Table of Contents

In [173]:
```
%%javascript
/*********************************************************************************
*************
Known Mathjax Issue with Chrome - a rounding issue adds a border to the right of
mathjax markup
https://github.com/mathjax/MathJax/issues/1300
A quick hack to fix this based on stackoverflow discussions:
http://stackoverflow.com/questions/34277967/chrome-rendering-mathjax-equations-wi
th-a-trailing-vertical-line
*********************************************************************************
************/

$('.math>span').css("border-left-color","transparent")
```

In [174]:
```
%reload_ext autoreload
%autoreload 2
```

MIDS - w261 Machine Learning At Scale

Course Lead: Dr James G. Shanahan (**email** Jimi via James.Shanahan *AT* gmail.com)

Assignment - HW10

Name: Nilesh Bhoyar
Class: MIDS w261 (Section *Your Section Goes Here*, e.g., Fall 2016 Group 1)
Email: nilesh.bhoyar@iSchool.Berkeley.edu
StudentId 26302327 **End of StudentId**
Week: 10

NOTE: please replace `1234567` with your student id above

Due Time: HW is due the Tuesday of the following week by 8AM (West coast time). I.e., Tuesday, April 4, 2017 in the case of this homework.

Instructions

MIDS UC Berkeley, Machine Learning at Scale
DATSCIW261 ASSIGNMENT #10

Version 2017-3-24

IMPORTANT

This homework can be completed locally on your computer.

=== INSTRUCTIONS for SUBMISSIONS ===

Follow the instructions for submissions carefully.

Each student has a `HW-<user>` repository for all assignments.

Click this link to enable you to create a github repo within the MIDS261 Classroom:
https://classroom.github.com/assignment-invitations/3b1d6c8e58351209f9dd865537111ff8 (https://classroom.github.com/assignment-invitations/3b1d6c8e58351209f9dd865537111ff8)
and follow the instructions to create a HW repo.

Push the following to your HW github repo into the master branch:

- Your local HW6 directory. Your repo file structure should look like this:

```
HW-<user>
    --HW3
       |__MIDS-W261-HW-03-<Student_id>.ipynb
       |__MIDS-W261-HW-03-<Student_id>.pdf
       |__some other hw3 file
    --HW4
       |__MIDS-W261-HW-04-<Student_id>.ipynb
       |__MIDS-W261-HW-04-<Student_id>.pdf
       |__some other hw4 file
    etc..
```

Useful References

- Karau, Holden, Konwinski, Andy, Wendell, Patrick, & Zaharia, Matei. (2015). Learning Spark: Lightning-fast big data analysis. Sebastopol, CA: O'Reilly Publishers.
- Hastie, Trevor, Tibshirani, Robert, & Friedman, Jerome. (2009). The elements of statistical learning: Data mining, inference, and prediction (2nd ed.). Stanford, CA: Springer Science+Business Media. (Download for free here (http://statweb.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf))

HW Problems

HW10.0: Short answer questions

What is Apache Spark and how is it different to Apache Hadoop?

Fill in the blanks:

__Spark API consists of interfaces to develop applications based on it in Java, R, Scala, and Python languages (list languages).

__Using Spark, resource management can be done either in a single server instance or using a framework such as Mesos or Hadoop Yarn in a distributed manner and a simple cluster manager included in Spark itself called the Standalone Scheduler.

What is an RDD and show a fun example of creating one and bringing the first element back to the driver program.

```
RDD stands for resilient distributed dataset in Spark is simply an immutable distribut
ed collection of objects. Each RDD is split into multiple partitions, which may be com
puted on different nodes of the cluster. RDDs can contain any type of Python, Java, or
Scala objects, including user-defined classes.
```

In [175]:
```
data = [1,2,3,4]
rd = sc.parallelize(data)
print rd.take(1)
```

```
[1]
```

HW10.1 WordCount plus sorting

The following notebooks will be useful to jumpstart this collection of Homework exercises:

- Example Notebook with Debugging tactics in Spark (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/jqjllp8kmf1eolk/WordCountDebugging-Example.ipynb)
- Word Count Quiz (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/vgmpivsi4rvqz0s/WordCountQuiz.ipynb)
- Work Count Solution (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/dxv3dmp1vluuo8i/WordCountQuiz-Solution.ipynb)

In Spark write the code to count how often each word appears in a text document (or set of documents). Please use this homework document (with no solutions in it) as a the example document to run an experiment. Report the following:

- provide a sorted list of tokens in decreasing order of frequency of occurence limited to [top 20 most frequent only] and [bottom 10 least frequent].

OPTIONAL Feel free to do a secondary sort where words with the same frequncy are sorted alphanumerically increasing. Please refer to the following notebook (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/uu5afr3ufpm9fy8/SecondarySort.ipynb) for examples of secondary sorts in Spark. Please provide the following [top 20 most frequent terms only] and [bottom 10 least frequent terms]

NOTE [Please incorporate all referenced notebooks directly into this master notebook as cells for HW submission. I.e., HW submissions should comprise of just one notebook]__

In [176]:
```
!hdfs dfs -rm -r HW10
!hdfs dfs -mkdir HW10
!hdfs dfs -copyFromLocal Template.txt HW10/
```

```
17/07/23 12:43:54 INFO fs.TrashPolicyDefault: Namenode trash configuration: Dele
tion interval = 5760 minutes, Emptier interval = 360 minutes.
Moved: 'hdfs://nn-ia.s3s.altiscale.com:8020/user/nileshbhoyar/HW10' to trash at:
hdfs://nn-ia.s3s.altiscale.com:8020/user/nileshbhoyar/.Trash/Current
```

In [177]:
```
# START STUDENT CODE 10.1
# (ADD CELLS AS NEEDED)
!hdfs dfs -rm -r HW10/HW100/
import re

WORD_RE = re.compile(r"[\w']+")

logFileNAME = 'HW10/Template.txt'
text_file = sc.textFile(logFileNAME)
counts = text_file.flatMap(lambda line: re.findall(r"[\w']+",line)) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b, 1) \
             .map(lambda (a, b): (b, a)) \
             .sortByKey(0,3)  \
             .map(lambda (a, b): (b, a))

counts.saveAsTextFile("HW10/HW100/")
wordCounts = counts.collect()
print "Top 20 Most Frequent"
for i in wordCounts[0:20]:
    print i
print "Bottom 10 in  Most Frequent"
for i in wordCounts[-10:]:
    print i
# END STUDENT CODE 10.1
```

```
rm: `HW10/HW100/': No such file or directory
Top 20 Most Frequent
(u'n', 231)
(u'the', 77)
(u'a', 73)
(u'toc', 59)
(u'HW10', 55)
(u'1', 47)
(u'item', 38)
(u'div', 38)
(u'nbsp', 38)
(u'class', 38)
(u'span', 38)
(u'and', 37)
(u'of', 36)
(u'metadata', 35)
(u'data', 34)
(u'cell_type', 33)
(u'in', 33)
(u'source', 33)
(u'10', 30)
(u'KMeans', 26)
Bottom 10 in  Most Frequent
(u'Stanford', 1)
(u'April', 1)
(u'structure', 1)
(u'textFile', 1)
(u'gradient', 1)
(u'write', 1)
(u'image', 1)
(u'time', 1)
(u'directory', 1)
(u'2009', 1)
```

HW10.1.1

Modify the above word count code to count words that begin with lower case letters (a-z) and report your findings. Again sort the output words in decreasing order of frequency.

In [178]:
```
# START STUDENT CODE 10.1.1
# (ADD CELLS AS NEEDED)
!hdfs dfs -rm -r HW10/HW101/
import re



logFileNAME = 'HW10/Template.txt'
text_file = sc.textFile(logFileNAME)
counts = text_file.flatMap(lambda line: re.findall(r'\b[a-z][a-z]*\b',line)) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b, 1) \
             .map(lambda (a, b): (b, a)) \
             .sortByKey(0,3)  \
             .map(lambda (a, b): (b, a))

counts.saveAsTextFile("HW10/HW101/")
wordCounts = counts.collect()
print "Top 20 Most Frequent"
for i in wordCounts[0:20]:
    print i
print "Bottom 10 in  Most Frequent"
for i in wordCounts[-10:]:
    print i
# END STUDENT CODE 10.1.1
```

```
rm: `HW10/HW101/': No such file or directory
Top 20 Most Frequent
(u'n', 231)
(u'the', 77)
(u'a', 73)
(u'toc', 59)
(u'span', 40)
(u'div', 38)
(u'nbsp', 38)
(u'item', 38)
(u'class', 38)
(u'and', 37)
(u'of', 36)
(u'metadata', 35)
(u'data', 34)
(u'in', 33)
(u'source', 33)
(u'code', 25)
(u'this', 24)
(u'to', 23)
(u'for', 21)
(u'href', 20)
Bottom 10 in  Most Frequent
(u'computing', 1)
(u'no', 1)
(u'drivers', 1)
(u'application', 1)
(u'branch', 1)
(u'test', 1)
(u'separate', 1)
(u'structure', 1)
(u'time', 1)
(u'directory', 1)
```

HW10.2: MLlib-centric KMeans

Using the following MLlib-centric KMeans code snippet:

```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt


# Load and parse the data
# NOTE  kmeans_data.txt is available here
#          https://www.dropbox.com/s/q85t0ytb9apggnh/kmeans_data.txt?dl=0
data = sc.textFile("kmeans_data.txt")
parsedData = data.map(lambda line: array([float(x) for x in line.split(' ')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
        runs=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))

# Save and load model
clusters.save(sc, "myModelPath")
sameModel = KMeansModel.load(sc, "myModelPath")
```

NOTE

The **kmeans_data.txt** is available here https://www.dropbox.com/s/q85t0ytb9apggnh/kmeans_data.txt?dl=0 (https://www.dropbox.com/s/q85t0ytb9apggnh/kmeans_data.txt?dl=0)

TASKS

- Run this code snippet and list the clusters that your find.
- compute the Within Set Sum of Squared Errors for the found clusters. Comment on your findings.

In [179]:
```
!wget https://www.dropbox.com/s/q85t0ytb9apggnh/kmeans_data.txt?dl=0
!mv kmeans_data.txt?dl=0 kmeans_data.txt
```

```
--2017-07-23 12:44:11--  https://www.dropbox.com/s/q85t0ytb9apggnh/kmeans_data.t
xt?dl=0
Resolving www.dropbox.com... 162.125.5.1, 2620:100:601d:1::a27d:501
Connecting to www.dropbox.com|162.125.5.1|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://dl.dropboxusercontent.com/content_link/Fmyfk4WwvrKk2OO0bhYF7en
KcNNJHqeDlf1KZgM7Il3UbUBiVjYVaEWAx6KvSv8E/file [following]
--2017-07-23 12:44:12--  https://dl.dropboxusercontent.com/content_link/Fmyfk4Ww
vrKk2OO0bhYF7enKcNNJHqeDlf1KZgM7Il3UbUBiVjYVaEWAx6KvSv8E/file
Resolving dl.dropboxusercontent.com... 162.125.5.6
Connecting to dl.dropboxusercontent.com|162.125.5.6|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 72 [text/plain]
Saving to: `kmeans_data.txt?dl=0'

100%[======================================>] 72          --.-K/s   in 0s

2017-07-23 12:44:12 (16.9 MB/s) - `kmeans_data.txt?dl=0' saved [72/72]
```

In [180]:
```
!hdfs dfs -rm -r HW10/HW102/
!hdfs dfs -mkdir HW10/HW102/
!hdfs dfs -copyFromLocal kmeans_data.txt HW10/HW102/
```

```
rm: `HW10/HW102/': No such file or directory
```

In [181]:
```
# START STUDENT CODE 10.2
# (ADD CELLS AS NEEDED)
!hdfs dfs -rm -r HW10/HW102/myModelPath
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt


# Load and parse the data
# NOTE  kmeans_data.txt is available here
#          https://www.dropbox.com/s/q85t0ytb9apggnh/kmeans_data.txt?dl=0
data = sc.textFile("HW10/HW102/kmeans_data.txt")
parsedData = data.map(lambda line: array([float(x) for x in line.split(' ')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
        runs=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))

# Save and load model
clusters.save(sc, "HW10/HW102/myModelPath")
sameModel = KMeansModel.load(sc, "HW10/HW102/myModelPath")

# END STUDENT CODE 10.2
```

```
rm: `HW10/HW102/myModelPath': No such file or directory
Within Set Sum of Squared Error = 0.692820323028
```

In [182]:
```
print sameModel.clusterCenters
```

```
[DenseVector([0.1, 0.1, 0.1]), DenseVector([9.1, 9.1, 9.1])]
```

HW10.3: Homegrown KMeans in Spark

Download the following KMeans notebook (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/3nsthvp8g2rrrdh/EM-Kmeans.ipynb).

Generate 3 clusters with 100 (one hundred) data points per cluster (using the code provided). Plot the data. Then run MLlib's Kmean implementation on this data and report your results as follows:

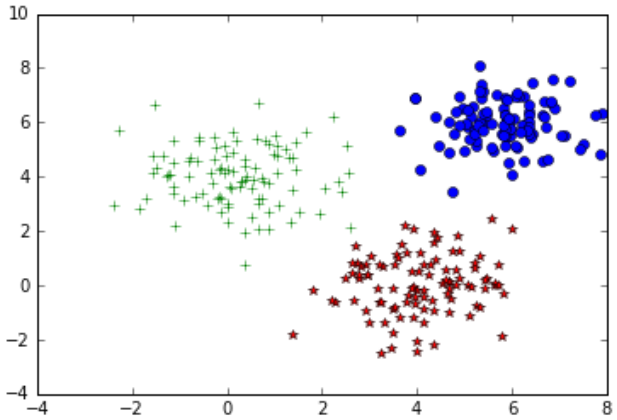
- plot the resulting clusters after 1 iteration, 10 iterations, after 20 iterations, after 100 iterations.
- in each plot please report the Within Set Sum of Squared Errors for the found clusters (as part of the title WSSSE). Comment on the progress of this measure as the KMEans algorithms runs for more iterations. Then plot the WSSSE as a function of the iteration (1, 10, 20, 30, 40, 50, 100).

In [183]:
```
%matplotlib inline
import numpy as np
import pylab
import json
size1 = size2 = size3 = 100
samples1 = np.random.multivariate_normal([4, 0], [[1, 0],[0, 1]], size1)
data = samples1
samples2 = np.random.multivariate_normal([6, 6], [[1, 0],[0, 1]], size2)
data = np.append(data,samples2, axis=0)
samples3 = np.random.multivariate_normal([0, 4], [[1, 0],[0, 1]], size3)
data = np.append(data,samples3, axis=0)
# Randomlize data
data = data[np.random.permutation(size1+size2+size3),]
np.savetxt('data.csv',data,delimiter = ',')
```

In [184]:
```
data.shape
```

Out[184]:
```
(300, 2)
```

In [185]:
```
pylab.plot(samples1[:, 0], samples1[:, 1],'*', color = 'red')
pylab.plot(samples2[:, 0], samples2[:, 1],'o',color = 'blue')
pylab.plot(samples3[:, 0], samples3[:, 1],'+',color = 'green')
pylab.show()
```



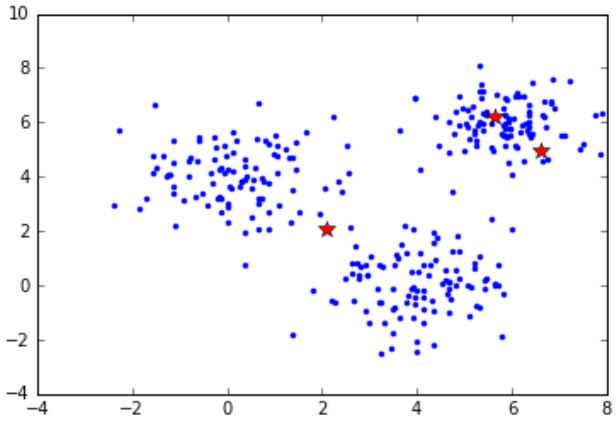
In [186]:
```
#plot centroids and data points for each iteration
def plot_iteration(means):
    pylab.plot(samples1[:, 0], samples1[:, 1], '.', color = 'blue')
    pylab.plot(samples2[:, 0], samples2[:, 1], '.', color = 'blue')
    pylab.plot(samples3[:, 0], samples3[:, 1],'.', color = 'blue')
    pylab.plot(means[0][0], means[0][1],'*',markersize =10,color = 'red')
    pylab.plot(means[1][0], means[1][1],'*',markersize =10,color = 'red')
    pylab.plot(means[2][0], means[2][1],'*',markersize =10,color = 'red')
    pylab.show()
```

In [187]:
```
!hdfs dfs -rm -r HW10/HW103/
!hdfs dfs -mkdir HW10/HW103/
!hdfs dfs -copyFromLocal data.csv HW10/HW103/
```

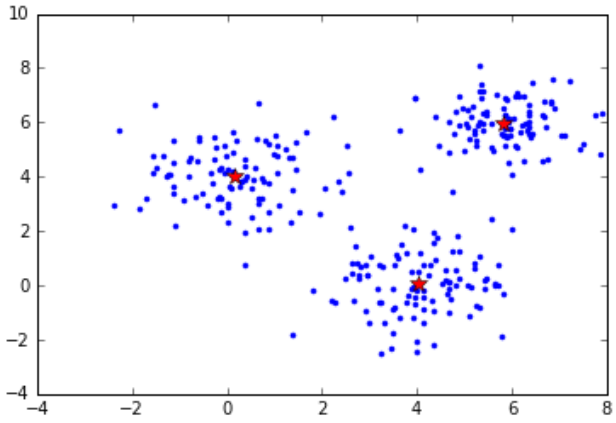
```
rm: `HW10/HW103/': No such file or directory
```

In [188]:
```
# START STUDENT CODE 10.3
# (ADD CELLS AS NEEDED)


from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt

# Load and parse the data
data = sc.textFile("HW10/HW103/data.csv")
parsedData = data.map(lambda line: array([float(x) for x in line.split(',')]))
# Evaluate clustering by computing Within Set Sum of Squared Errors
def error_sqr(point,clusters):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

for k in [1, 10, 20, 30, 40, 50, 100]:
# Build the model (cluster the data)
    clusters = KMeans.train(parsedData, 3, maxIterations=k,
        runs=k, initializationMode="random")
    for centroid in clusters.centers:
        print centroid
    WSSSE = parsedData.map(lambda point: error_sqr(point,clusters)).reduce(lambda
x, y: x + y)
    print("Within Set Sum of Squared Error = " + str(WSSSE))
    plot_iteration(clusters.centers)
# END STUDENT CODE 10.3
```

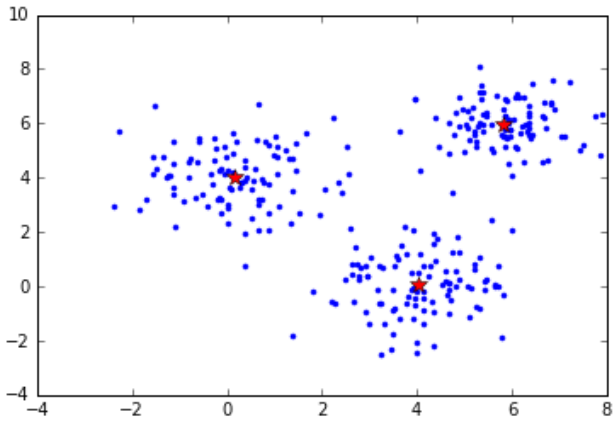
```
[ 2.10918219  2.04454064]
[ 5.64558979  6.2279898 ]
[ 6.60489731  4.92752057]
Within Set Sum of Squared Error = 684.943521309
```



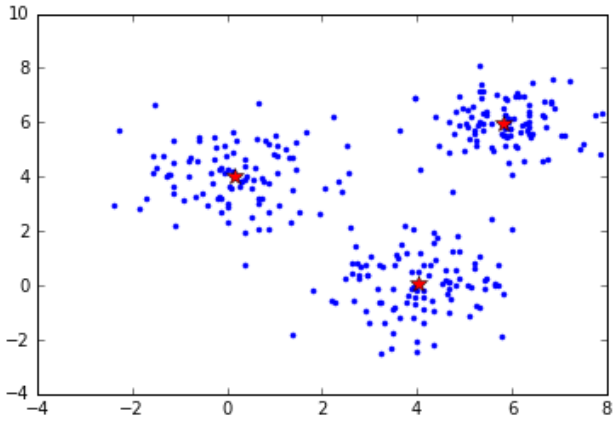
```
[ 5.82902034  5.97416566]
[ 0.17110352  4.03940425]
[ 4.03622674  0.05721978]
Within Set Sum of Squared Error = 368.135955925
```



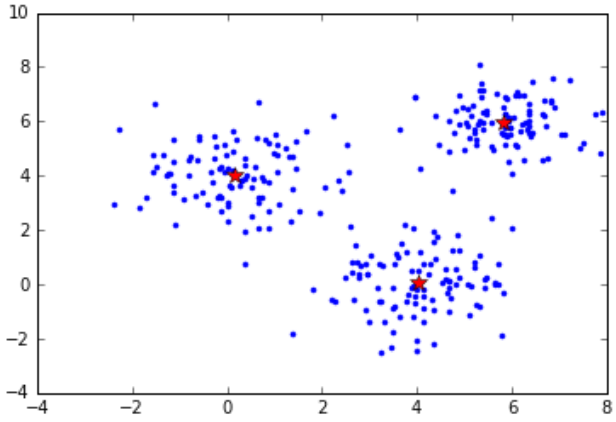
```
[ 5.82902034  5.97416566]
[ 4.03622674  0.05721978]
[ 0.17110352  4.03940425]
Within Set Sum of Squared Error = 368.135955925
```



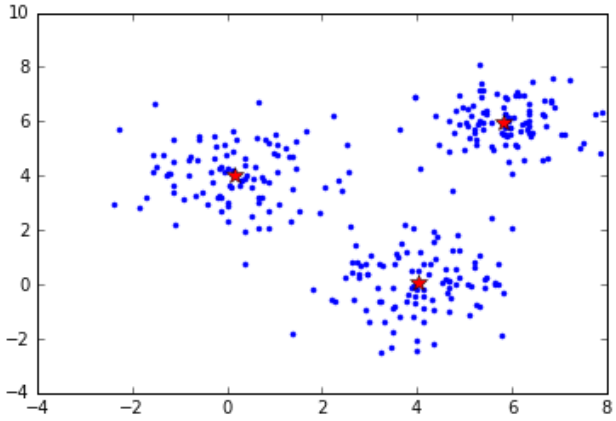
```
[ 0.17110352  4.03940425]
[ 4.03622674  0.05721978]
[ 5.82902034  5.97416566]
Within Set Sum of Squared Error = 368.135955925
```



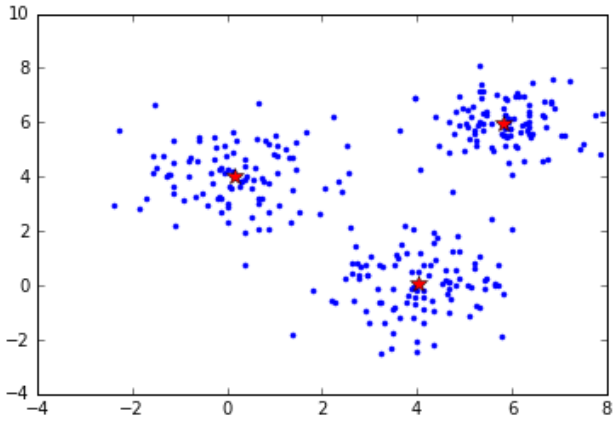
```
[ 0.17110352  4.03940425]
[ 5.82902034  5.97416566]
[ 4.03622674  0.05721978]
Within Set Sum of Squared Error = 368.135955925
```



```
[ 4.03622674  0.05721978]
[ 5.82902034  5.97416566]
[ 0.17110352  4.03940425]
Within Set Sum of Squared Error = 368.135955925
```



```
[ 4.03622674  0.05721978]
[ 0.17110352  4.03940425]
[ 5.82902034  5.97416566]
Within Set Sum of Squared Error = 368.135955925
```

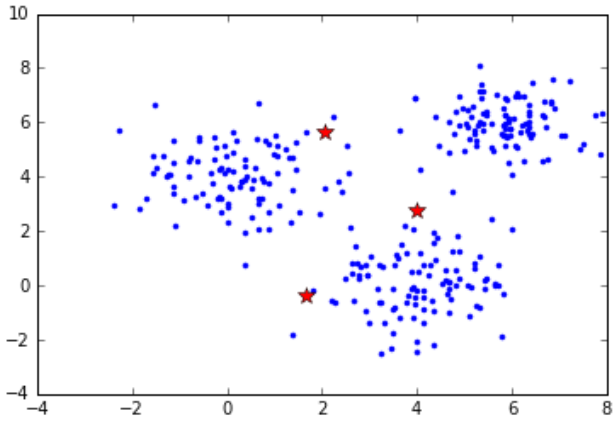


HW10.4: KMeans Experiments

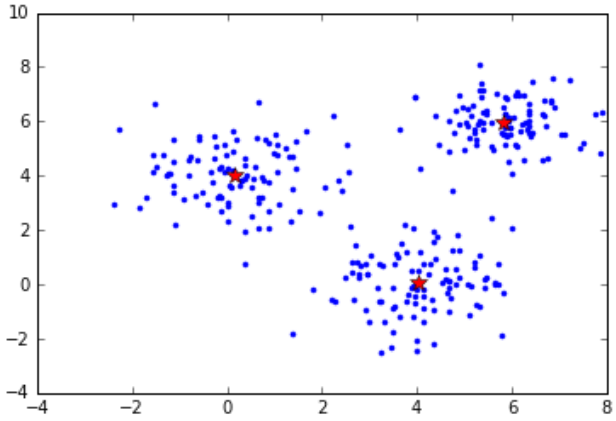
Using this provided homegrown Kmeans code (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/3nsthvp8g2rrrdh/EM-Kmeans.ipynb) repeat the experiments in HW10.3. Explain any differences between the results in HW10.3 and HW10.4.

In [189]:
```
# START STUDENT CODE 10.4
# (ADD CELLS AS NEEDED)



import numpy as np


#Calculate which class each data point belongs to
def nearest_centroid(line):
    x = np.array([float(f) for f in line.split(',')])
    closest_centroid_idx = np.sum((x - centroids)**2, axis=1).argmin()
    return (closest_centroid_idx,(x,1))

K = 3


D = sc.textFile("HW10/HW103/data.csv").cache()

def calcError(line):

    center = centroids[nearest_centroid(line)[0]]

    point = np.array([float(f) for f in line.split(',')])

    return sqrt(sum([x**2 for x in (point - center)]))
iter_num = 0
for k in [1, 10, 20, 30, 40, 50, 100]:
    # Initialization: initialization of parameter is fixed to show an example
    centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])
    for i in range(k):
        res = D.map(nearest_centroid).reduceByKey(lambda x,y : (x[0]+y[0],x[1]+y[
1])).collect()

        res = sorted(res,key = lambda x : x[0])  #sort based on clusted ID
        centroids_new = np.array([x[1][0]/x[1][1] for x in res])  #divide by clus
ter size
        centroids = centroids_new

    print "Iteration" + str(k)
    WSSSE = D.map(calcError).reduce(lambda x, y: x + y)

    print("Within Set Sum of Squared Error = " + str(WSSSE))
    print centroids
    plot_iteration(centroids)



# END STUDENT CODE 10.4
```

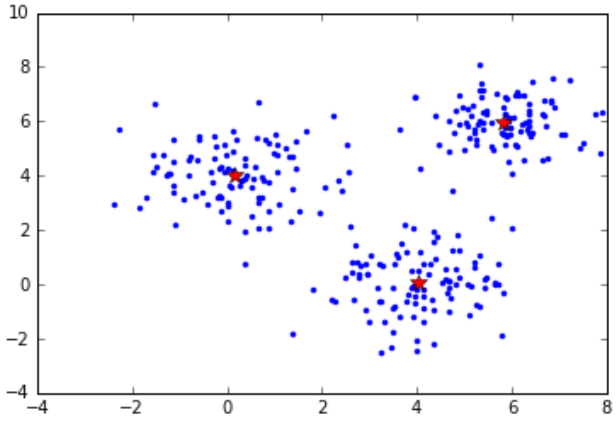
```
Iteration1
Within Set Sum of Squared Error = 828.166293166
[[ 1.67562611 -0.35973024]
 [ 3.99724001  2.76227162]
 [ 2.07060224  5.64188339]]
```



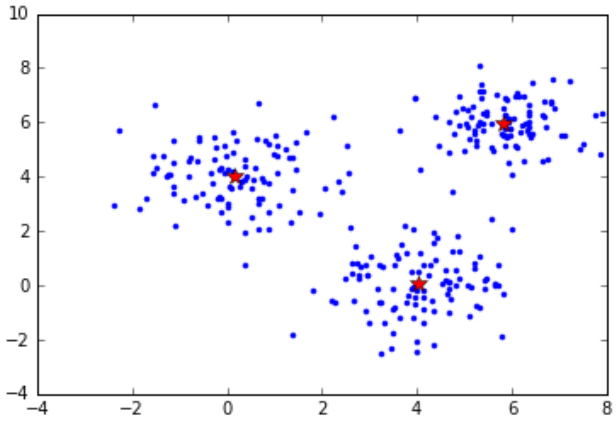
```
Iteration10
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



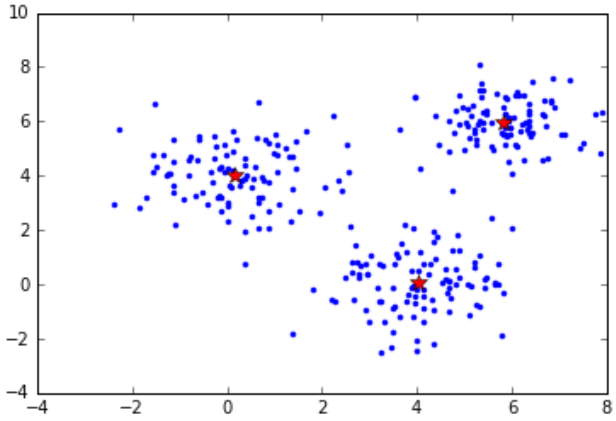
```
Iteration20
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



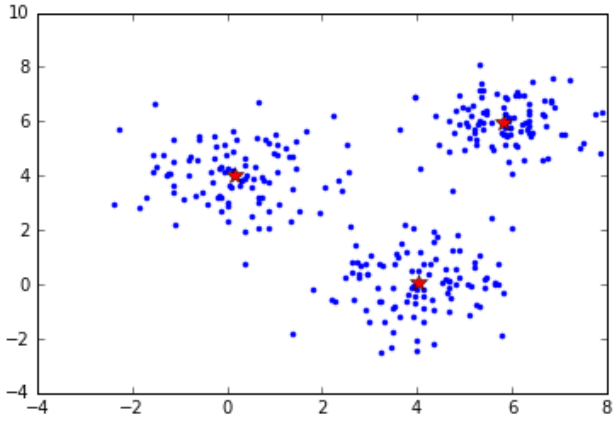
```
Iteration30
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



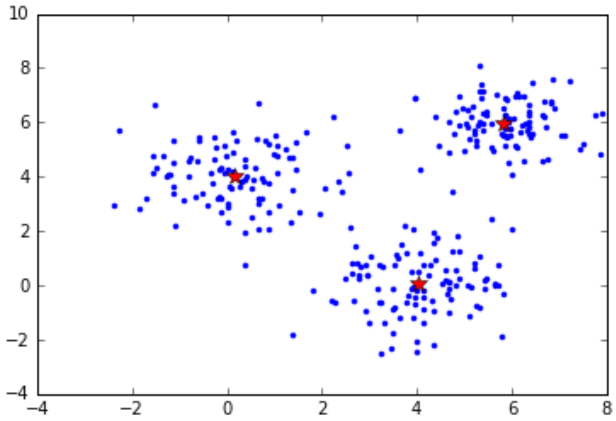
```
Iteration40
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



```
Iteration50
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



```
Iteration100
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



HW10.4.1: Making Homegrown KMeans more efficient

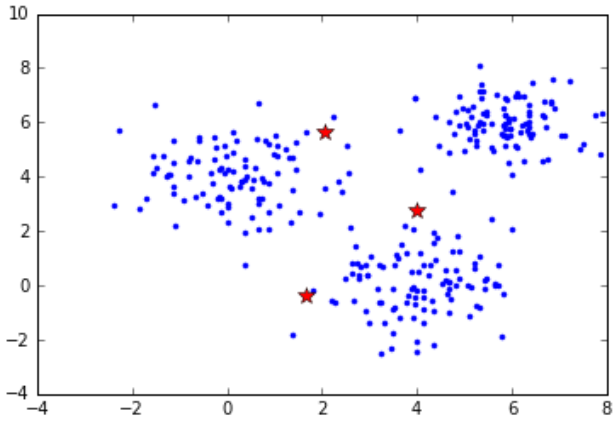
The above provided homegrown KMeans implentation in not the most efficient. How can you make it more efficient? Make this change in the code and show it work and comment on the gains you achieve.

Issue with 10.4.0 version is around the variable centroid which gets copied over each nod, which takes time. We will use broadcasting where variable will be copid on only one node and shared.

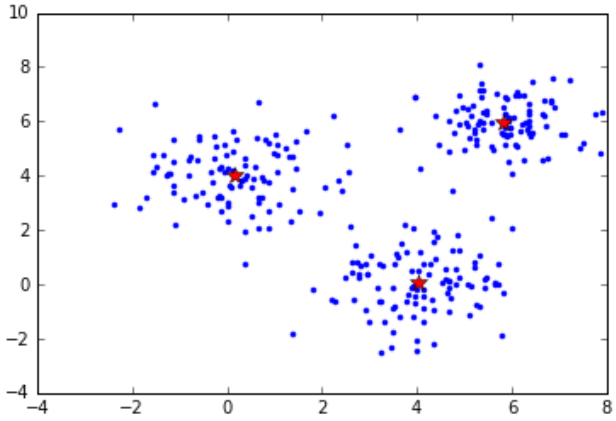
HINT: have a look at this linear regression notebook (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/atzqkc0p1eajuz6/LinearRegression-Notebook-Challenge.ipynb)

In [190]:
```
# START STUDENT CODE 10.4.1
# (ADD CELLS AS NEEDED)

import numpy as np


#Calculate which class each data point belongs to
def nearest_centroid(line):
    x = np.array([float(f) for f in line.split(',')])
    closest_centroid_idx = np.sum((x - wBroadcast.value)**2, axis=1).argmin()
    return (closest_centroid_idx,(x,1))

K = 3


D = sc.textFile("HW10/HW103/data.csv").cache()

def calcError(line):

    center = centroids[nearest_centroid(line)[0]]

    point = np.array([float(f) for f in line.split(',')])

    return sqrt(sum([x**2 for x in (point - center)]))
iter_num = 0
for k in [1, 10, 20, 30, 40, 50, 100]:
    # Initialization: initialization of parameter is fixed to show an example
    centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])
    wBroadcast = sc.broadcast(centroids)
    for i in range(k):
        res = D.map(nearest_centroid).reduceByKey(lambda x,y : (x[0]+y[0],x[1]+y[
1])).collect()

        res = sorted(res,key = lambda x : x[0])  #sort based on clusted ID
        centroids_new = np.array([x[1][0]/x[1][1] for x in res])  #divide by clus
ter size
        centroids = centroids_new
        wBroadcast = sc.broadcast(centroids)

    print "Iteration" + str(k)
    WSSSE = D.map(calcError).reduce(lambda x, y: x + y)

    print("Within Set Sum of Squared Error = " + str(WSSSE))
    print centroids
    plot_iteration(centroids)

# END STUDENT CODE 10.4.1
```

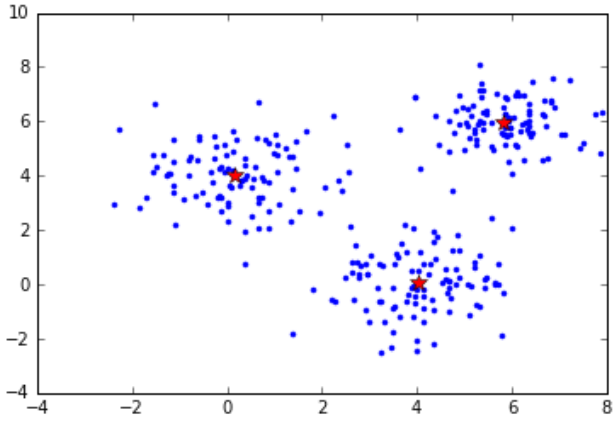
```
Iteration1
Within Set Sum of Squared Error = 828.166293166
[[ 1.67562611 -0.35973024]
 [ 3.99724001  2.76227162]
 [ 2.07060224  5.64188339]]
```



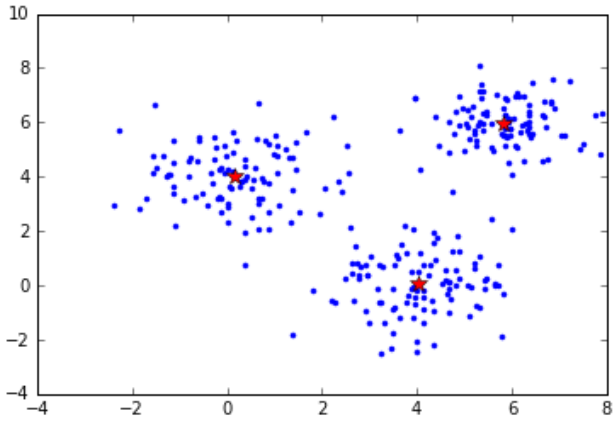
```
Iteration10
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



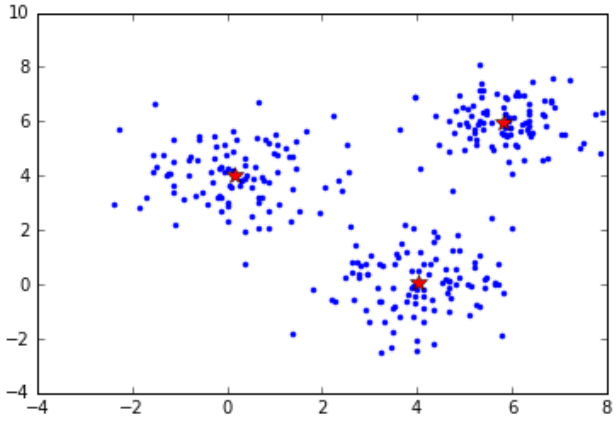
```
Iteration20
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



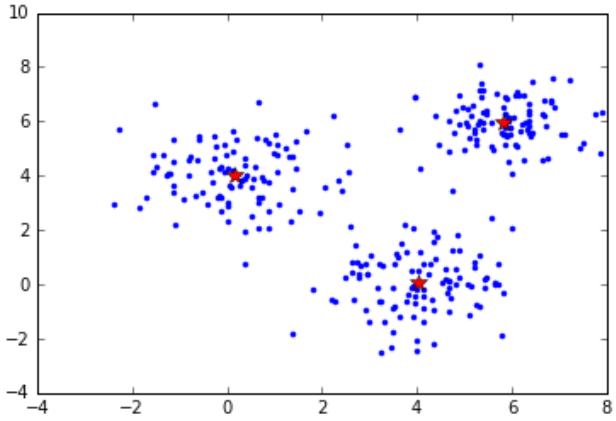
```
Iteration30
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



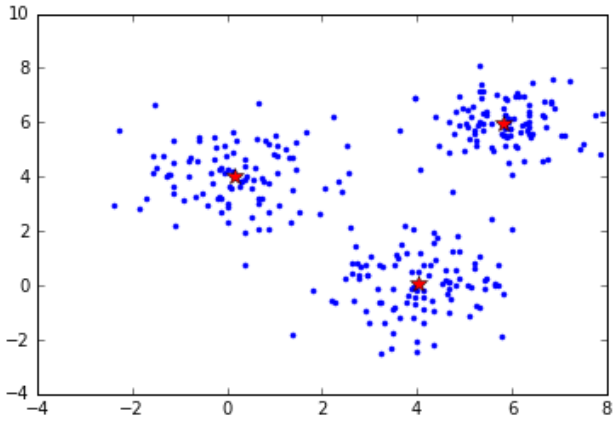
```
Iteration40
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



```
Iteration50
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



```
Iteration100
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



HW10.5: OPTIONAL Weighted KMeans

Using this provided homegrown Kmeans code (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/3nsthvp8g2rrrdh/EM-Kmeans.ipynb), modify it to do a weighted KMeans and repeat the experiements in HW10.3. Explain any differences between the results in HW10.3 and HW10.5.

NOTE: Weight each example as follows using the inverse vector length (Euclidean norm):

> weight(X)= 1/||X||,

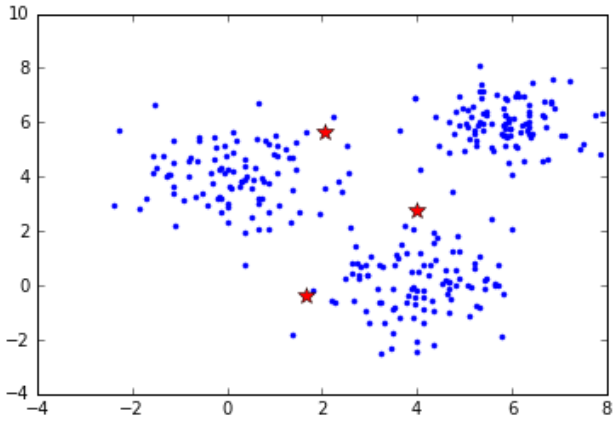
where ||X|| = SQRT(X.X)= SQRT(X1^2 + X2^2)

Here X is vector made up of two values X1 and X2.

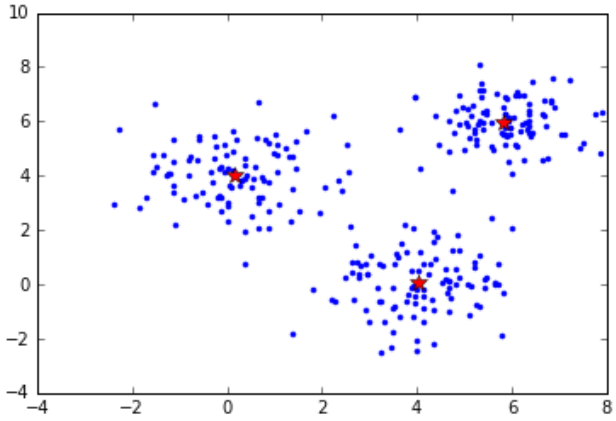
[Please incorporate all referenced notebooks directly into this master notebook as cells for HW submission. I.e., HW submissions should comprise of just one notebook]

In [197]:
```
# START STUDENT CODE 10.5
# (ADD CELLS AS NEEDED)
def nearest_centroid(line):
    x = np.array([float(f) for f in line.split(',')])
    wx = 1/abs(sqrt(x[0]**2 + x[1]**2))
    closest_centroid_idx = np.sum((x - wBroadcast.value)**2 * wx, axis=1).argmin(
)

    return (closest_centroid_idx,(x,1))

K = 3


D = sc.textFile("HW10/HW103/data.csv").cache()

def calcError(line):

    center = centroids[nearest_centroid(line)[0]]

    point = np.array([float(f) for f in line.split(',')])

    return sqrt(sum([x**2 for x in (point - center)]))
iter_num = 0
for k in [1, 10, 20, 30, 40, 50, 100]:
    # Initialization: initialization of parameter is fixed to show an example
    centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])
    wBroadcast = sc.broadcast(centroids)
    for i in range(k):
        res = D.map(nearest_centroid).reduceByKey(lambda x,y : (x[0]+y[0],x[1]+y[
1])).collect()

        res = sorted(res,key = lambda x : x[0])  #sort based on clusted ID
        centroids_new = np.array([x[1][0]/x[1][1] for x in res])  #divide by clus
ter size
        centroids = centroids_new
        wBroadcast = sc.broadcast(centroids)

    print "Iteration" + str(k)
    WSSSE = D.map(calcError).reduce(lambda x, y: x + y)

    print("Within Set Sum of Squared Error = " + str(WSSSE))
    print centroids
    plot_iteration(centroids)
# END STUDENT CODE 10.5
```

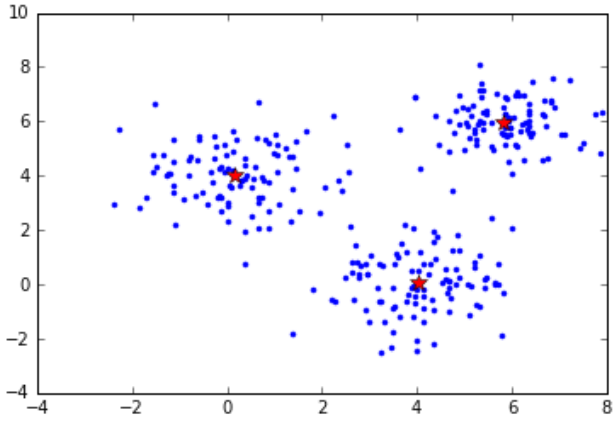
```
Iteration1
Within Set Sum of Squared Error = 828.166293166
[[ 1.67562611 -0.35973024]
 [ 3.99724001  2.76227162]
 [ 2.07060224  5.64188339]]
```



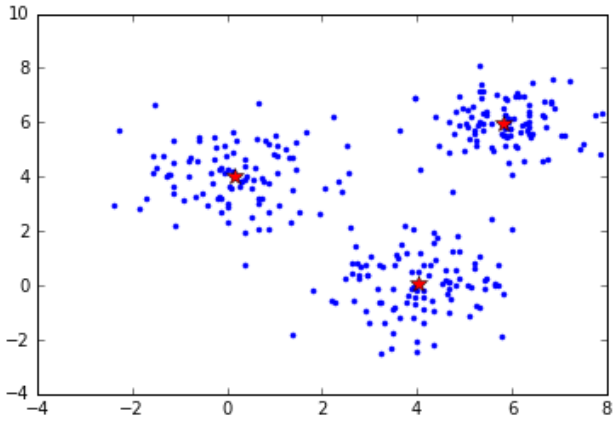
```
Iteration10
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



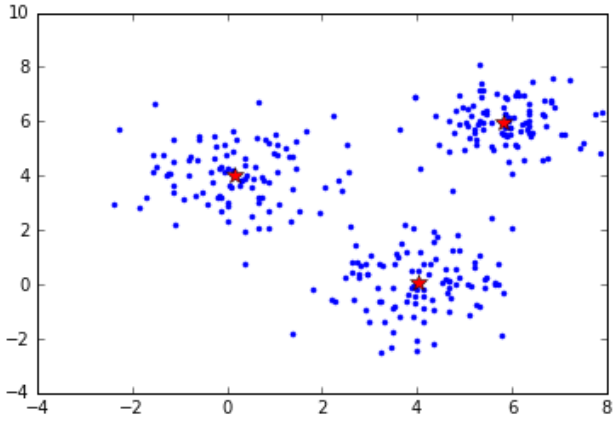
```
Iteration20
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



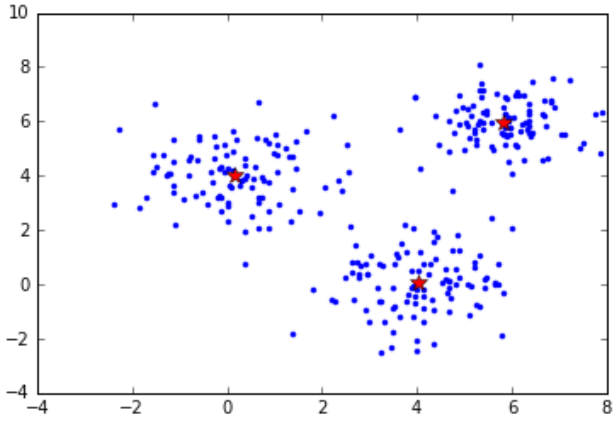
```
Iteration30
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



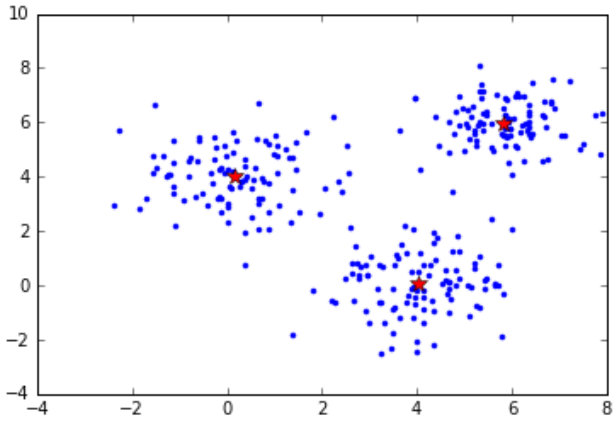
```
Iteration40
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



```
Iteration50
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



```
Iteration100
Within Set Sum of Squared Error = 368.135955925
[[ 4.03622674  0.05721978]
 [ 5.82902034  5.97416566]
 [ 0.17110352  4.03940425]]
```



HW10.6 OPTIONAL Linear Regression

Using this linear regression notebook (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/atzqkc0p1eajuz6/LinearRegression-Notebook-Challenge.ipynb):

- Generate 2 sets of data with 100 data points using the data generation code provided and plot each in separate plots. Call one the training set and the other the testing set.
- Using MLLib's LinearRegressionWithSGD train up a linear regression model with the training dataset and evaluate with the testing set. What a good number of iterations for training the linear regression model? Justify with plots (e.g., plot MSE as a function of the number of iterations) and words.

In [192]:
```
# START STUDENT CODE 10.6
# (ADD CELLS AS NEEDED)

# END STUDENT CODE 10.6
```

HW10.6.1 OPTIONAL Linear Regression

In the notebook provided above, in the cell labeled "Gradient descent (regularization)".

- Fill in the blanks and get this code to work for LASS0 and RIDGE linear regression.
- Using the data from HW10.6.1 tune the hyper parameters of your LASS0 and RIDGE regression. Report your findings with words and plots.

In [193]:
```
# START STUDENT CODE 10.6.1
# (ADD CELLS AS NEEDED)

# END STUDENT CODE 10.6.1
```

HW10.7 OPTIONAL Error surfaces

Here is a link to R code with 1 test drivers that plots the linear regression model in model space and in the domain space:

> https://www.dropbox.com/s/3xc3kwda6d254l5/PlotModelAndDomainSpaces.R?dl=0 (https://www.dropbox.com/s/3xc3kwda6d254l5/PlotModelAndDomainSpaces.R?dl=0)

Here is a sample output from this script:

> https://www.dropbox.com/s/my3tnhxx7fr5qs0/image%20%281%29.png?dl=0 (https://www.dropbox.com/s/my3tnhxx7fr5qs0/image%20%281%29.png?dl=0)

Please use this as inspiration and code a equivalent error surface and heatmap (with isolines) in Spark and show the trajectory of learning taken during gradient descent (after each n-iterations of Gradient Descent):

Using Spark and Python (using the above R Script as inspiration), plot the error surface for the linear regression model using a heatmap and contour plot. Also plot the current model in the original domain space for every 10th iteration. Plot them side by side if possible for each iteration: lefthand side plot is the model space(w0 and w01) and the righthand side plot is domain space (plot the corresponding model and training data in the problem domain space) with a final pair of graphs showing the entire trajectory in the model and domain space. Make sure to label your plots with iteration numbers, function, model space versus original domain space, MSE on the training data etc.

Also plot the MSE as a function of each iteration (possibly every 10th iteration). Dont forget to label both axis and the graph also. **[Please incorporate all referenced notebooks directly into this master notebook as cells for HW submission. I.e., HW submissions should comprise of just one notebook]**

In [194]:
```
# START STUDENT CODE 10.7
# (ADD CELLS AS NEEDED)

# END STUDENT CODE 10.7
```

In [ ]:
```

```