DATASCI W261: Machine Learning at Scale

Group: 10
Names:
Prabhakar Gundugola,
Yi Jin,
Jaime Villalpando

Emails:
prabhakar@berkeley.edu,
yjin@ischool.berkeley.edu,
jaimegvl@ischool.berkeley.edu

Time of Initial Submission: Mar 28, 2016
Week 10: Homework 10
Date: March 28, 2016
Time of Submission: 11:30 PM PT

In [1]:
```
!export SPARK_HOME=/usr/local/spark
```

In [2]:
```
import os
import sys

spark_home = os.environ.get('SPARK_HOME', None)
print 'SPARK_HOME is:' + spark_home

sys.path.insert(0, os.path.join(spark_home, 'python'))
sys.path.insert(0, os.path.join(spark_home, 'python/lib/py4j-0.8.2.1-sr
c.zip'))
execfile(os.path.join(spark_home, 'python/pyspark/shell.py'))
```

```
SPARK_HOME is:/usr/local/spark
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.5.1
      /_/

Using Python version 2.7.10 (default, May 28 2015 17:02:03)
SparkContext available as sc, HiveContext available as sqlContext.
```

HW 10.0: Short answer questions

What is Apache Spark and how is it different to Apache Hadoop? Apache Spark is a fast and general engine for large scale data processing. It can run on top of distributed file systems like Hadoop, GPFS, Tachyon, or in standalone mode.

Difference between Apache Spark and Apache Hadoop

Unlike Apache Hadoop, data processing in Spark is done in memory via RDD (Resilient Distributed Dataset).

Spark provides programmers with an application programming interface centered on a data structure called the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way.

Fill in the blanks:
Spark API consists of interfaces to develop applications based on it in Java, ***Scala, 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 ***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.
An RDD (Resilient Distributed Dataset) is the basic abstraction in Spark. It represents an immutable, partitioned collection of elements that can be operated on in parallel.

The class contains the basic operations available on all RDDs, such as **map, filter,** and **persist**.

RDDs are not materialized until an action is called. This is also known as **lazy evaluation**.

There are 2 types of RDDs -

- Base RDD (only values)
- Pair RDD (key-value pairs)

Example:
text_rdd = sc.textFile('data.csv').cache()
data_points = text_rdd.map(lambda line: [v for v in line.strip().split(',')])
print "First line:", data_points.take(1)

What is lazy evaluation and give an intuitoive example of lazy evaluation and comment on the massive computational savings to be had from lazy evaluation.
Lazy evaluation means something is not computed until an action is called. In Spark, when the RDD is loaded and transformed, Spark just stores the instructions on how the data should be transformed when an action is called. This way, it will have huge computational savings because the data is not really materialized until a result needs to be sent back to the server.

In [105]:
```
# Lazy evaluation example

# Let's take an array of elements
words_list = ['test', 'spark', 'program', 'to', 'find', 'directory']

# Store the array in RDD
words_rdd = sc.parallelize(words_list)

# Retrieve the first element
words_rdd.first()
#words_rdd.saveAsTextFile('hdfs://localhost:54310/user/root/testfile.tx
t')
```

Out[105]:
```
'test'
```

HW 10.1

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 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.

In [3]:
```
!chmod a+x /root/.bashrc
```

In [4]:
```
!hdfs dfs -copyFromLocal MIDS-MLS-HW-10.txt /user/root/wk10/hw101
```

```
copyFromLocal: `/user/root/wk10/hw101/MIDS-MLS-HW-10.txt': File exists
```

In [11]:
```
words_list = ['test', 'spark', 'program', 'to', 'find', 'directory']

words_rdd = sc.parallelize(words_list)

words_rdd.collect()
words_rdd.saveAsTextFile('hdfs://localhost:54310/user/root/testfile.tx
t')
```

In [7]:
```
import re

# Function to preprocess the line - remove punctuation and then split th
e line.
def preprocess_text(line):
    line = re.sub(u'[^A-Za-z0-9 ]+', '', line)
    return line.strip().split()

# Now parallelize the words in the text file, do the word count and then
sort
words_list = sc.textFile('hdfs://localhost:54310/user/root/wk10/hw101/MI
DS-MLS-HW-10.txt')
words_rdd = words_list.flatMap(preprocess_text).map(lambda x:(x,1)).redu
ceByKey(lambda x,y:x+y)
words_sorted = words_rdd.map(lambda x: (x[1], x[0])).sortByKey(ascending
=False).collect()

print "{:10s}\t{:10s}".format('Word', 'Count')
print "======================="
for count, word in words_sorted:
    print "{:10s}\t{:10s}".format(word, str(count))
```

```
Word            Count
========================
the             46
and             24
in              18
of              17
HW              13
a               12
for             11
data            10
code            10
to              9
is              8
with            7
this            7
Using           7
on              7
your            7
KMeans          7
clusters        6
iterations      5
from            5
as              5
regression      5
What            4
words           4
HW103           4
Sum             4
100             4
Comment         4
following       4
Squared         4
Spark           4
model           4
using           4
each            4
set             4
linear          4
example         4
one             4
x               4
Set             4
provided        4
list            3
findings        3
evaluation      3
available       3
lazy            3
training        3
count           3
Please          3
report          3
NOTE            3
```

```
Errors          3
plot            3
results         3
Within          3
import          3
after           3
or              3
plots           3
it              3
an              3
document        3
notebook        3
LASS0           2
Explain         2
httpswwwdropboxcomsq85t0ytb9apggnhkmeansdatatxtdl0      2
languages       2
Apache          2
homework        2
1061            2
per             2
Report          2
run             2
here            2
OPTIONAL        2
kmeansdatatxt   2
RIDGE           2
word            2
provide         2
snippet         2
y               2
myModelPath     2
how             2
Generate        2
follows         2
testing         2
frequency       2
X               2
between         2
be              2
found           2
team            2
via             2
point           2
that            2
differences     2
10              2
up              2
vector          2
cluster         2
any             2
blanks          2
In              2
repeat          2
```

```
2               2
center          2
points          2
decreasing      2
at              2
order           2
Fill            2
sameModel       1
all             1
strWSSSE        1
not             1
intuitoive      1
questions       1
carefully       1
03152016        1
write           1
group           1
implementation  1
HW105           1
Modify          1
Here            1
weeks           1
submissions     1
case            1
return          1
load            1
homeworks       1
compute         1
bringing        1
SPECIAL         1
runs            1
resource        1
DATSCI          1
math            1
Learning        1
where           1
datamaplambda   1
Java            1
generation      1
begin           1
consists        1
END             1
Run             1
please          1
labeled         1
cell            1
3               1
above           1
exercise        1
HW10            1
This            1
W261            1
interfaces      1
```

```
modify          1
20              1
fun             1
102             1
based           1
104             1
parameters      1
Weight          1
column          1
completing      1
SQRTXX          1
length          1
resulting       1
Download        1
UC              1
comment         1
letters         1
Gradient        1
distributed     1
done            1
initializationModerandom        1
array           1
clustering      1
use             1
106             1
submit          1
X2              1
Homeworks       1
numpy           1
sort            1
errorpoint      1
form            1
httpsdocsgooglecomformsd1ZOr9RnIeA06AcZDB6K1mJN4vrLeSmS2PD6Xm3eOiisvi
ewformuspsendform       1
tokens          1
forward         1
pysparkmllibclustering  1
line            1
clusterssavesc  1
had             1
Load            1
MLlibcentric    1
iteration       1
single          1
See             1
norm            1
clusterscentersclusterspredictpoint     1
def             1
sqrtsumx2       1
links           1
httpswwwdropboxcoms3nsthvp8g2rrrdhEMKmeansipynbdl0      1
parse           1
Call            1
```

```
tab             1
SQRTX12         1
DATA            1
WSSSE           1
different       1
develop         1
sqrt            1
answer          1
Machine         1
driver          1
regularization  1
tune            1
lower           1
occurence       1
Mesos           1
maxIterations10 1
savings         1
1               1
algorithms      1
arrayfloatx     1
hyper           1
show            1
text            1
Hadoop          1
KMeansModelloadsc        1
follow          1
Final           1
Euclidean       1
X22             1
inverse         1
Kmean           1
Linear          1
going           1
Berkeley        1
1062            1
do              1
good            1
get             1
evaluate        1
framework       1
made            1
progress        1
sorted          1
dataset         1
HW104           1
instructions    1
homegrown       1
server          1
assignments     1
API             1
either          1
output          1
runs10          1
```

```
Again           1
often           1
ASSIGNMENT      1
back            1
experiments     1
located         1
Build           1
measure         1
experiements    1
Save            1
parsedData      1
Evaluate        1
separate        1
find            1
MLlibs          1
experiment      1
weighted        1
Regression      1
1011            1
Short           1
creating        1
Team            1
103             1
101             1
by              1
105             1
la              1
ISVC            1
massive         1
first           1
Then            1
computational   1
1X              1
number          1
Scale           1
DropBox         1
management      1
give            1
appears         1
INSTURCTIONS    1
program         1
Plot            1
hundred         1
thru            1
KMeanstrainparsedData   1
Justify         1
more            1
httpswwwdropboxcomsatzqkc0p1eajuz6LinearRegressionNotebookChallengeip
ynbdl0  1
Assignments     1
train           1
MIDS            1
parsedDatamaplambda     1
```

```
work            1
can             1
KMeansModel     1
sets            1
Your            1
are             1
LinearRegressionWithSGD 1
computing       1
X1              1
MLLib           1
manner          1
az              1
httpsdocsgooglecomspreadsheetsd1ncFQl5Tovn16slD8mYjPnzMTPSfiGeLLzW8vs
Mjgeditusp sharing      1
documents       1
KMEans          1
descent         1
MLLibs          1
instance        1
Error           1
other           1
printWithin     1
sctextFilekmeansdatatxt 1
assignment      1
applications    1
linesplit       1
such            1
Teams           1
weightX         1
errorpointreducelambda  1
RDD             1
V13             1
element         1
```

HW 10.1.1

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 [8]:
```
def preprocess_lowertext(line):
    line = re.sub(u'[^A-Za-z0-9 ]+', '', line)
    return line.strip().split()

# Now parallelize the words in the text file, filter the words, do the w
ord count and then sort
words_list = sc.textFile('hdfs://localhost:54310/user/root/wk10/hw101/MI
DS-MLS-HW-10.txt')
lower_words_rdd = words_list.flatMap(preprocess_text).filter(lambda x: x
[0].islower()).map(lambda x:(x,1)).reduceByKey(lambda x,y:x+y)
lower_words_sorted = lower_words_rdd.map(lambda x: (x[1], x[0])).sortByK
ey(ascending=False).collect()

print "{:10s}\t{:10s}".format('Word', 'Count')
print "======================="
for count, word in lower_words_sorted:
    print "{:10s}\t{:10s}".format(word, str(count))
```

```
Word            Count
========================
the             46
and             24
in              18
of              17
a               12
for             11
data            10
code            10
to              9
is              8
with            7
this            7
on              7
your            7
clusters        6
iterations      5
from            5
as              5
regression      5
words           4
following       4
model           4
using           4
each            4
x               4
set             4
linear          4
one             4
example         4
provided        4
report          3
list            3
findings        3
evaluation      3
available       3
lazy            3
training        3
count           3
results         3
import          3
plots           3
or              3
plot            3
it              3
an              3
document        3
after           3
notebook        3
httpswwwdropboxcomsq85t0ytb9apggnhkmeansdatatxtdl0      2
languages       2
homework        2
```

```
per             2
run             2
here            2
word            2
kmeansdatatxt   2
provide         2
snippet         2
y               2
myModelPath     2
follows         2
vector          2
testing         2
frequency       2
between         2
be              2
found           2
team            2
how             2
via             2
point           2
that            2
differences     2
up              2
at              2
cluster         2
any             2
blanks          2
repeat          2
center          2
points          2
decreasing      2
order           2
sameModel       1
load            1
all             1
intuitoive      1
questions       1
carefully       1
write           1
group           1
implementation  1
had             1
weeks           1
submissions     1
case            1
return          1
homeworks       1
compute         1
bringing        1
runs            1
resource        1
exercise        1
where           1
```

```
datamaplambda   1
generation      1
strWSSSE        1
consists        1
please          1
cell            1
above           1
math            1
interfaces      1
modify          1
not             1
based           1
parameters      1
column          1
clusterscentersclusterspredictpoint     1
completing      1
length          1
resulting       1
comment         1
letters         1
distributed     1
done            1
initializationModerandom        1
array           1
clustering      1
use             1
submit          1
forward         1
numpy           1
sort            1
errorpoint      1
form            1
lower           1
tokens          1
pysparkmllibclustering  1
line            1
clusterssavesc  1
iteration       1
labeled         1
norm            1
fun             1
def             1
sqrtsumx2       1
links           1
httpswwwdropboxcoms3nsthvp8g2rrrdhEMKmeansipynbdl0      1
parse           1
single          1
tab             1
different       1
develop         1
sqrt            1
answer          1
begin           1
```

```
driver          1
regularization  1
tune            1
httpsdocsgooglecomformsd1ZOr9RnIeA06AcZDB6K1mJN4vrLeSmS2PD6Xm3eOiisvi
ewformuspsendform       1
occurence       1
maxIterations10 1
savings         1
algorithms      1
arrayfloatx     1
hyper           1
show            1
text            1
experiments     1
follow          1
find            1
inverse         1
la              1
program         1
do              1
good            1
get             1
evaluate        1
framework       1
made            1
progress        1
sorted          1
dataset         1
instructions    1
homegrown       1
server          1
assignments     1
either          1
runs10          1
often           1
back            1
located         1
are             1
measure         1
experiements    1
parsedData      1
separate        1
experiment      1
creating        1
output          1
by              1
massive         1
first           1
computational   1
number          1
weighted        1
management      1
give            1
```

```
appears         1
going           1
thru            1
more            1
httpswwwdropboxcomsatzqkc0p1eajuz6LinearRegressionNotebookChallengeip
ynbdl0  1
train           1
hundred         1
parsedDatamaplambda     1
work            1
can             1
computing       1
manner          1
az              1
httpsdocsgooglecomspreadsheetsd1ncFQl5Tovn16slD8mYjPnzMTPSfiGeLLzW8vs
Mjgedituspsharing       1
documents       1
descent         1
instance        1
other           1
printWithin     1
sctextFilekmeansdatatxt 1
assignment      1
applications    1
linesplit       1
such            1
weightX         1
errorpointreducelambda  1
sets            1
element         1
```

HW 10.2: KMeans a la MLLib

Using the following MLlib-centric KMeans code snippet:

NOTE 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)](https://www.dropbox.com/s/q85t0ytb9apggnh/kmeans_data.txt?dl=0)

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

In [23]:
```
!hdfs dfs -copyFromLocal kmeans_data.txt /user/root/wk10/hw102/
```

In [9]:
```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt

# 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)]))


def run_kmeans(filename, k, iterations, delimiter=' '):

    global clusters

    # Load and parse the data
    # NOTE  kmeans_data.txt is available here
    #          https://www.dropbox.com/s/q85t0ytb9apggnh/kmeans_data.tx
t?dl=0
    #data = sc.textFile("hdfs://localhost:54310/user/root/wk10/hw102/kme
ans_data.txt")
    data = sc.textFile(filename)
    parsedData = data.map(lambda line: array([float(x) for x in line.spl
it(delimiter)]))

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


    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
    myModelPath = filename.replace(filename.split('/')[-1], 'myModelPat
h')
    clusters.save(sc, myModelPath)
    sameModel = KMeansModel.load(sc, myModelPath)

    return [sameModel, WSSSE]
```

In [12]:
```
!hdfs dfs -rm -r /user/root/wk10/hw102/myModelPath
filename = "hdfs://localhost:54310/user/root/wk10/hw102/kmeans_data.txt"
centroids = run_kmeans(filename, 2, 1)
print "Number of iterations - 1 \n"
print centroids[0].clusterCenters, '\n\n'

!hdfs dfs -rm -r /user/root/wk10/hw102/myModelPath
print '\n\n'
centroids = run_kmeans(filename, 2, 5)
print "Number of iterations - 5 \n"
print centroids[0].clusterCenters
```

```
16/03/28 23:21:09 INFO fs.TrashPolicyDefault: Namenode trash configurat
ion: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /user/root/wk10/hw102/myModelPath
Within Set Sum of Squared Error = 0.692820323028
Number of iterations - 1

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


16/03/28 23:21:13 INFO fs.TrashPolicyDefault: Namenode trash configurat
ion: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /user/root/wk10/hw102/myModelPath



Within Set Sum of Squared Error = 0.692820323028
Number of iterations - 5

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

Answer: Given such a simple data set, the model converges very rapidly in just 5 iterations and the lower WSSSE value also helps that fact.

HW 10.3

Download the following KMeans notebook:

https://www.dropbox.com/s/3nsthvp8g2rrrdh/EM-Kmeans.ipynb?dl=0
(https://www.dropbox.com/s/3nsthvp8g2rrrdh/EM-Kmeans.ipynb?dl=0)

Generate 3 clusters with 100 (one hundred) data points per cluster (using the code provided). Plot the data.

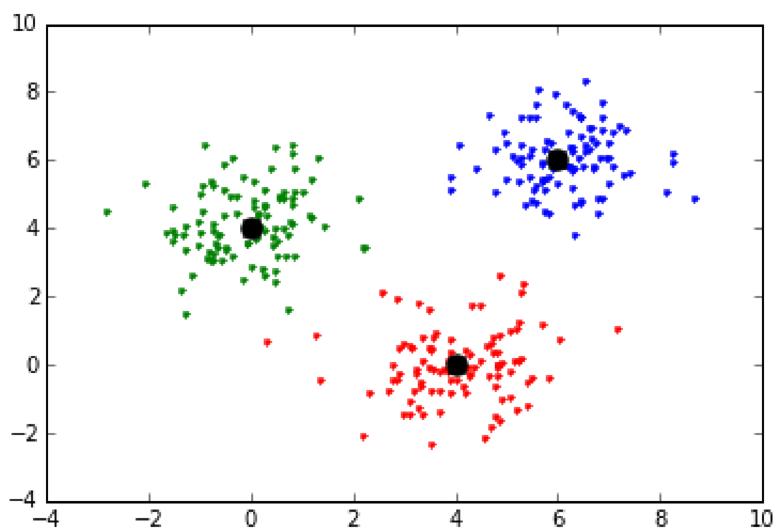
Data Generation

In [51]:
```
%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 = ',', fmt='%.8f')
```

In [52]:
```
!hdfs dfs -copyFromLocal data.csv /user/root/wk10/hw103
```

Data Visualization

In [53]:
```
def plot_samples(title, means):

    pylab.plot(samples1[:, 0], samples1[:, 1],'.', color = 'red')
    pylab.plot(samples2[:, 0], samples2[:, 1],'.',color = 'blue')
    pylab.plot(samples3[:, 0], samples3[:, 1],'.',color = 'green')

    pylab.plot(means[0][0], means[0][1],'o', markersize=10, color = 'bla
ck')
    pylab.plot(means[1][0], means[1][1],'o', markersize=10, color = 'bla
ck')
    pylab.plot(means[2][0], means[2][1],'o', markersize=10, color = 'bla
ck')


    pylab.show()

plot_samples('Plotting samples', [[4,0],[6,6],[0,4]])
```



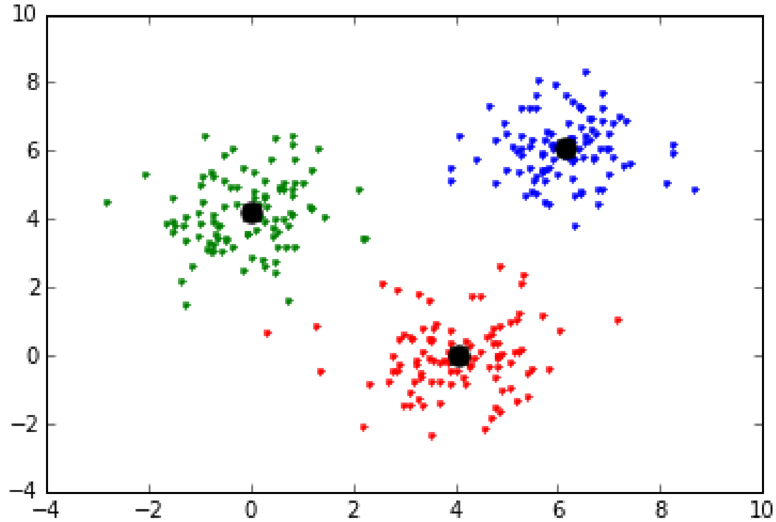
Then run MLlib's Kmean implementation on this data and report your results as follows:

-- 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. Comment on the progress of this measure as the KMEans algorithms runs for more iterations

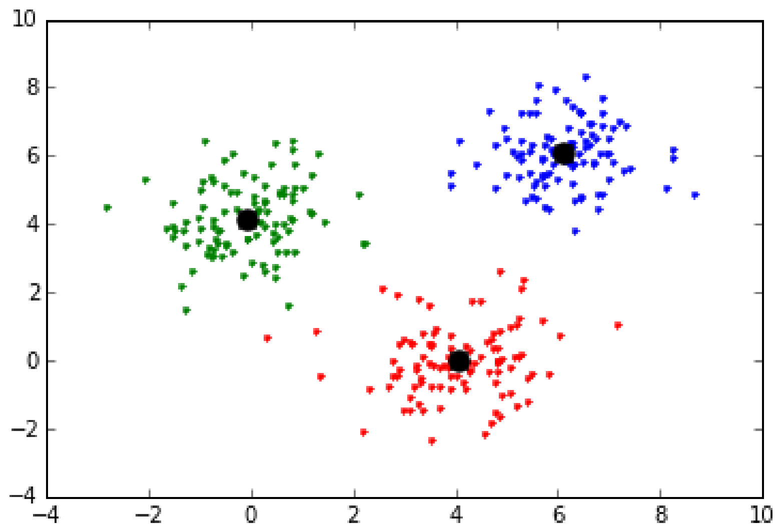
In [56]:
```
# HW 10.3 - Main driver program

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

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


def run_mllib_kmeans(filename, k, iterations):

    # Load and parse the data
    # NOTE  kmeans_data.txt is available here
    #          https://www.dropbox.com/s/q85t0ytb9apggnh/kmeans_data.tx
t?dl=0
    #data = sc.textFile("hdfs://localhost:54310/user/root/wk10/hw102/kme
ans_data.txt")
    data = sc.textFile(filename)
    parsedData = data.map(lambda line: array([float(x) for x in line.spl
it(',')])).cache()

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


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

    # Save and load model
    myModelPath = filename.replace(filename.split('/')[-1], 'myModelPat
h')
    clusters.save(sc, myModelPath)
    sameModel = KMeansModel.load(sc, myModelPath)

    return [sameModel, WSSSE]
```

In [57]:
```
centroids_list = []
ks = [1, 10, 20, 100]

for k in ks:
    filename = 'hdfs://localhost:54310/user/root/wk10/hw103/data.csv'
    !hdfs dfs -rm -r {filename.replace('data.csv', 'myModelPath')}
    print ''
    print "Iteration - ", k
    centroids = run_mllib_kmeans(filename, 3, k)
    centroids_list.append(centroids)
    print str(centroids[0].clusterCenters) + '\n'
    plot_samples('Plotting the clusters', centroids[0].clusterCenters)
```

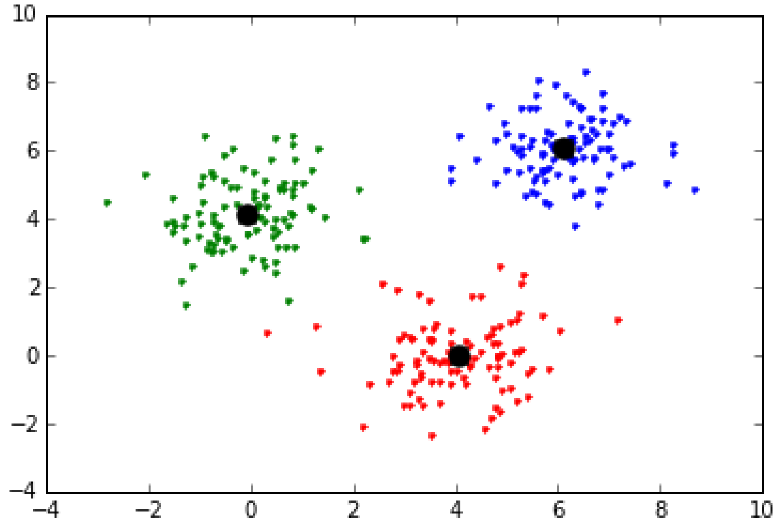
```
16/03/29 00:07:55 INFO fs.TrashPolicyDefault: Namenode trash configurat
ion: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hdfs://localhost:54310/user/root/wk10/hw103/myModelPath

Iteration -  1
Within Set Sum of Squared Error = 359.93815234
(DenseVector([6.148, 6.0969]), DenseVector([-0.0148, 4.1781]), DenseVec
tor([4.0588, 0.0025]))
```



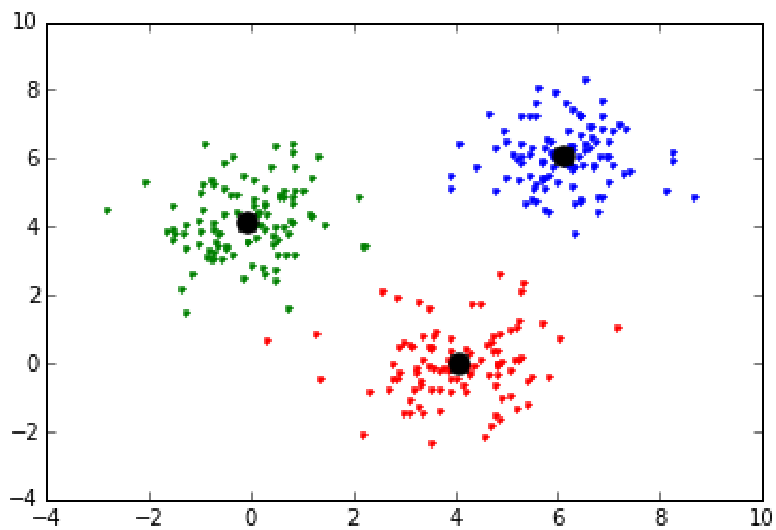
```
16/03/29 00:08:00 INFO fs.TrashPolicyDefault: Namenode trash configurat
ion: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hdfs://localhost:54310/user/root/wk10/hw103/myModelPath

Iteration -  10
Within Set Sum of Squared Error = 359.930084502
(DenseVector([6.1028, 6.0819]), DenseVector([-0.0922, 4.1549]), DenseVe
ctor([4.0588, 0.0025]))
```



```
16/03/29 00:08:04 INFO fs.TrashPolicyDefault: Namenode trash configurat
ion: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hdfs://localhost:54310/user/root/wk10/hw103/myModelPath

Iteration -  20
Within Set Sum of Squared Error = 359.930084502
(DenseVector([6.1028, 6.0819]), DenseVector([4.0588, 0.0025]), DenseVec
tor([-0.0922, 4.1549]))
```



```
16/03/29 00:08:09 INFO fs.TrashPolicyDefault: Namenode trash configurat
ion: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hdfs://localhost:54310/user/root/wk10/hw103/myModelPath

Iteration -  100
Within Set Sum of Squared Error = 359.930084502
(DenseVector([6.1028, 6.0819]), DenseVector([-0.0922, 4.1549]), DenseVe
ctor([4.0588, 0.0025]))
```



HW 10.4:

Using the KMeans code (homegrown code) provided repeat the experiments in HW10.3. Comment on any differences between the results in HW10.3 and HW10.4. Explain.

In [58]:
```
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 [68]:

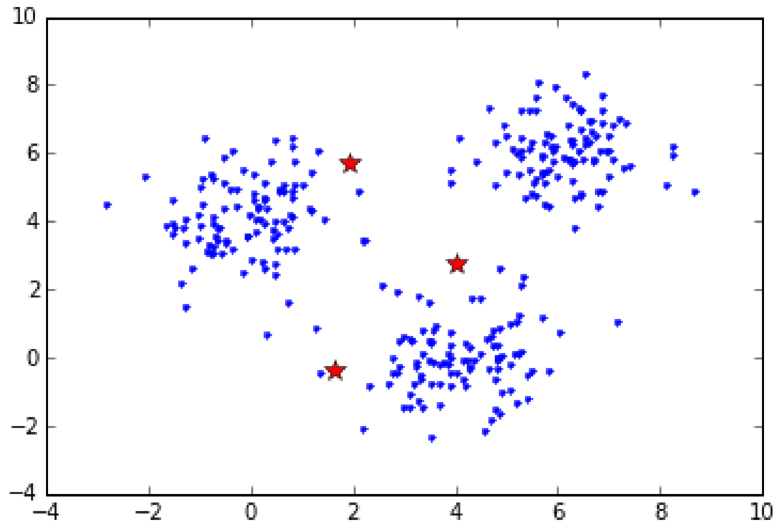
```
```

```
def error(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)]))

def run_kmeans(filename, k, iterations):
    # Initialization: initialization of parameter is fixed to show an ex
ample
    global centroids

    centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])

    D = sc.textFile(filename).cache()

    centroids_list = []
    centroids_list.append(centroids)
    iter_num = 0

    print "\n\n\n{:15s} {:<24s} {:<24s} {:<24s}".format('Iteration', 'Ce
ntroid 1', 'Centroid 2', 'Centroid 3')
    print "==================================================================
========================"

    format_str = "{:<15d} ({: 8.4f}, {: 8.4f}) ({: 8.4f}, {: 8.4f}) ({:
8.4f}, {: 8.4f})"
    print format_str.format(iter_num, centroids[0][0],centroids[0][1],
                            centroids[1][0], centroids[1][1],
                            centroids[2][0], centroids[2][1])

    for i in range(iterations):
        res = D.map(nearest_centroid).reduceByKey(lambda x,y : (x[0]+y
[0],x[1]+y[1])).collect()
        #res [(0, (array([  2.66546663e+00,   3.94844436e+03]), 1001)
),
        #     (2, (array([ 6023.84995923,  5975.48511018]), 1000)),
        #     (1, (array([ 3986.85984761,    15.93153464]), 999))]
        # res[1][1][1] returns 1000 here
        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])  #divid
e by cluster size
        if np.sum(np.absolute(centroids_new-centroids))<0.01:
            break
        iter_num = iter_num + 1
        centroids = centroids_new
        print format_str.format(iter_num,centroids[0][0],centroids[0]
[1],
                            centroids[1][0], centroids[1][1],
                            centroids[2][0], centroids[2][1])

    WSSE = D.map(error).reduce(lambda x,y:x+y)
    print "\nFinal Results (WSSE):", WSSE
    print "\n\n"
```

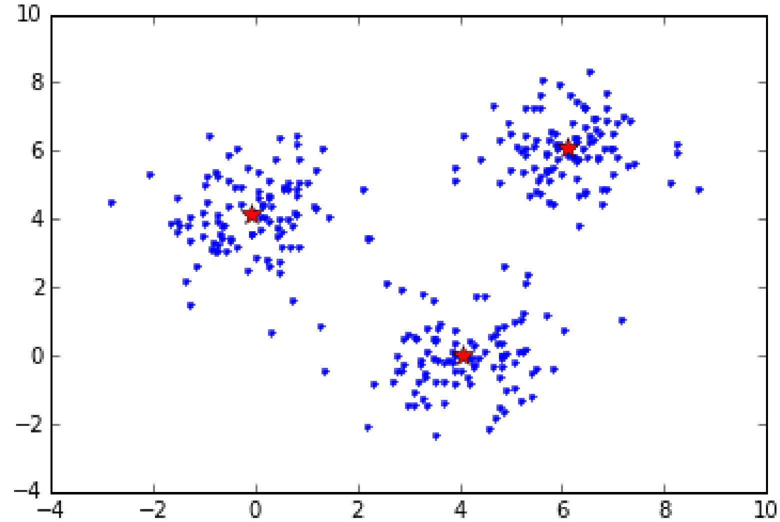
```
plot_iteration(centroids)
#print centroids
```

In [69]:
```
filename = 'hdfs://localhost:54310/user/root/wk10/hw103/data.csv'

ks = [1,10,20,100]

for k in ks:
    run_kmeans(filename, 3, k)
```

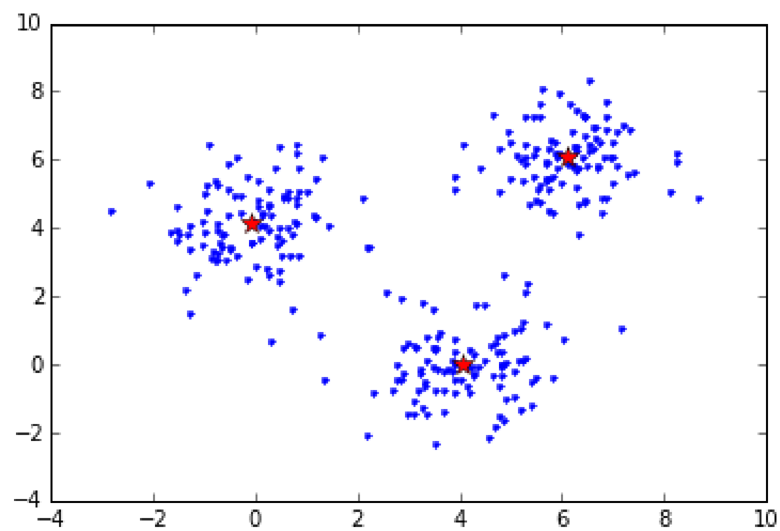
```
Iteration       Centroid 1                 Centroid 2                 Centr
oid 3
========================================================================
==============
0               (  0.0000,   0.0000) (  2.0000,   2.0000) (  0.0000,
7.0000)
1               (  1.6373,  -0.3811) (  4.0208,   2.7470) (  1.9430,
5.6945)

Final Results (WSSE): 868.784399924
```



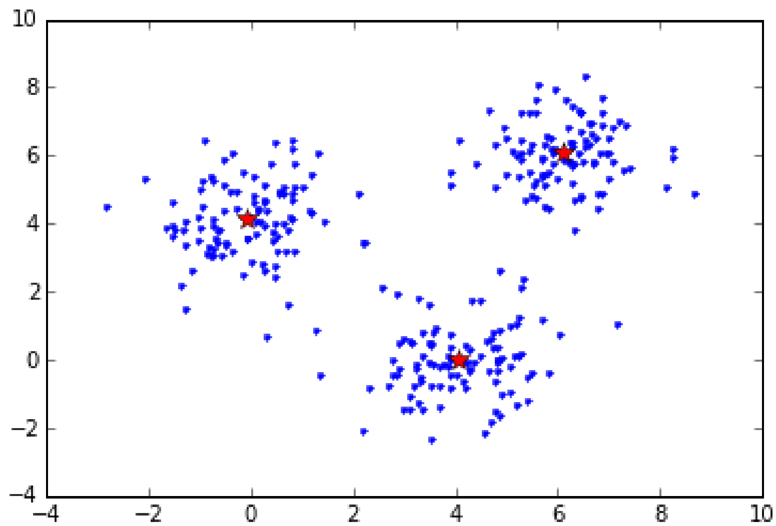

```
Iteration       Centroid 1                 Centroid 2                 Centr
oid 3
========================================================================
==============
0               (  0.0000,   0.0000) (  2.0000,   2.0000) (  0.0000,
7.0000)
1               (  1.6373,  -0.3811) (  4.0208,   2.7470) (  1.9430,
5.6945)
2               (  3.1573,  -0.2199) (  5.6736,   3.7761) (  1.4949,
5.0594)
3               (  3.8811,  -0.1056) (  6.0987,   5.7969) ( -0.0235,
4.2508)
4               (  4.0588,   0.0025) (  6.1028,   6.0819) ( -0.0922,
4.1549)

Final Results (WSSE): 359.930084502
```



```
Iteration       Centroid 1                Centroid 2                Centr
oid 3
============================================================================
==============
0               (  0.0000,   0.0000) (  2.0000,   2.0000) (  0.0000,
7.0000)
1               (  1.6373,  -0.3811) (  4.0208,   2.7470) (  1.9430,
5.6945)
2               (  3.1573,  -0.2199) (  5.6736,   3.7761) (  1.4949,
5.0594)
3               (  3.8811,  -0.1056) (  6.0987,   5.7969) ( -0.0235,
4.2508)
4               (  4.0588,   0.0025) (  6.1028,   6.0819) ( -0.0922,
4.1549)

Final Results (WSSE): 359.930084502
```



```
Iteration       Centroid 1               Centroid 2               Centr
oid 3
================================================================================
==============
0               (  0.0000,   0.0000) (  2.0000,   2.0000) (  0.0000,
7.0000)
1               (  1.6373,  -0.3811) (  4.0208,   2.7470) (  1.9430,
5.6945)
2               (  3.1573,  -0.2199) (  5.6736,   3.7761) (  1.4949,
5.0594)
3               (  3.8811,  -0.1056) (  6.0987,   5.7969) ( -0.0235,
4.2508)
4               (  4.0588,   0.0025) (  6.1028,   6.0819) ( -0.0922,
4.1549)

Final Results (WSSE): 359.930084502
```



As observed in 10.3 and 10.4, both homegrown and MLLib implementations converge rapidly by the 4th iteration and produce identical results at the end of the 4th iteration.

HW 10.5:

Using the KMeans code provided modify it to do a weighted KMeans and repeat the experiements in HW10.3. Comment on any differences between the results in HW10.3 and HW10.5. Explain.

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 X1 and X2.

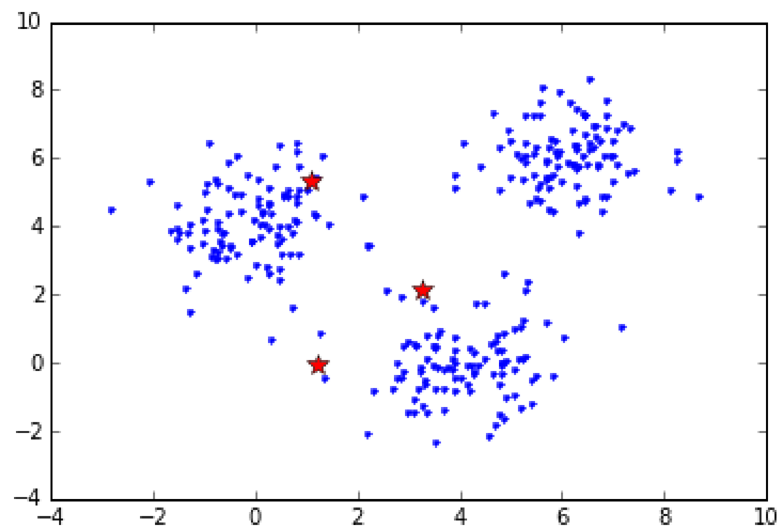
In [71]:

```
from numpy import linalg
# vector - (cluster_id, (x,y))
def weighted(vector):
    cluster_id = vector[0]
    x = vector[1][0]
    w = 1.0/linalg.norm(x)
    return cluster_id, (x*w, w)



def run_weighted_kmeans(filename, k, iterations):
    # Initialization: initialization of parameter is fixed to show an ex
ample
    global centroids

    centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])

    D = sc.textFile(filename).cache()

    centroids_list = []
    centroids_list.append(centroids)
    iter_num = 0

    print "\n\n\n{:15s} {:<24s} {:<24s} {:<24s}".format('Iteration', 'Ce
ntroid 1', 'Centroid 2', 'Centroid 3')
    print "==============================================================
======================="

    format_str = "{:<15d} ({: 8.4f}, {: 8.4f}) ({: 8.4f}, {: 8.4f}) ({:
8.4f}, {: 8.4f})"
    print format_str.format(iter_num, centroids[0][0],centroids[0][1],
                            centroids[1][0], centroids[1][1],
                            centroids[2][0], centroids[2][1])

    for i in range(iterations):
        res = D.map(nearest_centroid).map(weighted).reduceByKey(lambda
x,y : (x[0]+y[0],x[1]+y[1])).collect()
        #res [(0, (array([  2.66546663e+00,   3.94844436e+03]), 1001)
),
        #     (2, (array([ 6023.84995923,  5975.48511018]), 1000)),
        #     (1, (array([ 3986.85984761,    15.93153464]), 999))]
        # res[1][1][1] returns 1000 here
        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])  #divid
e by cluster size
        if np.sum(np.absolute(centroids_new-centroids))<0.01:
            break
        iter_num = iter_num + 1
        centroids = centroids_new
        print format_str.format(iter_num,centroids[0][0],centroids[0]
[1],
                            centroids[1][0], centroids[1][1],
```

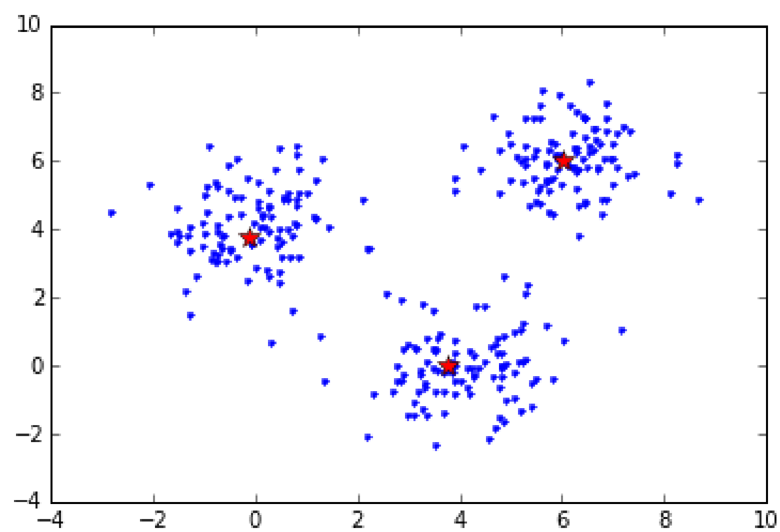
```
                                centroids[2][0], centroids[2][1])

WSSE = D.map(error).reduce(lambda x,y:x+y)
print "\nFinal Results (WSSE):", WSSE
print "\n\n"
plot_iteration(centroids)
#print centroids
```

In [73]:
```
filename = 'hdfs://localhost:54310/user/root/wk10/hw103/data.csv'

ks = [1,10,20,100]

for k in ks:
    run_weighted_kmeans(filename, 3, k)
```

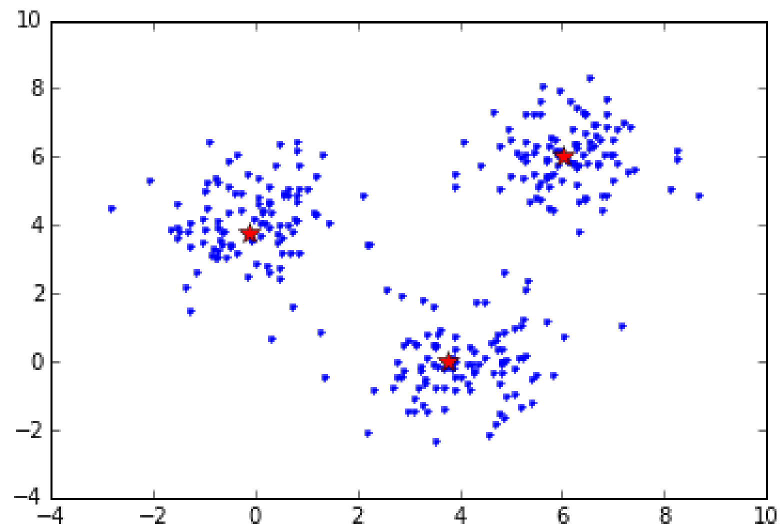
```
Iteration       Centroid 1                Centroid 2                Centr
oid 3
=========================================================================
==============
0               (  0.0000,   0.0000) (  2.0000,   2.0000) (  0.0000,
7.0000)
1               (  1.2029,  -0.0571) (  3.2624,   2.1485) (  1.0991,
5.3271)

Final Results (WSSE): 893.635715411
```



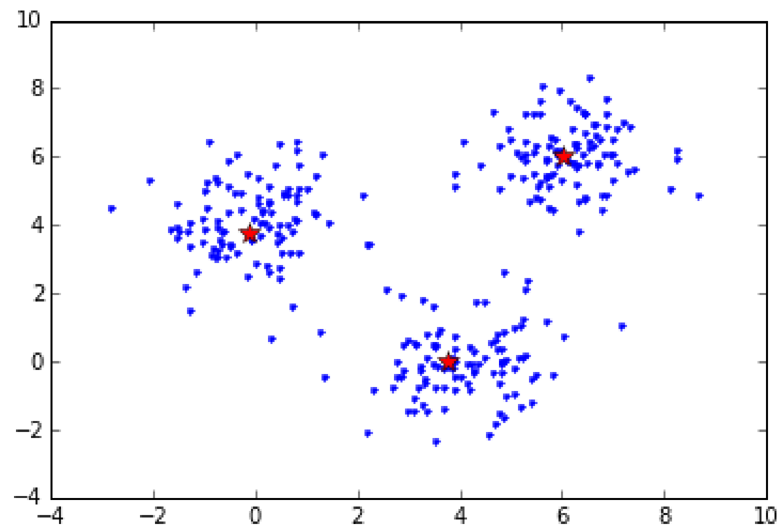
```
Iteration        Centroid 1                Centroid 2                Centr
oid 3
================================================================================
==============
0                (  0.0000,   0.0000) (  2.0000,   2.0000) (  0.0000,
7.0000)
1                (  1.2029,  -0.0571) (  3.2624,   2.1485) (  1.0991,
5.3271)
2                (  2.1327,  -0.1333) (  4.8421,   2.0834) (  0.7358,
4.5418)
3                (  3.0319,  -0.2604) (  5.4663,   3.6545) ( -0.0490,
4.0296)
4                (  3.5055,  -0.0999) (  5.9444,   5.4897) ( -0.1221,
3.9210)
5                (  3.7754,  -0.0465) (  6.0188,   5.9632) ( -0.1006,
3.7600)
6                (  3.7832,  -0.0273) (  6.0365,   6.0141) ( -0.1006,
3.7600)

Final Results (WSSE): 366.670795749
```



```
Iteration        Centroid 1                Centroid 2                Centr
oid 3
================================================================================
==============
0                (  0.0000,   0.0000) (  2.0000,   2.0000) (  0.0000,
7.0000)
1                (  1.2029,  -0.0571) (  3.2624,   2.1485) (  1.0991,
5.3271)
2                (  2.1327,  -0.1333) (  4.8421,   2.0834) (  0.7358,
4.5418)
3                (  3.0319,  -0.2604) (  5.4663,   3.6545) ( -0.0490,
4.0296)
4                (  3.5055,  -0.0999) (  5.9444,   5.4897) ( -0.1221,
3.9210)
5                (  3.7754,  -0.0465) (  6.0188,   5.9632) ( -0.1006,
3.7600)
6                (  3.7832,  -0.0273) (  6.0365,   6.0141) ( -0.1006,
3.7600)

Final Results (WSSE): 366.670795749
```



```
Iteration        Centroid 1                 Centroid 2                 Centr
oid 3
================================================================================
==============
0                (  0.0000,   0.0000) (  2.0000,   2.0000) (  0.0000,
7.0000)
1                (  1.2029,  -0.0571) (  3.2624,   2.1485) (  1.0991,
5.3271)
2                (  2.1327,  -0.1333) (  4.8421,   2.0834) (  0.7358,
4.5418)
3                (  3.0319,  -0.2604) (  5.4663,   3.6545) ( -0.0490,
4.0296)
4                (  3.5055,  -0.0999) (  5.9444,   5.4897) ( -0.1221,
3.9210)
5                (  3.7754,  -0.0465) (  6.0188,   5.9632) ( -0.1006,
3.7600)
6                (  3.7832,  -0.0273) (  6.0365,   6.0141) ( -0.1006,
3.7600)

Final Results (WSSE): 366.670795749
```



HW 10.6: Linear Regression

HW 10.6.1 Using the following linear regression notebook:

https://www.dropbox.com/s/atzqkc0p1eajuz6/LinearRegression-Notebook-Challenge.ipynb?dl=0 (https://www.dropbox.com/s/atzqkc0p1eajuz6/LinearRegression-Notebook-Challenge.ipynb?dl=0)

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 and words

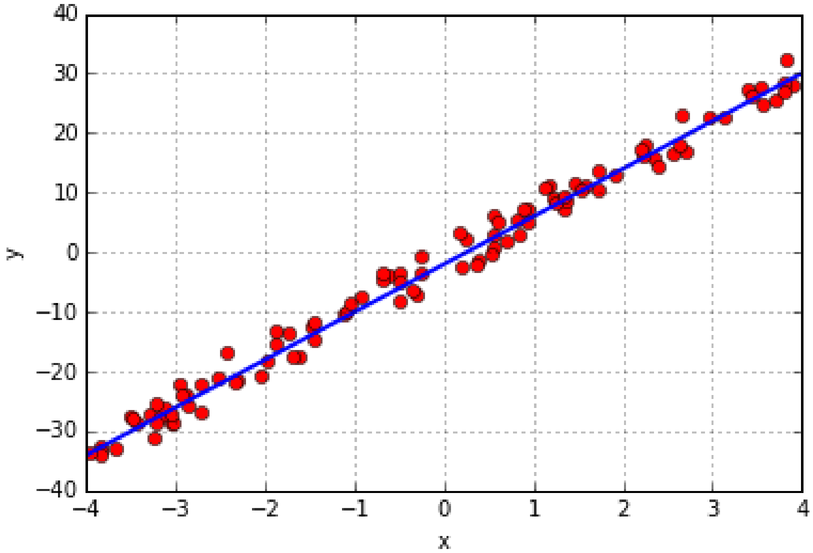
Data Generation

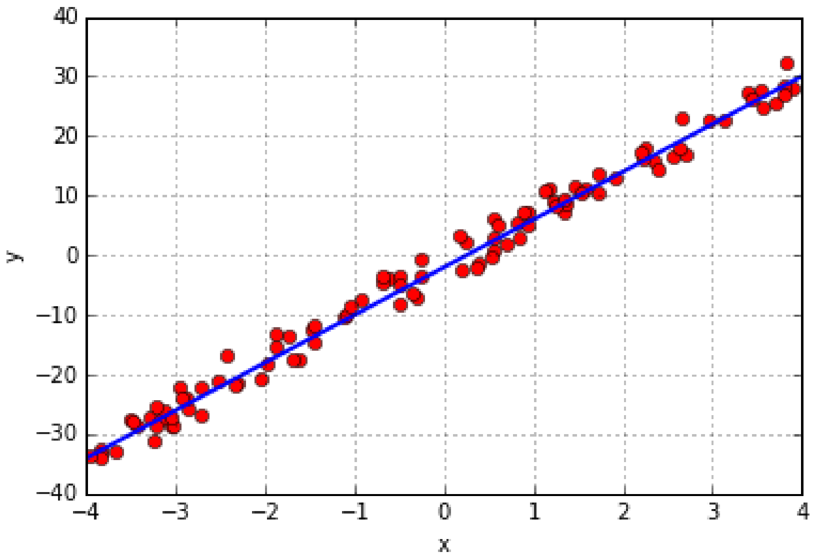
In [88]:
```
import numpy as np
import csv
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithS
GD
def data_generate(fileName, w=[0,0], size=100, seed=0):
    np.random.seed(0)
    x = np.random.uniform(-4, 4, size)
    noise = np.random.normal(0, 2, size)
    y = (x * w[0] + w[1] + noise)
    data = zip(y, x)
    with open(fileName,'wb') as f:
        writer = csv.writer(f)
        for row in data:
            writer.writerow(row)
    return True
```

Data Visualization

In [81]:
```
%matplotlib inline
import matplotlib.pyplot as plt
def dataPlot(file, w):
    with open(file, 'r') as f:
        reader = csv.reader(f)
        for row in reader:
            plt.plot(float(row[1]), float(row[0]),'o'+'r')
    plt.xlabel("x")
    plt.ylabel("y")
    x = [-4, 4]
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x,y, linewidth=2.0)
    plt.grid()
    plt.show()
```

In [84]:
```
w = [8,-2]
data_generate('train_data.csv', w, 100, 200)
dataPlot('train_data.csv',w)

data_generate('test_data.csv', w, 100, 600)
dataPlot('test_data.csv',w)
```





In [85]:
```
!hdfs dfs -copyFromLocal train_data.csv /user/root/wk10/hw106/
!hdfs dfs -copyFromLocal test_data.csv /user/root/wk10/hw106/
```

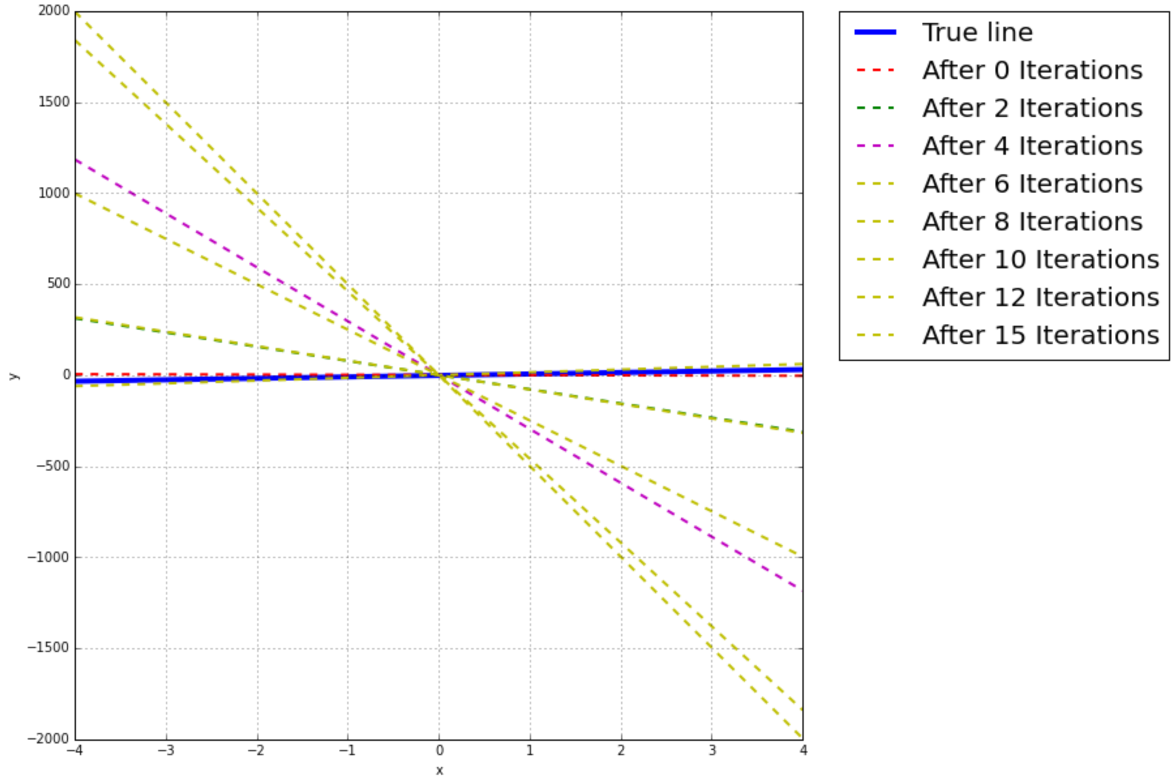
In [103]:

```
def ierationsPlot(train_file, test_file, w):
    x = [-4, 4]

    y = [(i * w[0] + w[1]) for i in x]

    plt.figure(figsize=(10,10))
    plt.plot(x, y, 'b', label="True line", linewidth=4.0)

    #data = sc.textFile(fileName).map(lambda line: [float(v) for v in li
ne.split(',')]+[1.0]).cache()
    #n = data.count()
    train_data = sc.textFile(train_file). \
        map(lambda line: [float(v) for v in line.split(',')]). \
        map(lambda point: LabeledPoint(point[0], [point[1]])).cache()

    test_data = sc.textFile(test_file).map(lambda line: [float(v) for v
in line.split(',')]+[1.0]).cache()

    n = train_data.count()

    np.random.seed(400)
    w = np.random.normal(0,1,2)
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x, y, 'r--', label="After 0 Iterations", linewidth=2.0)

    squared_error = test_data.map(lambda d: (d[0] - np.dot(w, d[1:]))**
2).reduce(lambda a, b: a + b)
    print "Mean Squared Error after 0 iterations: " + str(squared_error/
n)

    lr = LinearRegressionWithSGD.train(train_data, iterations=2, initial
Weights=array([1.0]))
    y = [lr.predict([i]) for i in x]
    plt.plot(x, y, 'g--', label="After 2 Iterations", linewidth=2.0)
    squared_error = test_data.map(lambda d: (d[0] - lr.predict([d[1]]))*
*2).reduce(lambda a, b: a + b)
    print "Mean Squared Error after 2 iterations: " + str(squared_error/
n)

    lr = LinearRegressionWithSGD.train(train_data, iterations=4, initial
Weights=array([1.0]))
    y = [lr.predict([i]) for i in x]
    plt.plot(x, y, 'm--', label="After 4 Iterations", linewidth=2.0)
    squared_error = test_data.map(lambda d: (d[0] - lr.predict([d[1]]))*
*2).reduce(lambda a, b: a + b)
    print "Mean Squared Error after 4 iterations: " + str(squared_error/
n)

    lr = LinearRegressionWithSGD.train(train_data, iterations=6, initial
Weights=array([1.0]))
    y = [lr.predict([i]) for i in x]
    plt.plot(x, y, 'y--', label="After 6 Iterations", linewidth=2.0)
    squared_error = test_data.map(lambda d: (d[0] - lr.predict([d[1]]))*
```

```
*2).reduce(lambda a, b: a + b)
    print "Mean Squared Error after 6 iterations: " + str(squared_error/
n)

    lr = LinearRegressionWithSGD.train(train_data, iterations=8, initial
Weights=array([1.0]))
    y = [lr.predict([i]) for i in x]
    plt.plot(x, y, 'y--', label="After 8 Iterations", linewidth=2.0)
    squared_error = test_data.map(lambda d: (d[0] - lr.predict([d[1]]))*
*2).reduce(lambda a, b: a + b)
    print "Mean Squared Error after 8 iterations: " + str(squared_error/
n)

    lr = LinearRegressionWithSGD.train(train_data, iterations=10, initia
lWeights=array([1.0]))
    y = [lr.predict([i]) for i in x]
    plt.plot(x, y, 'y--', label="After 10 Iterations", linewidth=2.0)
    squared_error = test_data.map(lambda d: (d[0] - lr.predict([d[1]]))*
*2).reduce(lambda a, b: a + b)
    print "Mean Squared Error after 10 iterations: " + str(squared_erro
r/n)

    lr = LinearRegressionWithSGD.train(train_data, iterations=12, initia
lWeights=array([1.0]))
    y = [lr.predict([i]) for i in x]
    plt.plot(x, y, 'y--', label="After 12 Iterations", linewidth=2.0)
    squared_error = test_data.map(lambda d: (d[0] - lr.predict([d[1]]))*
*2).reduce(lambda a, b: a + b)
    print "Mean Squared Error after 12 iterations: " + str(squared_erro
r/n)

    lr = LinearRegressionWithSGD.train(train_data, iterations=15, initia
lWeights=array([1.0]))
    y = [lr.predict([i]) for i in x]
    plt.plot(x, y, 'y--', label="After 15 Iterations", linewidth=2.0)
    squared_error = test_data.map(lambda d: (d[0] - lr.predict([d[1]]))*
*2).reduce(lambda a, b: a + b)
    print "Mean Squared Error after 15 iterations: " + str(squared_erro
r/n)

    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, fontsize=20, borderaxesp
ad=0.)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.grid()
    plt.show()
```

In [104]:
```
test_file = 'hdfs://localhost:54310/user/root/wk10/hw106/test_data.csv'
train_file = 'hdfs://localhost:54310/user/root/wk10/hw106/train_data.cs
v'
ierationsPlot(train_file, test_file,w=[8,-2])
```

```
Mean Squared Error after 0 iterations: 464.394955261
Mean Squared Error after 2 iterations: 39743.4673949
Mean Squared Error after 4 iterations: 495972.259201
Mean Squared Error after 6 iterations: 1379283.53188
Mean Squared Error after 8 iterations: 1175230.41634
Mean Squared Error after 10 iterations: 355568.279027
Mean Squared Error after 12 iterations: 41004.4054943
Mean Squared Error after 15 iterations: 281.227699857
```



In [ ]: