K-Means and Gaussian Mixture Models

Marvin Cheng
Department of Computer Science
University of Washington
Seattle, Washington 98195

Peter Dwersteg
Department of Computer Science
University of Washington
Seattle, Washington 98195

Tobias Kahan
Department of Computer Science
University of Washington
Seattle, Washington 98195

John MacKinnon
Department of Computer Science
University of Washington
Seattle, Washington 98195

June 4, 2012

Abstract

In this paper we examine two steps in the clustering process: determining the number of clusters and assigning data points to those clusters. For the first step – determining the number of clusters – we use k-means, a nonparametric clustering algorithm using euclidean distance. In the second step we use a Gaussian Mixture Model to assign the data to clusters. The Gaussian Mixture Model approach utilizes soft assignments, so that each data point has a weighted assignment to each cluster. Finally we examine the utility of our algorithm on different types of data – both synthetic and real.

While our motivation for exploring clustering algorithms stemmed from an interest in their applications, our project ended up being largely one of theoretical merit. That said, in deciding upon the final format for our clustering process we did perform some experimentation with both k-means and EM in order to explore some of the more salient features of the two algorithms. All of our findings were largely consistent with those of the literature reviewed prior to implementation; that both algorithms were largely comparable in both speed and

effectiveness in the majority of data sets, with one notable exception: in data where clusters were poorly defined (i.e. data with lots of overlap between clusters). The hard-clustering of k-means was not able to effectively convey this overlap. EM on the other hand performs soft, probabilistic assignment of each data point to each cluster. Of course the ultimate goal of any clustering algorithm is to assign data points to sources, but EM at least provides us with some metric of measuring the certainty of our assignments. For this reason we chose EM as the ultimate cluster assignment when working with real data.

1 Background

Although Machine Learning is not a new field, advances in computing power have unlocked the ability to analyze mountains of data relatively quickly. Further advancing the field, the Internet has given businesses the ability to collect valuable data at an unprecedented rate. The canonical example is Google which uses algorithms that learn a users preferences and then display search results and advertisements tailored to that user. Another example is in the field of Computational Biology, where researchers use learning algorithms to find relationships and structure among DNA, RNA, and other proteins. These two cases embody the rapid expansion of Machine Learning.

The recent growth of Machine Learning has led to a variety of learning algorithms. Two important subsets of Machine Learning algorithms are supervised learners and unsupervised learners. Supervised learners generally necessitate that a human classify data as belonging to one group or another. An example is a learning algorithm that classifies images of facial expressions as smiling or frowning based on the properties of the image. This algorithm first needs a human to classify a set of images as either smiling or frowning. Then, the algorithm can classify additional images as smiling or frowning based on what it learned from the training data, i.e. the images which the human classified.

Unsupervised learning – the subject of this paper – consists of algorithms which seek to find some relationship or structure in data without knowing the true nature of the data. An example is a learning algorithm that groups text as coming from the same author. This algorithm has no prior knowledge

about which authors produce which kinds of text, so it is limited to grouping the text based on similar features.[1]

A major subset of unsupervised learning is clustering algorithms. Clustering algorithms attempt to group data based on natural structure. The output of a clustering algorithm is a set of clusters to which data points are assigned. These assignments can be hard, each data point is assigned to only one cluster, or soft, each data point has a weighted assignment to each cluster.

2 Methodology

The assignment of data to clusters is a form parameter estimation: the properties of each cluster are estimated from the data. In general, when estimating a parameter from data, we maximize the likelihood of the missing parameter through the process of Maximum Likelihood Estimation (MLE). MLE is a process in which the likelihood function of a parameter is differentiated with respect to the missing parameter, set equal to 0, and solved for that parameter[3]. However, this is not possible in many problems, particularly those in which maximizing the likelihood function is analytically intractable. Thus, more sophisticated techniques must be used in order to estimate the missing parameters. In the field of clustering, k-means and Expectation Maximization (EM) are two such techniques.

2.1 K-means

K-Means is the simpler of the two algorithms explored; it assumes no underlying distribution, and uses hard assignment based upon euclidean distance of data points from each mean. The goal of the algorithm is to minimize the within-cluster sum of squares, or in other words, to find means and assignments for our clusters such that the sum of the square distance for each point to the mean of the cluster it is assigned is minimized. Analytically:

$$\arg\min_{S} \sum_{j=1}^{k} \sum_{x_i \in j} ||x_i - \mu_j||^2$$

where x_i represents data point i, and μ_j represents the mean of cluster j

(it should be noted that x_i has been assigned to cluster j, otherwise it does not contribute to the within-cluster sum of squares for j).

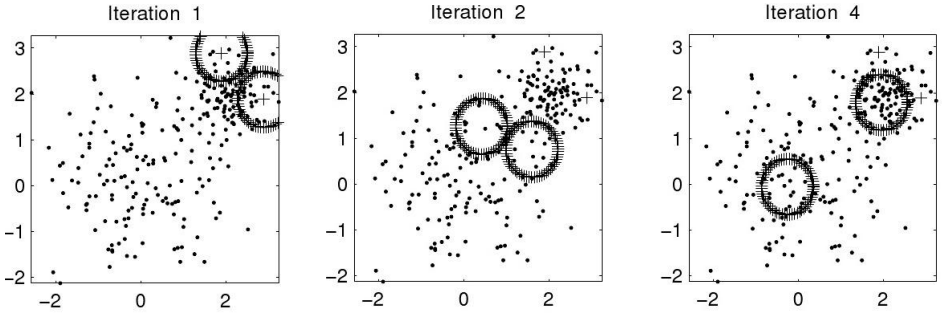
The algorithm begins under the assumption of k underlying clusters to be found. K random values are then chosen from the data to act as initial estimates of the means for each cluster. For example, if the attributes of the data set are height and weight 2-tuples, k random height-weight pairs would be chosen from the range of all height-weight items in the data. With our means for each cluster assumed, we now begin the assignment phase: each data point is assigned to the cluster center that is closest to it via euclidean distance. That is, for each cluster S_j is defined such that:

$$S_j = \left\{ x_i \mid ||x_i - \mu_j||^2 \leq ||x_i - \mu_m||^2 \; \forall m \neq j \right\}$$

After every point has been assigned (and our thus our clusters defined), we then re-evaluate each mean. In this step, each of the k means is recalculated as the mean of the data points that are assigned to it. That is:

$$\mu_j = \frac{\sum_{x_i \in S_j} x_i}{|S_j|}$$

This process of cluster assignment and cluster-mean re-estimation is iterated until the means remain unchanged from one iteration to the next, indicating that the position of the k means as been determined and that each data point as been assigned to its proper cluster. A nice property found through both our research and our implementation is that K-Means will converge for almost every real set of data. A visual example of k-means running is as follows:



Iteration 1: A set of data points, with k = 2 random cluster-centers chosen.

Iteration 2: The cluster centers after the first iteration of assignment and

cluster-center re-estimation. Note that the given centers already match the data significantly better than in step 1.

Iteration 4: The cluster centers after convergence. These are the k = 2 means of the data set that minimize the within-cluster sum of squares.

2.2 EM with Gaussian Mixture Model

In general, EM is an algorithm that assumes the data is from some underlying distribution; the missing parameters from this distribution – the random vector Θ – are estimated by the introduction of another random vector (which we will call Z). Z represents the likelihoods of the data, given the current value of Θ. That is, in the "Expectation Step" (E-step), the algorithm uses the data and the current value of Θ to estimate Z: the likelihood of each data point given our current model. In the "Maximization Step" (M-step), we use the data and Z to compute an MLE for Θ. The E and M steps are iterated – where (under very mild assumptions) each of these iterations is guaranteed to increase the log likelihood of the overall model. This means the process will eventually converge to a local maximum of the likelihood function; finding a global maximum is a matter of initialization of Θ or Z values, an issue which we did not explore in our analysis due to the general effectiveness of our naive initialization techniques.

For our clustering processes, the EM algorithm was applied to the assumption that an underlying Gaussian Mixture Model (GMM) generated the data. Such mixture models are widely used in practice, as models with sufficient components (or clusters) often empirically match the arbitrary distribution from which the data was drawn. Further, it is often well justified, as the hidden parameters are what drive the data, and it is common for such parameters to be normally distributed when observed in large quantities. For our data, we assumed multi-dimensionality, so our process will be explained in terms of multi-dimension Gaussian distributions.[2]

Given the assumption of an underlying GMM, Θ corresponds to the mean and spread of each Gaussian cluster from which the data was observed. As in any EM, the iteration process may be started by making an initial guess for either Z or Θ; we arbitrarily chose to guess Θ. In this initialization, we picked our initial μ (or mean) values for each cluster by randomly selecting k points from our data. Our σ covariance matrices that governed each cluster's spread were then initialized to the Identity matrix with size equal to the

number of dimensions of our data, times some constant. This constant was determined by calculating the covariance matrix of the entire data set and selecting the first entry; this constant did not affect convergence, but helped in guaranteeing that spread was governed by a value with magnitude on the order of the data; this was necessary in the first iteration of the E-step in order to guarantee non-zero Z values for each data point for at least one of the clusters.

2.2.1 The E-step

For the E-step, we assume the data and our current values of Θ. For each data entry (a vector of observations in the multivariate case), we must calculate the likelihood of the data coming from each cluster. This can be interpreted as $P(C_j|O_i)$ where C_j is the event that data entry x_i was drawn from cluster j. O_i is the event that entry x_i was observed. Using Bayes' rule, we may rewrite this $P(C_j|O_i) = \frac{P(O_i|C_j)*P(C_j)}{P(O_i)}$

This significantly increases the tractability of our calculations, as $P(O_i|C_j)$ may simply be interpreted as the likelihood of drawing point x_i from a Gaussian distribution governed by μ_j and $\sigma_j, P(C_j)$ may be interpreted as prior likelihood for cluster j or prior belief that we might sample from j (initialized to be uniform among all clusters and denoted τ_j), and $P(O_i)$ may be computed similarly using the Law of Total Probability. This step provides all of the soft assignments for each data point in relation to each cluster.

2.2.2 The M-step

In this step, we assume Z – the likelihood assignments calculated by the E-step – as well as the data in order to recalculate Θ. Because we are dealing with a Gaussian Mixture Model, this means re-estimating each μ and σ in order to maximize the likelihood of our model. These values take on forms very similar to what we might expect – that is, the mean is just an average of all points, weighted by their likelihoods, and the covariance matrices for σ are just the covariance of the data, weighted by their likelihoods. Analytically:

$$\mu_j = \frac{\sum_{i=1}^{n} Z_{ij} * x_i}{\sum_{i=1}^{n} Z_{ij}}$$

$$\sigma_j = \frac{\sum_{i=1}^{n} Z_{ij} * (x_i - \mu_j)^T * (x_i - \mu_j)}{\sum_{i=1}^{n} Z_{ij}}$$

We also re-calculated the estimates of the prior probabilities, τ_j for each cluster. This simply comes out to be the average of the likelihood of each data point being drawn from that distribution. That is,

$$\tau_j = \frac{1}{n} * \sum_{i=1}^{n} Z_{ij}$$

2.2.3 Likelihood

One important step in EM is controlling the iterations and determining convergence; for this we rely on the likelihood of the model. The overall likelihood comes from determining the product over all clusters of the marginal likelihood for each cluster. Analytically:

$$L = \prod_{i=1}^{n} \sum_{j=1}^{k} \tau_j * f(x_i|\Theta_j)$$

In practice, the log form of this equation makes finding the likelihood more tractable.

2.3 Our Algorithm for Clustering

For our data sets, we determined the appropriate number of means by iterating K-Means over different values of k. The output of the algorithm for each k was the sum of the distances from each point to its mean. An obvious problem with a naive approach for this methodology comes from fitting clusters to our n data points; k = n means would result in the smallest total distance, but would be useless from a clustering point of view. Thus we added this sum of squares to a tuning factor of $n * log(k)$, where n is the number of observed data points. This tuning factor punished higher values of k, so as to balance out our desire for a good fit, and a reasonable number of means. We regarded the value of k that gave the best result for a data set to be the most appropriate number of means for the data set, which we then applied to EM.

With our k now chosen, we ran EM on our data set to ultimately provide us with our cluster-assignments. Though EM provides soft assignments, we performed hard-assignment by assigning each point to the cluster that corresponded with its highest likelihood (i.e. The largest Z_ij value). Upon

analysis, if data points seemed miss-classified, we were always able to return to the soft likelihoods for assignments in order to evaluate the algorithm's effectiveness.

3 Results

In this section we analyze our algorithm on synthetic data, discuss real data collection, and analyze our algorithm on real data.

3.0.1 Synthetic Data Analysis

We synthetically generated two datasets to observe the behavior of our algorithm. Dataset 1 (see Figure 1) consists of 2 clusters which are well defined and do not overlap; dataset 2 (see Figure 2) is more ambiguous, containing two clusters with much overlap. These two datasets represent different levels of difficulty for a clustering algorithm and datasets should expose the weakness of k-means in clustering overlapping data. Recall that k-means is the first step of our algorithm and as such dataset 2 should throw off our algorithm even though EM should do a good job of clustering dataset 2.

Our algorithm applies k-means iteratively, incrementing the number of means with each iteration. This process is presented by Figure 3 and Figure 4. Notice that k-means does about as bad as it could on dataset 2, putting half of each synthetic cluster in the learned clusters.

We include tables of how our iterated k-means performs on the two datasets. "Within SS" refers to the total sum of squares, within clusters. Total is our optimization function which we want to minimize – it is the sum of Within SS and Penalty.

In the first table – corresponding to dataset 1 – we see that k-means correctly outputs 2 clusters. However in the second table – corresponding to dataset 2 – we see that the optimization function continues to decrease far beyond the true number of means (i.e. 2). This erratic behavior points to the limitations of k-means when applied to overlapping data.

Figure 5 shows EM running on dataset 1. This represents a complete run of our algorithm – our k-means iterator output 2 means and we ran EM on the data with 2 means as the parameter. Figure 6 shows EM running on dataset 2. However, instead of using the output of our k-means iterator, we applied EM with 2 means as the parameter. This is to show how EM

Table 1: Dataset 1

Iteration	Within SS	Penalty	Total
1	961.60875	0.00000	961.6087
2	173.63500	69.31472	**242.9497**
3	141.76697	109.86123	251.6282
4	116.37278	138.62944	255.0022
5	97.31744	160.94379	258.2612
6	81.83388	179.17595	261.0098
7	65.24665	194.59101	259.8377
8	52.07389	207.94415	260.0180
9	46.01228	219.72246	265.7347
10	38.45418	230.25851	268.7127

Table 2: Dataset 2

Iteration	Within SS	Penalty	Total
1	1610.0809	0.00000	1610.0809
2	1011.2075	69.31472	1080.5222
3	726.8875	109.86123	836.7487
4	482.7251	138.62944	621.3545
5	367.9962	160.94379	528.9400
6	350.1264	179.17595	529.3023
7	210.4215	194.59101	405.0125
8	187.4002	207.94415	395.3444
9	137.0132	219.72246	356.7357
10	130.4846	230.25851	360.7431

performs better that k-means on overlapping data.

As the figures show, EM – like k-means – does well on dataset 1. What is interesting is that EM clusters dataset 2 very well, whereas k-means performed horribly on this data.

3.0.2 Real Data Collection

Synthetic data provides an efficient way to test multiple types of data. However, the end goal of Machine Learning is to learn from real phenomenon. Fortunately, the Internet abounds with fast and free real life data to test our algorithm. We used a method called screen scraping – a process of acquiring data from a source using programmatic techniques.

The first attempt at scrapping was UFC fight data. However, the data contained inconsistencies and lacked a complete index of all UFC fighters. We also scraped lotto data from the SOCR site (http://wiki.stat.ucla.edu/socr/index.php). We found that we were able to pull the date and lotto numbers, but running our K means and EM implementation resulted in one cluster, which we found not comprehensive enough for our project.

Ultimately, we decided to use observed height and weight data from SOCR. Specifically the Major League Baseball height and weight, and adolescent height and weight data sets. We purposely chose these two datasets so that we know for sure that we had a good chance of producing two clusters when we run the K means and EM algorithms.

R could not parse the data sets without some work on our end. We used Regular Expressions to parse the raw HTML and retrieve the data that we needed.

Example Regular Expression

```
<tr>\n<td>(.*?)</td><td>(.*?)</td><td>(.*?)</td><td>(.*?)</td>
<td>(.*?)</td><td>(.*?)</td><td>(.*?)</td><td>(.*?)\n</td></tr>
```

The important part of this regular expression is (.*?), which is non-greedy when matching any character. Breaking down the symbol meanings themselves we get the following.

Symbol	Meaning
.	Matches any character
*	0 or more instances
?	0 or 1 instance
()	Grouping

Figure 1: Distinct Clusters

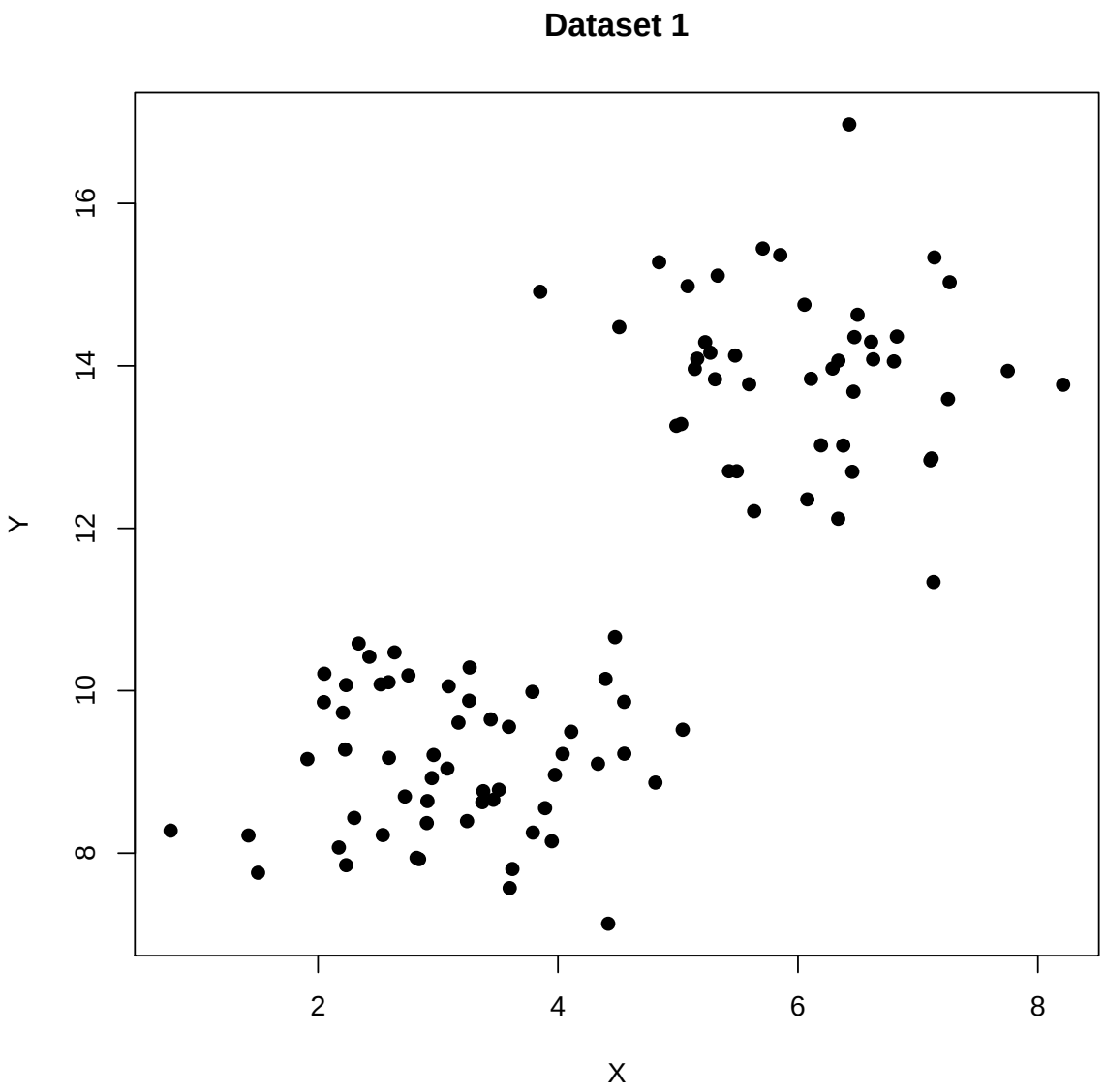


Figure 2: Overlapping Clusters

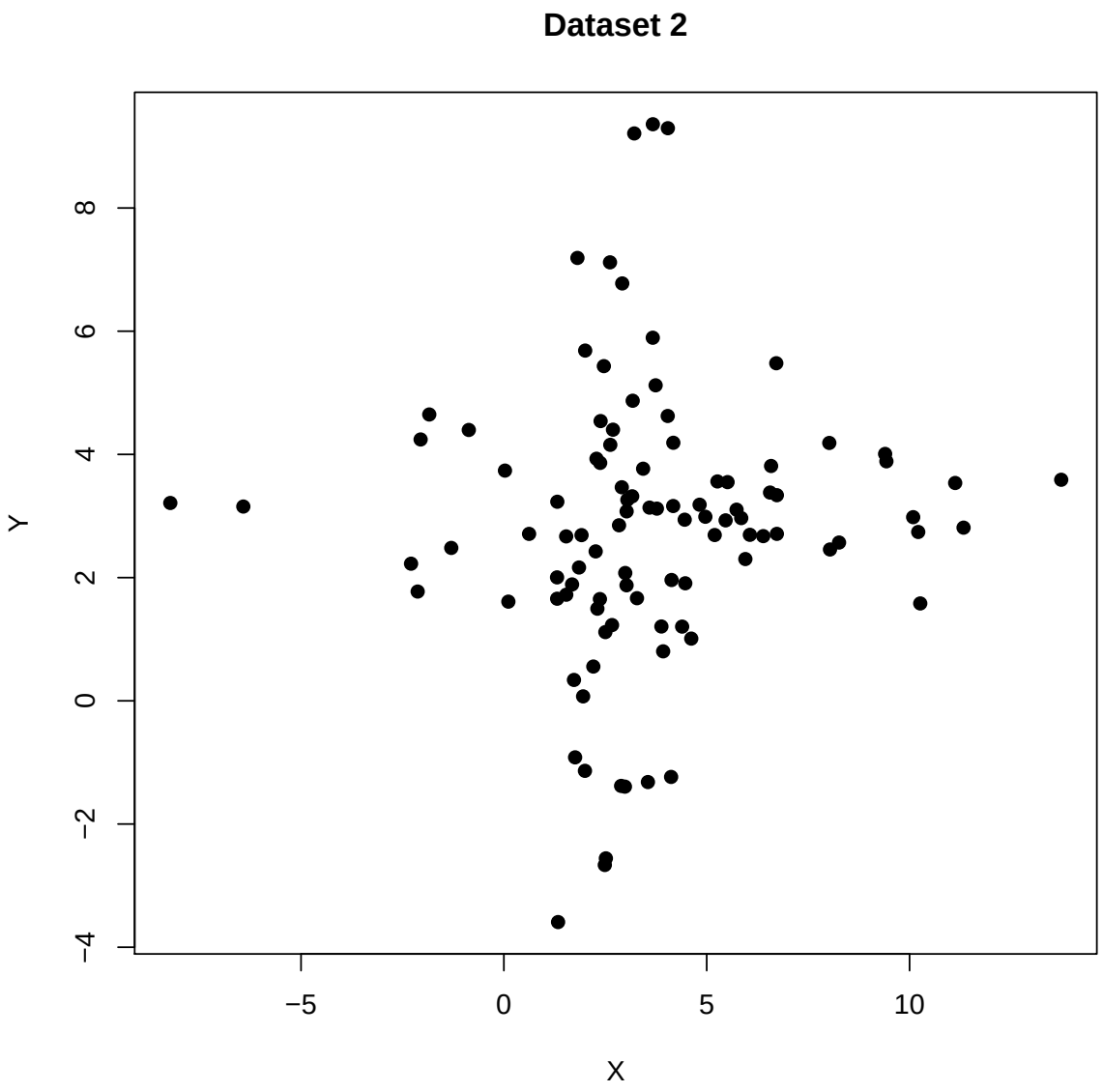


Figure 3: Dataset 1

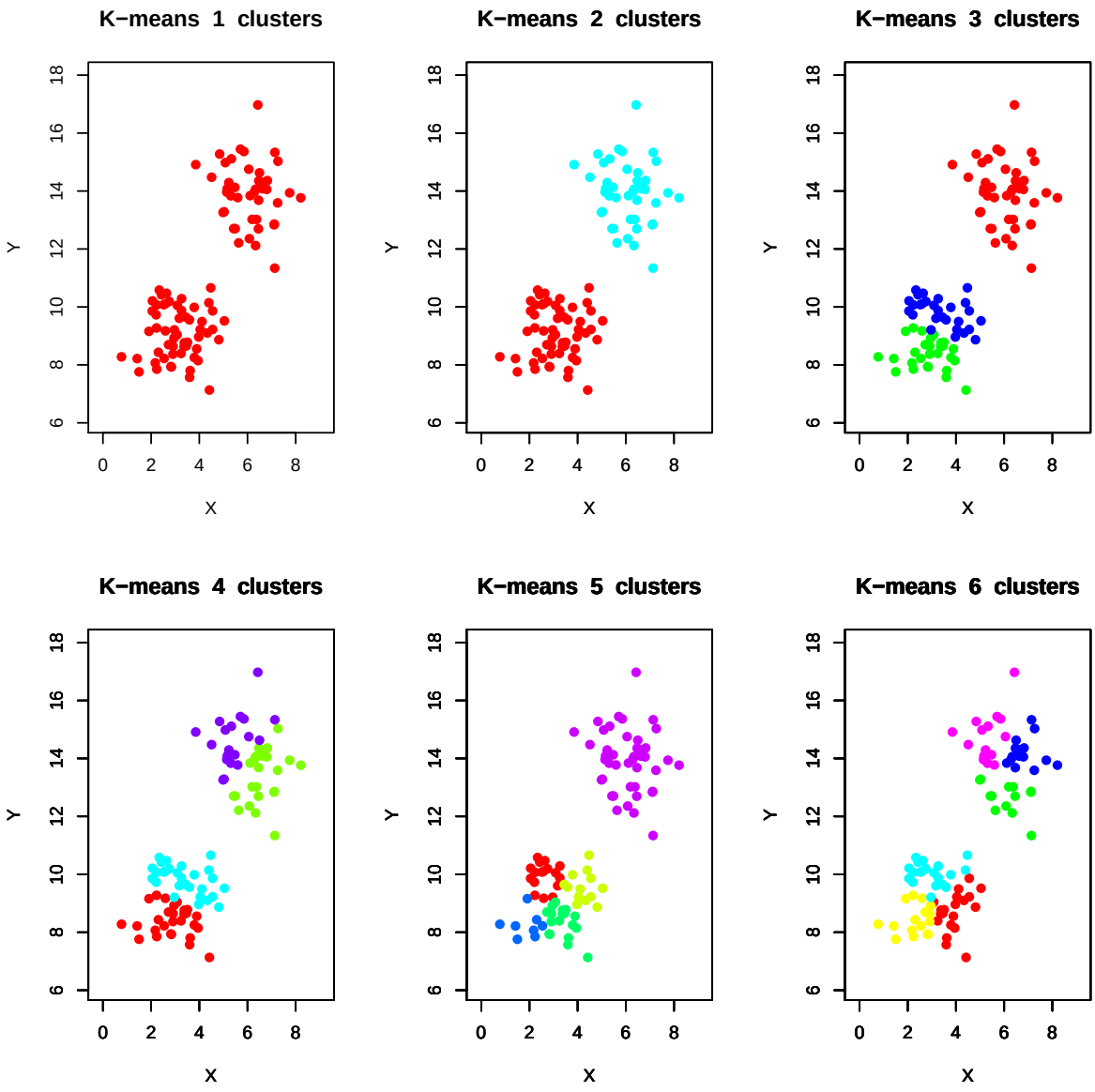


Figure 4: Dataset 2

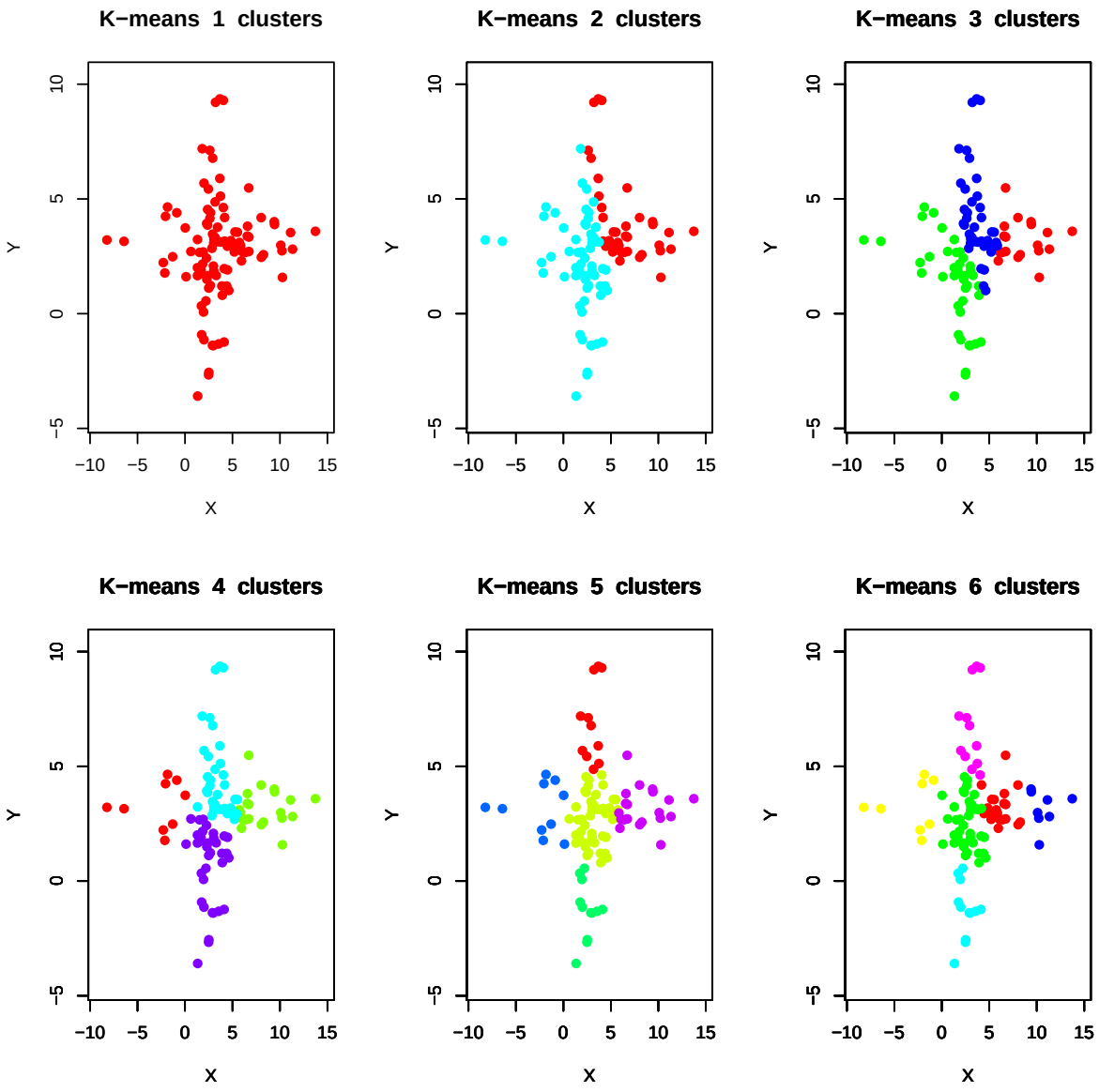


Figure 5: Dataset 2

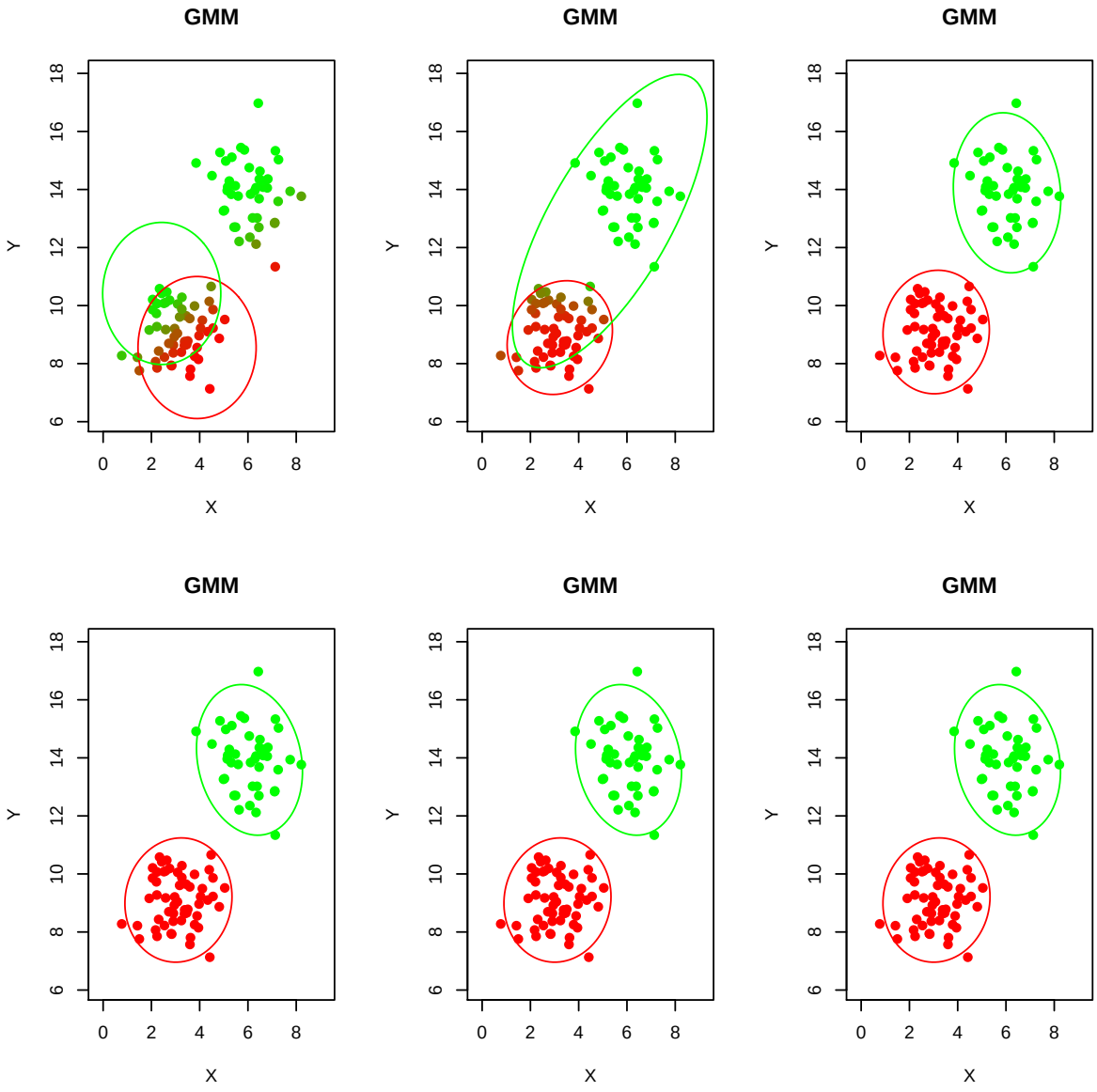
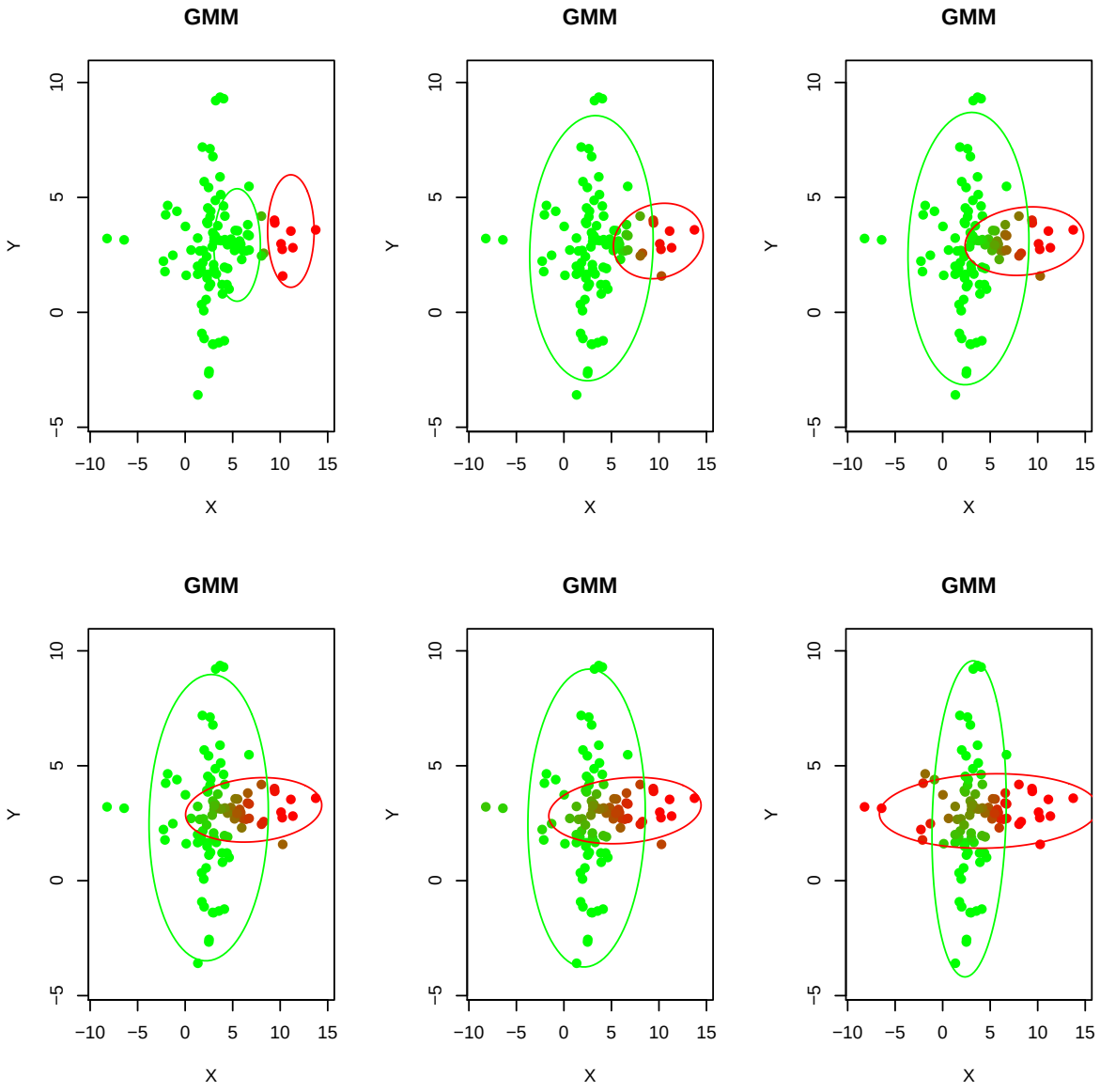


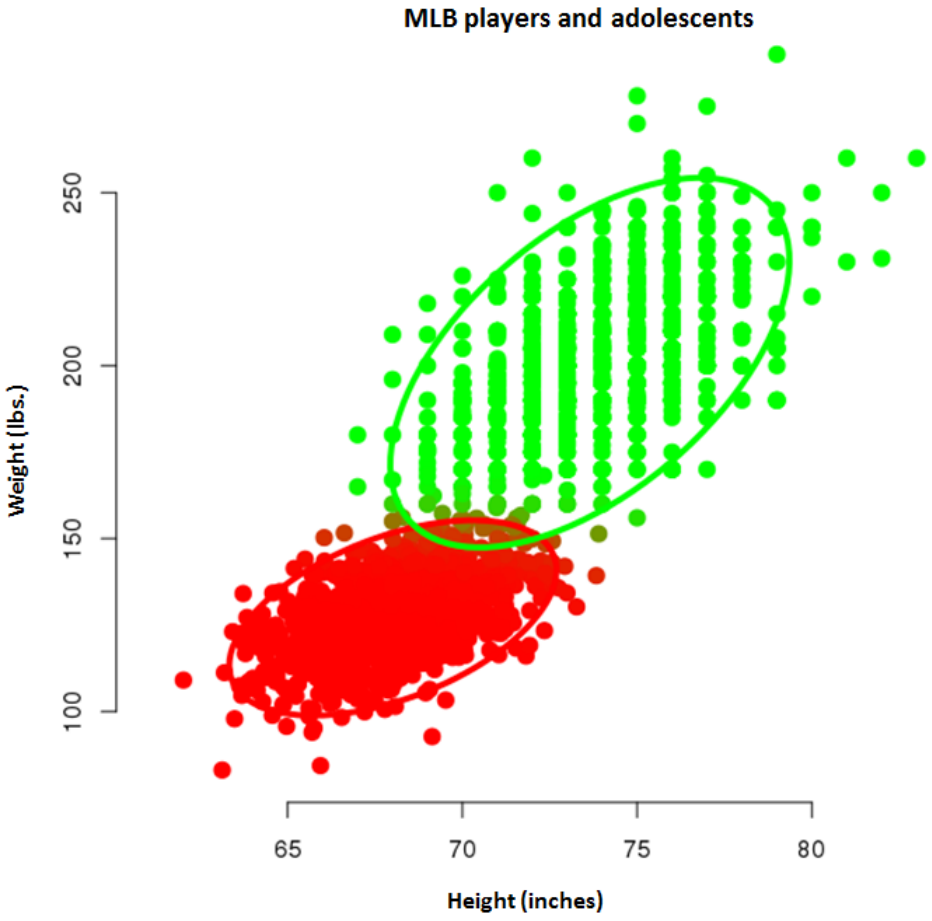
Figure 6: Dataset 2



The complete code for parsing the data from the HTML files can be found in the appendix.

3.0.3 Real Data Analysis

Though we did not take a primarily data-driven approach, we did still test our algorithm on several real-life, or applied data sets (largely to verify that our methods still perform as expected). The applied data actually ended up being 'simpler' than our synthetic data, in that clusters were much more clearly defined. As would be expected, our algorithm performed very well. In the MLB/adolescent data for height-weight pairs, the results were as follows:



There is a very distinct set of two clusters in the data, and the data is successfully partitioned into two very reasonable clusters. It is fairly simple to make out the MLB data by its columned data (due to rounding of heights to integer values in the census), and see that almost without exception, nearly all baseball players were partitioned into the green cluster. We can also see several mis-classified adolescents with particularly large height and weight values (which can be seen as points in the green cluster with heights that do not conform to the rounded values of the columns), but in general this clustering performed extremely well. There are some fringe-assignments that fall between the two clusters, which can be made out as the brownish, or mixed-color points. This is inherent in any clustering where clusters are not perfectly defined, and our EM correctly provided us with likelihoods (soft assignments) indicative of the unsureness of those assignments; this manifested itself as mixed red and green values in the coloring scheme of those data points.

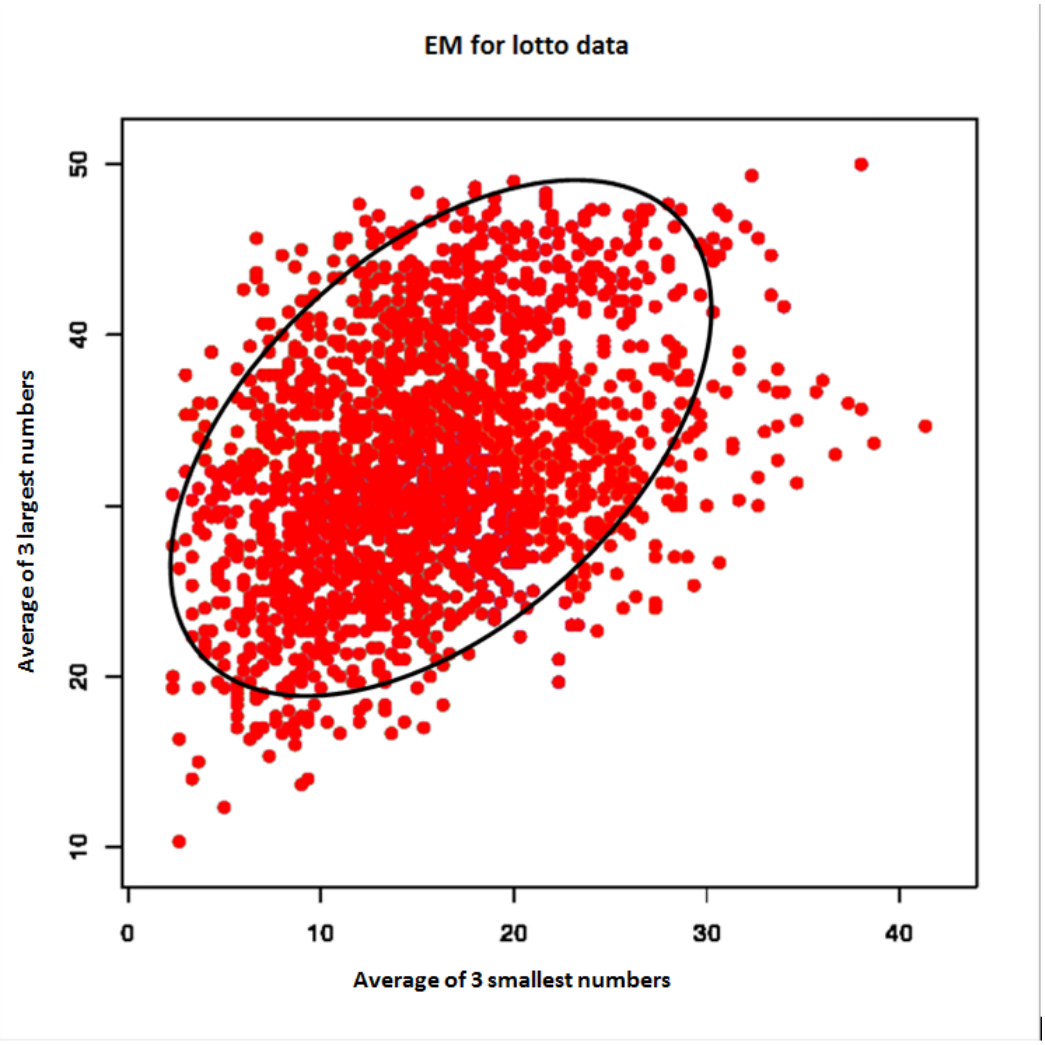
Perhaps even simpler, but worth noting, is our algorithm's performance on the lotto data (see next page).

It is easy to see that a 1-cluster model is a reasonable fit for this data (i.e. There isn't really an underlying mixture of Gaussian distributions driving the data). While a fairly trivial example of clustering, this example was useful in validating that our algorithm could accurately determine the number of clusters within the data.

4 Discussion

Though our research began with the intention of applying clustering to novel data in order to try and uncover underlying factors driving or resulting from the data, the process of implementing our clustering algorithms turned out to be a very enlightening one in itself.

Our findings largely indicated that both EM and K-means are effective clustering heuristics. Both techniques are largely comparable in accuracy, as well as speed (though K-means appears to have had a slight advantage on particularly large data sets most likely due to the relative slowness of computing the marginal log likelihood of the data for EM, compared to the quicker computation of the sum of square distances for K-means). However, there were several distinguishing factors that were discovered between the two as well.



Computationally, there were no apparent weaknesses to EM. While perhaps marginally slower than K-means, EM was still very fast, generally converging in no more than 10-20 iterations. When applied to synthetic data, we found that the means, covariance, and prior sampling likelihoods for our clusters were nearly identical to the models from which we generated the data. Furthermore, EM proved very useful in its ability to provide likelihoods of its clustering assignments, which we interpreted as, more or less, sureness of assignments. This was especially apparent in our synthetic data, in which there was a decent amount of overlap between the two data sets. K-means was unable to produce any metric for measuring this uncertainty, while EM gracefully presented us with a measurable degree of certainty in our

partitioning of the data. Due to their conceptual similarity in determining cluster-membership, but advantage in utility for EM, EM may generally be thought of as a more sophisticated version of K-means. For these reasons, we concluded that if both algorithms are available, EM is the superior heuristic.

Though EM was found to produce 'better' results, we did find that k-means has an advantage in ease of implementation. The algorithm is much simpler, and is very extensively documented, so the entire process of writing the code for K-Means in R took approximately 6 hours including testing and debugging. Most of the difficulties encountered involved determining how to properly implement certain functionality in R, such as vector manipulations in data frames. Implementing of EM was not quite as simple a process. Difficulties arose even before jumping into code it is clear even from the definition of EM that the algorithm involves much more complex mathematics. This further translates into more opportunity for bugs to slip in the code. For example, if covariance matrices are initialized to the Identity matrix (as we originally attempted), initial calculation of likelihoods may result in 0-values for each cluster, because the magnitude of the data far exceeded the spread allowed by using the Identity matrix for the covariance of a cluster; these translated to NaN values in the Z_{ij} matrix once each row was normalized. This particular bug was fixed by initializing each covariance matrix to the covariance of the overall data (thereby putting it on the correct 'scale'), though many more subtle bugs presented themselves throughout the implementation process. The ease with which K-Means can be implemented in comparison to EM gives it the advantage when the data set to be clustered is simple and absolutely perfect results or soft clustering are not required. It is for this reason that, if we were to work on an project for applied clustering with no code available to us, we would simply implement K-means, and spend more time on the actual analysis of data.

The above findings are apparent in our uses of the two clustering techniques for our final algorithm. Due to its relative advantage in speed (and certainly acceptable accuracy), K-means was used in iteratively determining our value of k. Once we had determined k, we used EM's superior clustering to actually assign the data.

One subtlety of our algorithm was also in the use of the tuning parameter in rating the performance of K-means on each value of k. Perfecting this punishment algorithm required some experimentation. We conducted online research to find potential equations, and came across several methods. The first involved adding a constant to the total distance. The constant was

equal to the average distance of the data points to their means, divided by dth root of k, where d is the dimensionality of the data. We could not get this algorithm to produce the proper number of means on our artificial data. The next algorithm we tried, and finally settled on, was to multiply the total distance by a tuning factor. The first tuning factor we used was k. This was found to increase the results too quickly, making any comparison pointless. The best tuning factor found was log(k). This factor produced correct results on our training data where k = 2.

These findings were all quite useful interesting, and we feel confident of our results due to verification with both real and synthetic data. However, there are several things we are not entirely happy with, and would like to test, given more time. One such thing was our lack of experimentation with the size of k. Between both our real and simulated data, the largest k ever became was 3 (and was usually 2). We found with the lotto data that our algorithm for determining k was fairly reliable, but due to our relatively small values found for k, we never tested our EM on data with many clusters. Furthermore, some methods of analysis were not incredibly statistically relevant. That is, in taking the averages of lotto numbers (to decrease dimensionality of the data), we aren't really finding anything important about the data. Performing some form of principal component analysis on the data beforehand might be an interesting avenue to explore in the future.

Similarly, though our implementations proved quite enlightening, and consistent with our prior research, we would have liked to perform more analysis on our real-life data. For example, in the MLB data, it would be interesting to see if we could find clusters for height-weight ratios that corresponded to particular positions, or on-base averages. Similarly, we would have liked to explore how including more dimensions in our data would have affected results. Both the EM and K-means code were generalized to fit any number of dimensions, though we only experimented with two (due to its ease of visual interpretation). On a theoretical note, it would be interesting as well to see if increasing dimensionality affected performance for both algorithms equally. Given more time, these are all interesting questions we would like to answer.

All in all, this was a very informative, and fun process. We were able to experiment with sophisticated statistics algorithms, and dabbled quite extensively in R programming. We liked it.

5 Contributions

Divided by group member and listed alphabetically.

5.1 Marvin Cheng

I was responsible for the data aspect of the project. With a plethora a data on the web and thousands of data entries, a challenge that I had to take up was making the data readable through R. At first, I started scraping UFC fighter and lotto data, but then decided to go with the height and weight data as it worked better for the group. Doing so required the use of regular expressions, and 'trial and error' and prototyping of the data into R.

5.2 Peter Dwersteg

I was in charge of all the aspects of the project related to the K-Means algorithm. This included researching the algorithm, coding it in R, researching and writing the penalization algorithm, creating powerpoint slides and presenting the algorithm to the class, and writing the sections of the final paper that involved K-Means. I was also quite involved in the group discussions about what to do for the project, and how we wanted all the different parts to fit together.

5.3 Tobias Kahan

My main task was visualizing the synthetic data for the presentation and paper. I wrote the Background and Intro and compiled the LaTeX.

5.4 John MacKinnon

I was in charge of all things related to EM. That is, I worked thorough its implementation (including all bugs!), and its integration into our final algorithm. I also handled the research and documentation in the paper behind the theory of EM.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*, page 3. Springer Scince+Business Media, LLC, 2006.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*, pages 435–438. Springer Scince+Business Media, LLC, 2006.

[3] Marina Meila. Probability and statistics for computer science. 2007.