Table of Contents

In [1]:
```
%%javascript
/*********************************************************************************
****************
Known Mathjax Issue with Chrome - a rounding issue adds a border to the right o
f mathjax markup
https://github.com/mathjax/MathJax/issues/1300
A quick hack to fix this based on stackoverflow discussions:
http://stackoverflow.com/questions/34277967/chrome-rendering-mathjax-equations-
with-a-trailing-vertical-line
*********************************************************************************
***************/

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

MIDS - w261 Machine Learning At Scale

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

Assignment - HW4

Name: *Your Name Goes Here*
Class: MIDS w261 (Section *Your Section Goes Here*, e.g., Fall 2016 Group 1)
Email: *Your UC Berkeley Email Goes Here*@iSchool.Berkeley.edu
StudentId 26302327 **End of StudentId**
Week: 4

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

Due Time: HW is due the Tuesday of the following week by 8AM (West coast time). I.e., Tuesday, Feb 7, 2017 in the case of this homework.

Table of Contents

1 Instructions

Back to Table of Contents

MIDS UC Berkeley, Machine Learning at Scale DATSCIW261 ASSIGNMENT #4

Version 2017-26-1

IMPORTANT

This homework can be completed locally on your computer

=== INSTRUCTIONS for SUBMISSIONS ===

Follow the instructions for submissions carefully.

NEW: Going forward, each student will have a `HW-<user>` repository for all assignments.

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

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

- Your local HW4 directory. Your repo file structure should look like this:

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

2 Useful References

Back to Table of Contents

- See async lecture and live lecture

HW Problems

Back to Table of Contents

HW4.0

Back to Table of Contents

What is MrJob? How is it different to Hadoop MapReduce? What are the mapper_init, mapper_final(), combiner_final(), reducer_final() methods? When are they called?

```
mrjob provides a pythonic API to work with Hadoop Streaming, and allows the user to
work with any objects as keys and mappers. mrjob is the easiest route to writing Py
thon programs that run on Hadoop. If you use mrjob, you'll be able to test your cod
e locally without installing Hadoop or run it on a cluster of your choice.

mrjob has extensive integration with Amazon Elastic MapReduce. Once you're set up,
it's as easy to run your job in the cloud as it is to run it on your laptop.

Hadoop MapReduce Framework takes care of actual implementation/Execution of map/Red
uce/combine/shuffle activities. While MrJob provided the framework to write program
s that internally genertes the instruction for MapReduce. It provides the level of
abstraction for devwelopers from complex MapReduce Syntax.

mapper_init(),mapper_final,combiner_final and Reducer_final are method in class MRJ
ob .We define subclass of MRjob and override these methods while writing Mrjobs.
```

MRJob.mapper_init()

```
Re-define this to define an action to run before the mapper processes any input.

One use for this function is to initialize mapper-specific helper structures.
```

MRJob.mapper_final()

```
Re-define this to define an action to run after the mapper reaches the end of input
.
```

MRJob.combiner_final()

```
Re-define this to define an action to run after the combiner reaches the end of inp
ut.
```

MRJob.reducer_final()

Re-define this to define an action to run after the reducer reaches the end of input.

HW4.1

Back to Table of Contents What is serialization in the context of MrJob or Hadoop? When it used in these frameworks? What is the default serialization mode for input and outputs for MrJob?

In computer science, in the context of data storage, serialization is the process of translating data structures or object state into a format that can be stored or transmitted and reconstructed later.

Hadoop

Though we see data in a structured form, the raw form of data is a sequence or stream of bits. This raw form of data is the one that travels over the network and is stored in RAM or any other persistent media. Serialization is the process of converting structured data into its raw form. Deserialization is the reverse process of reconstructing structured forms from the data's raw bit stream form.

MrJob

While working with MRJob, it is critical to understand that MRJob is an abstraction layer above Hadoop Streaming, and does not extend or modify Hadoop Streaming's normal behaviors. Hadoop streaming works through Unix piping, and uses stdout and stderr as communication channels. In Unix environments, the protocol of data transfer is text streams – mappers, reducers and Hadoop libraries exchange data through text.

Therefore, data structures common in programming languages (e.g. dictionary in Python) have to be serialized to text for Hadoop Streaming to understand and consume. The implication of this on total sorting is this: since Hadoop only sees text, custom data structures (as keys) will be sorted as strings in their serialized forms.

RawValueProtocol is default input protocol seerialization mode for Mrjobs.

The default output and internal protocols are both JSONProtocol ,which reads and writes JSON strings separated by a tab character.

In [2]:
```
%%bash
curl -L http://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/anon
ymous-msweb.data -o anonymous-msweb.data
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
t
                                 Dload  Upload   Total   Spent    Left  Speed
100 1389k  100 1389k    0     0   875k      0  0:00:01  0:00:01 --:--:--  877k
```

In [3]:
```
!head -20 anonymous-msweb.data
```

```
I,4,"www.microsoft.com","created by getlog.pl"
T,1,"VRoot",0,0,"VRoot"
N,0,"0"
N,1,"1"
T,2,"Hide1",0,0,"Hide"
N,0,"0"
N,1,"1"
A,1287,1,"International AutoRoute","/autoroute"
A,1288,1,"library","/library"
A,1289,1,"Master Chef Product Information","/masterchef"
A,1297,1,"Central America","/centroam"
A,1215,1,"For Developers Only Info","/developer"
A,1279,1,"Multimedia Golf","/msgolf"
A,1239,1,"Microsoft Consulting","/msconsult"
A,1282,1,"home","/home"
A,1251,1,"Reference Support","/referencesupport"
A,1121,1,"Microsoft Magazine","/magazine"
A,1083,1,"MS Access Support","/msaccesssupport"
A,1145,1,"Visual Fox Pro Support","/vfoxprosupport"
A,1276,1,"Visual Test Support","/vtestsupport"
```

HW4.2 Preprocess log file data

Back to Table of Contents

Recall the Microsoft logfiles data from the async lecture. The logfiles are described are located at:

https://kdd.ics.uci.edu/databases/msweb/msweb.html (https://kdd.ics.uci.edu/databases/msweb/msweb.html)
http://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/ (http://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/)

This dataset captures which areas (Vroots) of www.microsoft.com each user visited in a one-week timeframe in Feburary 1998.

Data Format

```
The data is in an ASCII-based sparse-data format called "DST". Each line of the dat
a file starts with a letter which tells the line's type. The three line types of in
terest are:
-- Attribute lines:
For example, 'A,1277,1,"NetShow for PowerPoint","/stream"'
Where:
  'A' marks this as an attribute line,
  '1277' is the attribute ID number for an area of the website (called a Vroot),
  '1' may be ignored,
  '"NetShow for PowerPoint"' is the title of the Vroot,
  '"/stream"' is the URL relative to "http://www.microsoft.com (http://www.microsof
t.com)"



Case and Vote Lines:
For each user, there is a case line followed by zero or more vote lines.
For example:
  C,"10164",10164
  V,1123,1
  V,1009,1
  V,1052,1
Where:
  'C' marks this as a case line,
  '10164' is the case ID number of a user,
  'V' marks the vote lines for this case,
  '1123', 1009', 1052' are the attributes ID's of Vroots that a user visited.
  '1' may be ignored.
```

</PRE>

```
Here, you must transform/preprocess the data on a single node (i.e., not on a clust
er of nodes) from the following format:
```

- `C,"10001",10001   #Visitor id 10001`
- `V,1000,1          #Visit by Visitor 10001 to page id 1000`
- `V,1001,1          #Visit by Visitor 10001 to page id 1001`
- `V,1002,1          #Visit by Visitor 10001 to page id 1002`

In [4]:
```
#START STUDENT CODE42
import csv

# Create files to write the processed data to
outfile = open('anonymous-msweb-preprocessed.data', 'wb')
outWriter = csv.writer(outfile)
attributes = open('attributes.csv', 'wb')
attWriter = csv.writer(attributes)

with open('anonymous-msweb.data', 'r') as infile:
    for line in csv.reader(infile):

        # Check if this line is an attribute
        # If it is, write to the attributes file
        if line[0] == 'A':
            attWriter.writerow([line[1], line[3], line[4]])

        # Save Visitor ID to populate on next records.
        elif line[0] == 'C':
            visitor_id = line[2]

        # Check if this line is a visit with a page ID
        # If it is, write it with the current visitor_id
        elif line[0] == 'V':
            outWriter.writerow([line[0], line[1], line[2], 'C', visitor_id])

outfile.close()
attributes.close()
#END STUDENT CODE42
```

In [5]:
```
!head -10 anonymous-msweb-preprocessed.data
!wc -l anonymous-msweb-preprocessed.data
```

```
V,1000,1,C,10001
V,1001,1,C,10001
V,1002,1,C,10001
V,1001,1,C,10002
V,1003,1,C,10002
V,1001,1,C,10003
V,1003,1,C,10003
V,1004,1,C,10003
V,1005,1,C,10004
V,1006,1,C,10005
98654 anonymous-msweb-preprocessed.data
```

HW4.3 Find the most frequent pages

Back to Table of Contents

Find the 5 most frequently visited pages using MrJob from the output of 4.2 (i.e., transfromed log file).

In [6]:
```
%load_ext autoreload
%autoreload 2
```

In [7]:
```
%%writefile MostFrequentVisits.py
#!/usr/bin/env python
#START STUDENT CODE43

from mrjob.job import MRJob
from mrjob.step import MRStep
import re
import csv

def read_csvLine(line):
    # Given a comma delimited string, return fields
    for row in csv.reader([line]):
        return row

class MRMostVisits(MRJob):

#    OUTPUT_PROTOCOL = JSONValueProtocol
    SORT_VALUES = True

    def __init__(self, *args, **kwargs):
        super(MRMostVisits, self).__init__(*args, **kwargs)
        self.N = 5



    def mapper_get_pages(self, _, line):
        # yield page numbers now
        fields = read_csvLine(line)
        yield fields[1], 1

    def combiner_count_pages(self, page, counts):
        # sum the all counts in json format
        yield (page, sum(counts))

    def reducer_count_pages(self, page, counts):
        # send all (num_occurrences, word) pairs to the same reducer.
        # num_occurrences is so we can easily use Python's max() function.
        yield None, (sum(counts), page)

    # discard the key; it is just None
    def reducer_find_top_pages(self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=counts, value=word
        word_count_pairs = sorted(word_count_pairs,reverse=True)
        for i in range(self.N):#as only top 5 pages requested
            yield word_count_pairs[i]

    def steps(self):  #pipeline of Map-Reduce jobs
        return [
            MRStep(
                    mapper=self.mapper_get_pages,       # STEP 1: word count st
ep
                    combiner=self.combiner_count_pages,
                    reducer=self.reducer_count_pages),
            MRStep(reducer=self.reducer_find_top_pages) # Step 2: most frequent
word
        ]

if __name__ == '__main__':
    MRMostVisits.run()


#END STUDENT CODE43
```

```
Overwriting MostFrequentVisits.py
```

In [8]:
```
#Unit test
!chmod a+x MostFrequentVisits.py
!python MostFrequentVisits.py anonymous-msweb-preprocessed.data
```

```
No configs found; falling back on auto-configuration
ignoring partitioner keyword arg (requires real Hadoop): 'org.apache.hadoop.ma
pred.lib.KeyFieldBasedPartitioner'
Creating temp directory /tmp/MostFrequentVisits.root.20170606.033353.327879
Running step 1 of 2...
Running step 2 of 2...
Streaming final output from /tmp/MostFrequentVisits.root.20170606.033353.32787
9/output...
10836   "1008"
9383    "1034"
8463    "1004"
5330    "1018"
5108    "1017"
Removing temp directory /tmp/MostFrequentVisits.root.20170606.033353.327879...
```

In [9]:
```
from MostFrequentVisits import MRMostVisits
mr_job = MRMostVisits(args=['anonymous-msweb-preprocessed.data'])


with mr_job.make_runner() as runner:
    # Run MRJob
    runner.run()

    # Write stream_output to file
    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

```
No handlers could be found for logger "mrjob.sim"

(10836, u'1008')
(9383, u'1034')
(8463, u'1004')
(5330, u'1018')
(5108, u'1017')
```

HW4.4 Find the most frequent visitor

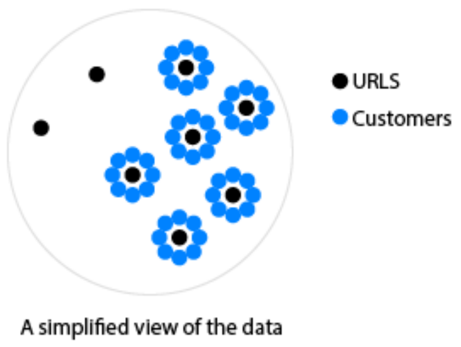
Back to Table of Contents

Find the most frequent visitor of each page using MrJob and the output of 4.2 (i.e., transfromed log file). In this output please include the webpage URL, webpageID and Visitor ID. You may get a weird result. HINT: The maximum visits by any visitor to any given webpage is 1.

In [10]:
```
from IPython.display import Image, HTML
```

In [11]:
```
Image('ms-data.png')
```

Out[11]:



In [12]:
```
%%writefile mostFrequentVisitors.py
#!/usr/bin/env python
#START STUDENT CODE44
from mrjob.job import MRJob
from mrjob.step import MRStep
import re
import csv
from collections import defaultdict
import operator
def read_csvLine(line):
    # Given a comma delimited string, return fields
    for row in csv.reader([line]):
        return row

class MRMostVisitor(MRJob):

#    OUTPUT_PROTOCOL = JSONValueProtocol
    SORT_VALUES = True
    mydict = {}
    def __init__(self, *args, **kwargs):
        super(MRMostVisitor, self).__init__(*args, **kwargs)





    def mapper_get_pages(self, _, line):
        # yield each word in the line
        fields = read_csvLine(line)
        #using hint:the maximum visits by any visitor to any given webpage is 1
.
        yield fields[1],{fields[4]:1}

    def combiner_count_pages(self, page, visits):
        # sum the words we've seen so far

        visitsdict =defaultdict(int)
        for visit in visits:
            for custID in visit.keys():

                visitsdict[custID] += visit[custID]
        yield page,visitsdict
    def reducer_init_function(self):
        with open('attributes.csv', 'rb') as f:
            reader = csv.reader(f)
            for row in reader:
                #populating this dictionary to get the url link in future
                #not memory efficient but works as master data is limited.
                self.mydict[row[0]] = row[2]

    def reducer_find_top_pages(self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=counts, value=word
        #word_count_pairs = sorted(word_count_pairs,reverse=True)
        count =0
        word_count_pairs = sorted(word_count_pairs)
        for pagedetails in word_count_pairs:
               print pagedetails


    def reducer_count_pages(self, page, visits):
        # send all (num_occurrences, word) pairs to the same reducer.
        # num_occurrences is so we can easily use Python's max() function.
        visitsdict =defaultdict(int)
        for visit in visits:
            for custID in visit.keys():

                visitsdict[custID] += visit[custID]
        #find max by value
```

```
Overwriting mostFrequentVisitors.py
```

In [13]:
```
!chmod +x mostFrequentVisitors.py
```

In [14]:
```
from mostFrequentVisitors import MRMostVisitor
mr_job = MRMostVisitor(args=['anonymous-msweb-preprocessed.data', "--file" ,"at
tributes.csv"])


with mr_job.make_runner() as runner:
    # Run MRJob
    runner.run()

    # Write stream_output to file
    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

```
1000,/regwiz,36585,1
1001,/support,23995,1
1002,/athome,35235,1
1003,/kb,22469,1
1004,/search,35540,1
1005,/norge,10004,1
1006,/misc,27495,1
1007,/ie_intl,19492,1
1008,/msdownload,35236,1
1009,/windows,23995,1
1010,/vbasic,20915,1
1011,/officedev,40152,1
1012,/outlookdev,37811,1
1013,/vbasicsupport,32727,1
1014,/officefreestuff,20914,1
1015,/msexcel,16662,1
1016,/excel,33755,1
1017,/products,37091,1
1018,/isapi,35237,1
1019,/mspowerpoint,16765,1
1020,/msdn,39325,1
1021,/visualc,35234,1
1022,/truetype,15906,1
1023,/spain,16079,1
1024,/iis,20447,1
1025,/gallery,35234,1
1026,/sitebuilder,23990,1
1027,/intdev,35234,1
1028,/oledev,11191,1
1029,/clipgallerylive,33083,1
1030,/ntserver,20447,1
1031,/msoffice,22505,1
1032,/games,35542,1
1033,/logostore,38870,1
1034,/ie,35540,1
1035,/windowssupport,35546,1
1036,/organizations,22505,1
1037,/windows95,19490,1
1038,/sbnmember,36585,1
1039,/isp,26948,1
1040,/office,16078,1
1041,/workshop,35234,1
1042,/vstudio,18312,1
1043,/smallbiz,28326,1
1044,/mediadev,40224,1
1045,/netmeeting,20917,1
1046,/iesupport,18496,1
1048,/publisher,33083,1
1049,/supportnet,33329,1
1050,/macoffice,30757,1
1051,/scheduleplus,32702,1
1052,/word,20914,1
1053,/visualj,36585,1
1054,/exchange,23200,1
1055,/kids,39791,1
1056,/sports,27954,1
1057,/powerpoint,39792,1
1058,/referral,19490,1
1059,/sverige,19263,1
1060,/msword,20914,1
1061,/promo,25070,1
1062,/msaccess,36585,1
1063,/intranet,39793,1
1064,/activeplatform,42285,1
1065,/java,33247,1
1066,/musicproducer,39362,1
1067,/frontpage,33738,1
1068,/vbscript,19548,1
1069,/windowsce,32702,1
1070,/activex,35234,1
```

HW4.5 Clustering Tweet Dataset

Back to Table of Contents

Here you will use a different dataset consisting of word-frequency distributions for 1,000 Twitter users. These Twitter users use language in very different ways, and were classified by hand according to the criteria:

0: Human, where only basic human-human communication is observed.

1: Cyborg, where language is primarily borrowed from other sources (e.g., jobs listings, classifieds postings, advertisements, etc...).

2: Robot, where language is formulaically derived from unrelated sources (e.g., weather/seismology, police/fire event logs, etc...).

3: Spammer, where language is replicated to high multiplicity (e.g., celebrity obsessions, personal promotion, etc...)

Check out the preprints of recent research, which spawned this dataset:

- http://arxiv.org/abs/1505.04342 (http://arxiv.org/abs/1505.04342)
- http://arxiv.org/abs/1508.01843 (http://arxiv.org/abs/1508.01843)

The main data lie in the accompanying file:

- topUsers_Apr-Jul_2014_1000-words.txt (https://www.dropbox.com/s/6129k2urvbvobkr/topUsers_Apr-Jul_2014_1000-words.txt?dl=0)

and are of the form:

USERID,CODE,TOTAL,WORD1_COUNT,WORD2_COUNT,.... . .

where

USERID = unique user identifier CODE = 0/1/2/3 class code TOTAL = sum of the word counts

Using this data, you will implement a 1000-dimensional K-means algorithm in MrJob on the users by their 1000-dimensional word stripes/vectors using several centroid initializations and values of K.

Note that each "point" is a user as represented by 1000 words, and that word-frequency distributions are generally heavy-tailed power-laws (often called Zipf distributions), and are very rare in the larger class of discrete, random distributions. For each user you will have to normalize by its "TOTAL" column. **Try several parameterizations and initializations** :

- (A) K=4 uniform random centroid-distributions over the 1000 words (generate 1000 random numbers and normalize the vectors)
- (B) K=2 perturbation-centroids, randomly perturbed from the aggregated (user-wide) distribution
- (C) K=4 perturbation-centroids, randomly perturbed from the aggregated (user-wide) distribution
- (D) K=4 "trained" centroids, determined by the sums across the classes. Use use the (row-normalized) class-level aggregates as 'trained' starting centroids (i.e., the training is already done for you!). Note that you do not have to compute the aggregated distribution or the class-aggregated distributions, which are rows in the auxiliary file:

- topUsers_Apr-Jul_2014_1000-words_summaries.txt (https://www.dropbox.com/s/w4oklbsoqefou3b/topUsers_Apr-Jul_2014_1000-words_summaries.txt?dl=0)

Row 1: Words Row 2: Aggregated distribution across all classes Row 3-6 class-aggregated distributions for clases 0-3 For (A), we select 4 users randomly from a uniform distribution [1,...,1,000] For (B), (C), and (D) you will have to use data from the auxiliary file:

- topUsers_Apr-Jul_2014_1000-words_summaries.txt (https://www.dropbox.com/s/w4oklbsoqefou3b/topUsers_Apr-Jul_2014_1000-words_summaries.txt?dl=0)

This file contains 5 special word-frequency distributions:

- (1) The 1000-user-wide aggregate, which you will perturb for initializations in parts (B) and (C), and

In [15]:
```
from numpy import random
numbers = random.sample(1000)
```

Take these 1000 numbers and add them (component-wise) to the 1000-user aggregate, and then renormalize to obtain one of your aggregate-perturbed initial centroids.

In [16]:
```
################################################################################
####
##Geneate random initial centroids around the global aggregate
##Part (B) and (C) of this question
################################################################################
####
def startCentroidsBC(k):
    counter = 0
    for line in open("topUsers_Apr-Jul_2014_1000-words_summaries.txt").readline
s():
        if counter == 2:
            data = re.split(",",line)
            globalAggregate = [float(data[i+3])/float(data[2]) for i in range(1
000)]
        counter += 1
    #perturb the global aggregate for the four initializations
    centroids = []
    for i in range(k):
        rndpoints = random.sample(1000)
        peturpoints = [rndpoints[n]/10+globalAggregate[n] for n in range(1000)]
        centroids.append(peturpoints)
        total = 0
        for j in range(len(centroids[i])):
            total += centroids[i][j]
        for j in range(len(centroids[i])):
            centroids[i][j] = centroids[i][j]/total
    return centroids
```

For experiments A, B, C and D and iterate until a threshold (try 0.001) is reached. After convergence, print out a summary of the classes present in each cluster. In particular, report the composition as measured by the total portion of each class type (0-3) contained in each cluster, and discuss your findings and any differences in outcomes across parts A-D.

K-Means

K-means is a clustering method that aims to find the positions μi,i=1...k of the clusters that minimize the distance from the data points to the cluster. K-means clustering solves:

$$\arg\min_c \sum_{i=1}^{k} \sum_{x \in c_i} d(x, \mu_i) = \arg\min_c \sum_{i=1}^{k} \sum_{x \in c_i} \|x - \mu_i\|_2^2$$

where c_i is the set of points that belong to cluster i. The K-means clustering uses the square of the Euclidean distance $d(x, \mu_i) = \|x - \mu_i\|_2^2$. This problem is not trivial (in fact it is NP-hard), so the K-means algorithm only hopes to find the global minimum, possibly getting stuck in a different solution.

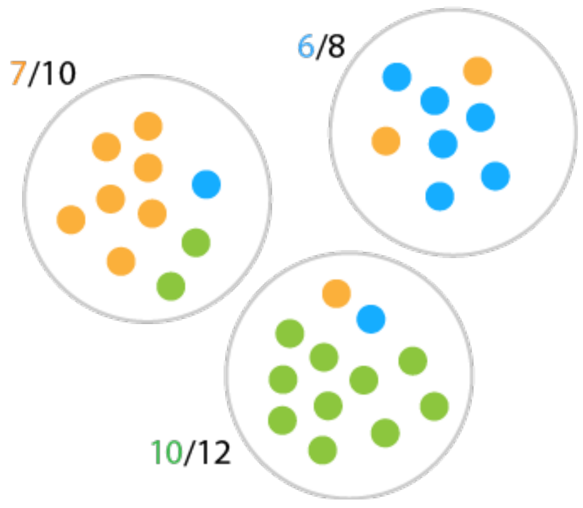
K-means algorithm

The Lloyd's algorithm, mostly known as k-means algorithm, is used to solve the k-means clustering problem and works as follows. First, decide the number of clusters k. Then:

1. Initialize the center of the clusters	μ_i = some value , $i = 1, \ldots, k$
2. Attribute the closest cluster to each data point	$c_i = \{j : d(x_j, \mu_i) \leq d(x_j, \mu_l), l \neq i, j = 1, \ldots, n\}$
3. Set the position of each cluster to the mean of all data points belonging to that cluster	$\mu_i = \frac{1}{\|c_i\|} \sum_{j \in c_i} x_j, \forall i$
4. Repeat steps 2-3 until convergence	
Notation	$\|c\|$ = number of elements in c

Calculating purity

$$\text{Purity} = \frac{7 + 6 + 10}{10 + 8 + 12} = 76.6\%$$



In [17]:

```
%%bash
rm topUsers_Apr-Jul_2014_1000-words.txt
rm topUsers_Apr-Jul_2014_1000-words_summaries.txt
curl -L https://www.dropbox.com/s/6129k2urvbvobkr/topUsers_Apr-Jul_2014_1000-wo
rds.txt?dl=0 -o topUsers_Apr-Jul_2014_1000-words.txt
curl -L https://www.dropbox.com/s/w4oklbsoqefou3b/topUsers_Apr-Jul_2014_1000-wo
rds_summaries.txt?dl=0 -o topUsers_Apr-Jul_2014_1000-words_summaries.txt
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
t
                                 Dload  Upload   Total   Spent    Left  Speed
100 2493k  100 2493k    0     0   643k      0  0:00:03  0:00:03 --:--:-- 3375k
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
t
                                 Dload  Upload   Total   Spent    Left  Speed
100 31952  100 31952    0     0  25077      0  0:00:01  0:00:01 --:--:-- 1950k
```

In [18]:
```
%%writefile Kmeans.py
#!/usr/bin/env python
#START STUDENT CODE45
from numpy import argmin, array, random
from mrjob.job import MRJob
from mrjob.step import MRStep
from itertools import chain
import os
import re

#Calculate find the nearest centroid for data point
#copied from http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/oppgyfqxphlh69g/
MrJobKmeans_Corrected.ipynb
def MinDist(datapoint, centroid_points):
    datapoint = array(datapoint)
    centroid_points = array(centroid_points)
    diff = datapoint - centroid_points
    diffsq = diff*diff
    # Get the nearest centroid for each instance
    minidx = argmin(list(diffsq.sum(axis = 1)))
    return minidx

#Check whether centroids converge

#copied from http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/oppgyfqxphlh69g/
MrJobKmeans_Corrected.ipynb
def stop_criterion(centroid_points_old, centroid_points_new,T):
    oldvalue = list(chain(*centroid_points_old))
    newvalue = list(chain(*centroid_points_new))
    Diff = [abs(x-y) for x, y in zip(oldvalue, newvalue)]
    # if centroids are not changing enough
    Flag = True
    for i in Diff:
        if(i>T):
            Flag = False
            break
    return Flag

class MRKmeans(MRJob):
    centroid_points=[]
    k = 4
    n = 1000
    def steps(self):
        return [
            MRStep(mapper_init = self.mapper_init, mapper=self.mapper,combiner
= self.combiner,reducer=self.reducer)
               ]
    #load centroids info from file
    def mapper_init(self):
         self.centroid_points = [map(float,s.split('\n')[0].split(',')) for s i
n open("centroids.txt").readlines()]

    #load data and output the nearest centroid index and data point
    def mapper(self, _, line):
        total = 0
        data = re.split(',',line)
        ID = data[0]
        code = int(data[1])
        users = [ID]
        codes = [0,0,0,0] #one hot representations
        codes[code] = 1
        #normalize coordinates first
        coords = [float(data[i+3])/float(data[2]) for i in range(1000)]
        for coord in coords:
            total += coord


        #generate key value pair.  Key --> 0-3 class of user and then value is
user/1/cordinates and codes
```

```
Overwriting Kmeans.py
```

In [19]:
```
%%writefile kmeans_runner.py
#!/usr/bin/env python
#START STUDENT CODE45_RUNNER


#!/usr/bin/env python
from numpy import random
from Kmeans import MRKmeans
from itertools import chain
import re,sys
mr_job = MRKmeans(args=["topUsers_Apr-Jul_2014_1000-words.txt","--file","centro
ids.txt"])

thresh = 0.0001

scriptName,part = sys.argv

## only stop when distance is below thresh for all centroids
def stopSignal(k,thresh,newCentroids,oldCentroids):
    stop = 1
    for i in range(k):
        dist = 0
        for j in range(len(newCentroids[i])):
            dist += (newCentroids[i][j] - oldCentroids[i][j]) ** 2
        dist = dist ** 0.5
        if (dist > thresh):
            stop  = 0
            break
    return stop

def stop_criterion(centroid_points_old, centroid_points_new,T):
    oldvalue = list(chain(*centroid_points_old))
    newvalue = list(chain(*centroid_points_new))
    Diff = [abs(x-y) for x, y in zip(oldvalue, newvalue)]
    Flag = True
    for i in Diff:
        if(i>T):
            Flag = False
            break
    return Flag
###############################################################################
###
# Use four centroids from the coding
###############################################################################
###
def partA():

    centroids = []
    for i in range(k):
        rndpoints = random.sample(1000)
        total = sum(rndpoints)
        #normalize first
        centroid = [pt/total for pt in rndpoints]
        centroids.append(centroid)
    return centroids
###############################################################################
####
def startCentroidsBC(k):
    counter = 0
    for line in open("topUsers_Apr-Jul_2014_1000-words_summaries.txt").readline
s():
        if counter == 1:
            data = re.split(",",line)
            globalAggregate = [float(data[i+3])/float(data[2]) for i in range(1
000)]
        counter += 1
    #perturb the global aggregate for the four initializations
    centroids = []
    for i in range(k):
```

```
Overwriting kmeans_runner.py
```

In [20]:
```
!chmod +x kMeans.py kmeans_runner.py
```

In [21]:
```
#generate summary now
#report the composition as measured by the total portion of each class type (0-
3) contained in each cluster,
#and discuss your findings and any differences in outcomes across parts A-D.
from collections import defaultdict
import csv
import pandas as pd
import numpy as np
from itertools import chain
from numpy import argmin, array, random
#Calculate find the nearest centroid for data point
def MinDist(datapoint, centroid_points):
    datapoint = array(datapoint)
    centroid_points = array(centroid_points)
    diff = datapoint - centroid_points
    diffsq = diff*diff
    # Get the nearest centroid for each instance
    minidx = argmin(list(diffsq.sum(axis = 1)))
    return minidx
def generate_summary():

    centroids = [map(float,s.split('\n')[0].split(',')) for s in open("centroid
s.txt").readlines()]

    #no of classes in final model
    k = len(centroids)
    pred_count = [0 for x in range(k)]
    cluster_id = ['0','1','2','3']
    cluster_id_total = ['0','1','2','3','Total']
    numc = defaultdict(int)
    numc_preds = defaultdict(int)
    ndata = np.zeros((4, 5),dtype=np.int16)
    with open('topUsers_Apr-Jul_2014_1000-words.txt') as myfile:
        for line in csv.reader(myfile):
            true_class = int(line[1])
            point = [float(i) for i in line[3:]]
            #generate predictions for latest centroids
            pred_class = int(MinDist(point, centroids))
            numc[true_class] += 1
            numc_preds[(true_class,pred_class)] += 1
            ndata[true_class,pred_class] += 1
            ndata[true_class,4] = numc[true_class]


    dta = pd.DataFrame(ndata,cluster_id, cluster_id_total)
    print dta
```

In [22]:
```
!python kMeans_runner.py A > Adata.txt
generate_summary()
```

```
    0   1   2    3  Total
0   7  17  26  702    752
1   0  89   0    2     91
2   0  50   2    2     54
3  59  17   3   24    103
```

In [23]:
```
!./kMeans_runner.py B > Bdata.txt
generate_summary()
```

```
      0   1  2  3  Total
0   727  25  0  0    752
1     2  89  0  0     91
2     3  51  0  0     54
3    77  26  0  0    103
```

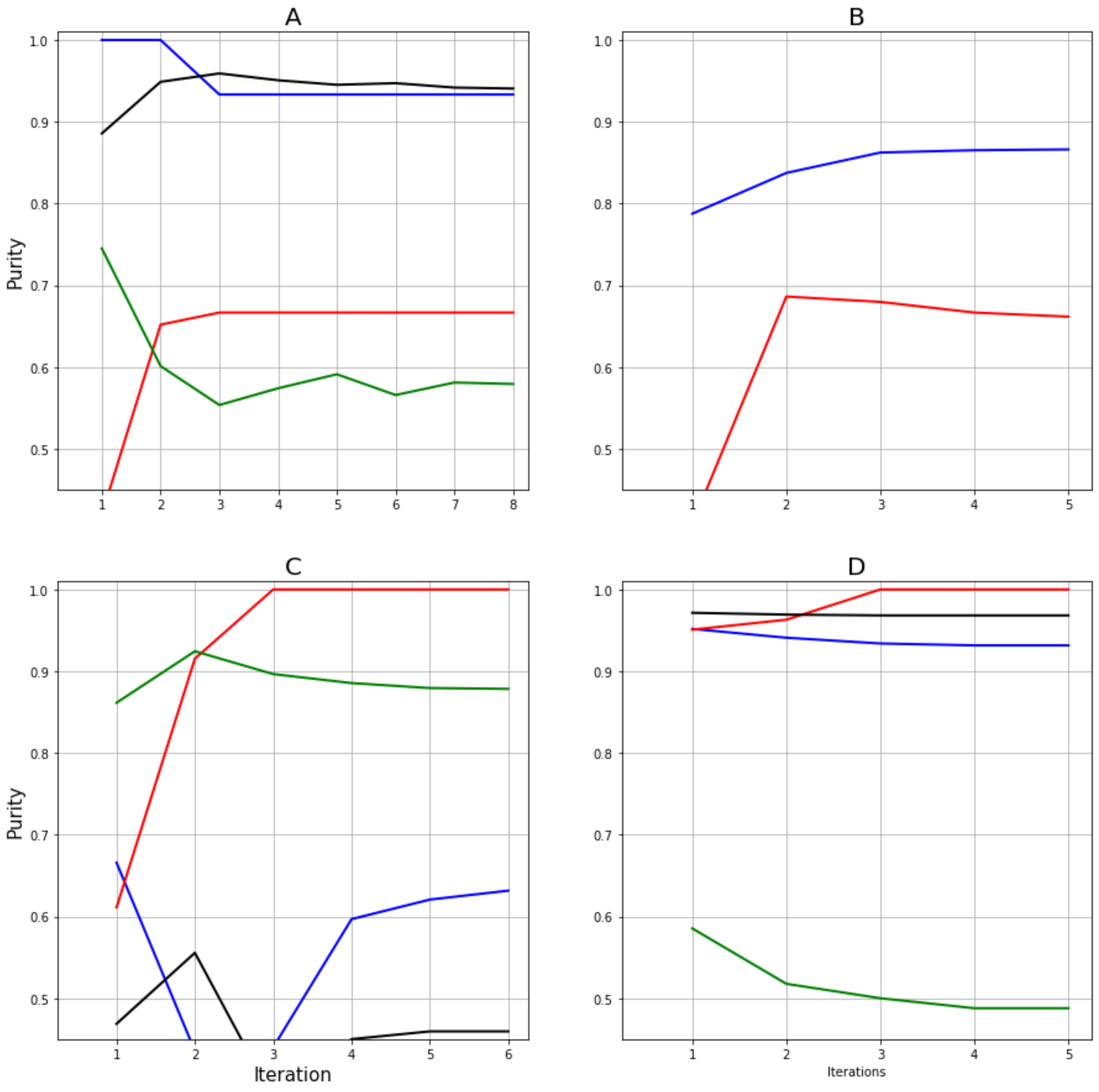
In [24]:
```
!./kMeans_runner.py C > Cdata.txt
generate_summary()
```

```
    0   1    2   3  Total
0   2   0  691  59    752
1   0  51    0  40     91
2  15   0    1  38     54
3   1   0   52  50    103
```

In [25]:
```
!./kMeans_runner.py D > Ddata.txt
generate_summary()
```

```
      0   1   2   3  Total
0   682   0  54  16    752
1     0  51  40   0     91
2     3   3  48   0     54
3    10   0  20  73    103
```

In [26]:
```
from matplotlib import pyplot as plot
import numpy as np
from collections import defaultdict
import re
%matplotlib inline


k = 4
plot.figure(figsize=(15, 15))

## function loads data from any of the 4 initializations
def loadData(filename):
    purities = defaultdict(list)
    #iterations = [i+1 for i in range(1000)]
    f = open(filename, 'r')
    for line in f:
        line  = line.strip()
        data = re.split(",",line)
        #iterations.append(int(data[0]))
        i = 0
        for i in range(len(data)):
            if i:

                purities[i].append(float(data[i]))
    return purities

## load purities for initialization A
purities = defaultdict(list)
purities = loadData("Adata.txt")
iterations = [i+1 for i in range(len(purities[1]))]


## plot purities for initialization A
plot.subplot(2,2,1)
plot.axis([0.25, max(iterations)+0.25,0.45, 1.01])
plot.plot(iterations,purities[1],'b',iterations,purities[2],'r',iterations,puri
ties[3],'g',iterations,purities[4],'black',lw=2)
plot.ylabel('Purity',fontsize=15)
plot.title("A",fontsize=20)
plot.grid(True)

## load purities for initialization B
purities = defaultdict(list)
purities = loadData("Bdata.txt")
iterations = [i+1 for i in range(len(purities[1]))]

## plot purities for initialization B
plot.subplot(2,2,2)
plot.axis([0.25, max(iterations)+0.25,0.45, 1.01])
plot.plot(iterations,purities[1],'b',iterations,purities[2],'r',lw=2)
plot.title("B",fontsize=20)
plot.grid(True)

## load purities for initialization C
purities = defaultdict(list)
purities = loadData("Cdata.txt")
iterations = [i+1 for i in range(len(purities[1]))]

## plot purities for initialization C
plot.subplot(2,2,3)
plot.axis([0.25, max(iterations)+0.25,0.45, 1.01])
plot.plot(iterations,purities[1],'b',iterations,purities[2],'r',iterations,puri
ties[3],'g',iterations,purities[4],'black',lw=2)
plot.xlabel('Iteration',fontsize=15)
plot.ylabel('Purity',fontsize=15)
plot.title("C",fontsize=20)
plot.grid(True)

## load purities for initialization D
```


A
B
C
D
Purity
Purity
Iteration
Iterations
1.0
0.9
0.8
0.7
0.6
0.5
1
2
3
4
5
6
7
8

K-Means Findings

```
D performes better than rest of them.
Also local minima is different for every  time we run A,B and C. this is because ou
tput depeneds on the initialization factor. A,B and C are initilized randomly so ou
tput is different for each time.

Lets try to explain why such unsatisfactory results.
```

K-Means assumptions

The prior probability for all k clusters are the same, i.e. each cluster has roughly equal number of observations.

```
 If we look at the composition plots we can see there are 752 human and very few cy
borgs/Robots. So this assumption is not holding with current dataset.
```

k-means assume the variance of the distribution of each attribute (variable) is spherical;

```
There are some words which have high frequency on robots and same is true for Cybor
gs. this causes different variance for all classes . hence we are again voilating t
his assumption.

these voilations are causing unsatisfactory results.  Even for Part- D where result
s are better for Class-2 category composition is
_____|_0___1_____2____3____Total
|1   | 0|  51|  40 |  0 |    91

So among 91 examples almost 40 have been misclassified i.e. error rate > 40%
```

HW4.6 (OPTIONAL) Scaleable K-MEANS++

Back to Table of Contents

Over half a century old and showing no signs of aging, k-means remains one of the most popular data processing algorithms. As is well-known, a proper initialization of k-means is crucial for obtaining a good final solution. The recently proposed k-means++ initialization algorithm achieves this, obtaining an initial set of centers that is provably close to the optimum solution. A major downside of the k-means++ is its inherent sequential nature, which limits its applicability to massive data: one must make k passes over the data to find a good initial set of centers. The paper listed below shows how to drastically reduce the number of passes needed to obtain, in parallel, a good initialization. This is unlike prevailing efforts on parallelizing k-means that have mostly focused on the post-initialization phases of k-means. The proposed initialization algorithm k-means|| obtains a nearly optimal solution after a logarithmic number of passes; the paper also shows that in practice a constant number of passes suffices. Experimental evaluation on realworld large-scale data demonstrates that k-means|| outperforms k-means++ in both sequential and parallel settings.

Read the following paper entitled "Scaleable K-MEANS++" located at:

http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf (http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf)

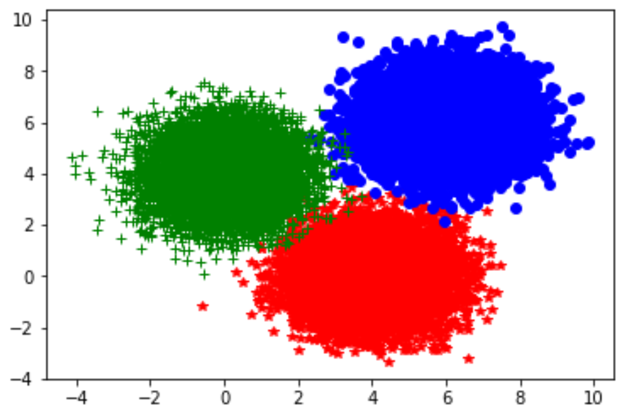
In MrJob, implement K-MEANS|| and compare with a random initializtion when used in conjunction with the kmeans algorithm as an initialization step for the 2D dataset generated using code in the following notebook:

https://www.dropbox.com/s/lbzwmyv0d8rocfq/MrJobKmeans.ipynb?dl=0 (https://www.dropbox.com/s/lbzwmyv0d8rocfq/MrJobKmeans.ipynb?dl=0)

Plot the initialation centroids and the centroid trajectory as the K-MEANS|| algorithms iterates. Repeat this for a random initalization (i.e., pick a training vector at random for each inital centroid) of the kmeans algorithm. Comment on the trajectories of both algorithms. Report on the number passes over the training data, and time required to run both clustering algorithms. Also report the rand index score for both algorithms and comment on your findings.

Random initilization of centroids causes local minima as well as more number of iterations to come up with good clusters. The k-means++ algorithm addresses the second of these obstacles by specifying a procedure to initialize the cluster centers before proceeding with the standard k-means optimization iterations. So to implement this with toy dataset , I have done changes only in initilization section in runner class. I have explained algorithm in that section.

In [27]:
```
%matplotlib inline
import numpy as np
import pylab
size1 = size2 = size3 = 10000
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('Kmeandata.csv',data,delimiter = ",")
pylab.plot(samples1[:, 0], samples1[:, 1],'*', color = 'red')
pylab.plot(samples2[:, 0], samples2[:, 1],'o',color = 'blue')
pylab.plot(samples3[:, 0], samples3[:, 1],'+',color = 'green')
pylab.show()
```



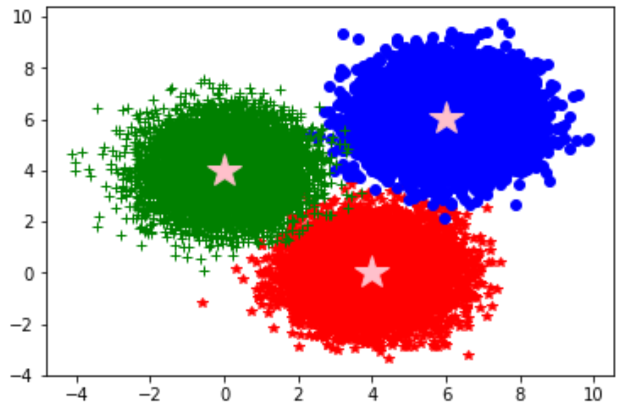
In [28]:
```
%%writefile Kmeansplusplus.py
from numpy import argmin, array, random
from mrjob.job import MRJob
from mrjob.step import MRStep
from itertools import chain
import os
import numpy as np

#Calculate find the nearest centroid for data point
def MinDist(datapoint, centroid_points):
    datapoint = array(datapoint)
    centroid_points = array(centroid_points)
    diff = datapoint - centroid_points
    diffsq = diff*diff
    # Get the nearest centroid for each instance
    minidx = argmin(list(diffsq.sum(axis = 1)))
    return minidx

#Check whether centroids converge
def stop_criterion(centroid_points_old, centroid_points_new,T):
    oldvalue = list(chain(*centroid_points_old))
    newvalue = list(chain(*centroid_points_new))
    Diff = [abs(x-y) for x, y in zip(oldvalue, newvalue)]
    Flag = True
    for i in Diff:
        if(i>T):
            Flag = False
            break
    return Flag

class MRKmeans(MRJob):
    centroid_points=[]
    k=3
    def steps(self):
        return [
            MRStep(mapper_init = self.mapper_init, mapper=self.mapper,combiner
= self.combiner,reducer=self.reducer)
               ]
    #load centroids info from file
    def mapper_init(self):
        #print "Current path:", os.path.dirname(os.path.realpath(__file__))
        print "Current path:", os.path.dirname(os.path.realpath(__file__))

        self.centroid_points = [map(float,s.split('\n')[0].split(',')) for s in
open("Centroids.txt").readlines()]
        #open('Centroids.txt', 'w').close()

        print "Centroids: ", self.centroid_points

    #load data and output the nearest centroid index and data point
    def mapper(self, _, line):
        D = (map(float,line.split(',')))
        yield int(MinDist(D,self.centroid_points)), (D[0],D[1],1)
    #Combine sum of data points locally
    def combiner(self, idx, inputdata):
        sumx = sumy = num = 0
        for x,y,n in inputdata:
            num = num + n
            sumx = sumx + x
            sumy = sumy + y
        yield idx,(sumx,sumy,num)
    #Aggregate sum for each cluster and then calculate the new centroids
    def reducer(self, idx, inputdata):
        centroids = []
        num = [0]*self.k
        for i in range(self.k):
            centroids.append([0,0])
        for x, y, n in inputdata:
            num[idx] = num[idx] + n
```

```
Overwriting Kmeansplusplus.py
```

In [29]:
```
%reload_ext autoreload
%autoreload 2
from numpy import random
from Kmeansplusplus import MRKmeans, stop_criterion
mr_job = MRKmeans(args=['Kmeandata.csv', '--file=Centroids.txt'])

#Geneate initial centroids
centroid_points = []
k = 3
for i in range(k):
    centroid_points.append([random.uniform(-3,3),random.uniform(-3,3)])
with open('Centroids.txt', 'w+') as f:
        f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_point
s)

# Update centroids iteratively
i = 0
while(1):
    # save previous centoids to check convergency
    centroid_points_old = centroid_points[:]
    print "iteration"+str(i)+":"
    with mr_job.make_runner() as runner:
        runner.run()
        # stream_output: get access of the output
        for line in runner.stream_output():
            key,value =  mr_job.parse_output_line(line)
            print key, value
            centroid_points[key] = value

        # Update the centroids for the next iteration
        with open('Centroids.txt', 'w') as f:
            f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_p
oints)

    print "\n"
    i = i + 1
    if(stop_criterion(centroid_points_old,centroid_points,0.01)):
        break
print "Centroids\n"
print centroid_points
```

```
iteration0:
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[-1.02930834725, -0.543001559131], [-1.34784112485, -1.5138067332
6], [1.1572266911, -0.515538079839]]
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[-1.02930834725, -0.543001559131], [-1.34784112485, -1.5138067332
6], [1.1572266911, -0.515538079839]]
0 [-0.7786595645000001, 3.9720601882]
2 [4.1607818047, 3.2163990547]


iteration1:
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[-0.7786595645, 3.9720601882], [-1.34784112485, -1.51380673326],
[4.1607818047, 3.2163990547]]
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[-0.7786595645, 3.9720601882], [-1.34784112485, -1.51380673326],
[4.1607818047, 3.2163990547]]
0 [-0.0855876861, 4.0158778246]
1 [2.5164964204, -1.1789655072]
2 [5.0288124423, 3.192423839]


iteration2:
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[-0.0855876861, 4.0158778246], [2.5164964204, -1.1789655072], [5.
0288124423, 3.192423839]]
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[-0.0855876861, 4.0158778246], [2.5164964204, -1.1789655072], [5.
0288124423, 3.192423839]]
0 [-0.006653317000000001, 4.0041957726]
1 [3.8016179099, -0.3209583304]
2 [5.7419557947, 5.1113427422]


iteration3:
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[-0.006653317, 4.0041957726], [3.8016179099, -0.3209583304], [5.7
419557947, 5.1113427422]]
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[-0.006653317, 4.0041957726], [3.8016179099, -0.3209583304], [5.7
419557947, 5.1113427422]]
0 [0.0027740085000000003, 3.9973312542]
1 [3.9941686180000002, -0.0048563036]
2 [5.9886787709, 5.9923377636]


iteration4:
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[0.0027740085, 3.9973312542], [3.994168618, -0.0048563036], [5.98
86787709, 5.9923377636]]
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[0.0027740085, 3.9973312542], [3.994168618, -0.0048563036], [5.98
86787709, 5.9923377636]]
0 [0.0049084433, 3.9984271267000002]
1 [4.0004683637, 0.012788272100000001]
2 [5.9976016937, 6.0161877837]


iteration5:
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[0.0049084433, 3.9984271267], [4.0004683637, 0.0127882721], [5.99
76016937, 6.0161877837]]
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[0.0049084433, 3.9984271267], [4.0004683637, 0.0127882721], [5.99
76016937, 6.0161877837]]
0 [0.0049084433, 3.9984271267000002]
1 [4.0004683637, 0.012788272100000001]
2 [5.9976016937, 6.0161877837]
```

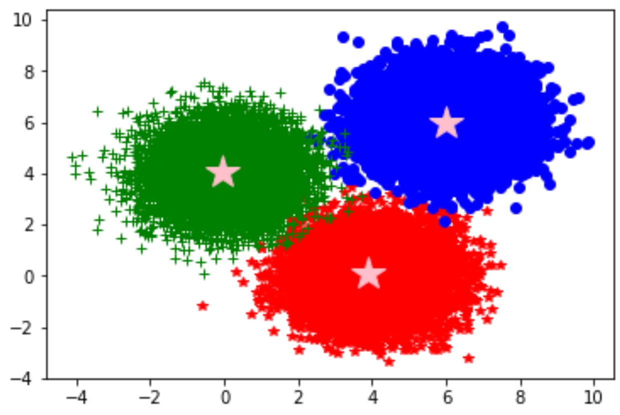
In [30]:
```
pylab.plot(samples1[:, 0], samples1[:, 1],'*', color = 'red')
pylab.plot(samples2[:, 0], samples2[:, 1],'o',color = 'blue')
pylab.plot(samples3[:, 0], samples3[:, 1],'+',color = 'green')
for point in centroid_points:
    pylab.plot(point[0], point[1], '*',color='pink',markersize=20)
pylab.show()
```



In [31]:
```
%reload_ext autoreload
%autoreload 2
from numpy import random
from Kmeansplusplus import MRKmeans, stop_criterion
mr_job = MRKmeans(args=['Kmeandata.csv', '--file=Centroids.txt'])

#Geneate initial centroids
centroid_points = []
k = 3
######Initialization method for smart K++ initiatilization
#pseudo Code
#1a. Take one center c1, chosen uniformly at random from X .
#1b. Take a new center ci choosing x ∈ X with probability proportal to the dist
ance from first selection.
#1c. Repeat Step 1b. until we have taken k centers altogether.

data = np.loadtxt('Kmeandata.csv',delimiter = ",")
#random first selection
centroids = data[np.random.choice(range(data.shape[0]),1), :]
data_ex = data[:, np.newaxis, :]

# run k - 1 passes through the data set to select the initial rest of the centr
oids.
while centroids.shape[0] < 3 : #as K=3 for toy data
            #print (centroids)
            euclidean_dist = (data_ex - centroids) ** 2
            distance_arr = np.sum(euclidean_dist, axis=2)
            min_location = np.zeros(distance_arr.shape)
            min_location[range(distance_arr.shape[0]), np.argmin(distance_arr,
axis=1)] = 1
            # calculate J
            j_val = np.sum(distance_arr[min_location == True])
            # calculate the probability distribution
            prob_dist = np.min(distance_arr, axis=1)/j_val
            # select the next centroid using the probability distribution calcu
lated before
            centroids = np.vstack([centroids, data[np.random.choice(range(data.
shape[0]),1, p = prob_dist), :]])
centroid_points = centroids

#for i in range(k):
#    centroid_points.append([random.uniform(-3,3),random.uniform(-3,3)])
with open('Centroids.txt', 'w+') as f:
        f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_point
s)

# Update centroids iteratively
i = 0
while(1):
    # save previous centoids to check convergency
    centroid_points_old = centroid_points[:]
    print "iteration"+str(i)+":"
    with mr_job.make_runner() as runner:
        runner.run()
        # stream_output: get access of the output
        for line in runner.stream_output():
            key,value =  mr_job.parse_output_line(line)
            print key, value
            centroid_points[key] = value

        # Update the centroids for the next iteration
        with open('Centroids.txt', 'w') as f:
            f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_p
oints)

    print "\n"
    i = i + 1
    if(stop_criterion(centroid_points_old,centroid_points,0.01)):
        break
```

```
iteration0:
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[2.76430655035, 0.153844264159], [5.98732444339, 5.60060237499],
[-1.14601330174, 4.43183521684]]
Current path: /media/notebooks/DataScience/W261/homework/new/HW4
Centroids:  [[2.76430655035, 0.153844264159], [5.98732444339, 5.60060237499],
[-1.14601330174, 4.43183521684]]
0 [3.9042106899, 0.0827272887]
1 [5.9843050938, 5.9949797942]
2 [-0.057092259900000004, 4.0561702107]


Centroids

[[ 3.90421069  0.08272729]
 [ 5.98430509  5.99497979]
 [-0.05709226  4.05617021]]
```

In [32]:
```
pylab.plot(samples1[:, 0], samples1[:, 1],'*', color = 'red')
pylab.plot(samples2[:, 0], samples2[:, 1],'o',color = 'blue')
pylab.plot(samples3[:, 0], samples3[:, 1],'+',color = 'green')
for point in centroid_points:
    pylab.plot(point[0], point[1], '*',color='pink',markersize=20)
pylab.show()
```



This is really great as centroid are selected as part of first iteration only. So performance wise its great as it finds minima with minimal iterations.

4.6.1 (OPTIONAL) Apply K-MEANS||

Back to Table of Contents

Apply your implementation of K-MEANS|| to the dataset in HW 4.5 and compare to the a random initalization (i.e., pick a training vector at random for each inital centroid)of the kmeans algorithm. Report on the number passes over the training data, and time required to run all clustering algorithms. Also report the rand index score for both algorithms and comment on your findings.

In [33]:
```
%%writefile Kmeansparallel.py
#!/usr/bin/env python
#START STUDENT CODE46
from numpy import argmin, array, random
from mrjob.job import MRJob
from mrjob.step import MRStep
from itertools import chain
import os
import re

#Calculate find the nearest centroid for data point
#copied from http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/oppgyfqxphlh69g/
MrJobKmeans_Corrected.ipynb
def MinDist(datapoint, centroid_points):
    datapoint = array(datapoint)
    centroid_points = array(centroid_points)
    diff = datapoint - centroid_points
    diffsq = diff*diff
    # Get the nearest centroid for each instance
    minidx = argmin(list(diffsq.sum(axis = 1)))
    return minidx

#Check whether centroids converge

#copied from http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/oppgyfqxphlh69g/
MrJobKmeans_Corrected.ipynb
def stop_criterion(centroid_points_old, centroid_points_new,T):
    oldvalue = list(chain(*centroid_points_old))
    newvalue = list(chain(*centroid_points_new))
    Diff = [abs(x-y) for x, y in zip(oldvalue, newvalue)]
    # if centroids are not changing enough
    Flag = True
    for i in Diff:
        if(i>T):
            Flag = False
            break
    return Flag

class MRKmeans(MRJob):
    centroid_points=[]
    k = 4
    n = 1000
    def steps(self):
        return [
            MRStep(mapper_init = self.mapper_init, mapper=self.mapper,combiner
= self.combiner,reducer=self.reducer)
               ]
    #load centroids info from file
    def mapper_init(self):
         self.centroid_points = [map(float,s.split('\n')[0].split(',')) for s i
n open("centroids.txt").readlines()]

    #load data and output the nearest centroid index and data point
    def mapper(self, _, line):
        total = 0
        data = re.split(',',line)
        ID = data[0]
        code = int(data[1])
        users = [ID]
        codes = [0,0,0,0] #one hot representations
        codes[code] = 1
        coords = [float(data[i+3])/float(data[2]) for i in range(1000)]
        for coord in coords:
            total += coord


        #generate key value pair.  Key --> 0-3 class of user and then value is
user/1/cordinates and codes
        yield (int(MinDist(coords,self.centroid_points)),[users,1,coords,codes]
Overwriting Kmeansparallel.py
```

In [34]:
```
%%writefile kmeans_runner_plus.py
#!/usr/bin/env python
#START STUDENT CODE46_RUNNER


#!/usr/bin/env python
from numpy import random
from Kmeansparallel import MRKmeans
import re,sys
import numpy as np
mr_job = MRKmeans(args=["topUsers_Apr-Jul_2014_1000-words.txt","--file","centro
ids.txt"])

thresh = 0.0001

scriptName,part = sys.argv

## only stop when distance is below thresh for all centroids
def stopSignal(k,thresh,newCentroids,oldCentroids):
    stop = 1
    for i in range(k):
        dist = 0
        for j in range(len(newCentroids[i])):
            dist += (newCentroids[i][j] - oldCentroids[i][j]) ** 2
        dist = dist ** 0.5
        if (dist > thresh):
            stop  = 0
            break
    return stop

################################################################################
###
# Use four centroids from the coding
################################################################################
###
def startCentroidsA():
    k = 4
    centroids = []
    for i in range(k):
        rndpoints = random.sample(1000)
        total = sum(rndpoints)
        centroid = [pt/total for pt in rndpoints]
        centroids.append(centroid)
    return centroids
################################################################################
####
def startCentroidsBC(k):
    counter = 0
    for line in open("topUsers_Apr-Jul_2014_1000-words_summaries.txt").readline
s():
        if counter == 1:
            data = re.split(",",line)
            globalAggregate = [float(data[i+3])/float(data[2]) for i in range(1
000)]
        counter += 1
    #perturb the global aggregate for the four initializations
    centroids = []
    for i in range(k):
        rndpoints = random.sample(1000)
        peturpoints = [rndpoints[n]/10+globalAggregate[n] for n in range(1000)]
        centroids.append(peturpoints)
        total = 0
        for j in range(len(centroids[i])):
            total += centroids[i][j]
        for j in range(len(centroids[i])):
            centroids[i][j] = centroids[i][j]/total
    return centroids

################################################################################
```

```
Overwriting kmeans_runner_plus.py
```

In [35]:
```
!chmod a+x kmeans_runner_plus.py
```

In [36]:
```
!./kMeans_runner_plus.py A
```

```
iteration0:
1,0.752
iteration1:
2,0.752
```

As expected only 2 iterations are required to find clusters

HW4.7 (OPTIONAL) Canopy Clustering

Back to Table of Contents

An alternative way to intialize the k-means algorithm is the canopy clustering. The canopy clustering algorithm is an unsupervised pre-clustering algorithm introduced by Andrew McCallum, Kamal Nigam and Lyle Ungar in 2000. It is often used as preprocessing step for the K-means algorithm or the Hierarchical clustering algorithm. It is intended to speed up clustering operations on large data sets, where using another algorithm directly may be impractical due to the size of the data set.

For more details on the Canopy Clustering algorithm see:

https://en.wikipedia.org/wiki/Canopy_clustering_algorithm (https://en.wikipedia.org/wiki/Canopy_clustering_algorithm)

Plot the initialation centroids and the centroid trajectory as the Canopy Clustering based K-MEANS algorithm iterates. Repeat this for a random initalization (i.e., pick a training vector at random for each inital centroid) of the kmeans algorithm. Comment on the trajectories of both algorithms. Report on the number passes over the training data, and time required to run both clustering algorithms. Also report the rand index score for both algorithms and comment on your findings.

In [ ]:

4.7.1 Apply Canopy Clustering based K-MEANS

Back to Table of Contents

Apply your implementation Canopy Clustering based K-MEANS algorithm to the dataset in HW 4.5 and compare to the a random initalization (i.e., pick a training vector at random for each inital centroid)of the kmeans algorithm. Report on the number passes over the training data, and time required to run both clustering algorithms. Also report the rand index score for both algorithms and comment on your findings.

In [ ]:

Back to Table of Contents

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