Cluster Analysis using K-Means Clustering

***Abstract*—This paper examines the use of K-Means clustering as an application of unsupervised learning. It explores the use of different initialization strategies, distance metrics, as well as a mathematical approach to analysis of good clustering using the Dunn Index. The applicability of K-Means as a clustering algorithm is also discussed.**

Keywords—K-Means, clustering, unsupervised learning, classification, K-Means++, centroid, initialization strategy, Dunn Index, cluster verification.

I. Introduction

Unsupervised learning is an aspect of the broader machine learning space which is distinct from supervised learning in that while an algorithm may know the attributes or feature set of an example of data, it does not know the classification of said data. The primary goal of unsupervised learning is to ascribe pattern to otherwise label-less data and therefore with minimal human supervision—hence its namesake.

One such exercise in unsupervised learning is cluster analysis, or an approach to group and segment similar data into clusters to extrapolate relationships within data. An example of a cluster analysis algorithm is the K-Means clustering algorithm [1], which is an iterative approach to determining where cluster lie within data.

This report will explore the K-Means algorithm at length including the algorithm itself, two possible initialization strategies to employ within the algorithm, three distance metrics, as well as how to determine an optimal `k` parameter for the algorithm using the Dunn Index. The suitability of this algorithm in clustering four different sets of data is also examined.

II. K-Means Clustering

K-Means is an iterative approach to classification for unsupervised learning: provided a set of data, it is usually trivial to visually inspect and determine where clusters lie. However, algorithmically, it is not as simple. By placing special points within the data called centroids, the algorithm iteratively repositions them to be closer to collections of points and under ideal conditions, ultimately leads to centroids being positioned within the center of clusters [1], allowing the clusters to be identified and enumerated.

However, note that this presupposes ideal conditions [2]: the algorithm requires two parameters to function: `k`, or how many clusters is assumed to exist within the data, and how many iterations (epochs) to perform. With an inappropriate `k` or an insufficient number of epochs, the algorithm does not tend to yield great results. Additionally, there is a stochastic element to the algorithm where the initialization of centroids can lead to erratic or unexpected behavior likewise leading to poor results.

The pseudocode for the algorithm is below:

```
procedure k-means(integer k, integer e) is:
  load data into the coordinate system
  initialize k centroids within the system
  for each epoch e do:
    for each point p of data do:
      find the nearest centroid c to p
      consider p "belonging" to c
    endfor
    for each centroid c do:
      m := mean position of all p of c
      reposition c to m
    endfor
  endfor
endprocedure
```

The crux of the algorithm lies within two portions of the above: the initialization of centroids and determining membership of points to centroids [1].

The initialization strategy used to generate centroids is typically a naïve approach: distributing centroids at uniformly random positions. Considering the lower- and upper-bounds of the data, centroids may be placed within the bounds of the data with a random position. However, a more "intelligent" approach is to instead consider placing centroids within the bounds of the data but maximally distant from each other; this initialization strategy, called K-Means++, ensures centroids are placed within clusters themselves as clusters will be distant from each other to some degree.

Below is the pseudocode for random uniform initialization strategy:

```
procedure random-init(points ps, integer k) is:
  list of centroids := empty
  max-x := maximum x of any p
  min-x := minimum x of any p
  max-y := maximum y of any p
  min-y := minimum y of any p
  for range 1..k do:
    c := random (min-x..max-x, min-y..max-y)
    list of centroids += c
  endfor
  return list of centroids
endprocedure
```

The issue with random initialization of centroids is the algorithm is at the mercy of chance [6]; in other words, some centroid configurations may lead to poor results or possibly no results at all. Consider a scenario where later, the membership of a centroid is determined to have a size of zero.

The K-Means++ initialization strategy [4] is similar as it is still stochastic but considers where points lie within the data rather than just the bounds of the data:

```
procedure km++(points ps, integer k) is:
  list of centroids := empty
  x, y := position of random point in ps
  c := (x, y)
  list of centroids += c
  for range 2..k do:
    distances := empty
    for each point p in ps do:
      c := nearest centroid to p
      d := distance between p and c
      distances += d
    endfor
    x, y := position of p(max(distances))
    c := (x, y)
    list of centroids += c
  endfor
  return list of centroids
endprocedure
```

This strategy determines the furthest point from nearest centroids and initializes successive centroids after the first at those positions. This ensures each centroid is at a position of a point (therefore membership is guaranteed) and that the centroids are maximally distance from each other.

Determining the membership of points to centroids is an arithmetic affair: a point "belongs" to a centroid if it is closest to said centroid. However, there are multitudinous means to determine distance between points and centroids: several distance metrics such as Euclidean, Manhattan (rectilinear), and Chebyshev determine the distance between two points but with a differing calculation. This may lead to different results but also different computation times.

The three metrics mentioned, Euclidean, Manhattan, and Chebyshev, consider distances in different ways:

$$d_{p,q} = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2}$$

The Euclidean distance between points `p` and `q` is the root of sum squared differences between each coordinate axis. It is the "direct distance" between two points, considering a linear line between points as the distance. Note it consists of three additions and three exponentiations for a Euclidean distance calculation in two dimensions.

$$d_{p,q} = |q_x - p_x| + |q_y - p_y|$$

The Manhattan distance between points `p` and `q` is the sum of the magnitude of differences between each coordinate axis. Also referred to as rectilinear distance, in two dimensions it considers the distance in each axis rather than a linear line. It consists of three additions.

$$d_{p,q} = \max(|q_x - p_x|, |q_y - p_y|)$$

The Chebyshev distance between points `p` and `q` is like the Manhattan distance in two dimensions but rather than considering all axes, it only considers the greatest distance among axes. This requires only two additions to compute.

The arithmetic operations are mentioned as to a computer, an addition is "cheaper" than an exponentiation and of course less operations is also cheaper. From the three, Euclidean metrics are the most expensive whereas Chebyshev metrics are the least expensive [5].

Despite each metric still calculating the distance between points, not all of them satisfy the *triangle inequality* which for the purpose of distance calculations would suggest that ordering is preserved between metrics [3].

Intuitively, a distance calculation would yield the distance between two points, but "accuracy" is lost in ordering of distances between sets of points when transitioning from one metric to another. If point `p` is closest to point `q` in Euclidean space, it may not be in Manhattan space, to give an example.

III. Methodology Of Analysis

The algorithm is examined by comparing the performance using the two initialization strategies as well as the three distance measures. Additionally, differing values for `k` are examined.

In order to quantify how one parameter performs over another, the Dunn Index [6] is used:

$$DI_m = \frac{\min\limits_{1 \le i < j \le m} \delta(C_i, C_j)}{\max\limits_{1 \le k \le m} \Delta k}$$

Where the numerator is the minimal distance between centroids and the denominator is the maximum cluster size (physical size, i.e. diameter). The distance between centroids is calculated using one of the distance metrics and a cluster's diameter is simply the maximal distance between any two points in that cluster:

```
procedure dunn(points ps, clusters cs) is:
  num := ∞
  for cluster ci in cs do:
    for cluster ci in cs do:
      if ci is not cj do:
        num = min(num, distance(ci, cj))
      endif
    endfor
  endfor
  den := 0
  for cluster c in cs do:
    for point pi in ps do:
      for point pj in ps do:
        if pi and pj are in c do:
          den = max(den, distance(pi, pj))
        endif
      endfor
    endfor
  endfor
  return num/den
endprocedure
```

The Dunn Index is a measure of suitable clustering with a higher Dunn Index suggesting better clustering [6]. Each does consider the `k` parameter used: a more appropriate `k` would mean clusters are further apart, as clusters will be distant to some degree, and a more appropriate `k` would also mean clusters are more compact, thus influencing the size of clusters.

Four data sets are used to analyze the efficacy of the K-Means clustering algorithm: `S1`, `S2`, `S3`, and `S4`. Each set has a nominal `k` value of 15. In order, the sets become increasingly compact and visually, differentiability of clusters is diminished. This allows the performance of the algorithm to be examined on not only multiple sets, but of sets with different characteristics.

Each test's results consider the best ten results from thirty possible results: this allows statistical significance given enough runs of the algorithm but focuses on the best results as K-Means is stochastic and not every run is successful.

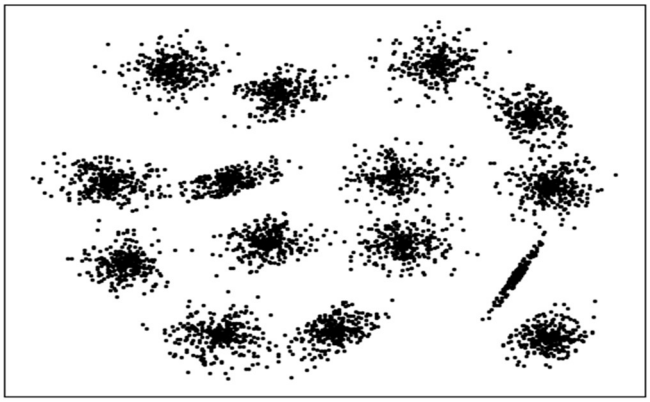
IV. Analysis of Results

A. *Initialization Strategy*

First, the initialization strategy is analyzed. By loading the data and generating centroids using each initialization strategy, how the algorithm approaches clustering is examined.

Using `S1` as an example and using one run of the algorithm, the data is first loaded into a two-dimensional space and plotted:

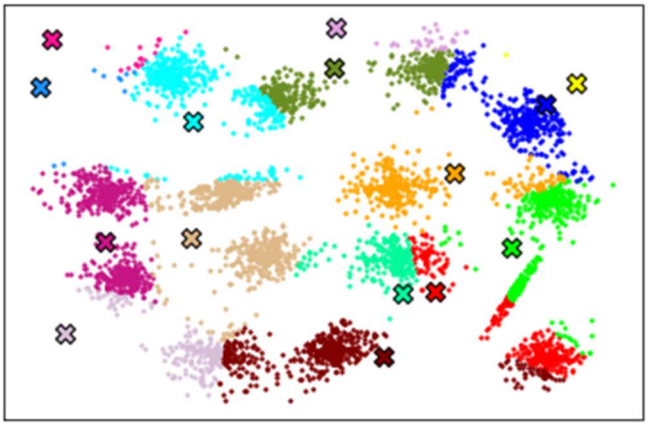
Plot I. S1 Data Set



Plot I. Point data for S1 data set, no centroids.

Visually, fifteen clusters can be identified, and using random uniform initialization of centroids, the "proposed" clusters can be seen by color-coding points based on their nearest centroid:

Plot II. Random Initialization of Centroids

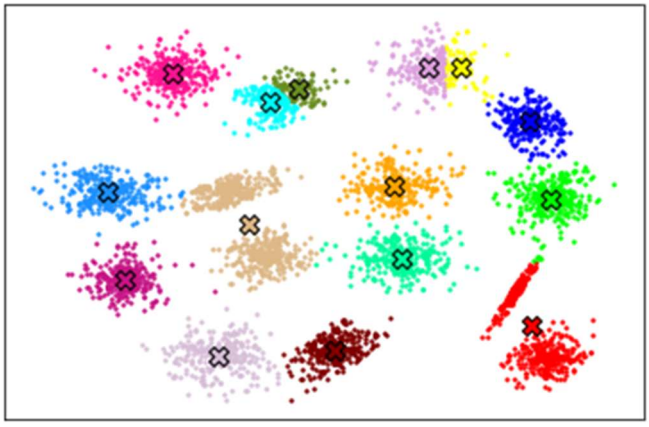


Plot II. S1 set, centroids randomly initialized and color-coded.

In this plot, centroids are denoted by an "x". Of note is some centroids were placed outside of range of a point and while it has member points, it does not superimpose upon a point, one of the distinctions between random uniform and K-Means++ strategies [4].

As the algorithm progresses, the centroids are repositioned, membership of points is recalculated, and the cycle repeats over enough iterations. An example result using random initialization is as below:

Plot III. Final Result from Random Initialization



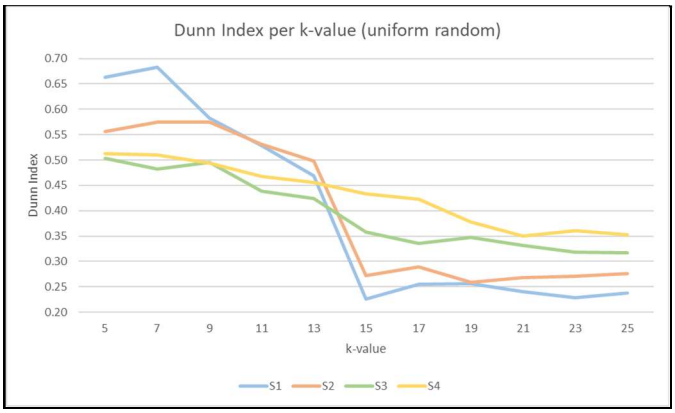
Plot III. S1 set, result of K-Means using random initialization.

Using a random initialization strategy, the result is underwhelming: from fifteen clusters, nine were correctly found, two clusters contain two centroids, and two centroids contain two clusters.

The Dunn Index for this clustering is `0.1424`, which is considered a low Dunn Index as will be learned later. The problem with random initialization is the centroids are placed in such a way that doesn't maximize distance between centroids and when the algorithm proceeds, this results in pairing of clusters and/or pairing of centroids. Occasionally the algorithm may indeed work, but it is uncommon.

Comparing different `k` values for random uniform initialization yields the below graph:

Graph I. Uniform Random DI per K-Value

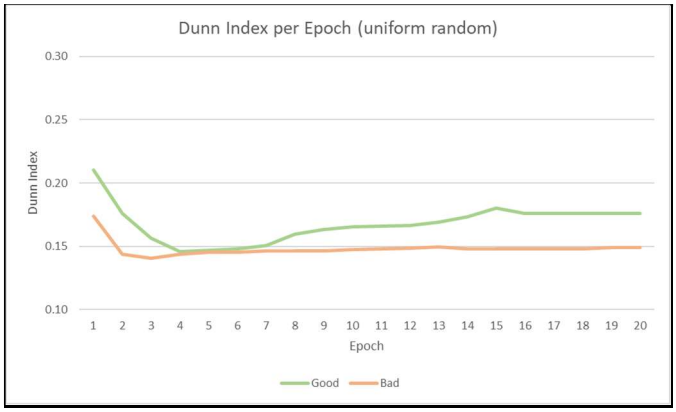


Graph I. K-Value compared per set using Random Initialization.

As mentioned previously, a higher Dunn Index indicates better clustering, yet with a nominal `k` of `15`, the reverse is found. This is due purely by chance: using a random initialization strategy yields good results rarely: a poor K-Means result would lead to a smaller Dunn Index, and over thirty runs, the ten best results are simply not good results, leading to poor data.

Further compounding this is if you examine how the Dunn Index changes over each iteration of the algorithm:

GRAPH II. UNIFORM RANDOM DI PER EPOCH



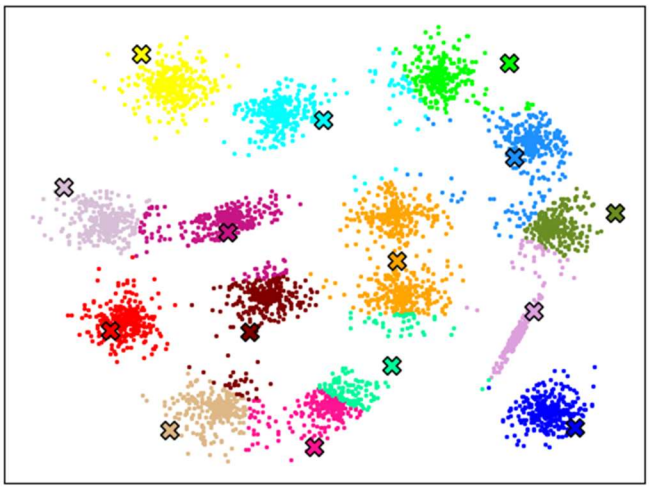
Graph II. Dunn Index compared over epochs, k=15, S1.

In this, a sample of "good" and "bad" results were taken to visualize how the Dunn Index changes over time with each. For both, there is a decline in Dunn Index from the first iteration, although a good run does increase slightly before stagnating. The bad result, however, stagnates immediately [4].

Unfortunately, all this suggests is that this initialization strategy is poor for all data sets used as the system converges to a suboptimal solution and stagnates after few epochs. However, there is still merit in comparing to a better initialization strategy.

As before, data was loaded into the coordinate space and centroids were generated, but using the K-Means++ strategy:

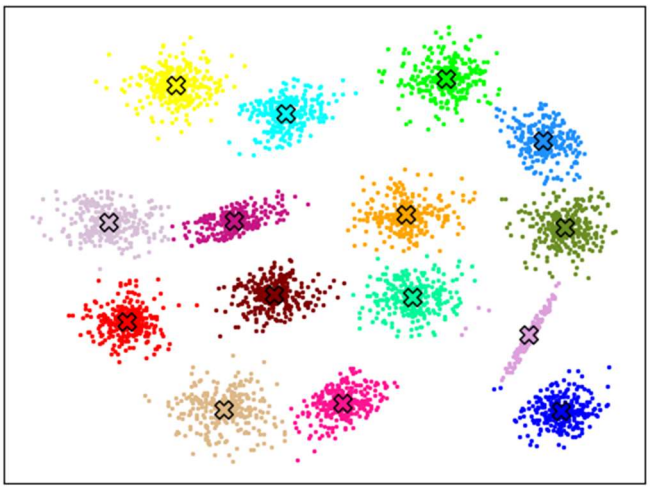
PLOT IV. K-MEANS++ INITIALIZATION OF CENTROIDS



Plot IV. S1 set, centroids initialized via KM++, color-coded.

As before, centroids were placed. Of note is that each centroid is superimposed on a point in the system and that centroids are evenly dispersed (i.e. with maximal distances between centroids). And as before, centroids are iteratively repositioned base on centroid membership. The result of the algorithm is as below:

PLOT V. FINAL RESULT FROM KM++ INITIALIZATION



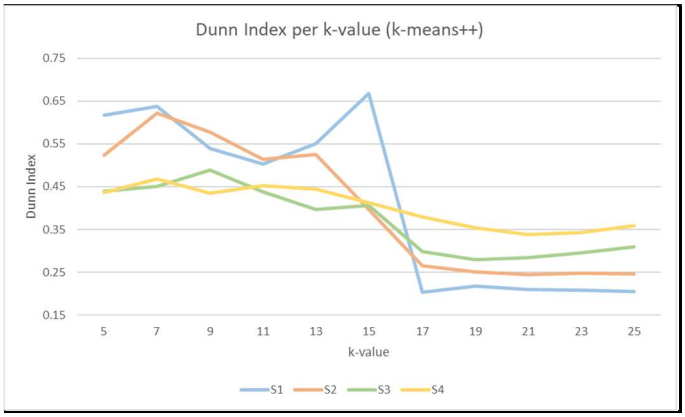
Plot V. S1 set, result of K-Means using KM++ initialization.

Unlike when using random initialization, the K-Means++ strategy was able to correctly identify all fifteen clusters with a Dunn Index of `0.7023`, a much higher result than with random initialization.

However, K-Means++ is not infallible. Not unlike random initialization, there is a stochastic element inherent to the strategy as the first centroid is placed in a random position. Due to this, the algorithm can still "fail" but the probability of a good result is much greater.

Running the algorithm over many instances and each data set, a graph as before is created:

GRAPH III. KM++ DI PER K-VALUE



Graph III. K-Value per set using KM++ initialization.

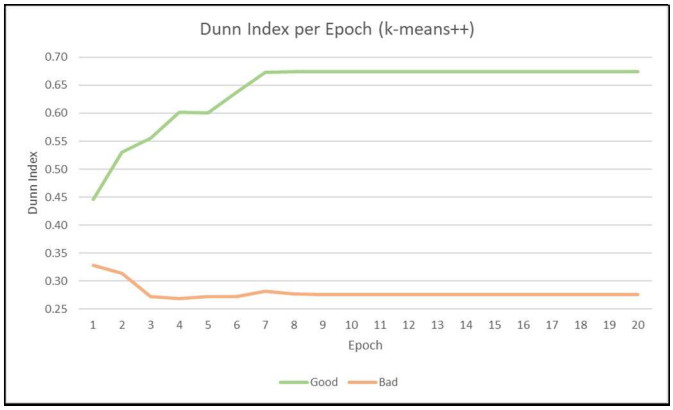
This demonstrates the expected behavior of the algorithm: a higher Dunn Index meaning better clustering. It is clear that the `k` value of `15` is optimal, at least for `S1`. Concerning the other

sets, due to the reduced separability of the clusters, the performance is impacted. However, a relatively high Dunn Index is still found for nominal k.

Interestingly, for both initialization strategies, there is a turbulent curve leading up to the nominal k, with a dramatic drop after. To use S1 as an example, a k of 5 and 7 is found to be very close to nominal whereas any k above 15 is found to be much worse at clustering. Of special note is S4 is difficult to cluster irrespective of k value chosen, as the Dunn Index is equivalent up to a k of 15 with the characteristic drop after.

An analysis of the Dunn Index and how it changes over each epoch is also possible using the K-Means++ strategy:

GRAPH IV. K-MEANS++ DI PER EPOCH



Graph IV. Dunn Index compared over epochs, k=15, S1.

As before, a sample of "good" and "bad" runs of the algorithm are aggregated and averaged. A poor K-Means using K-Means++, not unlike random initialization, stagnates early and converges to a suboptimal solution immediately. Contrastingly, however, a great result has Dunn Index increase over time until an optimum is found. This is the expected behavior of K-Means.

What is interesting about this graph is the initial Dunn Index for a good result is higher than the initial Dunn Index for a poor result. This harkens back to random initialization whereby a poor random initialization yields a poor result. K-Means++ is not immune to this fact; K-Means++ is not perfect [4], as in it does not produce a perfect result every time, the likelihood of a better initialization is however higher.

The significance of initialization strategy is obvious: provided a random initialization strategy has failed to produce enough data to compare yet K-Means++ has, K-Means++ is considered a better alternative to random initialization.

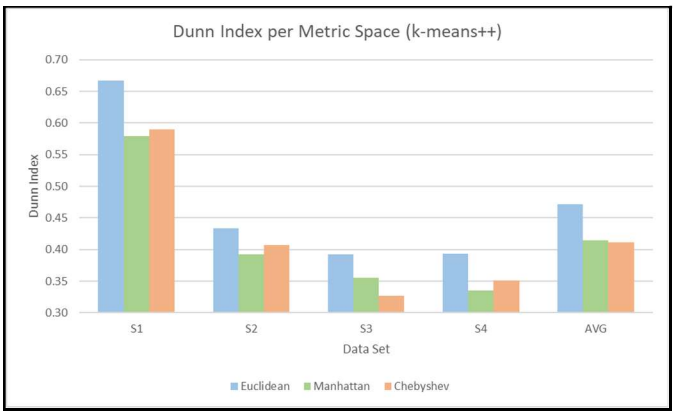
There is but one caveat to this conclusion, however: using random initialization, centroids can be placed in linear time ($O(n)$) whereas the K-Means++ approach is slightly slower in logarithmic time ($O(n \log n)$). However, the results are much more consistent (consistently favorable) [5] using K-Means++.

B. Distance Metric

Using both a nominal k value and initialization strategy, the effect of the distance metric used can be examined. The k value and initialization strategy remain constant and at optimal values to compare only the distance metric between data sets: using poor parameter choices for this test may skew results or show less significance than otherwise.

The results for each data set using each distance metric, alongside an average of all cases is below:

GRAPH V. DI PER DISTANCE METRIC



Graph V. Dunn Index per distance metric, S1-S4, k=15, KM++.

On comparing the distance metrics, an ANOVA test is performed to determine statistical significance of the parameter being variable. The data used is the average case between all data sets. An ANOVA test will determine if changing the distance metric matters in determining the Dunn Index, with a null hypothesis of no significance.

The f-statistic for this test yields an $F = 11.6885$ with a p-value of 0.00022 which is lower than the hypothesis p-value of 0.05, thus the null hypothesis is rejected and there is indeed relevance to the distance metric used.

It is clear from visualizing that using Euclidean metrics yields the best performance, however Manhattan and Chebyshev metrics show little difference. To verify this, another ANOVA test can be performed on just those two samples, yielding an f-statistic of $F = 0.03721$ with a p-value of 0.849, much higher than the null hypothesis p-value of 0.05, thus the null hypothesis is accepted: there is no statistical significance in choosing one of those two metrics over the other.

Previously mentioned was that each distance metric requires different computational time to evaluate. To quantify the difference, the algorithm is adept at displaying these differences as each run of the algorithm performs hundreds of thousands of distance calculations.

Over several runs, the time of each execution is tabulated and averaged:

TABLE I. TIME PER EXECUTION

Distance Metric	*Time*
Euclidean	7.383s
Manhattan	6.486s
Chebyshev	6.635s

Table I. Comparing execution time per distance metric.

While Euclidean metrics is decidedly slower than the other two, it is not a significant difference, at least for the data sets used. For a 13.8% increase in execution time versus Manhattan metrics, for example, there is a 14.6% increase in Dunn Index.

C. K-parameter

Referring to **Graph I** and **Graph III**, several values for k were tested for each data set. Despite random initialization being a poor strategy for centroid generation, some information can still be inferred from both graphs.

Predominant between both strategies is that a less than nominal k value can still adequately cluster the data; it is only when you exceed the optimal k does performance deteriorate rapidly. This is prevalent for all data sets.

Using K-Means++, if you consider the maximal k before the dramatic drop in Dunn Index as the nominal k, the optimal clustering is found for these values for each data set:

TABLE II. OPTIMAL K PER DATA SET

Data Set	*Optimal k*
S1	15
S2	14
S3	15
S4	13

Table II. Optimal k parameter found empirically for each set.

Finding k this way is referred to as "elbowing", where the Dunn Index may increase or remain close to stationary for successive values, and then dramatically drop at a certain k value. This can be construed as the optimal k.

As to why every test does not determine 15 to be optimal k: this is due to the clusters having less separability and how clusters overlap in some data sets. S1, having the most separability, shows clear distinction between clusters and thus the K-Means algorithm can perform better.

V. CONCLUSION

K-Means clustering is sensitive to the parameters used, primarily the initialization strategy used, as the probability of achieving good results is diminished when using a naïve approach with random centroid initialization. The K-Means++ initialization strategy instead performs a more "informed" initialization and leads to greater results.

The type of distance metric used has a significant effect on the performance of the algorithm, although the difference is not dramatic. Using Euclidean metrics will yield greater accuracy of results but at a slower computation time. Contrastingly, between Manhattan and Chebyshev metrics, both perform approximately the same. There is indeed a trade-off between accuracy and execution time, however.

Using the Dunn Index, an optimal k can be found; additionally, comparing the k values can elucidate how well that k is able to cluster the data. Not only can an optimal k be discovered, but a range of appropriate k values can be approximated by comparing their Dunn Indices on each execution of the algorithm.

REFERENCES

[1] E. W. Forgy, "Cluster analysis of multivariate data: efficiency versus interpretability of classifications," Biometrics, vol. 21, no. 3, pp. 768–769, 1965.

[2] G. Hamerly and C. Elkan, "Alternatives to the k-means algorithm that find better clusterings," 2002.

[3] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "'NP-hardness of Euclidean sum-of-squares clustering," Machine Learning, vol. 75, no. 2, pp. 245–249, 2009.

[4] S. Vassilvitskii and D. Arthur, "k-means : the advantages of careful seeding," 2007, pp. 1027–1035.

[5] S. Vassilvitskii and D. Arthur, "How slow is the k-means method?," ACM New York, pp. 144–153, 2006.

[6] J. C. Dunn, "Well-Separated Clusters and Optimal Fuzzy Partitions," Journal of Cybernetics, vol. 4, no. 1, pp. 95–104, 1973.

[7] D. J. Ketchen, "The Application of Cluster Analysis in Strategic Management Research: An Analysis and Critique," Strategic Management Journal, vol. 17, pp. 441–458, 1996.