Final

Alejandro J. Rojas
ale@ischool.berkeley.edu
W261: Machine Learning at Scale
Week: 15
*Apr 27, 2016, 7:15 PM EST
*

MIDS Machine Learning at Scale End of Term exam, Week 15, Spring, 2016

===Exam Schedule (All times are in California Time)===

4:00 PM - 6:00 PM

Exam location is at:

```
https://www.dropbox.com/s/k9mw3rnr86iktk6/MIDS-MLS-End-of-Term-2016-Spring-2016-04-27-Published.txt?dl=0
```

===Instructions for exam ===

Instructions:

Within 2 hours of receiving my email can you email me with two attachments:
-- ipython notebook that you use for calculations -- your end of term responses using the following format.

ET1:a,c, ET2:d ET3:c .....

And complete exam submission form (http://goo.gl/forms/ggNYfRXz0t (http://goo.gl/forms/ggNYfRXz0t)).

Good luck, Jimi

1: Please acknowledge receipt of exam by sending a quick reply to the instructors 2: Review the submission form first to scope it out (it will take a 5-10 minutes to input your answers and other information into this form) 3: Please keep all your work and responses in ONE (1) notebook only (and submit via the form) 4: Please make sure that the NBViewer link for your Submission notebook works 5: Please do NOT discuss this exam with anyone (including your class mates) until after 8AM (West coast time) Friday, April 29, 2016 6: This is a take home exam. Please complete by yourself without assistance from others.

Please use your live session time from week 8 to complete this mid term (plus an additional 30 minutes if you need it). This is an open book exam meaning you can consult webpages and textbooks (but not each other or other people). Please complete this exam by yourself.

Please submit your solutions and notebook via the following form:

```
http://goo.gl/forms/ggNYfRXz0t
```

=====Exam questions begins here=====

===== ET:1 Using one-hot-encoding, a categorical feature with four distinct values would be represented by how many features?

(a) 1 feature (b) 2 features (c) 4 features (d) none of the above

===== ET:2 In the following (and also referring to HW12: http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/1wb2rdqbet54y1h/MIDS-MLS-Project-Criteo-CTR.ipynb (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/1wb2rdqbet54y1h/MIDS-MLS-Project-Criteo-CTR.ipynb)) we have hashed the three sample points using numBuckets=4 and numBuckets=100. Complete the three statements below about these hashed features summarized in the following table using each answer once.

Name Raw Features 4 Buckets 100 Buckets sampleOne [(0, 'mouse'), (1, 'black')] {2: 1.0, 3: 1.0} {14: 1.0, 31: 1.0} sampleTwo [(0, 'cat'), (1, 'tabby'), (2, 'mouse')] {0: 2.0, 2: 1.0} {40: 1.0, 16: 1.0, 62: 1.0} sampleThree [(0, 'bear'), (1, 'black'), (2, 'salmon') {0: 1.0, 1: 1.0, 2: 1.0} {72: 1.0, 5: 1.0, 14: 1.0}

With 100 buckets, sampleOne and sampleThree both contain index 14 due to __.

(a) A hash collision (b) Underlying properties of the data (c) The fact that used 100 buckets (d) none of the above

===== ET:3 In the following (and also referring to HW12: http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/1wb2rdqbet54y1h/MIDS-MLS-Project-Criteo-CTR.ipynb (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/1wb2rdqbet54y1h/MIDS-MLS-Project-Criteo-CTR.ipynb)) we have hashed the three sample points using numBuckets=4 and numBuckets=100. Complete the three statements below about these hashed features summarized in the following table using each answer once.

Name Raw Features 4 Buckets 100 Buckets sampleOne [(0, 'mouse'), (1, 'black')] {2: 1.0, 3: 1.0} {14: 1.0, 31: 1.0} sampleTwo [(0, 'cat'), (1, 'tabby'), (2, 'mouse')] {0: 2.0, 2: 1.0} {40: 1.0, 16: 1.0, 62: 1.0} sampleThree [(0, 'bear'), (1, 'black'), (2, 'salmon') {0: 1.0, 1: 1.0, 2: 1.0} {72: 1.0, 5: 1.0, 14: 1.0}

It is likely that sampleTwo has two indices with 4 buckets, but three indices with 100 buckets due to __.

(a) A hash collision (b) Underlying properties of the data (c) The fact that we go from 4 to 100 buckets (d) none of the above

===== ET:4 In the following (and also referring to HW12: http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/1wb2rdqbet54y1h/MIDS-MLS-Project-Criteo-CTR.ipynb (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/1wb2rdqbet54y1h/MIDS-MLS-Project-Criteo-CTR.ipynb)) we have hashed the three sample points using numBuckets=4 and numBuckets=100. Complete the three statements below about these hashed features summarized in the following table using each answer once.

Name Raw Features 4 Buckets 100 Buckets sampleOne [(0, 'mouse'), (1, 'black')] {2: 1.0, 3: 1.0} {14: 1.0, 31: 1.0} sampleTwo [(0, 'cat'), (1, 'tabby'), (2, 'mouse')] {0: 2.0, 2: 1.0} {40: 1.0, 16: 1.0, 62: 1.0} sampleThree [(0, 'bear'), (1, 'black'), (2, 'salmon') {0: 1.0, 1: 1.0, 2: 1.0} {72: 1.0, 5: 1.0, 14: 1.0}

It is likely that sampleTwo has two indices with 4 buckets, but three indices with 100 buckets due to __.

(a) A hash collision (b) Underlying properties of the data (c) The fact that we use 4 buckets (d) none of the above

===== ET:5 When applying numerical machine learning approaches (and for non-numerical approaches if required) to big data problems which of the following steps are could be used during modeling and are recommended:

(a) Convert categorical features to numerical features via one-hot-encoding and store in a dense representation (b) Transform categorical features using hashing regardless of how many unique categorical values exist in training and test data (c) Use matrix factorization to remap your input vectors to latent concepts (d) none of the above

===== ET:6 When dealing with numercial data which of the following are ways to deal with missing data:

(a) Delete records that have missing input values (b) Standardize the data and set all missing values to 1 (one) (c) Use K-nearest neighbours based on the test set to fill in missing values in the training set (d) none of the above

===== ET:7 In the Criteo project, we're trying to predict what:

(a) Revenue from click events (b) Click-through vs not click event (c) Probability of a click event (d) none of the above

===== ET:8 Which of the following are true about the purpose of a loss function?

(a) It's a way to penalize a model for incorrect predictions (b) It precisely defines the optimization problem to be solved for a particular learning model (c) Loss functions can be used for modeling both classification and regression problems (d) none of the above

===== ET:9: When implementing Logistic Regression with Regularization in Spark which of the following apply

(a) When lambda equals one, it provides the same result as standard logistic regression
(b) One only needs to modify the standard logistic regression by modifying the Mapper
(c) Can be framed as minimizing a convex function (d) none of the above

===== ET:10 In the context of ecommerce you have just deployed a new conversion rate prediction model to production. This model (aka treatment model) will challenge the control nodel (i.e., the current model) in AB Test manner to see if it can be produce better revenue. Here is the data that was taken from this live AB Test.

CONTROL MODEL (our new CTR model) Impression ID Revenue
1 0.5020.50 3 3.00. . . . . 200003.00 20001 3.0033.00 3 3.00. . . . . 50, 0013.00 ..... 100,000 $4.00

All other impressions in this 100,000 sample resulted in zero transactions and therefore zero revenue.

TREATMENT MODEL (our new CTR model) Impression ID Revenue
1 1.5020.50 3 0.00. . . . . 50, 0013.00 ..... 100,000 $4.00

All other impressions in this 100,000 sample resulted in zero transactions and therefore zero revenue.

P-values are a common way to determine the statistical significance of a test. The smaller it is, the more confident you can be that the test results are due to something other than random chance. A common p-value of .05 is a 5% significance level. Similarly, a p-value of .01 is a 1% significance level. A p-value of .20 is a 20% significance level. For this problem set the p-value to 0.01

Which of the following are true:

(a) Based on revenue there is no statistical significant difference between the Control and the Treatment at p-value of 0.05 for a one-sided t-test (b) Based on transaction rates (tranactions that generated revenue versus not) there is no statistical significant difference between the Control and the Treatment at p-value of 0.05 for a one-sided t-test

(c) AB testing using differences in revenue for this problem is a useful means of determining if the Treatment conversion rate prediction model is better than the control model. (d) none of the above

===== ET:11 Given this graph expressed in the form of an adjacency list,

Node adjacentNode:weightAssociatedWithEdge N1 N6:10, N2:2 N2 N3:1 N3 N4:1 N4 N5:1 N5 N6:1 N6 N7:1 N7 N8:1 N8 N9:1

Using the parallel breadth-first search algorithm for determining the shortest path from a single source, how many iterations are required to discover the shortest distances to all nodes from Node 1

A 7
B 8
C 13 D None of the above

===== ET:12 When parallelizing support vector machines and related algorithms in map-reduce frameworks, which of the following statements are true:

(a) In the context of support vector machines, nonlinear kernels such as quadratic kernels can be readily parallelized in map reduce frameworks such as Spark (b) In the context of support vector machines, linear kernels can be readily parallelized in map reduce frameworks such as Spark (c) Sequential learning via algorithms such perceptron can take advantage of map-reduce frameworks and yield the same results as a single core implementation with significant reductions in training time (d) Field-aware Factorization Machines can easily distributed to take advantage of map-reduce frameworks

===== ET:13 Given the following paired RDDs RDD1 = {(1, 2), (3, 4), (3, 6)} RDD2 = {(3, 9) (3, 6)}

Using PySpark, write code to perform an inner join of these paired RDDs. What is the resulting RDD? Make your Spark available in your notebook:

A: [(3, (4, 9)), (3, (6, 9))] B: [(3, (4, 9)), (3, (4, 6)), (3, (6, 9)), (3, (6, 6))] C: [(3, (4, 9)), (3, (4, 6)), (3, (6, 9)), (3, (6, 9))] D: None of the above

===== ET:14 You have been tasked to build a predictive model to forecast beer sales for a chain of stores. After doing basic exploratory analysis on the data, what is the first thing you do regarding modeling?

(a) Construct a baseline model (b) Determine a metric to evaluate your machine learnt models (c) Split your data into training, validation and test subsets (or split using cross fold validatation) (d) All of the of the above

===== ET:15 Use Spark and the following notebook, https://www.dropbox.com/s/6s5ph41h74bggwi/Linear-Regression-on-Beer-Data.ipynb?dl=0 (https://www.dropbox.com/s/6s5ph41h74bggwi/Linear-Regression-on-Beer-Data.ipynb?dl=0) to answer this question.

The mean absolute percentage error (MAPE), also known as mean absolute percentage deviation (MAPD), is a measure of prediction accuracy of a model for say a forecasting method in statistics, for example in trend estimation. It usually expresses accuracy as a percentage, and is defined by the formula:

MAPE = average over all examples (100*Abs(Actual - Predicted) / Actual))

Note when Actual is zero that test row is dropped from the evaluation.

Construct a mean model for target variable CASES18PK. Calculate the MAPE for the mean model over the training set. Select the closest answer.

(a) 200% (b) 250% (c) 20% (d) 180%

===== ET:16 Use Spark and the following notebook, https://www.dropbox.com/s/6s5ph41h74bggwi/Linear-Regression-on-Beer-Data.ipynb?dl=0 (https://www.dropbox.com/s/6s5ph41h74bggwi/Linear-Regression-on-Beer-Data.ipynb?dl=0) to answer this question.

The target variable CASES18PK is skewed, so take the log of it (and make it more normally distributed) and compute the MAPE of the mean model for CASES18PK Select the closest answer to your calculated MAPE.

(a) 200% (b) 30% (c) 20% (d) 10%

===== ET:17 Use Spark and the following notebook, https://www.dropbox.com/s/6s5ph41h74bggwi/Linear-Regression-on-Beer-Data.ipynb?dl=0 (https://www.dropbox.com/s/6s5ph41h74bggwi/Linear-Regression-on-Beer-Data.ipynb?dl=0) to answer this question.

Build a linear regression model using the following variables:

Log(CASES18PK) ~ log(PRICE12PK), log(PRICE18PK), log(PRICE30PK)

Calculate MAPE over the test set and select the closest answer.

(a) 4.3% (b) 4.6% (c) 3.5% (d) 3.9%

===== ET:18 Recall that Spark automatically sends all variables referenced in your closures to the worker nodes. While this is convenient, it can also be inefficient because (1) the default task launching mechanism is optimized for small task sizes, and (2) you might, in fact, use the same variable in multiple parallel operations, but Spark will send it separately for each operation. As an example, say that we wanted to write a Spark program that looks up countries by their call signs (e.g., the call sign for Ireland is EJZ) by prefix matching in an table. In the following the "signPrefixes" variable is essentially a table with two columns "Sign" and "Country Name". The goal is to join the following tables:

```
signPrefixes table with columns "Sign" and "Country Name"
contactCounts table with columns "Sign" and "count"

to yield  a new table:
```

countryContactCounts with the following columns "Country Name" and "count"

Use Spark and the following notebook, https://www.dropbox.com/s/6s5ph41h74bggwi/Linear-Regression-on-Beer-Data.ipynb?dl=0 (https://www.dropbox.com/s/6s5ph41h74bggwi/Linear-Regression-on-Beer-Data.ipynb?dl=0) to answer this question.

..... Other code...

Country lookup code

Helper functions for looking up the call signs

def lookupCountry(sign, prefixes): pos = bisect.bisect_left(prefixes, sign) return prefixes[pos].split(",")[1]

def loadCallSignTable(): f = open("callsign_tbl_sorted.txt", "r") return f.readlines()

Lookup the locations of the call signs on the RDD contactCounts. We load a list of call sign prefixes to country code to support this lookup.

signPrefixes = loadCallSignTable()

def processSignCount(sign_count, signPrefixes): country = lookupCountry(sign_count[0], signPrefixes) count = sign_count[1] return (country, count)

countryContactCounts = (contactCounts .map(lambda signCount: processSignCount(signCount, signPrefixes)) .reduceByKey((lambda x, y: x + y)))

countryContactCounts.saveAsTextFile(outputDir + "/countries.txt")

How can we modfify this code to make it more efficient? Choose one response only

(a) modify line 18 with sc.broadcast(loadCallSignTable()) (b) Use accumulators to store the counts for each country (c) The code is already optimal (d) none of the above is correct

In [1]:
```
import os
import sys
#Escape L for line numbers
spark_home = os.environ['SPARK_HOME'] = '/Users/Lissette/Documents/
datascience/software/spark-1.5.2-bin-hadoop2.6'
##spark_home = os.environ['SPARK_HOME'] = '/Users/jshanahan/Dropbo
x/Lectures-UC-Berkeley-ML-Class-2015/spark-1.5.0-bin-hadoop2.6/'
#spark_home = os.environ['SPARK_HOME'] = '/Users/james.shanahan/Dro
pbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.5.0-bin-hadoop2.6/'
#spark_home = os.environ['SPARK_HOME'] = '/Users/liang/Downloads/sp
ark-1.3.0-bin-hadoop2.4/'
if not spark_home:
    raise ValueError('SPARK_HOME enviroment variable is not set')
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-
src.zip'))
execfile(os.path.join(spark_home,'python/pyspark/shell.py'))
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.5.2
      /_/

Using Python version 2.7.11 (default, Dec  6 2015 18:57:58)
SparkContext available as sc, HiveContext available as sqlContext.
```

In [2]:
```
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).argmi
n()
    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 =
'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 [7]:
```
RDD1 = {(1, 2), (3, 4), (3, 6)}
```

In [9]:
```
RDD2 = {(3, 9),(3, 6)}
```

In [13]:
```
RDD1.join(RDD2).map(k)
```

```
---------------------------------------------------------------------
---------
AttributeError                            Traceback (most recent c
all last)
<ipython-input-13-14839d2c64cc> in <module>()
----> 1 RDD1.join(RDD2)

AttributeError: 'set' object has no attribute 'join'
```

ET15

In [ ]:

In [14]:

```
%%writefile callsign_tbl_sorted.txt
3AZ, Monaco (Principality of)
3BZ, Mauritius (Republic of)
3CZ, Equatorial Guinea (Republic of)
3DM, Swaziland (Kingdom of)
3DZ, Fiji (Republic of)
3FZ, Panama (Republic of)
3GZ, Chile
3UZ, China (People's Republic of)
3VZ, Tunisia
3WZ, Viet Nam (Socialist Republic of)
3XZ, Guinea (Republic of)
3YZ, Norway
3ZZ, Poland (Republic of)
4CZ, Mexico
4IZ, Philippines (Republic of the)
4KZ, Azerbaijani Republic
4LZ, Georgia (Republic of)
4MZ, Venezuela (Republic of)
4OZ, Montenegro (Republic of)                                (WRC-07)
4SZ, Sri Lanka (Democratic Socialist Republic of)
4TZ, Peru
4UZ, United Nations
4VZ, Haiti (Republic of)
4WZ, Democratic Republic of Timor-Leste   (WRC-03)
4XZ, Israel (State of)
4YZ, International Civil Aviation Organization
4ZZ, Israel (State of)
5AZ, Libya (Socialist People's Libyan Arab Jamahiriya)
5BZ, Cyprus (Republic of)
5GZ, Morocco (Kingdom of)
5IZ, Tanzania (United Republic of)
5KZ, Colombia (Republic of)
5MZ, Liberia (Republic of)
5OZ, Nigeria (Federal Republic of)
5QZ, Denmark
5SZ, Madagascar (Republic of)
5TZ, Mauritania (Islamic Republic of)
5UZ, Niger (Republic of the)
5VZ, Togolese Republic
5WZ, Samoa (Independent State of)
5XZ, Uganda (Republic of)
5ZZ, Kenya (Republic of)
6BZ, Egypt (Arab Republic of)
6CZ, Syrian Arab Republic
6JZ, Mexico
6NZ, Korea (Republic of)
6OZ, Somali Democratic Republic
6SZ, Pakistan (Islamic Republic of)
6UZ, Sudan (Republic of the)
6WZ, Senegal (Republic of)
6XZ, Madagascar (Republic of)
```

```
6YZ, Jamaica
6ZZ, Liberia (Republic of)
7IZ, Indonesia (Republic of)
7NZ, Japan
7OZ, Yemen (Republic of)
7PZ, Lesotho (Kingdom of)
7QZ, Malawi
7RZ, Algeria (People's Democratic Republic of)
7SZ, Sweden
7YZ, Algeria (People's Democratic Republic of)
7ZZ, Saudi Arabia (Kingdom of)
8IZ, Indonesia (Republic of)
8NZ, Japan
8OZ, Botswana (Republic of)
8PZ, Barbados
8QZ, Maldives (Republic of)
8RZ, Guyana
8SZ, Sweden
8YZ, India (Republic of)
8ZZ, Saudi Arabia (Kingdom of)
9AZ, Croatia (Republic of)
9DZ, Iran (Islamic Republic of)
9FZ, Ethiopia (Federal Democratic Republic of)
9GZ, Ghana
9HZ, Malta
9JZ, Zambia (Republic of)
9KZ, Kuwait (State of)
9LZ, Sierra Leone
9MZ, Malaysia
9NZ, Nepal
9TZ, Democratic Republic of the Congo
9UZ, Burundi (Republic of)
9VZ, Singapore (Republic of)
9WZ, Malaysia
9XZ, Rwandese Republic
9ZZ, Trinidad and Tobago
A2Z, Botswana (Republic of)
A3Z, Tonga (Kingdom of)
A4Z, Oman (Sultanate of)
A5Z, Bhutan (Kingdom of)
A6Z, United Arab Emirates
A7Z, Qatar (State of)
A8Z, Liberia (Republic of)
A9Z, Bahrain (State of)
ALZ, United States of America
AOZ, Spain
ASZ, Pakistan (Islamic Republic of)
AWZ, India (Republic of)
AXZ, Australia
AZZ, Argentine Republic
BZZ, China (People's Republic of)
C2Z, Nauru (Republic of)
```

```
C3Z, Andorra (Principality of)
C4Z, Cyprus (Republic of)
C5Z, Gambia (Republic of the)
C6Z, Bahamas (Commonwealth of the)
C7Z, World Meteorological Organization
C9Z, Mozambique (Republic of)
CEZ, Chile
CKZ, Canada
CMZ, Cuba
CNZ, Morocco (Kingdom of)
COZ, Cuba
CPZ, Bolivia (Republic of)
CUZ, Portugal
CXZ, Uruguay (Eastern Republic of)
CZZ, Canada
D3Z, Angola (Republic of)
D4Z, Cape Verde (Republic of)
D5Z, Liberia (Republic of)
D6Z, Comoros (Islamic Federal Republic of the)
D9Z, Korea (Republic of)
DRZ, Germany (Federal Republic of)
DTZ, Korea (Republic of)
DZZ, Philippines (Republic of the)
E2Z, Thailand
E3Z, Eritrea
E4Z, Palestinian Authority
E5Z, New Zealand - Cook Islands                          (WRC-07)
E6Z, New Zealand - Niue
E7Z, Bosnia and Herzegovina (Republic of)                (WRC-07)
EHZ, Spain
EJZ, Ireland
EKZ, Armenia (Republic of)
ELZ, Liberia (Republic of)
EOZ, Ukraine
EQZ, Iran (Islamic Republic of)
ERZ, Moldova (Republic of)
ESZ, Estonia (Republic of)
ETZ, Ethiopia (Federal Democratic Republic of)
EWZ, Belarus (Republic of)
EXZ, Kyrgyz Republic
EYZ, Tajikistan (Republic of)
EZZ, Turkmenistan
FZZ, France
GZZ, United Kingdom of Great Britain and Northern Ireland
H2Z, Cyprus (Republic of)
H3Z, Panama (Republic of)
H4Z, Solomon Islands
H7Z, Nicaragua
H9Z, Panama (Republic of)
HAZ, Hungary (Republic of)
HBZ, Switzerland (Confederation of)
HDZ, Ecuador
```

```
HEZ, Switzerland (Confederation of)
HFZ, Poland (Republic of)
HGZ, Hungary (Republic of)
HHZ, Haiti (Republic of)
HIZ, Dominican Republic
HKZ, Colombia (Republic of)
HLZ, Korea (Republic of)
HMZ, Democratic People's Republic of Korea
HNZ, Iraq (Republic of)
HPZ, Panama (Republic of)
HRZ, Honduras (Republic of)
HSZ, Thailand
HTZ, Nicaragua
HUZ, El Salvador (Republic of)
HVZ, Vatican City State
HYZ, France
HZZ, Saudi Arabia (Kingdom of)
IZZ, Italy
J2Z, Djibouti (Republic of)
J3Z, Grenada
J4Z, Greece
J5Z, Guinea-Bissau (Republic of)
J6Z, Saint Lucia
J7Z, Dominica (Commonwealth of)
J8Z, Saint Vincent and the Grenadines
JSZ, Japan
JVZ, Mongolia
JXZ, Norway
JYZ, Jordan (Hashemite Kingdom of)
JZZ, Indonesia (Republic of)
KZZ, United States of America
L9Z, Argentine Republic
LNZ, Norway
LWZ, Argentine Republic
LXZ, Luxembourg
LYZ, Lithuania (Republic of)
LZZ, Bulgaria (Republic of)
MZZ, United Kingdom of Great Britain and Northern Ireland
NZZ, United States of America
OCZ, Peru
ODZ, Lebanon
OEZ, Austria
OJZ, Finland
OLZ, Czech Republic
OMZ, Slovak Republic
OTZ, Belgium
OZZ, Denmark
P2Z, Papua New Guinea
P3Z, Cyprus (Republic of)
P4Z, Netherlands (Kingdom of the) - Aruba
P9Z, Democratic People's Republic of Korea
PIZ, Netherlands (Kingdom of the)
```

```
PJZ, Netherlands (Kingdom of the) - Netherlands Caribbean
POZ, Indonesia (Republic of)
PYZ, Brazil (Federative Republic of)
PZZ, Suriname (Republic of)
RZZ, Russian Federation
S3Z, Bangladesh (People's Republic of)
S5Z, Slovenia (Republic of)
S6Z, Singapore (Republic of)
S7Z, Seychelles (Republic of)
S8Z, South Africa (Republic of)
S9Z, Sao Tome and Principe (Democratic Republic of)
SMZ, Sweden
SRZ, Poland (Republic of)
SSM, Egypt (Arab Republic of)
STZ, Sudan (Republic of the)
SUZ, Egypt (Arab Republic of)
SZZ, Greece
T2Z, Tuvalu
T3Z, Kiribati (Republic of)
T4Z, Cuba
T5Z, Somali Democratic Republic
T6Z, Afghanistan (Islamic State of)
T7Z, San Marino (Republic of)
T8Z, Palau (Republic of)
TCZ, Turkey
TDZ, Guatemala (Republic of)
TEZ, Costa Rica
TFZ, Iceland
TGZ, Guatemala (Republic of)
THZ, France
TIZ, Costa Rica
TJZ, Cameroon (Republic of)
TKZ, France
TLZ, Central African Republic
TMZ, France
TNZ, Congo (Republic of the)
TQZ, France
TRZ, Gabonese Republic
TSZ, Tunisia
TTZ, Chad (Republic of)
TUZ, Côte d'Ivoire (Republic of)
TXZ, France
TYZ, Benin (Republic of)
TZZ, Mali (Republic of)
UIZ, Russian Federation
UMZ, Uzbekistan (Republic of)
UQZ, Kazakhstan (Republic of)
UZZ, Ukraine
V2Z, Antigua and Barbuda
V3Z, Belize
V4Z, Saint Kitts and Nevis
V5Z, Namibia (Republic of)
```

```
V6Z, Micronesia (Federated States of)
V7Z, Marshall Islands (Republic of the)
V8Z, Brunei Darussalam
VGZ, Canada
VNZ, Australia
VOZ, Canada
VQZ, United Kingdom of Great Britain and Northern Ireland
VRZ, China (People's Republic of) - Hong Kong
VSZ, United Kingdom of Great Britain and Northern Ireland
VWZ, India (Republic of)
VYZ, Canada
VZZ, Australia
WZZ, United States of America
XIZ, Mexico
XOZ, Canada
XPZ, Denmark
XRZ, Chile
XSZ, China (People's Republic of)
XTZ, Burkina Faso
XUZ, Cambodia (Kingdom of)
XVZ, Viet Nam (Socialist Republic of)
XWZ, Lao People's Democratic Republic
XXZ, China (People's Republic of) - Macao                (WRC-07)
XZZ, Myanmar (Union of)
Y9Z, Germany (Federal Republic of)
YAZ, Afghanistan (Islamic State of)
YHZ, Indonesia (Republic of)
YIZ, Iraq (Republic of)
YJZ, Vanuatu (Republic of)
YKZ, Syrian Arab Republic
YLZ, Latvia (Republic of)
YMZ, Turkey
YNZ, Nicaragua
YRZ, Romania
YSZ, El Salvador (Republic of)
YUZ, Serbia (Republic of)                                    (WR
C-07)
YYZ, Venezuela (Republic of)
Z2Z, Zimbabwe (Republic of)
Z3Z, The Former Yugoslav Republic of Macedonia
Z8Z, South Sudan (Republic of)
ZAZ, Albania (Republic of)
ZJZ, United Kingdom of Great Britain and Northern Ireland
ZMZ, New Zealand
ZOZ, United Kingdom of Great Britain and Northern Ireland
ZPZ, Paraguay (Republic of)
ZQZ, United Kingdom of Great Britain and Northern Ireland
ZUZ, South Africa (Republic of)
ZZZ, Brazil (Federative Republic of)
```

```
Writing callsign_tbl_sorted.txt
```

In [15]:
```
#..... Other code...
#Country lookup code

# Helper functions for looking up the call signs

def lookupCountry(sign, prefixes):
    pos = bisect.bisect_left(prefixes, sign)
    return prefixes[pos].split(",")[1]


def loadCallSignTable():
    f = open("callsign_tbl_sorted.txt", "r")
    return f.readlines()

# Lookup the locations of the call signs on the
# RDD contactCounts. We load a list of call sign
# prefixes to country code to support this lookup.
signPrefixes = loadCallSignTable()


def processSignCount(sign_count, signPrefixes):
    country = lookupCountry(sign_count[0], signPrefixes)
    count = sign_count[1]
    return (country, count)

contactCounts = sc.parallelize([["ZMZ", 1], ["ZMZ", 3]])

countryContactCounts = (contactCounts
                        .map(lambda signCount: processSignCount(sig
nCount, signPrefixes))
                        .reduceByKey((lambda x, y: x + y)))

#countryContactCounts.saveAsTextFile("tmp/countries.txt")
```

In [16]:

```
%%writefile beerSales.txt
Week    PRICE12PK       PRICE18PK       PRICE30PK       CASES12PK
CASES18PK       CASES30PK
1       19.98   14.10   15.19   223.5   439     55.00
2       19.98   18.65   15.19   215.0   98      66.75
3       19.98   18.65   13.87   227.5   70      242.00
4       19.98   18.65   12.83   244.5   52      488.50
5       19.98   18.65   13.16   313.5   64      308.75
6       19.98   18.65   15.19   279.0   72      111.75
7       19.98   18.65   13.92   238.0   47      252.50
8       20.10   18.73   14.42   315.5   85      221.25
9       20.12   18.75   13.83   217.0   59      245.25
10      20.13   18.75   14.50   209.5   63      148.50
11      20.14   18.75   13.87   227.0   57      229.75
12      20.12   18.75   13.64   216.5   54      312.00
13      20.12   13.87   14.31   169.0   404     96.75
14      20.13   14.27   13.85   178.0   380     123.25
15      20.14   18.76   14.20   301.5   65      200.50
16      20.14   18.77   13.64   266.5   40      359.75
17      20.13   13.87   14.33   182.5   456     113.50
18      20.13   14.14   13.14   159.0   176     136.50
19      20.13   18.76   13.81   285.5   61      225.50
20      20.13   18.72   15.19   360.0   91      122.25
21      20.13   18.76   13.13   263.0   59      443.75
22      19.18   18.76   13.63   443.5   83      322.75
23      14.78   18.74   15.19   1101.5  41      53.00
24      16.04   18.75   13.89   814.0   47      140.75
25      20.12   18.75   14.28   365.0   84      210.75
26      19.75   18.75   15.19   510.0   85      110.50
27      19.65   18.75   13.12   580.5   116     568.25
28      19.69   13.79   13.78   251.0   544     115.50
29      20.12   13.49   15.19   237.0   890     58.75
30      20.12   14.89   15.19   302.5   371     77.25
31      20.13   13.94   15.19   229.5   557     66.25
32      20.14   13.67   15.19   188.5   775     50.00
33      15.14   14.43   15.19   795.5   236     46.50
34      14.33   18.75   15.19   1556.5  43      65.75
35      16.24   18.22   13.14   807.5   63      252.75
36      19.93   14.06   13.45   243.0   469     179.00
37      21.06   14.43   13.00   201.5   335     226.25
38      21.19   19.48   13.60   294.0   75      288.50
39      21.23   15.15   14.46   220.5   461     114.25
40      20.12   13.79   14.94   255.5   817     70.00
41      14.73   14.31   15.19   920.5   200     47.75
42      14.57   19.50   15.19   730.0   32      98.75
43      15.94   13.85   15.19   262.5   460     77.00
44      20.70   14.23   13.43   209.5   751     160.50
45      19.57   19.31   14.37   283.0   70      143.50
46      19.60   19.29   15.19   262.5   80      133.00
47      19.94   13.76   15.19   310.0   523     68.75
48      21.28   13.45   15.19   278.5   741     81.75
49      14.56   15.13   15.19   741.5   130     56.25
```

```
50      14.39   19.43   15.19   1316.0  69      68.75
51      16.81   13.26   15.19   449.0   493     49.25
52      19.86   13.92   15.19   505.0   814     76.50
```

```
Writing beerSales.txt
```

In [17]:
```
df=sc.textFile("beerSales.txt")
```

In [18]:
```
df.take(1)
```

Out[18]:
```
[u'Week\tPRICE12PK\tPRICE18PK\tPRICE30PK\tCASES12PK\tCASES18PK\tCA
SES30PK']
```

In [19]:
```
df.take(2)
```

Out[19]:
```
[u'Week\tPRICE12PK\tPRICE18PK\tPRICE30PK\tCASES12PK\tCASES18PK\tCA
SES30PK',
 u'1\t19.98\t14.10\t15.19\t223.5\t439\t55.00']
```

In [32]:
```
data = df.collect()
```

In [33]:
```
data = [data[i].split('\t') for i in range(len(data))]
```

In [35]:
```
data
```

Out[35]:

```
[[u'Week',
  u'PRICE12PK',
  u'PRICE18PK',
  u'PRICE30PK',
  u'CASES12PK',
  u'CASES18PK',
  u'CASES30PK'],
 [u'1', u'19.98', u'14.10', u'15.19', u'223.5', u'439', u'55.00'],
 [u'2', u'19.98', u'18.65', u'15.19', u'215.0', u'98', u'66.75'],
 [u'3', u'19.98', u'18.65', u'13.87', u'227.5', u'70', u'242.00'],
 [u'4', u'19.98', u'18.65', u'12.83', u'244.5', u'52', u'488.50'],
 [u'5', u'19.98', u'18.65', u'13.16', u'313.5', u'64', u'308.75'],
 [u'6', u'19.98', u'18.65', u'15.19', u'279.0', u'72', u'111.75'],
 [u'7', u'19.98', u'18.65', u'13.92', u'238.0', u'47', u'252.50'],
 [u'8', u'20.10', u'18.73', u'14.42', u'315.5', u'85', u'221.25'],
 [u'9', u'20.12', u'18.75', u'13.83', u'217.0', u'59', u'245.25'],
 [u'10', u'20.13', u'18.75', u'14.50', u'209.5', u'63', u'148.5
0'],
 [u'11', u'20.14', u'18.75', u'13.87', u'227.0', u'57', u'229.7
5'],
 [u'12', u'20.12', u'18.75', u'13.64', u'216.5', u'54', u'312.0
0'],
 [u'13', u'20.12', u'13.87', u'14.31', u'169.0', u'404', u'96.7
5'],
 [u'14', u'20.13', u'14.27', u'13.85', u'178.0', u'380', u'123.2
5'],
 [u'15', u'20.14', u'18.76', u'14.20', u'301.5', u'65', u'200.5
0'],
 [u'16', u'20.14', u'18.77', u'13.64', u'266.5', u'40', u'359.7
5'],
 [u'17', u'20.13', u'13.87', u'14.33', u'182.5', u'456', u'113.5
0'],
 [u'18', u'20.13', u'14.14', u'13.14', u'159.0', u'176', u'136.5
0'],
 [u'19', u'20.13', u'18.76', u'13.81', u'285.5', u'61', u'225.5
0'],
 [u'20', u'20.13', u'18.72', u'15.19', u'360.0', u'91', u'122.2
5'],
 [u'21', u'20.13', u'18.76', u'13.13', u'263.0', u'59', u'443.7
5'],
 [u'22', u'19.18', u'18.76', u'13.63', u'443.5', u'83', u'322.7
5'],
 [u'23', u'14.78', u'18.74', u'15.19', u'1101.5', u'41', u'53.0
0'],
 [u'24', u'16.04', u'18.75', u'13.89', u'814.0', u'47', u'140.7
5'],
 [u'25', u'20.12', u'18.75', u'14.28', u'365.0', u'84', u'210.7
5'],
 [u'26', u'19.75', u'18.75', u'15.19', u'510.0', u'85', u'110.5
0'],
 [u'27', u'19.65', u'18.75', u'13.12', u'580.5', u'116', u'568.2
5'],
```

```
 [u'28', u'19.69', u'13.79', u'13.78', u'251.0', u'544', u'115.5
0'],
 [u'29', u'20.12', u'13.49', u'15.19', u'237.0', u'890', u'58.7
5'],
 [u'30', u'20.12', u'14.89', u'15.19', u'302.5', u'371', u'77.2
5'],
 [u'31', u'20.13', u'13.94', u'15.19', u'229.5', u'557', u'66.2
5'],
 [u'32', u'20.14', u'13.67', u'15.19', u'188.5', u'775', u'50.0
0'],
 [u'33', u'15.14', u'14.43', u'15.19', u'795.5', u'236', u'46.5
0'],
 [u'34', u'14.33', u'18.75', u'15.19', u'1556.5', u'43', u'65.7
5'],
 [u'35', u'16.24', u'18.22', u'13.14', u'807.5', u'63', u'252.7
5'],
 [u'36', u'19.93', u'14.06', u'13.45', u'243.0', u'469', u'179.0
0'],
 [u'37', u'21.06', u'14.43', u'13.00', u'201.5', u'335', u'226.2
5'],
 [u'38', u'21.19', u'19.48', u'13.60', u'294.0', u'75', u'288.5
0'],
 [u'39', u'21.23', u'15.15', u'14.46', u'220.5', u'461', u'114.2
5'],
 [u'40', u'20.12', u'13.79', u'14.94', u'255.5', u'817', u'70.0
0'],
 [u'41', u'14.73', u'14.31', u'15.19', u'920.5', u'200', u'47.7
5'],
 [u'42', u'14.57', u'19.50', u'15.19', u'730.0', u'32', u'98.75'],
 [u'43', u'15.94', u'13.85', u'15.19', u'262.5', u'460', u'77.0
0'],
 [u'44', u'20.70', u'14.23', u'13.43', u'209.5', u'751', u'160.5
0'],
 [u'45', u'19.57', u'19.31', u'14.37', u'283.0', u'70', u'143.5
0'],
 [u'46', u'19.60', u'19.29', u'15.19', u'262.5', u'80', u'133.0
0'],
 [u'47', u'19.94', u'13.76', u'15.19', u'310.0', u'523', u'68.7
5'],
 [u'48', u'21.28', u'13.45', u'15.19', u'278.5', u'741', u'81.7
5'],
 [u'49', u'14.56', u'15.13', u'15.19', u'741.5', u'130', u'56.2
5'],
 [u'50', u'14.39', u'19.43', u'15.19', u'1316.0', u'69', u'68.7
5'],
 [u'51', u'16.81', u'13.26', u'15.19', u'449.0', u'493', u'49.2
5'],
 [u'52', u'19.86', u'13.92', u'15.19', u'505.0', u'814', u'76.5
0']]
```

In [36]:
```
data= data[1:]
```

In [37]:
```
data
```

Out[37]:

```
[[u'1', u'19.98', u'14.10', u'15.19', u'223.5', u'439', u'55.00'],
 [u'2', u'19.98', u'18.65', u'15.19', u'215.0', u'98', u'66.75'],
 [u'3', u'19.98', u'18.65', u'13.87', u'227.5', u'70', u'242.00'],
 [u'4', u'19.98', u'18.65', u'12.83', u'244.5', u'52', u'488.50'],
 [u'5', u'19.98', u'18.65', u'13.16', u'313.5', u'64', u'308.75'],
 [u'6', u'19.98', u'18.65', u'15.19', u'279.0', u'72', u'111.75'],
 [u'7', u'19.98', u'18.65', u'13.92', u'238.0', u'47', u'252.50'],
 [u'8', u'20.10', u'18.73', u'14.42', u'315.5', u'85', u'221.25'],
 [u'9', u'20.12', u'18.75', u'13.83', u'217.0', u'59', u'245.25'],
 [u'10', u'20.13', u'18.75', u'14.50', u'209.5', u'63', u'148.5
0'],
 [u'11', u'20.14', u'18.75', u'13.87', u'227.0', u'57', u'229.7
5'],
 [u'12', u'20.12', u'18.75', u'13.64', u'216.5', u'54', u'312.0
0'],
 [u'13', u'20.12', u'13.87', u'14.31', u'169.0', u'404', u'96.7
5'],
 [u'14', u'20.13', u'14.27', u'13.85', u'178.0', u'380', u'123.2
5'],
 [u'15', u'20.14', u'18.76', u'14.20', u'301.5', u'65', u'200.5
0'],
 [u'16', u'20.14', u'18.77', u'13.64', u'266.5', u'40', u'359.7
5'],
 [u'17', u'20.13', u'13.87', u'14.33', u'182.5', u'456', u'113.5
0'],
 [u'18', u'20.13', u'14.14', u'13.14', u'159.0', u'176', u'136.5
0'],
 [u'19', u'20.13', u'18.76', u'13.81', u'285.5', u'61', u'225.5
0'],
 [u'20', u'20.13', u'18.72', u'15.19', u'360.0', u'91', u'122.2
5'],
 [u'21', u'20.13', u'18.76', u'13.13', u'263.0', u'59', u'443.7
5'],
 [u'22', u'19.18', u'18.76', u'13.63', u'443.5', u'83', u'322.7
5'],
 [u'23', u'14.78', u'18.74', u'15.19', u'1101.5', u'41', u'53.0
0'],
 [u'24', u'16.04', u'18.75', u'13.89', u'814.0', u'47', u'140.7
5'],
 [u'25', u'20.12', u'18.75', u'14.28', u'365.0', u'84', u'210.7
5'],
 [u'26', u'19.75', u'18.75', u'15.19', u'510.0', u'85', u'110.5
0'],
 [u'27', u'19.65', u'18.75', u'13.12', u'580.5', u'116', u'568.2
5'],
 [u'28', u'19.69', u'13.79', u'13.78', u'251.0', u'544', u'115.5
0'],
 [u'29', u'20.12', u'13.49', u'15.19', u'237.0', u'890', u'58.7
5'],
 [u'30', u'20.12', u'14.89', u'15.19', u'302.5', u'371', u'77.2
5'],
 [u'31', u'20.13', u'13.94', u'15.19', u'229.5', u'557', u'66.2
```

```
5'],
 [u'32', u'20.14', u'13.67', u'15.19', u'188.5', u'775', u'50.0
0'],
 [u'33', u'15.14', u'14.43', u'15.19', u'795.5', u'236', u'46.5
0'],
 [u'34', u'14.33', u'18.75', u'15.19', u'1556.5', u'43', u'65.7
5'],
 [u'35', u'16.24', u'18.22', u'13.14', u'807.5', u'63', u'252.7
5'],
 [u'36', u'19.93', u'14.06', u'13.45', u'243.0', u'469', u'179.0
0'],
 [u'37', u'21.06', u'14.43', u'13.00', u'201.5', u'335', u'226.2
5'],
 [u'38', u'21.19', u'19.48', u'13.60', u'294.0', u'75', u'288.5
0'],
 [u'39', u'21.23', u'15.15', u'14.46', u'220.5', u'461', u'114.2
5'],
 [u'40', u'20.12', u'13.79', u'14.94', u'255.5', u'817', u'70.0
0'],
 [u'41', u'14.73', u'14.31', u'15.19', u'920.5', u'200', u'47.7
5'],
 [u'42', u'14.57', u'19.50', u'15.19', u'730.0', u'32', u'98.75'],
 [u'43', u'15.94', u'13.85', u'15.19', u'262.5', u'460', u'77.0
0'],
 [u'44', u'20.70', u'14.23', u'13.43', u'209.5', u'751', u'160.5
0'],
 [u'45', u'19.57', u'19.31', u'14.37', u'283.0', u'70', u'143.5
0'],
 [u'46', u'19.60', u'19.29', u'15.19', u'262.5', u'80', u'133.0
0'],
 [u'47', u'19.94', u'13.76', u'15.19', u'310.0', u'523', u'68.7
5'],
 [u'48', u'21.28', u'13.45', u'15.19', u'278.5', u'741', u'81.7
5'],
 [u'49', u'14.56', u'15.13', u'15.19', u'741.5', u'130', u'56.2
5'],
 [u'50', u'14.39', u'19.43', u'15.19', u'1316.0', u'69', u'68.7
5'],
 [u'51', u'16.81', u'13.26', u'15.19', u'449.0', u'493', u'49.2
5'],
 [u'52', u'19.86', u'13.92', u'15.19', u'505.0', u'814', u'76.5
0']]
```

In [50]:
```
packs=[]
for line in data:
    packs.append(int(line[5]))
```

In [51]:
```
import numpy as np
```

In [54]:
```
mean = np.mean(packs)
```

In [59]:
```
mape = []
for p in packs:
    mape.append(abs(p-mean)/p*100)
```

In [60]:
```
np.mean(mape)
```

Out[60]:
```
200.44422332990541
```

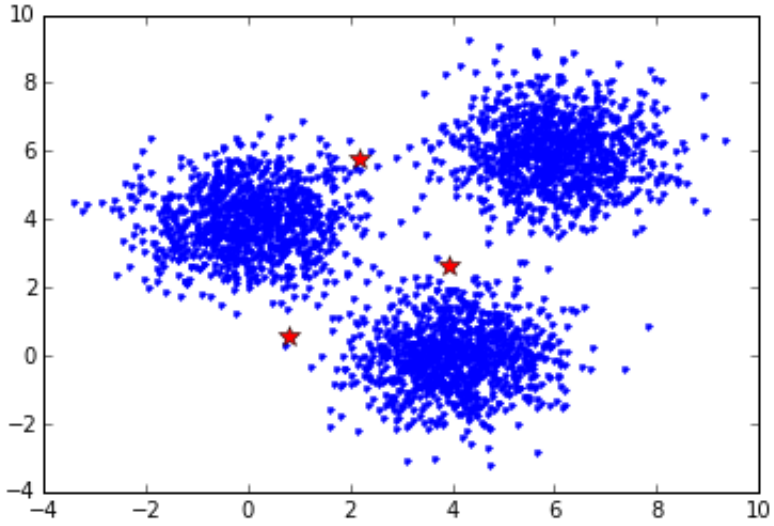
In [61]:
```
mape
```

Out[61]:

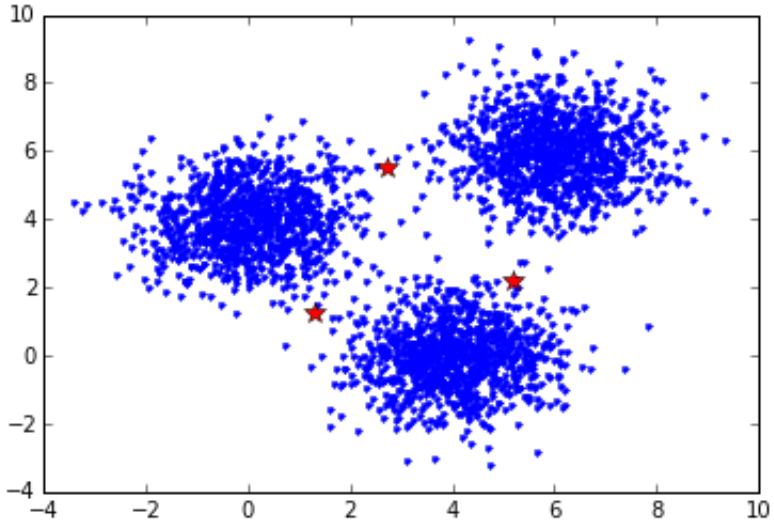
```
[41.532328719116876,
 161.91130298273154,
 266.67582417582418,
 393.60207100591708,
 301.05168269230768,
 256.49038461538458,
 446.11292962356794,
 201.96832579185519,
 335.03911342894395,
 307.41758241758237,
 350.30364372469631,
 375.32051282051276,
 36.467060167555218,
 32.454453441295549,
 294.88165680473372,
 541.68269230769226,
 43.712044534412961,
 45.836975524475513,
 320.77553593947033,
 182.0583262890955,
 335.03911342894395,
 209.24467099165892,
 526.03189493433388,
 446.11292962356794,
 205.56318681318677,
 201.96832579185519,
 121.26989389920423,
 52.817449095022631,
 71.16032843560933,
 30.815882230976577,
 53.918657643971834,
 66.880893300248147,
 8.7597783572359766,
 496.9141323792486,
 307.41758241758237,
 45.272265048384455,
 23.381171067738237,
 242.2307692307692,
 44.3225429667946,
 68.583466716881645,
 28.336538461538453,
 702.10336538461536,
 44.201505016722415,
 65.822493086141549,
 266.67582417582418,
 220.84134615384613,
 50.922929842623923,
 65.361258175023366,
 97.440828402366847,
 271.98996655518391,
 47.936495553128417,
 68.467680967680963]
```

In [7]:
```
K = 3
# Initialization: initialization of parameter is fixed to show an e
xample
centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])

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

    ####### We add the broadcasting of the centroids #########
    cBroadcast = sc.broadcast(centroids)
    #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])  #divi
de 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)
print "Final Results:"
print centroids
```

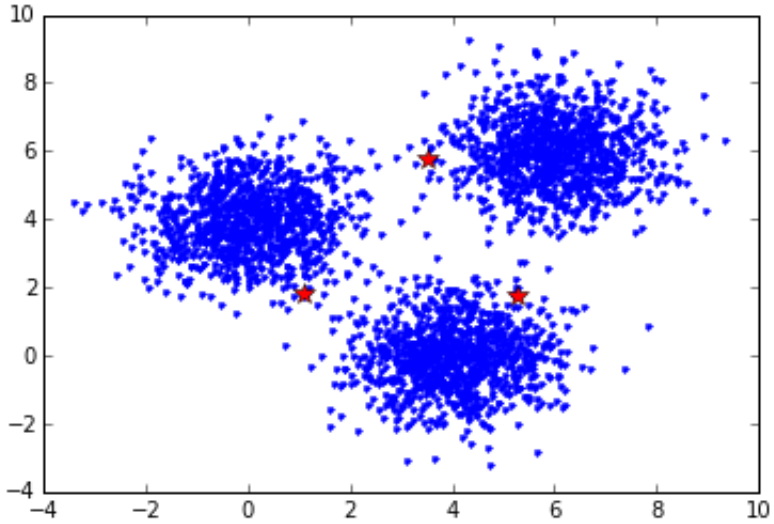
```
Iteration0
[[ 0.7928395   0.58581889]
 [ 3.94419807  2.62014238]
 [ 2.19179827  5.75110449]]
```



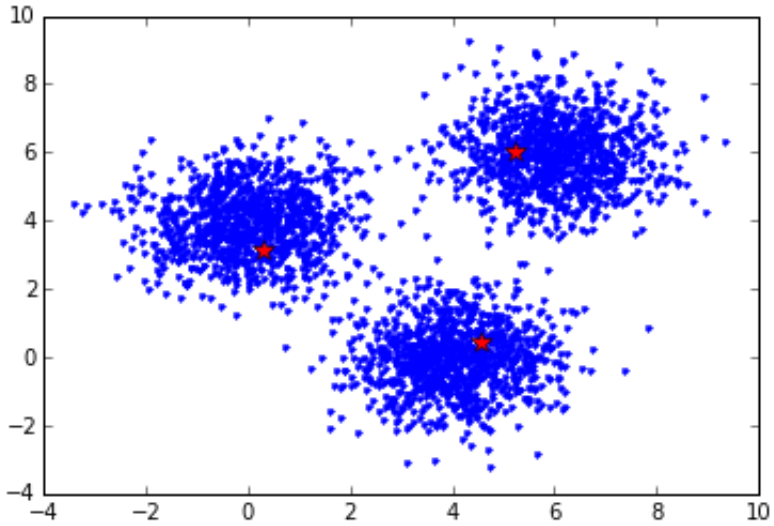
```
Iteration1
[[ 1.3100182   1.26199947]
 [ 5.18651521  2.17850522]
 [ 2.73071362  5.52071783]]
```



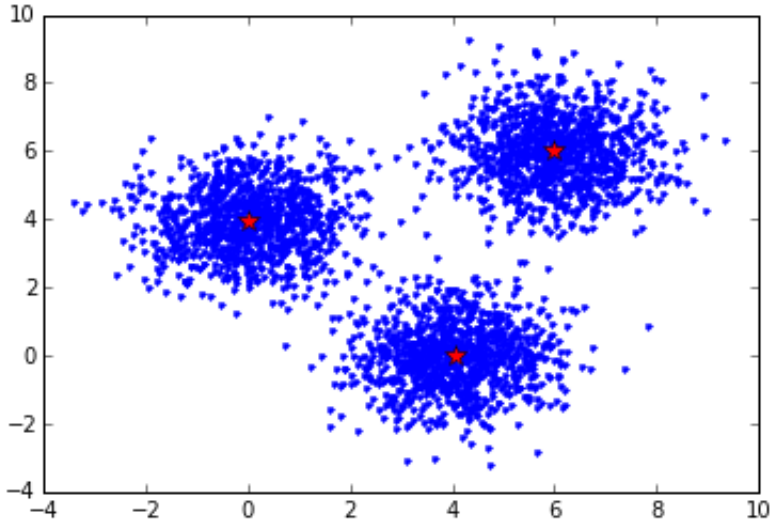
```
Iteration2
[[ 1.0740023   1.83590496]
 [ 5.2700425   1.73811837]
 [ 3.51631484  5.75866253]]
```



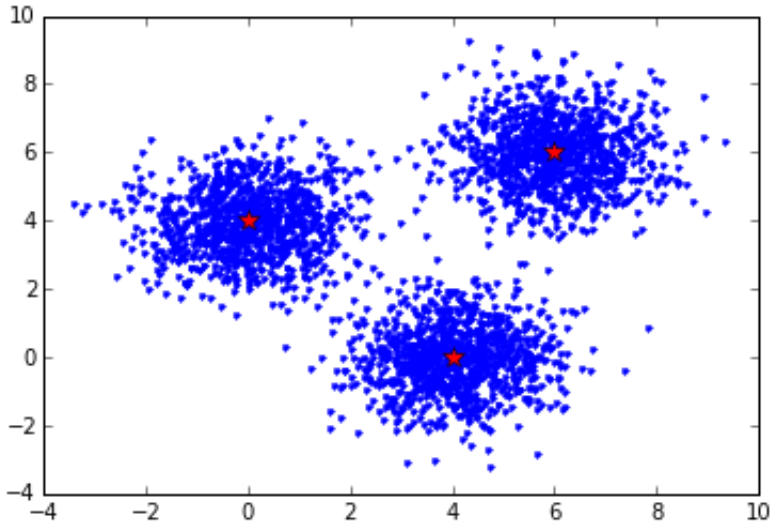

```
Iteration3
[[ 0.31669948  3.10905782]
 [ 4.55608738  0.41412711]
 [ 5.2139967   6.0379929 ]]
```




```
Iteration4
[[ 0.0107966   3.96004144]
 [ 4.04862205 -0.02311502]
 [ 5.98462538  6.00082204]]
```



```
Iteration5
[[ 0.00643579  3.9864617 ]
 [ 4.03103087 -0.01201179]
 [ 5.99828184  6.00327702]]
```



```
Final Results:
[[ 0.00643579  3.9864617 ]
 [ 4.03103087 -0.01201179]
 [ 5.99828184  6.00327702]]
```

In [ ]:

MLlib Kmeans

In [44]:
```
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.sp
lit(',')]))

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

```
[ 0.04365268  4.01254729]
[ 5.98103957  5.98221712]
[ 4.01573841 -0.05511536]
```

End of HW11.0

HW11.3 Logistic Regression

Generate 2 sets of linearly separable data with 100 data points each using the data generation code provided below and plot each in separate plots. Call one the training set and the other the testing set.

def generateData(n):

""" generates a 2D linearly separable dataset with n samples. The third element of the sample is the label """

xb = (rand(n)*2-1)/2-0.5

yb = (rand(n)*2-1)/2+0.5

xr = (rand(n)*2-1)/2+0.5

yr = (rand(n)*2-1)/2-0.5

inputs = []

for i in range(len(xb)):

inputs.append([xb[i],yb[i],1])

inputs.append([xr[i],yr[i],-1])

return inputs

Modify this data generation code to generating non-linearly separable training and testing datasets (with approximately 10% of the data falling on the wrong side of the separating hyperplane. Plot the resulting datasets.

NOTE: For the remainder of this problem please use the non-linearly separable training and testing datasets.

Using MLLib train up a LASSO logistic regression model with the training dataset and evaluate with the testing set. What a good number of iterations for training the logistic regression model? Justify with plots and words.

Derive and implement in Spark a weighted LASSO logistic regression. Implement a convergence test of your choice to check for termination within your training algorithm .

Weight the above training dataset as follows: Weight each example 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.

Evaluate your homegrown weighted LASSO logistic regression on the test dataset. Report misclassification error (1 - Accuracy) and how many iterations does it took to converge.

Does Spark MLLib have a weighted LASSO logistic regression implementation. If so use it and report your findings on the weighted training set and test set.

Data Generation

In [9]:
```
from collections import namedtuple
import numpy as np
import csv

def generateData(n):
    """ generates a 2D linearly separable dataset with n samples. T
he third element of the sample is the label """

    np.random.seed(0)
    xb = np.random.normal(0,0.5,n)-0.5
    yb = np.random.normal(0,0.5,n)+0.5
    xr = np.random.normal(0,0.5,n)+0.5
    yr = np.random.normal(0,0.5,n)-0.5
    inputs = []
    for i in range(len(xb)):
        inputs.append([xb[i],yb[i],1])
        inputs.append([xr[i],yr[i],-1])
    return inputs
```

In [11]:
```
train_set = generateData(50)
test_set = generateData(50)
```

In [38]:
```
w = np.array([8, -3, -1])
```

Data Visualization

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
def dataPlot(dataset):
    cols = {'1': 'r', '-1': 'b'}

    for row in dataset:
        plt.plot(float(row[0]), float(row[1]), cols[str(row[2])]
+'o')
    plt.xlabel("COPPER")
    plt.ylabel("CLP")
    x1 = [-2,2]
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, linewidth=2.0)
    plt.grid()
    plt.show()
```

In [2]:
```
dataPlot(train_set)
```

```
---------------------------------------------------------------------
---------
NameError                                 Traceback (most recent c
all last)
<ipython-input-2-551dd531f595> in <module>()
----> 1 dataPlot(train_set)

NameError: name 'train_set' is not defined
```

Start Spark

In [19]:
```
import os
import sys
##spark_home = os.environ['SPARK_HOME'] = '/Users/liang/Downloads/s
park-1.4.1-bin-hadoop2.6/'
##if not spark_home:
##    raise ValueError('SPARK_HOME enviroment variable is not set')
##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-src.zip'))
##execfile(os.path.join(spark_home,'python/pyspark/shell.py'))
```

Gradient descent (with regularization)

In [55]:

```
from collections import namedtuple
import numpy as np

Point = namedtuple('Point', 'x y')

def readPoint(line):
    label = data[2]
    x = [data[0], data[1], 1.0]  #add bias term
    return Point(x, label)

def vectorWeight(v1, v2):
    weight = 1.0/((v1**2+v2**2)**0.5)
    if weight < 0.1:
        weight = 0.1
    elif weight > 10:
        weight = 10
    return weight

def WeightedlogisticRegressionGD(data, wInitial=None, learningRate=
0.05, iterations=10, regParam=0.01, regType=None, stopCriteria=0.00
01):
    featureLen = len(data.take(1)[0].x)
    #total_weight = data.count()
    total_weight = data.map(lambda p: vectorWeight(p.x[0], p.x
[1])).reduce(lambda a,b: a+b)
    if wInitial is None:
        w = np.random.normal(size=featureLen) # w should be broadca
sted if it is large
    else:
        w = wInitial
    for i in range(iterations):
        print "Iteration %s"%(i+1)
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: vectorWeight(p.x[0], p.x[1])*
(1 / (1 + np.exp(-p.y*np.dot(wBroadcast.value, p.x)))-1) * p.y * n
p.array(p.x)).reduce(lambda a, b: a + b)
        #gradient = data.map(lambda p: (1 / (1 + np.exp(-p.y*np.dot
(wBroadcast.value, p.x)))-1) * p.y * np.array(p.x))\
        #            .reduce(lambda a, b: a + b)
        if regType == "Ridge":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term,
ignored in regularization
        elif regType == "Lasso":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term,
ignored in regularization
            wReg = (wReg>0).astype(int) * 2-1
        else:
            wReg = np.zeros(w.shape[0])
        gradient = gradient + regParam * wReg  #gradient:  GD of Sq
aured Error+ GD of regularized term
```

```
        wdelta = learningRate * gradient / total_weight # scale by
total weight

        if sum(abs(wdelta))<=stopCriteria*sum(abs(w)): #Convergence
condition
            print "convergenced reached at iteration %"%(i+1)
            break
        #w = w - learningRate * gradient / n
        w = w - wdelta
        print w
    print "total iterations: %s"%(i+1)
    return w
```

In [57]:
```
train_data_rdd = sc.parallelize(train_set).map(readPoint).cache()
WeightedlogisticRegressionGD(train_data_rdd)
```

```
---------------------------------------------------------------------
---------
Py4JJavaError                             Traceback (most recent c
all last)
<ipython-input-57-fa49e88c73fe> in <module>()
      1 train_data_rdd = sc.parallelize(train_set).map(readPoint).
cache()
----> 2 WeightedlogisticRegressionGD(train_data_rdd)

<ipython-input-55-a136b043f6a7> in WeightedlogisticRegressionGD(da
ta, wInitial, learningRate, iterations, regParam, regType, stopCri
teria)
     20     featureLen = len(data.take(1)[0].x)
     21     #total_weight = data.count()
---> 22     total_weight = data.map(lambda p: vectorWeight(p.x[0],
p.x[1])).reduce(lambda a,b: a+b)
     23     if wInitial is None:
     24         w = np.random.normal(size=featureLen) # w should b
e broadcasted if it is large

/Users/Lissette/Documents/datascience/software/spark-1.5.2-bin-had
oop2.6/python/pyspark/rdd.pyc in reduce(self, f)
    797             yield reduce(f, iterator, initial)
    798
--> 799         vals = self.mapPartitions(func).collect()
    800         if vals:
    801             return reduce(f, vals)

/Users/Lissette/Documents/datascience/software/spark-1.5.2-bin-had
oop2.6/python/pyspark/rdd.pyc in collect(self)
    771         """
    772         with SCCallSiteSync(self.context) as css:
--> 773             port = self.ctx._jvm.PythonRDD.collectAndServe
(self._jrdd.rdd())
    774         return list(_load_from_socket(port, self._jrdd_des
erializer))
    775

/Users/Lissette/Documents/datascience/software/spark-1.5.2-bin-had
oop2.6/python/lib/py4j-0.8.2.1-src.zip/py4j/java_gateway.py in __c
all__(self, *args)
    536         answer = self.gateway_client.send_command(command)
    537         return_value = get_return_value(answer, self.gatew
ay_client,
--> 538                 self.target_id, self.name)
    539
    540         for temp_arg in temp_args:

/Users/Lissette/Documents/datascience/software/spark-1.5.2-bin-had
oop2.6/python/pyspark/sql/utils.pyc in deco(*a, **kw)
     34     def deco(*a, **kw):
     35         try:
```

```
---> 36             return f(*a, **kw)
     37         except py4j.protocol.Py4JJavaError as e:
     38             s = e.java_exception.toString()

/Users/Lissette/Documents/datascience/software/spark-1.5.2-bin-had
oop2.6/python/lib/py4j-0.8.2.1-src.zip/py4j/protocol.py in get_ret
urn_value(answer, gateway_client, target_id, name)
    298                 raise Py4JJavaError(
    299                     'An error occurred while calling {0}
{1}{2}.\n'.
--> 300                     format(target_id, '.', name), value)
    301             else:
    302                 raise Py4JError(

Py4JJavaError: An error occurred while calling z:org.apache.spark.
api.python.PythonRDD.collectAndServe.
: org.apache.spark.SparkException: Job aborted due to stage failur
e: Task 0 in stage 29.0 failed 1 times, most recent failure: Lost
task 0.0 in stage 29.0 (TID 80, localhost): org.apache.spark.api.p
ython.PythonException: Traceback (most recent call last):
  File "/Users/Lissette/Documents/datascience/software/spark-1.5.2
-bin-hadoop2.6/python/lib/pyspark.zip/pyspark/worker.py", line 11
1, in main
    process()
  File "/Users/Lissette/Documents/datascience/software/spark-1.5.2
-bin-hadoop2.6/python/lib/pyspark.zip/pyspark/worker.py", line 10
6, in process
    serializer.dump_stream(func(split_index, iterator), outfile)
  File "/Users/Lissette/Documents/datascience/software/spark-1.5.2
-bin-hadoop2.6/python/lib/pyspark.zip/pyspark/serializers.py", lin
e 263, in dump_stream
    vs = list(itertools.islice(iterator, batch))
  File "/Users/Lissette/Documents/datascience/software/spark-1.5.2
-bin-hadoop2.6/python/pyspark/rdd.py", line 794, in func
    initial = next(iterator)
  File "<ipython-input-55-a136b043f6a7>", line 22, in <lambda>
  File "<ipython-input-55-a136b043f6a7>", line 13, in vectorWeight
ValueError: The truth value of an array with more than one element
is ambiguous. Use a.any() or a.all()

        at org.apache.spark.api.python.PythonRunner$$anon$1.read(P
ythonRDD.scala:166)
        at org.apache.spark.api.python.PythonRunner$$anon$1.<init>
(PythonRDD.scala:207)
        at org.apache.spark.api.python.PythonRunner.compute(Python
RDD.scala:125)
        at org.apache.spark.api.python.PythonRDD.compute(PythonRD
D.scala:70)
        at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.sc
ala:300)
        at org.apache.spark.rdd.RDD.iterator(RDD.scala:264)
        at org.apache.spark.scheduler.ResultTask.runTask(ResultTas
```

```
k.scala:66)
        at org.apache.spark.scheduler.Task.run(Task.scala:88)
        at org.apache.spark.executor.Executor$TaskRunner.run(Execu
tor.scala:214)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(Threa
dPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(Thre
adPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:745)

Driver stacktrace:
        at org.apache.spark.scheduler.DAGScheduler.org$apache$spar
k$scheduler$DAGScheduler$$failJobAndIndependentStages(DAGSchedule
r.scala:1283)
        at org.apache.spark.scheduler.DAGScheduler$$anonfun$abortS
tage$1.apply(DAGScheduler.scala:1271)
        at org.apache.spark.scheduler.DAGScheduler$$anonfun$abortS
tage$1.apply(DAGScheduler.scala:1270)
        at scala.collection.mutable.ResizableArray$class.foreach(R
esizableArray.scala:59)
        at scala.collection.mutable.ArrayBuffer.foreach(ArrayBuffe
r.scala:47)
        at org.apache.spark.scheduler.DAGScheduler.abortStage(DAGS
cheduler.scala:1270)
        at org.apache.spark.scheduler.DAGScheduler$$anonfun$handle
TaskSetFailed$1.apply(DAGScheduler.scala:697)
        at org.apache.spark.scheduler.DAGScheduler$$anonfun$handle
TaskSetFailed$1.apply(DAGScheduler.scala:697)
        at scala.Option.foreach(Option.scala:236)
        at org.apache.spark.scheduler.DAGScheduler.handleTaskSetFa
iled(DAGScheduler.scala:697)
        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoo
p.doOnReceive(DAGScheduler.scala:1496)
        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoo
p.onReceive(DAGScheduler.scala:1458)
        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoo
p.onReceive(DAGScheduler.scala:1447)
        at org.apache.spark.util.EventLoop$$anon$1.run(EventLoop.s
cala:48)
        at org.apache.spark.scheduler.DAGScheduler.runJob(DAGSched
uler.scala:567)
        at org.apache.spark.SparkContext.runJob(SparkContext.scal
a:1824)
        at org.apache.spark.SparkContext.runJob(SparkContext.scal
a:1837)
        at org.apache.spark.SparkContext.runJob(SparkContext.scal
a:1850)
        at org.apache.spark.SparkContext.runJob(SparkContext.scal
a:1921)
        at org.apache.spark.rdd.RDD$$anonfun$collect$1.apply(RDD.s
cala:909)
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOp
```

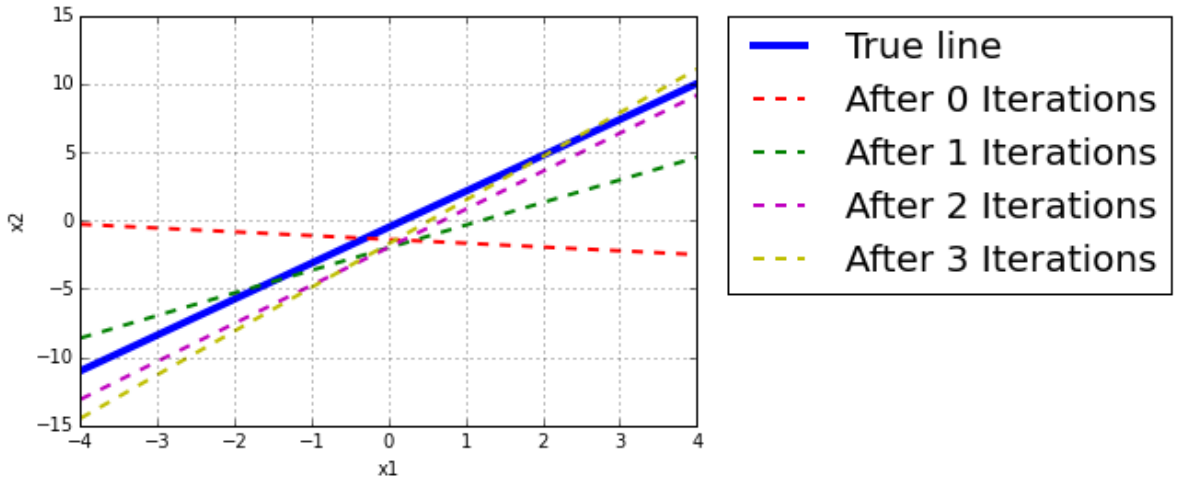
```
erationScope.scala:147)
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOp
erationScope.scala:108)
        at org.apache.spark.rdd.RDD.withScope(RDD.scala:310)
        at org.apache.spark.rdd.RDD.collect(RDD.scala:908)
        at org.apache.spark.api.python.PythonRDD$.collectAndServe
(PythonRDD.scala:405)
        at org.apache.spark.api.python.PythonRDD.collectAndServe(P
ythonRDD.scala)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Met
hod)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMetho
dAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Delegat
ingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.jav
a:231)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngin
e.java:379)
        at py4j.Gateway.invoke(Gateway.java:259)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractComm
and.java:133)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.GatewayConnection.run(GatewayConnection.java:207)
        at java.lang.Thread.run(Thread.java:745)
Caused by: org.apache.spark.api.python.PythonException: Traceback
(most recent call last):
  File "/Users/Lissette/Documents/datascience/software/spark-1.5.2
-bin-hadoop2.6/python/lib/pyspark.zip/pyspark/worker.py", line 11
1, in main
    process()
  File "/Users/Lissette/Documents/datascience/software/spark-1.5.2
-bin-hadoop2.6/python/lib/pyspark.zip/pyspark/worker.py", line 10
6, in process
    serializer.dump_stream(func(split_index, iterator), outfile)
  File "/Users/Lissette/Documents/datascience/software/spark-1.5.2
-bin-hadoop2.6/python/lib/pyspark.zip/pyspark/serializers.py", lin
e 263, in dump_stream
    vs = list(itertools.islice(iterator, batch))
  File "/Users/Lissette/Documents/datascience/software/spark-1.5.2
-bin-hadoop2.6/python/pyspark/rdd.py", line 794, in func
    initial = next(iterator)
  File "<ipython-input-55-a136b043f6a7>", line 22, in <lambda>
  File "<ipython-input-55-a136b043f6a7>", line 13, in vectorWeight
ValueError: The truth value of an array with more than one element
is ambiguous. Use a.any() or a.all()

        at org.apache.spark.api.python.PythonRunner$$anon$1.read(P
ythonRDD.scala:166)
        at org.apache.spark.api.python.PythonRunner$$anon$1.<init>
(PythonRDD.scala:207)
```

```
        at org.apache.spark.api.python.PythonRunner.compute(Python
RDD.scala:125)
        at org.apache.spark.api.python.PythonRDD.compute(PythonRD
D.scala:70)
        at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.sc
ala:300)
        at org.apache.spark.rdd.RDD.iterator(RDD.scala:264)
        at org.apache.spark.scheduler.ResultTask.runTask(ResultTas
k.scala:66)
        at org.apache.spark.scheduler.Task.run(Task.scala:88)
        at org.apache.spark.executor.Executor$TaskRunner.run(Execu
tor.scala:214)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(Threa
dPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(Thre
adPoolExecutor.java:617)
        ... 1 more
```

Plot w in iterations

In [22]:
```
def ierationsPlot(fileName, truew):
    x1 = [-4, 4]

    w = truew
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'b', label="True line", linewidth=4.0)

    np.random.seed(800)
    w = np.random.normal(0,1,3)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'r--', label="After 0 Iterations", linewidth=
2.0)

    data = sc.textFile(fileName).map(readPoint).cache()
    w = logisticRegressionGD(data, w, iterations=30)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'g--', label="After 1 Iterations", linewidth=
2.0)

    w = logisticRegressionGD(data, w, iterations=30)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'm--', label="After 2 Iterations", linewidth=
2.0)

    w = logisticRegressionGD(data, w, iterations=30)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'y--', label="After 3 Iterations", linewidth=
2.0)

    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, fontsize=20, border
axespad=0.)
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.grid()
    plt.show()
```

In [23]:
```
ierationsPlot(train_data)
```



Gradient descent (regularization)

In [24]:
```
def logisticRegressionGDReg(data, wInitial=None, learningRate=0.05,
iterations=50, regParam=0.01, regType=None):
    featureLen = len(data.take(1)[0].x)
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen) # w should be broadca
sted if it is large
    else:
        w = wInitial
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: (1 / (1 + np.exp(-p.y*np.dot
(wBroadcast.value, p.x)))-1) * p.y * np.array(p.x))\
                    .reduce(lambda a, b: a + b)
        if regType == "Ridge":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term,
ignored in regularization
        elif regType == "Lasso":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term,
ignored in regularization
            wReg = (wReg>0).astype(int) * 2-1
        else:
            wReg = np.zeros(w.shape[0])
        gradient = gradient + regParam * wReg  #gradient:  GD of Sq
aured Error+ GD of regularized term
        w = w - learningRate * gradient / n
    return w
```

Ridge Regression

In [25]:
```
np.random.seed(400)
logisticRegressionGDReg(data, iterations=50, regParam=0.1, regType
="Ridge")
```

Out[25]:
```
array([ 0.74835721, -0.17134752, -0.39953122])
```

Lasso Regression

In [26]:
```
np.random.seed(400)
logisticRegressionGDReg(data, iterations=50, regParam=0.1, regType
="Lasso")
```

Out[26]:
```
array([ 0.74788215, -0.17092861, -0.397468  ])
```

Mllib

In [ ]:
```
lrm = LogisticRegressionWithSGD.train(sc.parallelize(train_data_rd
d), iterations=10)
```

In [ ]:
```

```