Big Data Processing – ECS765

Clustering of Large Datasets

# Table of Contents

Table of Contents 2

Use Case 3

Literature Review 3

General 3

K-Means 3

Implementation of K-Means 5

Results & Experimentation 5

# Use Case

This work is intended to investigate and identify the different ways that an unstructured dataset can be split to identify clusters within it. Data from Stack Overflow containing user details, posts, comments and a variety of other fields split across multiple files is being used.

This investigation (using K-Means) is to identify how different features of Stack Overflow users can be used to find correlations between these features and involvement with website.

We will primarily focus on the users.txt file of the dataset. This contains data on users’ reputation rating, number of ‘up votes’ and ‘down votes’ a user has received for their contributions, the date they joined the website, the date they last joined the website and so on. Clustering based on these features, we expect to find meaningful clusters within the data.

# Literature Review

## General

Big Data is usually defined by three characteristics, the three Vs - Volume, Velocity and Variety. (Big Data Clustering: Algorithms and Challenges, Zerhari, Lahcen, Mouline, 2015) and further identified as data which is difficult to capture and process in a normal environment.

A recurring theme throughout all of the literature is that single machine processing of big data is unrealistic due to the size of the dataset resulting in an unacceptable execution time of the clustering task.

Traditional clustering algorithms were mainly designed to run on considerably smaller amounts of data and on one machine (i.e. not designed to be parallelised), and most cannot be modified to run in a parallel fashion. One of the challenges with these algorithms in general is that the amount of memory required to hold the appropriate data in memory may not sufficient. The algorithms cost may be non-linear such that for larger datasets the computation cost makes the algorithm unusable in a bit data situation (Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining: Strehl, 2002).

Network traffic is another significant challenge in the Hadoop environment as data is replicated between machines in the cluster requiring more disk space and bandwidth in the cluster overall. The network transfer of the files from the master node is also a consideration. Due to the way Hadoop is implemented the HDFS file system is not used as effectively as it could be due to the way that files are accessed from the individual nodes in the system (The Hadoop Distributed Filesystem: Balancing Portability and Performance: Shafer, Rixner, Cox, 2010).

There are different types of clustering applications, some types more effective than others for running across clustered machines. These are identified as partitioned, hierarchal, and density based. K-Means is an example of a partition based clustering algorithm. Hierarchal algorithms work recursively to find nested clusters, whereas partition based clustering divides data and works iteratively until convergence (A Survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis: Fahad et. al, 2014).

Newer algorithms specifically designed for use on a cluster of machines include MapReduce which was invented by Google (MapReduce: Simplified Data Processing on Large Clusters: Dean, Ghemawat, 2004) and is implemented in Apache Hadoop. The algorithm takes data into multiple map nodes which create key-value pairs for data input, and then the reducer tasks take the output (the key-value pairs) and combine the results for each key, resulting in the final output of the task.

## K-Means

The K-Means clustering algorithm (MacQueen, 1967) is one of the simplest algorithms to solve the clustering problem and is still one of the most popular clustering techniques today (Top 10 algorithms in data mining: Wu et. al. 2008).

The number of clusters to be identified (k) is defined by the user and used by the algorithm to select k random points in the dataset. The appropriate value of k clusters is generally found by some level of trial and error. These random points become the centres of each of the clusters (k). As these points are random the output on every execution will be different, resulting in differing levels of accuracy per execution.

Once the centres of the clusters are defined the point with the nearest minimised squared difference is added into the cluster, and the centre becomes the mean for the next iteration of the algorithm to identify the next nearest point. This iterative process continues until either a fixed number of iterations (defined by the user) have completed, or iterations continue until there are no changes between continuous iterations.

The formal definition of the K-Means algorithm (E211 Lecture Notes, Ioannis Patras, QMUL) is as follows:



Traditional K-Means runs as a single-threaded operation, but the algorithm is “embarrassingly parallel” (Parallel Implementation of K-Means on Multi-Core Processors, Fahim, 2014). The paper describes the minor modifications of the traditional algorithm to execute in parallel which has been implemented in this work, further details are described in the implementation section below.

Traditional K-Means also requires all of the data being accessible from memory (or swap) which can be a limiting factor in Big Data applications. In memory constrained applications the K-means algorithm can be scaled to handle large datasets by using sequential file access in a process that calculates K-Means each time it reads data from the file and continues until all of the data has been processed (Fast and Accurate k-means for Large Datasets: Shindler, Wong, Meyerson, 2011).

It is possible to better select the cluster centroids, therefore improving the accuracy of the output. This is through is an improved algorithm called K-means++ (k-means++: The Advantages of Careful Seeding, Arthur, Vassilvitskii, 2007). k-means++ has not been implemented as part of this work but could be relatively easily added to the clustering code. With improved centroid positioning it has been observed that in some scenarios that the speed of the actual clustering process can be reduced by up to 50%.

# Implementation of K-Means

Data processing has been carried out using Scala and Spark by writing a generic K-Means algorithm in Scala. The algorithm has been written in a way that it can take arbitrary multi- dimensional input removing the need to rewrite the code to allow for maximum flexibility when investigating different combinations of features.

In order to take advantage of the parallel resources available, modifications to the k-means algorithm were proposed in order to take advantage of the resources available. It was determined that the most significant processing task in the k-means algorithm would be the calculation of distances for each data point from the centroid. Intuitively, a method was proposed to parallelise this directly. The calculation of distances could be split into tasks, with the distances returned to the master node. However, this approach would require large quantities of data to be transferred across the network in order to return all distance results to the master which would present a bottleneck.

Further research revealed that the distribution of a subset of the data to each node and calculation of the k-means for each subset, may provide more efficient processing of data by reducing network activity. This is achieved by partitioning the dataset and running the clustering on the different nodes separately. According to (Parallel K–Means Algorithm on Distributed Memory Multiprocessors, Joshi, 2003) the first machine processes the initial centroids and sends these to the distributed machines. Each distributed machine is therefore only required to work on that specific subset of the dataset rather than the full set. The master node calculates the mean values, and partitions the dataset again.

Our program is broken down into three scala classes which do the following:

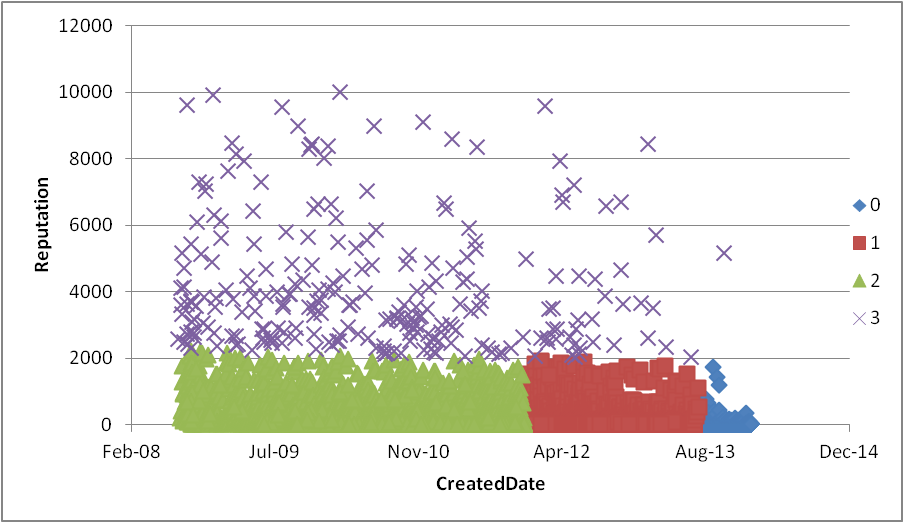
Main – Initialises a spark.context object and a spark.sql.context object with which the rest of the program works. Calls functions within the XMLParser class and KMeans class in order to process the data and output results.

XMLParser – Takes the relevant text files as input and, based on a given XML schema, reads the text files as XML and produces a spark.sql.DataFrame object.

KMeans – Takes the Data Frame, and a given number of iterations, and performs clustering based on the features defined by the user on line 15 (selecting relevant columns from the Data Frame). Cluster centres are initialised as points randomly chosen from the data (line 21). The clustering results are outputted to a text file on the Hadoop File System in the form of a list of pairs containing, for each data point, the cluster index (0 to K) and the values of the features. The centre vectors of each cluster, and the count of points assigned to each cluster, are printed to the console at each iteration, allowing the user to observe convergence.

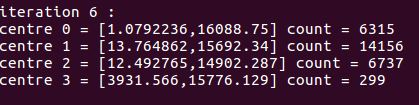
# Results & Experimentation

To prove the program works, a two dimensional clustering was performed on a small subset of the data and the results plotted in Excel. In this case K was set to 4, and the features included were Reputation and CreatedDate. The results of this are presented below:



CreatedDate is converted by our program from a timestamp string in the text file, into an integer representing days since 01/01/1970. This has been reconverted into a date for visualisation purposes here. Users with reputation greater than 10000 have been excluded for clarity.

The output to the command line shows that this clustering converged after 7 iterations:



We see that 4 clusters emerged here:

0 contains the newest users in the dataset who have low reputation as expected.

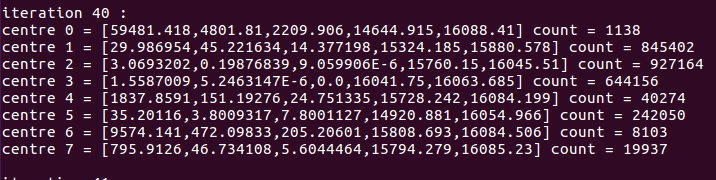
1 contains users who joined in the period of around Jan-12 to Aug-13 with low reputation.

2 contains older users, also with low reputation.

3 appears to contain all the users who have reputation greater than around 2000 (as the command line output shows, they have a mean reputation of 3931.6). There are 299 users in this cluster.

Having shown that the program successfully performs K-means clustering, it is then possible to run the program with various different features and values of K.

By including “Reputation”, “UpVotes”, “DownVotes”, “CreationDate” and “LastAccessDate” and setting K to 8, we obtain the following results (largely stabilised after around 40 iterations):



Rounding the mean values at each centre, and converting the dates into a meaningful form, we see that 8 clusters are found with the following centres and counts:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Cluster | Reputation | UpVotes | DownVotes | CreationDate | LastAccessDate | Count |
| 0 | 59481 | 418 | 2210 | 05/02/2010 | 18/01/2014 | 1,138 |
| 1 | 30 | 45 | 14 | 16/12/2011 | 25/06/2013 | 845,402 |
| 2 | 3301 | 0 | 9 | 24/02/2013 | 07/12/2013 | 927,164 |
| 3 | 2 | 5 | 0 | 03/12/2013 | 25/12/2013 | 644,156 |
| 4 | 1838 | 151 | 25 | 23/01/2013 | 14/01/2014 | 40,274 |
| 5 | 35 | 3 | 8 | 08/11/2010 | 16/12/2013 | 242,050 |
| 6 | 9574 | 472 | 205 | 14/04/2013 | 15/01/2014 | 8,103 |
| 7 | 796 | 46 | 6 | 30/03/2013 | 15/01/2014 | 19,937 |

Given the dimensionality it is difficult to plot the results to see the spread of the clusters. It would be possible to build in a calculation of the spread such as the covariance for each cluster, but this is not implemented here.