Implementation of a Particle Swarm Optimization Clustering Algorithm in Apache Spark for High Dimensional Data

Matthew Sherar,
m.sherar@queensu.ca

School of Computing, Queen's University,
Kingston, ON, Canada

Abstract. Data clustering is an unsupervised learning technique critical in many domains for exploratory analysis, data synthesis, and preprocessing. Particle Swarm Optimization (PSO) is an evolutionary computing technique that has proven to produce more compact clusters than other partitional clustering techniques for a wide range of data. Apache Spark is an in-memory big data analytics framework which uses parallel distributed processing to analyze large amount of data faster than most other existing data analytic tools. Spark's library of data analytic functions does not include the PSO algorithm. In this paper we present our implementation of a hybrid K-Means PSO (KMPSO) clustering algorithm in Apache Spark and demonstrate the performance gained in Spark by comparing our implementation with an implementation of KMPSO in MATLAB. We demonstrate that KMPSO can produce better clustering results than Spark's built-in clustering algorithms, and that Apache Spark enables efficient scaling of resources to handle large and complex workloads. Clustering of text data is a complex and difficult task which most traditional clustering algorithms are unsuited for. We implementing the PSO-Variable Weighting (PSOVW) algorithm, a subspace clustering algorithm, in Apache Spark we demonstrate that it is an effective parallel clustering algorithm, suitable for clustering high dimensional data in tests with two texts datasets.

Table of Contents

1 Introduction

Clustering is an unsupervised learning process with widespread applications. Clustering algorithms group data into clusters on the basis of similarity measures. In general, it is desirable to have high similarity between the data in the same cluster and dis-similarity or separation of data among different clusters, and in this sense clustering is a multi-objective optimization problem. Particle Swarm Optimization (PSO) is an evolutionary global search algorithm founded on the cognitive and social behaviour of swarming animals. It has been demonstrated that PSO is suited to multi-objective, dynamic optimization problems. When applied to clustering, particles in the swarm represent a set of clusters in the problem space. Experiments have demonstrated that PSO can provide tighter clusters than other common clustering techniques such as K-Means clustering[5].

The rapid progression of the scale and complexity of data and the reliance of businesses, governments and individuals on this data for operations and decision making has created a Big Data Ecosystem. Apache Spark is a cluster computing engine first developed at UC Berkley's AMPLab, that was open sourced in 2014 and has been widely adopted since. Spark is very efficient for iterative data processing tasks, and therefore, is well suited for large scale machine learning problems[21]. A PSO implementation in Hadoop's MapReduce has demonstrated the improvements in efficiency that are realized from a parallel implementation. Spark can be up to 100x faster than MapReduce for iterative data processing tasks due to the optimizations associated with keeping intermediate results in memory. This experiment will analyze a PSO implementation in Apache Spark versus Apache Spark's built-in clustering algorithms. While PSO clustering algorithms have been evaluated extensively on artificial and small scale datasets, there has been little work done on exploring their effectiveness on high dimensional, large, real world datasets. The availability of user-friendly, highly efficient and scalable data processing frameworks is a very recent development. The accessibility to a wide range of robust algorithms on these platforms is essential for their success. We detail the process of converting a standard algorithm into a parallel implementation. This work demonstrates the effectiveness of PSO for clustering both small and big data domains that are comparable to industry applications.

1.1 Contribution

1. Implement PSO Clustering Algorithm in Apache Spark
2. Extend the general PSO using the following adaptations:
   (a) K-Means PSO Hybrid Algorithm
   (b) Sinusoidal PSO Inertia Function
   (c) PSO-Variable Weighting Algorithm for subspace clustering
3. Validate the algorithms against both small and large data benchmarks which included
   (a) Compare the performance of KMPSO against Spark's built-in Bisecting K-Means and K-Means Algorithms
   (b) Pre-processing text data for clustering
   (c) Clustering of labelled text data
   (d) Clustering of unlabelled text data for exploratory analysis
4. Discuss the process of parallelizing algorithms with Apache Spark

2 Background

2.1 Particle Swarm Optimization

Evolutionary computation is a family of algorithms based on the processes of biological evolution and natural selection[17]. Particle Swarm Optimisation (PSO) was first proposed by Kennedy and Eberhart[16] based on the social behavior of flocking birds or swarming fish as they move together in search of food or to avoid predators. In PSO, particles are placed in the search space, and each particle evaluates a fitness function at its current positions[24]. Each particle then moves through the search space according to the globally best position of particle in the swarm, the best position of the individual's particles past locations, and a stochastic component of movement.

Each particle is represented by three N_d dimensional vectors. One vector represents its current position, one vector its previous best position in the space, and one vector represents the particles current velocity [24]. Each particles velocity and position are updated by the following equations:

Table 1: Notations

Symbol	Represents
N_d	Number of dimensions in search space
N_c	Number of clusters
N_p	Number of particles
D	Number of data points
a_k	kth data point
$pbest$	personal best position
$gbest$	global best position
x_i	position of the ith particle
x_{ij}	position of the jth cluster in ith particle
v_i	velocity of the ith particle
r_1, r_2	random real values $\in [0, 1]$
c_1, c_2	social and local swarm constants
m	velocity inertia
w_{ik}	weight of dimension k for cluster i

$$v_i = mv_i + r_1c_1(pbest_i - x_i) + c_2r_2(gbest - x_i) \tag{1}$$

$$x_i = x_i + v_i \tag{2}$$

For the rest of this paper the notation given in Table 1 will be used.

Riccardo Poli [24] outlines the original algorithm as in Algorithm 1:

```
Initialize a population array of particles with random positions and velocities on $N_d$ dimensions in the search
 space
while Termination criteria not met do
    foreach Particle in the swarm do
        evaluate the optimization function in $N_d$ variables
    end
    if Current evaluation better than $pbest$ then
        set $pbest$ to current position;
    end
    Find the particle with the best evaluation in the swarm, and assign it to the variable $gbest$
    Update the velocity and position of the particle according to equations (1) and (2)
end
```

Algorithm 1: Original Particle Swarm Optimization Algorithm

PSO algorithms are efficient at solving complicated, multi-peak problems, and are less sensitive to initial parameters than other evolutionary computational algorithms [8] [25]. One of the main shortcomings of PSO is premature convergence. J.J Liang proposed the Comprehensive Learning Particle Swarm Optimization algorithm as a solution to the premature convergence problem [19]. Instead of each particle in the swarm being influenced by just the *gbest* and *pbest* particle, Liang proposes the following velocity update equation:

$$v_i^d = mv_i^d + r_1 k_1 (pbest_{f_i(d)}^d - x_i^d), \tag{3}$$

where for each dimension d, $f_i(d)$ determines which particle's *pbest* will be used to update the velocity of the current particle. To determine $f_i(d)$, a parameter P_c is defined for each particle by the equation:

$$P_{c_i} = 0.05 + 0.45 * \frac{\exp\frac{(10(i-1))}{(N_p-1)} - 1}{\exp(10) - 1}, \tag{4}$$

for each particle i and swarm size N_p. Then Algorithm 2 is used to determine $f_i(d)$. Liang demonstrates that the CLPSO algorithm ensures greater swarm diversity than the original PSO solution, resulting in an improved global search.

One of the largest shortcomings of PSO is premature convergence. Several adaptations have been made to address this issue, and recently in their paper Jiang et. al. propose the "Convergence-Divergence" mechanism [14]. The inertia value of each particle's velocities follows a sinusoidal function as described below:

$$m = \frac{m_{max} - m_{min}}{2} \cdot \cos\left(\frac{C \cdot \pi \cdot t}{t_{max}}\right) + \frac{m_{max} - m_{min}}{2} \tag{5}$$

Using this sinusoidal inertia function the inertia has two divergent stages where the inertia is close to m_{max} and two convergent phases where the inertia is close to m_{min}. This allows for a better global search during the divergent phases and improved convergence to optima during the convergent phases.

```
foreach Particle in the swarm, $p_i$ do
    foreach Dimension of search space do
        Generate random number in [0,1] = $rp$
        if $rp > P_{c_i}$ then
            Particle learns from its own pBest
        end
        else
            Select two particles at random from the swarm  Choose the particle with the greater fitness value
            Update $p_i$ using this particle's pBest according to equation (3)
        end
    end
end
```

Algorithm 2: CLPSO Update Algorithm

2.2 Apache Spark

Apache Spark is an open source cluster computing engine first developed at UC Berkley's AMPLab. Spark is very efficient for iterative data processing tasks, and therefore well suited for large scale machine learning problems, with an MLlib library providing implementations of several machine learning algorithms [21]. Apache Spark functions similar to the Hadoop MapReduce framework, with two main phases of its execution: a) a Map phase and b) a Reduce phase [6], however Spark has been shown to be 100 times faster in some applications. Gopalani et al. cites these performance speedups as a result of being able to cache intermediate results with Spark, and relatively low-latency data sharing across parallel operations compared to MapReduce[10]. Apache Spark also supports streaming data by discretizing the data into small batches that are buffered into the memory of its worker nodes which perform the data processing before passing the results back to the Spark Engine [11]. The performance and versatility of Spark has resulted in a widespread adoption of the framework as shown in Figure 1 and accelerated its development.

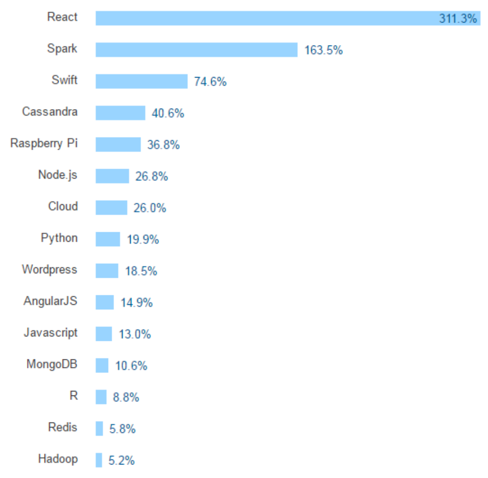


Fig. 1: 2016 Popularity growth [a]

[a] https://www.datanami.com/2016/06/08/apache-spark-adoption-numbers/

2.3 KMPSO for Clustering

Jain et al. provides a review on clustering since its introduction when the K-Means algorithm was originally published in 1955 [13]. Jain outlines the fundamental uses of clustering to understand underlying structure, identify similarity between data, and summarize data by their prototypes. Jain acknowledges that many of the issues described by Jain and Dubes in their 1988 paper are still relevant today, such as, what is a cluster? How many clusters are presented in the data?

K-Means clustering is one of the most well known and commonly used algorithms for basic data clustering. While K-Means algorithms are straightforward to understand, implement and run, they are unsuitable for many types of data, and not ideal for parallel implementation. One of the main drawbacks of the clustering effectiveness of the K-Means algorithms is their sensitivity to the initial centroids, which are set to random data points. By combining the K-Means algorithm with Particle Swarm Optimization, this issue is avoided. In the KMPSO algorithm, a swarm of N_p particles are initialized, each with a set of cluster centroids which are randomly sampled points from the dataset. Then K-Means is run for a fixed number of iterations, and the centroids of the K-Means clusters are set to the *gBest* particle. The PSO clustering algorithm is then executed. The diversity of the swarm ensures that a global search is conducted, thus not resulting in cluster centroids which are dependant on the initial choice.

2.4 Text Clustering

Attempting to cluster high dimensional data magnifies many of the questions that revolve around clustering. Text datasets are some of the most common, useful, and highly complex datasets available. The ability to cluster documents into like groups is an important step in information retrieval, summarizing documents, and as a preprocessing step for many machine learning approaches or algorithms. Representing textual information in a condensed numerical form is critical for efficient and effective text mining. One of the most common representations of text is a term-frequency inverse-document-frequency (TF-IDF) representation. Term-frequency is the number of occurrences of a word in a document. Document frequency is the total number of documents in which a term is present.The inverse document frequency is the logarithm of the document frequency divided by the total number of documents. The TF-IDF value of a term t_j in a document d_i is given by:

$$tf \cdot idf(t_j, d_i) = tf_{ij} \cdot \log \frac{df_j}{n}, \tag{6}$$

where tf_{ij} is the number of occurrences of t_j in d_i, df_j is the number of documents t_j occurs in, and n is the total number of documents in the dataset.

A TF-IDF matrix is a usually a very high-dimensional and sparse matrix. A dataset is called sparse when only a few attributes out of the total attribute space are non-zero for each data point. The dimension space of text dataset is approximately the size of the vocabulary of all the text documents. Since any one document would only contain a fraction of all the words in the whole vocabulary, the TF-IDF representation of a document in a large dataset will mostly be zeros with a few non-zero values.

Many of the issues in clustering high-dimensional data arise because as the dimensionality of the dataset increases, the distances among data points become large. This is known as the *Curse of Dimensionality.*

2.5 Subspace Clustering

It is most often the case in high-dimensional data, that the data points are clustered together among some subset of the entire dimension space. The idea of clustering data points in a high dimensional space by finding subspaces of high densities was first proposed by Aggrawal [2]. Subspace clustering is still a very important and promising area of research.

Parsons et. al. review the history of subspace clustering and detail various subspaces algorithms that have been proposed [22]. They describe subspace clustering as an extension of feature selection, where a search is conducted to find relevant attributes in a dataset. Parsons et. al. describe a simple clustering example of data to cluster which K-Means preforms poorly since the clusters are spread over irrelevant attributes, and Principal Component Analysis (dimensionality reduction) is not useful as the irrelevant dimensions are preserved in the compression. The authors overview the main methods of subspace clustering which are given briefly here.

Bottom-up methods find the density of points along each dimension and then attempt to join dense areas of the entire dimensional space. One of the first, and most well known algorithms to use this method was the CLIQUE algorithm [2].

The other main method of subspace clustering is known as the *top-down* approach. These algorithms start by finding clusters in the entire dimensional space of the dataset, then assign a weight to each dimension for each cluster. Iteratively, the cluster centroids and weights are adjusted to find clusters in relevant subspaces. This is the general method that the Particle Swarm Optimization Variable Weighting algorithm follows.

Parsons et. al. conduct a study comparing the best bottom-up and top-down approach. They conclude that both algorithms have drawbacks. Bottom-up methods scaled well as the number of instance and the number of dimensions increase, but performance declines as the size of the subspaces in which the clusters are present increases. Top-down methods can be better at finding relevant dimensions, but can be a slow algorithm and forces clusters to be spherical in shape.

Projected clustering algorithms, a subclass of subspace clustering algorithms and also first proposed by Aggarwal, aim to find a the cluster centroid of each cluster, and the set of dimensions on which each cluster exists [1]. PSOVW is a variant of a projected clustering algorithm.

3 Related Work

Esmin reviews the variations of PSO that have been used for high-dimensional clustering applications [8]. Several of the implementations are applied to clustering documents and text data, the details of these are explained below. In general, while PSO can provide better clustering results on text data than other clustering algorithms, the slow convergence to optima in high dimensional space is cited as the largest downfall of PSO. The authors cite the degeneration of particle velocities, and the presence of local optimal solutions for the very slow convergence to global optimal in high dimensional search spaces. Finally the authors provide what they believe to be key areas of work in this area. These include multi-objective PSO algorithms, PSO variants algorithms for feature selection phase and clustering algorithm analysis phase, development and use of a multi similarity measure with multiple swarm algorithms and new strategies to find the optimal number of the clusters without any prior knowledge.

3.1 Work Related to KMPSO

Cui evaluates dimensionality reduction techniques used to pre-process text data for PSO clustering [4]. Cui acknowledges that while PSO can provide clusters with smaller intra-cluster distances than K-means clustering methods, it becomes computationally intensive as the dimensionality increases. Cui experiments using Latent Semantic Indexing (LSI) and Random Projection (RP) as pre-processing dimensionality reduction techniques for text data before using PSO to cluster the data. It was found that while LSI and RP significantly reduce runtime, the F-measure evaluation of the clustering results were slightly worse than that with no dimensionality reduction (and much worse if there were less than 400 dimensions).

Chunne used a PSO algorithm to cluster streaming twitter data on the Hadoop MapReduce platform [3]. The twitter data is processed by tokenizing, stemming and removing stop words from each tweet. The dataset is represented as a set of individual tweet vectors, where each individual tweet vector is the set of term weights of all words, where the term weight is calculated based on the number of occurrences of each word. Each particle in the swarm is a set of cluster locations that searches, using a PSO algorithm running in parallel across several nodes, for an optimal set of cluster locations based on tweets in the word space. It was discovered that using K-Means to set the initial cluster locations resulted in faster convergence.

Cui used a PSO algorithm to perform document clustering, finding that it was able to produce more compact clusters than a K-Means clustering algorithm [5] . Each document is represented as a point in the vector space, where the document vector is defined identically to that of [11]. The basic PSO algorithm used describes each particle as a vector of cluster locations in the word space, with each particle adjusting all clusters to find optimum values. A hybrid approach is also presented which uses PSO to find initial cluster locations then a K-Means algorithm to refine the optimum values to increase computational efficiency.

Lam uses cluster matching: sequencing each particles cluster centroids to correspond with the global best particle order, to enhance particle collaboration and improve search [18] . The high dimensionality and complexity of gene expression data makes clustering a computationally expensive task. Experiments show that PSO-KM CM2 algorithm achieves faster convergence than KMPSO CM, KMPSO and PSO.

Gaikwad identifies one of the major downsides to the K-Means clustering algorithm as being unable to accommodate new data arriving via a data stream [9]. Instead they implement an adaptive PSO algorithm to continue iteration from its previous state to accommodate all new data that is continuously arriving.

3.2 Work Related to PSOVW

The main body of work on which this PSOVW implementation is based is by Lu et. al. in which they implement the original PSOVW Algorithm [20]. Lu et. al. take ideas from a few subspace clustering algorithms.

Jing et. al. propose an Entropy-weighted K-Means subspace clustering algorithm [15]. Their algorithm follows a top-down approach in which each cluster centroid is defined in the full dimensional space, but then each dimension of each cluster centroid is assigned a weight. They attempt to minimize within-cluster dispersion while also maximizing the negative cluster weight entropy. They use Lagrangian multipliers method to derive an expression that solves the multi-objective optimization problem. Their experiments on synthetic and real test data showed that their algorithm was able to find subspace clusters more effectively than other K-Means and subspace clustering algorithms.

Elhamifar et. al. review common approaches to subspace clustering algorithms and propose the Sparse Subspace Clustering algorithm (SSC) [7]. Their SSC algorithm takes an algebraic approach, first finding a few similar (those in the same subspace) data points for each data point in the dataset, and then use that information to perform spectral clustering to identify the clusters. The algebraic fact they make use of in the first step of their algorithm is that a *data point in a union of subspaces can be efficiently reconstructed by a combination of other points in the dataset.* The use of a spectral clustering approach allows for the identification of noise and outliers in the data. Their tests conducted on clustering faces and motions in videos found that their algorithm could compete with state of the art methods.

4 Implementation

4.1 KMPSO Spark Implementation

In a clustering implementation of PSO, each particle in the swarm represents a set of cluster centers. Each cluster center is an N_d dimensional vector. Each particle in the swarm is represented as:

$$x_i = \{c_1, c_2, ..., c_n\} \tag{7}$$

$$v_i = \{v_1, v_2, ..., v_n\} \tag{8}$$

$$pbest_i = \{pbc_1, pbc_2, ...pbc_n\} \tag{9}$$

To evaluate the fitness of each particle, we calculate the minimum distance between each data point and the nearest cluster in the swarm. The distance between each cluster and each data point is calculated according to euclidean distance, given in equation below:

$$d(x_{ij}, a_k) = \sqrt{\sum_{n=0}^{N_d} (x_{ij}(n) - a_k(n))^2}. \tag{10}$$

Algorithm 3 outlines the pseudocode of the KMPSO implementation using Apache Spark. The algorithm utilizes three of the core parallel functionalities of Apache Spark. Data in Spark is stored in Resilient Distributed Datasets (RDDs) which are immutable data structures distributed among the worker nodes in the computing cluster. These RDDs allow for operations and transformations on the data to be applied in parallel. In the PSO clustering algorithm implemented in Spark, the data which is to be clustered is stored in RDDs. A swarm is initialized by sampling data points randomly from the dataset, using built-in Spark functionality and setting these as the initial cluster centroids for each particle. The main speedup from using Spark results from determining each particles fitness in parallel. To compute a particles fitness, each cluster center in the particle needs to be compared to each data point. In order to parallelize this process, all the particles in the swarm are sent to each worker node via Spark's Broadcast functionality as shown in Figure 2. Then each each worker node can compute the error from each data point stored in an RDD on the worker node. An Accumulator variable, which is another built-in Spark functionality, is a means to store each particle's error from each data point distributed across the worker nodes, and then allow the error to be collected and summed at the driver node when all the error calculation is complete.

As the size of the swarm and the computation involved in updating swarm positions and velocities is relatively minor compared to computing the error, this can be done locally in the driver node. Once all particles are updated, they are rebroadcast to the worker nodes to repeat the process of error calculation in the next iteration.

4.2 PSOVW Spark Implementation

The implementation of the Particle Swarm Optimization Variable-Weighting (PSOVW) clustering algorithm follows the description of the algorithm detailed by Yanping Lu et al. [20]. The version that uses fitness functions most suitable for text clustering is followed. In the PSOVW algorithm each particle consists of a set of cluster centers, a set of weights for each cluster, for each dimension, and a set of velocities for the

```
Data: Dataset to be clustered = $data$
Data: Number of clusters = $N_c$
Data: Number of particles = $N_p$
Data: Number of iterations = $iterations$
$particles \longleftarrow ArrayBuffer$
while not all particles initialized do
    Initialize $N_c$ clusters sampled from $data$
    Initialize $N_c$ random velocity vectors
    Add particle to particles
end
Create two additional particles using MLLib. and add them to the swarm
$particles.toArray$
$gBest \longleftarrow MLlib.Kmeans(data, N_c, numIterations).clusterCenters$  // gBest particle initialized with
 K-Means centroids
$minSwarmError \longleftarrow MLlib.Kmeans(data, N_c, numIterations).computeCost$
Broadcast Particles to worker nodes
for $k \longleftarrow 1...iterations$ do
    Update inertia according to equation (9)
    $error \longleftarrow$ Spark Accumulator        // Accumulator is a vector to sum errors for each particle
    // Spark mapPartitions used to iterate through $data$
    foreach $d \in data$ do
        foreach $p \in particles$ do
            Calculate min distance from cluster in $p$ to $d$
            $errors.add$(min Distance)
        end
    end
    Destroy Broadcast variable
    Collect errors into an Array $error$
    foreach $p \in particles$ do
        if $error(p) < p.error$ then
            Set $pbest$ to current position
        end
        if $error(p) < minSwarmError$ then
            $minSwarmError \longleftarrow error(p)$
            $gBest \longleftarrow p$
        end
        foreach $cluster \in p$ do
            Update Cluster Velocity
            Update Cluster Position
            Update Particle Error from error(p)
        end
    end
    Broadcast updated particles
end
```

Algorithm 3: PSO-KM Apache Spark Implementation

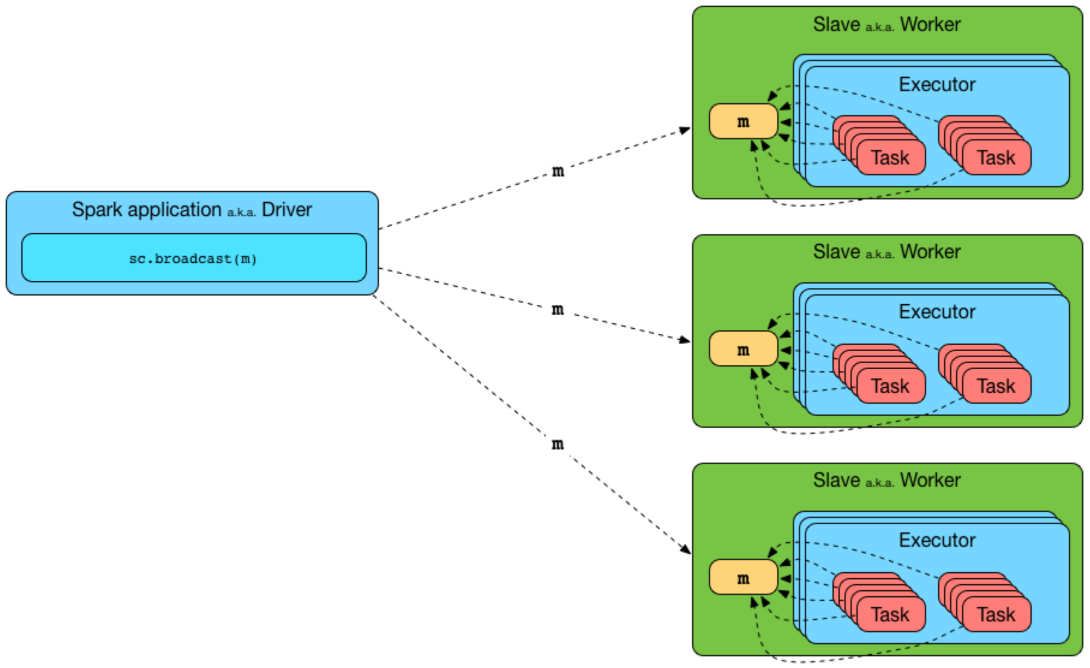


Fig. 2: Broadcast Variables in Spark [a]

[a] https://github.com/jaceklaskowski/mastering-apache-spark-book/blob/master/book-intro.adoc

weights of each of the clusters. The basic principles of swarm evolution are the same as in the original PSO algorithm and as the PSOKM algorithm. Each particle's position and velocity is initialized, and on each iteration the overall fitness of each particle is calculated and then the velocities and positions of each particle are updated.

The aim of the PSOVW is to find clusters that are defined in relevant subspaces of the entire dimension space. To achieve this using particle swarm, the position of the swarm is set as the variable weights for each cluster for every dimension. These weights are initialized to values in the range $[0, 1]$. Velocities are initialized to random values in the range $[-0.25, 0.25]$.

To determine the membership of each data point d_k to a cluster in a particle the following equation is used:

$$u_{l,i} = \begin{cases} 1 & \text{if } \sum_{j=0}^{N_d} \left(\frac{w_{lj}}{\sum_{j=0}^{N_d} w_{lj}}\right) \cdot d(x_{lj}, a_{ij}) \leq \sum_{j=0}^{N_d} \left(\frac{w_{lj}}{\sum_{j=0}^{N_d} w_{lj}}\right) \cdot d(x_{lj}, a_{ij}) \text{ for } 0 \leq l \leq N_c \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

The equation below is used to calculate the centroid positions each iteration.

$$x_{lj} = \frac{\sum_{i=0}^{N_c} u_{li} \cdot a_{ij}}{\sum_{i=0}^{N_c} u_{li}} \tag{12}$$

Finally the overall fitness of each particle is calculated using the equation below.

$$F(W) = \sum_{l=0}^{N_c} \sum_{i=0}^{D} \sum_{j=0}^{N_d} u_{li} \cdot \left(\frac{w_{lj}}{\sum_{j=0}^{N_d} w_{lj}}\right)^{\beta} \cdot d(x_{lj}, a_{ij}) \tag{13}$$

The PSOVW algorithm can realize more speedup gains as a result of parallelization compared to the PSOKM algorithm. This is because there are several steps to the algorithm which can be computed in parallel. A brief pseudocode is given below.

```
foreach Particle $\in$ Particle Swarm do
    Initialize cluster centroids to random data points
    Initialize weights and velocities to random numbers
end
for $k \longleftarrow 1...iterations$ do
    Broadcast swarm to all RDD's
    Map over data points and for each particle assign data point to cluster centroid according to 11
    foreach Particle $\in$ ParticleSwarm do
        Map over all data points, accumulate sum of data points belonging to each cluster, update cluster
        centroids according to 12
    end
    Map over data points and calculate fitness function for each particle using 13 with Spark Accumulator
    Use ?? to update positions and velocities for each particle
end
```

Algorithm 4: PSOVW Apache Spark Implementation

4.3 Experimental Setup

The focus of these experiments is in evaluating the KMPSO algorithm in Apache Spark, against Apache Spark's built in K-Means algorithm. We evaluate clustering results of both on a range of simple, small datasets, and high dimensional complex datasets. While there has been a significant amount of research done on PSO clustering algorithms, these experiments seek to evaluate the performance on a wide range of real-world datasets, in a common, accessible and scalable environment. In addition to evaluating clustering results, we also examine the effects of using different inertia functions on the convergence rate of the KMPSO algorithm. Finally we assess the scalability of KMPSO algorithm, and its computational resource requirements compared to Apache Spark's built in K-Means algorithm.

There have been many variations of a hybrid PSO-KM algorithm. In our experiments we chose parameters $c_1 = c_2 = 1.49$ according to [5]. The inertia follows the sinusoidal function given in equation(9).

4.4 Datasets

The implementation of PSOKM on Spark was tested on a standard set of common clustering benchmarks, and then on several larger text datasets.

Small Datasets

- **Diamond:** An artificial dataset of with 9 classes and 2 features
- **Wisconsin Brest Cancer Diagnosis:** A labelled dataset classify breast cancer tumours as malignant or benign with 30 features
- **Fourty:** An Artificial dataset with 40 classes and 2 features

High Dimensional Datasets

- **Presidential Debates:**[1] A dataset containing a transcript of the electoral debates containing a dialogue labelled by speaker of all participants in the debate.
- **20newsgroups:** [2] A collection of 20000 short text documents classified into 20 well separated and evenly distributed classes. 4 classes were selected (comp.graphics, sci.med, politics.mideast and sports.baseball) each with 100 documents.

[1] https://www.kaggle.com/mrisdal/2016-us-presidential-debates
[2] http://qwone.com/?jason/20Newsgroups/

4.5 Text Clustering - Preprocessing

The transformation of raw text into meaningful and efficient representations is a large and open area of research. To cluster the text in the datasets listed above, prior to transforming the text into the $TF-IDF$ matrix, all text is converted to lower case, and a regex command was used to filter non-word characters. The text was tokenized, whereby each word document is represented as an array of words on which the $TF-IDF$ transformation is applied. These preprocessing steps follow the procedures outlined by Pentreath [23]. The $TF-IDF$ matrix is computed using the $TF-IDF$ transformer included Apache-Sparks MLlib. The default dimension of $TF-IDF$ matrix computed by Spark is 2^{18}. The final dimensionality of the $TF-IDF$ matrix for the 20Newsgroups dataset was 400x5832.

4.6 Evaluation Metrics

Several evaluation metrics will be used to analyze performance. The first is a simple and reliable metric commonly used to evaluate partitional clustering algorithms such as K-Means.

Sum of Euclidean Distance: The euclidean distance between a cluster center x_i and a data point is defined in equation 10. For each particle in the swarm, the error is taken as the minimum distance between a point and any cluster defined in below:

$$\sum_{k=0}^{D} \min_{\forall x_{ij} \in x_j} d(x_{ij}, d_k) \tag{14}$$

F-Measure: The second measure used is the F-Measure, described here similarly to [4]. The F-Measure requires data that is already assigned class labels. To evaluate the cluster centroids generated by the clustering algorithm, it is necessary to compute the precision of the clusters, P and the recall, R. P and R are given below:

$$P(i,j) = \frac{N_{ic}}{N_i} \tag{15}$$

$$R(i,j) = \frac{N_{ic}}{N_c} \tag{16}$$

where N_{ic} represents the number of data points labelled as class i and clustered into cluster c, N_c represents the number of items in class c and N_i is the number of data points clustered into centroid i. A single clusters F-measure can then be calculated as follows:

$$F(i) = \frac{2PR}{P+R}, \tag{17}$$

and the total F-measure for a set of cluster centroids as:

$$F = \frac{\sum_{i=1}^{N_c} N_i \cdot F(i)}{\sum_{i=1}^{N_c} N_i}. \tag{18}$$

Entropy: Finally Entropy provides a measure of the distribution of classes within a given cluster [12]. Entropy also uses pre-assigned class labels to determine the number of data points in each class that have been clustered to each centroid. Below n_{ij} is the number of data points of class i clustered into centroid j, and n_j is the total number of items clustered into centroid j.

$$e_j = \sum_{i=1}^{N_c} \frac{n_{ij}}{n_j} \log \frac{n_{ij}}{n_j} \tag{19}$$

$$E = \sum_{j=1}^{N_c} \frac{n_j}{D} e_j \tag{20}$$

4.7 Experimental Results

The average total run-time of the KMPSO algorithm is shown in Figure 3 for both MATLAB and Spark implementations. In these experiments both algorithms were run for 50 iterations. The Spark implementation of KMPSO is over 200% faster than the MATLAB implementation, clearly demonstrating the efficiency resulting from computing each particles fitness function in parallel.

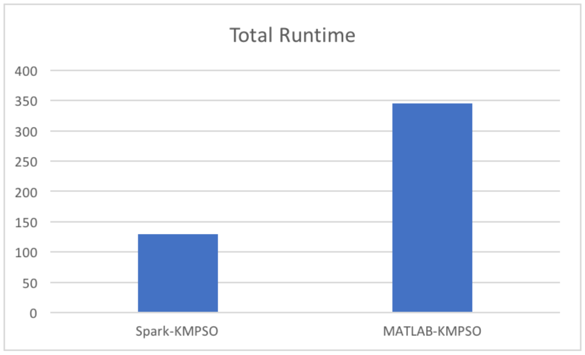


Fig. 3: Scaled vs Standalone Total Computation Time

Figure 4 shows the average per-iteration computation time for each algorithm. The KMPSO per-iteration time is over 10x faster in Spark than in MATLAB. The reason the overall computation times for the two implementations are more similar than the per-iteration computation time is due to the greater initialization time required by Spark to read in and distribute the data among RDD's. Once this process is complete, the full effect of parallelization become apparent.

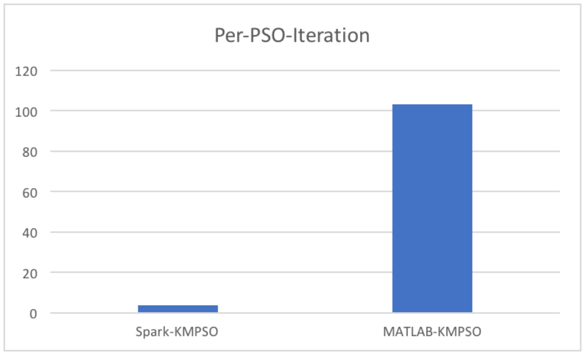


Fig. 4: Scaled vs Standalone Per-Iteration Computation Time

To test the scalability of the KMPSO algorithm on Apache Spark, experiments were conducted on different sizes of computing clusters. All nodes in the clusters are AWS c3.2Large instances. As a comparison, Apache Spark's built-in K-Means algorithm was also tested on each computing cluster. The results in Figure 5 demonstrate that as the size of the cluster increases, the faster the KMPSO algorithm can execute. Although the K-Means algorithm runs faster, which is to be expected, there is very little speed-up realized from increasing the computation size. This is likely due to the fact that in each iteration of the K-Means algorithm, the average centroid of each cluster of data points needs to be computed, whereas in the KMPSO algorithm, each cluster centroid is simply updated based on it's velocity which is only dependant on the other particles in the swarm.

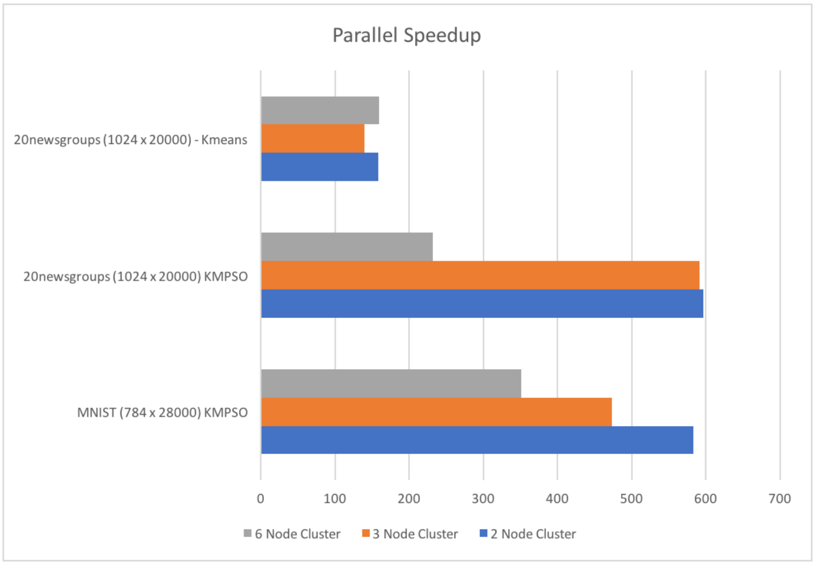
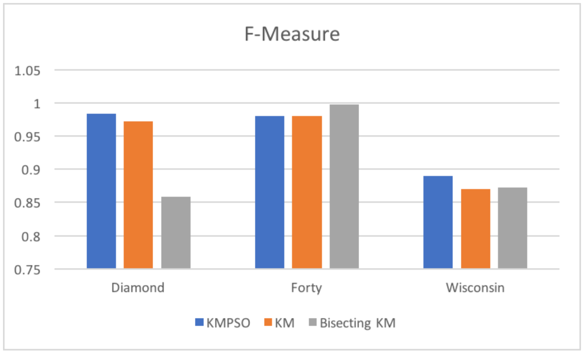
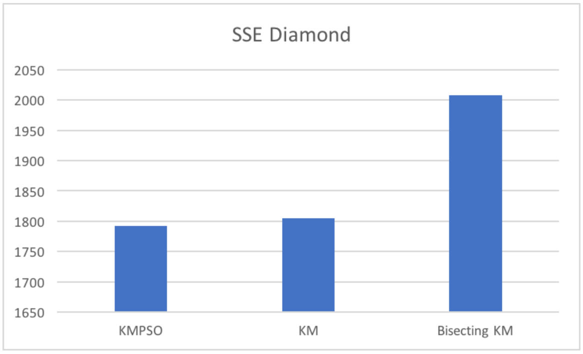


Fig. 5: Cluster Size Speedup

The KMPSO and K-Means clustering algorithms were evaluated on their clustering performance using the Forty, Diamond and Wisconsin datasets on a single-node cluster. We conducted 30 experimental trials with each algorithm, with each of the basic datasets. Each algorithm was run for a fixed number of 200 iterations. A swarm with 10 particles was used, and in the initialization of the KMPSO algorithm Apache Spark's MLlib K-Means iteration was run for 60 iterations to determine the initial *gBest* particle clusters.



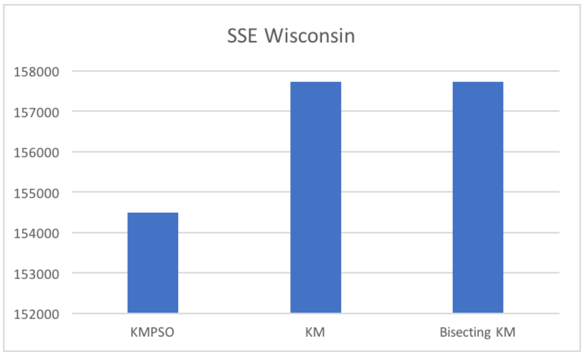
(a) Small Dataset F-Measure



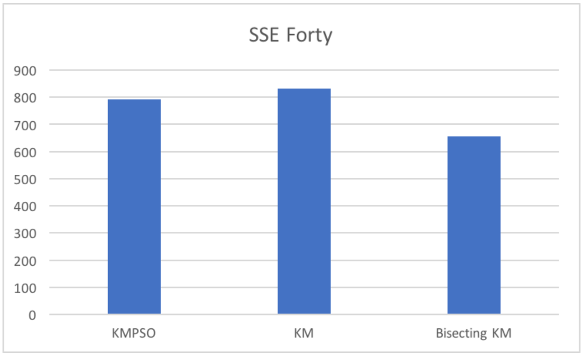
(b) SSE Diamond Dataset

Fig. 6: Clustering Evaluation

In Figure 6 (a) shows the results of the F-Measure classification accuracy for the Wisconsin, Forty and Diamond datasets. An F-Measure value of 1 would indicate perfect clustering accuracy. For both the Diamond and Wisconsin dataset the KMPSO achieves a lower F-Measure than both K-Means ands Bisecting K-Means. For the Fourty dataset the Bisecting K-Means algorithm achieves almost perfect clustering accuracy.



(a) SSE Wisconsin Dataset



(b) SSE Fourty Dataset

Fig. 7: Clustering Evaluation

In Figure 6 (b) and 7 (a) and (b) are the is the sum of euclidean distance results. We see in the real-world dataset, the Wisconsin dataset, that the KMPSO algorithm achieves superior clustering in this example of relatively dispersed clusters over 30 dimensions. In the Diamond dataset, in which the data is suitable to be clustered by K-Means algorithms, the KMPSO achieves a lower error.

When testing using the 20Newsgroups dataset a cluster of one driver and two worker nodes, all of which were c3.2xlarge AWS dedicated instances, were used. Below the average of three trials of entropy and F-Measure is shown.

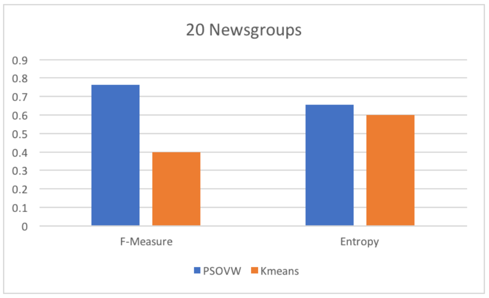


Fig. 8: 20 Newsgroups Clustering Results

In Figure 8 we see that the PSOVW algorithm is able to achieve an F-measure of 0.8 on clustering documents into their four distinct groups. This far exceeds that standard K-Means clustering algorithm. In fact, when the cluster assignments from the K-Means function were examined, all documents were being assigned to the same cluster. This explains K-Means poor F-Measure result while the K-Means clustering entropy is actually lower than the PSOVW: 3 of 4 of K-Means clusters have 0 entropy as they have no documents at all. While many classification algorithms would be able to achieve near 100% accuracy in clustering these documents, a classification algorithm is able to learn from feedback it receives from previous predictions. For a clustering algorithm to effectively identify the relevant subspaces in which clusters exist with about 80% accuracy with no external information is a good result. While recent subspace clustering methods such as the Orthogonal

Matching Pursuit algorithm proposed by You et. al.[26] are able to achieve very high clustering accuracy on similar datasets, the PSOVW is a simple, adaptable and scalable algorithm.

5 Application - US Election Debates

To demonstrate the usefulness of the PSOVW algorithm a document containing the transcript of the US Presidential Debates betweens Hillary Clinton and Donald Trump was clustered. Each data point in the dataset corresponds to a unique response given in the debate. To achieve a variety of different clusters, the PSOVW was run set to 8 clusters. In the analysis below we present the variable weights corresponding to three of the clusters identified by PSOVW. Some of the terms that correspond to the most relevant dimensions are labelled for each cluster. In addition, a sample of the phrases that were clustered into each cluster are given.

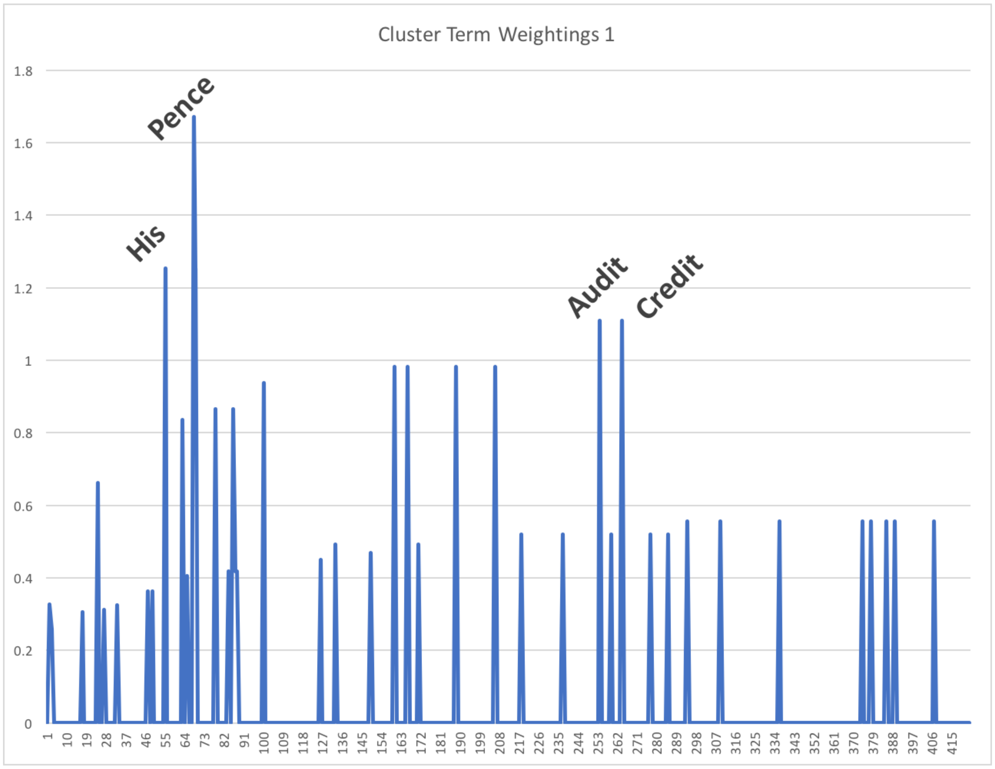


Fig. 9: Cluster 1 Term Weights

In Figure 9 we can see the dimensions which represent the subspace that cluster 1 is defined in. This cluster contains phrases from both speakers, and most phrases revolve around Trumps taxes.

Cluster 1 Phrases

– ”I don’t mind releasing – I’m under a routine audit. And it’ll be released. And – as soon as the audit’s finished”
– ”New York – New York has done an excellent job. And I give credit – I give credit across the board going back two mayors”
– ”I am interested to hear whether he’ll defend his running mate’s not releasing taxes and not paying taxes.”

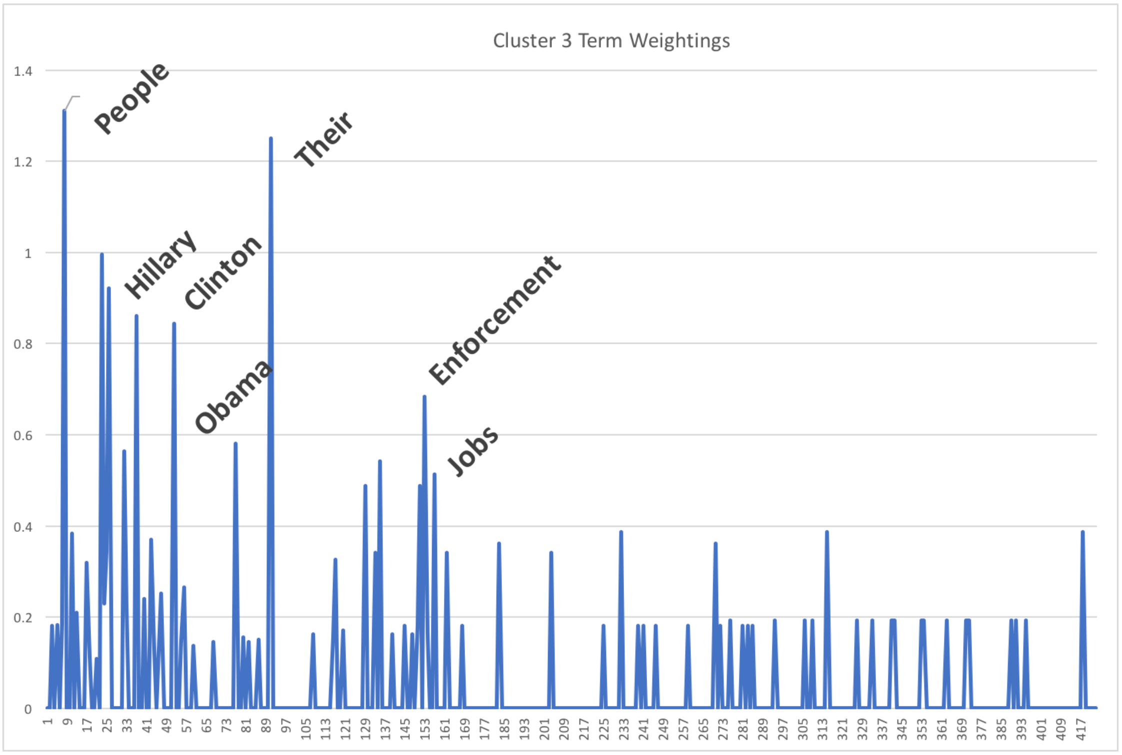


Fig. 10: Cluster 3 Term Weights

Phrases that belong to cluster 3 exist in a higher dimensional subspace than those for cluster 1, as we can see by the larger number of spikes in the variable weights shown in Figure 10. These phrases are almost all spoken by Donald Trump, and mostly focus on his attacks towards his rival. Therefore, as is in the selection of phrases below, most contain Hillary Clinton's name.

Cluster 3 Phrases

– ”But there's a – there's a reason why people question the trustworthiness of Hillary Clinton. And that's because they're paying attention.”
– ”Hillary Clinton – Hillary Clinton – Hillary Clinton failed to renegotiate a status of forces agreement”
– ”There are millions more people living in poverty today than the day that Barack Obama with Hillary Clinton at his side”
– ”We have different views on – on refugee issues and on immigration. Hillary and I want to do enforcement based on”
– ”It's called extreme vetting. We are going to areas like Syria where they're coming in by the tens of thousands because of Barack Obama. And Hillary Clinton wants to allow a 550 percent increase over Obama. People are coming into our country like we have no idea who they are”

Figure 11 shows that cluster 8 is dominated by the dimension corresponding to the word 'she'. As a result almost all the phrases in this cluster are by Donald where he is referring to Hillary Clinton as 'she'. As Hillary Clinton was Secretary of State, most of his comments challenge her policies, thus why terms like 'ISIS' are also relevant for this dimension.

Cluster 8 Phrases

– ”She talks about solar panels. We invested in a solar company”

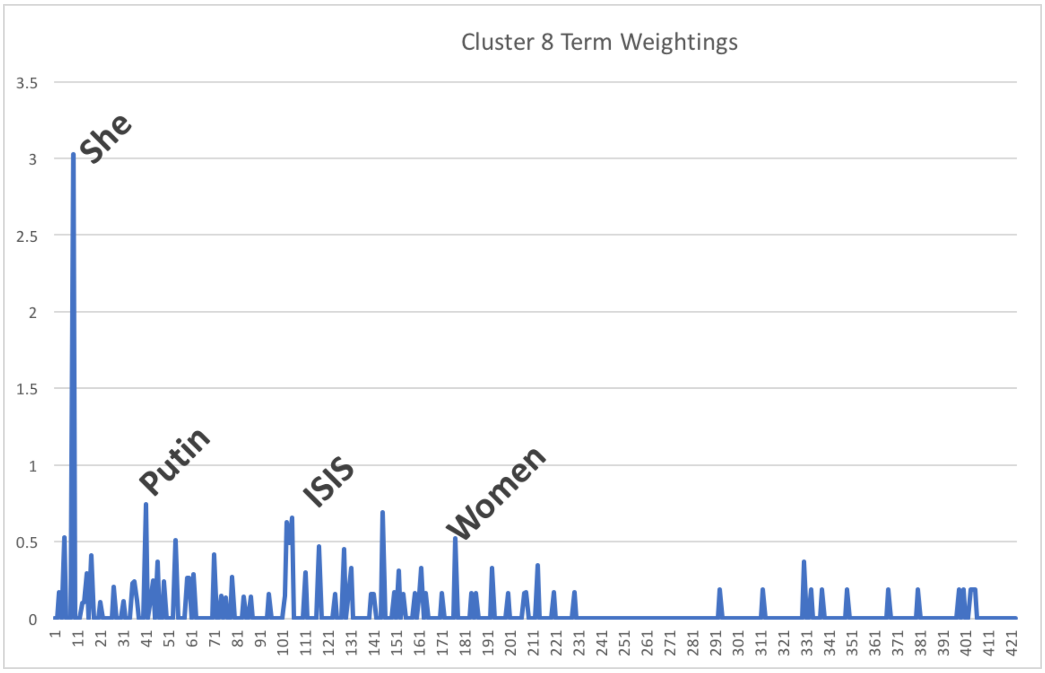


Fig. 11: Cluster 8 Term Weights

- ”And look at her website. You know what? It’s no difference than this. She’s telling us how to fight ISIS. Just go to her website. She tells you how to fight ISIS on her website. I don’t think General Douglas MacArthur would like that too much.”
- ”I have a much better – she spent – let me tell you – she spent hundreds of millions of dollars on an advertising – you know”
- ”She doesn’t have the look. She doesn’t have the stamina. I said she doesn’t have the stamina. And I don’t believe she does have the stamina. To be president of this country”
- ”The Clinton Foundation accepted foreign contributions from foreign governments and foreign donors while she was secretary of state”
- ”Excuse me. She just went about 25 seconds over her time”
- ”Excuse me. Because she has been a disaster as a senator. A disaster.”
- ”She’s got tremendous – she’s got tremendous hatred. And this country cannot take another four years of Barack Obama”

6 Discussion

The Apache Spark implementation of the KMPSO algorithm, on of the main contribution of this work, was shown to be much faster than a single-node MATLAB implementation of the same algorithm. Evolutionary algorithms are an important and promising tool in many applications and area’s of research. Particle Swarm Optimization in particular is widely used for industrial applications, and as these industrial processes become increasingly digitized and data-driven, it will be essential to have scalable algorithms that can effectively optimize processes and solve engineering problems of scale. This implementation of KMPSO scales well given increased computational power, thanks to Apache Spark’s efficient data-processing engine.

While Apache Spark is a very popular and effective tool for many applications, it is a relatively young technology that is rapidly being developed. A key to the success of Spark will be the ease at which businesses and individuals can implement custom algorithms and solutions for a diverse set of problems. The implementation of KMPSO in Spark was not straightforward. Spark’s built-in Vector library has limited functionality available in the public API's, and some very basic operations are missing such as adding two vectors. Without being able to use Spark's built-in Vector library to compute the Particle Swarms velocity and position, mean't that all Vectors had to be converted to Breeze Vector's, which is a linear algebra engine built for Scala. This required needless and expensive conversion of Spark Vectors to Breeze Vectors and back again at the end of the computation.

On the other hand, Spark's Accumulator, Map, Reduce and Broadcast functions make parallelization of algorithms straightforward. Particles could be sent to all the worker nodes with one Broadcast function call, the fitness function could be evaluated in parallel using the Map function. All the errors for each particle could be collected and sent back to the master node with the Accumulator function. Since Spark handles all the optimization of distributing the data and computations, this makes the code clean and easy to understand.

The KMPSO algorithm was shown to be able to achieve greater or equal clustering accuracy on test datasets than existing K-Means and bisecting K-Means implementations on Spark. In addition, the adaptability of the PSO algorithm to different problems simply by modifying or adding fitness functions allows it to be used to solve a wide range of clustering problems. PSO is widely used in industrial and engineering applications, and the ability to cluster large amounts of data using fitness functions unique to the needs of the problems, and achieve an effective global search of the solution space, leaves a wide range of possible applications for a Spark implementation of the KMPSO algorithm.

Like most traditional clustering algorithms, the KMPSO algorithm preforms poorly at clustering very high dimensional data, due to the *Curse of Dimensionality.* Following a previous work to implement a PSO driven subspace clustering algorithm, we demonstrated that high dimensional data could be effectively clustered in parallel using the PSOVW algorithm implemented in Apache Spark. The application of clustering US Presidential Debates shows how information about the relevant dimensions to each cluster can be insightful to describe the attributes, significance and differences among clusters. Given a large amount of text data to cluster, using the PSOVW algorithm, one could easily summarize the topics and characteristics of each cluster with just the variable weights assigned to each cluster.

7 Conclusion

We contributed parallel implementation of KMPSO and PSOVW algorithms in Apache Spark. Spark is a popular data processing engine that is quickly gaining functionality and usefulness in tackling machine learning problems. We demonstrated the scalability of the KMPSO algorithm and its robust clustering effectiveness. In our tests with the PSOVW, it was seen that the PSOVW was able to effectively cluster text, and significantly outperformed traditional clustering algorithms implemented in Apache Spark.

The future work in this area of research is to extend the KMPSO and PSOVW to be suitable for streaming data. Apache Spark has a powerfully streaming engine called Spark Streaming. Many of todays data processing needs involve streaming data. The ability to effectively cluster and label steaming data, such as tweets, news stories, or error logs would be very useful in many scenarios. There has been work completed on streaming clustering algorithms, including with PSO. PSO could be an effective streaming clustering tool, as the use of multiple swarms and multiple fitness functions could allow for a unique interaction between existing clusters and new data points.

There is also future work to be done with specific applications of the PSOVW algorithm. One application would be in clustering medical doctors patient notes to be able to understand trends, diagnosis and promote knowledge transfer. Further experiments with different types of text data such as Tweets is also a possible extension of this work.

References

1. Aggarwal, C.C., Wolf, J.L., Yu, P.S., Procopiuc, C., Park, J.S.: Fast algorithms for projected clustering. In: ACM SIGMoD Record. vol. 28, pp. 61–72. ACM (1999)
2. Aggrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications, vol. 27. ACM (1998)
3. Chunne, A.P., Chandrasekhar, U., Malhotra, C.: Real time clustering of tweets using adaptive pso technique and mapreduce. In: Communication Technologies (GCCT), 2015 Global Conference on. pp. 452–457. IEEE (2015)
4. Cui, X., Beaver, J.M., Charles, J.S., Potok, T.E.: Dimensionality reduction particle swarm algorithm for high dimensional clustering. In: Swarm Intelligence Symposium, 2008. SIS 2008. IEEE. pp. 1–6. IEEE (2008)
5. Cui, X., Potok, T.E., Palathingal, P.: Document clustering using particle swarm optimization. In: Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE. pp. 185–191. IEEE (2005)
6. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
7. Elhamifar, E., Vidal, R.: Sparse subspace clustering: Algorithm, theory, and applications. IEEE transactions on pattern analysis and machine intelligence 35(11), 2765–2781 (2013)
8. Esmin, A.A., Coelho, R.A., Matwin, S.: A review on particle swarm optimization algorithm and its variants to clustering high-dimensional data. Artificial Intelligence Review 44(1), 23–45 (2015)
9. Gaikwad, K.S., Patwardhan, M.S.: Tweets clustering: Adaptive pso. In: India Conference (INDICON), 2014 Annual IEEE. pp. 1–6. IEEE (2014)
10. Gopalani, S., Arora, R.: Comparing apache spark and map reduce with performance analysis using k-means. International Journal of Computer Applications 113(1) (2015)
11. Gopalani, S., Arora, R.: Comparing apache spark and map reduce with performance analysis using k-means. International Journal of Computer Applications 113(1) (2015)
12. Huang, A.: Similarity measures for text document clustering. In: Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand. pp. 49–56 (2008)
13. Jain, A.K.: Data clustering: 50 years beyond k-means. Pattern recognition letters 31(8), 651–666 (2010)
14. Jiang, J., Ye, H., Lei, X., Meng, H., Wang, L.: Particle swarm optimization via convergence-divergence mechanism. JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE 12(4), 1349–1356 (2015)
15. Jing, L., Ng, M.K., Huang, J.Z.: An entropy weighting k-means algorithm for subspace clustering of high-dimensional sparse data. IEEE Transactions on knowledge and data engineering 19(8) (2007)
16. Kennedy, J.: Particle swarm optimization. In: Encyclopedia of machine learning, pp. 760–766. Springer (2011)
17. Kennedy, J.F., Kennedy, J., Eberhart, R.C., Shi, Y.: Swarm intelligence. Morgan Kaufmann (2001)
18. Lam, Y.K., Tsang, P.W.M., Leung, C.S.: Pso-based k-means clustering with enhanced cluster matching for gene expression data. Neural Computing and Applications 22(7-8), 1349–1355 (2013)
19. Liang, J.J., Qin, A.K., Suganthan, P.N., Baskar, S.: Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE transactions on evolutionary computation 10(3), 281–295 (2006)
20. Lu, Y., Wang, S., Li, S., Zhou, C.: Particle swarm optimizer for variable weighting in clustering high-dimensional data. Machine learning 82(1), 43–70 (2011)
21. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al.: Mllib: Machine learning in apache spark. Journal of Machine Learning Research 17(34), 1–7 (2016)
22. Parsons, L., Haque, E., Liu, H.: Subspace clustering for high dimensional data: a review. Acm Sigkdd Explorations Newsletter 6(1), 90–105 (2004)
23. Pentreath, N.: Machine Learning with Spark. Packt Publishing Ltd (2015)
24. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization. Swarm intelligence 1(1), 33–57 (2007)
25. Sun, K., Wei, X., Jia, G., Wang, R., Li, R.: Large-scale artificial neural network: Mapreduce-based deep learning. arXiv preprint arXiv:1510.02709 (2015)
26. You, C., Vidal, R.: Sparse subspace clustering by orthogonal matching pursuit. CoRR abs/1507.01238 (2015), http://arxiv.org/abs/1507.01238