Accelerating Large Scale Centroid-based Clustering with Locality Sensitive Hashing

Ryan McConville, Xin Cao, Weiru Liu, Paul Miller
School of Electrical, Electronic Engineering and
Computer Science
Queen's University Belfast
{rmcconville07, x.cao, w.liu, p.miller}@qub.ac.uk

***Abstract*—Most traditional data mining algorithms struggle to cope with the sheer scale of data efficiently. In this paper, we propose a general framework to accelerate existing clustering algorithms to cluster large-scale datasets which contain large numbers of attributes, items, and clusters. Our framework makes use of locality sensitive hashing (LSH) to significantly reduce the cluster search space. We also theoretically prove that our framework has a guaranteed error bound in terms of the clustering quality. This framework can be applied to a set of centroid-based clustering algorithms that assign an object to the most similar cluster, and we adopt the popular K-Modes categorical clustering algorithm to present how the framework can be applied. We validated our framework with five synthetic datasets and a real world *Yahoo! Answers* dataset. The experimental results demonstrate that our framework is able to speed up the existing clustering algorithm between factors of 2 and 6, while maintaining comparable cluster purity.**

I. Introduction.

Data clustering [1] is a widely used data mining technique for the unsupervised grouping of data points, items or patterns. The goal is to automatically discover these groupings from unlabelled data. The problem can usually be formulated as given n items, discover k groups using suitable similarity measures which maximise the degree of similarity of items in the same group, while minimising the degree of similarity of items in different groups.

Typically, in centroid-based clustering algorithms (e.g., K-Means [2] and K-Modes [3]), clusters are represented by a central vector, and the clustering task is usually defined as an optimisation problem: find k clusters and assign the items to the closest or most similar cluster such that a certain measure (e.g., the squared distances in K-Means) is minimised. Therefore, similarity (or distance) comparisons can be a major performance bottleneck when facing large scale data clustering. When k is very large, such clustering algorithms do not scale well and have poor performance in terms of efficiency.

Motivated by this, the key challenge to be addressed in this paper is to propose a novel clustering framework that can scale well with a massive number of clusters, where items may also be high dimensional. That is, clustering a dataset containing a large collection of high dimensional data into a large number of clusters represented by centroids. In such cases efficiently measuring the similarity of each item to each cluster centroid is critical in accelerating the clustering task.

Recently in both the research and industry communities increased emphasis has been placed on algorithms to mine the abundance of data being generated, so called *Big Data* analysis. One line of research to achieve this is to adapt existing, well-understood algorithms to handle larger scale data (e.g., [4]). We follow the same line of research in this paper. Our approach to solve this problem and improve efficiency is to make use of locality sensitive hashing (LSH) [5]. By utilising the LSH technique, we are able to find, for a given item to be clustered by a centroid-based clustering algorithm, all the other items that have a certain similarity above a predefined threshold. The objective is to build a *hash based index* of all similar items in the dataset to be clustered, and to utilise this index to obtain a *shortlist of candidate clusters* for the centroid-based clustering algorithm to operate on for this item. This method can eliminate dissimilar clusters before applying existing clustering algorithms, and it significantly accelerates the clustering process while maintaining clustering quality, which is the key novelty of our work.

In order to present how our framework can be applied to existing clustering approaches, we focus on the K-Modes algorithm in this paper. K-Modes [3] is a clustering algorithm for categorical data. We use the Jaccard similarity [6] to compute the similarity between two categorical items, and thus we adopt the min-wise independent permutations locality sensitive hashing scheme (MinHash) [7], which forms an LSH for Jaccard Similarity. We denote the algorithm accelerated with MinHash as MH-K-Modes. K-Modes is similar to the K-Means algorithm [2] but replaces the numeric distance calculations with a fast and simple dissimilarity measure for categorical data. Categorical data, sometimes also called nominal data, are variables that typically have two or more categories. Importantly, there is no intrinsic ordering to these category values. A simple example of this would be the colours of items, where a set can have a few or many different colours, and there is no standard ordering over a set of colours.

The K-Modes similarity measure is fast due in part to its simplicity, however we will later show that whenever the number of clusters in the data increase, and particularly when the number of attributes in the data also increases (i.e. it becomes higher dimensional), K-Modes can become extremely slow.

To evaluate our approach, we conduct experiments on five synthetic datasets, as well as a real world Yahoo! Answers dataset [8]. We investigate, compare, and analyse our MH-K-Modes algorithm and the original K-Modes algorithm in respect to a number of properties. We will analyse the time taken per iteration, as well as the number of iterations required for the algorithms to converge. We will also analyse the average size of the shortlist of candidate clusters for our algorithm, and how this varies with regard to the chosen algorithm input parameters.

Next we will analyse the number of moves, or cluster re-

assignments, per iteration of both our MH-K-Modes algorithm and the K-Modes algorithm. Finally we will compare the total time taken to cluster each dataset using various parameter settings for our MH-K-Modes algorithm and the K-Modes algorithm. Our algorithm includes an initial extra step before the cluster algorithm begins, which will be captured by this analysis.

The contributions of this paper are the following:

- We present a novel framework for improving the efficiency of existing centroid-based clustering algorithms using Locally Sensitive Hashing (LSH) to greatly reduce the cluster search space.
- We apply our framework to the well-known K-Modes algorithm and utilise the MinHash LSH algorithm.
- We prove that our method has a guaranteed error bound.
- Through experimentation we evaluate our framework on five synthetic datasets as well as a real world Yahoo! Answers dataset. We show that our framework achieves similar performance in terms of cluster purity but most significantly is more efficient. With the parameters tested we observe that our framework is more efficient by factors between 2x and 6x.

II. Related Work

The data clustering problem continues to have been studied intensively in recent years, and it has been used in many applications such as image segmentation [9]–[11], genetic mapping [12], community detection [13], etc. The centroid-based clustering algorithms such as K-Means [2] and K-Modes [3] have been widely applied due to their simplicity and easy implementation.

As has been pointed by several existing studies, centroid-based clustering is inefficient. For example, [14] outlines the efficiency problem with large numbers of data points and large numbers of clusters for the K-Means algorithm. In their approach the authors optimised the assignment of points to cluster centres using multiple random spatial partition trees such that only a small number of clusters need to be considered during the assignment step of K-Means. More specifically, this approach creates a neighbourhood for each point via a multiple random partition trees method and uses this neighbourhood to find the set of clusters within it. These clusters form the candidate set which are typically smaller than the full cluster set. Their approximate K-Means algorithm converged 2.5 times faster, and achieved better performance than the state-of-the-art approaches they compared.

[15] proposes the idea of canopies which represent divisions of overlapping subsets of the data. These canopies can be computed quickly and is followed by a second stage that preforms exact distance measurements among points in common canopies. The concept of canopies is used to improve the efficiency of clustering. This work supports high dimensional data, with large numbers of clusters and data points.

[4] shows how the K-Means algorithm can be scaled up to large vocabulary sizes in the computer vision domain. This is achieved through the use of a random forest approximate nearest neighbour algorithm. Similar in spirit to us, the costly exact distance comparison between points and clusters is replaced by an approximate measure using randomized k-d trees.

[16] presents an updated K-Means that addresses three core issues that have been identified for web scale clustering; latency, scalability and sparsity. They manage to achieve a decrease in computational cost by orders of magnitude with the use of mini-batch clustering for K-Means.

None of the above studies consider utilising locality sensitive hashing (LSH) to accelerate the clustering. The min-wise independent permutations locality sensitive hashing scheme (MinHash) is a technique for quickly approximating the Jaccard Similarity [6] between sets. It has been applied in numerous domains such as duplicate web page detection for a search engine [5], online news personalisation [17] and computer vision tasks [4].

There have been several data clustering algorithms proposed that use LSH in some manner. For example, the work [18] proposes to utilise LSH in clustering web pages. In this study, web pages are hashed in such a way that similar pages have a much higher probability of collision than dissimilar pages based on LSH. A graph is created for the web pages where each node represent a web page and each edge indicates that the two pages are similar according to LSH. Next, a graph partitioning algorithm is applied to divide the web pages into different clusters. This approach is different from our work in that we are interested in centroid based clustering algorithms, rather than graph partitioning algorithms. Also, building the graph is infeasible for very large datasets.

One approach [19] uses a centroid-based clustering algorithm K-Medoids with LSH but with the idea of developing a locality sensitive hashing method for generic metric spaces. The intention in this work is to improve the LSH algorithm, rather than the clustering algorithm.

Another example [20] evaluates the use of various LSH functions, specifically searching for high dimension SIFT descriptors. Their approach was inspired by the challenge of high dimensional nearest neighbour retrieval, which is a very expensive process. The authors proposed a technique called KLSH that makes use of the K-Means clustering algorithm. Although the idea here is similar to ours, there are a number of differences. First, our idea is to consider large numbers of clusters rather than just a large number of dimensions. Second, [20] uses K-Means clustering as a means of speeding up nearest neighbour search of large vectors via LSH, whilst our framework uses LSH as a means of speeding up the clustering algorithm.

In summary, to the best of our knowledge, this is the first work on increasing the efficiency of centroid-based clustering by using MinHash to reduce the cluster search space.

III. Large Scale Clustering

A. Preliminaries of K-Modes and MinHash

1) K-Modes: In order to describe our framework, we first introduce the K-Modes algorithm [21].

K-Modes differs from K-Means in three major ways; K-Modes uses a simple dissimilarity measure between items rather than the least squares method; cluster centroids are represented by modes rather than means; and it uses frequency based updating of modes to minimise the cost function.

The K-Modes algorithm can be summarised as:

- Select k initial modes from the dataset. Numerous methods exist for making this selection. A simple selection method would be to choose k random items.
- For each item assign it to the closest cluster based on the dissimilarity measure.
- Recalculate the modes of all clusters.
- Repeat the previous two steps until either no item has changed cluster, or the cost has minimised (Equation 4).

From these steps we can expect that when there are many items to cluster into (very large) k clusters, with each of the items having many attributes, step 2 could become an expensive and time consuming operation. This is due to each of the many large items requiring comparison to each of the many cluster centroids, and the comparisons multiply over time as algorithm continues to do this over many iterations.

Let X and Y be two categorical items described by m categorical attributes. The dissimilarity measure $d(X, Y)$ [3] computes the total mismatches of the corresponding attribute categories between X and Y. The fewer the mismatches, the more similar the two items are.

$$d\left(X, Y\right) = \sum_{j=1}^{m} \delta\left(x_j, y_j\right) \tag{1}$$

where

$$\delta\left(x_j, y_j\right) = \begin{cases} 0, & \text{if } (x_j = y_j), \\ 1, & \text{otherwise.} \end{cases} \tag{2}$$

Formally we can define the K-Modes algorithm as:

Let $\mathcal{X}$ be a set of categorical items each with m categorical attributes $A_1, \cdots, A_m$, where each A_j is a single attribute, e.g., 'Colour'. The domain of A_j, denoted by $\mathsf{DOM}(A_j)$, contains a set of category values, i.e., $\mathsf{DOM}(A_j) = \{a_1, a_2, \cdots, a_l\}$. For example, given an item $X_i = \{x_1, x_j, \cdots, x_m\} \in \mathcal{X}$, x_j may have the category value 'blue' for attribute colour A_j. A mode of $\mathcal{X} = \{X_1, \cdots, X_n\}$ is a vector $Q = [q_1, \cdots, q_m]$ that minimises:

$$D(\mathcal{X}, Q) = \sum_{i=1}^{n} d\left(X_i, Q\right) \tag{3}$$

Let $n_{c_{k,j}}$ be the number of items having the k_{th} category value $c_{k,j}$ for attribute A_j and $f_r\left(A_j = c_{k,j}|\mathcal{X}\right) = \frac{n_{c_{k,j}}}{n}$ the relative frequency of category $c_{k,j}$ in $\mathcal{X}$. The function $D\left(\mathcal{X}, Q\right)$ is minimised iff $f_r\left(A_j = q_j|\mathcal{X}\right) \geq f_r\left(A_j = c_{k,j}\right)$ for $q_j \neq c_{k,j}$ and all $j = 1, \ldots, m$.

Therefore, the optimisation problem for partitioning a set of n items each with m categorical attributes into k clusters is to minimise the following cost function:

$$P(\mathcal{W}, \mathcal{Q}) = \sum_{l=1}^{k} \sum_{i=1}^{n} w_{i,l} D(X_i, Q_l) \tag{4}$$

where $\mathcal{W}$ is the $n \times k$ membership matrix, $w_{i,l} \in W$, and $\mathcal{Q} = \{Q_1, ..., Q_l, ..., Q_k\}$ is a set of cluster centers.

2) MinHash / Locality Sensitive Hashing: We will use duplicate document detection, which is a common use case for MinHash, as an example to describe how it may be applied to our problem. Given a set of documents, we would first want to convert each set of words in the document into 'signatures' becoming a more compact representation of each document. These signatures are computed for each document by 'minhashing' the document a number of times. Given a word-document matrix where each column represents a document and each row indicates the presence (0/1) of a word in the document; 'Minhashing' would be choosing for each column, from a permutation of the rows, the row number of the first row which has a value 1 in that column, a process that is typically repeated a number of times. If we do this n times, each with a different permutation, the size of the signature would be n. To make this practical, the random permutations of the matrix can be simulated by the use of n randomly chosen hash functions. Given a row r in the word-document matrix, we use hash function $h()$ to simulate the permutation of r to the position $h(r)$. For example, let r be the third row, and let a hash function be $h(x) = 2x + 1 \mod 5$, then $h(r) = 2 * 3 + 1 \mod 5 = 2$. r is permuted to the second row according to this hash function. Similarly, this hash function is applied to all rows with value 1, and the smallest hash value (also the highest ranked row number) is chosen as the outcome of this particular hash function. This value is denoted as s_i (where i represents it is the i_{th} hash function used). Formally, the signature equation can be represented as

$$\mathsf{SIG_D} = (s_1, s_i, \ldots, s_n) \tag{5}$$

where $s_i = min(h_i(r_j)|j = 1, t)$, and then for $i = 1, \ldots, n$, such that t is the number of total rows with value 1, and n is the number of hash functions. The detailed description of the idea can be seen in Algorithm 1.

These signatures, although likely smaller than the original document, are only part of the solution for quickly estimating the similarity between documents. The next step is to further subdivide the signature produced above into rows(r) and bands(b). Each band will consist of r hash-values, which are input to another hash function that maps the band to a new bucket. Importantly there will be b sets of buckets to map to, one set for each band so no overlapping between bands can occur. Thus we can now say that if a band from each of the two documents map to the same bucket, they are candidate pairs.

MinHash is known to approximate the Jaccard Similarity between sets, and the choice of r and b have significance in that they determine the probability that two documents with a Jaccard Similarity (Equation 6) of s become candidate pairs.

$$\text{Jaccard-Sim}(\mathsf{X},\mathsf{Y}) = \frac{|\mathsf{X} \cap \mathsf{Y}|}{|\mathsf{X} \cup \mathsf{Y}|} \tag{6}$$

Specifically, the probability that the signatures are identical in at least one band, therefore making them a candidate pair, is $1-(1-s^r)^b$. We can exploit this in order to choose appropriate values for r and b prior to the clustering. The similarity s at which there is a 50% chance of two items becoming candidate pairs, at which the rise of the s curve is the steepest, is a function of b and r: $(1/b)^{1/r}$.

Algorithm 1: *SIGGEN*:MINHASH SIGNATURE GENERATION [7]

Input: *item*: a vector of categorical values from a single item
Input: H: hash functions $h_1, \ldots, h_m \in H$
Output: *signature*: a vector of length m

```
1 forall i in item do
2     h_min(i) = ∞
3 forall i in item do
4     forall j in H do
5         if h_j(i) < h_min(j) then
6             h_min(j) = h_j(i)
```

B. Our Algorithm MH-K-Modes

The integration of MinHash and LSH with K-Modes is detailed in Algorithm 2. Specifically, after the centroid initialisation step of K-Modes, we can make a single pass over the entire dataset, applying MinHash to each item. When we insert each item into a bucket as per MinHash procedure, we will also store a reference to the cluster that the item has been assigned to by K-Modes. Once this single pass over the dataset has been completed, we will have effectively produced an index data-structure of items to other similar items that we can query from the K-Modes algorithm. Therefore during each iteration of K-Modes, each time we encounter an item to assign to a cluster, we will query our MinHash index with the item to find the set of other similar items. Since each item contains a reference to the cluster it is currently assigned to, we can create a shortlist of candidate clusters for K-Modes. This is possible as when we MinHash the query item we will discover all the previously MinHashed items that the LSH function regarded as similar. From these items we retrieve their referenced clusters to build the shortlist. Our framework relies on the assumption (for which we will have a theoretical investigation in III-C) that we can achieve similar performance in terms of accuracy whenever the true positive (TP) cluster is in the shortlist, along with a small number of other false positive (FP) clusters. Once we have updated the clustering by assigning an item to a cluster we must update the MinHash index to reflect this. This is a fast operation as we merely update the items cluster that is stored via a reference or pointer.

Worth noting is, as seen in Lines 2-4 of Algorithm 2, we filter out any feature values that indicate that the feature is not present before the signature generation in Algorithm 1. This is useful in scenarios where each feature vector item may be large and may contain many values that are not present.

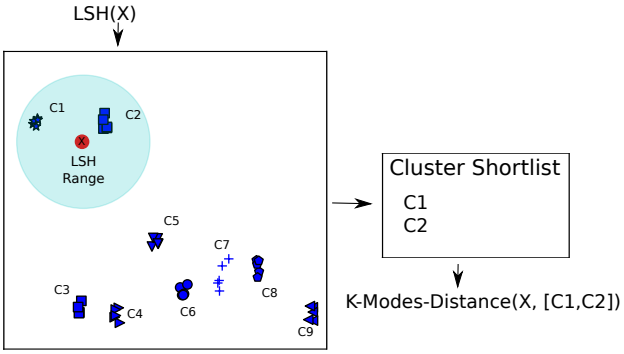


Fig. 1: Illustrating the idea of how LSH can be used to find the relevant clusters for the red coloured point X.

Such cases may include when you would like to represent a vocabulary with word presence indicated by Yes or No values. By excluding the 'No' values, MinHash is able to produce more useful similarity scores between two feature vectors, as many shared negative features in two vectors does not provide particularly useful information about the similarity of two sets. Usually we are only interested in the similarity of values present in the feature vectors.

We now present the steps of our modified MH-K-Modes algorithm.

- Select k initial modes from the dataset. Numerous methods exist for making this selection. A simple selection method would be to choose k random items from the dataset.
- For each item assign it to the closest cluster based on the dissimilarity measure.
- MinHash each item, effectively creating an index of similar items. In the index store a cluster reference for each item.
- For each item query the previously generated MinHash index for similar items and their clusters. Use these clusters rather than the full set to assign it to the closest cluster based on the dissimilarity measure. After each change, update the cluster reference in the MinHash index to the new cluster.
- Recalculate the modes of all clusters.
- Repeat the previous two steps until either no item has changed cluster, or the cost has minimised.

Compared to the original K-Modes algorithm described in Section III-A1, we now have a number of extra operations. One completely new step is the initial MinHash operation that is run only once at the beginning of the clustering process. We also have significantly modified the step in which the similarity of each item to existing clusters is calculated. It makes use of the MinHash index to pre-filter the potential cluster candidates during each iteration of the K-Modes algorithm.

C. Error Bound

Our proposed framework relies on the accurate indexing of clusters and matching of clusters to data items being clustered.

Algorithm 2: MINHASH PRE-FILTERING STEP

Input: $item$: item to be clustered
Input: H: set of hash functions
Output: $shortlist$ Shortlist of clusters to fully search

1 $filtered\_presence\_item$ = []
2 **foreach** *feature in item as* $feature$ **do**
3 **if** $feature$ *is present* **then**
4 add $feature$ to $filtered\_presence\_item$
5 $signature$ = $SIGGEN(filtered\_presence\_item, H)$
6 divide $signature$ into b bands each consisting of r hash values.
7 $buckets$ = set()
8 **foreach** b *in band* **do**
9 add $idx\_hash(b)$ to $buckets$
10 $shortlist$ = set()
11 **foreach** $bucket$ *in* $buckets$ **do**
12 add the clusters in $bucket$ to $shortlist$

Recall that in our method, when X is to be clustered, we first retrieve items similar to X from the index, and then create a shortlist of clusters from these items. We then compute the similarity between X and each cluster in the shortlist. Therefore, if the actual cluster that X shall be clustered to contains one item which is similar to X in the index, X will be correctly clustered (the correctness here means that the clustering result is the same as the original algorithm without using the index). Therefore, *the error of our method can only occur when the candidate clusters selected for an item do not in fact include the actual cluster that this item shall be clustered to*. In this section, we investigate theoretically properties of this error in the clustering framework, and we show that our method utilising MinHash has a guaranteed error bound.

In order to analyse the error caused by the MinHash index, we take clustering a categorical item X with m attributes as an example. We denote C_n as the actual cluster X should be clustered to (whose mode has the smallest dissimilarity with X) according to the original K-Modes. We can compute the probability that C_n is not in the candidate clusters list as follows.

If C_n is the best cluster, then in C_n there must exist an item such that it has the same values on at least one attribute as that of X (otherwise, the dissimilarity of the mode of C_n and X will be m and C_n is not the best cluster). Now let us denote this item in C_n as Y, the Jaccard-similarity of X and Y is at least $s = \frac{|X \cap Y|}{|X \cup Y|} \geq \frac{1}{2m-1}$. Hence, the probability that the signatures of X and Y agree in all rows of one particular band is at least s^r, and the probability that the signatures disagree in at least one row of each of the bands is at most $(1-s^r)^b$. This means that the probability of X and Y not being a candidate pair is at most $Pr = (1-s^r)^b = (1-(\frac{1}{2m-1})^r)^b$.

When clustering item X, if C_n is the cluster whose mode has the smallest dissimilarity with X but C_n is not obtained from the index, it means that no item Y' exists in C_n that can form a candidate pair with X according to the index. Given the similarity between X and Y is at least $s = \frac{1}{2m-1}$, we know that the probability of this error is at most $Pr^{|C_n|} = (1-(\frac{1}{2m-1})^r)^{b|C_n|}$, where b is the number of bands and r is the number of rows in MinHash.

Therefore the error bound can be very small even when the number of attributes are very large in the dataset. For example, in our experiments, a common number of attributes of an item in the datasets we used is 100. If we set $r = 1$ and $b = 25$ in MinHash, and we assume that a cluster has at least 20 items, the probability of the error when clustering an item in our framework is at most $Pr^{|C_n|} = (1-\frac{1}{199})^{25\times 20} = 0.08$. This explains why our framework improves clustering efficiency significantly while maintaining excellent clustering quality.

D. Choice of LSH Parameters

In Table I we can see the relationship between the band parameter, the Jaccard-similarity of two items, the probability of candidate pairs using MinHash, and the probability of MH-K-Modes finding the candidate cluster. With the number of rows set to 1, the greater the number of bands, the lower the Jaccard-similarity needs to be for the algorithm to find two items similar. In Table II with the number of rows set to 5, the same trend of more bands increases the probability of candidate pairs being found. However, it is clear that with the increased number of rows the probability of finding two items with an Jaccard-similarity as similar has decreased compared to Table I which had only 1 row.

One possible choice is to use MinHash with just one row and one band, which would have the effect of eliminating clusters that are extremely unlikely to have any similarity. Whilst with 100 bands it is possible to with 99% probability to find sets with a Jaccard-similarity of just 0.1 to become candidate pairs. With 800 bands, sets with a Jaccard-similarity an order of magnitude smaller have the same probability. The downside of this parameter setting is that it is likely to include many false positive clusters in the shortlist. This is overcome by increasing the number of rows, at the cost of introducing more false negatives in shortlist.

In Section III-A2 we discussed the probability of collisions occurring for given Jaccard-similarities, bands and rows. To reiterate, we can calculate the probability that two items become candidate pairs for any given combination of the Jaccard similarity, band and row parameters. However, since we only consider the cluster of each candidate pair when forming the shortlist (see Lines 10-12 of Algorithm 2) we do not need to find all item candidate pairs in order to achieve our goal of finding cluster candidate pairs. Instead, we only need to find just one item candidate pair from each candidate cluster. If there is a 10% probability of two items with a Jaccard-similarity of 0.1 becoming candidate pairs, and if there are 50 such candidate pair items in the cluster, then we have a high probability (99%)[1] that at least one of them will collide and thus provide us with the candidate cluster. Importantly, our application of MinHash means we can achieve much better performance without increasing the number of hash functions and thereby increasing computational overhead. This should provide an intuition as to how the choice of r and b will affect our efficiency and performance, and how the standard MinHash

[1] $1-(1-(0.1))^{50}$

selection criteria in terms of r and b need not be so strict with our framework.

Bands	Jaccard-similarity	Probability	MH-K-Modes Probability
10	0.01	0.09	0.61
10	0.1	0.65	1
10	0.2	0.89	1
10	0.5	0.99	1
100	0.001	0.009	0.09
100	0.01	0.3	0.97
100	0.1	0.99	1
100	0.5	1	1
100	0.8	1	1
800	0.0001	0.07	0.52
800	0.001	0.55	0.99
800	0.01	0.99	1
800	0.1	1	1

TABLE I: Probability of finding a candidate pair with a given Jaccard-similarity and given number of bands with a **row value of 1**. MH-K-Modes probability calculated assuming a minimum of 10 other items in the cluster with at least the given Jaccard-similarity.

Bands	Jaccard-similarity	Probability	MH-K-Modes Probability
10	0.1	0.0001	0.001
10	0.2	0.003	0.03
10	0.5	0.27	0.96
10	0.8	0.98	1
100	0.1	0.001	0.01
100	0.5	0.95	1
800	0.1	0.008	0.08
800	0.2	0.23	0.93
800	0.3	0.86	1

TABLE II: Probability of finding a candidate pair with a given Jaccard-similarity and given number of bands with a **row value of 5**. MH-K-Modes probability calculated assuming a minimum of 10 other items in the cluster with at least the given Jaccard-similarity.

IV. Experimentation

We will now provide experimental evidence of the effectiveness of our framework MH-K-Modes, in comparison to regular K-Modes, on a number of datasets. To make it easier to clearly analyse the properties of our framework, we will first provide our analysis on a number of synthetic datasets. Following this we will provide results on a dataset consisting of questions and topics from Yahoo! Answers [8] in order to demonstrate real-world effectiveness. For each dataset we hope to show that our algorithm will improve the efficiency of K-Modes algorithm, while maintaining comparable performance.

Our experimentation was carried out on machines with an Intel X5650 CPU and 96GB of RAM. Our implementation was single threaded and thus only used one of the available twelve cores on the machine. The programming language used for both the K-Modes algorithm and the MH-K-Modes algorithm is Python and made heavy use of the numpy package[2]for numerical computations.

[2]http://www.numpy.org/

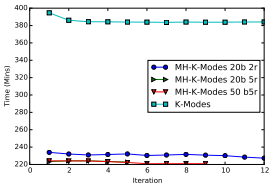
A. Results on Synthetic Data

Our synthetic datasets were generated with the datgen tool[3]. For all experiments we used a domain size of 40000 categorical values which can be used by each attribute when generating the dataset. Each item will be associated with one of the k clusters. This association is decided in the form of conjunctive rules formed from the attributes. For example, cluster 1 could require attribute A_1 having the categorical value A and attribute A_4 having the categorical value B, etc. Therefore when creating an item that belongs to cluster 1, attributes A_1 and A_4 would have the above values. The remaining attributes may be any other values. For our base experiments consisting of 100 attributes each item used a conjunctive rule involving between 40 and 80 attributes when creating data items for clusters. The remaining 20 to 60 attributes were not relevant to the cluster assignment. In experiments where the number of attributes were increased, these values were scaled in proportion to the number of attributes in the larger items.

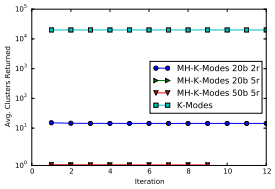
K-Modes has a number of potential initialisation methods for choosing the initial cluster centroids [3] [22]. As the objective of our work is to improve the clustering efficiency by optimising the item assignment process during each iteration, we will randomly select the k initial centroids. We note that for each experiment where we are evaluating various parameters of our MH-K-Modes algorithm and K-Modes algorithm, the same initial centroid points were selected. This prevents the initial centroid selection from influencing the performance and efficiency results.

1) Varying Number of Clusters: One of the most important parameters in validating how our algorithm scales is by exploring how it performs with datasets consisting of large numbers of clusters. As our algorithm is designed such that it reduces the cluster search space using Locality Sensitive Hashing, we expect to find a significantly smaller number of clusters on the shortlist for K-Modes to use. Indeed this is clearly seen in Figure 2b where consistently less clusters are included in the shortlist with all parameters tested on the MH-K-Modes algorithm. We expect a correlation between the number of clusters on the shortlist, and the time taken for each iteration of K-Modes. This expectation is confirmed in Figure 2a; all of our tested parameters resulted in less time spent per iteration. The parameters 20 bands and 5 rows appears to be the optimal tested whilst 50 bands and 5 rows offers almost no improvement in the average cluster shortlist size, despite the increased number of hash functions and therefore total time required to cluster (Figure 7a). This can be seen in Figure 2e. In Figure 2c we can see the number of times that an item was moved from one cluster to another during the K-Modes assignment steps. The trend here continues, the optimal value of 20 bands and 5 rows results in the least number of moves. Finally, the experiments also reveal in Figure 7a that not only does our MH-K-Modes algorithm always result in less time per iteration, in all tested cases it converged faster, resulting in less overall iterations. The best result was MH-K-Modes with 20 bands and 5 rows taking under 225 minutes per iteration, and converging after 5 iterations. This is in comparison to the original K-Modes which took around 380 minutes per iteration, and converged after 12 iterations.

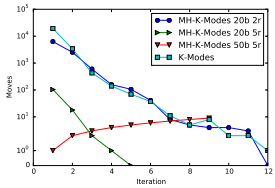
[3]http://www.datasetgenerator.com/source/



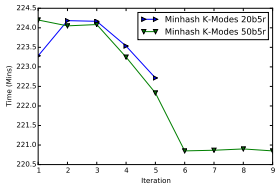
(a) Time taken per iteration



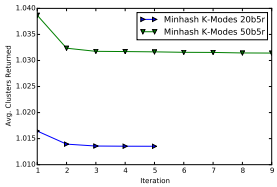
(b) Average shortlist size



(c) Average moves



(d) Closer look at Figure 2a



(e) Closer look at Figure 2b

Fig. 2: 90000 items with 100 attributes and 20000 clusters.

We have achieved promising results so far since the efficiency of our MH-K-Modes algorithm is significantly better than that of the K-Modes algorithm with 20 thousand clusters. Our next experiment will run with the same parameter settings, but with twice as many clusters. In Figure 3a we can see a similar trend to the previous experiment. All our tested parameters resulted in a less time per iteration. In this case, the time per iteration difference between our MH-K-Modes algorithm and the original K-Modes algorithm is more significant. We can see that previously we reduced the time taken per iteration from around 380 minutes to around 220 minutes. In Figure 3a the time is reduced from around 780 minutes (after 4 iterations) to just over 300 minutes. This is an improvement of around 480 minutes per iteration with 40 thousand clusters, compared to 160 minutes per iteration with 20 thousand clusters. Figure 3b provides a more clear picture of the time taken per iteration, by excluding the original K-Modes algorithm. One interesting observation is that the parameter combination of 20 bands and 5 rows appears to be an outlier when compared to the others as it takes around 380 minutes per iteration, as opposed to 315 minutes. Nonetheless it manages to converge in the smallest number of iterations, just 4. This shows that while time per iteration is important, so is the number of iterations before converging. Both Figure 2 and Figure 3 illustrates that our algorithm with large numbers of clusters is more efficient in both time per iteration and number of iterations before converging than the original K-Modes.

2) Varying Number of Items: The next aspect of our MH-K-Modes algorithm we would like to evaluate is how much more efficient our algorithm is when it comes to increasing the number of items in the dataset. As our algorithm is designed to reduce the cluster search space the more items there are to be clustered, it is reasonable to assume that the more time that will be saved overall as each item will make a time saving.

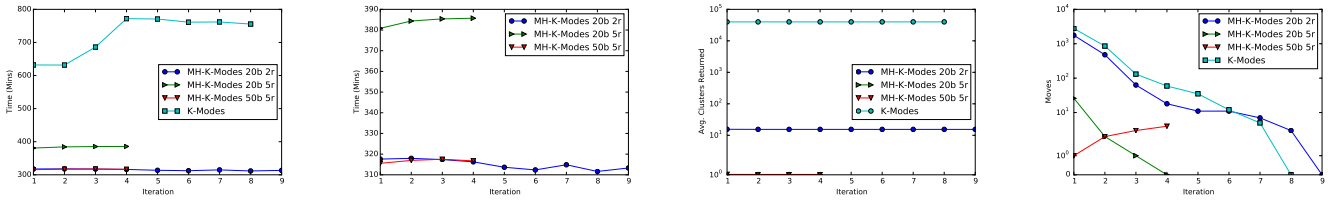
In order to investigate the effect of increasing the number of items on the efficiency of our algorithm, we generated a synthetic dataset consisting of 250 thousand items. We maintained 20 thousand clusters and 100 attributes as before. Figure 4 shows the results of this experiment. In Figure 4c it is clear that as we increased the number of items to be clustered, the total time taken per iteration increased. However, we can still see that our MH-K-Modes algorithm resulted in less iterations and less time per iteration. With 20 bands and 5 rows, our algorithm converged after 8 iterations, compared to the original K-Modes with 10 iterations. Our algorithm also took much less time per iteration, around 800 minutes until the sixth iteration, followed by around 1100 minutes per iteration for the final two iterations. On the other hand the original K-Modes algorithm took almost 1600 minutes per iterations consistently for 10 iterations. Accounting for the jump in time per iteration for our algorithm after 6 iterations, we still see an improvement of between 800 minutes (or 50%) and 500 minutes for each iteration. Figure 4a and Figure 4b exhibit the same trend as witnessed with 90 thousand items. In Figure 6a we plot the rate of growth in time taken for the K-Modes and MH-K-Modes algorithms to cluster both 90000 and 250000 items. It is clear from this that our algorithm scales more efficiently with regard to dataset size than the original K-Modes algorithm.

3) Varying Number of Attributes: We will now investigate the performance and efficiency of our algorithm when the number of attributes in the dataset is increased. Higher dimensional data is typically associated with increased complexity and running time as each item and centroid is much larger. Specifically we will report and analyse the results with 90000 items, 20000 clusters and 100, 200 and 400 attributes respectively. We expect that our framework will see improvements in efficiency as the number of attributes increase because each comparison will require more computation within the dissimilarity function (Equation 1 and Equation 2).

In Figure 5a our results reveal that with the increased number of attributes, doubled from 100 to 200, we maintain significant efficiency gains. Our best parameter selection for MH-K-Modes converged 101 hours faster than K-Modes with 100 attributes, and 104 hours faster with 200 attributes. With 400 attributes, Figure 6c displays an even greater increase in efficiency when comparing our algorithm with the original K-Modes algorithm. We will discuss the scaling aspect in more detail in Section IV-A4. Figure 5b also reinforces that our algorithm can significantly reduce the cluster search space, as the candidate cluster shortlist is consistently much smaller than the full search space by many orders of magnitude.

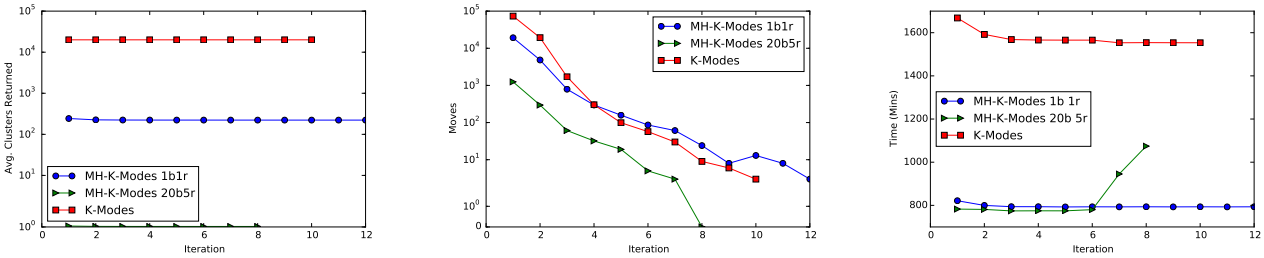
4) Synthetic Data Scaling Comparison: In previous sections we have analysed a number of properties of the two algorithms when varying various parameters of the datasets and algorithms. For the sake of clarity, we will also show and discuss how each of the algorithms scale in terms of efficiency with respect to the number of clusters, attributes and items.

In Figure 6a we show how, while both our MH-K-Modes



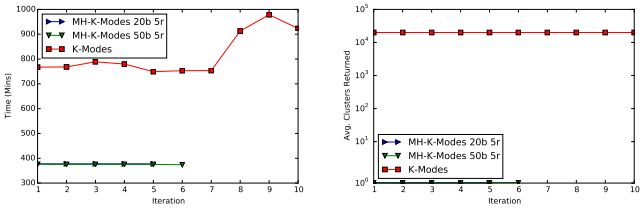
(a) Time taken per iteration (b) Excluding K-Modes results (c) Average shortlist size (d) Average number of moves

Fig. 3: 90000 items with 100 attributes and 40000 clusters.



(a) Average shortlist size (b) Average moves (c) Time taken per iteration

Fig. 4: 250000 items with with 100 attributes and 20000 clusters.



(a) Time taken per iteration (b) Average shortlist size

Fig. 5: 90000 items with 200 attributes and 20000 clusters.

algorithm and the original K-Modes algorithm grow almost linearly with the number of items, our algorithm has a slower rate of growth than that of K-Modes. This aligns with our expectations as each item will have a time saving from using the cluster shortlist, and this time saving accumulate with the number of items, contributing to the total time saving.

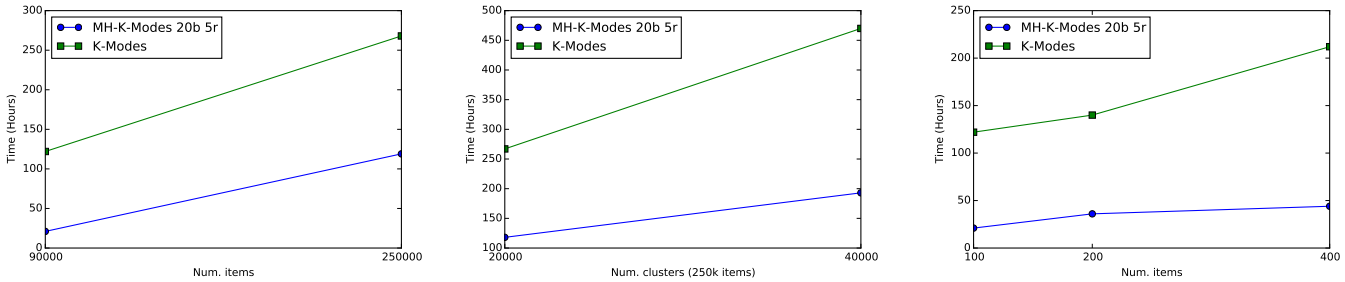
In Figure 6b it is clear that when we increase the number of clusters from 20000 to 40000 with 250000 items of data we see a much smaller rate of growth than the original K-Modes algorithm. In fact, our algorithm is able to cluster the dataset with 40000 clusters over 1.5 times faster than K-Modes clustering the dataset consisting of 20000 clusters, and over 2.5 times faster than K-Modes clustering the 40000 cluster dataset.

With higher dimensional data Figure 6c reveals that our algorithm MH-K-Modes scales at a much better rate than the original K-Modes algorithm. Specifically, the increase from 200 attributes to 400 attributes resulted in our algorithm only taking an extra 8 hours (36 to 44 hours), while the original K-Modes algorithm required an extra 72 hours (140 to 212 hours). Again, these results confirm how efficient our algorithm is and how the algorithm has indeed materialised our ideas of efficient clustering. By increasing the number of attributes, MH-K-Modes achieves a greater time saving per item comparison when compared to a lower dimensional data. Therefore as the data becomes higher dimensional, even when the candidate cluster shortlist size remains the same, we can expect greater efficiency savings.

5) Cluster Purity: For each experiment we also calculated the total cluster purity and the results are displayed in Figure 8. It is clear that in nearly all cases, our algorithm can manage to achieve a very similar cluster purity to the original K-Modes. This is a trade-off that is made for the increased efficiency in clustering.

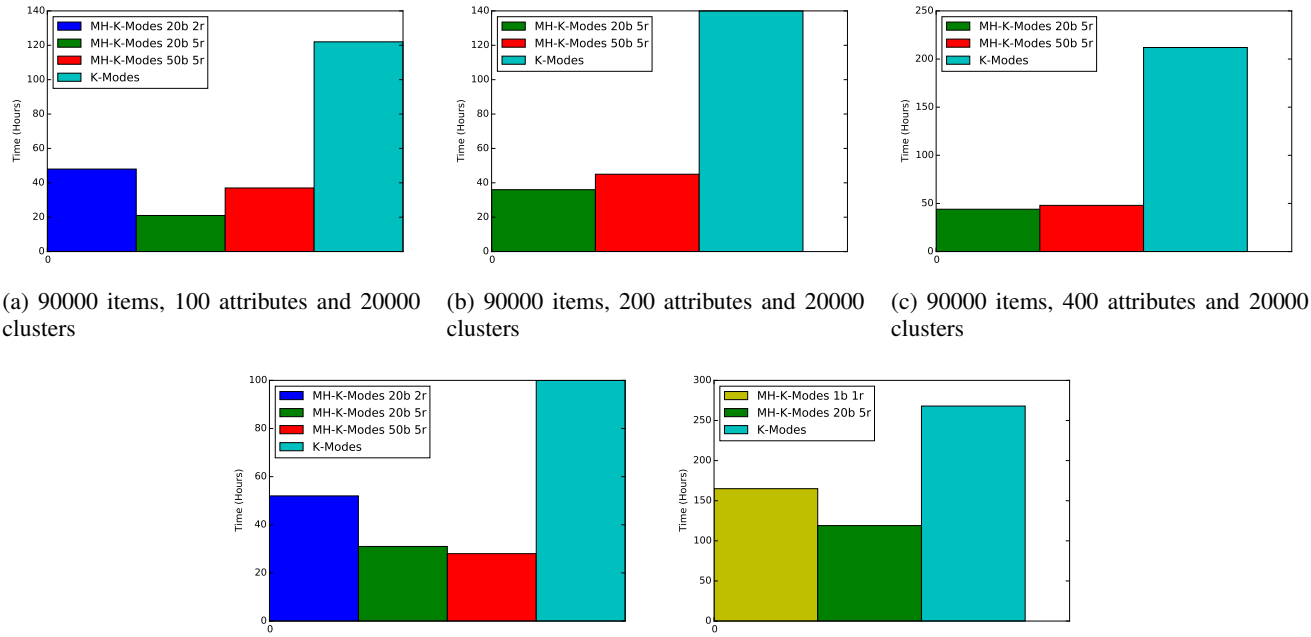
B. Results on Real Dataset: Yahoo! Answers

We will now validate our framework and algorithm on a real world dataset. Yahoo! Answers is a popular web service where users are able to ask questions and receive answers from other members of the community. When asking questions users are able to select the topic they believe best describes their question. These topics can be fine-grained and very specific to the question asked. It is the set of these topics that we will use as the ground truth for evaluation of our clustering. The objective of the clustering is: using the words of each question, group questions of the same topic together. In order to model this in a suitable format for categorical clustering, we will create a vocabulary of potential words, where each attribute value set will be Yes, No. This will indicate whether or not this word was present in the question.



(a) Scaling items (b) Scaling clusters (c) Scaling attributes

Fig. 6: Comparison of how our MH-K-Modes algorithm and K-Modes scale.



(a) 90000 items, 100 attributes and 20000 clusters

(b) 90000 items, 200 attributes and 20000 clusters

(c) 90000 items, 400 attributes and 20000 clusters

(d) 90000 items, 100 attributes and 40000 clusters

(e) 250000 items, 100 attributes and 20000 clusters

Fig. 7: Total time taken to cluster each synthetic dataset.

1) TF-IDF: To achieve reasonable results, we must first try and learn the important words from each topic. TF-IDF [23] (Term Frequency - Inverse Document Frequency) is a common weighting algorithm in text mining for statistically estimating the importance of words to a document from a collection of documents. The algorithm first calculates the frequency of each word in a single document as the initial step in calculating the significance of it. The second step of the algorithm is to calculate the 'inverse document frequency' which penalises words that occur in many documents and gives more significance to words that are rare across documents. The usefulness of this approach is clear when you consider a typical question asked on Yahoo! Answers. Here is a real example: "*im interested in being a zoologist but im not sure what do they really do.Does zoologist work only in zoo?*"

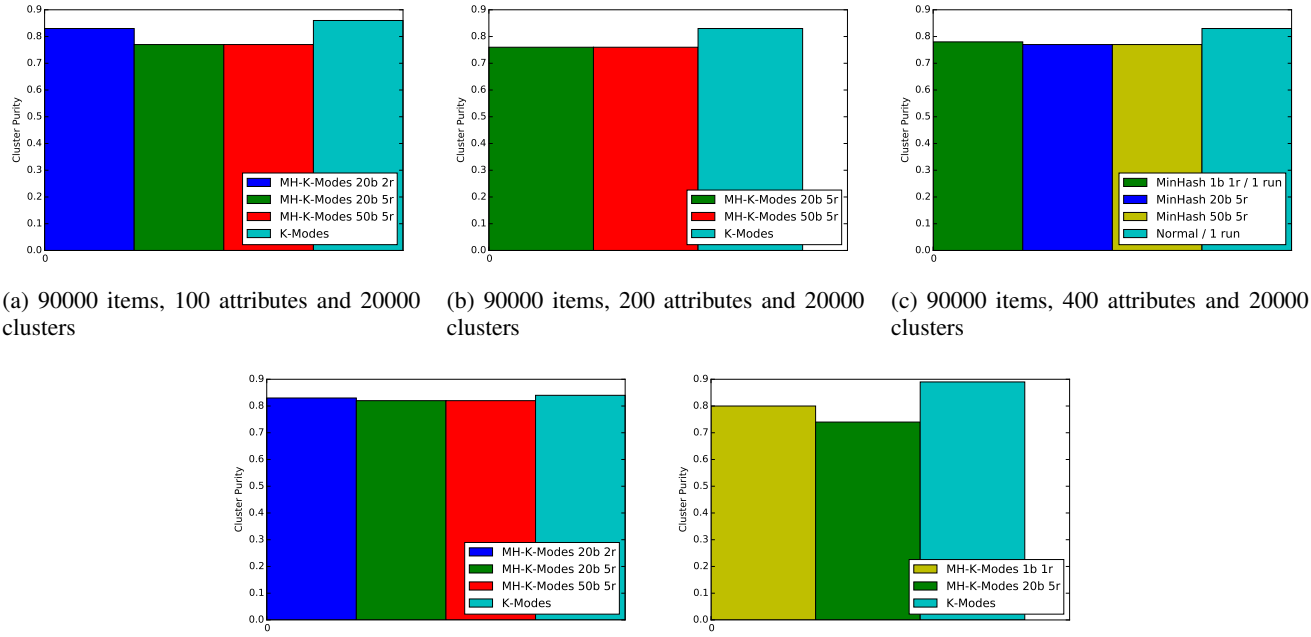
If we wish to assign this question to a topic 'Zoology', it is clear that most words would not be useful. We would expect words such as 'zoologist' and 'zoo' would be frequently occurring in the 'zoology' topic, where the rest of the question consists of words that would be frequent across many topics. Therefore we expect that they would be given a low score by the 'inverse document frequency' step of TF-IDF, leaving only the important words 'zoologist' and 'zoo' with high scores.

More formally we can state IDF as

$$idf(t_i) = log\frac{N}{n_i} \tag{7}$$

where t_i is the term we wish to calculate the importance of, N is the number of documents and n_i is the number of documents that t_i occurs in.

We will validate our framework on the Yahoo! Answers



(a) 90000 items, 100 attributes and 20000 clusters

(b) 90000 items, 200 attributes and 20000 clusters

(c) 90000 items, 400 attributes and 20000 clusters

(d) 90000 items, 100 attributes and 40000 clusters

(e) 250000 items, 100 attributes and 20000 clusters

Fig. 8: Comparison of cluster purity scores on the synthetic datasets.

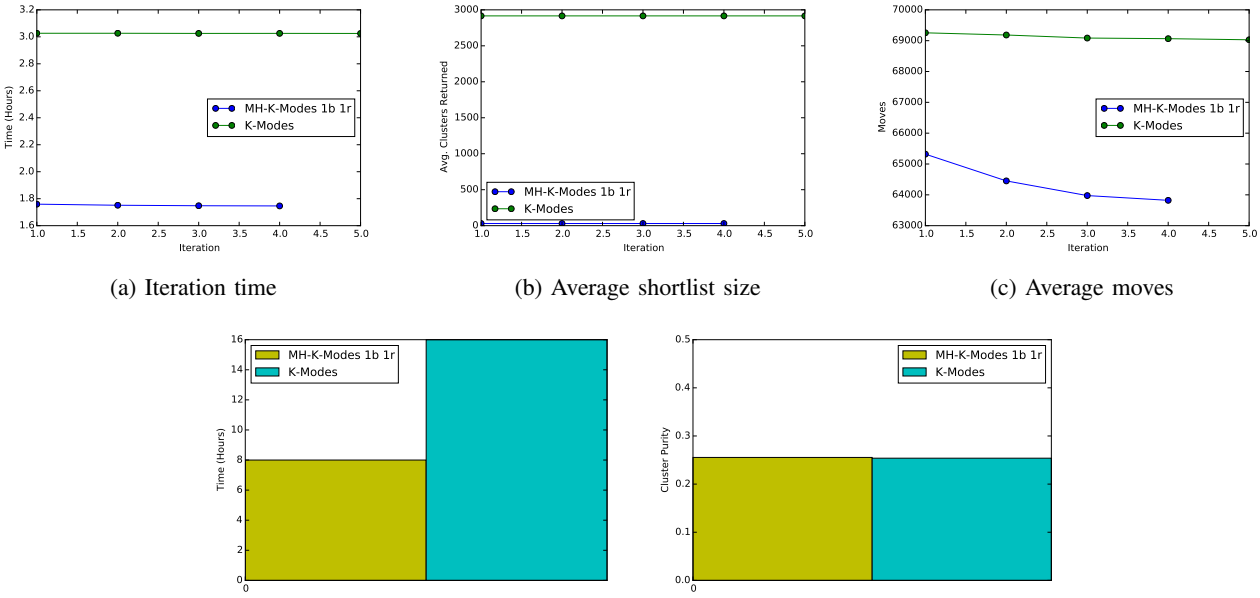
question dataset using the vocabulary of meaningful words extracted using TF-IDF for each topic. Each question is represented as a feature vector, where each feature is a binary indicator of the presence of the word in the question. The length of the feature vectors will be the size of our vocabulary. It is expected that each feature vector will be sparse, consisting mostly of negative binary indicators as each question will usually consist of only a few words out of the entire vocabulary. As our MinHash step does not take ordering of attributes into account we must augment each binary indicator with the name of the feature. That is, the value for the feature 'zoo' will become either 'zoo-0' or 'zoo-1' dependant on if it is present in the question.

For our first experiment on this dataset we extracted up to 100 questions from each of the 2916 topics. This gave us a total of 81036 items from the dataset to cluster. TF-IDF was used to extract the meaningful words from each topic, using up to 10000 words from each topic, and any word with a score over 0.7 was chosen to be included in the vocabulary. This resulted in each item consisting of 382 attributes. We must note that by using TF-IDF, and with a high threshold of 0.7, we are reducing the potential efficiency gains of our approach. As shown in Section IV-A3, greater numbers of attributes results in greater efficiency improvement with our algorithm. However, we chose to include the TF-IDF pre-processing step as performance in terms of cluster purity was poor without it, and we would like to make this experiment as realistic as possible.

Results from this experiment can be seen in Figure 9. The trends displayed are similar to those from the experiments on the synthetic datasets. Figure 9a clearly shows the improvements in the time taken per iteration, with our MH-K-Modes algorithm around 1.8 hours, compared to 3 hours for the original K-Modes algorithm. We also note that our algorithm converged after just 4 iterations, one iteration less than K-Modes. Figure 9b confirms our original motivation that our framework can create a shortlist of candidate clusters which is much smaller than the full set of all clusters, while the results of Figure 9c follows familiar trends in that our framework typically requires less movement of points between clusters each iteration. Crucially Figure 9d confirms that our framework is able to cluster the dataset faster than the K-Modes algorithm. MH-K-Modes was able to cluster our Yahoo! Answers dataset in half as much time as that required by the K-Modes algorithm. Figure 9e reveals that it was able to maintain almost exactly the same cluster purity, despite the significant increase in time savings.

We will now investigate how our algorithm MH-K-Modes and K-Modes will perform when we lower the TF-IDF scoring threshold, increasing the number of attributes each item has. As a result of lowering the threshold from 0.7 to 0.3, the number of attributes in each item increased from 382 to 2881. There are 157602 items to cluster in total with 2916 clusters.

In Figure 10 we can see a similar trend as before, our algorithm requires significantly less time per iteration which is clear in Figure 10a. Due to time constraints we set the maximum iterations to 10. In Figure 10c we see the now familiar trend of our framework creates a candidate cluster shortlist significantly smaller than the full set of clusters. This is a key reason for the efficiency gains of our algorithm. Figure 10d exhibits again the trend of our algorithm typically requiring less moves during each iterations assignment step.



(a) Iteration time (b) Average shortlist size (c) Average moves

(d) Total time taken (e) Cluster purity

Fig. 9: Yahoo! Answers questions with 0.7 TF-IDF terms.

The overall time taken for each parameter combination of our MH-K-Modes algorithm, as well as the original K-Modes algorithm is evident in Figure 10b. Here we can see that just 1 band and 1 row achieved the most efficient clustering, almost twice as fast as the original K-Modes algorithm, with around 200 hours compared to almost 400 hours. We expect that we would have found larger efficiency savings if we had not set the maximum iterations to 10, as both algorithms reached the threshold, but our MH-K-Modes algorithm appeared to be converging (Figure 10d). As we have shown before, our algorithm almost always converges faster than the original K-Modes algorithm.

2) Cluster Purity: As before, we also evaluated the purity of the resulting clusters found by both algorithms when clustering the Yahoo! Answers questions dataset created with the TF-IDF threshold set to 0.7. Figure 9e shows that we are able to achieve exactly the same cluster purity as the original K-Modes algorithm, even with the increased efficiency of taking just half the running time. We do note however that the cluster purity is quite low at just 25%. We believe this is due not just the difficulty of the problem, but also due to the fine-grained topic assignments of the data. With such a large number of very specific topics, it is to be expected that it will not be always straightforward for the algorithms to find the correct cluster out of a number of similar clusters. Furthermore, the topic assignments being user editable also makes establishing a proper and accurate ground truth clustering very difficult, as users can mistakenly choose the non-optimal topic for their question. Manually checking the question to topic assignments in the original data confirm this.

V. Conclusion

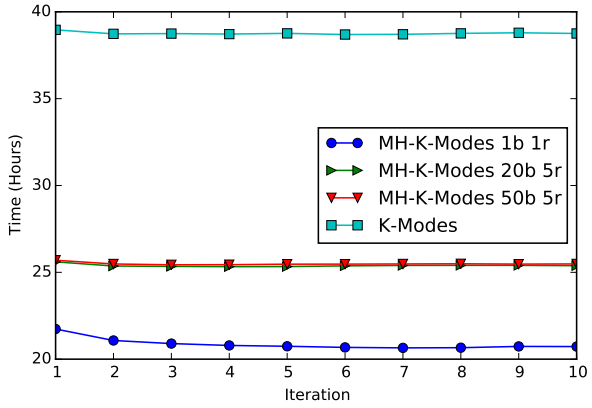
In this paper we proposed a framework for improving clustering efficiency for larger scale data by integration of locality sensitive hashing as a cluster search space reduction method. This framework had the objective of decreasing the number of distance comparisons by reducing the cluster search space for each item during the assignment step. We discussed how this framework could be used with the well-known K-Modes algorithm.

We also theoretically showed that our framework has a guaranteed error bound in terms of the clustering quality relative to the original clustering algorithm that must use the full cluster search space.

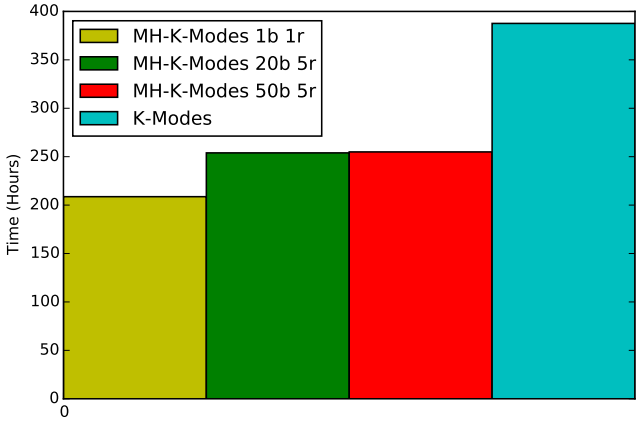
Finally we validated our framework by testing the efficiency and performance of both our MH-K-Modes algorithm and the original K-Modes algorithm on five synthetic datasets, as well as a real world Yahoo! Answers dataset. These experiments empirically proved the effectiveness of our framework for improving the efficiency of clustering large datasets which contain many clusters and attributes. We discovered both empirically and theoretically that we could achieve comparable cluster purity, but most importantly, in all tested parameter combinations and settings our algorithm MH-K-Modes was more efficient, successfully clustering the dataset at least 2 times faster and up to 6 times faster.

VI. Further Work

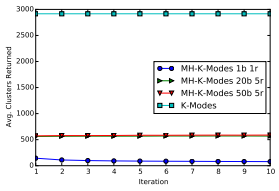
While our framework was implemented on the K-Modes clustering algorithm, evaluation on the performance and efficiency with other clustering algorithms would be worthwhile. Further, it would be interesting to investigate extending our framework to work with not only categorical data, but numeric



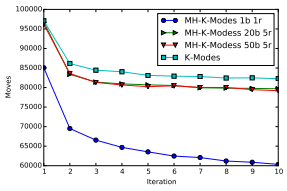
(a) Time taken per iteration



(b) Total time taken to converge



(c) Average shortlist size



(d) Average moves

Fig. 10: Yahoo! Answers questions with 0.3 TF-IDF terms (maximum of 10 iterations).

data, or combinations of both. Finally, adapting our algorithm to develop an online streaming clustering framework would be another exciting future research topic.

REFERENCES

[1] A. K. Jain, "Data Clustering: 50 Years Beyond K-Means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[2] J. B. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, 1967, pp. 281–297.

[3] Z. Huang, "Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values," *Data Mining and Knowledge Discovery*, vol. 2, no. 3, pp. 283–304, 1998.

[4] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[5] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the Web," *Computer Networks and ISDN Systems*, vol. 29, pp. 1157–1166, 1997.

[6] P. Jaccard, "Étude comparative de la distribution florale dans une portion des Alpes et des Jura," *Bulletin del la Société Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 1901.

[7] A. Z. Broder, "On the resemblance and containment of documents," in *Proceedings of the Compression and Complexity of Sequences 1997*, ser. SEQUENCES '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 21–29.

[8] "Yahoo! Answers Comprehensive Questions and Answers version 1.0." [Online]. Available: https://webscope.sandbox.yahoo.com/catalog.php?datatype=l

[9] J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[10] S. Varadarajan, H. Wang, P. Miller, and H. Zhou, "Fast convergence of regularised Region-based Mixture of Gaussians for dynamic background modelling," *Computer Vision and Image Understanding*, 2015.

[11] S. Varadarajan, P. Miller, and H. Zhou, "Region-based Mixture of Gaussians modelling for foreground detection in dynamic scenes," *Pattern Recognition*, 2015.

[12] J. Cheema and J. Dicks, "Computational approaches and software tools for genetic linkage map estimation in plants," *Briefings in Bioinformatics*, vol. 10, no. 6, pp. 595–608, 2009.

[13] M. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 103, no. 23, pp. 8577–82, 2006.

[14] J. Wang, J. Wang, Q. Ke, G. Zeng, and S. Li, "Fast Approximate K-Means via Cluster Closures," in *Multimedia Data Mining and Analytics*. Springer, 2013, pp. 373–395.

[15] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching," *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 169–178, 2000.

[16] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th international conference on World Wide Web*, 2010, p. 1177.

[17] A. S. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: scalable online collaborative filtering," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 271–280.

[18] T. Haveliwala, A. Gionis, and P. Indyk, "Scalable Techniques for Clustering the Web," *Third International Workshop on the Web and Databases (WebDB 2000)*, pp. 129–134, 2000.

[19] E. S. Silva and E. Valle, "K-medoids LSH : a new locality sensitive hashing in general metric space," *Brazilian Symposium on Database*, pp. 1–6, 2013.

[20] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognition Letters*, vol. 31, pp. 1348–1358, 2010.

[21] Z. Huang and M. K. Ng, "A fuzzy k-modes algorithm for clustering categorical data," *IEEE Transactions on Fuzzy Systems*, vol. 7, no. 4, pp. 446–452, 1999.

[22] F. Cao, J. Liang, and L. Bai, "A new initialization method for categorical data clustering," *Expert Systems with Applications*, vol. 36, no. 7, pp. 10 223–10 228, 2009.

[23] G. Salton and M. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, Ed., 1988.