HW10 CS 189

1.
Who else did you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

I worked on this homework with Ehimare Okoyomon, Prashanth Ganeth, and Daniel Mockaitis. We worked by getting together throughout the week and communicating on facebook.

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up}

Nicholas Lorio, 26089160

Question 6 Own Question.

Why is it important to have random centroid/starting locations for EM, when does EM fail to converge?

It is important to have different starting points for the method that include points far away from the direction of the iterations in order to understand the form of the log likelihood surface. Random initial centroids and multiple runs can help to ensure that the method does not appear to diverge.

EM has properties to increase its likelihood through each iteration. This does not imply convergence. It may approach the boundary of a parameter space rather than converge to a local maximum.

Often times it is the case that the likelihood has a ridge shape, this can cause EM to not perform as quickly as its reputation suggests. EM is also oftentimes not a good choice when there is substantial data missing. Can have issues with lack of identifiability which can be helped by specifying prior distributions

Nicholas Le
2008960

189 - HWSet 10

2.a. Min val objective when k=n (# clusters = # samples

$$\min_{C_1 \dots C_k} \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|_2^2 = 0$$

as each sample would be assigned to a cluster

2.b.

$$\min_{\mu_i \in \mathbb{R}^d} \lambda \|\mu_i\|_2^2 + \sum_{x_j \in C_i} \|x_j - \mu_i\|_2^2 = \min_{\mu_i \in \mathbb{R}^d} \lambda \mu_i^T \mu_i + \sum_{x_j \in C_i} (x_j - \mu_i)^T (x_j - \mu_i)$$

$$\text{gradient } \nabla_{\mu_i} = 2\lambda \mu_i + \sum_{x_j \in C_i} -2(x_j - \mu_i) = 0$$

$$\lambda \mu_i = \sum (x_j - \mu_i) = \sum x_j - \sum \mu_i$$

$$= \sum x_j - |C_i| \mu_i$$

$$\lambda \mu_i + |C_i| \mu_i = \sum_{x_j \in C_i} x_j$$

$$(\lambda + |C_i|) \mu_i = \sum_{x_j \in C_i} x_j$$

$$\mu_i = \frac{1}{|C_i| + \lambda} \sum_{x_j \in C_i} x_j$$

as required.

2.c. $\wedge$ students $\in \mathbb{R}^2$ euclidean world
k variables, need k locations to cluster (meetups)
'group @ closest point to each student
x_j := each of students
(a,b) := exam location

- let c_i for $i = 1, \ldots, k$ be the locations of the meeting pts.

- need to minimize not only the distance traveled by each student to the meeting points but also the distance the student must travel to the exam location. This adds the regularization term $\|\mu_i\|_2^2$ w/ $\lambda = 1$ to our objective

$$\min_{c_1 \ldots c_k} \sum_{i=1}^{k} \left(\sum_{x_j \in C_i} \left(\|x_j - \mu_i\|_2^2 \right) + 1 \cdot \|\mu_i\|_2^2 \right)$$

2.d. Kernel k means, need to set $i(j)$ to closest cluster

$i(j) = \arg\min_i$ "distance from element to it's cluster w/ the same id"

$= \arg\min_i$ "$\|\phi(x_j) - \phi(x_i)\|_2^2$ But normalized by size of cluster"

$= \arg\min_i \sum_{j \in C_i} \frac{1}{|C_i|} \|\phi(x_j) - \phi(x_i)\|_2^2$

$= \arg\min_i \sum_{j \in C_i} \frac{1}{|C_i|} (\phi(x_j)^T - \phi(x_i)^T)(\phi(x_j) - \phi(x_i))$

2.d. $\arg\min \sum_{j \in C_i} \frac{1}{|C_i|} \left(\phi(x_j)^T \phi(x_j) - 2\phi(x_j)^T \phi(x_i) + \phi(x_i)^T \phi(x_i) \right)$

$i(j) = \arg\min_i \sum_{j \in C_i} \frac{1}{|C_i|} \left(k(x_j, x_j) - 2k(x_j x_i) + k(x_i, x_i) \right)$

4.a.

Joint $p(x=x, z=1; \theta) = p(z=1)\, p(x=x \mid z=1; \theta) = \frac{1}{2}\mathcal{N}(x \mid \mu_1, \sigma_1^2)$

$p(x=x, z=2; \theta) = p(z=2; \theta)\, p(x=x \mid z=2; \theta) = \frac{1}{2}\mathcal{N}(x \mid \mu_2, \sigma_2^2)$

Marginal lik $p(x=x; \theta) = \sum_{k=1}^{2} \frac{1}{2}\left(\mathcal{N}(x \mid \mu_k, \sigma_k^2)\right)$

Loglik $\ell(x=x; \theta) = \log\left(\sum_{k=1}^{2} \frac{1}{2}\mathcal{N}(x \mid \mu_k, \sigma_k^2)\right)$

4.b. $\ell(x_1=x_1, \ldots, x_n=x_n; \theta) = \sum_{i=1}^{n} \log \sum_{k=1}^{2} \frac{1}{2}\mathcal{N}(x_i \mid \mu_k, \sigma_k^2)$

$= \log \prod_{i=1}^{n} p(x_i=x_i; \theta) =$

4.c. $q(z_1=z_1, \ldots, z_n=z_n) = \prod_{i=1}^{n} q_i(z_i=z_i)$

$\ell(x; \theta) := \ell(x=x, \theta)$, $p(x, k; \theta) := p(x=x, z=k; \theta)$

Show, given x_i $\Rightarrow \ell(x_i; \theta) \geq F_i(\theta; q_i)$

$F_i(\theta; q_i) := \sum_{k=1}^{2} q_i(k) \log p(x_i, k; \theta) + \sum_{k=1}^{2} q_i(k) \log \frac{1}{q_i(k)}$

$\ell(x=x_i; \theta) = \log p(x_i; \theta) = \log \sum_{k=1}^{2} p(x_i, k; \theta)$

$= \log \sum_{k=1}^{2} \frac{q(k \mid x_i; \theta)\, p(x_i, k; \theta)}{q(k \mid x_i; \theta)} = \log E_q\left[\frac{p(x_i, k; \theta)}{q(k \mid x_i; \theta)}\right]$

- using jensens inequality for concave functions {From histogram posted slides}

$$E[F(x)] \leq F(E[x])$$

$$\Rightarrow \log E_q[\bigcirc] \geq E_q\left[\log\left(\frac{P(x_i, k; \theta)}{q(k|x_i;\theta)}\right)\right]$$

$$= q(k|x_i;\theta) \cdot \sum_{k=1}^{2}\left(\log \frac{P(x_i,k;\theta)}{q(k|x_i;\theta}\right) \quad \text{, By definition of expectation wrt } q$$

$$= q(k|x_i;\theta) \cdot \sum_{k=1}^{2}\left(\log P(x_i,k;\theta) - \log(q(k|x_i;\theta))\right)$$

$$= \sum_{k=1}^{2} q(k|x_i;\theta)\log P(x_i,k;\theta) + \sum_{k=1}^{2} q(k|x_i;\theta)\cdot \log\left(\frac{1}{q(k|x_i;\theta)}\right)$$

as required.

thus $F(\theta; q) := \sum_{i=1}^{\hat{}} F_i(\theta; q_i)$ is a lower bound on the log likelihood.

$\downarrow$

E_q[complete LL] + entropy(q_i)

$\mathcal{L}(x_i; \theta, q_i) + H(q_i)$

4.b.

given θ^t

From part c: $\ell(\{x_i\}_{i=1}^n ; \theta^t) \geq \mathcal{F}(\theta^t ; q)$

for q w/ n parameters $\{q_i(z_i=1) \; i=1,\dots,n\}$

Verify $LL = \mathcal{F}(\theta^t ; q)$ if $q(z_1=z_1, \dots, z_n=z_n) = \prod_{i=1}^{n} p(z=z_i, x=x_i ; \theta^t)$

$$\mathcal{F}(\theta^t ; q) = H(q_i) + \mathcal{L}(x ; \theta^t, q_i)$$

$$= H(p_i) + \mathcal{L}(x ; \theta^t, p_i)$$

$$= \sum_{k=1}^{2} p(k \mid x_i ; \theta^t) \log\left(\frac{p(x_i, k ; \theta^t)}{p(k \mid x_i, \dots ; \theta^t)} \right) \quad \text{Bayes rule}$$

$$= \sum_{k=1}^{2} p(k \mid x ; \theta^t) \log\left(\frac{p(x_i, k ; \theta^t)\, p(x_i ; \theta^t)}{p(x_i, k ; \theta^t)} \right)$$

$$= \log p(x ; \theta^t) \sum_{k=1}^{2} p(k \mid x_i ; \theta^t) = \log(p(x_i ; \theta^t)) \cdot 1 = LL$$

as required.

4.e. $q_i^{t+1}(z_i = k) = p(z = k \mid x = x_i; \theta^t)$

$$q_i^{t+1}(z_i = 1) = p(z = 1 \mid x = x_i; \theta^t)$$

$$= \frac{p(z=1,\ x=x_i;\theta^t)}{p(x=x_i;\theta^t)} = \frac{p(x=x_i \mid z=1;\theta^t)\, p(z=1;\theta^t)}{p(x=x_i;\theta^t)}$$

- From part a : $= \dfrac{(2\pi\sigma_1^2)^{-1/2} \exp\left(-\frac{1}{2\sigma_1^2}(x_i-\mu_1)^2\right) \cdot \frac{1}{2}}{\frac{1}{2}\left((2\pi\sigma_1^2)^{-1/2}\exp\left(-\frac{1}{2\sigma_1^2}(x_i-\mu_1)^2\right) + (2\pi\sigma_2^2)^{-1/2}\exp\left(\frac{-1}{2\sigma_2^2}(x_i-\mu_2)^2\right)\right.}$

Cancellation

$$= \frac{\frac{1}{\sigma_1}\exp\left(\frac{-1}{2\sigma_1^2}(x_i-\mu_1)^2\right)}{\frac{1}{\sigma_1}\exp\left(\frac{-1}{2\sigma_1^2}(x_i-\mu_1)^2\right) + \frac{1}{\sigma_2}\exp\left(\frac{-1}{2\sigma_2^2}(x_i-\mu_2)^2\right)}$$ as required

- Similar argument can be used to show the required expression for $q_i^{t+1}(z_i = 2)$

- why make sense?

The a Distribution moves in the direction of the true maximum likelihood for the given label.

- Or we can solve for it by probability complement as

$$\frac{x}{x+y} + \frac{y}{x+y} = \frac{x+y}{x+y} = 1$$

4. E. $q_i^{t+1} := q_i^{t+1}(z_i=1), 1-q_i^{t+1}(z_i=2) \quad \theta = (\mu_1, \mu_2, \sigma_1, \sigma_2)$

$$\mathcal{L}(\theta; q^{t+1}) = \sum_{i=1}^{n} \ell(x_i; \theta, q_i^{t+1}) = \sum_{i=1}^{n} \log\left(\sum_{k=1}^{2} p(x_i, k; \theta)\right) q_i(k)$$

$$p(x|k)\, p(k)\, q_i(k)$$

$$= \sum_{i=1}^{n} \left(\log\left(\sum_{k=1}^{2} p(x_i|k;\theta) p(k;\theta)\right) q_i(k)\right)$$

$$= \sum_{i=1}^{n} q_i^{t+1} \cdot \log\left(N(x_i|\mu_1, \sigma_1^2)\frac{1}{2}\right) + (1-q_i^{t+1}) \log\left(\frac{N(x_i|\mu_2,\sigma_2^2)}{2}\right)$$

$$= \sum_{i=1}^{n} q_i^{t+1} \cdot \log\left((2\pi\sigma_1^2)^{-1/2} \exp\left(\frac{-1}{2\sigma_1^2}(x_i-\mu_1)^2\right)\right) + \log\left(\frac{1}{2}\right)$$

$$+ (1-q_i^{t+1}) \log\left((2\pi\sigma_2^2)^{-1/2} \exp\left(\frac{-1}{2\sigma_2^2}(x_i-\mu_2)^2\right) + \log\left(\frac{1}{2}\right)\right)$$

$$= \sum_{i=1}^{n} q_i^{t+1}\left(\log(2\pi)^{-1/2} - \log(\sigma_1) - \left(\frac{(x_i-\mu_1)^2}{2\sigma_1^2}\right) + \log(1/2)\right)$$

$$+ (1-q_i^{t+1})\left(\log(2\pi)^{-1/2} - \log(\sigma_2) - \left(\frac{(x_i-\mu_2)^2}{2\sigma_2^2}\right) + \log(1/2)\right)$$

$$= C - \sum_{i=1}^{n} \left[q_i^{t+1}\left(\frac{(x_i-\mu_1)^2}{2\sigma_1^2} + \log(\sigma_1)\right) + (1-q_i^{t+1})\left(\frac{(x_i-\mu_2)^2}{2\sigma_2^2} + \log(\sigma_2)\right)\right]$$

Where $C = \sum_{i=1}^{n}\left(q_i^{t+1}\left(\log(2\pi)^{-1/2} + \log(1/2)\right) + (1-q_i^{t+1})\left(\log(2\pi)^{-1/2} + \log(1/2)\right)\right)$

$$= n \cdot \log(2\pi)^{-1/2} + \log(1/2)$$

as required.

4.g.

i) $\delta\mathcal{L}/\delta\mu_1 = -\sum_{i=1}^{n}\left[q_i^{t+1}\,\frac{2\cdot(x_i-\mu_1)^1\cdot(-1)}{2\sigma_1^2}\right] = -\sum_{i=1}^{n} q_i^{t+1}\,\frac{(\mu_1-x_i)}{\sigma_1^2}$ as required

ii) $\delta\mathcal{L}/\delta\mu_2 = -\sum_{i=1}^{n}(1-q_i^{t+1})\,\frac{2(x_i-\mu_2)^1(-1)}{2\sigma_2^2}$

$= -\sum_{i=1}^{n}(1-q_i^{t+1})\,\frac{(\mu_2-x_i)}{\sigma_2^2}$

iii) $\delta\mathcal{L}/\delta\sigma_1 = 0 = \sum_{i=1}^{n}\left[q_i^{t+1}\left(\frac{(x_i-\mu_1)^2\cdot 2\cdot\sigma_1^{-3}}{2} + \frac{1}{\sigma_1}\right)\right]$

$= +\sum_{i=1}^{n} q_i^{t+1}\left(\frac{(x_i-\mu_1)^2}{\sigma_1^3} - \frac{1}{\sigma_1}\right)$ as required

iv) $\delta\mathcal{L}/\delta\sigma_2 = -\sum_{i=1}^{n}\left[(1-q_i^{t+1})\left[\frac{(x_i-\mu_2)^2\cdot(-2)\,\sigma_2^{-3}}{2} + \frac{1}{\sigma_2}\right]\right]$

$= +\sum_{i=1}^{n}(1-q_i^{t+1})\left[\frac{(x_i-\mu_2)^2}{\sigma_2^3} - \frac{1}{\sigma_2}\right]$ as required

4.h.

i) $$-\sum \frac{q_i^{t+1}(\mu_1 - x_i)}{\sigma_1^2} = 0$$

$$-\frac{1}{\sigma_1^2}\sum q_i^{t+1}\mu_1 + \frac{1}{\sigma_1^2}\sum q_i^{t+1} x_i = 0$$

$$\mu_1^{t+1} = \frac{\sum q_i^{t+1} x_i}{\sum q_i^{t+1}} = \mu_1^{t+1} \text{ as required}$$

ii) Similar argument as i), $$-\sum (1-q_i^{t+1})\frac{(\mu_2 - x_i)}{\sigma_2^2} = 0$$

iii) $$\sum \frac{q_i^{t+1}(x_i-\mu_1)^2}{\sigma_1^3} - \sum \frac{q_i^{t+1}}{\sigma_1} = 0$$

$$\sum q_i^{t+1}(x_i-\mu_1)^2 = \sum \frac{q_i^{t+1}}{\sigma_1}, \quad \sum q_i^{t+1}(x_i-\mu_1)^2 = \sum q_i^{t+1}\sigma_1^2$$

$$\sigma_1^{2t} = \frac{\sum q_i^{t+1}(x_i-\mu_1^{t+1})^2}{\sum q_i^{t+1}} = \sigma_1^{2(t+1)} = \frac{\sum q_i^{t+1}(x_i-\mu_1^{t+1})^2}{\sum q_i^{t+1}} \text{ as required}$$

iv) Similar argument as iii), $$+\sum \frac{(1-q_i^{t+1})(x_i-\mu_2)^2}{\sigma_2^3} - \frac{\sum(1-q_i^{t+1})}{\sigma_2} = 0$$

Reasons: Each updated element of θ, $\mu_1, \mu_2, \sigma_1, \sigma_2$ is just a weighted average, weighted variance of the observed data. This moves our distribution closer to it's true values through the updates.

4.i $p(x=x;\mu) = \frac{1}{2}\exp\left(\frac{-1/2(x-\mu)^2}{\sqrt{2\pi}}\right) + \frac{1}{2}\exp\left(\frac{-1/2(x+\mu)^2}{\sqrt{2\pi}}\right)$

$$\log\left(\hat{\prod_{i=1}}\frac{1}{2}\left(\exp\left(\frac{-1/2(x_i-\mu)^2}{\sqrt{2\pi}}\right) + \exp\left(\frac{-1/2(x_i+\mu)^2}{\sqrt{2\pi}}\right)\right)\right)$$

$$\ell(\{x\}_{i=1}^{n};\mu) = \hat{\sum_{i=1}^{n}} \log(1/2) + \log\left(\exp\left(\frac{-\frac{1}{2}(x_i-\mu)^2}{\sqrt{2\pi}}\right) + \exp\left(\frac{-\frac{1}{2}(x_i+\mu)^2}{\sqrt{2\pi}}\right)\right)$$

4.j. $q_i^{t+1}(z_i=1) =$

From part d, e $\left\{ \begin{array}{l} [q_i^{t+1}(z_i=1) = p(z=1 \mid x=x;\theta^t)] \\ = \frac{p(x=x \mid z=1;\theta^t)\,p(z=1;\theta^t)}{p(x=x;\theta^t)} \end{array} \right.$

$\mu_1 = \mu^t$

$\mu_2 = -\mu^t$

$\sigma_1 = \sigma_2 = 1$

$$= \frac{(2\pi)^{-1/2}\exp\left(-\frac{1}{2}(x_i-\mu)^2\right)\cdot 1/2}{\frac{1}{2}\left((2\pi)^{-1/2}\exp\left(-\frac{1}{2}(x_i-\mu)^2\right) + (2\pi)^{-1/2}\exp\left(-\frac{1}{2}(x_i+\mu)^2\right)\right)}$$

sub $\mu = \mu^t$

$$= \exp\left((x_i-\mu^t)^2/2\right) \Big/ \left(\exp\left((x_i-\mu^t)^2/2\right) + \exp\left((x_i+\mu^t)^2/2\right)\right)$$

4.k5 plug into part F

$$\mathcal{L}(\mu; q^{t+1}) = \sum_{i=1}^{n} \sum_{k=1}^{2} q_i(k) \cdot \log\left(p(x_i, k; \theta)\right)$$

... as in part F

$$= \sum_{i=1}^{n} q_i^{t+1} \log\left(N(x_i \mid \mu, 1) \cdot \tfrac{1}{2}\right) + (1 - q_i^{t+1}) \log\left(N(x_i \mid -\mu, 1) \cdot \tfrac{1}{2}\right)$$

$$= \sum_{i=1}^{n} q_i^{t+1} \left(\log(1/2) + \log\left((2\pi)^{-1/2} \exp\left(-\tfrac{1}{2}(x_i - \mu)^2\right)\right)\right.$$

$$\left. + (1 - q_i^{t+1}) \left(\log(1/2) + \log(2\pi)^{-1/2} + -\tfrac{1}{2}(x_i + \mu)^2\right)\right.$$

$$= C - \sum_{i=1}^{n} q_i^{t+1} \frac{(x_i - \mu)^2}{2} + (1 - q_i^{t+1}) \frac{(x_i + \mu)^2}{2}$$

w/ $C = n \cdot \left(\log(1/2) + \log(2\pi)^{-1/2}\right)$

4.k. ii) $\partial \mathcal{L} / \partial \mu = -\sum_{i=1}^{n} \left(q_i^{t+1} \tfrac{2(x_i - \mu)}{2} \cdot (-1) + \tfrac{(1 - q_i^{t+1}) 2(x_i + \mu)}{2} \right)$

$$= -\sum q_i^{t+1}(x_i - \mu) - \sum (1 - q_i^{t+1})(x_i + \mu) = 0$$

$$\sum q_i^{t+1} x_i - \sum q_i^{t+1} \mu - \sum (1 - q_i^{t+1}) x_i - \sum (1 - q_i^{t+1}) \mu = 0$$

$$\mu^{k} = \mu^{t+1} = \frac{\sum_{i=1}^{n} (2 q_i^{t+1} - 1) x_i}{\sum_{i=1}^{n} (1 - q_i^{t+1})}$$

4. (b). ii) $\partial \mathcal{L}/\partial \mu = -\sum_{i=1}^{n} q_i^{t+1}(\mu - x_i) - \sum (1-q_i^{t+1})(x_i+\mu) = 0$

$$\sum q_i^{t+1}(x_i - \mu) = \sum (1-q_i^{t+1})(x_i+\mu)$$

$$\sum q_i^{t+1} x_i - \sum q_i^{t+1}\mu = \sum (1-q^{t+1})x_i + \sum (1-q^{t+1})\mu$$

$$\sum (q_i^{t+1} - 1 + q^{t+1})\, x_i = \sum (1 - q_i^{t+1} + q_i^{t+1})\,\mu$$

$$\sum (2q_i^{t+1} - 1)\, x_i = n \cdot \mu$$

$$\boxed{\mu = \frac{1}{n}\sum_{i=1}^{n} (2q_i^{t+1} - 1)\, x_i} \quad \text{as required}$$

4. c) $\partial\left(\ell\left(\{x_i\}_{i=1}^{n}; \mu\right)\cdot \frac{1}{n}\right)/\partial\mu$

$$\partial\left(\sum_{i=1}^{n} \log(1/2) + \log\left(\exp\left(\frac{-1/2(x_i-\mu)^2}{\sqrt{2\pi}}\right) + \exp\left(\frac{-1/2(x_i+\mu)^2}{\sqrt{2\pi}}\right)\right)\cdot \frac{1}{n}\right)/\partial\mu$$

$$\frac{\partial}{\partial\mu}\left(\frac{1}{n}\left(n\cdot\log(1/2) + \sum_{i=1}^{n}\log\left(\exp(-(x_i-\mu)^2/2\sqrt{2\pi}) + \exp(-(x_i+\mu)^2/2\sqrt{2\pi})\right)\right)\right)$$

$$= \frac{1}{n}\sum \log\left(\frac{-2(x_i-\mu)}{2\sqrt{2\pi}}\exp\left(-(x_i-\mu)^2/2\sqrt{2\pi}\right) + \frac{-2(x_i+\mu)\cdot 1}{2\sqrt{2\pi}}\exp\left(\frac{-(x_i+\mu)^2}{2\sqrt{2\pi}}\right)\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi}}\left((x_i-\mu)\exp\left(-(x_i-\mu)^2/2\sqrt{2\pi}\right) - (x_i+\mu)\exp\left(\frac{-(x_i+\mu)^2}{2\sqrt{2\pi}}\right)\right)\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi}}\left[x_i\left(\exp\left(-(x_i-\mu)^2/2\sqrt{2\pi}\right) - \exp\left(-(x_i+\mu)^2/2\sqrt{2\pi}\right)\right)\right.\right.$$

$$\left.\left. - \mu\left(\exp\left(-(x_i-\mu)^2/2\sqrt{2\pi}\right) + \exp\left(-(x_i+\mu)^2/2\sqrt{2\pi}\right)\right)\right]\right)$$

4. f

$$\frac{d}{d\mu}\left(\frac{1}{n}\ell(\{x_i\}_{i=1}^n;\mu)\right) = \frac{\partial}{\partial\mu}\left(\frac{1}{n}\left(-n\log(2\sqrt{2\pi}) + \sum_{i=1}^n \log\left(e^{-\frac{1}{2}(x-\mu)^2} + e^{-\frac{1}{2}(x+\mu)^2}\right)\right)\right)$$

$$= \frac{1}{n}\sum_{i=1}^n \frac{1}{e^{-\frac{1}{2}(x-\mu)^2} + e^{-\frac{1}{2}(x+\mu)^2}}\left(\begin{array}{l} e^{-\frac{1}{2}(x-\mu)^2}\cdot(-(x-\mu))(-1) \\ + e^{-\frac{1}{2}(x+\mu)^2}\cdot(-(x+\mu))\end{array}\right)$$

$$= \frac{1}{n}\sum \frac{1}{e^{-\frac{1}{2}(x-\mu)^2} + e^{-\frac{1}{2}(x+\mu)^2}}\left(\begin{array}{l} x_i\left(e^{-\frac{1}{2}(x-\mu)^2} - e^{-\frac{1}{2}(x+\mu)^2}\right) \\ -\mu\left(e^{-\frac{1}{2}(x-\mu)^2} + e^{-\frac{1}{2}(x+\mu)^2}\right)\end{array}\right)$$

$$= \frac{1}{n}\sum_{i=1}^n \frac{e^{-\frac{1}{2}(x-\mu)^2} - e^{-\frac{1}{2}(x+\mu)^2}}{e^{-\frac{1}{2}(x-\mu)^2} + e^{-\frac{1}{2}(x+\mu)^2}} x_i - \mu$$

- Show that $= 2w_i - 1$ w/ $w_i = \dfrac{e^{-(x_i-\mu)^2/2}}{e^{-(x_i-\mu)^2/2} + e^{-(x_i+\mu)^2/2}}$

$$2w-1 = \frac{2e^{-(x_i-\mu)^2/2}}{e^{-(x_i-\mu)^2/2} + e^{-(x_i+\mu)^2/2}} - \left(\frac{e^{-(x_i-\mu)^2/2} + e^{-(x_i+\mu)^2/2}}{e^{-(x_i-\mu)^2/2} + e^{-(x_i+\mu)^2/2}}\right)$$

$$= \frac{e^{-(x_i-\mu)^2/2} - e^{-(x_i-\mu)^2/2}}{e^{-(x_i-\mu)^2/2} + e^{-(x_i-\mu)^2/2}}$$

$$= \frac{1}{n}\sum (2w_i-1)(x_i) - \mu = \frac{1}{n}\sum (2w_i-1)x_i - \frac{1}{n}\sum \mu_i$$

$$= \frac{1}{n}\sum (2w_i-1)x_i - \frac{1}{n}\mu\sum 1 = \boxed{\left[\frac{1}{n}\sum (2w_i-1)x_i\right] - \mu} \text{ (as required)}$$

- plugged into our update function

$$\mu_{GA}^{t+1} = \mu_{GA}^t + \alpha\, d/d\mu\, \ell(\{x_i\}_{i=1}^n;\mu)\big|_{\mu=\mu_{GA}^t}$$

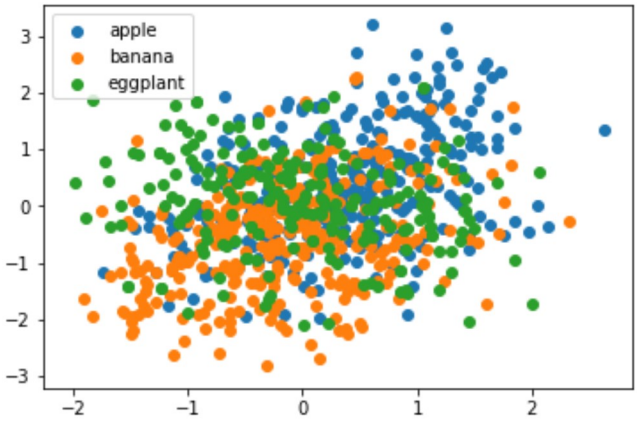
$$\Rightarrow \mu^{t+1} = \mu^t + \alpha\left(\left[\frac{1}{n}\sum (2w_i-1)x_i\right] - \mu_t\right) = \boxed{(1-\alpha)\mu_{GA}^t + \alpha\left[\frac{1}{n}\sum (2w_i-1)x_i\right]} \text{ as required}$$

Question 3

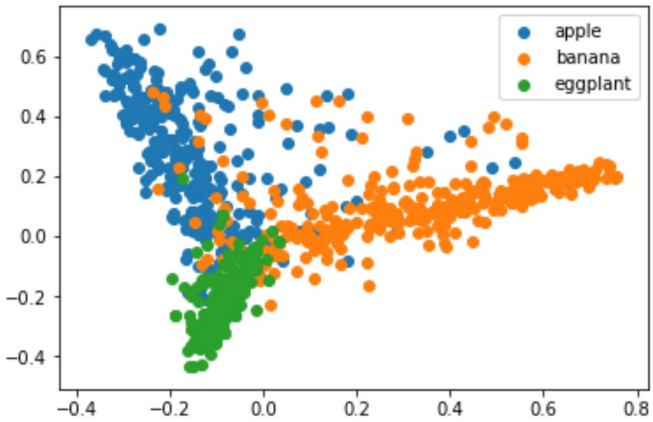
In [2]:
```
%matplotlib inline
```

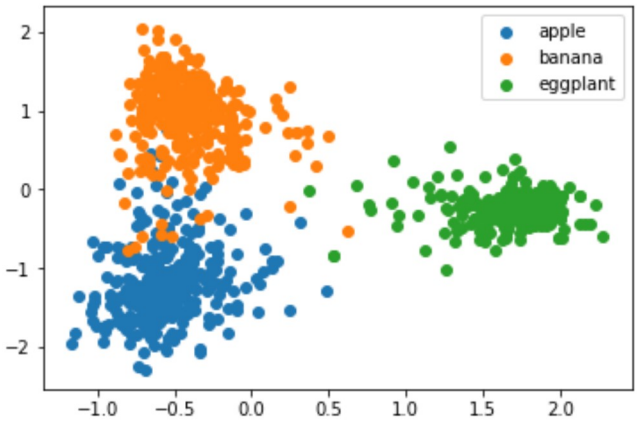
3.a, 3.b, 3.c

In [3]:
```
from numpy.random import uniform
from numpy.random import randn
import random
import time

import matplotlib.pyplot as plt

from scipy.linalg import eig
from scipy.linalg import sqrtm
from numpy.linalg import inv
from numpy.linalg import svd

from utils import create_one_hot_label
from utils import subtract_mean_from_data
from utils import compute_covariance_matrix

import numpy as np
import numpy.linalg as LA

import sys
from numpy.linalg import svd
import IPython



class Project2D():

    '''
    Class to draw projection on 2D scatter space
    '''

    def __init__(self,projection, clss_labels):

        self.proj = projection
        self.clss_labels = clss_labels


    def project_data(self,X,Y,white=None):

        '''
        Takes list of state space and class labels
        State space should be 2D
        Labels shoud be int
        '''

        p_a = []
        p_b = []
        p_c = []

        ###PROJECT ALL DATA###
        proj = np.matmul(self.proj,white)

        X_P = np.matmul(proj,np.array(X).T)

        for i in range(len(Y)):

            if Y[i] == 0:
                p_a.append(X_P[:,i])
            elif Y[i] == 1:
                p_b.append(X_P[:,i])
            else:
                p_c.append(X_P[:,i])
```




```
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:475: DataConv
ersionWarning: Data with input dtype int64 was converted to float64 by StandardS
caler.
  warnings.warn(msg, DataConversionWarning)
```





c (Observations)

Of the three projections, CCA performs the best while random projections performs the worst.

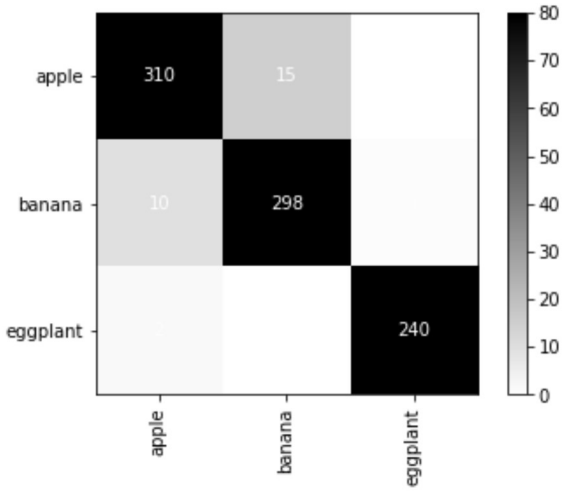
The Random projections are the worst because when bringing a value down from a dimension of 729 to 2, the 2 features you choose can drastically influence your results since some will have much more of a significant impact than others. With such a large sample space, it is highly improbable that we would pick the exact 2 components that maximize differences (it is as probable as picking the exact 2 components that minimize differences). Thus our results are all clustered together.

PCA and CCA both try to pick the directions of most variance so it makes sense that their results are much different and better than random projections. CCA performs better than PCA because PCA only takes into account the input data in order to determine the areas of largest variance Σ_{XX} while CCA uses this the variance of the observations and the relationship between both to figure out the relationship between the inputs and their respective outputs. Therefore CCA's results are slightly better than those of PCA.

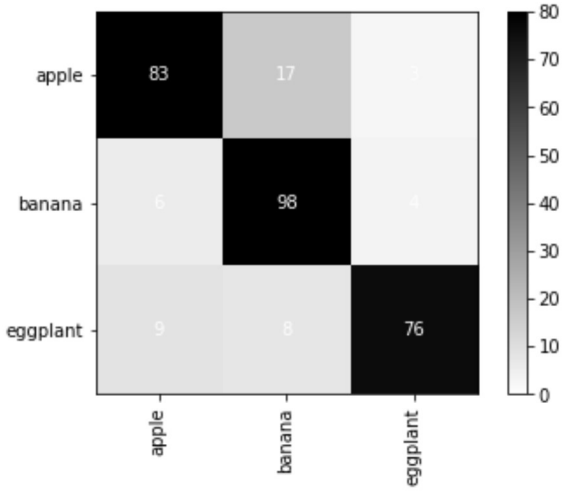
3.d

In [4]:
```
from numpy.random import uniform
import random
import time


import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA

import sys


from projection import Project2D, Projections
from confusion_mat import getConfusionMatrixPlot

from ridge_model import Ridge_Model
from qda_model import QDA_Model
from lda_model import LDA_Model
from svm_model import SVM_Model
from logistic_model import Logistic_Model


CLASS_LABELS = ['apple','banana','eggplant']




class Model():
    """ Generic wrapper for specific model instance. """

    def __init__(self, model):
        """ Store specific pre-initialized model instance. """

        self.model = model


    def train_model(self,X,Y):
        """ Train using specific model's training function. """

        self.model.train_model(X,Y)

    def test_model(self,X,Y):
        """ Test using specific model's eval function. """
        if hasattr(self.model, "evals"):
            labels = np.array(Y)
            p_labels = self.model.evals(X)

        else:
            labels = [] # List of actual labels
            p_labels = [] # List of model's predictions
            success = 0 # Number of correct predictions
            total_count = 0 # Number of images

            for i in range(len(X)):

                x = X[i] # Test input
                y = Y[i] # Actual label
                y_ = self.model.eval(x) # Model's prediction
                labels.append(y)
                p_labels.append(y_)

                if v == v :
```

```
Computing Confusion Matrix
[[310  15   0]
 [ 10 298   1]
 [  2   0 240]]
```




```
Computing Confusion Matrix
[[83 17  3]
 [ 6 98  4]
 [ 9  8 76]]
```

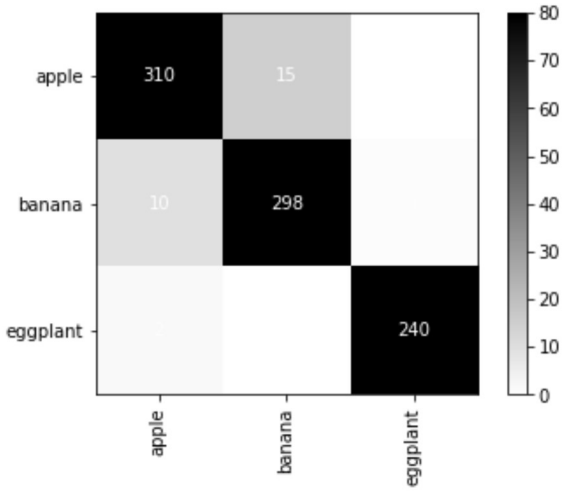



```
Computing Confusion Matrix
```

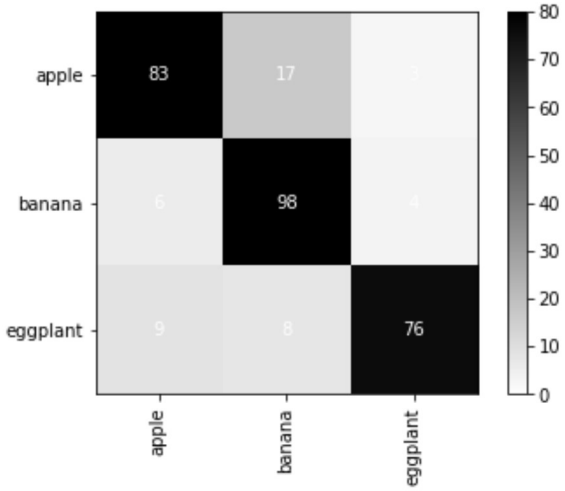
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-4-8c057abf8257> in <module>()
    110
    111     model.train_model(X,Y)
--> 112     model.test_model(X,Y)
    113     model.test_model(X_val,Y_val)
    114

<ipython-input-4-8c057abf8257> in test_model(self, X, Y)
     68         print("Computing Confusion Matrix")
     69         # Compute Confusion Matrix
---> 70         getConfusionMatrixPlot(labels,p_labels,CLASS_LABELS)
     71
     72

/mnt/c/Users/nicho/Berkeley/hw10-data/fruit_veg/confusion_mat.py in getConfusion
MatrixPlot(true_labels, predicted_labels, alphabet)
     80
     81     # Generate confusion matrix using sklearn.metrics
---> 82     cm = confusion_matrix(true_labels, predicted_labels)
     83     print(cm)
     84

/usr/local/lib/python3.5/dist-packages/sklearn/metrics/classification.py in conf
usion_matrix(y_true, y_pred, labels, sample_weight)
    248
    249     """
--> 250     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    251     if y_type not in ("binary", "multiclass"):
    252         raise ValueError("%s is not supported" % y_type)

/usr/local/lib/python3.5/dist-packages/sklearn/metrics/classification.py in _che
ck_targets(y_true, y_pred)
     79     if len(y_type) > 1:
     80         raise ValueError("Classification metrics can't handle a mix of {
0} "
---> 81                          "and {1} targets".format(type_true, type_pred))
     82
     83     # We can't have more than one value on y_type => The set is no more
needed

ValueError: Classification metrics can't handle a mix of multiclass and unknown
targets
```

In [5]:
```
import random
import time


import glob
import os
import pickle
import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA

import sys
from numpy.linalg import inv
from numpy.linalg import det
from sklearn.svm import LinearSVC
from projection import Project2D, Projections


class LDA_Model():

    def __init__(self,class_labels):

        ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED COVARIANCE MAT
RIX TO PREVENT IT BEING SINGULAR ###
        self.reg_cov = 0.001
        self.NUM_CLASSES = len(class_labels)
        self.Xs = {}
        self.mean = {}
        #self.cov = None
        self.cov_inv = None



    def train_model(self,X,Y):
        ''''
        FILL IN CODE TO TRAIN MODEL
        MAKE SURE TO ADD HYPERPARAMTER TO MODEL

        '''
        n = len(Y)
        for i in range(0, self.NUM_CLASSES):
            self.Xs[i] = []

        for i in range(n):
            # Y is a single class number, not one-hot encoded
            self.Xs[Y[i]].append(X[i])

        for x, arr in self.Xs.items():
            self.mean[x] = np.mean(np.array(arr), axis = 0).reshape((-1,1))
        np_X = np.array(X)
        # self.cov = compute_covariance_matrix(np_X, np_X) + self.reg_cov*np.eye(np
_X.shape[1])
        self.cov_inv = np.linalg.inv(compute_covariance_matrix(np_X, np_X) + self.r
eg_cov*np.eye(np_X.shape[1]))


    def eval(self,x):
        ''''
        Fill in code to evaluate model and return a prediction
        Prediction should be an integer specifying a class
        '''
        probs = []
        x = x.reshape((-1, 1))
```

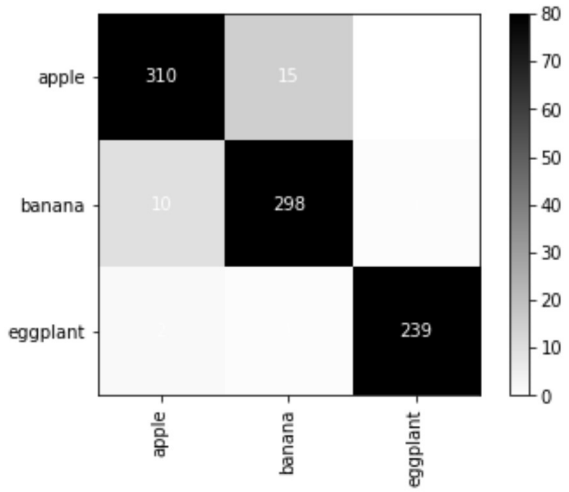
```
Computing Confusion Matrix
[[310  15   0]
 [ 10 298   1]
 [  2   0 240]]
```



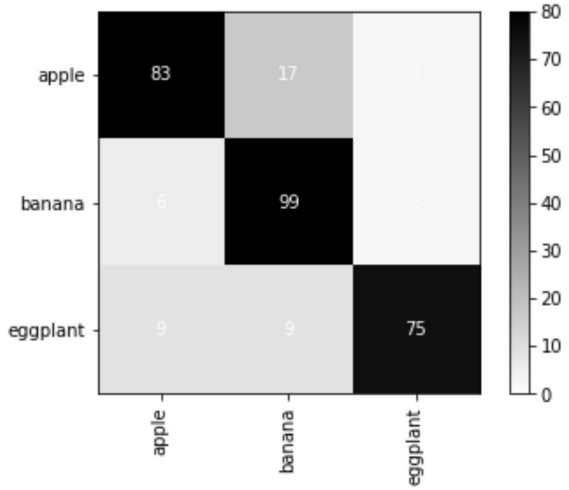

```
Computing Confusion Matrix
[[83 17  3]
 [ 6 98  4]
 [ 9  8 76]]
```




```
Computing Confusion Matrix
[[310  15   0]
 [ 10 298   1]
 [  2   1 239]]
```




```
Computing Confusion Matrix
[[83 17  3]
 [ 6 99  3]
 [ 9  9 75]]
```

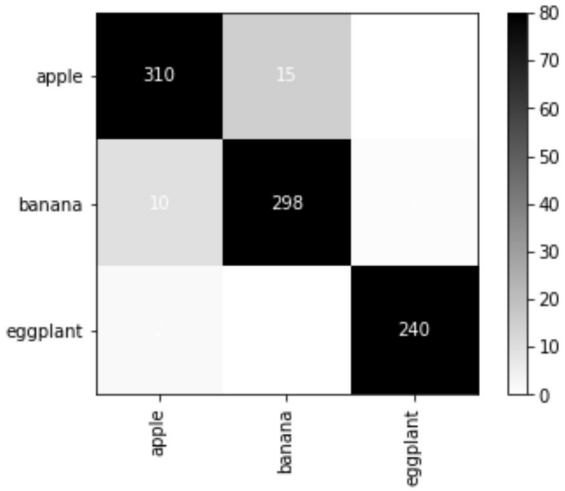



```
Computing Confusion Matrix
```

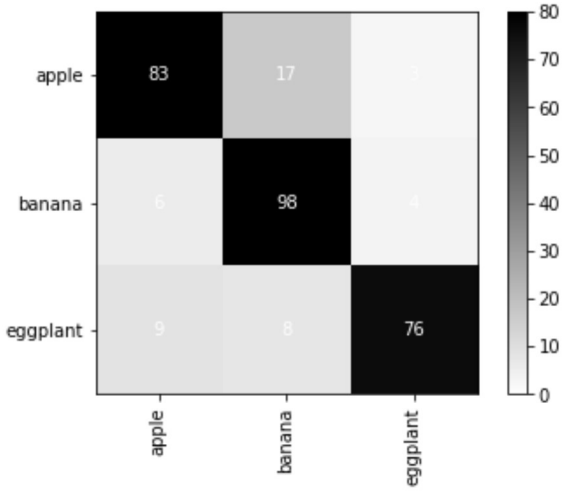
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-5-557f49b4f093> in <module>()
    120
    121     model.train_model(X,Y)
--> 122     model.test_model(X,Y)
    123     model.test_model(X_val,Y_val)
    124

<ipython-input-4-8c057abf8257> in test_model(self, X, Y)
     68         print("Computing Confusion Matrix")
     69         # Compute Confusion Matrix
---> 70         getConfusionMatrixPlot(labels,p_labels,CLASS_LABELS)
     71
     72

/mnt/c/Users/nicho/Berkeley/hw10-data/fruit_veg/confusion_mat.py in getConfusion
MatrixPlot(true_labels, predicted_labels, alphabet)
     80
     81     # Generate confusion matrix using sklearn.metrics
---> 82     cm = confusion_matrix(true_labels, predicted_labels)
     83     print(cm)
     84

/usr/local/lib/python3.5/dist-packages/sklearn/metrics/classification.py in conf
usion_matrix(y_true, y_pred, labels, sample_weight)
    248
    249     """
--> 250     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    251     if y_type not in ("binary", "multiclass"):
    252         raise ValueError("%s is not supported" % y_type)

/usr/local/lib/python3.5/dist-packages/sklearn/metrics/classification.py in _che
ck_targets(y_true, y_pred)
     79     if len(y_type) > 1:
     80         raise ValueError("Classification metrics can't handle a mix of {
0} "
---> 81                          "and {1} targets".format(type_true, type_pred))
     82
     83     # We can't have more than one value on y_type => The set is no more
needed

ValueError: Classification metrics can't handle a mix of multiclass and unknown
targets
```

In [6]:
```
import random
import time


import numpy as np
import numpy.linalg as LA


from numpy.linalg import inv
from numpy.linalg import det

from projection import Project2D, Projections


class QDA_Model():

    def __init__(self,class_labels):

        ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED COVARIANCE MAT
RIX TO PREVENT IT BEING SINGULAR ###
        self.reg_cov = 0.01
        self.NUM_CLASSES = len(class_labels)
        self.Xs = {}
        self.mean = {}
        # self.cov = {}
        self.cov_inv = {}
        self.reg_mtx = None
        self.reg = {}


    def train_model(self,X,Y):
        ''''
        FILL IN CODE TO TRAIN MODEL
        MAKE SURE TO ADD HYPERPARAMTER TO MODEL

        '''
        n = len(Y)
        for i in range(0, self.NUM_CLASSES):
            self.Xs[i] = []

        for i in range(n):
            # Y is a single class number, not one-hot encoded
            self.Xs[Y[i]].append(X[i])

        self.reg_mtx = self.reg_cov*np.eye(len(X[0]))

        for x, arr in self.Xs.items():
            np_arr = np.array(arr)
            self.mean[x] = np.mean(np_arr, axis = 0).reshape((-1,1))
            # self.cov[x] = compute_covariance_matrix(np_arr, np_arr) + self.reg_mt
x
            cov = compute_covariance_matrix(np_arr, np_arr) + self.reg_mtx
            self.cov_inv[x] = np.linalg.inv(cov)
            self.reg[x] = np.log(np.linalg.det(cov))



    def eval(self,x):
        ''''
        Fill in code to evaluate model and return a prediction
        Prediction should be an integer specifying a class
        '''
        probs = []
        x = x.reshape((-1, 1))
```

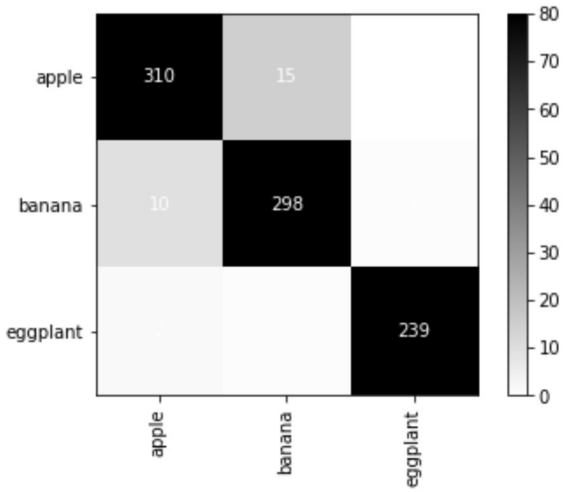
```
Computing Confusion Matrix
[[310  15   0]
 [ 10 298   1]
 [  2   0 240]]
```



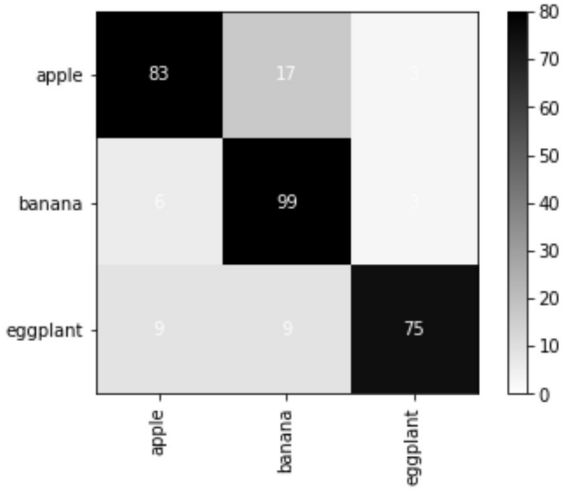

```
Computing Confusion Matrix
[[83 17  3]
 [ 6 98  4]
 [ 9  8 76]]
```



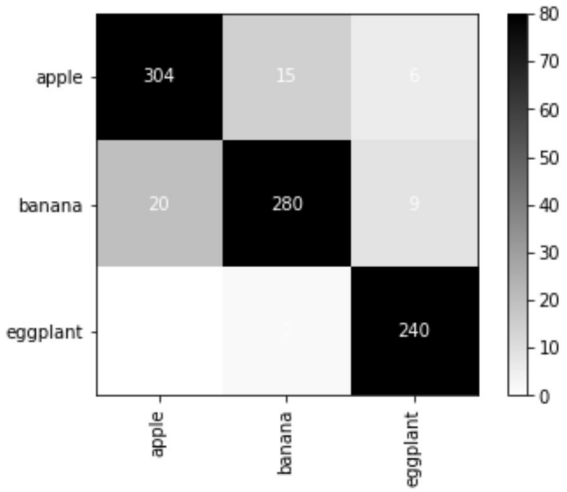

```
Computing Confusion Matrix
[[310  15   0]
 [ 10 298   1]
 [  2   1 239]]
```



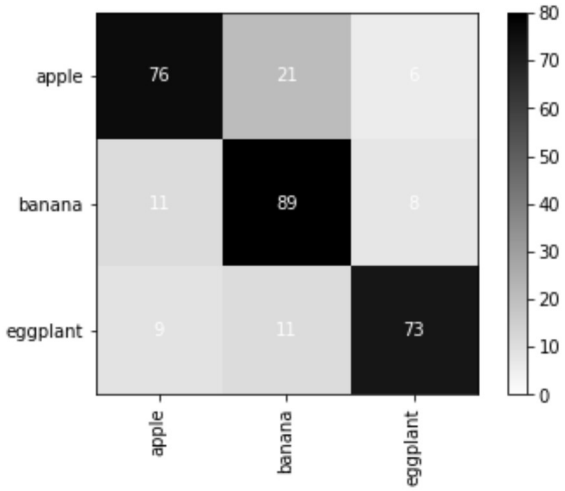

```
Computing Confusion Matrix
[[83 17  3]
 [ 6 99  3]
 [ 9  9 75]]
```



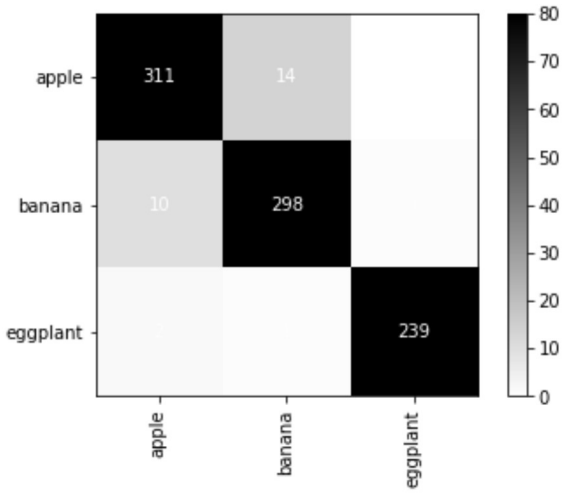

```
Computing Confusion Matrix
[[304  15   6]
 [ 20 280   9]
 [  0   2 240]]
```



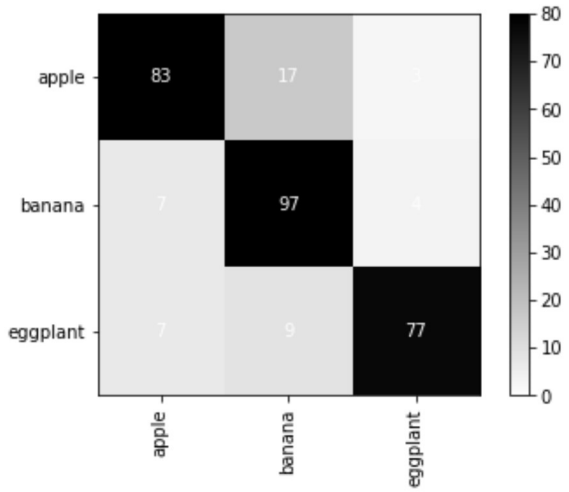
```
Computing Confusion Matrix
[[76 21  6]
 [11 89  8]
 [ 9 11 73]]
```



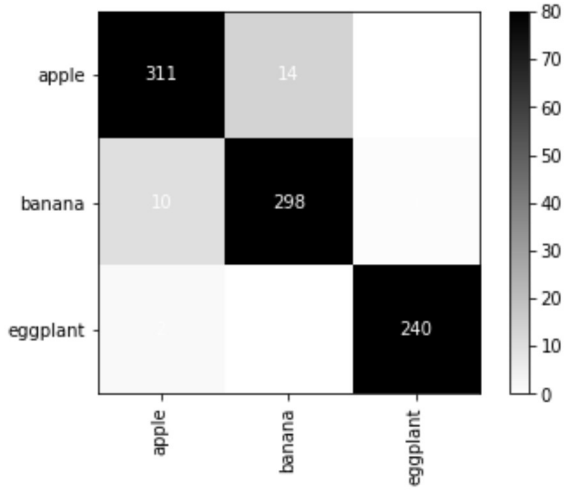

```
Computing Confusion Matrix
[[311  14   0]
 [ 10 298   1]
 [  2   1 239]]
```



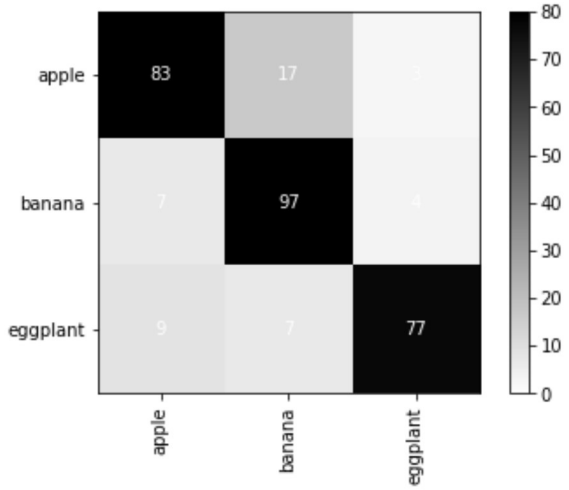

```
Computing Confusion Matrix
[[83 17  3]
 [ 7 97  4]
 [ 7  9 77]]
```




```
Computing Confusion Matrix
[[311  14   0]
 [ 10 298   1]
 [  2   0 240]]
```

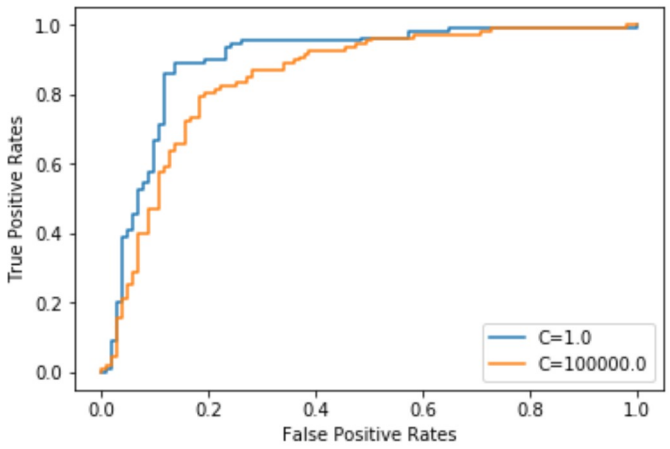



```
Computing Confusion Matrix
[[83 17  3]
 [ 7 97  4]
 [ 9  7 77]]
```



In [7]:
```
from numpy.random import uniform
import random
import time


import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA

import sys


from projection import Project2D, Projections
from confusion_mat import getConfusionMatrixPlot

from ridge_model import Ridge_Model
from qda_model import QDA_Model
from lda_model import LDA_Model
from svm_model import SVM_Model
from logistic_model import Logistic_Model

import matplotlib.pyplot as plt


CLASS_LABELS = ['apple','banana']




def compute_tp_fp(thres, scores, labels):
    scores = np.array(scores)
    prediction = (scores > thres)
    tp = np.sum(prediction * labels)
    tpr = 1.0 * tp / np.sum(labels)

    fp = np.sum(prediction * (1-labels))
    fpr = 1.0*fp / np.sum(1-labels)
    return tpr, fpr

def plot_ROC(tps, fps):
    # plot
    plt.plot(fps, tps)
    plt.ylabel("True Positive Rates")
    plt.xlabel("False Positive Rates")



def ROC(scores, labels):
    thresholds = sorted(np.unique(scores))
    thresholds = [-float("Inf")] + thresholds + [float("Inf")]
    tps = []
    fps = []

    # student code start here
    # TODO: Your code
    for thresh in thresholds:
        t,f = compute_tp_fp(thresh, scores, labels)
        tps.append(t)
        fps.append(f)
    # student code end here

    return tps, fps
```

```
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:433: ComplexW
arning: Casting complex values to real discards the imaginary part
  array = np.array(array, dtype=dtype, order=order, copy=copy)
```

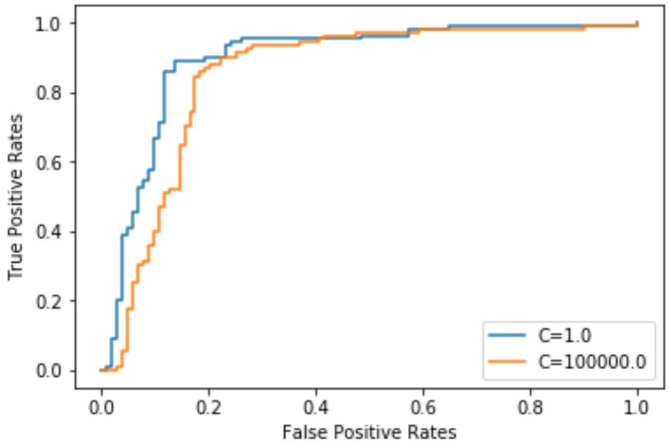


i (Observations)

The regularization weight C = 1.0 is better because for all points it seems as it is just as good or better than C = 100000.0 at reaching the goal of TPR = 1.0 and FPR = 0.0. An estimation of success could be the area under the curve because this measures accuracy and C = 1.0 has the larger area under the curve. Interpreting the graph, it seems like an increase in guessing positive at the beginning of the curve causes the TPR to have a greater increase relative to FPR when C = 1.0 compared to C = 100000.0, while for the rest of the curve the two are basically the same.

In [10]:
```
def eval_with_ROC(method, train_X, train_Y, val_X, val_Y, C):
    m = method(CLASS_LABELS)
    m.C = C
    m.train_model(train_X, train_Y)
    scores = m.scores(val_X)
    # change the scores here
    scores = 10.0 * np.array(scores)

    tps, fps = ROC(scores, val_Y)
    plot_ROC(tps, fps)

if __name__ == "__main__":

    # Load Training Data and Labels
    X = list(np.load('little_x_train.npy'))
    Y = list(np.load('little_y_train.npy'))
    X, Y = trim_data(X, Y)

    # Load Validation Data and Labels
    X_val = list(np.load('little_x_val.npy'))
    Y_val = list(np.load('little_y_val.npy'))
    X_val, Y_val = trim_data(X_val, Y_val)

    CLASS_LABELS = ['apple','banana']


    # Project Data to 200 Dimensions using CCA
    feat_dim = max(X[0].shape)
    projections = Projections(feat_dim,CLASS_LABELS)
    cca_proj,white_cov = projections.cca_projection(X,Y,k=2)

    X = projections.project(cca_proj,white_cov,X)
    X_val = projections.project(cca_proj,white_cov,X_val)


    ####RUN SVM REGRESSION#####
    eval_with_ROC(SVM_Model, X, Y, X_val, Y_val, 1.0)
    eval_with_ROC(SVM_Model, X, Y, X_val, Y_val, 100000.0)
    plt.legend(["C=1.0", "C=100000.0"])
    plt.show()
```

```
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:433: ComplexW
arning: Casting complex values to real discards the imaginary part
  array = np.array(array, dtype=dtype, order=order, copy=copy)
```



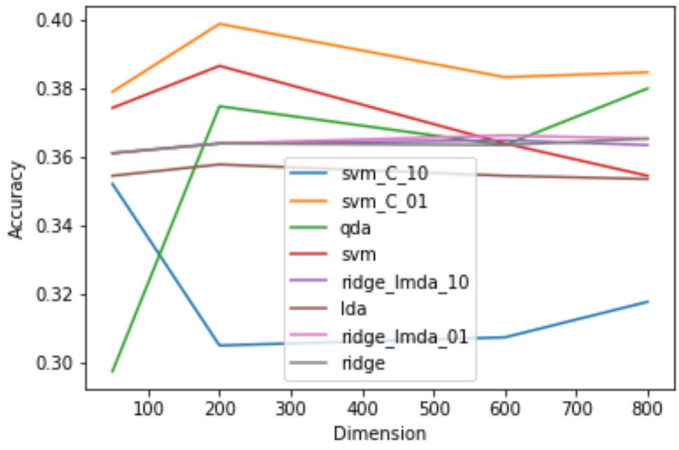
j (Observations)

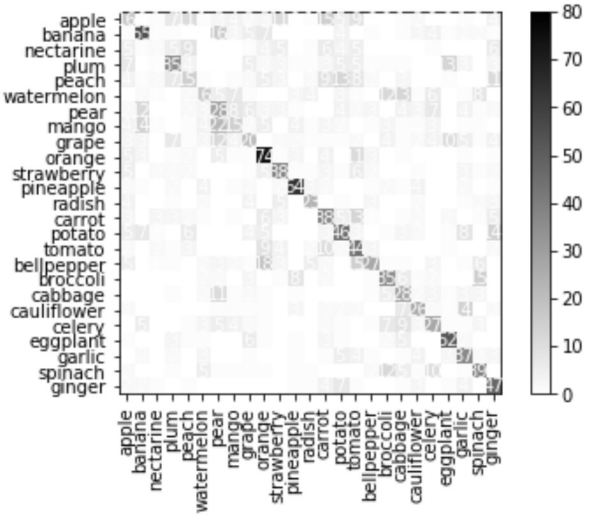
If you multiply the scores output by the classifier by a factor of 10.0, the ROC curve's general trend as well as the better C value (C = 1.0) does not change. Multiple trials shows that the graphs themselves change a little bit every time and a good portion of the time C=1.0 and C = 100000.0 are very similar.

3.k

In [12]:
```
import cv2
import IPython
from numpy.random import uniform
import random
import time


import glob
import os
import pickle
import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA

import sys


from projection import Project2D, Projections

from confusion_mat import getConfusionMatrix
from confusion_mat import plotConfusionMatrix

from ridge_model import Ridge_Model
from qda_model import QDA_Model
from lda_model import LDA_Model
from svm_model import SVM_Model


class LDA_Model():

    def __init__(self,class_labels):

        ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED COVARIANCE MAT
RIX TO PREVENT IT BEING SINGULAR ###
        self.reg_cov = 0.001
        self.NUM_CLASSES = len(class_labels)
        self.Xs = {}
        self.mean = {}
        #self.cov = None
        self.cov_inv = None



    def train_model(self,X,Y):
        ''''
        FILL IN CODE TO TRAIN MODEL
        MAKE SURE TO ADD HYPERPARAMTER TO MODEL

        '''
        n = len(Y)
        for i in range(0, self.NUM_CLASSES):
            self.Xs[i] = []

        for i in range(n):
            # Y is a single class number, not one-hot encoded
            self.Xs[Y[i]].append(X[i])

        for x, arr in self.Xs.items():
            self.mean[x] = np.mean(np.array(arr), axis = 0).reshape((-1,1))
        np_X = np.array(X)
        # self.cov = compute_covariance_matrix(np_X, np_X) + self.reg_cov*np.eye(np
_X.shape[1])
        self.cov_inv = np.linalg.inv(compute_covariance_matrix(np_X, np_X) + self.r
```

```
svm_C_10
k = 50
k = 200
k = 600
k = 800
svm_C_01
k = 50
k = 200
k = 600
k = 800
qda
k = 50
k = 200
k = 600
k = 800
svm
k = 50
k = 200
k = 600
k = 800
ridge_lmda_10
k = 50
k = 200
k = 600
k = 800
lda
k = 50
k = 200
k = 600
k = 800
ridge_lmda_01
k = 50
k = 200
k = 600
k = 800
ridge
k = 50
k = 200
k = 600
k = 800
```



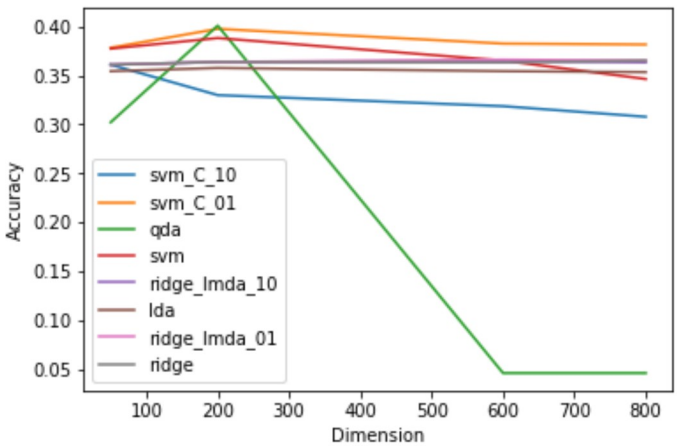

apple
banana
nectarine
plum
peach
watermelon
pear
mango
grape
orange
strawberry
pineapple
radish
carrot
potato
tomato
bellpepper
broccoli
cabbage
cauliflower
celery
eggplant
garlic
spinach
ginger
80
70
60
50
40
30
20
10
0

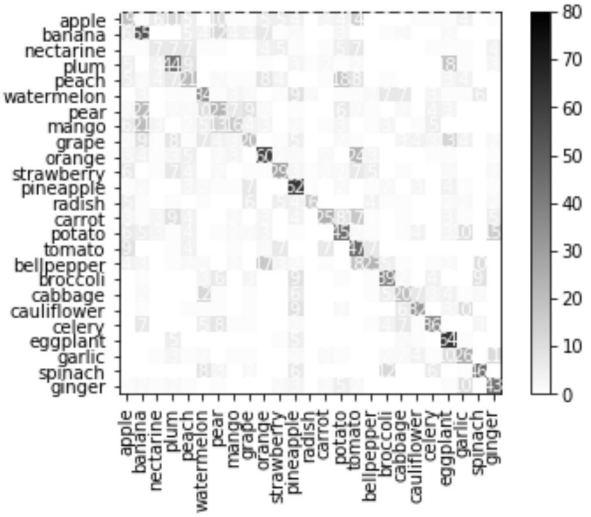
In [28]:
```
import cv2
import IPython
from numpy.random import uniform
import random
import time


import glob
import os
import pickle
import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA

import sys


from projection import Project2D, Projections

from confusion_mat import getConfusionMatrix
from confusion_mat import plotConfusionMatrix

from ridge_model import Ridge_Model
from qda_model import QDA_Model
from lda_model import LDA_Model
from svm_model import SVM_Model


class LDA_Model():

    def __init__(self,class_labels):

        ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED COVARIANCE MAT
RIX TO PREVENT IT BEING SINGULAR ###
        self.reg_cov = 0.001
        self.NUM_CLASSES = len(class_labels)
        self.Xs = {}
        self.mean = {}
        #self.cov = None
        self.cov_inv = None



    def train_model(self,X,Y):
        ''''
        FILL IN CODE TO TRAIN MODEL
        MAKE SURE TO ADD HYPERPARAMTER TO MODEL

        '''
        n = len(Y)
        for i in range(0, self.NUM_CLASSES):
            self.Xs[i] = []

        for i in range(n):
            # Y is a single class number, not one-hot encoded
            self.Xs[Y[i]].append(X[i])

        for x, arr in self.Xs.items():
            self.mean[x] = np.mean(np.array(arr), axis = 0).reshape((-1,1))
        np_X = np.array(X)
        # self.cov = compute_covariance_matrix(np_X, np_X) + self.reg_cov*np.eye(np
_X.shape[1])
        self.cov_inv = np.linalg.inv(compute_covariance_matrix(np_X, np_X) + self.r
```

```
svm_C_10
k = 50
k = 200
k = 600
k = 800
svm_C_01
k = 50
k = 200
k = 600
k = 800
qda
k = 50
k = 200
k = 600
```

```
/home/nlorio/.local/lib/python3.5/site-packages/ipykernel_launcher.py:118: Runti
meWarning: divide by zero encountered in log
```

```
k = 800
svm
k = 50
k = 200
k = 600
k = 800
ridge_lmda_10
k = 50
k = 200
k = 600
k = 800
lda
k = 50
k = 200
k = 600
k = 800
ridge_lmda_01
k = 50
k = 200
k = 600
k = 800
ridge
k = 50
k = 200
k = 600
k = 800
```

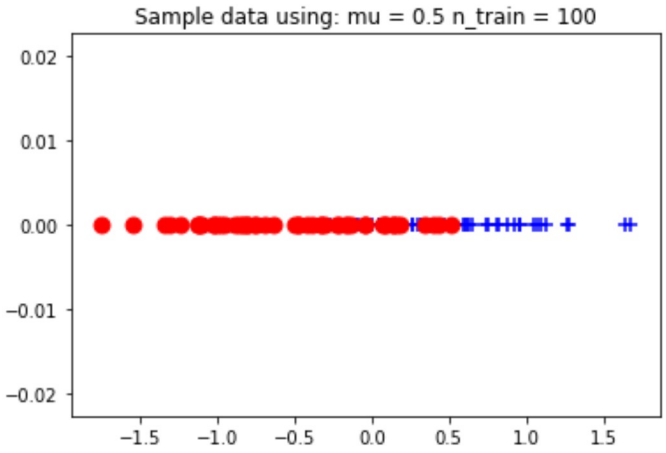


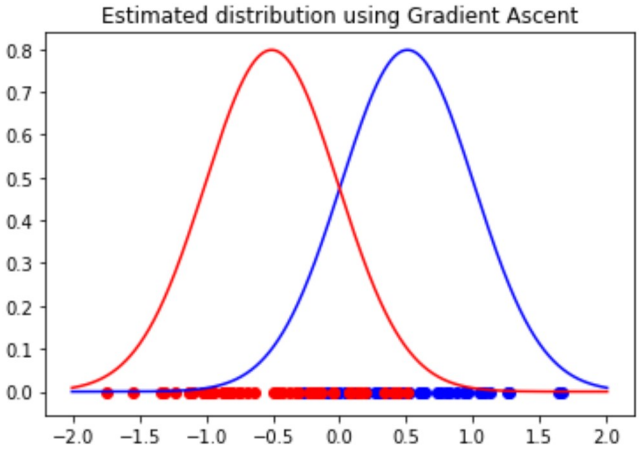


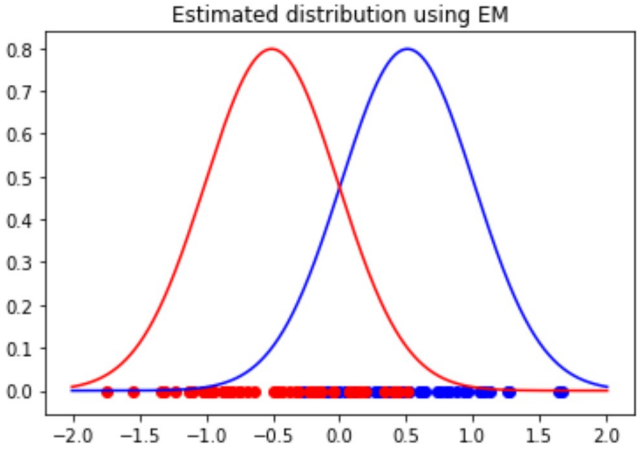
Question 4

4.m

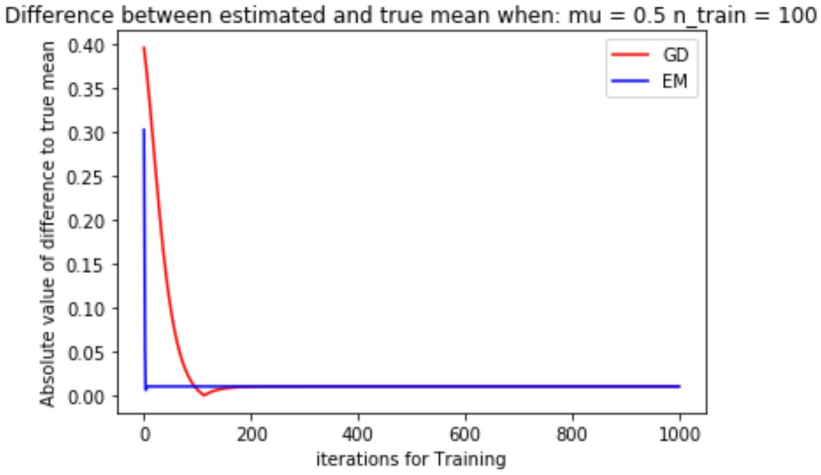
In [26]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

def generate_data(mu_1, mu_2, n_examples, sigma_1 = 0.5, sigma_2 = 0.5):
    # Generate sample data points from 2 distributions: (mu_1, sigma_1) and (mu_2,
sigma_2)
    d_1 = np.random.normal(mu_1, sigma_1, n_examples)
    d_2 = np.random.normal(mu_2, sigma_2, n_examples)
    x = np.concatenate([d_1, d_2])
    return d_1, d_2, x

def plot_data(d_1, d_2):
    # Plot scatter plot of data samples, labeled by class
    plt.figure()
    plt.scatter(d_1, np.zeros(len(d_1)), c='b', s=80., marker='+')
    plt.scatter(d_2, np.zeros(len(d_2)), c='r', s=80.)
    plt.title("Sample data using: mu = " + str(mu) + " n_train = " + str(len(d_1)+l
en(d_2)))
    plt.show()
    return

def plot_data_and_distr(d_1, d_2, mu_1, mu_2, sigma_1=0.5, sigma_2=0.5, title = "")
:
    # Plot scatter plot of data samples overlayed with distribution of estimated me
ans: mu_1 and mu_2
    plt.scatter(d_1, np.zeros(len(d_1)), c='b')
    scale = [min(mu_1-3*sigma_1, mu_2-3*sigma_2), max(mu_1+3*sigma_1, mu_2+3*sigma_
2)]
    plt.scatter(d_2, np.zeros(len(d_2)), c='r')
    x_axis = np.arange(scale[0], scale[1], 0.001)
    plt.plot(x_axis, norm.pdf(x_axis,mu_1, sigma_1), c='b')
    x_axis = np.arange(scale[0], scale[1], 0.001)
    plt.plot(x_axis, norm.pdf(x_axis,mu_2,sigma_2), c='r')
    plt.title(title)
    plt.show()


def grad_ascent(x, mu, mu_true, sigma = 0.5, iterations = 1000):
    # Run gradient ascent on the likelihood of a point belonging to a class and com
pare the estimates of the mean
    #     at each iteration with the true mean of the distribution
    # Note: the original dataset comes from 2 distributions centered at mu and -mu,
which this takes into account
    #     with each update
    diff_mu = []
    alpha = 0.05
    for i in range(iterations):
        phi_1 = np.exp(-np.square(x-mu)/(2*np.square(sigma)))
        phi_2 = np.exp(-np.square(x+mu)/(2*np.square(sigma)))
        w = phi_1/(phi_1 + phi_2)
        em = (1/len(x))*np.sum((2*w - 1)*x)
        mu = mu*(1-alpha) + alpha*em
        diff_mu.append(np.abs(mu-mu_true))
    return mu, sigma,diff_mu


def em(x, mu, mu_true, sigma = 0.5, iterations = 1000):
    # Run the EM algorithm to find the estimated mean of the dataset
    # Note: the original dataset comes from 2 distributions centered at mu and -mu,
which this takes into account
    #     with each update
    diff_mu = np.zeros(iterations)
    for i in range(iterations):
```

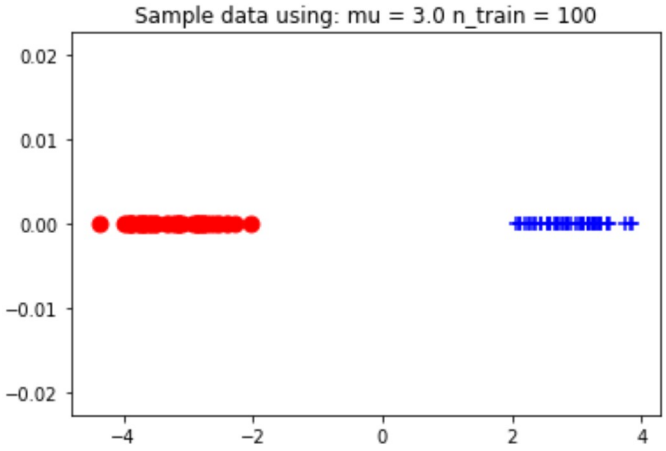


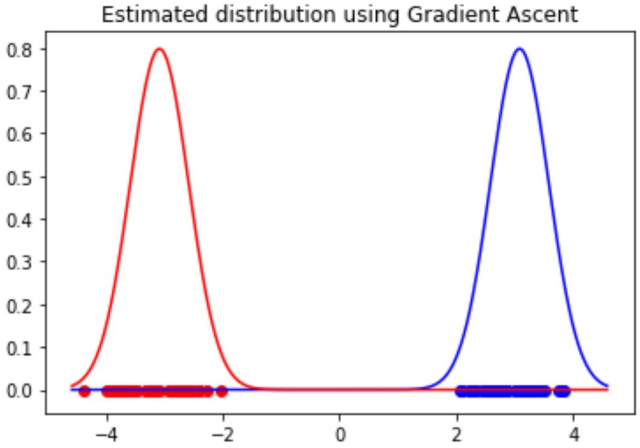


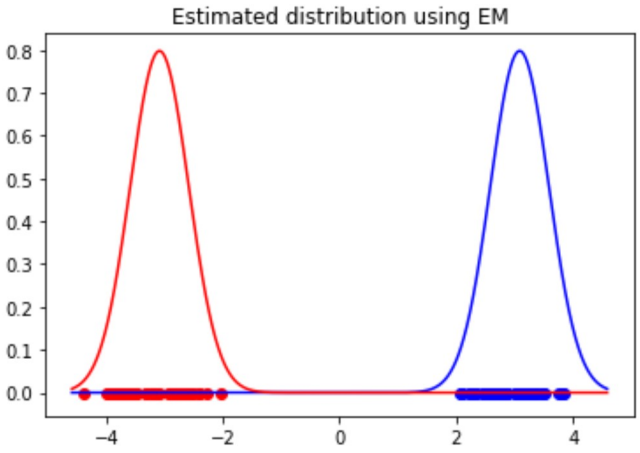



```
------------
n_points: 100 , True mean:0.500, GA (final) estimate:0.510, EM (final) estimate:
0.510
------------
```

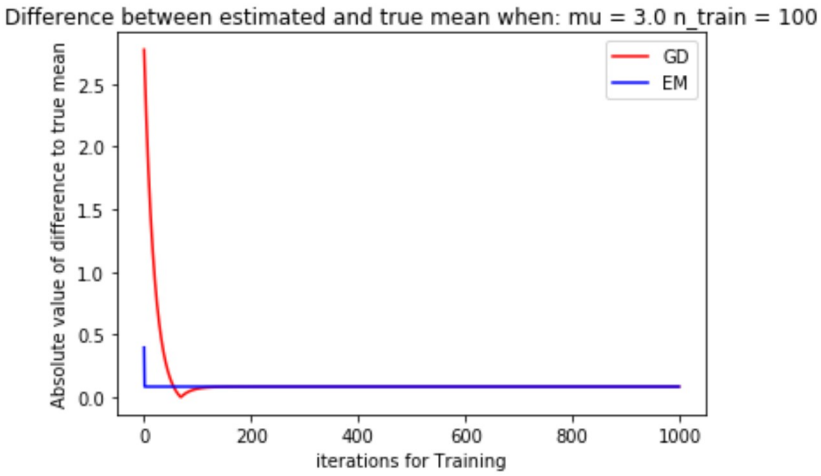

Difference between estimated and true mean when: mu = 0.5 n_train = 100
Absolute value of difference to true mean
iterations for Training
GD
EM
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
200
400
600
800
1000


Sample data using: mu = 3.0 n_train = 100
0.02
0.01
0.00
−0.01
−0.02
−4
−2
0
2
4


Estimated distribution using Gradient Ascent
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−4
−2
0
2
4




```
------------
n_points: 100 , True mean:3.000, GA (final) estimate:3.085, EM (final) estimate:
3.085
------------
```



4.m Observations

Similarities: Both reach the final estimation succesfully.

Differences: EM converges to the final estimation faster and also does not overshoot the target estimation or fluctuate around it.

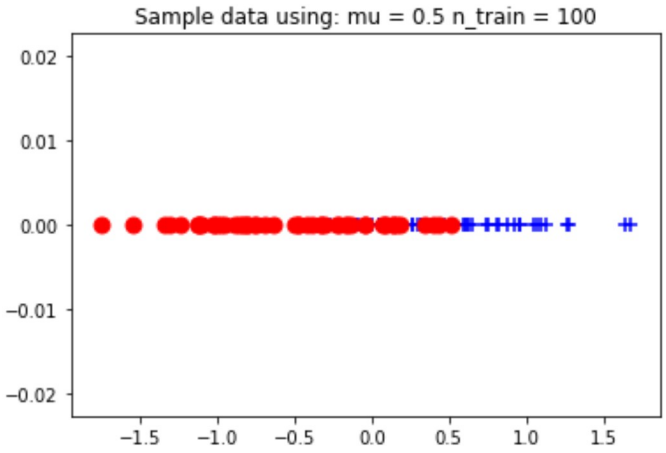
From part (l) we derived the gradient descent algorithm by taking the derivative of the EM algorithm and setting it to 0. We showed that the EM algorithm and the Gradient Ascent algorithm are equivalent when we scale by (1/n). The updates derived in the previous parts show how similar the two algorithms are.

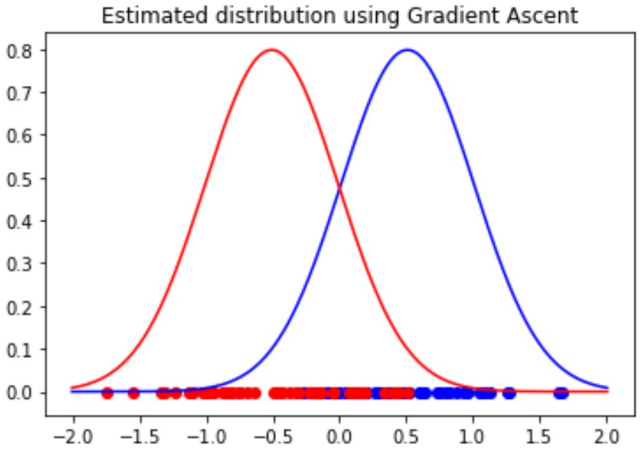
From our plots above we see that EM and Gradient Ascet both create very similar distributions and have the same estimates for the mean.

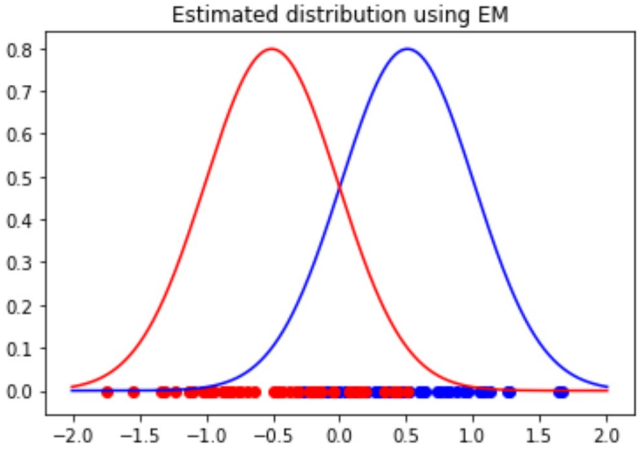
EM appears to be the better option for both of the cases, near and far, plotted above. It converges faster and does not fluctuate. EM always updates Q and theta to make the free energy a closer and closer bound to the true log likelihood estimate while gradient descent uses derivatives to find the direction of increase to approach the likelihood, so it makes sense the both converge.

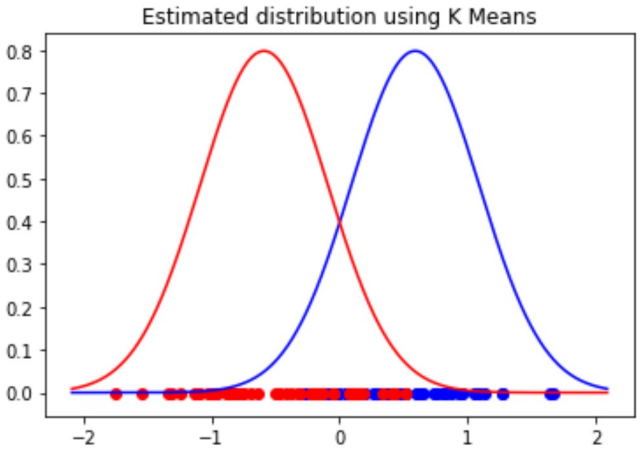
As well, EM does not depend on the hyper parameter alpha in order to succeed.The plots complement the similarity of the two algorithms we saw in the previous two parts.

In [27]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

def generate_data(mu_1, mu_2, n_examples, sigma_1 = 0.5, sigma_2 = 0.5):
    # Generate sample data points from 2 distributions: (mu_1, sigma_1) and (mu_2,
sigma_2)
    d_1 = np.random.normal(mu_1, sigma_1, n_examples)
    d_2 = np.random.normal(mu_2, sigma_2, n_examples)
    x = np.concatenate([d_1, d_2])
    return d_1, d_2, x

def plot_data(d_1, d_2):
    # Plot scatter plot of data samples, labeled by class
    plt.figure()
    plt.scatter(d_1, np.zeros(len(d_1)), c='b', s=80., marker='+')
    plt.scatter(d_2, np.zeros(len(d_2)), c='r', s=80.)
    plt.title("Sample data using: mu = " + str(mu) + " n_train = " + str(len(d_1)+l
en(d_2)))
    plt.show()
    return

def plot_data_and_distr(d_1, d_2, mu_1, mu_2, sigma_1=0.5, sigma_2=0.5, title = "")
:
    # Plot scatter plot of data samples overlayed with distribution of estimated me
ans: mu_1 and mu_2
    plt.scatter(d_1, np.zeros(len(d_1)), c='b')
    scale = [min(mu_1-3*sigma_1, mu_2-3*sigma_2), max(mu_1+3*sigma_1, mu_2+3*sigma_
2)]
    plt.scatter(d_2, np.zeros(len(d_2)), c='r')
    x_axis = np.arange(scale[0], scale[1], 0.001)
    plt.plot(x_axis, norm.pdf(x_axis,mu_1, sigma_1), c='b')
    x_axis = np.arange(scale[0], scale[1], 0.001)
    plt.plot(x_axis, norm.pdf(x_axis,mu_2,sigma_2), c='r')
    plt.title(title)
    plt.show()


def grad_ascent(x, mu, mu_true, sigma = 0.5, iterations = 1000):
    # Run gradient ascent on the likelihood of a point belonging to a class and com
pare the estimates of the mean
    #     at each iteration with the true mean of the distribution
    # Note: the original dataset comes from 2 distributions centered at mu and -mu,
which this takes into account
    #     with each update
    diff_mu = []
    alpha = 0.05
    for i in range(iterations):
        phi_1 = np.exp(-np.square(x-mu)/(2*np.square(sigma)))
        phi_2 = np.exp(-np.square(x+mu)/(2*np.square(sigma)))
        w = phi_1/(phi_1 + phi_2)
        em = (1/len(x))*np.sum((2*w - 1)*x)
        mu = mu*(1-alpha) + alpha*em
        diff_mu.append(np.abs(mu-mu_true))
    return mu, sigma,diff_mu


def em(x, mu, mu_true, sigma = 0.5, iterations = 1000):
    # Run the EM algorithm to find the estimated mean of the dataset
    # Note: the original dataset comes from 2 distributions centered at mu and -mu,
which this takes into account
    #     with each update
    diff_mu = np.zeros(iterations)
    for i in range(iterations):
```

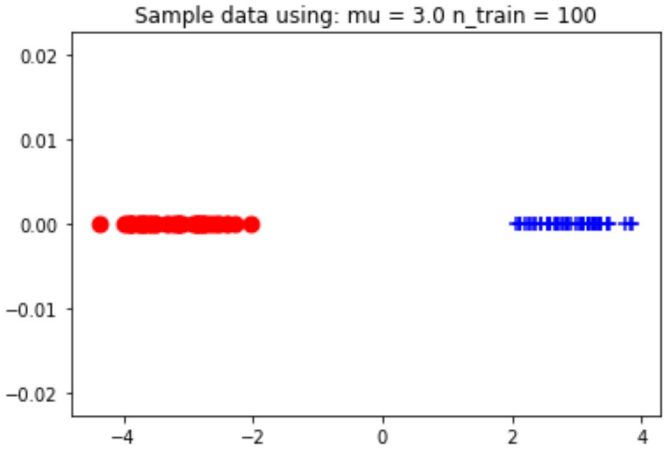

Sample data using: mu = 0.5 n_train = 100
0.02
0.01
0.00
−0.01
−0.02
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

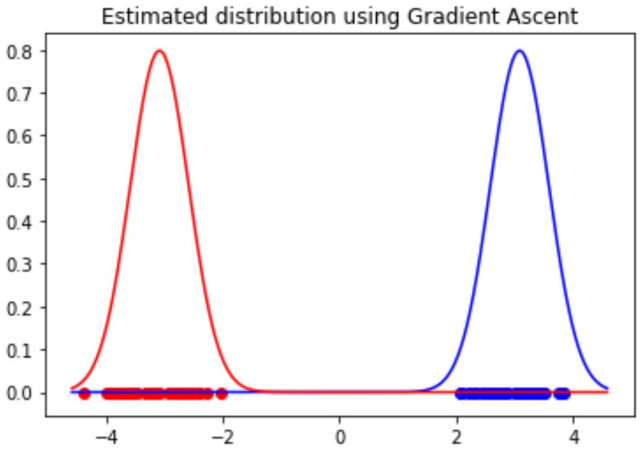

Estimated distribution using Gradient Ascent
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

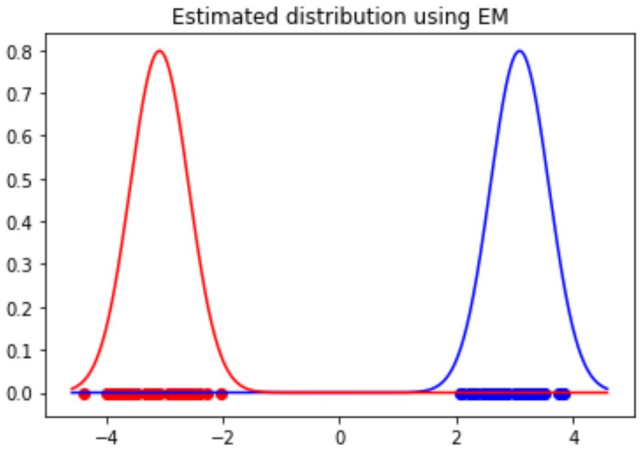

Estimated distribution using EM
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

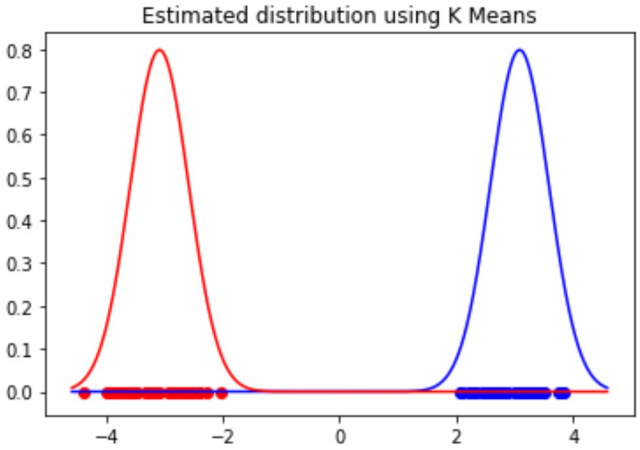



```
------------
True mean:0.500, GA (final) estimate:0.510, EM (final) estimate:0.510, K-Means (
final) estimate:0.591
------------
```










```
------------
True mean:3.000, GA (final) estimate:3.085, EM (final) estimate:3.085, K-Means (
final) estimate:3.085
------------
```

4.n Observations

The final estimates for EM and GA are the same as before. K means has a slightly higher final estiamte when the two mixtures from the data are close. However when the two mixtures from the data are farther apart than the final estimates, estimated distributions for all three methods are the same. K means depends on whether or not Mu is large or small.

To conclude, EM converges to a good estimates of Mu faster than GA. GA is a weighted version of EM, it is slower than EM. K means will perform differently based off of the size of mu.

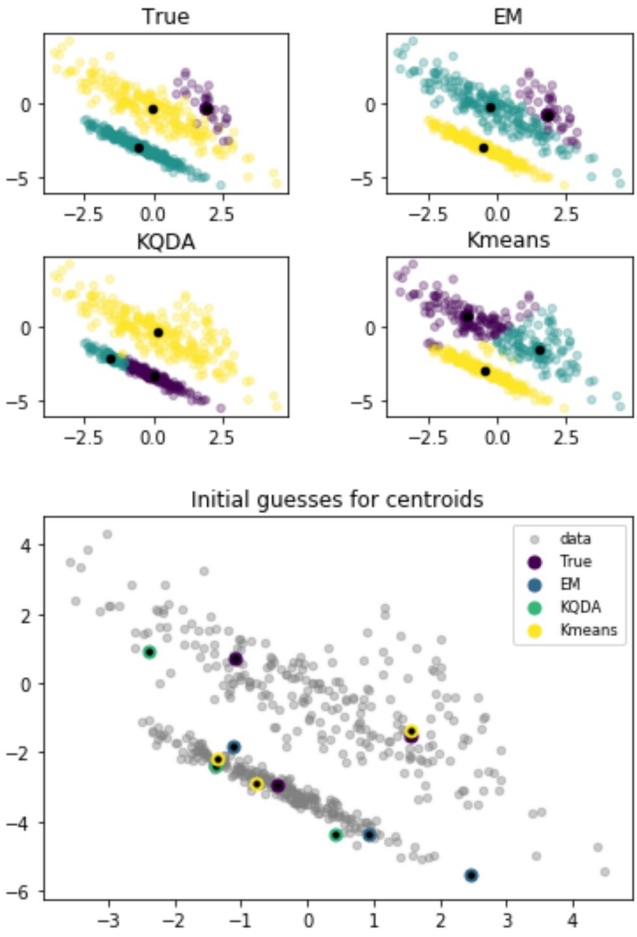
Question 5

In [13]:
```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
import os

#This function generate random mean and covariance
def gauss_params_gen(num_clusters, num_dims, factor):
    mu = np.random.randn(num_clusters,num_dims)*factor
    sigma = np.random.randn(num_clusters,num_dims,num_dims)
    for k in range(num_clusters):
        sigma[k] = np.dot(sigma[k],sigma[k].T)

    return (mu, sigma)

#Given mean and covariance generate data
def data_gen(mu, sigma, num_clusters, num_samples):
    labels = []
    X = []
    cluster_prob = np.array([np.random.rand() for k in  range(num_clusters)])
    cluster_num_samples = (num_samples * cluster_prob / sum(cluster_prob)).astype(i
nt)
    cluster_num_samples[-1] = num_samples-sum(cluster_num_samples[:-1])

    for k, ks in enumerate(cluster_num_samples):
        labels.append([k]*ks)
        X.append(np.random.multivariate_normal(mu[k], sigma[k], ks))

    # shuffle data
    randomize = np.arange(num_samples)
    np.random.shuffle(randomize)
    X =  np.vstack(X)[randomize]
    labels =  np.array(sum(labels,[]))[randomize]

    return X, labels


def data2D_plot(ax, x, labels, centers, cmap, title):
    data = {'x0': x[:,0], 'x1': x[:,1], 'label': labels}
    ax.scatter(data['x0'], data['x1'], c=data['label'], cmap=cmap, s=20, alpha=0.3)
    ax.scatter(centers[:, 0], centers[:, 1], c=np.arange(np.shape(centers)[0]), cma
p=cmap, s=50, alpha=1)
    ax.scatter(centers[:, 0], centers[:, 1], c='black', cmap=cmap, s=20, alpha=1)
    ax.title.set_text(title)

def plot_init_means(x, mus, algs, fname):
    import matplotlib.cm as cm
    fig = plt.figure()
    plt.scatter(x[:,0], x[:,1], c='gray', cmap='viridis', s=20, alpha= 0.4, label='
data')
    for mu, alg, clr in zip(mus, algs, cm.viridis(np.linspace(0, 1, len(mus)))):
        plt.scatter(mu[:,0], mu[:, 1], c=clr, s=50, label=alg)
        plt.scatter(mu[:, 0], mu[:, 1], c='black', s=10, alpha=1)
    legend = plt.legend(loc='upper right', fontsize='small')
    plt.title('Initial guesses for centroids')
    fig.savefig(fname)

def loss_plot(loss, title, xlabel, ylabel, fname):
        fig = plt.figure(figsize = (13, 6))
        plt.plot(np.array(loss))
        plt.title(title)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        fig.savefig(fname)
```

5.a

In [14]:
```
if __name__ == "__main__":
    # Question asks us to run with factor =1. Error when factor=1.
    experiments(seed=11, factor=1.1, num_samples=500, num_clusters=3)
```

```
iteration 1, update 0.0002
iteration 2, update 13.08859979497106
iteration 3, update 17.994530630958934
iteration 4, update 23.517668049172244
iteration 5, update 32.24912506245573
iteration 6, update 47.421343999307055
iteration 7, update 66.5282555182323
iteration 8, update 66.67704792032646
iteration 9, update 62.34033482167979
iteration 10, update 104.07268668725965
iteration 11, update 208.00335186351208
iteration 12, update 198.86335964955526
iteration 13, update 7.972945310798536
iteration 14, update 6.526079715927608
iteration 15, update 9.197308405457079
iteration 16, update 11.654578087263644
iteration 17, update 13.279245372021592
iteration 18, update 12.081000929005313
iteration 19, update 8.263932660612909
iteration 20, update 4.248422932323479
iteration 21, update 1.5676814379073676
iteration 22, update 0.2580425866897258
iteration 23, update 0.21430090602359542
iteration 24, update 0.22461382893322934
iteration 25, update 0.04855756544407086
iteration 26, update 0.5304205984596138
iteration 27, update 1.2759998227860478
iteration 28, update 2.3597993114393603
iteration 29, update 3.4834995675421396
iteration 30, update 4.134570443733196
iteration 31, update 4.229203147858129
iteration 32, update 3.886384749251306
iteration 33, update 3.214327134529526
iteration 34, update 2.4554977658407324
iteration 35, update 1.8375454957672446
iteration 36, update 1.422556352714878
iteration 37, update 1.1717009553644857
iteration 38, update 1.0266410736802527
iteration 39, update 0.9400726621705644
iteration 40, update 0.8806849991213994
iteration 41, update 0.8305338860971005
iteration 42, update 0.7809723291804858
iteration 43, update 0.729029574689207
iteration 44, update 0.674740655493224
iteration 45, update 0.6194138236559183
iteration 46, update 0.5646382311996376
iteration 47, update 0.5117995010275536
iteration 48, update 0.4619041250271039
iteration 49, update 0.41556871349939684
iteration 50, update 0.37308235660111677
iteration 51, update 0.3344900965245188
iteration 52, update 0.29967181276435895
iteration 53, update 0.2684063982430871
iteration 54, update 0.24041934390504593
iteration 55, update 0.2154155920567291
iteration 56, update 0.19310076556507738
iteration 57, update 0.173193893799521
iteration 58, update 0.1554342658114365
iteration 59, update 0.13958442631053458
iteration 60, update 0.12543076456267954
iteration 61, update 0.11278269073432057
iteration 62, update 0.10147105495912001
iteration 63, update 0.09134622428018702
iteration 64, update 0.08227606980244673
```



Seed: Determines randomness. Factor: Determines how far apart the clusters are.

The initial guesses for centroids of all the methods are determined randomly.

Expectation maximization clearly performs the best. In this simulation, EM correctly determines cluster membership.

KQDA, a variant of the hard EM method, does not fully determine cluster membership correctly.

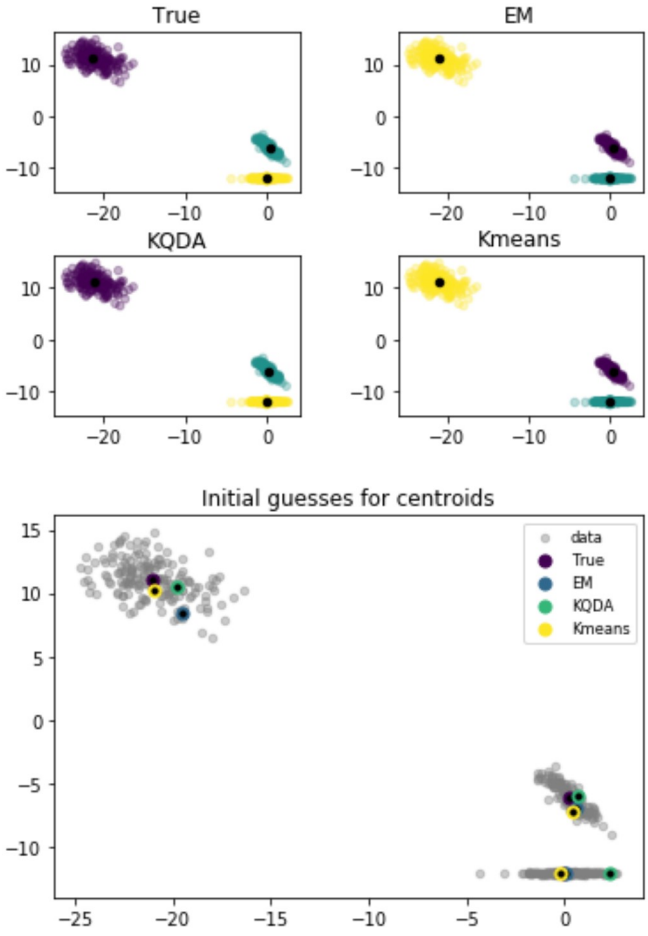
Kmeans does not fully determine cluster membership correctly.

EM works better on data sets that actually have gaussian clusters, it will typically perform much better on these forms of data sets than on those that dont provide a good fit at all. If there is not a good fit, there will be high variance in runtime.

5.b

In [9]:
```
if __name__ == "__main__":
    experiments(seed=63, factor=10, num_samples=500, num_clusters=3)
```

```
iteration 1, update 0.0002
iteration 2, update 26.950255622375835
iteration 3, update 1.820012585085351e-09
iteration 1, update 2e-05
iteration 2, update 0.1596065015345911
iteration 3, update 0.0
iteration 1, update 2e-05
iteration 2, update 0.0
```



Seed: Determines randomness Factor: Determines how far apart the clusters are

Increasing factor size increases the distance between the clusters. In this case, all methods for classification correctly determine cluster membership. EM has the fastest convergence, when there is not too much data missing, therefore it is the preferred method.

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4433949/ (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4433949/)