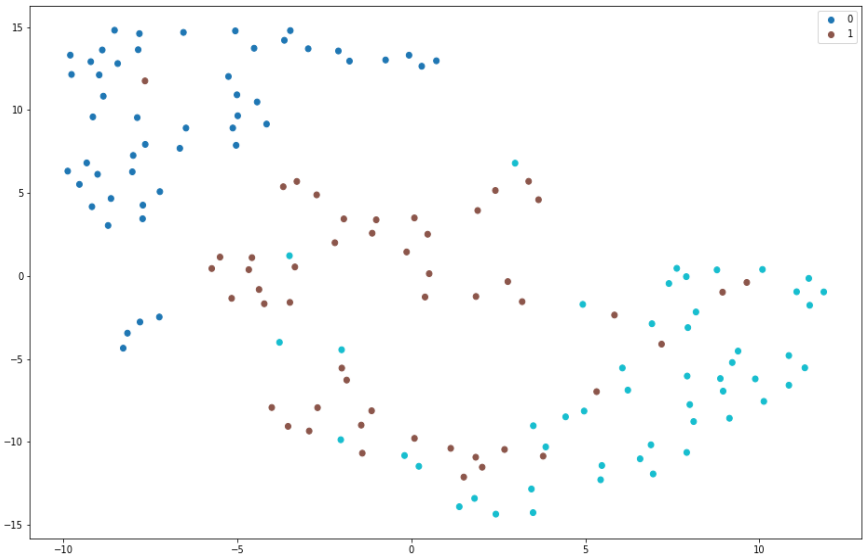


Figure 29: Iris dataset latent space t-SNE representation

In the Iris dataset we had a decrease in performance using our methods and the t-SNE representation of the latent space depicts why. The datapoints of the same label have become more distant from each other and the clusters have become entangled.

Australian dataset:

1) Initial space

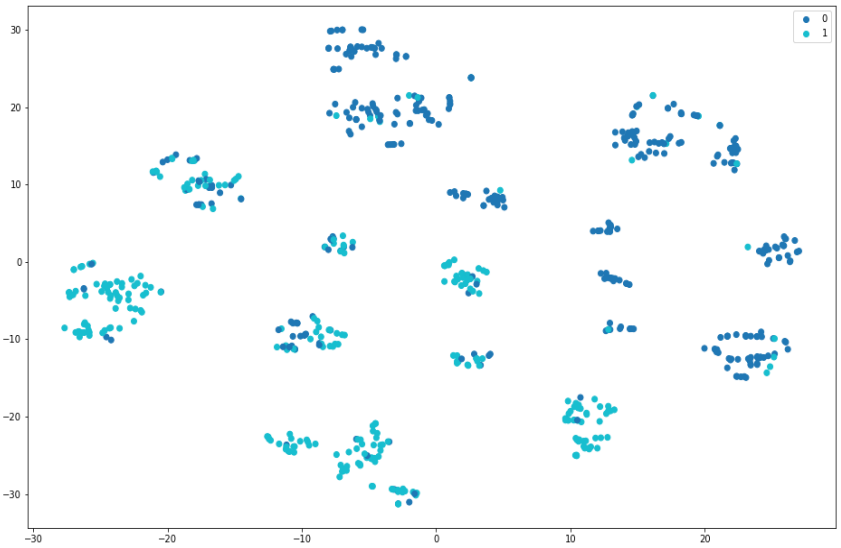


Figure 30: Australian dataset initial space t-SNE representation

2) Latent space

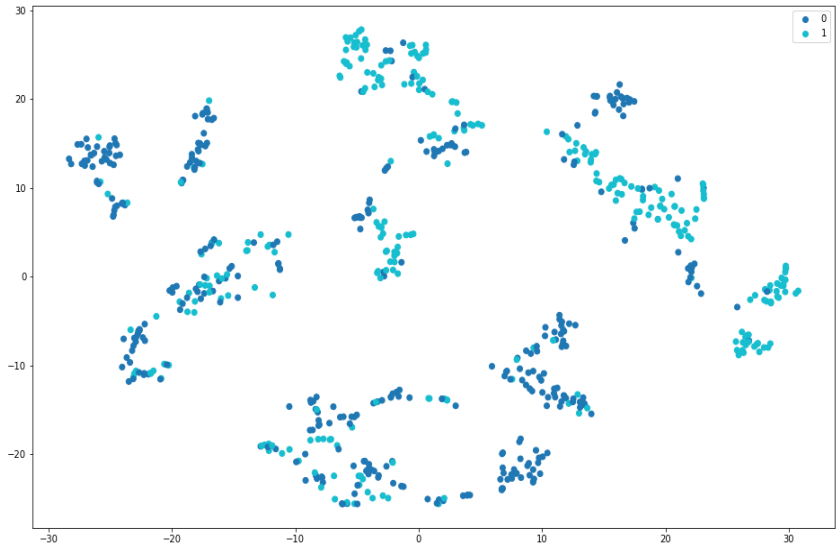


Figure 31: Australian dataset latent space t-SNE representation

In the Australian dataset we had a decrease in performance using our methods and the t-SNE representation of the latent space depicts why. The clusters have become more entangled.

MNIST dataset:

1) Initial space

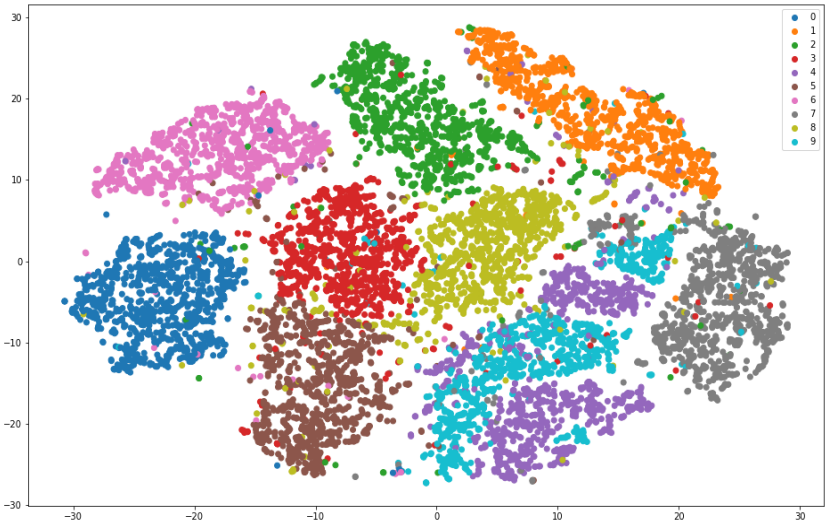


Figure 32: MNIST dataset initial space t-SNE representation

2) Latent space

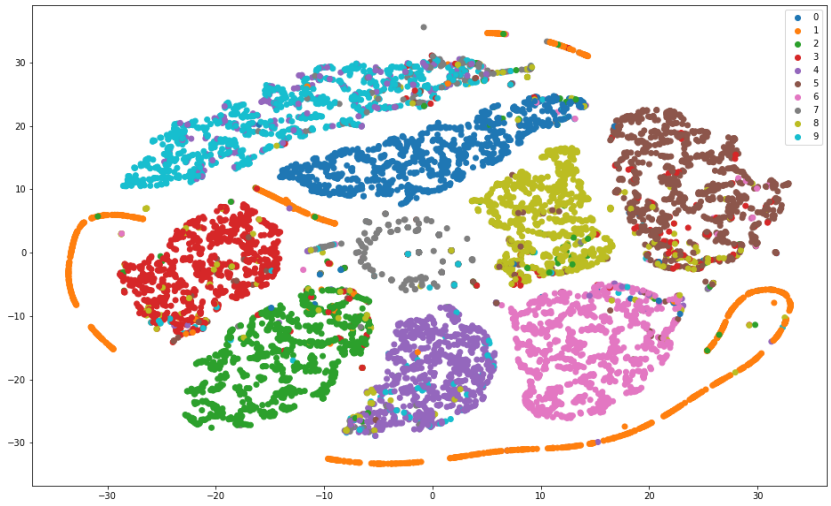


Figure 33: MNIST dataset latent space t-SNE representation

The improvement in the data space that is evident from the boxplots for this dataset, and that was achieved using the CNN – autoencoder architecture, is depicted also in the above t-SNE representations. In the initial space, almost all the clusters are really close to each other, and some invade others' territory. The light blue and purple clusters are entangled, while some data points from the red cluster have been scattered far from the red circle that represents the majority. In contrast, the latent space offers much more distinct clusters. The light blue and purple clusters have their own territory, despite some outliers, while the red cluster has pulled most of its data points closer to the circle.

Fashion-MNIST dataset:

1) Initial space

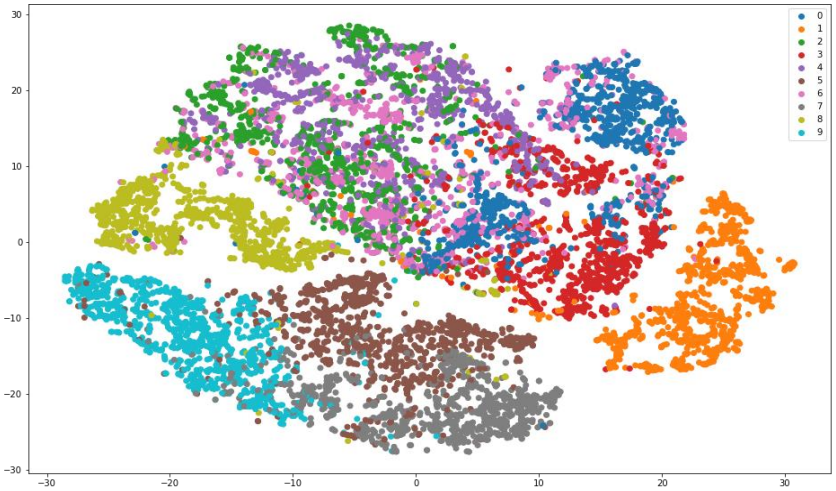


Figure 34: Fashion-MNIST dataset initial space t-SNE representation

2) Latent space

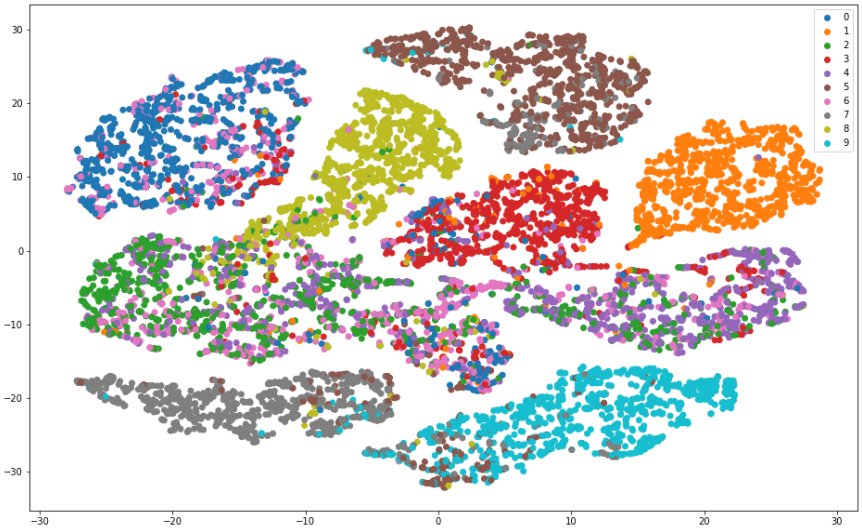


Figure 35: Fashion-MNIST dataset latent space t-SNE representation

The improvement in the fashion-MNIST that is evident from the boxplots for this dataset, and that was achieved using the CNN – autoencoder architecture, is depicted also in the above t-SNE representations. In the initial space, the purple, green, blue, pink and red clusters are entangled, making it difficult for the traditional clustering algorithms to work well. Clusters light blue, brown and grey are also very close to each other. To some extent, these issues have been resolved in the latent space, although improvements are still feasible. Considering the k – means accuracy went roughly from 0.5 to 0.7 (the latter score concerning the latent space coming from one of the best choices of k), the t-SNE representation gives us a solid estimation of the real picture once again.

4. Conclusions and future work

4.1 Conclusion

The experimental study showed that Deep Clustering methods are a significant improvement on traditional clustering algorithms, especially when the data is complex and high-dimensional. Having tested the proposed methods on various datasets, we notice that performance is enhanced in most cases. We have noticed that the parameters of the architectures we use play a vital role in the performance of our methods. For example, different choices in activation functions, optimisers, number of layers, number of neurons in each layer and datapoints normalisation can give us vastly different results in the clustering metrics.

We proved that the concept of training the autoencoder using the cluster centroids is a procedure that improves the data space in practice. By taking a look at the table of metrics in section 3.6, one can notice that the hidden representations of Multilayer Perceptrons enhanced the clustering performance of the traditional algorithms, as did the latent representations of autoencoders trained without the centroids. But the extent of the enhancement is, more often than not, not as significant as our concept of training the autoencoder using the centroids. From the centroid-trained autoencoders we extracted, in some cases, representations that improved clustering performance to an extent that, to the best of our knowledge, has not been reached before. A great example is the autoencoder-trained-with-centroids method's performance in the 10x_73k dataset.

Spaces that are cluster-friendly generally provide us with cluster-friendly two-dimensional t-SNE representations as well. In cases that we achieved massive improvement in clustering performance, the t-SNE representation of the latent space had clear groups of data points that formed distinct clusters.

The CNN deep network architectures we tried worked particularly well for the MNIST and Fashion-MNIST datasets. Those experiments were the only ones in which the increase in the number of centroids always resulted in enhanced clustering performance in the latent space. In other datasets, choosing too many centroids sometimes lowered the performance.

Last but not least, the overclustering criterion we introduced for picking the initial number of clusters, while not perfect, is a good estimation for the overclustering hyperparameter K. However, its highest-ranked suggestion is sometimes misleading, therefore one would be better off taking into consideration the collection of, for example, the 3 highest-ranked options the criterion suggests.

4.2 Suggestions for future work

Since (latent and MLP-transformed) spaces that are cluster-friendlier are associated with cluster-friendlier two-dimensional t-SNE representations, we put forward the idea of running the traditional clustering algorithms on the t-SNE 2-dimensional data to assess their performance there. The dimensionality reduction of the latent and MLP-transformed spaces could be further improved by concentrating all the important information in just two or three dimensions and help the traditional algorithms even more.

Bibliography

[1] Heaton, J. (2018). Ian goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning., pp. 96 - 104, 2016, available at http://www.deeplearningbook.org

[2] Diday, E., & Simon, J. C. (1976). Clustering analysis. In *Digital pattern recognition* (pp. 47-94). Springer, Berlin, Heidelberg.

[3] Algorithms for clustering data, Anil K. Jain and Richard C. Dube, Prentice-Hall, Inc., Division of Simon and Schuster One Lake Street Upper Saddle River, NJ, United States, pp. 58 – 60.

[4] Adams, R. P. (2009). Hierarchical Agglomerative Clustering. In *Proc. Ninth SIAM Data Mining Conf.(SDM'09),* (pp. 510-516).

[5] Wu, J. (2017). Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, *5*(23), 495.

[6] Delashmit, W. H., & Manry, M. T. (2005, May). Recent developments in multilayer perceptron neural networks. In *Proceedings of the seventh Annual Memphis Area Engineering and Science Conference, MAESC*.

[7] Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research*, *9*(11), pp. 2-6.

[8] McDaid, A. F., Greene, D., & Hurley, N. (2011). Normalized mutual information to evaluate overlapping community finding algorithms. *arXiv preprint arXiv:1110.2515*.

[9] Shutaywi, M., & Kachouie, N. N. (2021). Silhouette analysis for performance evaluation in machine learning with applications to clustering. *Entropy*, *23*(6), 759.

[10] Alsabti, K., Ranka, S., & Singh, V. (1997). An efficient k-means clustering algorithm.

[11] Z. Zhang, "Improved Adam Optimizer for Deep Neural Networks," *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018, pp. 1-2, doi: 10.1109/IWQoS.2018.8624183.

[12] Dunne, R. A., & Campbell, N. A. (1997, June). On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne* (Vol. 181, p. 185). Citeseer.

[13] Dubey, A. K., & Jain, V. (2019). Comparative study of convolution neural network's relu and leaky-relu activation functions. In *Applications of Computing, Automation and Wireless Systems in Electrical Engineering* (pp. 873-880). Springer, Singapore.

[14] Yeung, K. Y., & Ruzzo, W. L. (2001). Details of the adjusted rand index and clustering algorithms, supplement to the paper an empirical study on principal component analysis for clustering gene expression data. *Bioinformatics*, *17*(9), 763-774.

[15] Sokolova, M., Japkowicz, N., & Szpakowicz, S. (2006, December). Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation. In *Australasian joint conference on artificial intelligence* (pp. 1015-1021). Springer, Berlin, Heidelberg.

[16] Dua, Dheeru and Graff, Casey, 2017, {UCI} Machine Learning Repository, http://archive.ics.uci.edu/ml/datasets/pen-based+recognition+of+handwritten+digits , University of California, Irvine, School of Information and Computer Sciences

[17] Dua, Dheeru and Graff, Casey, 2017, {UCI} Machine Learning Repository, https://archive.ics.uci.edu/ml/datasets/statlog+(australian+credit+approval) , University of California, Irvine, School of Information and Computer Sciences

[18] Kanal, L. N. (2003). Perceptron. In *Encyclopedia of Computer Science* (pp. 1383-1385).

[19]The European Genome-phenome archive (EGA), dataset: EGAD00001005161, https://ega-archive.org/datasets/EGAD00001005161/files

[20] MNISTdatabase. (2022,July 31). In *Wikipedia*. Available at https://en.wikipedia.org/wiki/MNIST_database

[21] Fashion MNIST. (2022, July 28). In *Wikipedia*, available at https://en.wikipedia.org/wiki/Fashion_MNIST

[22] Srinath, K. R. (2017). Python–the fastest growing programming language. *International Research Journal of Engineering and Technology*, *4*(12), 354-357.

[23] Stevens, E., Antiga, L., & Viehmann, T. (2020). *Deep learning with PyTorch*. Manning Publications.

[24] Oliphant, T. E. (2006). *A guide to NumPy* (Vol. 1, p. 85). USA: Trelgol Publishing.

[25] Kramer, O. (2016). Scikit-learn. In *Machine learning for evolution strategies* (pp. 45-53). Springer, Cham.

[26] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... & Van Mulbregt, P. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*, *17*(3), 261-272.

[27] McKinney, W. (2011). pandas: a foundational Python library for data analysis and statistics. *Python for high performance and scientific computing*, *14*(9), 1-9.

[28] Tosi, S. (2009). *Matplotlib for Python developers*. Packt Publishing Ltd.

[29] Nutakki, G. C., Abdollahi, B., Sun, W., & Nasraoui, O. (2019). An introduction to deep clustering. In *Clustering Methods for Big Data Analytics* (pp. 73-89). Springer, Cham.

Appendix

The code for this research was written in Python [22] programming language. The following python libraries are necessary for running every aspect of the research that was coded:

• PyTorch [23] is an open-source machine learning framework. It is especially flexible in constructing neural networks for machine learning applications.

• Numpy [24] library is fundamental package for scientific computations using Python. It also enables the user to create matrices and manipulate them.

• Scikit-learn [25] is a library used in machine learning and data science applications.

• SciPy [26] is an open-source library used for mathematics, science, and engineering.

• Pandas [27] is a fast, powerful, flexible, and easy to use open-source data analysis and manipulation tool.

• Matplotlib [28] is a comprehensive library for creating static, animated, and interactive visualizations.