CS 181 HW3 Writeup

Joshua Lee and Salvatore Rinchiera
March 14, 2013

Problem 1

a.

Saying that the maximum distance between any two dimensions is at most epsilon is equivalent to saying that the difference between all dimensions must be less than epsilon. Since we know x is placed in the center and $\varepsilon \in (0, 1/2)$, we know that all y's must fall into the range $(1/2 - \varepsilon, 1/2 + \varepsilon)$. Since y's are placed uniform randomly, the probability is proportional to length:

$$p(\max_m |x_m - y_m| \leq \varepsilon) = \prod_{m \in M} p(|x_m - y_m| \leq \varepsilon) = (2\varepsilon)^M$$

b.

The window you want the dimensions of $\boldsymbol{y}$'s is always at most 2ε. This is maximal and true for all dimensions when $\boldsymbol{x}$ is placed at the center of the hyper cube. Now that $\boldsymbol{x}$ is being placied randomly, if some of the dimensions of $\boldsymbol{x}$ get close to the edge of the hypercube (i.e. close to 0 or 1), then the window the corresponding $\boldsymbol{y}$ dimension can fall into is smaller than 2ε, therefore the probability is less.

c.

We have already shown that $p(\max_m |x_m - y_m| \leq \varepsilon) \leq (2\varepsilon)^M)$. Note that $||\boldsymbol{x} - \boldsymbol{y}|| \geq \max_m |x_m - y_m|$. This is obvious because the square root of the sum of the squared differences between all dimensions is clearly greater than the maximum difference between any one dimension. Therefore, we know that $||\boldsymbol{x} - \boldsymbol{y}||$ must have a SMALLER probability than $\max_m |x_m - y_m|$ of being less than ε, precisely because $||\boldsymbol{x} - \boldsymbol{y}|| \geq \max_m |x_m - y_m|$. Therefore, $p(||\boldsymbol{x} - \boldsymbol{y}|| \leq \epsilon) \leq (2\varepsilon)^M$

d.

Let $p = (2\varepsilon)^M$. Using our result from (c), the probability that a single point will be greater than ε away from $\boldsymbol{x}$ at least $(1 - p)$. So, the probability all N points are greater than ε away from $\boldsymbol{x}$ at least $(1 - p)^N$. Therefore the probability that at least 1 point is at most ε away from $\boldsymbol{x}$ is $1 - (1 - p)^N$. So we have that $1 - (1 - p)^N \geq 1 - \delta$.

e.

Rearranging the inequality from (d), we have that:

$$N \geq \frac{\ln(\delta)}{\ln(1-(2\varepsilon)^M)}$$

This means that to have at least probability $(1-\delta)$ of having 2 points be less than ε apart, we need at least N points, which increases as M increases! Thus, as we have higher dimension data, we need more data to have the same probability of datum to be close to each other. This is the mathematical formulation of what we termed the "Curse of Dimensionality" in lecture.

Problem 2

a.

Maximum Likelihood (ML)

$\theta_{\text{ML}} = \arg\max_{\theta} Pr(\mathcal{D}|\theta)$ (e.g. $\frac{N_1}{N}$ in the case of Bernoulli)

Maximum a posteriori (MAP)

$\theta_{\text{MAP}} = \arg\max_{\theta} Pr(\mathcal{D}|\theta)Pr(\Theta = \theta)$ where $Pr(\Theta = \theta)$ is our prior.
(e.g. $\frac{\alpha+N_1-1}{\alpha+\beta+N-2}$ in the case of Bernoulli).

Full Bayesian (FB)

$Pr(\boldsymbol{x}|\mathcal{D}) = \int_{\boldsymbol{\Theta}} Pr(\boldsymbol{x}|\boldsymbol{\theta})Pr(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta}$

Using each method's respective θ's, we can then calculate $Pr(\boldsymbol{x}|\mathcal{D})$ using the model distribution we choose. For example, in the case we choose to model using a Bernoulli distribution, we have that $Pr(\boldsymbol{x}|\mathcal{D}) = \theta$.

b. MAP method can be considered "more Bayesian" than ML because MAP uses a prior distribution on the parameters which represents the knowledge about the parameters not contained in the training data. The ML approach uses no such prior distribution and only tries to optimize the parameters for the given data.

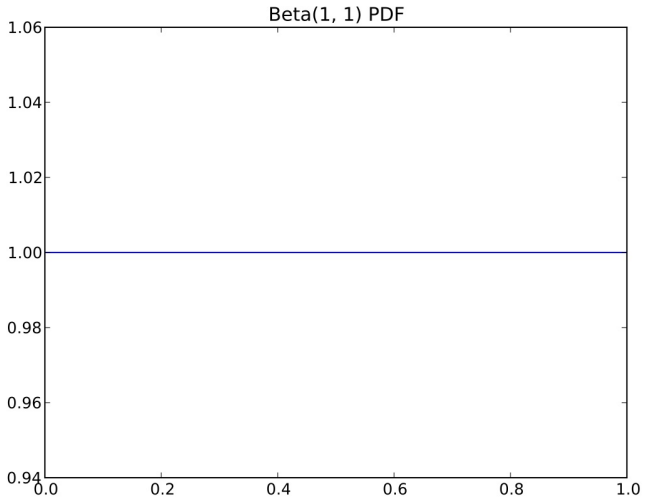
c. ML can suffer from overfitting, especially when the data size is small and is not a good, representative data set. By using a prior distribution on the parameters, MAP allows us to introduce knowledge outside the data to our parameters, which helps avoid just fitting the parameters exactly to the data we are given.

Full Bayesian is much more complicated and keeps a probability distribution over the set of

parameter values. This makes things much more difficult to computer and in some cases may not even converge (i.e. the integral is not finite). So MAP can be advantageous for programming and time complexity reasons.

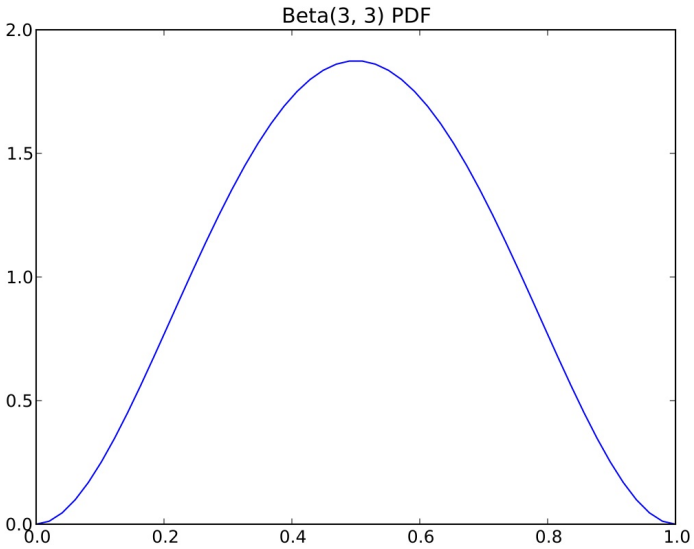
d. Note that given a prior of Beta(**a**, **b**), we can think of **a** as the number of observed successes and **b** as the number of observed failures (or if they are unobserved success/failures, what we think the success and failures are)

Beta (1,1) intuition: Mathematically this is the same as the uniform distribution. Here we believe that there are equal probabilities for the team to win and lose, since we have observed very little data.

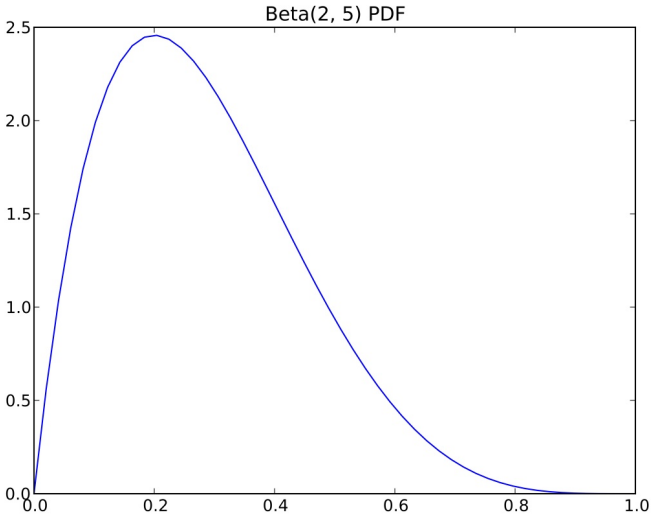


Beta (3, 3) intuition: This looks very much like a normal distribution. Our prior believes that there have been an equal number of wins and losses so it expects that future wins and losses have the same probability of occurring, exposed to some normal variation.

Sorry for poor spacing -- due to conversion from Google Doc to PDF



Beta(2, 5): This distribution is screwed to the left, We have a prior that believes there have been more losses than wins which translates to the prior probability of a loss being higher.



e. The Beta distribution is useful when used together with MAP estimation for reasoning with Bernoulli random variables because the Beta distribution is the conjugate prior of the Bernoulli random variable. That is, the posterior distribution is also a Beta distribution, updated with the sample outcome of the Bernoulli (or series of Bernoullis in the case of a Binomial).

f.

Let $Pr(w_4|\theta)$ be the probability the Harvard football team wins the fourth game of the 2012-13 season, given the parameters θ. We use the Bernoulli distribution as our model distribution.

2011-12 record: 9 wins - 1 loss
2012-13 record: 3 wins - 0 losses (first three games)

Maximum Likelihood (ML)

For ML, we do not use a prior, so we ignore the data from the 2011 - 12 season.
Using the data from the 2012 - 2013 season:

$$N = 3,\ N_1 = 3,\ N_0 = 0,\ \theta_{\text{ML}} = \frac{N_1}{N} = 1$$

Therefore, $Pr(w_4|\theta_{\text{ML}}) = \theta_{\text{ML}}^{w_4}(1-\theta_{\text{ML}})^{(1-w_4)} = 1^{w_4} = 1$

Solution: $Pr(w_4|\theta_{\text{ML}}) = 1$

Maximum a posteriori (MAP)

For MAP, we use the prior distribution:

$$Pr(\Theta = \theta) = \text{Beta}(\Theta = \theta|\alpha, \beta) = \frac{1}{Z}\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

where $Z = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ (normalizing constant) and $\alpha = 9, \beta = 1$ (using the 2011-12 season data).

Now we want to update our prior based on the data, giving us:

$$Pr(\Theta = \theta|\mathcal{D}) = \text{Beta}(\Theta = \theta|9 + N_1, 1 + N - N_1) = \text{Beta}(\Theta = \theta|12, 1)$$

since Beta is the conjugate prior of the Bernoulli and we have $N_1 = 3, N = 3$. So:

$$\theta_{\text{MAP}} = \frac{\alpha + N_1 - 1}{\alpha + \beta + N - 2} = \frac{9 + 3 - 1}{9 + 1 + 3 - 2} = 1$$

Therefore, $Pr(w_4|\theta_{\text{MAP}}) = \theta_{\text{MAP}}^{w_4}(1-\theta_{\text{MAP}})^{(1-w_4)} = 1^{w_4} = 1$

Solution: $Pr(w_4|\theta_{\text{MAP}}) = 1$

Full Bayesian (FB)

Again we use the prior $\Pr(\boldsymbol{\theta}) = \text{Beta}(\boldsymbol{\theta}|9,1)$ and posterior $\Pr(\boldsymbol{\theta}|\mathcal{D}) = \text{Beta}(\boldsymbol{\theta}|12,1)$ we calculated above.

Using the FB method where we now consider the distribution over all possible parameter values, we have:

$$\theta_{\text{BF}} = \int_0^1 Pr(w_4|\theta)Pr(\theta|\mathcal{D})d\theta = \frac{\alpha + N_1}{\alpha + \beta + N} = \frac{9+3}{9+1+3} = \frac{12}{13}$$

Therefore, $Pr(w_4|\theta_{\text{BF}}) = \theta_{\text{BF}}^{w_4}(1-\theta_{\text{BF}})^{(1-w_4)} = \frac{12}{13}^{w_4}\frac{1}{13}^{(1-w_4)}$

Solution: $Pr(w_4|\theta_{\text{BF}}) = \frac{12}{13}^{w_4}\frac{1}{13}^{(1-w_4)}$

Problem 3

a.

K-Means clustering objective:

$$L(\{\boldsymbol{\mu}_k\}_{k=1}^K, \{\boldsymbol{r}_n\}_{n=1}^N) = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$$

Using gradient descent, we can update each prototype based on a single data point by moving it along the gradient multiplied by a learning rate.

$$\mu_k^{(r+1)} \leftarrow \mu_k^{(r)} - \alpha \frac{\delta}{\delta u_k} L(\{\boldsymbol{\mu}_k\}_{k=1}^K, \{\boldsymbol{r}_n\}_{n=1}^N, \boldsymbol{x})$$

To get the derivative of the loss function with respect to the prototype we can calculate the partial derivative of each dimension. Thankfully, the partial derivatives of $\frac{\delta}{\delta u_k}$ is 0 for any squared sum that does not include μ_k Therefore we can simplify...

$$\frac{\delta}{\delta \boldsymbol{\mu}_k} L(\{\boldsymbol{\mu}_k\}_{k=1}^K, \{\boldsymbol{r}_n\}_{n=1}^N, \boldsymbol{x}) =$$

$$\frac{\delta}{\delta \boldsymbol{\mu}_k} \sum_{j=1}^{K} r_{xj}||\boldsymbol{x} - \boldsymbol{\mu}_j||^2 =$$

$$\frac{\delta}{\delta \boldsymbol{\mu}_k} \sum_{j=1}^{K} r_{xj}((x_1 - \mu_{j_1})^2 + (x_2 - \mu_{j_2})^2 + (x_3 - \mu_{j_3})^2 + \cdots + (x_M - \mu_{j_M})^2) =$$

$$\frac{\delta}{\delta \boldsymbol{\mu}_k} r_{xk}((x_1 - \mu_{k_1})^2 + (x_2 - \mu_{k_2})^2 + (x_3 - \mu_{k3})^2 + \cdots + (x_M - \mu_{k_M})^2) =$$

$$r_{xk}(-2(x_1 - \mu_{k_1}) - 2(x_2 - \mu_{k_2}) - 2(x_3 - \mu_{k3}) - \cdots - 2(x_M - \mu_{k_M})) =$$

$$-2r_{xk} \sum_{m=1}^{M} x_m - \mu_{k_m}$$

We can let our alpha absorb our -2. For a single datapoint, x our update function becomes...

$$\mu_k^{(r+1)} \leftarrow \mu_k^{(r)} - \alpha r_{xk} \sum_{m=1}^{M} x_m - \mu_{k_m}$$

r_{xk} is 1 if x is in cluster k, and 0 otherwise. M is the dimension of the datapoints and our prototypes. If we want to use batch gradient descent to update our prototype we simply sum over all datapoints.

$$\mu_k^{(r+1)} \leftarrow \mu_k^{(r)} - \alpha \sum_{n=1}^{N} r_{nk} \sum_{m=1}^{M} x_{n_m} - \mu_{k_{n_m}}$$

r_{nk} is 1 if x_n is in cluster k, and 0 otherwise.

Note that alpha is a learning rate that we can adjust.

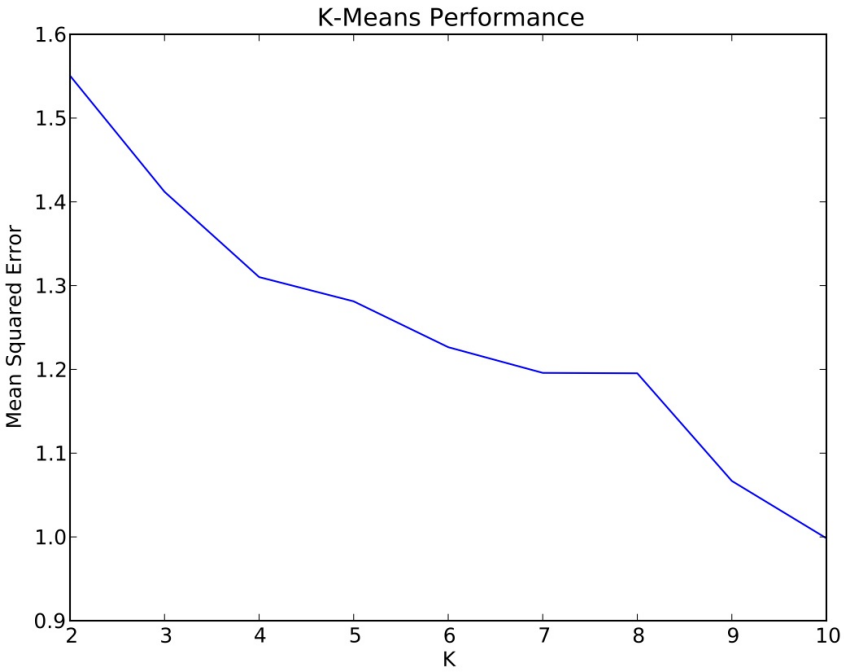
b.

Both algorithms make use of a prototype that is chosen based on the proximity of data points to it. They calculate this proximity using squared loss. In PCA we project data points from high dimensions to lower dimensions and attempt to minimize square loss to our original data points to the projected one, while maximizing the variance. In K-Means we do not project our data points onto a prototype, but simply calculate the distances of data points away from a prototype. K-means assumes symmetry around a prototype. Therefore, datasets for which this is not the case are better clustered by PCA. At the same time, since PCA reduces the dimensionality of the data points, we can see that datasets for which some dimensions are less relevant than others may be better clustered by PCA. This is only true if we choose the least important dimension to project on. On the other hand, if every dimension is influential to our clusters, K-Means, which considers values in every dimension, may be stronger.

Problem 4

a. K-means

(a) Plot of Mean Squared Error vs. K



(b) Which K would you choose?

K corresponds to the number of clusters we believe exist in the data. Ideally, we would look at what the data represents and find a number of groups that we believe represent all the semantically interesting sets. Unfortunately since the data has so many dimensions, we are not able to simply graph the data and eyeball groups.

If we were forced to pick a number for K, we would pick 7. Although as we increase K the mean squared error will continue to decrease, we start to overfit data. We could always just choose K to be the number of data points and have 0 MSE, but obviously this is not meaningful. We choose 7 because that is where the slope seemed to flatten out, suggesting that the marginal gains of additional clusters would be minimal, suggesting we may be splitting a single semantic group in two.

b. HAC

(a)

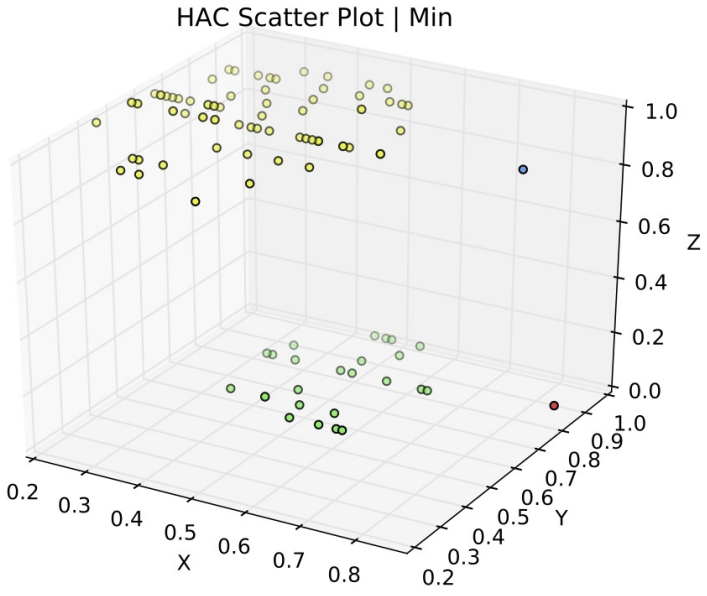
Min Table

	# Instances
Cluster 1	1
Cluster 2	1
Cluster 3	25
Cluster 4	73

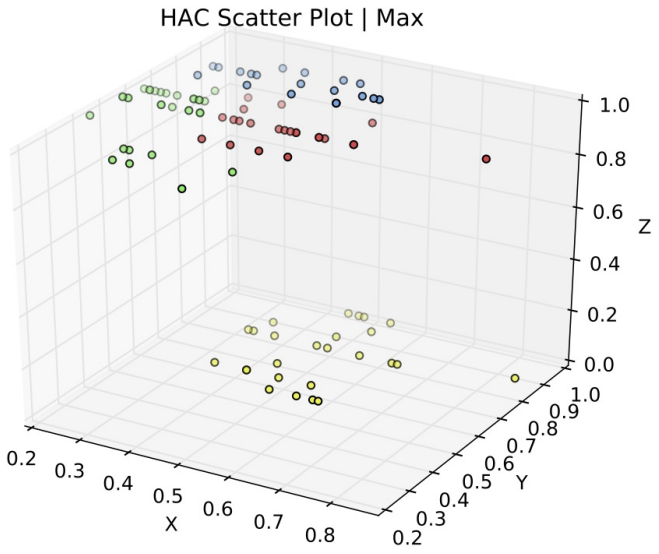
Max Table

	# Instances
Cluster 1	19
Cluster 2	23
Cluster 3	32
Cluster 4	26

Min Scatter Plot



Max Scatter Plot



The MIN clusters are grouped together very clearly, with a lot of space between each cluster. The MAX clusters have a lot less room between clusters, especially between the red, green, and blue clusters.

This does make sense given the definition of the metrics. Both metrics only takes a single pair between the clusters to represent the distance between clusters. For MIN, at each step, it will identify the two closest points not in the same cluster, and merge their two clusters. So points that are close to each other will get merged together always, regardless of what cluster they already belong to.

MAX works differently. Now for the . Thus as clusters grow bigger, it in some ways becomes more difficult for new points to enter the cluster, because even if a point is really close to one point in the cluster, it may be far away from another point in the cluster. This is why we start to see clusters form very close to each other.

(b)

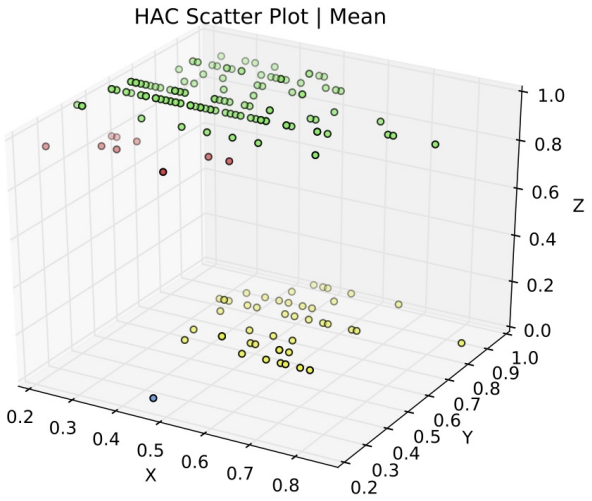
Mean Table

	# Instances
Cluster 1	1
Cluster 2	9
Cluster 3	143
Cluster 4	47

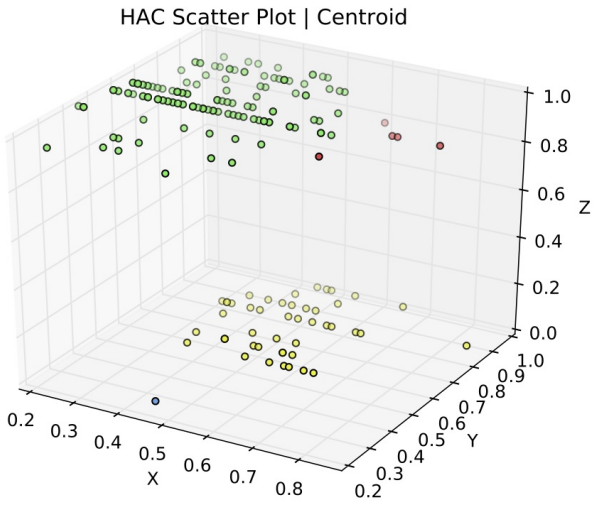
Centroid Table

	# Instances
Cluster 1	1
Cluster 2	5
Cluster 3	147
Cluster 4	47

Mean Scatter Plot



Centroid Scatter Plot



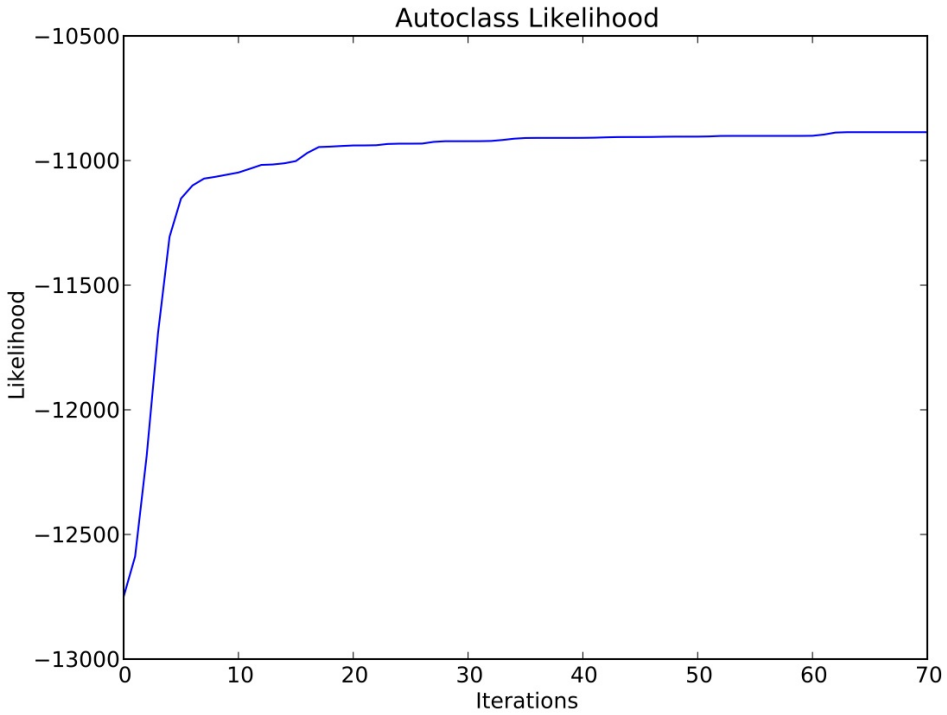
The clusters produced by these two algorithms are very similar. This makes sense given the definition of the metrics. Mean takes average difference between all pairs in each cluster while centroid takes the difference between the centroid of each cluster. Thus, both metrics are taking all points into each cluster into account when calculating the distance, just in a slightly different way. So, it is not surprising the outcomes are very similar, but slightly different.

c. Autoclass

(a) Using epsilon = 1e-5, it took **71** iterations for my parameters to converge.

Note that since we initialize our parameters with random weights, this number (as well as the graph below) varies between runs, but order of magnitude (and the trend) is the same.

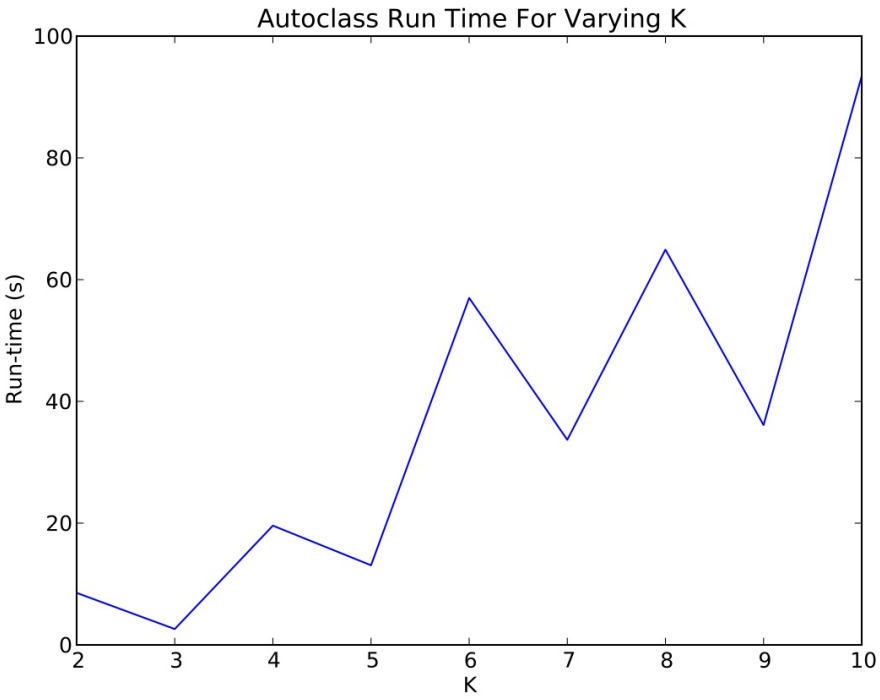
(b) Plot of log likelihood of data vs. iteration number



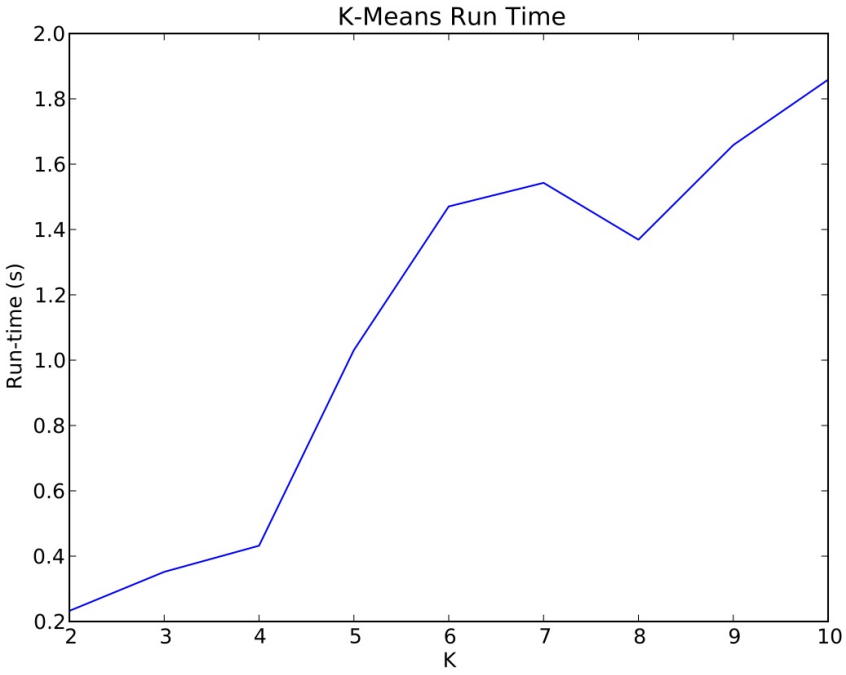
(c) Run-time performance of Autoclass vs. K-means

K-means is on the order of 100 times **faster** than Autoclass. See graphs below.

Autoclass Performance Plot

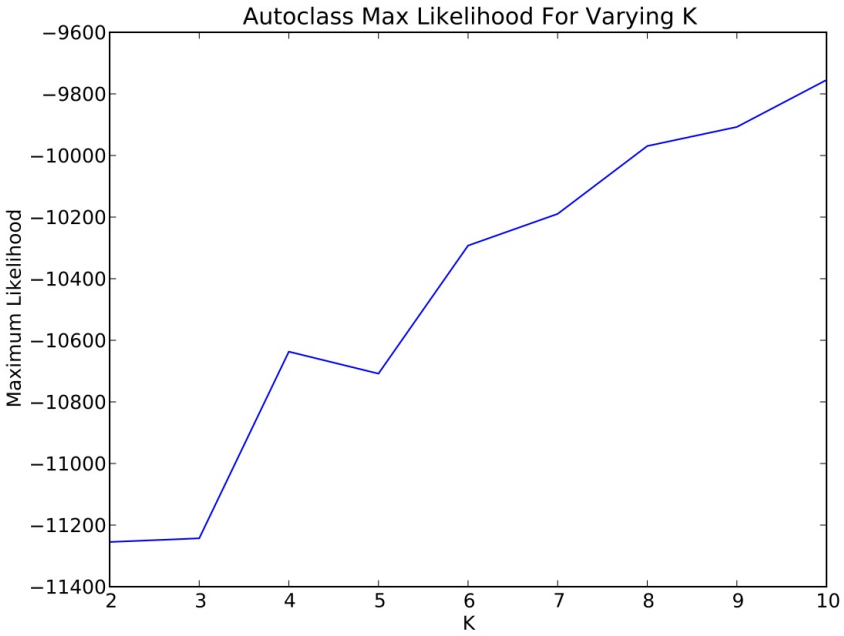


K-means Performance Plot



(d) [Extra Credit] Autoclass comparison with K-means

Plot of **log** likelihood vs. K for Autoclass



Similar to my reasoning for K-means, I would pick K to be 7. This is the point at which the slope of my line seems to flatten, so the trade off between better performance and chance of overfitting seems to be evening out.