In [1]:
```
%%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-with-a-trailing-vertical-line
*****************************************************************************************************/

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

In [2]:
```
%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: Nina Kuklisova
Class: MIDS w261 (2016 Section 2)
Email: nkuklisova@iSchool.Berkeley.edu
Week: 10

Table of Contents

1 Instructions

Back to Table of Contents

- Homework submissions are due by Tueday, 07/28/2016 at 11AM (West Coast Time).

- Prepare a single Jupyter note, please include questions, and question numbers in the questions and in the responses. Submit your homework notebook via the following form:
  - Submission Link - Google Form (https://docs.google.com/forms/d/1ZOr9RnIe_A06AcZDB6K1mJN4vrLeSmS2PD6Xm3eOiis/viewform usp=send_form)

Documents:

- IPython Notebook, published and viewable online.
- PDF export of IPython Notebook.

2 Useful References

Back to Table of Contents

- 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))

3 HW Problems

Back to Table of Contents

HW10.0: Short answer questions

Back to Table of Contents

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

Spark is an optimized engine that supports general execution graphs over an RDD. Spark allows increased parallelism and higher-level data management. As a result, it is much faster tahn Hadoop, and has specialized libarries for machine learning, graph processing, and database management.

Fill in the blanks: Spark API consists of interfaces to develop applications based on it in Java, Scala, Python and R languages.

Using Spark, resource management can be done either in a single server instance or using a framework such as Mesos or YARN in a distributed manner.

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

RDD (Resilient Distributed Datasets) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.

In [3]:

```
#Fun Example:

rdd = sc.parallelize('file_to_be_processed.txt')  #distributes the string
rdd.first()
```

Out[3]: 'f'

HW10.1 WordCount plus sorting

Back to Table of Contents

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. Plseas 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 [2]:
```
## Set up Spark

import os
import sys
import pyspark
from pyspark.sql import SQLContext

# We can give a name to our app (to find it in Spark WebUI) and config
ure execution mode
# In this case, it is local multicore execution with "local[*]"
app_name = "example-logs"
master = "local[*]"
conf = pyspark.SparkConf().setAppName(app_name).setMaster(master)
sc = pyspark.SparkContext(conf=conf)
sqlContext = SQLContext(sc)
print sc
print sqlContext
```

```
<pyspark.context.SparkContext object at 0x7f5ba0397e50>
<pyspark.sql.context.SQLContext object at 0x7f5b77217390>
```

In [86]:
```
## Drivers & Runners

# complete word count
#Count words in file/directory
logFileNAME = 'enronemail_1h.txt'
text_file = sc.textFile(logFileNAME)

counts = text_file.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b, 1) \
    .sortBy(lambda x: -x[1])

wordCounts = counts.collect()

print ("20 most popular terms:")
print wordCounts[:20]

print ("10 least popular terms:")
print wordCounts[-10:]
```

```
20 most popular terms:
[(u'', 3504), (u'the', 1240), (u'to', 908), (u'and', 646), (u'of', 5
55), (u'a', 514), (u'in', 412), (u'your', 389), (u'you', 376), (u'fo
r', 368), (u'@', 361), (u'on', 253), (u'this', 243), (u'is', 243), (
u'""', 239), (u'will', 234), (u'i', 232), (u'|', 228), (u'be', 216),
(u'that', 213)]
10 least popular terms:
[(u'kinds', 1), (u'pinion/hou/ect', 1), (u'webpage,', 1), (u'inheren
tly', 1), (u'my,', 1), (u'my.', 1), (u'coffee,', 1), (u'"">>', 1), (
u'commencement', 1), (u'volumes', 1)]
```

HW10.1.1

Back to Table of Contents

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 [4]:
```
## Code goes here

counts = text_file.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, int(word.islower()))) \
    .reduceByKey(lambda a, b: a + b, 1) \
    .sortBy(lambda x: -x[1])

## Drivers & Runners
wordCounts = counts.collect()

print ("20 most popular terms:")
print wordCounts[:20]

print ("10 least popular terms:")
print wordCounts[-10:]
```

```
20 most popular terms:
[(u'the', 1240), (u'to', 908), (u'and', 646), (u'of', 555), (u'a', 5
14), (u'in', 412), (u'your', 389), (u'you', 376), (u'for', 368), (u'
on', 253), (u'this', 243), (u'is', 243), (u'will', 234), (u'i', 232)
, (u'be', 216), (u'that', 213), (u'with', 197), (u'have', 168), (u'a
re', 168), (u'we', 161)]
10 least popular terms:
[(u'249.', 0), (u'11)', 0), (u'12/15', 0), (u'8,703', 0), (u'*', 0),
(u'877.', 0), (u'07:47', 0), (u'-->', 0), (u'9,908', 0), (u'"">>', 0
)]
```

HW10.2: MLlib-centric KMeans

Back to Table of Contents

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 [5]:
```
%%writefile kmeans_data.txt
0.0 0.0 0.0
0.1 0.1 0.1
0.2 0.2 0.2
9.0 9.0 9.0
9.1 9.1 9.1
9.2 9.2 9.2
```

```
Overwriting kmeans_data.txt
```

In [6]:
```
## Code goes here
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt


# Load and parse the data
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, init
ializationMode="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")
```

```
/usr/local/spark/python/pyspark/mllib/clustering.py:176: UserWarning
: Support for runs is deprecated in 1.6.0. This param will have no e
ffect in 1.7.0.
  "Support for runs is deprecated in 1.6.0. This param will have no
effect in 1.7.0.")

Within Set Sum of Squared Error = 0.692820323028


Py4JJavaErrorTraceback (most recent call last)
<ipython-input-6-c0273e617df9> in <module>()
     21
```

```
     22 # Save and load model
---> 23 clusters.save(sc, "myModelPath")
     24 sameModel = KMeansModel.load(sc, "myModelPath")

/usr/local/spark/python/pyspark/mllib/clustering.py in save(self, sc
, path)
    150         java_centers = _py2java(sc, [_convert_to_vector(c) f
or c in self.centers])
    151         java_model = sc._jvm.org.apache.spark.mllib.clusteri
ng.KMeansModel(java_centers)
--> 152         java_model.save(sc._jsc.sc(), path)
    153 
    154     @classmethod

/usr/local/spark/python/lib/py4j-0.9-src.zip/py4j/java_gateway.py in
__call__(self, *args)
    811         answer = self.gateway_client.send_command(command)
    812         return_value = get_return_value(
--> 813             answer, self.gateway_client, self.target_id, sel
f.name)
    814 
    815         for temp_arg in temp_args:

/usr/local/spark/python/pyspark/sql/utils.py in deco(*a, **kw)
     43     def deco(*a, **kw):
     44         try:
---> 45             return f(*a, **kw)
     46         except py4j.protocol.Py4JJavaError as e:
     47             s = e.java_exception.toString()

/usr/local/spark/python/lib/py4j-0.9-src.zip/py4j/protocol.py in get
_return_value(answer, gateway_client, target_id, name)
    306                 raise Py4JJavaError(
    307                     "An error occurred while calling {0}{1}{
2}.\n".
--> 308                     format(target_id, ".", name), value)
    309             else:
    310                 raise Py4JError(

Py4JJavaError: An error occurred while calling o87.save.
: org.apache.hadoop.mapred.FileAlreadyExistsException: Output direct
ory file:/home/jovyan/work/root/Documents/MIDS/W 261 Machine Learnin
g at Scale/myModelPath/metadata already exists
        at org.apache.hadoop.mapred.FileOutputFormat.checkOutputSpec
s(FileOutputFormat.java:132)
        at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHado
opDataset$1.apply$mcV$sp(PairRDDFunctions.scala:1179)
        at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHado
opDataset$1.apply(PairRDDFunctions.scala:1156)
        at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHado
```

```
opDataset$1.apply(PairRDDFunctions.scala:1156)
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOper
ationScope.scala:150)
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOper
ationScope.scala:111)
        at org.apache.spark.rdd.RDD.withScope(RDD.scala:316)
        at org.apache.spark.rdd.PairRDDFunctions.saveAsHadoopDataset
(PairRDDFunctions.scala:1156)
        at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHado
opFile$4.apply$mcV$sp(PairRDDFunctions.scala:1060)
        at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHado
opFile$4.apply(PairRDDFunctions.scala:1026)
        at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHado
opFile$4.apply(PairRDDFunctions.scala:1026)
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOper
ationScope.scala:150)
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOper
ationScope.scala:111)
        at org.apache.spark.rdd.RDD.withScope(RDD.scala:316)
        at org.apache.spark.rdd.PairRDDFunctions.saveAsHadoopFile(Pa
irRDDFunctions.scala:1026)
        at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHado
opFile$1.apply$mcV$sp(PairRDDFunctions.scala:952)
        at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHado
opFile$1.apply(PairRDDFunctions.scala:952)
        at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHado
opFile$1.apply(PairRDDFunctions.scala:952)
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOper
ationScope.scala:150)
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOper
ationScope.scala:111)
        at org.apache.spark.rdd.RDD.withScope(RDD.scala:316)
        at org.apache.spark.rdd.PairRDDFunctions.saveAsHadoopFile(Pa
irRDDFunctions.scala:951)
        at org.apache.spark.rdd.RDD$$anonfun$saveAsTextFile$1.apply$
mcV$sp(RDD.scala:1457)
        at org.apache.spark.rdd.RDD$$anonfun$saveAsTextFile$1.apply(
RDD.scala:1436)
        at org.apache.spark.rdd.RDD$$anonfun$saveAsTextFile$1.apply(
RDD.scala:1436)
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOper
ationScope.scala:150)
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOper
ationScope.scala:111)
        at org.apache.spark.rdd.RDD.withScope(RDD.scala:316)
        at org.apache.spark.rdd.RDD.saveAsTextFile(RDD.scala:1436)
        at org.apache.spark.mllib.clustering.KMeansModel$SaveLoadV1_
0$.save(KMeansModel.scala:131)
        at org.apache.spark.mllib.clustering.KMeansModel.save(KMeans
Model.scala:96)
```

```
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Metho
d)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodA
ccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Delegatin
gMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:606)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:2
31)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.
java:381)
        at py4j.Gateway.invoke(Gateway.java:259)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractComman
d.java:133)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.GatewayConnection.run(GatewayConnection.java:209)
        at java.lang.Thread.run(Thread.java:745)
```

The clusters found are:

In [6]:

```
for centroid in clusters.centers:
    print centroid
```

```
[ 0.1  0.1  0.1]
[ 9.1  9.1  9.1]
```

And the Within Set Sum of Squared Errors for the found clusters is:

In [7]:

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

```
Within Set Sum of Squared Error = 0.692820323028
```

This means that by running the code above, we identified clusters centered at (0.1, 0.1, 0.1) and (9.1, 9.1, 9.1), with a Within Set Sum of Squared Errors = 0.6928.

HW10.3: Homegrown KMeans in Spark

Back to Table of Contents

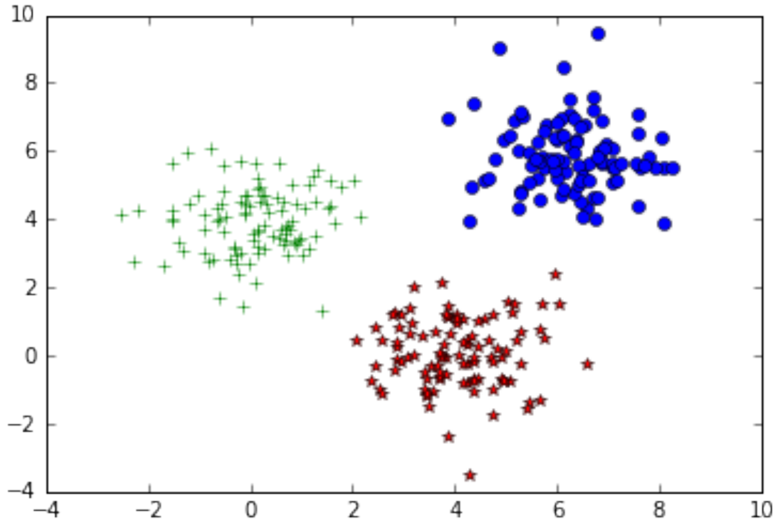
Download the following KMeans notebook (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/3nsthvp8g2rrrdh/EM-Kmeans.ipynb).

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 [71]:
```
%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]], size
1)
data = samples1
samples2 = np.random.multivariate_normal([6, 6], [[1, 0],[0, 1]], size
2)
data = np.append(data,samples2, axis=0)
samples3 = np.random.multivariate_normal([0, 4], [[1, 0],[0, 1]], size
3)
data = np.append(data,samples3, axis=0)
# Randomlize data
data = data[np.random.permutation(size1+size2+size3),]
np.savetxt('data.csv',data,delimiter = ',')
```

In [72]:
```
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()
```



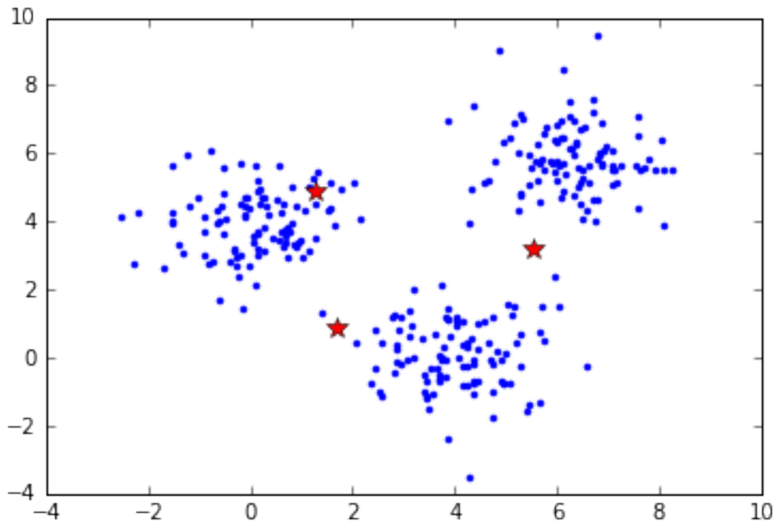
In [73]:
```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt

# Load and parse the data
data = sc.textFile("data.csv")
parsedData = data.map(lambda line: array([float(x) for x in line.split
(',')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 3, maxIterations=20,
        runs=10, initializationMode="random")
for centroid in clusters.centers:
    print centroid
```

```
[ 0.05657819  3.97671718]
[ 6.26427572  5.86742717]
[ 4.02084087  0.08266008]
```

In [74]:
```
import os
import sys
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))

#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 = 're
d')
    pylab.plot(means[1][0], means[1][1],'*',markersize =10,color = 're
d')
    pylab.plot(means[2][0], means[2][1],'*',markersize =10,color = 're
d')
    pylab.show()

def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))
```

In [75]:
```
#K = 3
# Initialization: initialization of parameter is fixed to show an exam
ple
centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])


D = sc.textFile("./data.csv")
parsedData = D.map(lambda line: array([float(x) for x in line.split(',
')]))

wse = []
iter_num = -1
for i in range(101):
    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 cluster size
    if np.sum(np.absolute(centroids_new-centroids))<0.000001:
        break
    iter_num +=1
    centroids = centroids_new
    if i in [1, 10, 20, 100]:
        print "Iteration" + str(iter_num)
        print centroids
        plot_iteration(centroids)
        WSSSE = parsedData.map(lambda point: error(point)).reduce(lamb
da x, y: x + y)
        print("Within Set Sum of Squared Error = " + str(WSSSE))

print "Final Results:"
print centroids
```

```
Iteration1
[[ 1.70557734  0.8494642 ]
 [ 5.54041084  3.19625608]
 [ 1.2730003   4.91916871]]
```




```
Within Set Sum of Squared Error = 358.527731123
Final Results:
[[ 4.02084087  0.08266008]
 [ 6.26427572  5.86742717]
 [ 0.05657819  3.97671718]]
```

This process converges in less than 10 iterations, so we can't compare its WSE after multiple 20, 50 or 100 iterations.

HW10.4: KMeans Experiments

Back to Table of Contents

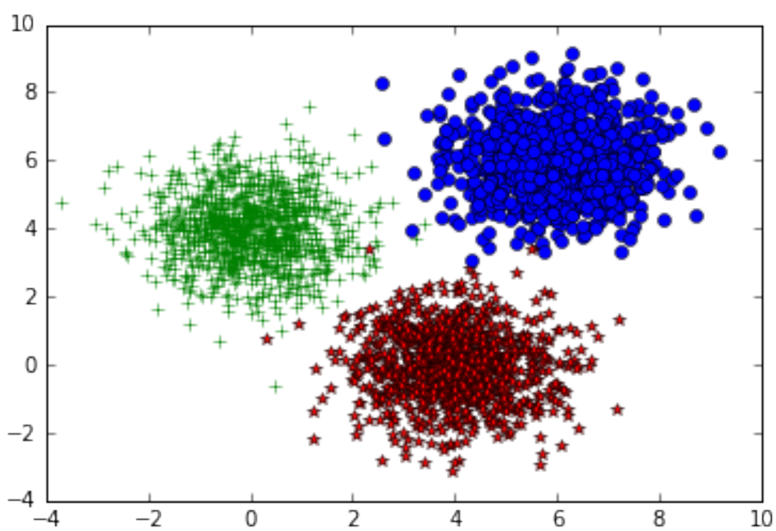
Using this provided homegrown Kmeans code (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/3nsthvp8g2rrrdh/EM-Kmeans.ipynb) repeat the experiments in HW10.3. Explain any differences between the results in HW10.3 and HW10.4.

Data Generation

In [65]:
```
 %matplotlib inline
import numpy as np
import pylab
import json
size1 = size2 = size3 = 1000
samples1 = np.random.multivariate_normal([4, 0], [[1, 0],[0, 1]], size
1)
data = samples1
samples2 = np.random.multivariate_normal([6, 6], [[1, 0],[0, 1]], size
2)
data = np.append(data,samples2, axis=0)
samples3 = np.random.multivariate_normal([0, 4], [[1, 0],[0, 1]], size
3)
data = np.append(data,samples3, axis=0)
# Randomlize data
data = data[np.random.permutation(size1+size2+size3),]
np.savetxt('data.csv',data,delimiter = ',')
```

Data Visualiazation

In [66]:
```
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()
```

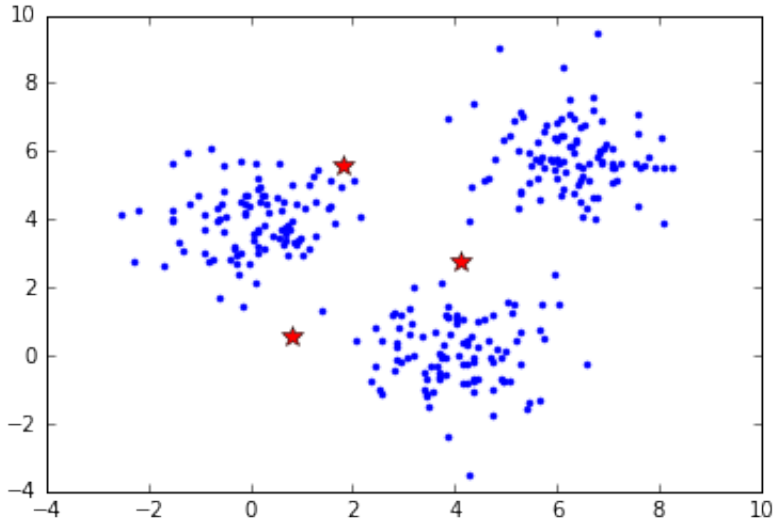


In [67]:
```
import numpy as np

#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))

#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 = 're
d')
    pylab.plot(means[1][0], means[1][1],'*',markersize =10,color = 're
d')
    pylab.plot(means[2][0], means[2][1],'*',markersize =10,color = 're
d')
    pylab.show()
```

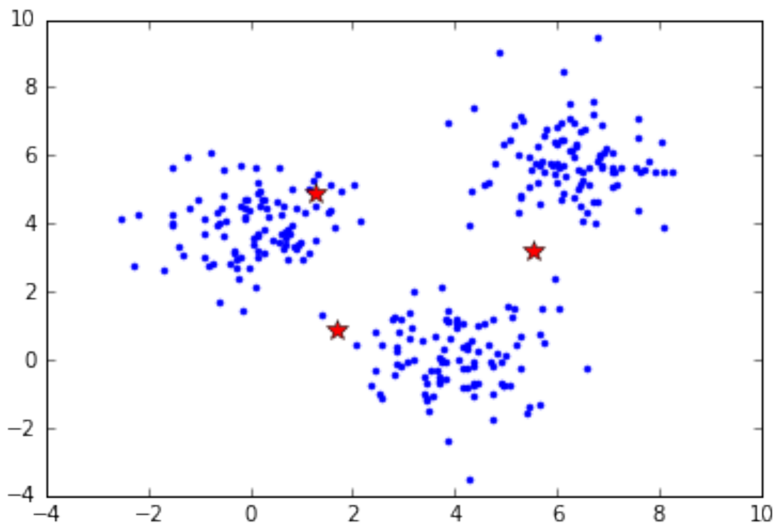
Distributed KMeans in Spark

In [77]:
```
K = 3
# Initialization: initialization of parameter is fixed to show an exam
ple
centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])

D = sc.textFile("./data.csv") #.cache()
parsedData = D.map(lambda line: array([float(x) for x in line.split(',
')]))


iter_num = 0
for i in range(10):
    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 cluster size
    if np.sum(np.absolute(centroids_new-centroids))<0.01:
        break
    print "Iteration" + str(iter_num)
    iter_num = iter_num + 1
    centroids = centroids_new
    print centroids
    plot_iteration(centroids)
    WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x
, y: x + y)
    print("Within Set Sum of Squared Error = " + str(WSSSE))
print "Final Results:"
print centroids
```

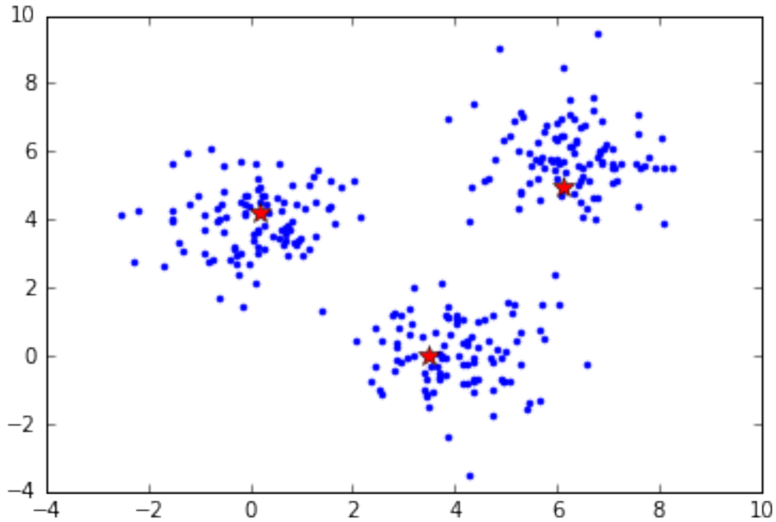
```
Iteration0
[[ 0.82299188  0.56796758]
 [ 4.12230006  2.76406673]
 [ 1.79893782  5.57176309]]
```



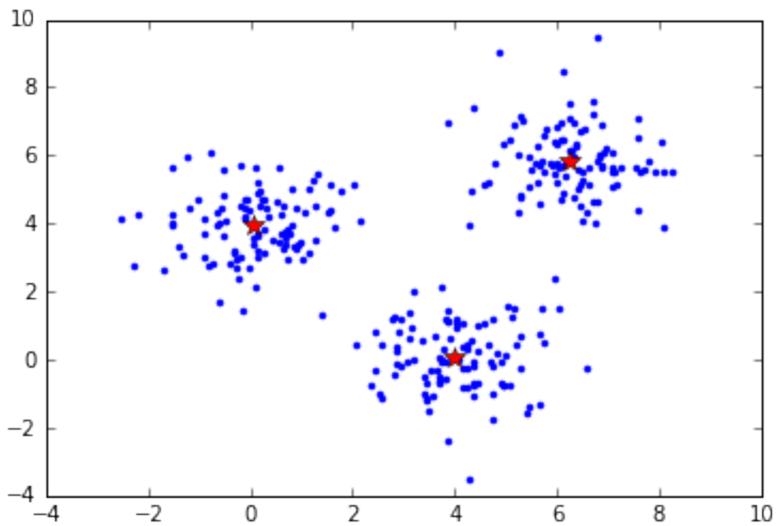

```
Within Set Sum of Squared Error = 358.527731123
Iteration1
[[ 1.70557734  0.8494642 ]
 [ 5.54041084  3.19625608]
 [ 1.2730003   4.91916871]]
```



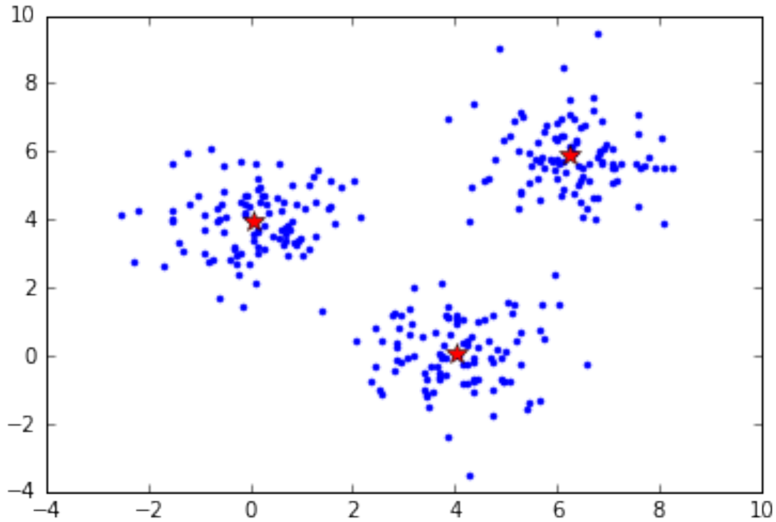

```
Within Set Sum of Squared Error = 358.527731123
Iteration2
[[ 3.50693603  0.0118081 ]
 [ 6.12758883  4.96650481]
 [ 0.20215824  4.19267863]]
```




```
Within Set Sum of Squared Error = 358.527731123
Iteration3
[[ 4.00163563  0.05975173]
 [ 6.26107859  5.83283378]
 [ 0.05657819  3.97671718]]
```




```
Within Set Sum of Squared Error = 358.527731123
Iteration4
[[ 4.02084087  0.08266008]
 [ 6.26427572  5.86742717]
 [ 0.05657819  3.97671718]]
```




```
Within Set Sum of Squared Error = 358.527731123
Final Results:
[[ 4.02084087  0.08266008]
 [ 6.26427572  5.86742717]
 [ 0.05657819  3.97671718]]
```

MLlib KMeans

In [69]:
```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt

# Load and parse the data
data = sc.textFile("data.csv")
parsedData = data.map(lambda line: array([float(x) for x in line.split
(',')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 3, maxIterations=20,
        runs=10, initializationMode="random")
for centroid in clusters.centers:
    print centroid
```

```
[ 6.02635913  6.01593624]
[ 0.00545017  3.97702197]
[ 3.97598146 -0.02083839]
```

There doesn't seem to be much difference between the two codes, besides slightly faster convergence of the first one.

HW10.4.1: Making Homegrown KMeans more efficient

Back to Table of Contents

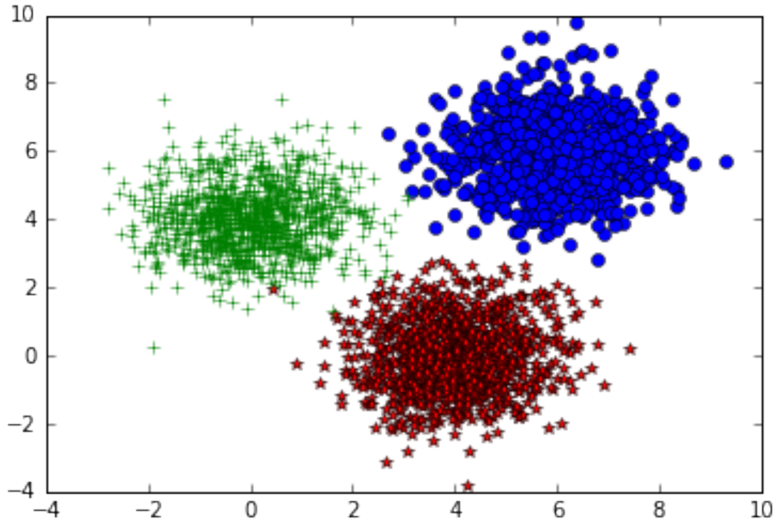
The above provided homegrown KMeans implentation in not the most efficient. How can you make it more efficient? Make this change in the code and show it work and comment on the gains you achieve.

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

This notebook hints to use broadcasting.

In [87]:
```
 %matplotlib inline
import numpy as np
import pylab
import json
size1 = size2 = size3 = 1000
samples1 = np.random.multivariate_normal([4, 0], [[1, 0],[0, 1]], size
1)
data = samples1
samples2 = np.random.multivariate_normal([6, 6], [[1, 0],[0, 1]], size
2)
data = np.append(data,samples2, axis=0)
samples3 = np.random.multivariate_normal([0, 4], [[1, 0],[0, 1]], size
3)
data = np.append(data,samples3, axis=0)
# Randomlize data
data = data[np.random.permutation(size1+size2+size3),]
np.savetxt('data.csv',data,delimiter = ',')
```

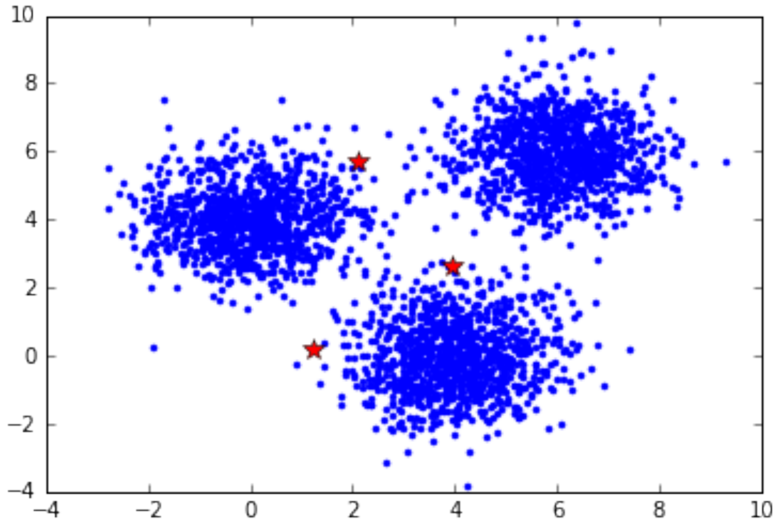
In [88]:
```
pylab.plot(samples1[:, 0], samples1[:, 1],'*', color = 'red')
pylab.plot(samples2[:, 0], samples2[:, 1],'o',color = 'blue')
pylab.plot(samples3[:, 0], samples3[:, 1],'+',color = 'green')
pylab.show()
```



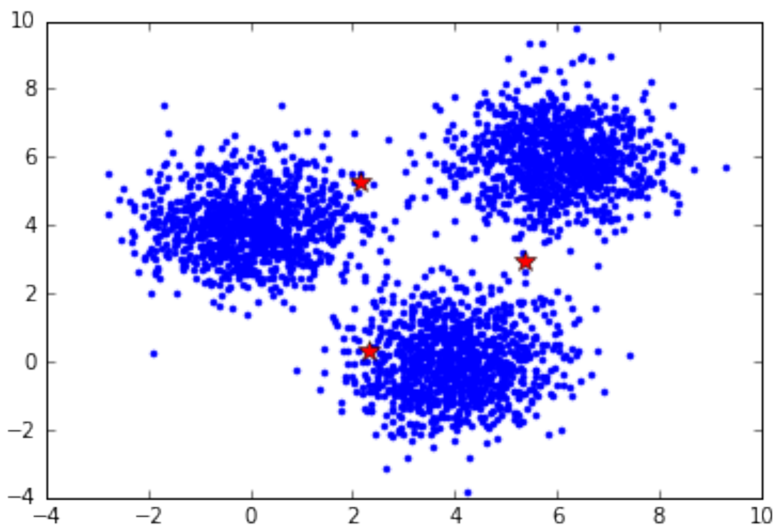
In [89]:
```
import numpy as np

#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))

#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 = 're
d')
    pylab.plot(means[1][0], means[1][1],'*',markersize =10,color = 're
d')
    pylab.plot(means[2][0], means[2][1],'*',markersize =10,color = 're
d')
    pylab.show()
```

In [102]:
```
K = 3
# Initialization: initialization of parameter is fixed to show an exam
ple
centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])

D = sc.textFile("./data.csv") #.cache()
parsedData = D.map(lambda line: array([float(x) for x in line.split(',
')]))
featureLen = len(parsedData.take(1)[0])-1
n = parsedData.count()
learningRate=0.05
w = np.random.normal(size=featureLen) # w should be broadcasted if it
is large
wBroadcast = sc.broadcast(w)   #make available in memory as read-only
to the executors (for mappers and reducers)

iter_num = 0

for i in range(10):
    wBroadcast = sc.broadcast(w)   #make available in memory as read-o
nly to the executors (for mappers and reducers)

    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 cluster size
    if np.sum(np.absolute(centroids_new-centroids))<0.01:
        break
    print "Iteration" + str(iter_num)
    iter_num = iter_num + 1
    centroids = centroids_new
    print centroids
    plot_iteration(centroids)
    w = w - learningRate/n
    WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x
, y: x + y)
    #print("Within Set Sum of Squared Error = " + str(WSSSE))
print "Final Results:"
print centroids
```

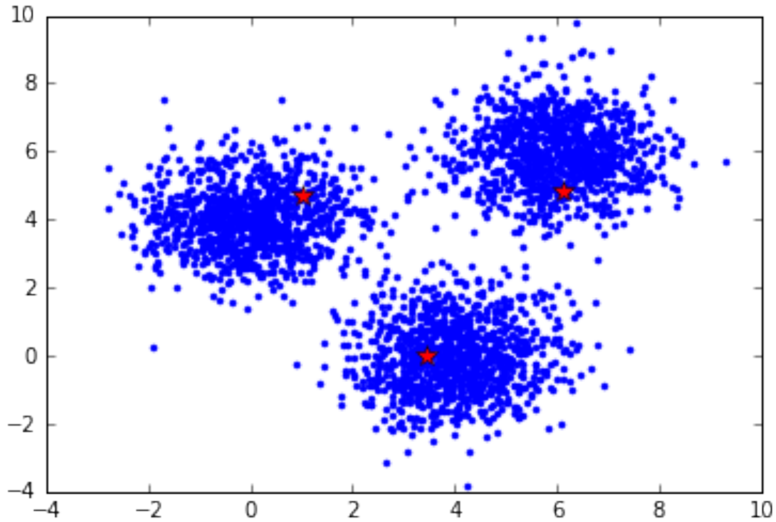
```
Iteration0
[[ 1.22669683  0.16077825]
 [ 3.94327877  2.65087401]
 [ 2.10937398  5.70463155]]
```



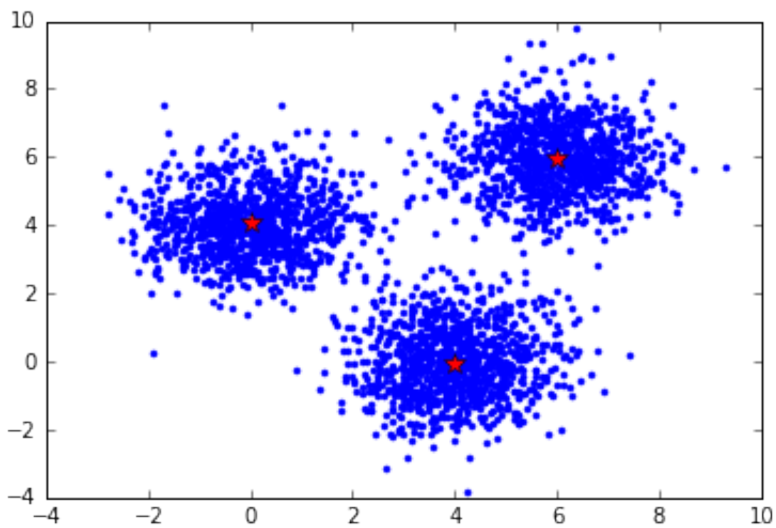

```
Iteration1
[[ 2.33581065  0.31989059]
 [ 5.38171434  2.92142592]
 [ 2.14111435  5.25234035]]
```



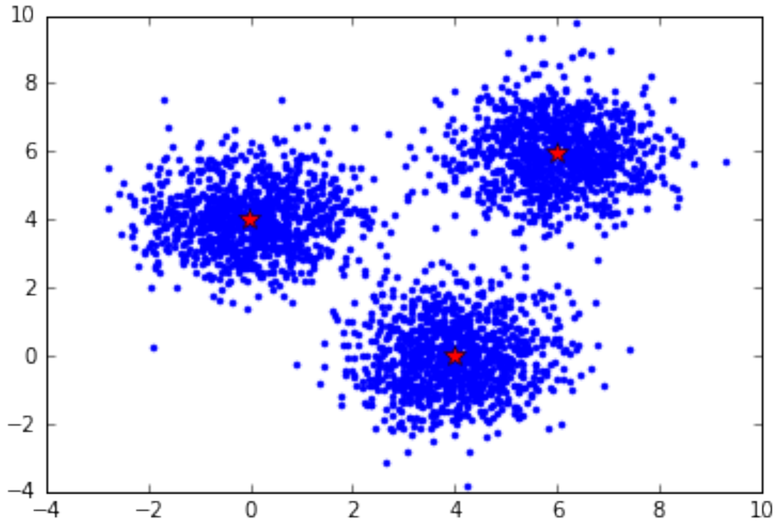

```
Iteration2
[[ 3.45392296 -0.02907791]
 [ 6.10580771  4.83576349]
 [ 1.03525368  4.71383842]]
```




```
Iteration3
[[ 3.97168289 -0.0378151 ]
 [ 6.0099452   5.95150522]
 [ 0.02883925  4.05006856]]
```




```
Iteration4
[[  3.99487381e+00  -2.23365997e-02]
 [  5.98446616e+00   5.97814320e+00]
 [ -4.60516486e-03   4.02860262e+00]]
```



```
Final Results:
[[  3.99487381e+00  -2.23365997e-02]
 [  5.98446616e+00   5.97814320e+00]
 [ -4.60516486e-03   4.02860262e+00]]
```

With broadcasting, we can make the serial tasks run in parallel. As a result, we made this process run faster.

HW10.5: OPTIONAL Weighted KMeans

Back to Table of Contents

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 [71]:
```
## Code goes here
```

In [72]:
```
## Drivers & Runners
```

In [73]:
```
## Run Scripts, S3 Sync
```

HW10.6 OPTIONAL Linear Regression

Back to Table of Contents

HW10.6.1 OPTIONAL Linear Regression

Back to Table of Contents

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.

HW10.6.2 OPTIONAL Linear Regression

Back to Table of Contents

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 [74]:
```
## Code goes here
```

In [75]:
```
## Drivers & Runners
```

In [76]:
```
## Run Scripts, S3 Sync
```

HW10.7 OPTIONAL Error surfaces

Back to Table of Contents

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 [77]:
```
## Code goes here
```

In [78]:
```
## Drivers & Runners
```

In [79]:
```
## Run Scripts, S3 Sync
```

Back to Table of Contents

------- END OF HOWEWORK --------

```
In [ ]:
```