CHARACTERIZING KMEANS CLUSTERING METHODS TO ACCELERATE EXPERIMENT RUN-TIMES

by

YAFFE KRAVITZ

A.A.S., Clinton Community College, 2010
B.S., University of Colorado Denver, 2016

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Master of Science
Electrical Engineering
2017
This thesis for the Master of Science degree by

Yaffe Kravitz

has been approved for the

Electrical Engineering Program

by

Dan Connors, Advisor

Chao Liu

Stephen Gedney

Date: July 29, 2017
CHARACTERIZING KMEANS CLUSTERING METHODS TO ACCELERATE EXPERIMENT RUN-TIMES

Thesis directed by Professor Dan Connors

ABSTRACT

Machine learning algorithms have the potential to unlock solutions to ambitious problems in a myriad of scientific and industrial fields. Clustering is an unsupervised machine learning technique in which data objects are grouped, based on similarity, into clusters. Objects within a cluster are alike in some mathematical way and objects in different clusters are distinctly different. In this way, clustering uncovers the hidden structure of a dataset to discover how many data objects are similar and the number of clusters. Unfortunately clustering techniques vary in two experimental ways: clustering execution time and the accuracy of the final clustering results. Furthermore, many clustering algorithms require multiple experimental evaluations that, in turn, generate prohibitive clustering execution time [1].

Kmeans is the primary clustering algorithm in the data mining and unsupervised machine learning spheres that defines a single clustering experiment by defining the set of similar points that are each closest to 'K' cluster centroids. The algorithm finds a solution of centroids in the dataset by iteratively calculating the multi-dimensional 'D' distances between each of the experimental 'K' cluster centers and the 'N' data points. While the base Kmeans algorithm is parallelizable across these iterations, there are two exacerbating problematic conditions of the algorithm’s use: starting seed selection and the selectivity the number of K-centroids for a dataset. Each of these conditions requires multiple 'E' experiments to evaluate within each dataset. The work outlined in this thesis investigates the optimization and the restructuring of the Kmeans algorithm to use multiple sub-samplings of a target dataset to generate accurate K-centroid seeds that eliminate excessive iterations of the base clustering algorithm.
The form and content of this abstract are approved. I recommend its publication.

Approved: Dan Connors
DEDICATION

I dedicate this work to my love, Rebekah, and our four amazing children, Anna, Sarah, Jonathan, and Samuel who have stood by me and supported me throughout my educational career. Without their love, support, and never-ending supply of patience, this endeavor would have been nearly impossible.
ACKNOWLEDGMENT

This thesis would not have been possible without the generous support of Professor Dan Connors. The valuable expertise and depth of knowledge in the computer engineering field that Dr. Connors provided was instrumental in achieving this milestone. I also owe a great debt of gratitude to those who encouraged and motivated me to start this journey and see it through to completion.
# TABLE OF CONTENTS

CHAPTER

I. INTRODUCTION .................................................. 1

II. BACKGROUND .................................................... 4
    Machine Learning .............................................. 4
    Categories of Learning ....................................... 5
        Supervised, Semi-supervised, and Unsupervised Learning .. 5
    Regression ..................................................... 6
    Classification ............................................... 6
    Clustering ................................................... 9
    Dimensionality Reduction ................................. 10
    Kmeans Clustering ........................................... 11
        Initialization Methods ................................ 17
        Initialization Sensitivity .............................. 17
        Cluster Affinity ......................................... 21
    Available Kmeans Code Packages ......................... 22
    Variations of Clustering Algorithms ....................... 22
    Kmeans Algorithms ......................................... 23
    Machine Learning Datasets ................................. 23

III. APPROACH .................................................... 25
    Refined Initialization of Centroids ....................... 25
    Running the Experiments ................................. 28

IV. EXPERIMENTAL RESULTS .................................. 31
    Results ..................................................... 31
        Algorithm vs Clustering Time ......................... 31

V. CONCLUSION .................................................. 40

REFERENCES ..................................................... 41
CHAPTER I
INTRODUCTION

Machine learning algorithms have the potential to unlock solutions to ambitious problems in a myriad of scientific and industrial fields. Traditional machine learning concepts can be divided into four categories: regression, classification, clustering, and dimensionality analysis. Varieties of algorithms in each of these machine learning domains require multiple parameters to explore to achieve the optimal model and analysis. As clustering is one approach to extracting hidden structures of data, it is significantly useful at the start of exploring the utility of a dataset. For that reason, it is critical to cut through the computational barriers of the deployment early in an investigation of a dataset to unearth advantageous insights related to exploring the efficiency and expediency of machine learning concepts of regression and classification.

Cluster analysis (or clustering) is the task of grouping a set of similar objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters) [2]. The clustering task is deployed in exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics [3].

While the overall concept of clustering is to place data objects into groups based on similarity, the definition of similarity may vary depending on the intended goal of exploring the dataset. In general, objects within a cluster are alike in some mathematical way and objects in different clusters are distinctly different. Common views of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions. Similarly, clustering uncovers the hidden structure of a dataset to discover how many data objects are similar and the number of clusters [4]. Cluster analysis is not one specific algorithm
but the general task to be solved. In this way, there are many various clustering algorithms that differ significantly in their notion of what constitutes grouping similarity in a cluster and how efficiently to find the similarity within clusters.

Kmeans is one of the primary clustering algorithms used in the data mining and unsupervised machine learning environments. Kmeans defines a single clustering experiment by defining the set of similar points that are each closest to 'K' cluster centroids. The algorithm finds a solution of centroids in the dataset by iteratively calculating the multi-dimensional 'D' distances between each of the experimental 'K' cluster centers and the 'N' data points. While the base Kmeans algorithm is parallelizable across these iterations, there are two exacerbating problem space conditions of the algorithm’s use: starting seed selection and the selectivity of the number of K-centroids for a dataset. Each of these conditions require multiple 'E' experiments to evaluate within each dataset. Thus to explore the hidden structure of a single dataset requires significant computational work in calculating 'E' experiments of 'K' centroids for an 'NxD' sized data [5].

The bulk of the Kmeans calculations are performed to find centroids that minimize a mathematical score between the data-points and each centroid. One of the inherent issues with the Kmeans algorithm arises in the way the initial starting centroids are selected and that a number of initial iterations are exhausted to migrate centroids from their initial seed positions to positions within the multidimensional space that match the inherent characteristics. There are a number of approaches to initializing Kmeans centroids, most commonly is to use randomly selected data points and other generalized heuristics.

The refinement Kmeans approach overcomes the computational barrier of clustering by sampling a large dataset to uncover candidate centroids to initiate the search of initial seed points for the full dataset. This thesis investigates the optimization of the refinement Kmeans algorithm by evaluating various sampling sizes across a wide
range of target datasets. Moreover, a variety of Kmeans algorithms are evaluated to additionally determine further impact to the computational requirements of finding the optimal clustering in a dataset. The goal is to uncover data characteristics that might guide data science researchers in setting optimal Kmeans evaluation parameters that overcome the prohibitive computational costs of performing full dataset exploration. The methods for characterizing the results are clustering execution time and the clustering affinity (a measure of accuracy of the final clustering results).

This thesis is organized as follows: Chapter II discusses the background of the Kmeans clustering algorithm. Chapter III examines the Kmeans refinement approach and thesis experimental framework for evaluating multiple clustering evaluations. The experimental results section, Chapter IV, shows performance data for the clustering of selected datasets. Finally, Chapter V concludes this thesis.
CHAPTER II  
BACKGROUND

Machine Learning

Machine learning is a subfield of computer science that refers to the equipping of computers with the ability to learn without explicit programming. The terms “machine learning” and “artificial intelligence” will often be used interchangeably and, while the concepts are connected, there are differences. The goal of artificial intelligence is to create a machine that can mimic the human mind. To do this, it will need a machine to have learning capabilities. Artificial intelligence also encompasses knowledge representation, abstract thinking, and reasoning. Machine learning’s only focus is to create software that can learn from past experience - hence the term “machine learning”. Thus, while artificial intelligence is the embodiment of recreating human characteristics in a machine, the focus of machine learning is only on the development of computer programs that can change and learn when exposed to new data.

Three basic elements of machine learning are seen throughout the tens of thousands of machine learning algorithms in existence, not to mention the hundreds of new algorithms developed every year. These three components are as follows:

- **Representation**: display of knowledge
- **Evaluation**: judging of hypotheses
- **Optimization**: generation of models

Representation simply refers to how the knowledge is represented. For example, depending on the characteristics of the dataset, it may be understood best with either a decision tree or a set of rules. Instances, graphical models, neural networks, and model ensembles are all examples of ways knowledge can be represented.
Evaluation is the way hypotheses are judged (or “evaluated”). Accuracy, prediction and recall, mean squared error, cost, and likelihood are all common examples of evaluation.

Optimization is the way models are generated; this is also known as the search process. Examples include combinatorial optimization, convex optimization, and constrained optimization. All machine learning algorithms are a basic combination of these three components. In other words, these three categories represent the framework to understand any machine learning algorithm [6].

**Categories of Learning**

Machine learning can be summed up into four different categories: classification, regression, clustering, and dimensionality reduction (see figures II.1 - II.8). These four categories encompass the world of supervised, semi-, and unsupervised learning (sometimes, reinforcement is also included but can be considered a subcategory to the unsupervised learning category and not really a separate category).

- Regression: predicting a value
- Classification: predicting a type or category
- Clustering: finding the hidden structure of data
- Dimensionality Reduction: getting the set of principal variables in data

**Supervised, Semi-supervised, and Unsupervised Learning**

Classification and regression are quite similar - for both systems, the goal is to generate a function to predict values for unseen observations. It is important that during the training of the function, labels are available to the algorithm.

Regression falls under the supervised learning focus, while classification is considered to be a semi-supervised technique. To assess the performance of a trained model,
the labels of real observations can be compared to the predicted labels. Labeling can be a tedious burden and is often done by humans, so techniques that do not require labeled data are sought out and they are known as unsupervised learning. An example of unsupervised learning includes clustering, where groups of observations that are similar are located. It does not require specific labeled observations. Assessing the performance of this trained model is more difficult than a more straightforward supervised learning algorithm because there are no real labels to use for comparison.

Some techniques overlap between the supervised and the unsupervised learning mediums into a land of gray. This so-called “semi-supervised learning” could involve a lot of observations that are not labeled and a few which are labeled. Clustering can be used to group similar observations and the information about the clusters (and the few labels) can be used to assign a class to the unlabeled observations (then this gives us more labeled information to perform unsupervised learning on) [7].

**Regression**

Regression problems are machine learning problems that try to predict an input value based on previous information. The variables input are called predictors and the output is known as the response (see Figure II.2). In some ways, regression and classification are very similar in that they both exemplify trying to estimate a function that maps input to output based on earlier observations. In regression, however, the search is to find an actual value, not just a class of an observation. An example of what regression could be useful for would be modeling credit scores based on a person’s record of payments. All regression models have at least two things in common: first, the response is always quantitative and second, they will always need input knowledge of previous input-to-output observations to erect a model.

**Classification**

A classification problem involves predicting whether a given observation belongs to a certain category. Classification is the process of taking an input and mapping it
to a discrete label. For example, this is the case when an image is being assigned to the label of either a “cat” or a “dog”. With only two choices, this would be known as two-class or binomial classification. Often times, there are many more categories (for
instance, when trying to predict the winner of a sports tournament), and the problem is known as multi-class classification.

Figure II.3: ScikitLearn Algorithm Cheat Sheet - Classification [8]

Classification involves training on earlier observations to create a classifier that will try to predict the class of future observations. In the binomial classification referenced above, a model would predict the difference between a dog and a cat given a photo with one or the other. The system might be trained on many different observations of dogs as well as many different observations of cats, to come up with classifiers that would be able to tell if a new, previously unseen, photo of a dog or cat is either classified as a dog or classified as a cat (see Figure II.4). Once a classifier is constructed, it can be used to take arbitrary input (like a photo of a dog or cat), and label the image as a dog or cat (see Figure II.5).

Figure II.4: Classification: Building a Classifier

The possible applications of this technique are broad and promising. For example, after significant studies of medical examinations determine that certain vital signs and
symptoms are related to a specific disease, that information can be used to build a predictor that will allow a reasonable prediction to be made that a new patient with unseen signs and symptoms might have this disease and may need treatment. A simple example would be the yes-or-no prediction simplicity of “is this tumor cancerous?” A basic question with profound implications. It is important to use a qualitative output and to remember that there is foreknowledge of the classes to which observations can belong.

**Clustering**

Clustering is an attempt to group similar objects together into clusters where the clusters themselves are dissimilar. Good clustering methods produce high quality clusters with high similarity within clusters and low similarity compared to other clusters [9]. This similarity can be calculated by using the notion of “affinity”. Clustering is different from classification and regression in that there is no need for foreknowledge of the labels and sometimes there is no clear right or wrong in clustering (see Figure II.7). Clustering can be useful when there is no previous knowledge about any patterns inside samples of data. Different clusterings can reveal new information (sometimes useful information) about a potential hidden structure within a data set. A practical example of clustering could be segmenting a market of potential consumers based on their demographic features and purchasing histories. Another example of clustering could be to find groups of movies based on reviews of movies or features in each movie to find movies most like another movie [10].
Dimensionality Reduction

In machine learning, dimensionality reduction is the process of reducing the number of random variables in consideration by obtaining a set of principal variables. The process is useful to speed up algorithm execution and it can actually be helpful in the final clustering accuracy or classification. If there is faulty input data or too much
noise, a less than desirable algorithm performance can be seen. Removing data that is actually dis-informative could find more general classification regions and rules and better performance can be seen on new data[11].

One of the ways to reduce input dimensionality involving a high number of missing or irrelevant values is called Principal Component Analysis (PCA). PCA is a statistical procedure for identifying the principal components or the number of uncorrelated variables from a large set of data. To explain the maximum amount of variance with the fewest number of principal components is the chief concern of PCA. Usually PCA is just one step in a series of analyses and is used to reduce the number of variables to avoid multicollinearity. The social sciences and marketing research arenas commonly use principal components analysis for their often very large data sets. Figure II.8 shows the ScikitLearn overview guidelines of using PCA for dimensionality reduction.

![Figure II.8: ScikitLearn Algorithm Cheat Sheet - Dimensionality Reduction [8]](image)

**Kmeans Clustering**

The Kmeans algorithm is a clustering algorithm that seeks to minimize the Euclidean distance between centroids and the members associated with those centroids. In reality, it attempts to find a hidden structure in a dataset, if one exists. k (number of clusters) is specified manually and a dataset is provided while the algorithm does the rest. It will always output a solution, but the solution may not be correct. If the goal is to cluster based on some similarity, the answer Kmeans provides may not always meet that criterion.
Kmeans is known for its simplicity and its ability to be used for a myriad of data types. The Kmeans algorithm is a clustering method defined by its partitioning based, non-hierarchical methodology. Given a set of objects \( X \), Kmeans partitions every data point \( x_i \) into a cluster \( c_j \) (where \( j \leq k \) and \( k \), the specified number of clusters, is a parameter that must be provided) that minimize the contained group sum of squared errors. Partitioning is done in such a way as to minimize the objective function.

\[
\sum_{j=1}^{k} \sum_{x_i \in c_j} ||x_i - \mu_j||^2
\]

where \( \mu_j \) is the centroid of cluster \( c_j \).

The kmeans algorithm begins by initializing \( k \) cluster centers. The data points are then assigned to one of the existing clusters in accordance with the square of the Euclidean distance from the center of each cluster (centroid). The centroid of each cluster is then updated to become the mean center of all the data points in the cluster as a result of the change in membership. The process of assigning and re-assigning membership and updating the centroids is repeated until there is no more change in the value of any of the cluster centers, also known as convergence.

Assignment step: Every point \( x_i \) in \( X \) is assigned to the closest centroid’s cluster.

Update step: Centroid of every cluster is recalculated based on the mean point of all the data points.

One way to make kmeans work more expediently is to pick the best value for \( k \). Another way is to intelligently select the starting centroids.

See Figures II.9 - II.16 for an example of kmeans in action:
Figure II.9: Cluster Initialization

Figure II.10: Membership Assignment
Figure II.11: Moving Centroids to Mean of Data Points

Figure II.12: Membership Reassignment
Figure II.13: Moving Centroids to Mean of Data Points

Move centroids to mean of points in clusters

Figure II.14: Membership Reassignment

Re-assign membership based on distance of points to centroids
Figure II.15: No Movement - Centroids at Local Minimum

Figure II.16: Convergence - No Change in Membership
Initialization Methods

There are several different methods used to initialize Kmeans, of them, the common ones are Forgy and Random Partition [12]. Random Partition does just that, randomly assigns all the data points to different clusters and then finds the mean of each of the clusters and uses the means as the initial starting centroids. Forgy takes the initial means and spreads them out more, whereas the Random Partition method tends to have the means closer to the center. Another initialization method, the one used in this thesis, by Bradley and Fayyad [13] generally does much better for getting the clustering results. Kmeans++ is another algorithm that initializes the centroids differently:

1. Choose one center uniformly at random from among the data points.

2. For each data point $x$, compute $D(x)$, the distance between $x$ and the nearest center that has already been chosen.

3. Choose one new data point at random as a new center, using a weighted probability distribution where a point $x$ is chosen with probability proportional to $D(x)^2$.

4. Repeat Steps 2 and 3 until $k$ centers have been chosen.

5. Now that the initial centers have been chosen, proceed using standard Kmeans clustering.

Initialization Sensitivity

Kmeans is an extremely valuable tool in the machine learning toolbox, but it doesn’t come without flaws. In computing, there is no such thing as a perfect algorithm that applies universally across the board to every possible dataset. Kmeans
is no exception. One of the issues with Kmeans is that it is highly sensitive to initialization. What that means is on the first step in the algorithm, centroids must be assigned before the iterative process begins. One of the iterative steps is to measure the distance from data points to centroids, so those initial centroids must be placed somewhere to start. It has been an area of much research to figure out a good way to initialize the centroids, and there are some great solutions out there, but this thesis proposes expanding on some of the ideas to improve the Kmeans algorithm even more.

There have also been several attempts to improve the efficiency and time it takes to run the Kmeans algorithm. Because of its iterative nature, larger dimensionality datasets can take a significant amount of time to produce a result. Improving the efficiency and speed that Kmeans takes to perform clustering alone is not much help if it produces an incorrect answer, an incorrect answer produced in a shorter amount of time is not usually helpful.

Improving the initialization can actually speed up the algorithm because if the starting centroids can be placed intelligently in such a way that they are closer to where the final iteration would place the centroids, the number of iterations needed to converge is usually reduced, which typically equates to a reduction in total clustering time.

In some cases, Kmeans is not capable of ever finding the correct clustering result because of the nature of the dataset. In the following sample dataset, Kmeans did not cluster the result correctly (see Figure II.7) and never would correctly cluster it because the hidden structure is not one that a mean-based clustering algorithm would find. For this example, a density based algorithm, such as DBSCAN, would be more suitable to carry out this task (see Figure II.18).

While Kmeans is probably the most intuitively straight-forward clustering method (where an algorithm groups data in k clusters based on some measure of similarity),
it is not without its setbacks. This algorithm’s speed and simplicity are unparalleled, however, one of the issues with Kmeans is that if it is not seeded with good initial centroid points, it can end up with an incorrect clustering, even for data that should be easily clustered.

For example, Figure II.19 shows an initial seeding of centroids where the initial points were selected randomly. In the resultant clustering, figure II.20 shows how a poor initialization selection for the centroids can result in a bad clustering result -
this would show up as a poor affinity score compared to the correct clustering.

Figure II.19: Clustering (k=3) - Poor Initial Seeding of Centroids with Random Initialization

Figure II.20: Clustering (k=3) - Poor Initial Seeding of Centroids Leads to Poor Clustering Results

The refinement algorithm has the following merits of operation:

• performs Kmeans on a subset of sampled data so that the system completes much faster.

• the answer potentially is better as it is derived by multiple findings, compared to that of the naive Kmeans that starts from a single set of centroids.
Cluster Affinity

A concept for evaluating the assignment of data objects to clusters is a mathematically derived score known generally as affinity. In short, affinity is a calculated measure of how well data objects are assigned to clusters. Figure II.21 details an example of affinity. If there were only 2 data points, with 2 clusters (k=2), affinity would be zero because the centroids would be the data points and the data points would be 0 distance from each of their respective centroids.

![Figure II.21: Affinity for k=2 with Two Data Points](image)

If there were 3 data points and 2 clusters (k=2), then the resultant clustering might look like Figure II.22, where the affinity for one of the data points is zero, and the affinity of the other centroid is the average sum of squared distances of the data points to the cluster’s mean center point (centroid).

![Figure II.22: Affinity for k=2 with Three Data Points](image)
Available Kmeans Code Packages

Several prominent code packages include the Kmeans clustering algorithm:

- OpenCV - http://docs.opencv.org/2.4/modules/core/doc/clustering.html
- R - https://www.rdocumentation.org/packages/stats/versions/3.4.0/topics/Kmeans
- ELKI (Environment for DeveLoping KDD-Applications Supported by Index-Structures) https://elki-project.github.io/tutorial

For this thesis, the mlpack Kmeans implementation is selected because it provides the necessary setup to carry out multiple experiments and provides the needed output information with minimal recoding. A Python framework that allows for carrying out several experiments with initializing the centroids was written for this thesis.

Variations of Clustering Algorithms

- Kmeans: Lloyd’s algorithm (the standard Kmeans algorithm)
- Jenks natural breaks optimization: Kmeans applied to univariate data
- Minkowski Weighted Kmeans: computes weights for features at each cluster
- Kmeans++: improved initial centroid seeding
- Mini-batch Kmeans: runs Kmeans on “mini batch” samples of larger datasets that exceed memory space
- Spherical Kmeans: for textual datasets
- Fuzzy C-Means Clustering: data points assigned to multiple clusters
**Kmeans Algorithms**

Several Kmeans algorithms will be evaluated in the thesis for their inherent differences when locating centroids under different circumstances: naive, pelleg-moore, elkan, hamerly, and dualtree. List of the Kmeans algorithms included in the mlpack [14] Kmeans program that were used:

- **naive**: the standard Lloyd iteration; takes $O(kN)$ time per iteration.

- **pelleg-moore**: the 'blacklist' algorithm, which builds a kd-tree on the data. This can be fast when $k$ is small and the dimensionality is reasonably low.

- **elkan**: Elkan’s algorithm for k-means, which maintains upper and lower distance bounds between each point and each centroid. This can be very fast, but it does not scale well to the case of large $N$ or $k$, and uses a lot of memory.

- **hamerly**: Hamerly’s algorithm is a variant of Elkan’s algorithm that handles memory usage much better and thus can operate with much larger datasets than Elkan’s algorithm.

- **dualtree**: the dual-tree algorithm for k-means builds a kd-tree on both the centroids and the points in order to prune away as much work as possible. This algorithm is most effective when both $N$ and $k$ are large.

**Machine Learning Datasets**

The approach to testing new clustering approaches will evaluate several datasets to capture different data characteristics and trends.

The following datasets (see Table II.1) with multiple height and width (dimensionality) variations were run using several different sample sizes along with different percentages of the original dataset. The datasets are included from the UC Irvine Machine Learning archive website [15].
Table II.1: Machine Learning Datasets

<table>
<thead>
<tr>
<th>Data File</th>
<th>Number of Observations</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>131000</td>
<td>35</td>
</tr>
<tr>
<td>Chem</td>
<td>47832</td>
<td>155</td>
</tr>
<tr>
<td>Diabetic</td>
<td>13820</td>
<td>41</td>
</tr>
<tr>
<td>GroupLens</td>
<td>100000</td>
<td>3</td>
</tr>
<tr>
<td>HAR</td>
<td>26523</td>
<td>127</td>
</tr>
<tr>
<td>HYG</td>
<td>119618</td>
<td>14</td>
</tr>
<tr>
<td>Iris</td>
<td>149</td>
<td>4</td>
</tr>
<tr>
<td>KDDCUP</td>
<td>131000</td>
<td>74</td>
</tr>
<tr>
<td>Poker</td>
<td>131000</td>
<td>11</td>
</tr>
<tr>
<td>Power</td>
<td>131000</td>
<td>6</td>
</tr>
<tr>
<td>Power2</td>
<td>1000</td>
<td>6</td>
</tr>
<tr>
<td>RelationNetwork</td>
<td>53413</td>
<td>22</td>
</tr>
<tr>
<td>Test</td>
<td>1000</td>
<td>3</td>
</tr>
</tbody>
</table>
CHAPTER III

APPROACH

Refined Initialization of Centroids

The common approach to using Kmeans clustering is to generate a minimum of 10 experiments for a full target dataset. The machine learning standard of scikit-learn deploys 10 experiments and carries out the affinity scoring to select the best final clustering. Running 10 experiments on large datasets is both time consuming and compute intensive. Many environments have limited computational power for an excessive number of trials. To get around needing to run multiple experiments on the entire dataset, the goal is to run multiple experiments on smaller samples of the dataset. Running Kmeans on 1%, or even 5%, of the original dataset saves a significant amount of time. The goal is to get the same clustering performance in much less time and with much less computing power usage.

The method proposed by Bradley and Fayyad in their paper “Refining initial points for Kmeans clustering” [13] is implemented in mlpack. This strategy samples points from the dataset and runs Kmeans clustering on those points multiple times, saving the resulting clusters. Then, Kmeans clustering is run on those clusters, yielding the original number of clusters. The centroids of those resulting clusters are used as initial centroids for Kmeans clustering on the entire dataset. The approach is as follows: starting with an original dataset with 1 million data points (shown in Figure III.1.), multiple samples (8 samples of 1% of the original dataset) are gathered and can be visualized in Figure III.2. Kmeans is then run on all samples as depicted in Figure III.3. The final step of the refinement process is to take the 8 clusterings and run Kmeans again on them. The result provides the mean points of the clusters to use as the starting centroids for the final Kmeans run on the original dataset. Figure III.4) presents the final Kmeans run on the original dataset.
Figure III.1: Simulated Original Data

Figure III.2: Eight Random Samples - Each Containing 1% of Simulated Original Dataset (10,000 Datapoints per Sample)

Figure III.3: Kmeans Clustering on Sub-Samples for k=4

Figure III.5 shows the clustering when the initial centroids are seeded from the combined centroids and Kmeans is run on the original data. Figure III.6 illustrates the run of Kmeans clustering on the original data set with the intelligently selected
Figure III.4: Combined Centroids from Subsamples - Getting the Mean of Each

Figure III.5: Seeding Kmeans with Intelligently Selected Starting Centroids

Figure III.6: Kmeans on Simulated Original Dataset with Refinement

starting centroids. Figure III.7 compares the clustering to randomly selected starting centroids. It shows how initializing the centroids to a “good” location can have a significant impact on the amount of computing required to reach the final clustering.

Figure III.7: Kmeans on Simulated Original Dataset without Refinement
Running the Experiments

As shown in Figure III.8, the process to begin running the experiments requires the import of relevant python modules, defining global variables, and providing the input and output path structure. The second step involves setting up the experimental parameters. This can be done either by editing the python script directly, or modifying a configuration file that holds all the values that should be run by the script. The script begins by scanning the input directory for data files. Once the data files are found, the python module loads the paths, filenames, and the size and dimensionality information into a dictionary. The output paths are also set up based on the experimental parameters. The path dictionary is indexed in a way that makes it easy to determine each experimental file output.

As mlpack-Kmeans produces multiple data files for each Kmeans run, it is necessary to establish an intuitive file naming convention for the output files. Each run of mlpack-Kmeans produces 3 output files: a centroids file that contains the coordinates of the centroids found, an associations file that includes the original dataset along with an appended column to show cluster membership of each row, and a logfile that contains the information about the run, including clustering time, number of iterations to convergence, and any errors or other messages associated with the run. As an example, an experimental run on a single dataset with 4 different algorithms, 4 values of k, 4 sample sizes, and 4 sampling percentages will produce 256 calls to mlpack-Kmeans and 768 output files. The convention that is used puts the parameter values right in the file name. For example, HAR_4_100_0.10_elkan is the filename base for the experiment that runs on the HAR dataset with a value of k=4, 100 samples of 10% sampling, and the elkan clustering algorithm.

Figure III.9 shows the remainder of the process. Once the paths are defined, the python script sets up the command to send to the operating system to run the mlpack-Kmeans algorithm. There are several parameters that are passed to the program and
the python script formats those based on the experimental setup described previously. It will parse through all the experimental parameters and run the algorithm on the given dataset(s). While this process is running, after generating an assignment file, there is a python function to reduce the size of the assignments file by removing the original dataset from the file and leaving the remaining cluster membership information. Once the original dataset is removed, the file is renamed to ‘membership’. Without this process, the original dataset would be reproduced hundreds of times and the limited hard drive space would be consumed.

After running the Kmeans algorithm, there is a separate function that calculates the affinity of each run. This process parses through all the experimental runs and calculates the affinity based on the membership file, the centroids file, and the membership file. It uses the centroids file to establish the centroid coordinates, and the membership file to establish which points (rows) in the original dataset file corre-
spond with which cluster. The affinity is summed and a weighted score is recorded in another file labeled 'affinity' - effectively adding a fourth file to the output run.

Once all the results are produced (it can take several hours to run all the experiments), another function gathers the results into a data dictionary in python. Clustering time is extracted from the log file of each run and the affinity is pulled from the affinity file from each run. The data dictionary provides convenient access to the data without having to pull it from the files each time a plot is generated. The data is then plotted to display relevant trends and allow conclusions to be made about the different aspects of the experimental run.

Figure III.9: Code Path to Calculate and Assess Affinity
CHAPTER IV
EXPERIMENTAL RESULTS

The quality of the clustering is important as it provides insight into how well the Kmeans algorithm performed. Affinity is the term for how close the data points are to the cluster they are members of after a Kmeans algorithm has completed a run. Execution time alone is not as critical as getting the correct result. Getting an incorrect result faster is not a solution that provides any help to the task. The final goal of this thesis is to try and establish a matrix (or score card) that shows what algorithm, sample size, and percent value (percent of original dataset for the samples) should be used when running the Kmeans algorithm on a given type or size of dataset. If execution time is more important than accuracy, then one algorithm might do the trick, and 10% sampling with 100 samples might not be the way to go. If the quality of the clustering result (affinity) is more important, then there may be an algorithm and sampling that works better on a particular dataset.

If the experiment is run with 10% sampling and 100 samples, the execution might be five times worse in execution time than the base, unrefined run. The quest might be for better affinity and the execution time might not matter as much. In this case though, if the results are the same as the unrefined run, the one with better execution time is typically the better choice.

Results

Algorithm vs Clustering Time

Figure IV.1 shows the various Kmeans clustering algorithms for the dataset of KDDCUP using k=5. The results are standardized to use naive execution time as the base scaled to 1. In that way, any Kmeans algorithm that runs faster than naive will have a lower bar plot and value below 1. The figure demonstrates that the different Kmeans algorithms have the capacity to achieve significantly faster execution than naive. The hamerly algorithm is approximately 5 times faster than base Kmeans,
although both the elkan and pelleg-moore techniques are close contenders respectively performing 3x and 2x faster than naive.

Figure IV.1: KDDCUP Dataset: Algorithms vs. Clustering Time for $k=5$

Figure IV.2 shows the trend between the dataset, clustering time and $k$. The trend of the results show that as $k$ increases, it takes longer to cluster the dataset. KDDCUP, RelationNetwork, and HAR are larger datasets in terms of both observations and dimensionality, so it is expected that they would take longer to cluster. The resulting function of this figure helps to characterize the influence of running different algorithms for different datasets. The analysis of the experimental results can be used as a valuable reference for how to run the Kmeans algorithms on different shaped datasets. One algorithm might go really fast, but at the expense of a good clustering outcome – another algorithm might be slow, but provide a much better clustering solution. The trend of increasing the $K$ number of clusters increases the execution time of clustering which has a higher impact for some datasets.

Figure IV.3 shows the trend between the HYG dataset, weighted affinity (or per-point affinity) and the value of $k$. The results show that there is a decreasing trend
for affinity as k increases, but for k=6, the affinity was actually worse than when k=3. This observation points to the fact that sometimes Kmeans does not always cluster the data correctly which is why multiple experiments are often needed to verify the results.

Figure IV.3: HYG Dataset vs. k vs. Weighted Affinity for Naive Algorithm

Figure IV.4 shows the trend between dataset size and execution time. The clus-
tering time has been normalized to the 'Basic' dataset run and the plot is showing in log scale. In general, the trend shows an increase in execution time as the data size gets larger. The data size is a multiple of the number of observations and the dimensionality of the dataset.

Figure IV.4: Total Clustering Time per Dataset

Figure IV.5 shows a breakdown of the clustering time with respect to k as well as the sampling size. The base run is the case where no refinement was used on the Kmeans run, and the rest of the lines represent clustering time for the different experimental runs with different sampling sizes of percentages of the original dataset. The plot shows that there is an improvement in clustering time when using a sample size of 10 and sampling 1% of the original dataset.

Figure IV.6 shows the sum of execution time compared to the sum of weighted affinity for each dataset on an example run with 8 clusters (k=8).
Figure IV.5: Clustering Time Compared to Base for Kmeans on Kddcup Algorithm

Figure IV.6: Total Weighted Affinity and Clustering Time over Dataset for k=8
Figure IV.7 shows how the affinity of each cluster is generally inversely proportional to k. This trend occurs because as the data is clustered into more separate clusters, typically the average within-cluster distance decreases as well. If all data points became clusters, there would be zero average within-cluster distance and the affinity would be zero - which does not help to find any hidden structures in the data.

![Weighted affinity vs k (across all experiments), Diabetic](image)

Figure IV.7: Affinity Trends Down as k Increases on Diabetic Dataset

Figure IV.8 shows how the clustering time is related to the algorithm and the number of clusters (k). This demonstrates how different algorithms perform across the spectrum of k values for the “Diabetic” dataset. As also previously shown, the affinity tends to trend down as the value of k increases. Because there were no refinement techniques used to plot this chart, the affinity scores did not see much deviation between the algorithms.

In this next chart, Figure IV.9 shows how refinement can actually reduce the affinity over some of the naive runs. The plot shows that the pelleg-moore algorithm,
Figure IV.8: Naive Algorithm Takes Longer for Higher Values of k

Figure IV.9: Refinement Techniques Can Sometimes Yield Better Affinity
with no refinement provided the lowest weighted affinity score, as well as a significantly lower execution time than the naive algorithm with no refinement (denoted on the plot as sample percent = 0 and sample size = 0). There is, however, a refinement technique that beats the naive in both weighted affinity and execution time. The dualtree algorithm with 100 samples and a 1% sampling rate scores just above the lowest affinity score. As would be expected, the elkan algorithm, using 100 samples at a 10% sampling rate, while it also beat the naive, no-refinement experimental run, the execution time is the highest on the chart. These results show that by characterizing the algorithm and experimental setup, the overall clustering time and outcome can be significantly improved by selecting the optimal refinement techniques, algorithms, and parameters.

Figure IV.10 shows PCA analysis and how it could potentially improve upon the execution time if the primary components that contribute to a majority of the variance are filtered out and clustered. This plot shows that some of the components can probably be eliminated while still maintaining a statistically reasonable result after clustering. More work is needed to explore this in-depth as there is a potential for using PCA analysis before running the Kmeans algorithm, which could improve the process even more. As can be seen, roughly 3 of the variables account for 70% of the problem variance. With that in mind, refinement techniques may be able to use this information to better implement the experimental runs.
Figure IV.10: PCA on the Power Dataset Showing Problem Variance Contribution

28.6% of parameters contribute to 70% of the problem variance
CHAPTER V
CONCLUSION

The outcome of running several different experiments on approximately a dozen datasets provided insight into how the Kmeans algorithm behaves on datasets of diverse sizes. With the limited computing power available, it took over 8 hours to run all the experiments on the 13 datasets. The extreme execution time is one reason why this research is important: not everyone has access to large computing resources, so running experiments like this on larger datasets can take several weeks, or even months.

The examination of the refinement Kmeans technique demonstrated good potential in overcoming execution time of the traditional Kmeans. Further exploration is warranted to help build on this research. Anything that can be done to help speed up the execution time while also improving the clustering process is valuable to the scientific and data research communities, not only in terms of energy savings, but also time savings and getting results faster to help solve the problem at hand. Whether it is mining data to predict elections, or evaluating cancer rates from patient studies, the world of machine learning and clustering will no doubt continue to be a significant part of the next several decades of technological advancement.
REFERENCES


