Clustering big data

Dan Filimon

June 9, 2013

Contents

1 Rationale

Telescopes surveying the Universe in search of exoplanets, DNA sequencers decoding our genes, governments monitoring cities, and people all over the world exchanging messages, images, videos produce across the Internet produce exabytes of data every day.

Researchers, businesses and governments all over the world are trying to make sense of the ever-increasing collection of data they have access to.

It needs to be stored, searched through, processed and analyzed to extract valuable information. Such a large volume of data comes with its own set of challenges and in the past few years, novel programming paradigms have emerged that make analyzing this data a lot more accessible. There are many systems that are designed to handle "big data". Out of these, the most famous one is MapReduce [**?**] from Google, for batch processing and more recently Storm [**?**] originally from Backtype, now acquired by Twitter.

Many applications that have previously been impossible are now a reality thanks to massive data sets. For example, quality statistical machine Translation can only work at very large scale. While the accuracy of currently available commerical fails to capture more subtle nuances, it is certainly good enough to convey the general meaning.

Large scale data analysis is a broad field with many overlapping disciplines like machine learning, data mining, data science etc.

Analyzing big data is challening. The complexity of many algorithms makes them difficult to parallelize and even when this is possible, because of the scale of the data, clusters of many machines are needed to process the data in reasonable time.

The mainframe era of the 80s, displaced by the affordable personal computer is making a comeback under the guise of "cloud computing" precisely because the complexity of maintaining a large enough cluster of machines to process "big data" makes it a ripe target for outsourcing.

Companies now offer "clouds", clusters of virtual machines that support MapReduce style processing through open-source Hadoop distributions like Amazon (Amazon Web Services), Google (Google Compute Engine), Microsoft (Windows Azure) have vast data centers at their disposal and offer infrastructure as a service to be billed by the hour (or minute).

Time is quite literally money in this market which makes fast processing even more critical. When processing data at terrabye scale and beyond especially in pipelines, the first priority is usually to reduce the data through some aggregation such that the resulting size is much smaller, for example storing statistics instead of the raw data.

One of the most useful algorithms to analyze data with is clustering. Intuitively the aim of clustering is to find clumps of data that are more similar to one another than the others.

The traditional use of clustering is for revealing interesting patterns in the data — for example, if n good quality clusters do form, this could mean there are n types of users in the system.

It's also especially useful for getting a good representative sample of the data that should behave similarly to the original distribution. This is important because sometimes there are no obvious ways of aggregating the data being processed directly to reduce its size. If the data is clustered however, the resulting clusters' centroids can be used as input for the next stage of processing.

2 Clustering

There are many kinds of clustering which vary significantly in their approach in building the clusters. The kind of clustering we'll be discussing in this paper is "centroid" based.

To apply any clustering, or machine learning for that matter, the data needs to be transformed from its raw representation to "feature vectors". The corresponding feature vector for a data item should capture plausibly relevant characteristics of the item in a numeric form. For example, when working with text documents, feature vectors containing frequencies a given word in a document compared to the corpus are useful.

This paper doesn't attempt to further describe the various ways used to encode data into feature vectors or assess the usefulness of a subset of features. This is requires domain-specific knowledge and is the goal of feature engineering. We will assume the data is already available as feature vectors and will now formalize the clustering problem.

We are given n vectors (also referred to as a points) in $\mathbf{R}^d$ and an integer k, and a distance measure, $dist$. We aim to group the n points into k disjoint sets X_1 through X_k. The mean of the points in cluster i (or, disjoint set X_i) is called the "centroid" of that cluster, which we call c_i.

We want our clustering to be good in some sense, so we must define a measure of quality to optimize for. For this, we need a distance measure, $dist$, making the combined vector space and measure a metric space.

We use T_c, the total cost of the clustering, which is the sum of the distances from each point to the centroid of the cluter it is assigned to.

$$T_c = \sum_{i=1}^{k} \left(\sum_{\mathbf{x_{ij}} \in \mathbf{X_i}} dist(\mathbf{x_{ij}}, \mathbf{c_i}) \right) \tag{1}$$

In this formulation, the number of clusters, k is fixed, but this problem is related to the facility placement problem, which essentially identical except in that the number of cluster can vary.

The main issue with this formulation is that the optimization problem we're trying to solve is NP-hard in the general case. This has the very important implication that it is infeasible to find an optimal solution to this problem. Any polynomial-time algorithm we can devise will at best be an approximation scheme. This turns out to not be as dire as it first sounds, because "good enough" is fine for virtually all applications and also because real data tends to have additional properties that results in stronger quality guarantees.

2.0.1 Distance measure

One difference users used to clustering literature might have noticed in this formulation is that our choice of distance measure is not fixed. Normally most papers on clustering work with $||\mathbf{x}-\mathbf{y}||_2^2$, the squared Euclidean distance (or squared $L2$-norm). We acknowledge this and indeed some results we use for our implementations have this assumption. As implementors of a machine learning library however, we need to support user extensibility and we feel that other than providing documentation to our users, we shouldn't enforce other kinds of restrictions. It turns out that most distances used in practice are variants of this distance measure (for example the $L2$-norm and the cosine distance), in which case the results likely the same. If some completely different distance measure is used, we can't provide any guarantees.

2.1 Quality

When it comes to the quality of the clusters generated by any algorithm, it is often said that "clustering is in the eye of the beholder". In unsupervised learning, which this problem is a part of, there is no known "right answer". Compare this with classification or regression where an example is either correctly clasified or predicted, or it isn't.

Things are not so clear-cut in this case. While the total cost is certainly what we optimize for, multiple measure have been devised over the years that attempt to formalize the degree a clustering is "good". Intuitively, we want clusters that are:

- compact, meaning that the points in a cluster are close together (the intra-cluster distances are small between any two points)
- well separated, meaning that two different clusters will be relatively far apart (the inter-cluster distances are large between any two clusters)

The Dunn Index and Davies-Bouldin Index try to express compactness and separability in one score. These are called "internal" scores because they only at one clustering.

Additionally, it is often useful to compare two different clusterings and to see how similar they are. This is useful especially when comparing different clusteirng algorithms. A widely used score for this is the Adjusted Rand Index based off the "confusion matrix" (same idea as with classification).

2.1.1 Dunn Index

$$D = 0 \tag{2}$$

2.1.2 Davies-Bouldin Index

$$DB = 0 \tag{3}$$

2.1.3 Adjusted Rand Index

3 k-means

The most famous, simple and quite venerable clustering algorithm, known since the 50s is k-means and later Lloyd's method.

It first starts by chosing k point out of the n as seeds. These will be the first centroids (they're not "really" centroids since they are not the means of the points). The algorithm then proceeds by doing some number of iterations of the following steps:

1. assign each point to the cluster whose centroid is closest to it
2. recompute the existing centroids

Multiple iterations are performed until the maximum number of iterations is reached, until the quality plateaus or until no point changes cluster during the assignment phase.

Figure 1: These are some example points we'll cluster in 3 groups

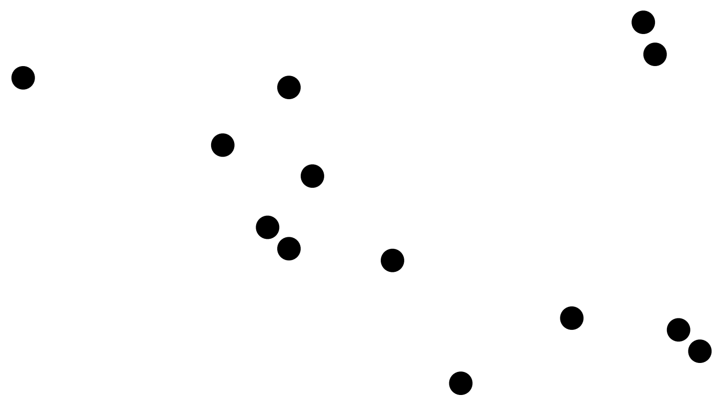


Figure 2: Here we selected 3 points as the initial centroids

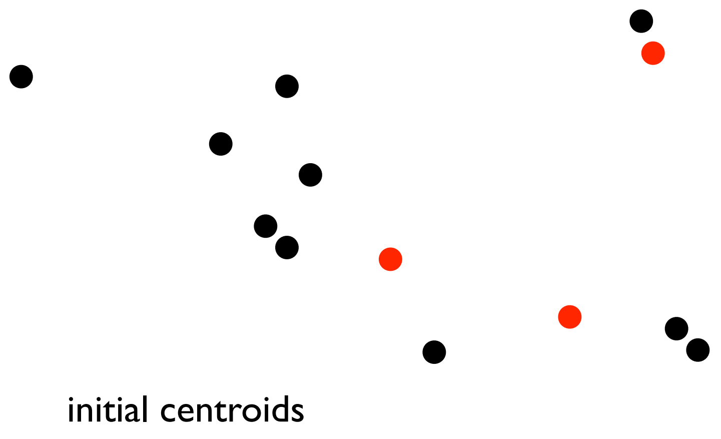


Figure 3: Here we assign each point to the cluster whose centroid it's closest to

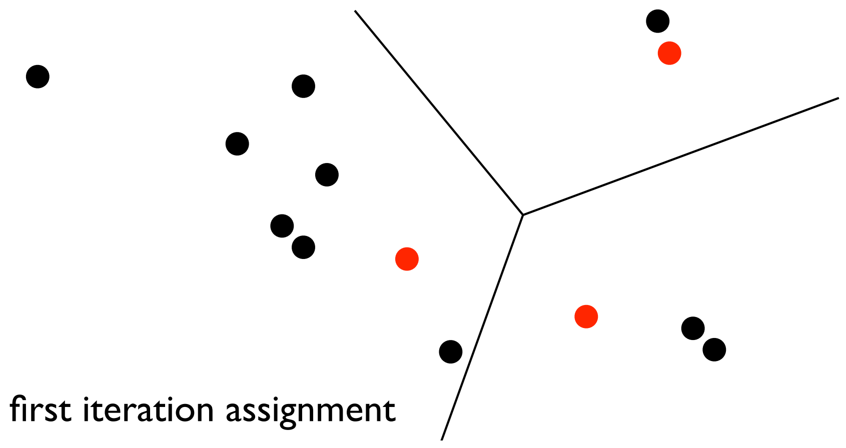


Figure 4: After asigning each point to a cluster, we compute the centroids of those clusters as the mean of the points in that cluster

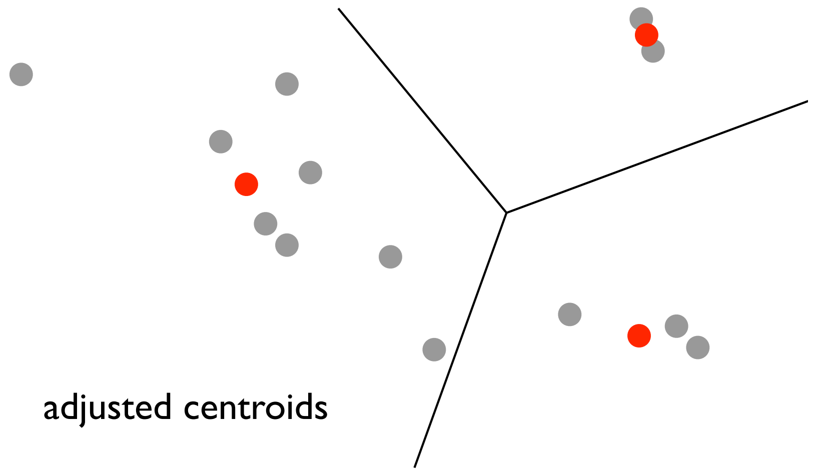


Figure 5: Second iteration assignment

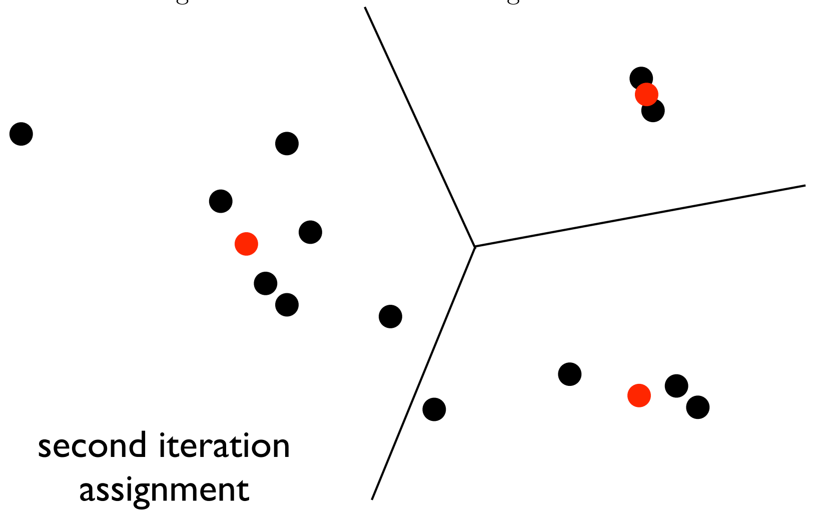
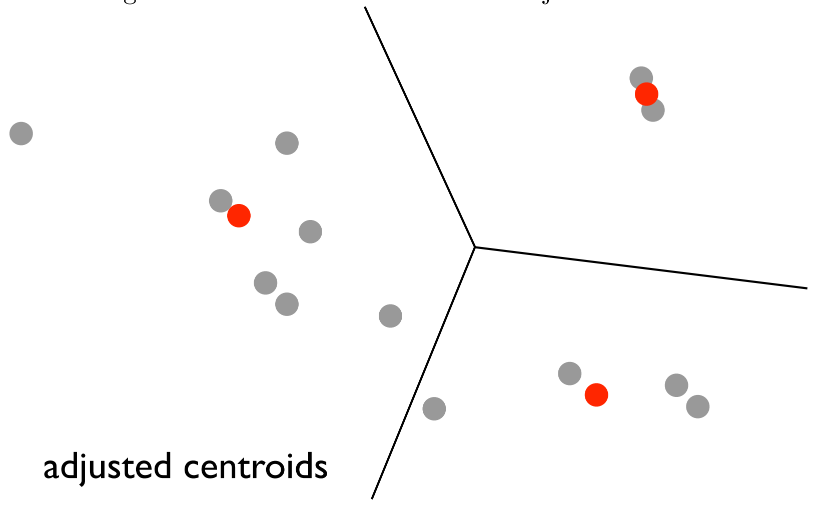


Figure 6: Second iteration centroid adjustment



3.1 Algorithm

The Python-style pseudocode for the algorithm described above is:

```
centroids = select k points from n
while not done:
    for p in points:
        cmin = None
        for c in centroids:
            d = dist(p, c)
            if d < dmin:
                dmin = d
                cmin = c
```

The crucial (unexplained) details are:

1. centroid initialization: how are the k points selected to become centroids at the beginning?

2. stopping condition: how long do we need to do the point assignment and centroid adjustment?

These two questions have a variety of solutions each of them resulting in a slightly different version of the algorithm, but with mostly similary results.

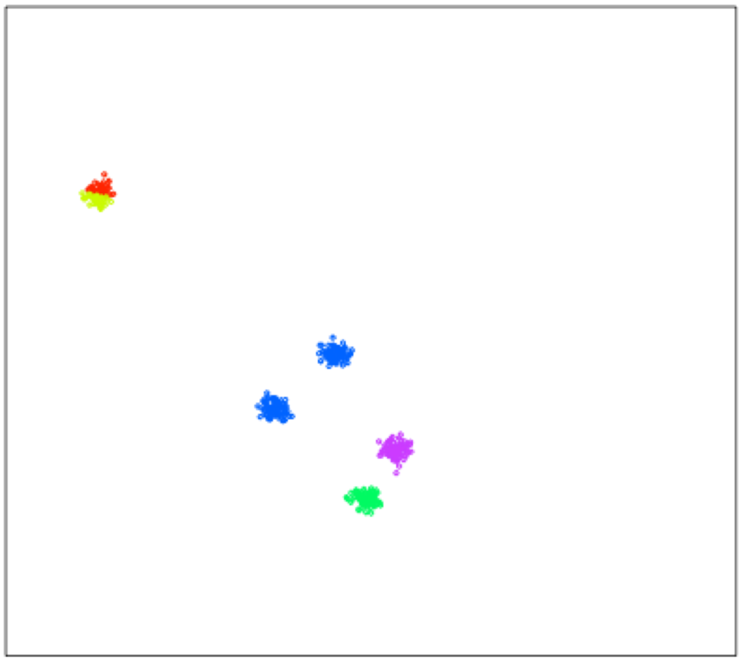
3.2 Initialization

Selecting the first k points to become centroids, or "seeding" as it is also known is the single most important factor in getting a high-quality clustering.

3.2.1 Random selection

The simplest solution is to simply select k points of the n randomly. This produces a seed but has no real guarantees about what these points are. For a good quality clustering, assuming there are k real clusters and we wish to identify them, the seeds should ideally come from the k clusters and no two seeds should come from the same clusters. If this is true, because we assumed the clusters really do exist in the data, the cluster assignment will set nearly all the points in the right cluster in just a few steps. The downside of this approach is illustrated in the figure below. There, two seeds are selected from the same real cluster, causing it to be split among the two k-means clusters (red and yellow). This means, another cluster (the blue one) will have to contain points from two real clusters.

Figure 7: Here is a case where the random initialization failed and a single real cluster was split between the yellow and red k-means clusters



Additionally, like many machine learning algorithms that only result in locally optimal solutions, k-means benefits from multiple restarts. Multiple restarts are even more important as this makes it nearly impossible to not have a good set of initial centroids.

3.2.2 k-means++

There are ways of improving the selection of centroids through the intuitive idea that we want to have seeds that are as far apart to each other as possible. This effectively eliminates the problem of having two seeds in the same cluster. While there are multiple ways of doing this, the most widely used one probably being [**?**]. The approach I implemented is described in [**?**] and is summarized below. In [**?**], Ostrovsky et. all introduce a new condition, called ϵ-*separability* that formalizes what data should look like for a good clustering to exist and give an new way of sampling the initial centroids with provably-good guarantees. This sampling is fairly similar to k-means++ and they show how it can produce a constant-factor approximation of the optimal clustering with just one Lloyd step provided the data is ϵ-separable.

From an implementor's perspective, checking if the data is indeed ϵ-separable is of little interest. We would need to know the optimum clustering cost for k clusters and $k-1$ clusters and if we could compute those, we'd just do it directly. The new sampling method however, is useful and we describe it below (based on section 4.1.1 in [**?**]).

First, we select two centroids, c_1 and c_2 with probability proportional to $dist(c_1, c_2)$. [**?**] uses $dist(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||_2^2$ as the distance measure, but as explained above, we generalize this in the implemetation to any user supplied function. We now have the first 2 centroids.

Then, to select the $(i+1)$-th centroid, call the i already selected centers, $\mathbf{c}_1 \dots \mathbf{c}_i$. The new point is selected randomly with probability proportional to $\min_{j \in \{1 \dots i\}} dist(\mathbf{x}, \mathbf{c}_j)$.

3.2.3 Convergence

3.2.4 Outliers

3.2.5 Nearest neighbor search

3.3 Ball k-means

4 Large scale

5 Batch processing: MapReduce

6 Online processing: Storm

7 k-means as a MapReduce

7.1 Algorithm

7.2 Discussion

7.2.1 Multiple passes through the data

7.2.2 Random initialization

8 Making big data small

9 Streaming k-means

9.1 Algorithm

9.2 Discussion

10 Clustering big data: MapReduce

10.1 Mahout implementation

11 Clustering big data: Storm

11.1 Storm prototype

12 Results

12.1 Quality

12.1.1 Data sets

12.1.2 Quality measures

12.1.3 Experiment

12.1.4 Discussion

12.2 Speed

12.2.1 Data sets

12.2.2 Cluster

12.2.3 Experiment

12.2.4 Discussion