Cold start is coming: How to approximate the optimal set of initial prototypes for clustering sequence data online

Silviu Fucarev[1] , **Azqa Nadeem**[1] , **Sicco Verwer**[1]
[1]TU Delft

Abstract

Clustering data is a classic topic in the academic community and in the industry. It is by and large one of the most popular unsupervised classification techniques. It is fast and flexible as it can accommodate all kinds of data when a suitable similarity metric is found. SeqClu is an online k-medoids prototype based clustering algorithm designed to handle large quantities of sequence data. Our main focus is the role initialization plays in the performance of SeqClu. In this paper we show that Greedy Heuristics perform significantly better than K-medoids heuristics. In the context of Greedy Heuristics we show that these can be combined together to achieve potentially better accuracy if a proper metric to choose the initialization results is elected.

1 Introduction

Clustering is a ubiquitous method for classifying data in an unsupervised fashion. It allows forming groups and structures from unlabeled data while revealing the inner structure of a dataset. The main idea is to leverage the concept of similarity and use methods of measuring similarity in order to group the elements of a data source.

This simple idea has given birth to a plethora of algorithms which are widely used in industry and academia for tasks linked to classification, recommendation etc. A few examples are k-means, k-medoids, hierarchical clustering. These have been proven to be able to consistently solve clustering tasks whenever the whole dataset fits in memory. For instances where the dataset is too big, online versions of these algorithms have been developed and have been shown to perform reasonably well.[1]

Despite all the positives listed above any clustering algorithm has a weakness, the cold start. This phenomenon happens when the algorithm cannot cluster any point because it has no cluster structure information. It is accepted that the optimality of the initialization has high influence over the optimality of the whole clustering. To this end many heuristics have emerged however there still is no common ground on which one to use and when. K-means uses random initialization, maximin, k-means++ or any of the above listed heuristics with repetition to minimize the chance of an unfortunate initialization. In the offline context clustering algorithms recalculate their representative points a number of times proportional to their number of max iterations, a luxury which an online clustering algorithm does not have. So to be able to approximate an offline clustering, the online algorithm needs to select good initialization from the start and use only a small sample of the data.

The case of SeqClu is similar to most online clustering algorithms built on top of k-medoids, with the exception that to approximate a cluster, we use 5 prototypes instead of one medoid. While this contributes to a more stable clustering result, it also adds a layer of complexity as the number of initial points we have to find is multiplied by a factor of 5.

Research Question

We want to research how classic offline initialization heuristics approximate the optimal initial prototype sets. We will study the influence of hyperparameters such as initial batch size and finally we will compare the accuracy to that of offline SeqClu.

Paper Outline

The paper is structured in the following way. Section 2 comprises the description of SeqClu as well as the hypothesis and the methodology applied in the process of research. Section 3 describes several initialization methods and the motivation behind them. Section 4 dives into the details of the experimental setup, the goals of the experiment and the obtained results. Finally Section 6 will summarise the key findings of the paper and outline possible future work.

Related Work

Research on online clustering algorithms has been a hot topic ever since data size started going exponential. In this sense we should mention pioneering works such as Anna Choromanska's Online Clustering with Experts [1], Edo Liberty's Online K-means [10] as well as Vincent Cohen-Addad's Online k-means [2]. The above works help better understand the problem of online clustering however algorithm initialization (especially online) is barely discussed.

[1]A more in depth look at online clustering will follow in the related works subsection

More specialised online clustering algorithm such as Grua's Clustream GT [6] and Islam's BOCEDS [7] treat the problem of clustering various types of data such as evolving data in the health domain, or data which requires a density based approach. Regardless the discussion of initialization is limited and inconclusive.

On another hand initialization heuristics have themselves been a popular research topic [4], [8] however these mainly focus on offline algorithms.

2 Methodology

Our research project builds on top of the baseline SeqClu which is an online, prototype based k-medoids algorithm. The baseline version has two stages, the initialization and the classification stage. During the initialization the clusters are initialized with 5 prototypes of the same class[2]. The classification stage simulates the online fashion of processing the data. It classifies points one by one. A point is assigned to the cluster ***i*** which has the lowest average distance to it. After each step the cluster is updated by replacing the farthest prototype ***j*** from the newly classified point with the point itself. The rules of the algorithm are formalized below:

$$i = argmin_{c_i \in C} \frac{1}{|c_i|} \sum_{p \in c_i} DTW(x, p) \tag{1}$$

$$j = argmax_{p_j \in c_i} DTW(x, p_j) \tag{2}$$

DTW is used for measuring similarity among sequences.

The goal is to come up with one or more solutions for prototype initialization that will allow clustering the data with a high degree of accuracy and will help reveal the structure of the clusters in the dataset.

Hypothesis

We are interested in the particularities of the performance of the online version of SeqClu with various initialization heuristics compared to the baseline and the offline SeqClu. We expect online SeqClu to present a good approximation of the offline algorithm in similar conditions and to approach the metrics of the baseline version with best case scenario initialization.

Results Validation

The algorithm is tested on 2 datasets. The first one is the UJI Pen Handritten Characters. [5] It is comprised handwritten characters represented as sequences of 2 dimensional coordinates. It has representations for all the letters in the English alphabet and digits from 0 to 9. For letters both the uppercase and lowercase versions are treated as the same class. The second is a toy dataset consisting of families of sine functions. Within families, the functions differ within the bounds of a predefined error. The dataset can be generated on the fly by specifying the frequency, phase, and error bound for every family of functions. Both datasets contain labelled data.

[2]This is only done as a proof of concept to show that with a successful initialization a high accuracy can be achieved

The performance of the algorithm is measured using the Silhouette [11], accuracy and Loss, where the Loss is calculated in the following way.

$$Loss = \frac{1}{|C|} \sum_{c \in C} \frac{1}{|c|} \sum_{x \in c} DTW(x, c) \tag{3}$$

The accuracy is the ration between true positives (*tp*) and the total number of clustered sequences (*N*).

$$acc = \frac{|tp|}{|N|} \tag{4}$$

To get a better understanding of the quality of the clustering, the metrics measured in the online algorithm are going to be compared to several other implementations namely the online baseline and the offline variant. The baseline is expected to be a best case scenario for a particular permutation of the dataset as it will be initialized with 5 prototypes of the same class. The offline comparison between the performance of the online and the offline version serves to demonstrate the trade off between keeping all the data in memory and classifying it in one go.

3 Greedy Heuristics

As mentioned previously one of the main goals of this paper is to investigate how the classic initialization heuristics fare in an online setting and how do they stack up against each other. One of the most popular heuristics for algorithms such as k-means, is to select as initial centroids, points that are sufficiently far apart from each other so as not to arrive at the situation that 2 or more of the centroids are very close to each other.

We can already see the outlines of the first heuristic, or rather the first class of heuristics that we are going to come up with. Namely we will select *c most diverse* points to be the basis of our prototypes. We will employ a procedure similar to the one Marina Drosou describes as *Greedy Construction*[4]. The main idea behind it is to select *c* sequences which have the highest pairwise distances among them, where *c* is the number of clusters in the dataset.

We should note that since we operate in an online setting the sequences we will end up selecting as basis of our prototypes may or may not correspond with the sequences which are globally farthest away from each other. Given that we operate under a memory constraint we will buffer the first *k* sequences that arrive and pick our representative sequences from this batch.

Greedy Construction v1

The first version of the Greedy Construction heuristic makes use of the assumption that the clusters are not overlapping and are far enough so that the *k* farthest points from each other will belong to different clusters. Furthermore it relies on another assumption, that the *p-1* closest points to the chosen prototype will also belong to the same cluster. Of course these assumptions might not hold if the dataset is very noisy, the clusters are close to each other, if the initial batch contains less than *p* instances of a particular class or if due to the

Algorithm 1 Greedy Construction

```
1: $DM : Distance\ Matrix\ of\ size\ k\ by\ k$
2: $c : number\ of\ clusters$
3: $k : batch\ size$
4: procedure GreedyConstructionV1($DM, c, k$)
5:     $i, j \leftarrow argmax(DTW(i, j))$    ▷
6:     $C \leftarrow \{\{i\}, \{j\}\}$
7:     while $|C| \leq c - 1$ do    ▷ repeat until c clusters
8:         $p \leftarrow argmax(DTW(p, C))$
9:         $C \leftarrow C \vee \{\{p\}\}$
10:    end while
11:    $P \leftarrow \{i \in 1, k\ \ \&\ \ c \in C||(DTW(i, c), i, c)\}$
12:    for all $(dist, i, c) \in P$ do
13:        if $|c| \leq p - 1\ \ \&\ \ i \notin C$ then
14:            $c \leftarrow c \vee i$
15:        end if
16:    end for
17:    Return C
18: end procedure
```

greedy nature the heuristic lands onto an outlier. However in the course of experiments, we will prove that even such a naive heuristic can generate good enough results.

GreedyConstruction v2

In the previous heuristic the greedy step happens when we select the initial *c* farthest points from each other. One could argue that what makes an initialization good is not only the diversity of the initial points but also the fact that the prototypes of a cluster are sufficiently close to each other and are not on the outskirts of the cluster. The motivation behind this is that in sufficiently separated clusters, points that are close to each other belong to the same cluster. So for a particular point *c* the smaller the sum of distances between itself and its *p-1* closest neighbours, the higher the probability that the majority of its closest neighbours belong to the same cluster.

This is where the idea for a modified GreedyConstruction heuristic stems. In this heuristic we will minimize the sum of distances of a point's nearest neighbours while maximizing the distance between the formed prototype clusters.

As previously stated GreedyConstructionV2 aims to minimize the sum of distances between the central medoid and the other 4 prototypes first and then maximize the diversity of the clusters. It achieves this by doing the following precomputations. First the distance matrix for the initialization batch is computed, which results in a *k by k* matrix. Then another matrix (IM) is generated on the basis of the distance matrix. In every row IM contains the indices of the distances sorted ascendingly. Then the rows are sorted vertically according to the sum of the first *p* distances corresponding to the first *p* indices in every row of IM. In such a way the first row corresponds to the point which has the tightest knit cluster of size *p* around itself. The sums corresponding to every element index are computed and saved in another array S of size *k*.

The algorithm proceeds in the following way. The first cluster with prototypes is going to be the cluster corresponding to the first row of IM. Next we compute the farthest point which was not used previously, by taking the last entry in the

Algorithm 2 Greedy Construction

```
Require: IM is sorted by distance to i for all rows $i \leftarrow 1, k$
Require: IM is sorted vertically by sum of first p distances
    on each row i for $i \leftarrow 1, k$
Require: IM contains indices
Require: S is the array of sums of first 4 closest neighbours
    for every point
1: procedure GCV2($IM, S, c, k, p$)
2:     $pointer \leftarrow 1$
3:     $farthPnts \leftarrow \{\}$
4:     $C \leftarrow \{\}$
5:     while $|C| < c$ do    ▷ repeat until c clusters
6:         $C \leftarrow C \vee AddPrototypesOf(IM, pointer, p)$
7:         $farthPoint, farthPnts \leftarrow$
    $getfarthPoint(IM, S, farthPnts, p, k, pointer)$
8:         $pointer \leftarrow farthPoint$
9:     end while
10:    $P \leftarrow \{i \in 1, k\ \ \&\ \ c \in C||(DTW(i, c), i, c)\}$
11:    for all $(dist, i, c) \in P$ do
12:        if $|c| \leq p - 1\ \ \&\ \ i \notin C$ then
13:            $c \leftarrow c \vee i$
14:        end if
15:    end for
16:    Return C
17: end procedure
```

```
1: procedure AddPrototypesOf($IM, pointer, p$)
2:     $C \leftarrow \{\}$
3:     for $i \leftarrow 1, p + 1$ do
4:         $C \leftarrow C \vee IM[pointer][i]$
5:     end for
6:     Return C
7: end procedure
```

```
1: procedure getFarthPoint($IM, S, farthPnts, p, k, j$)
2:     for $i \leftarrow k + 1, p$ do
3:         if $IM[j][i] \notin farthPnts$ then
4:             $farthOfJ \leftarrow IM[j][i]$
5:             $break$
6:         end if
7:         $j \leftarrow j - 1$
8:     end for
9:     $neighbrs \leftarrow getPClosestNeighbrs(IM, farthOfJ)$
10:    $farth \leftarrow argmin(S[i] for$ i $\in neighbrs)$
11:    $farthPnts \leftarrow fartestPnts \vee farth$
12:    Return farth, farthPnts
13: end procedure
```

row in IM corresponding to the pointer. If that entry was already used we take the previous entry, thus taking the second farthest away point from the pointer. The next pointer is the point in the neighbourhood of the farthest point which has the tightest knit cluster around him (lowest sum of distance to its p neighbours).

Diversity or Similarity?

The 2 initialization methods spark an interesting debate on what would be more important in an online environment when it comes to initialization. A definite answer to this question would require a multitude of examples with data or even a rigorous proof. The later will not be found in this paper. However we would like to shift the attention of the reader not to the answer of the question but rather to the situations and the way in which SeqClu behaves as a result of employing one initialization or another on a particular permutation of a dataset. An interesting question would be how these methods can complement each other in an online environment as many of the precomputations can be reused. This will be further investigated in the course of the experimental phase.

4 Clustering Heuristics

A somewhat different approach to tackling the initialization of prototypes is performing offline clustering on the initial batch of data. The idea to perform offline clustering on an initial batch of data is also taken from Marina Drosou's paper on Comparing Diversity Heuristics.

The 2 heuristics proposed are both offline K-medoids algorithms without prototypes. These are based on assumptions similar to the Greedy Heuristics, namely that the clusters are sufficiently spread apart, so that the k initial medoids with the biggest pairwise distance among them will correspond to different clusters, also it relies on the idea that points that are from the same cluster will be closer to one another.

The 2 heuristics differ in their eagerness to select their initial medoids. The first one, K-Medoids greedy, greedily selects c farthest points from each other and performs offline clustering with these as medoids.

K-Medoids Greedy

K-Medoids++

A small modification of the previous heuristic emerges from doubts similar to the Greedy Construction heuristics. What if the heuristic lands on outlier data? Do we really need the farthest points from each other to achieve diversity in the case of clustering heuristics? Given that k-means++ is a well known improvement of the classic k-means algorithm, a similar improvement can be implemented in the case of k-medoids as well. In our case k-medoids++ will first randomly pick a sequence from the initial batch and then pick subsequent sequences based on probabilities proportional to the square of their distances to the previously picked seuqence(s). Similar to the previous heuristic the prototypes will be the clusters resulting from running offline k-medoids on the initial batch of sequences.

Algorithm 3 K-MedoidsGreedy

```
1: procedure KMEDOIDSGREEDY($DM, c, k$)
2:     $i, j \leftarrow argmax(DTW(i, j))$
3:     $C \leftarrow \{\{i\}, \{j\}\}$
4:     while $|C| \leq c - 1$ do                ▷ repeat until c clusters
5:         $p \leftarrow argmax(DTW(p, C))$
6:         $C \leftarrow C \vee \{\{p\}\}$
7:     end while
8:     $medoids \leftarrow C$
9:     while Any of the medoids change do
10:        for $i \leftarrow 1, k$ do
11:            $c_i \leftarrow argmin_c(i, c)$
12:            $C[c_i] \leftarrow C[c_i] \vee i$
13:            $newMedoids \leftarrow \{\}$
14:            for all $c \in C$ do
15:                $newMedoid$
16:                $\leftarrow argmin_{i \in c} \frac{1}{|c|} \sum_{j \in c \setminus i} DTW(i, j)$
17:                $newMedoids \leftarrow$
18:                $newMedoids \vee newMedoid$
19:            end for
20:            if $newMedoids = medoids$ then
21:                Return C
22:            end if
23:
24:        end for
25:        Return C
26:
```

Algorithm 4 K-Medoids++

```
1: procedure $KMedoids++(DM, c, k)$
2:     $i, \leftarrow randomInt(k)$
3:     $C \leftarrow \{\{i\},\}$
4:     while $|C| < c$ do                ▷ repeat until c clusters
5:         $probabilities \leftarrow getProbabilities(DM, C)$
6:         $p \leftarrow pickWithProbability(k, probabilities)$
7:         $C \leftarrow C \vee \{\{p\}\}$
8:     end while
9:     $medoids \leftarrow C$
10:    while Any of the medoids change do
11:        for $i \leftarrow 1, k$ do
12:            $c_i \leftarrow argmin_c(i, c)$
13:            $C[c_i] \leftarrow C[c_i] \vee i$
14:            $newMedoids \leftarrow \{\}$
15:            for all $c \in C$ do
16:                $newMedoid$
17:                $\leftarrow argmin_{i \in c} \frac{1}{|c|} \sum_{j \in c \setminus i} DTW(i, j)$
18:                $newMedoids \leftarrow$
19:                $newMedoids \vee newMedoid$
20:            end for
21:            if $newMedoids = medoids$ then
22:                Return C
23:            end if
24:
25:        end for
26:        Return C
27:
```

```
1:  procedure $getProbabilities(DM, C)$
2:      $probs \leftarrow \{\}$
3:      for all $i \leftarrow 1, k+1$ do
4:          $probs[i] \leftarrow (min_{c \in C} DTW(c, i))^2$
5:      end for
6:      for all $i \leftarrow 1, k+1$ do
7:          $probs \leftarrow probs / \sum_{p \in probs}$
8:      end for
9:      Return probs
10: end procedure=0
```

5 Experimental work

As we previously mentioned we are interested in 3 metrics when it comes to our initialization methods, namely *accuracy*, *loss*, and *silhouette*. An obvious hyperparameter which needs tuning for each method is the batch size. We could assume that all methods can simply make do with $c \times p$ but our conclusions would not be as valuable.

For a better understanding of the clustering we will also record the quality of the initial prototypes as a function of the number of classes present and the presence of a majority class in every prototype cluster.

Similarly, we want to understand whether any one method in the 2 classes we have come up with is preferable over all dataset permutations, or whether they compliment each other.

6 Experimental Setup and Results

Setup

The performance of the initialization heuristics is measured against two datasets. The main dataset is UJI Pen Handwritten Characters [5] and performance on it will be the main source of conclusions. The second dataset called 'Sine' will be used to reinforce or debate ideas, and conclusions formulated on the basis of the previous dataset.

For the experiment to be more likely to generate in meaningful results we set up the following pipeline. The 2 datasets are permuted 30 times and every permutation is saved separately so that it can be accessed. Together with the permutations we precompute the corresponding distance matrix so that calculations are only done once and valuable time can be saved in the course of the experiment.

The SeqClu implementation is built with modularity in mind, so that different initialization methods can be used with the same implementation code. Similarly an experiment infrastructure is built so that the classification of the data and the evaluation of results can be done independently.

Batch size

For the batch size optimization we used 10 permutations of the 'handwritten' dataset which contained data belonging to 4 classes, namely the letters 'W', 'O', 'S', 'C'. The choice of these classes was based on their relative distance to each other. We have clusters like 'W' and 'O' which are well separated from all the other clusters, and we also have clusters 'C', 'S' which are very close together and often can be interpreted as the same cluster.

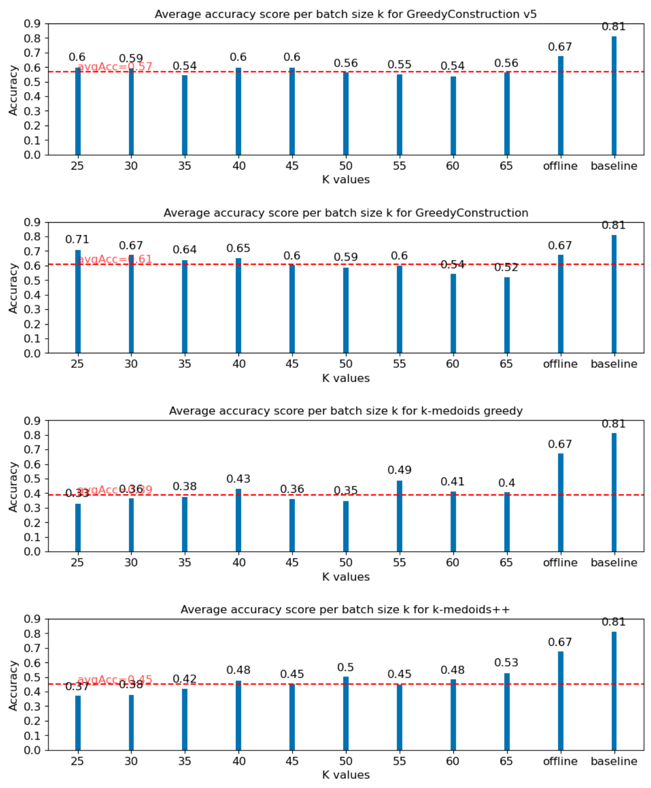

Figure 1: Average accuracy values per batch size for GreedyConstructionV2, GreedyConstructionV1, K-Medoids Greedy and K-Medoids++

As previously mentioned the batch size is the size of the initial batch of points fed to the initialization heuristic so that the prototypes can be initialized. The goal of this experiment is to establish optimal values of batch sizes for every heuristic and investigate the existence of a trend between bath size and clustering goodness.

We also need to mention that different batch sizes we used for both datasets given that the size of the data differed significantly. The table below summarises sheds light on batch sizes per dataset as well as dataset sizes.

Ds	Sz	Batch sizes								
UJI	211	25	30	35	40	45	50	55	60	65
sine	100	22	24	26	28	30	32	34	36	38

Table 1: Batch sizes per dataset

Results

Greedy Methods

There are several things that are worth discussing from the results. The accuracy results in Figure 1 show that the methods have a pretty similar performance, it is interesting that for smaller batch sizes the GreedyConstructionV1 shows on average a better accuracy than the second method, however as the batch size is increased their accuracy values converge, and for GreedyConstructionV1 it even drops.

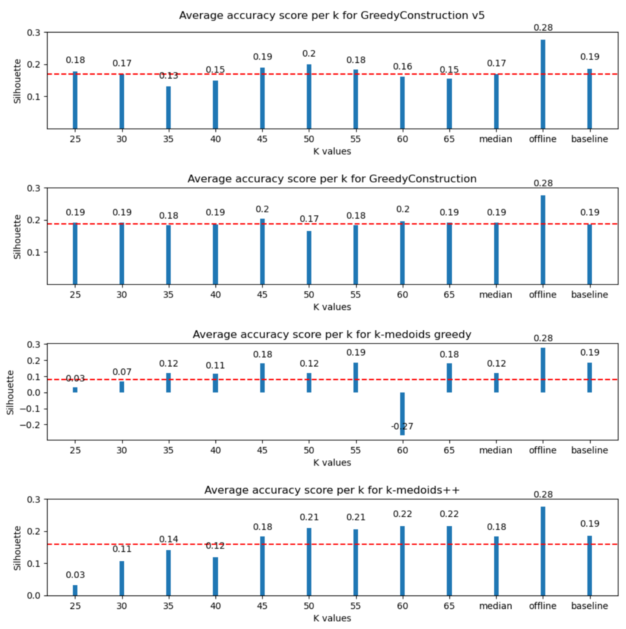

Figure 2: Average Silhouette Values per batch size for GreedyConstructionV2, GreedyConstructionV1, k-Medoids Greedy and k-Medoids++

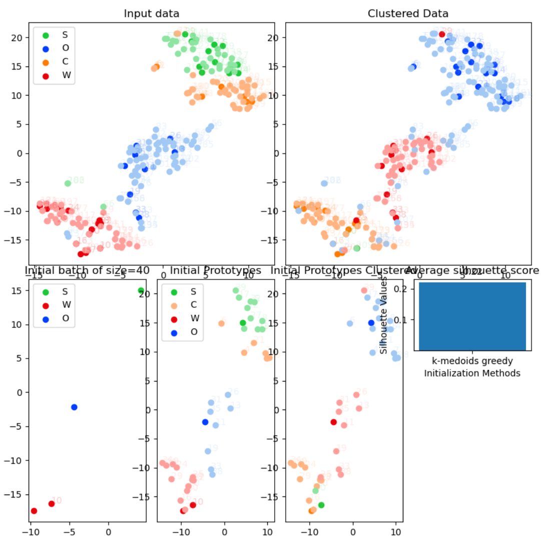

Figure 3: Highest Accuracy Clustering for K-Medoids Greedy initialization. K-Medoids Greedy deals badly with clusters that are close to each other

In terms of comparison to the offline version which ran on the same data with a similar initialization we can clearly see that the online initialization methods showcase only a small drop in accuracy. In terms of silhouette score the drop is more significant which says that in terms of minimizing distances within clusters, the offline version generates a superior solution.

Looking at the results from the perspective of finding optimal values for the batch size *(k)* we see that on average smaller batch size are preferred to larger ones (Figure 1). It is interesting that the value of batch size does not seem to have significant effects on the goodness of clustering in terms of silhouette (Figure 2) and loss (Figure 20), even showcasing similar performance to the best case scenario initialization *(baseline)*.

While small batch sizes for GreedyConstructionV1 showcase a good enough result, these still fall short of the baseline, which has best case scenario initialization. We can explain this with by simply looking at the true labels of the initial prototypes. Even though every set of 5 prototypes are mutual closest neighbours to one another, there is no real guarantee that they belong to the same cluster (i.e. have the same labels). The influence of these ”wrong” clusters can be in turn aggravated or fixed by the cluster update rules that SeqClu has. However this is a vast topic in itself and surely deserves attention on its own.

Going back to our greedy heuristics, we can conclude that the these are stable, solutions for clustering initialization which provide a good enough performance compared to the offline algorithm and the baseline.

Clustering Methods

From the results above it is clear that clustering methods require bigger batch sizes to showcase an accuracy and silhouette value similar to their greedy counterparts. In terms of comparing the 2 clustering methods, we see that the k-medoids++ variant is preferable in most of the situations (Figure 1, 2).

The k-medoids++ heuristic results in both higher accuracy, and higher silhouette score compared to the k-medoids greedy heuristic. This is explained by the fact that the greedy nature of the former heuristic oftentimes results in imbalanced clusters, a scenarios that leads to clusters whose sizes vary dramatically 4. This is amplified when an unfortunate choice of the initial medoid which results in a heterogeneous cluster, is further amplified by the characteristic swap step of the k-medoids algorithm. Thus, we can conclude that the k-medoids++ method is significantly more stable than its greedy clustering counterpart.

One interesting find is that the K-Medoids++ heuristic manages to minimize the clustering the best in terms of silhouette coefficient (Figure 2), overperforming even compared to the baseline. Of course this is not reflected in the accuracy score, where the baseline still is vastly superior, but this is a valuable find in clustering data that is not labeled.

One clear pitfall of the clustering heuristics are the clusters that are too close together. Even with an appropriate batch size and a ’fortunate’ initialization the algorithm does not treat the clusters belonging to ’C’ and ’S’ as two separate clusters but rather as a single one Figure 3.

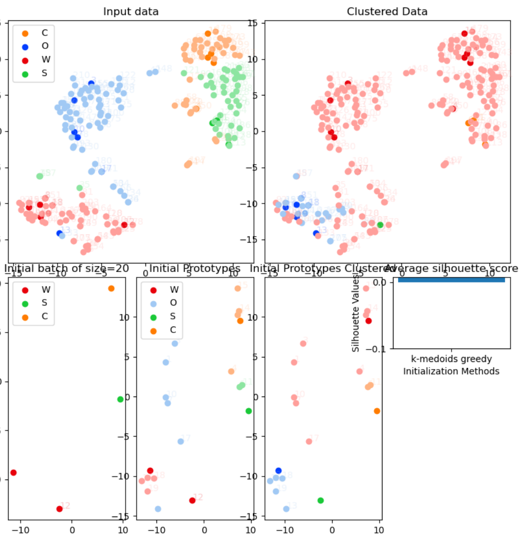

Figure 4: k-Medoids Greedy, often suffers from initial cluster imbalance

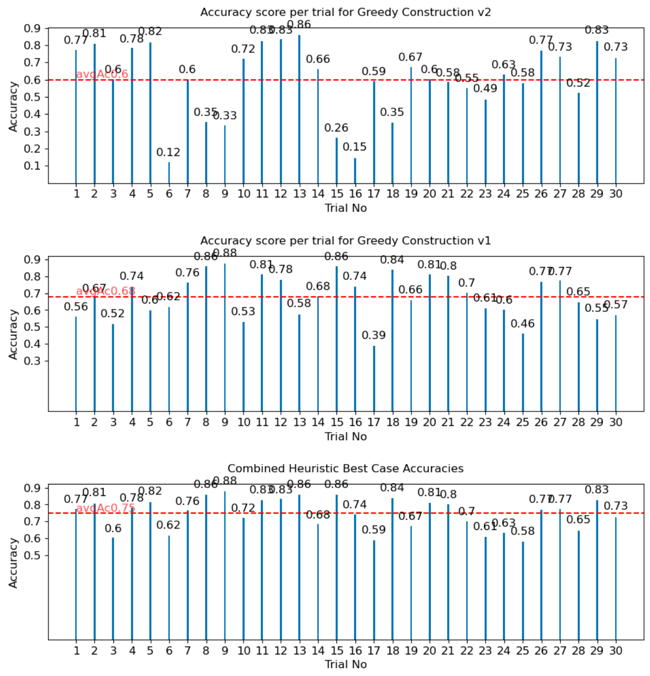

Figure 5: Average accuracy values per dataset permutation for GreedyConstructionV2, GreedyConstructionV1 for k=25

Prototype Cluster Size Imbalance

After effectuating the experiment, the cluster imbalance phenomenon has emerged as a clear shortcoming in comparison to greedy heuristics (Figure 4) which have fixed size prototype clusters. Although it seemed that the cluster imbalance can be solved by simply taking a bigger initial batch, evidence shows that even with sufficiently large batch size this phenomenon still happens (Figure 18).

Complementarity of Methods

A more attentive look at the accuracy data by the two greedy heuristics showcases another interesting find, namely that the 2 approaches, despite having similar performance on the handwritten dataset, exhibit different mistakes. Thus for some permutations Greedy Construction V1 has better accuracy, while for others Greedy Construction V2 has better accuracy (Figure 5).

This leads us to 2 possible scenarios. Either the methods are complementary, i.e. when one has poor accuracy the other has good accuracy, or they are not complementary but there still is some metric which can predict with some degree of accuracy which set of initial prototypes is better.

Our goal is to see whether we can achieve a higher accuracy than the highest average accuracy for the optimal value of the batch size. We select k=25 as optimal value of the initial batch size. Thus further experiments on complementarity of the methods will be conducted only for this value of the batch size.

To find out whether the methods were complementary we performed the Pearson correlation test [9] on a sample of accuracy values of both Greedy Initialization Methods. If the methods are complementary, we expect the test to yield a negative value smaller than *-0.5*. The test yielded *r=-0.02473* thus disproving the hypothesis that the methods are complementary. This can be further confirmed by Figure 6 where no negative slope trend can be observed.

This leaves us with the hope that we may be able to guess with some degree of accuracy whether a set of initial prototypes is better than another by using some metric such as *Silhouette*, *Davies-Bouldin Index* [3] or *Average Loss*.

When using Silhouette, we will prefer the set of prototypes with greater Silhouette value. When using Average Loss, or Davies-Bouldin Index, we will prefer the set of prototypes with smaller value of loss, or DB-Index respectively.

Further experiments show that none of the above metrics help determine a potentially more accurate clustering, thus the accuracy of the combined greedy heuristic is at best equal to that of the Greedy Construction V1. (Figures 17, 7, 8).

7 Responsible Research

Reproducibility

As one might expect with any scientific work, the experimental setup is as important as the conclusions and ideas presented in the paper. To this end it is obvious that both the data used in the experiments and the code generated during the process of research and experimentation will be made public on the Cyber Analytics Lab Github Page of TU Delft.

To get a better understanding of the ideas in the paper and the execution of the experiments the code will be well documented and refactored to reflect the best software engineering practices and to enable others to easily extend and modify the codebase.

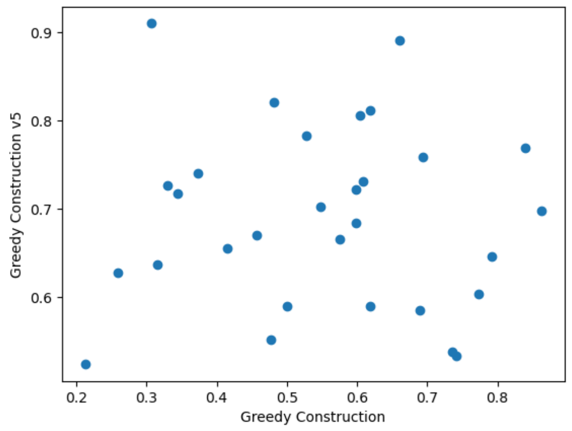

Figure 6: Lack of a negative correlation for accuracy values of GreedyConstructionV2, GreedyConstructionV1 for k=25

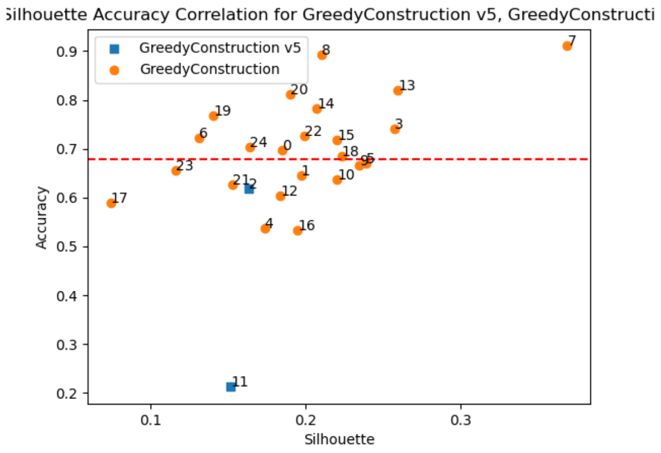

Figure 7: Accuracy when we choose the prototype set with highest silhouette score across 25 trials, batch size k=25

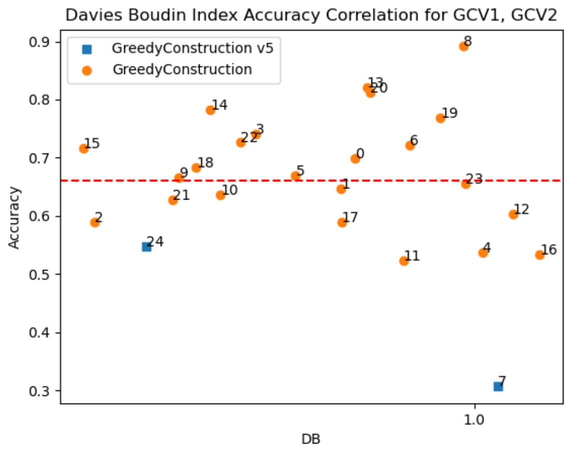

Figure 8: Accuracy when we choose the prototype set with lowest Davies Bouldin Index across 25 trials batch size k=25

Data Integrity

The data in the experiments is used as is and has not undergone any manipulation or tinkering. In order to give more credibility to the results, multiple experiments were attempted and logged.

Scientific Integrity

In the best traditions of scientific integrity, the paper will contain a bibliography with pointers to works which were fundamental in the process of this research project. Moreover ideas and any help will be dutifully noted in the paper.

8 Conclusions and Future Work

The online clustering of Sequences is an exciting and important topic both for the academic community and the industry. In this paper our main goal was answering how to best initialize the prototypes of SeqClu, in order to deal with the cold start phenomenon. We discussed the importance of a correct initialization and how to approximate the best case scenario with various heuristics. The main classes oh heuristics were the greedy and clustering heuristics.

We delved into optimizing an important hyperparameter like the initial batch size and we measured the performance of the algorithm with metrics like accuracy, silhouette score, and loss in order to establish best practices and explain the behaviour of the algorithm.

The two greedy heuristics proved to be preferable to both the clustering heuristics in terms of both accuracy and silhouette score. These dealt better (albeit not perfectly) with issues like clusters being too close together and proved to require smaller size initial batches in order to approximate the behaviour of the offline algorithm and the baseline algorithm. The fixed prototype cluster size made them less prone to cluster imbalance and enabled the subsequent stage of the SeqClu namely the classification phase to result in a good enough clustering. The experiments showed that a batch size of 25 points yielded the best accuracy and silhouette score. Clustering heuristics needed a bigger batch sizes (k=45, 50, 55) for maximum accuracy and silhouette score.

In terms of performance compared to the offline version, online SeqClu with any greedy heuristic presents no significant trade-off in terms of accuracy. The use of clustering heuristics results in inferior solution compared to offline SeqClu.

A closer look at the mistakes the greedy heuristics made, showed the potential of a combined heuristic which would choose the best prototype set by looking at a particular metric of cluster goodness with respect to the initial set of prototypes selected by either heuristic. Potential metrics like silhouette, DB-Index, and total loss were tested but the results showed no significant jump in accuracy.

Future Work

Albeit the clustering heuristics proved clearly inferior to the greedy heuristics presented in the paper, one can always argue that they simply were too naive to do the job. These heuristics have the benefit that they can be extended by applying local search, or evolutionary algorithms to the solutions (prototype clusters) they provide[8].

As for the greedy heuristics, these can be improved by finding a metric for the Combined Greedy Heuristic. Similarly, an interesting direction of research would be adding nondeterminism, which would allow using repetition to minimize the probability of picking unfortunate initial medoids.

References

[1] Anna Choromanska and Claire Monteleoni. Online clustering with experts. In Neil D. Lawrence and Mark Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 227–235, La Palma, Canary Islands, 21–23 Apr 2012. PMLR.

[2] Vincent Cohen-Addad, Benjamin Guedj, Varun Kanade, and Guy Rom. Online k-means clustering, 2019.

[3] David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979.

[4] Marina Drosou and E. Pitoura. Comparing diversity heuristics. 2009.

[5] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[6] Eoin Martino Grua, Mark Hoogendoorn, Ivano Malavolta, Patricia Lago, and A.E. Eiben. Clustream-gt: Online clustering for personalization in the health domain. In *IEEE/WIC/ACM International Conference on Web Intelligence*, WI '19, page 270–275, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Md. Kamrul Islam, Md. Manjur Ahmed, and Kamal Z. Zamli. A buffer-based online clustering for evolving data stream. *Information Sciences*, 489:113–135, 2019.

[8] L. Kazakovtsev and I. Rozhnov. Application of algorithms with variable greedy heuristics for k-medoids problems. *Informatica (Slovenia)*, 44, 2020.

[9] Wilhelm Kirch, editor. *Pearson's Correlation Coefficient*, pages 1090–1091. Springer Netherlands, Dordrecht, 2008.

[10] Edo Liberty, Ram Sriharsha, and Maxim Sviridenko. An algorithm for online k-means clustering, 2015.

[11] P. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

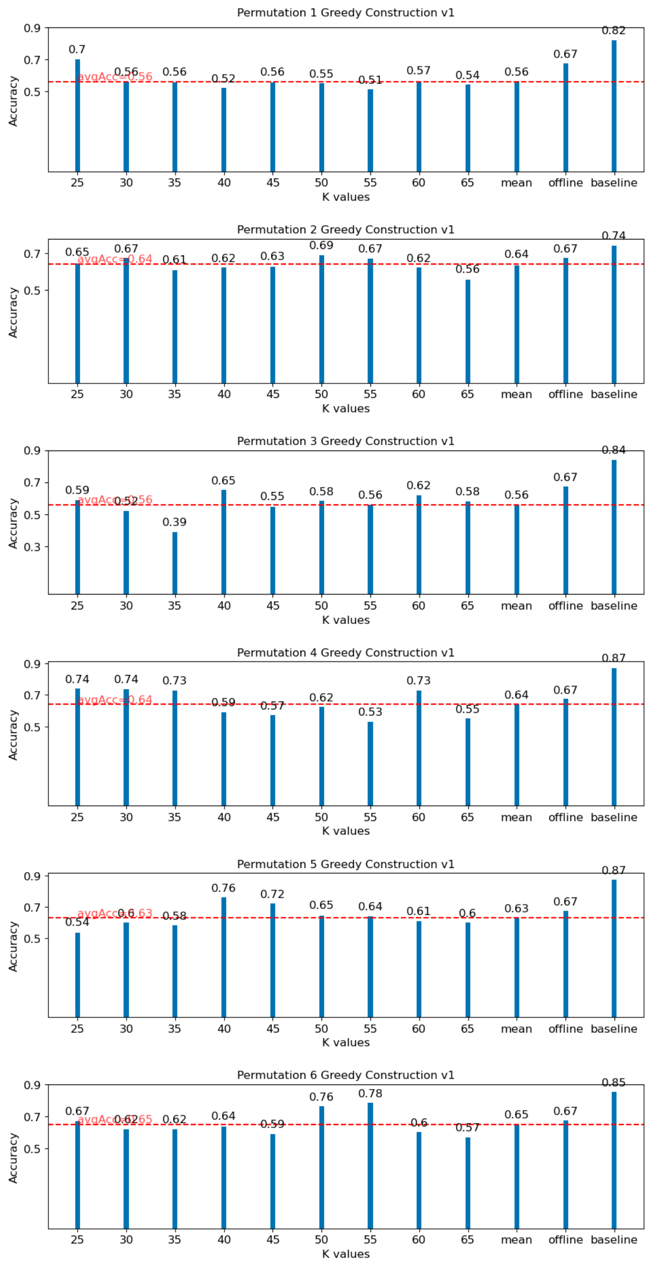

Figure 9: Average accuracy for the first 6 trials with GreedyConstructionV1 aggregated by batch size

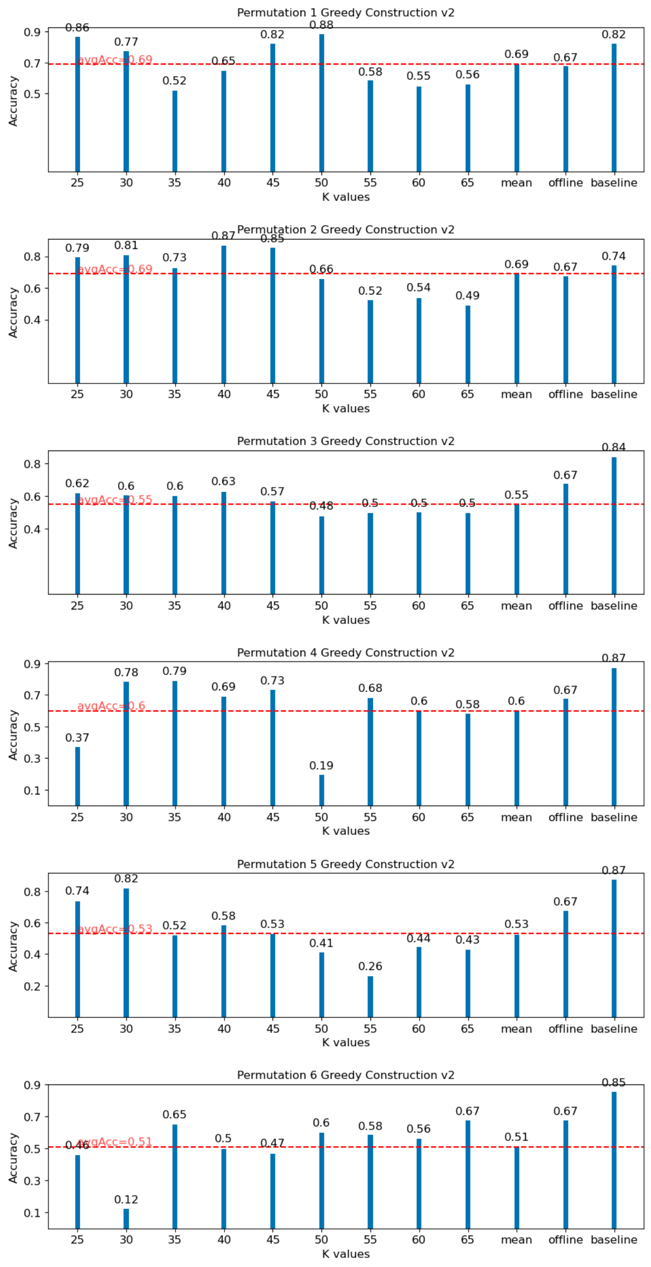

Figure 10: Average accuracy for the first 6 trials with GreedyConstructionV2 aggregated by batch size

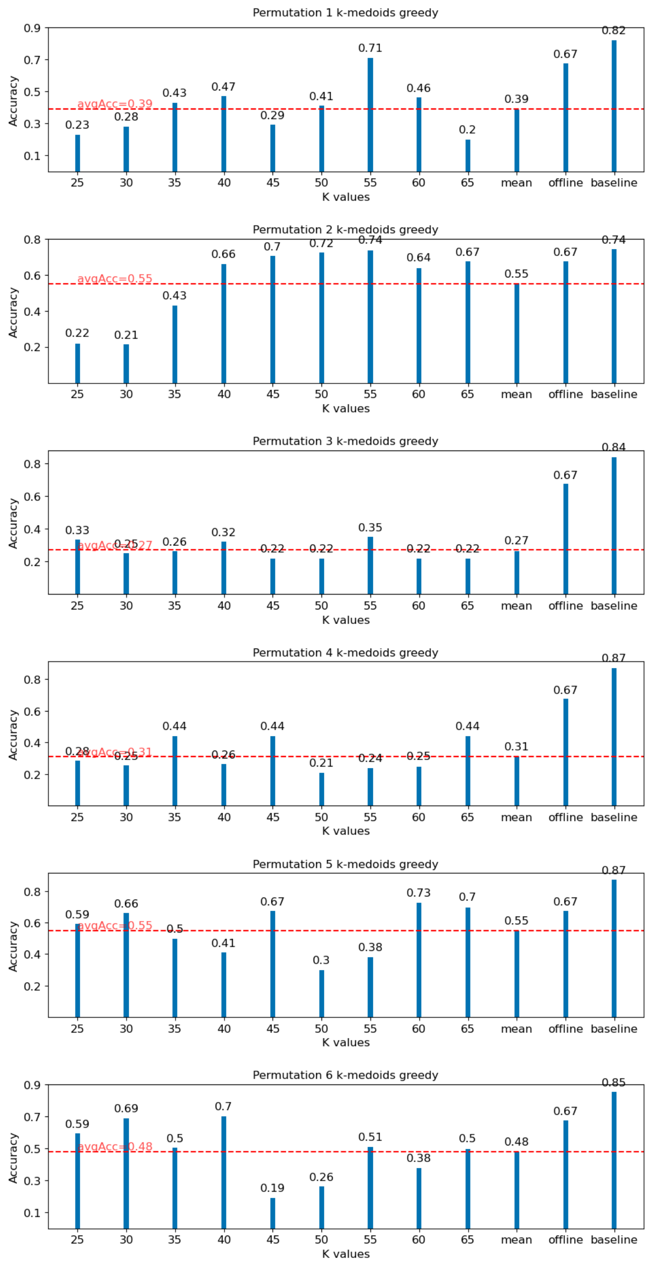

Figure 11: Average accuracy for the first 6 trials with KMedoids Greedy aggregated by batch size

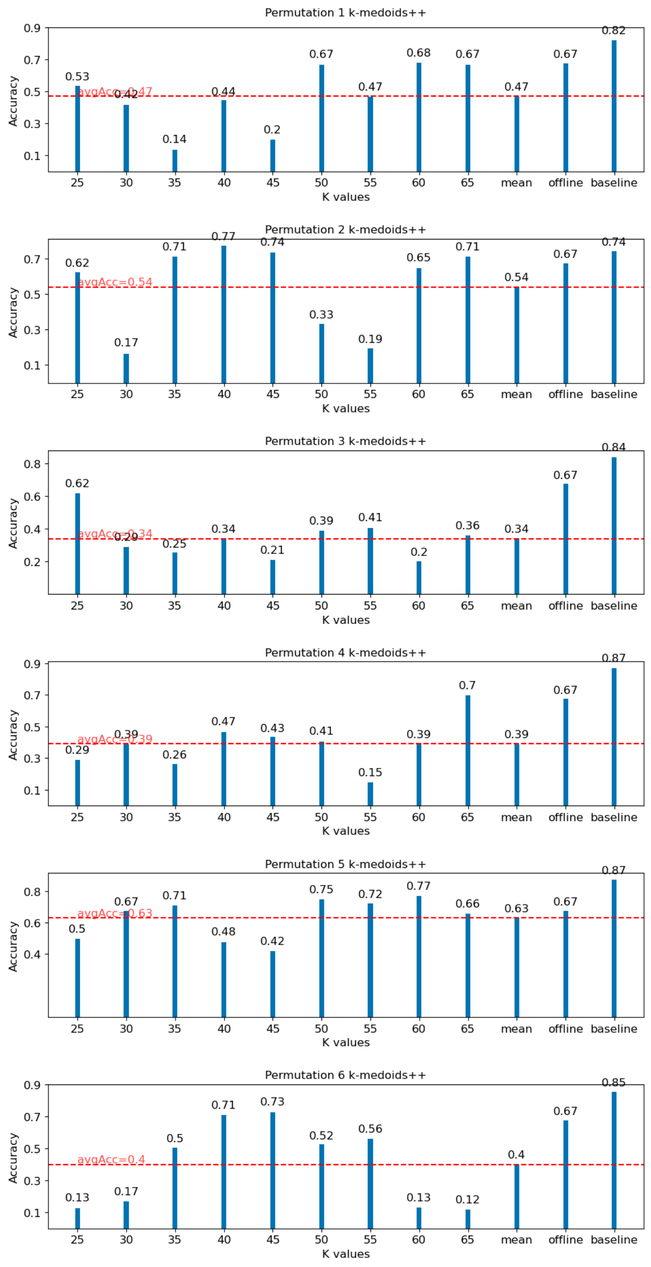

Figure 12: Clustering comparison Greedy Construction V1 vs Greedy Construction V2

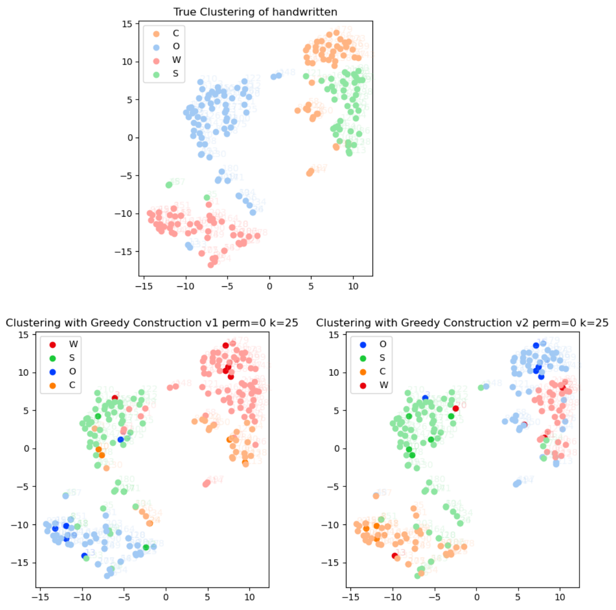

Figure 13: Clustering comparison Greedy Construction V1 vs Greedy Construction V2, GCV2 does better than GCV1

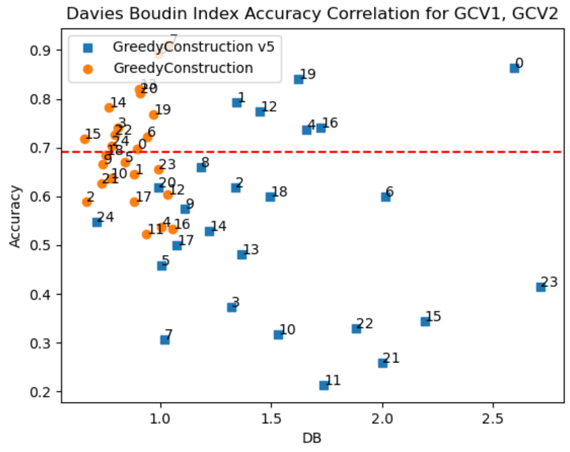

Figure 15: Accuracy vs DB-Index for batch size k=25 accross 25 trials

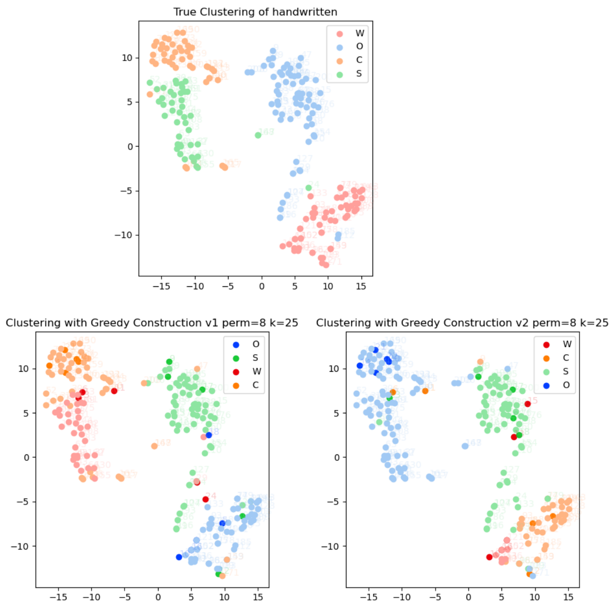

Figure 14: Clustering comparison Greedy Construction V1 vs Greedy Construction V2, GCV1 does better than GCV2

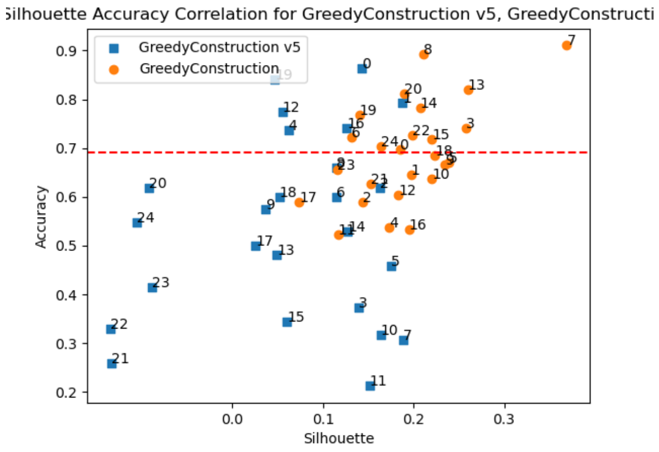

Figure 16: Accuracy vs Silhouette for batch size k=25 accross 25 trials

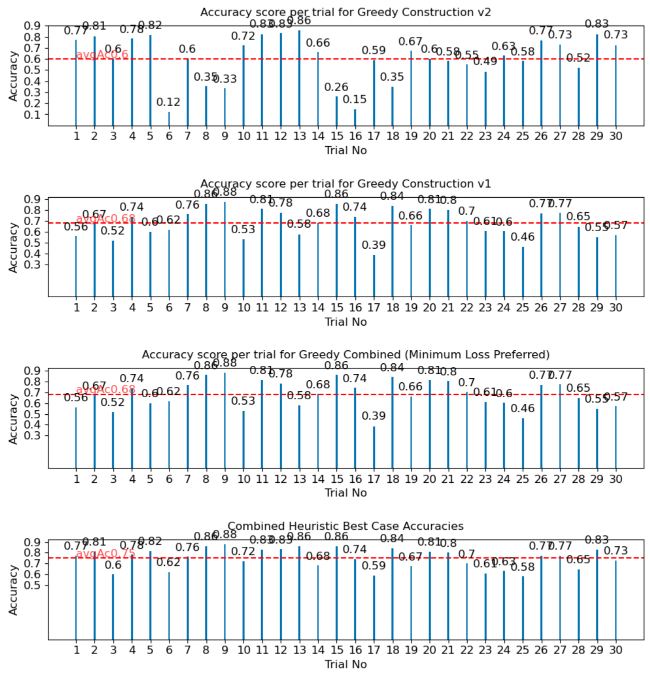

Figure 17: Accuracy when we choose the prototype set with lowest loss across 30 trials, batch size k=25

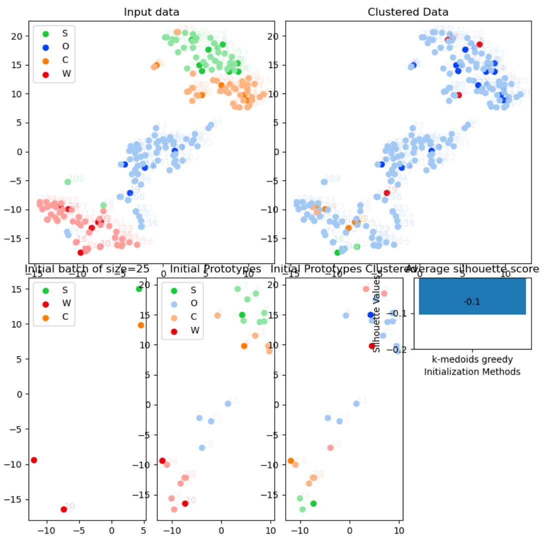

Figure 19: k-Medoids Greedy, often suffers from initial cluster imbalance

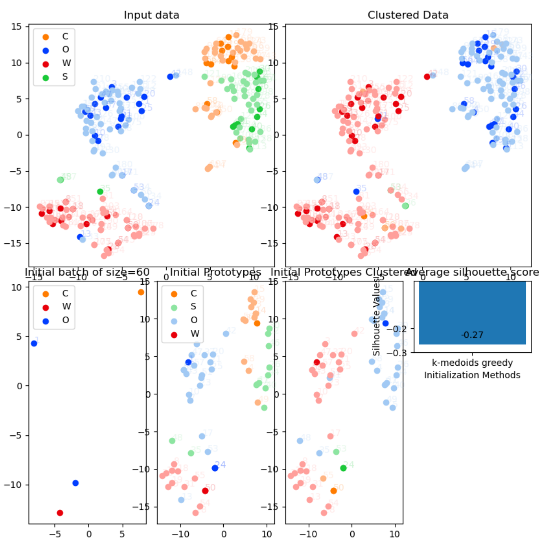

Figure 18: k-Medoids Greedy, Higher batch sizes do not eliminate initial cluster imbalance issues

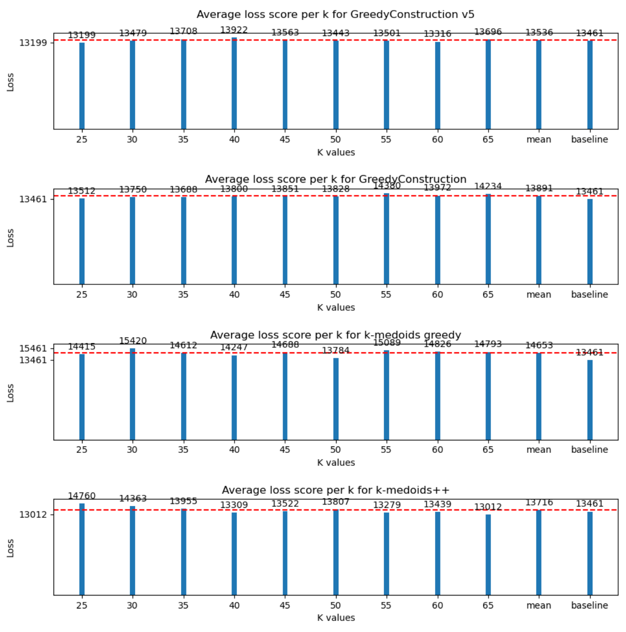

Figure 20: Average Loss Values per batch size for GreedyConstructionV2, GreedyConstructionV1, k-Medoids Greedy and k-Medoids++