DATASCI W261: Machine Learning at Scale

David Rose
david.rose@berkeley.edu
W261-1
Week 04
2015.09.23

HW4.0

- **MRJob**: a Python API framework for accessing the Hadoop streaming capabilities. It differs from MapReduce in that it acts as a higher-level interface to MapReduce, yet utilizes the Hadoop MapReduce functionality. It provides a mechanism for creating data processing pipelines that MapReduce, on its own, cannot.
- The ***_final() methods** are defined in the MRJob class, and as such can be overridden by classes that extend MRJob. The ***_final()** methods are executed when the input stream to the respective task is closed.

HW4.1

- **Serialization** converts an object into a bytestream that can be used for transporting the object over a network or to and from disk storage.
- Within Hadoop and MRJob processes data must be **serialized**, at a minimum, when it is first submitted to a map task, when it is spilled to disk, when it is submitted to a reduce task, and again when the results are written to disk.
- **Default modes** for MRJob serialization are:
  - INPUT_PROTOCOL = mrjob.protocol.RawValueProtocol
  - INTERNAL_PROTOCOL = mrjob.protocol.JSONProtocol
  - OUTPUT_PROTOCOL = mrjob.protocol.JSONProtocol

HW4.2

In [45]:
```
%%writefile hw_4_2_flatten.py
#!/usr/bin/python
'''
'''
from __future__ import print_function
import csv
import sys
with open(sys.argv[1], 'rb') as fin, open(sys.argv[2], 'w') as fou
    csvreader = csv.reader(fin, delimiter = ',', quotechar = '"')
    visitorid = ''
    for line in csvreader:
        if line[0] == 'C':
            visitorid = line[2]
            continue
        if line[0] == 'V':
            line.append(visitorid)
        print(','.join(line), file=fout)
```

```
Overwriting hw_4_2_flatten.py
```

In [46]:
```
# combine the page id and visitor id onto a single line for
# subsequent processing
!python hw_4_2_flatten.py anonymous-msweb.data flattened.data
```

HW4.3

In [47]:

```
%%writefile hw_4_3_mrjob.py
''' count and list the top five most frequently visited pages
'''
from __future__ import print_function
from mrjob.job import MRJob
from mrjob.step import MRStep
import sys

class FrequentPages(MRJob):

    def steps(self):
        return [
            MRStep(mapper=self.mapper,
                   combiner=self.combiner,
                   reducer=self.reducer,
                   reducer_final=self.reducer_final)
        ]

    def mapper(self, _, line):
        ''' enumerate page visits
        '''
        row = line.split(',')
        if row[0] == 'V':
            yield row[1], 1

    def combiner(self, page, i):
        ''' combine local results
        '''
        yield page, sum(i)

    # track top five frequent pages in sorted order
    topfive = [['',0]]

    def inserttopfive(self, page, total):
        ''' data structure and logic to maintain list of
            top five most frequent pages.
            This operation is performed here in the reducer
            and again in the driver to capture output from
            multiple reducers
        '''
        for j in range(0, len(self.topfive)):
            if total > self.topfive[j][1]:
                self.topfive.insert(j, (page, total))
                if len(self.topfive) > 5:
                    self.topfive.pop()
                break

    def reducer(self, page, i):
        ''' sum and sort page counts
        '''
        total = sum(i)
        self.inserttopfive(page, total)

    def reducer_final(self):
```

```
        ''' emit results
        '''
        for i in range(0, len(self.topfive)):
            yield(self.topfive[i][0], self.topfive[i][1])

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

```

Overwriting hw_4_3_mrjob.py

In [48]:

```
%%writefile hw_4_3_driver.py
from __future__ import print_function
from mrjob import util
import sys
from hw_4_3_mrjob import FrequentPages
util.log_to_null() # to suppress a 'no handler found' message

# list for storing most frequent pages
# we do this step here since multiple reducer tasks may run and th
# combined output needs to be processed
topfive = [['',0]]
def inserttopfive(page, total):
    for j in range(0, len(topfive)):
        if total > topfive[j][1]:
            topfive.insert(j, (page, total))
            if len(topfive) > 5:
                topfive.pop()
                break

mr_job = FrequentPages(args=sys.argv[1:])
with mr_job.make_runner() as runner:
    runner.run()
    for line in runner.stream_output():
        page, total = line.split()
        inserttopfive(page, int(total))
for i in range(0, len(topfive)):
    print('page: {}, visits: {}'.format(topfive[i][0],
                                        topfive[i][1]), file=sys.s

```

Overwriting hw_4_3_driver.py

In [49]:

```
# HW 4.3: Find the 5 most frequently visited pages using mrjob fro
!python hw_4_3_driver.py flattened.data --strict-protocols -r loca
```

```
page: "1008", visits: 10836
page: "1034", visits: 9383
page: "1004", visits: 8463
page: "1018", visits: 5330
page: "1017", visits: 5108
```

HW4.4

In [50]:

```
%%writefile hw_4_4_mrjob.py
''' count the number of page-visitor combinations and
    for each page list the most frequent visitors

    the data does not effectively support this operation since it
    only lists unique page visits, therefore every visitor
    will show up as having visited once, and therefore every visit
    is the most frequent visitor
'''
from __future__ import print_function
from mrjob.job import MRJob
import sys

class FrequentVisitors(MRJob):

    def mapper(self, _, line):
        ''' enumerate page visitors
        '''
        row = line.split(',')
        if row[0] == 'V':
            # page ID, visitor ID
            yield row[1], row[3]

    # data structures to manage reducer logic
    visitors = {}
    currentpage = ''

    def reducer(self, page, visitor):
        ''' sum page visitor counts
        '''
        if not page == self.currentpage:
            ''' page id has changed in the stream, so process and
                the information for the current page
            '''
            if len(self.visitors) > 0:
                frequentv = []
                maxv = 0
                for v in self.visitors:
                    if self.visitors[v] > maxv:
                        frequentv = [v]
                        maxv = self.visitors[v]
                    elif self.visitors[v] == maxv:
                        frequentv.append(v)
                # emit results
                for v in frequentv:
                    yield self.currentpage, v
            # reset counters
            self.visitors = {}
            self.currentpage = page
        for v in visitor:
            if not v in self.visitors:
                self.visitors[v] = 0
            self.visitors[v] += 1

```

```
    # process any remaining values after stream closes
    def reducer_final(self):
        if len(self.visitors) > 0:
            frequentv = []
            maxv = 0
            for v in self.visitors:
                if self.visitors[v] > maxv:
                    frequentv = [v]
                elif self.visitors[v] == maxv:
                    frequentv.append(v)
            for v in frequentv:
                yield self.currentpage, v

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

```

```
Overwriting hw_4_4_mrjob.py
```

In [51]:

```
%%writefile hw_4_4_driver.py
import csv
from mrjob import util
import sys
from hw_4_4_mrjob import FrequentVisitors

util.log_to_null() # to suppress a 'no handler found' message

# construct list of page ids and urls to satisfy the requirement
# this is essentially a join; it makes better sense to do this in
# the driver as doing so in hadoop offers no advantages and incurs
# additional network overhead; mrjob output is parsed and augmente
# the url information
pageinfo = {}
# read in the page attributes including the url; this could also h
# been preprocessed so that the page attributes were in their own
with open('flattened.data', 'rb') as fin:
    csvreader = csv.reader(fin, delimiter = ',', quotechar = '"')
    for line in csvreader:
        if line[0] == 'A':
            pageid = line[1]
            url = line[4]
            pageinfo[pageid] = url

mr_job = FrequentVisitors(args=sys.argv[1:])
with mr_job.make_runner() as runner:
    runner.run()
    for line in runner.stream_output():
        page, visitor = line.replace('"', '').split()
        print pageinfo[page], page, visitor

```

```
Overwriting hw_4_4_driver.py
```

In [52]:

```
# HW 4.4: Find the most frequent visitor of each page using
# mrjob and the output of 4.2
# In this output please include the webpage URL, webpageID
# and Visitor ID.
!python hw_4_4_driver.py flattened.data --strict-protocols -r loca
!head -n50 mrjob_4_4_output
!tail -n50 mrjob_4_4_output
```

```
/regwiz 1000 10001
/regwiz 1000 10010
/regwiz 1000 10039
/regwiz 1000 10073
/regwiz 1000 10087
/regwiz 1000 10101
/regwiz 1000 10132
/regwiz 1000 10141
/regwiz 1000 10154
/regwiz 1000 10162
/regwiz 1000 10166
/regwiz 1000 10201
/regwiz 1000 10218
/regwiz 1000 10220
/regwiz 1000 10324
/regwiz 1000 10348
/regwiz 1000 10376
/regwiz 1000 10384
/regwiz 1000 10409
```

HW4.5

In [53]:

```
''' utility script to get some statistics on the data set
'''
with open('topUsers_Apr-Jul_2014_1000-words.txt', 'rb') as fin:
    max = 0
    total = 0
    linecount = 0
    classes = {}
    for line in fin:
        row = line.split(',')
        count = int(row[2])
        clazz = row[1]
        if count > max: max = count
        total += count
        linecount += 1
        if not clazz in classes:
            classes[clazz] = 0
        classes[clazz] += 1
print(linecount, max, total, str(classes))

```

```
(1000, 1724608, 61819567, "{'1': 91, '0': 752, '3': 103, '2': 54}")
```

In [54]:

```
''' utility script to get some statistics on the data set
    look for the maximum word ratio after the data is normalized
'''
with open('topUsers_Apr-Jul_2014_1000-words.txt', 'rb') as fin:
    max = 0.0
    for line in fin:
        row = line.split(',')
        count = int(row[2])
        for i in range(3, len(row)):
            p = float(row[i]) / count
            if p > max:
                max = p
print(max)
```

```
0.409909665665
```

In [55]:

```
%%writefile hw_4_5_preprocess.py
''' preprocess the data set, normalizing the word counts
'''
from __future__ import print_function
import sys
with open(sys.argv[1], 'rb') as fin, open(sys.argv[2], 'w') as fou
    for line in fin:
        row = map(int, line.split(','))
        userid = row[0]
        code = row[1]
        total = row[2]
        for j in range(2, len(row)):
            row[j] = float(row[j]) / total
        print('{}'.format(row[0]), file = fout, end = '')
        for j in range(1, 3):
            print(',{}'.format(row[j]), file = fout, end = '')
        for j in range(3, len(row)):
            print(',{:0.8f}'.format(row[j]), file = fout, end = ''
        print('', file = fout)

```

```
Overwriting hw_4_5_preprocess.py
```

In [56]:

```
!python hw_4_5_preprocess.py topUsers_Apr-Jul_2014_1000-words.txt
```

In [57]:

```
%%writefile hw_4_5_mrjob.py
''' map/reduce approach to determining stable centroids using
    a K-means algorithm
'''
from __future__ import print_function
from numpy import argmin, array, random
from mrjob.job import MRJob
from mrjob.step import MRStep
from itertools import chain
import sys

# 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
    sumofsquares = list(diffsq.sum(axis = 1))
    minindex = argmin(sumofsquares)
    return minindex

# 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 = -1
    CENTROIDFILE = '/tmp/centroids.txt'

    def steps(self):
        return [
            MRStep(mapper_init = self.mapper_init,
                   mapper=self.mapper,
                   combiner = self.combiner,
                   reducer=self.reducer)
               ]
    # load initial centroids
    def mapper_init(self):
        self.centroid_points=[]
        with open(self.CENTROIDFILE, 'rb') as fin:
            header = True
            for line in fin:
                if header:
```

```
                    self.k = int(line.strip())
                    header = False
                else:
                    self.centroid_points.append(map(float,line.str
        # print the value of k back to the cache file
        with open(self.CENTROIDFILE, 'w') as fout:
            print('{}'.format(self.k), file = fout)

    #load data and output the nearest centroid index and data poi
    def mapper(self, _, line):
        D = (map(float,line.split(',')[3:]))
        centroid = MinDist(D, self.centroid_points)
        yield centroid, (D, 1)

    # aggregate data points locally
    def combiner(self, centroid, inputdata):
        count = 0
        bucket = [0] * 1000
        for data, n in inputdata:
            count += n
            data = map(float, data)
            for j in range(0, len(data)):
                bucket[j] += data[j]
        yield centroid, (bucket, count)


    # aggregate values for each centroid, then recalculate centro
    def reducer(self, idx, inputdata):
        centroids = []
        with open(self.CENTROIDFILE, 'rb') as fin:
            self.k = int(fin.readline().strip())
        num = [0] * self.k
        for i in range(self.k):
            centroids.append([0.0]*1000)
        for data, n in inputdata:
            num[idx] += n
            data = map(float, data)
            for j in range(0, len(data)):
                centroids[idx][j] += data[j]
        for j in range(0, len(centroids[idx])):
            centroids[idx][j] = centroids[idx][j] / num[idx]
        with open(self.CENTROIDFILE, 'a') as fout:
            print(','.join(str(i) for i in centroids[idx]), file =
        yield idx, (centroids[idx])

if __name__ == '__main__':
    MRKmeans.run()
```

```
Overwriting hw_4_5_mrjob.py
```

In [58]:

```
%%writefile hw_4_5_mrjob_labeler.py
''' assigns centroid labels to data points

    this step is done separately from the centroid calculations
    to simplify the data processing
'''
from __future__ import print_function
from numpy import argmin, array, random
from mrjob.job import MRJob
from mrjob.step import MRStep
import sys

from hw_4_5_mrjob import MinDist

class MRLabeler(MRJob):

    centroid_points=[]
    k = -1
    CENTROIDFILE = '/tmp/centroids.txt'

    def steps(self):
        return [
            MRStep(mapper_init = self.mapper_init,
                   mapper=self.mapper,
                   combiner = self.combiner,
                   reducer=self.reducer,
                   reducer_final=self.reducer_final)
                ]
    #load centroids info from file
    def mapper_init(self):
        self.centroid_points=[]
        with open(self.CENTROIDFILE, 'rb') as fin:
            header = True
            for line in fin:
                if header:
                    self.k = int(line.strip())
                    header = False
                else:
                    self.centroid_points.append(map(float,line.str

    # determine the closest centroid for each data point
    def mapper(self, _, line):
        row = line.strip().split(',')
        label = int(row[1])
        D = (map(float,row[3:]))
        centroid = MinDist(D, self.centroid_points)
        yield centroid, (label, 1)

    # combine sum of data points locally
    def combiner(self, centroid, inputdata):
        counts = [0, 0, 0, 0]
        for label, n in inputdata:
            counts[label] += n
        for i in range(len(counts)):
```

```
            yield centroid, (i, counts[i])


    # sum the counts for centroids and classes
    currentcentroid = ''
    counts = []
    def reducer(self, centroid, inputdata):
        if not centroid == self.currentcentroid:
            for i in range(len(self.counts)):
                yield self.currentcentroid, (i, self.counts[i])
            self.counts = [0, 0, 0, 0]
            self.currentcentroid = centroid
        for label, n in inputdata:
            self.counts[label] += n

    def reducer_final(self):
        for i in range(len(self.counts)):
            yield self.currentcentroid, (i, self.counts[i])


if __name__ == '__main__':
    MRKmeans.run()
```

```
Overwriting hw_4_5_mrjob_labeler.py
```

In [59]:

```
%%writefile hw_4_5_driver.py
''' driver script for Kmeans job
'''
from __future__ import print_function
from numpy import random
from hw_4_5_mrjob import MRKmeans, stop_criterion
from hw_4_5_mrjob_labeler import MRLabeler
from mrjob import util
import sys

util.log_to_null() # to suppress a 'no handler found' message
CENTROIDFILE = '/tmp/centroids.txt'

def countlabels():
    ''' count the number of each classification as labeled in the
        original data set
    '''
    with open('topUsers_Apr-Jul_2014_1000-words.txt', 'rb') as f:
        labels = {}
        for line in f:
            row = line.split(',')
            label = row[1]
            if not label in labels:
                labels[label] = 0
            labels[label] += 1
    return labels

def init_centroids_random_internal(k):
    ''' select initial centroids by choosing data points
        randomly from the data set
    '''
    randoms = sorted(random.randint(0, 1000, size = k))
    centroids = []
    with open('topUsers_Apr-Jul_2014_1000-words.txt', 'rb') as f:
        lineno = 0
        count = 0
        for line in f:
            if lineno in randoms:
                row = line.strip().split(',')
                data = map(float, row[3:])
                # normalize the values
                data = [i / float(row[2]) for i in data]
                centroids.append(data)
                count += 1
                if count == k:
                    break
            lineno += 1
    with open(CENTROIDFILE, 'w') as f:
        print('{}'.format(k), file = f)
        for tuple in centroids:
            print(','.join(str(i) for i in tuple), file = f)
    return centroids

def init_centroids_random_external(k):
```

```
    ''' create initial centroids by generating random values
    '''
    centroids = []
    for i in range(k):
        centroid = []
        for j in range(1000):
            centroid.append(random.uniform(0.0, .5))
        centroids.append(centroid)
    with open(CENTROIDFILE, 'w') as fout:
        print('{}'.format(k), file = fout)
        for tuple in centroids:
            print(','.join(str(i) for i in tuple), file = fout)
    return centroids

def init_centroids_perturbed(k):
    ''' create initial centroids by using aggregated values and
        perturbing them with random noise
    '''
    centroids = []
    with open('topUsers_Apr-Jul_2014_1000-words_summaries.txt', 'r
        for line in f:
            row = line.strip().split(',')
            if row[0] == 'ALL_CODES':
                data = map(float, row[3:])
                # normalize the values
                data = [i / float(row[2]) for i in data]
                for i in range(k):
                    centroid = []
                    for j in range(len(data)):
                        # modify each value with random number in
                        # range of +- the value; avoids a small nu
                        # by a large number
                        centroid.append(data[j] + random.uniform(-
                    centroids.append(centroid)
                break
    with open(CENTROIDFILE, 'w') as f:
        print('{}'.format(k), file = f)
        for tuple in centroids:
            print(','.join(str(i) for i in tuple), file = f)
    return centroids

def init_centroids_trained(k):
    ''' create initial centroids by choosing the class-specific ag
    '''
    centroids = []
    with open('topUsers_Apr-Jul_2014_1000-words_summaries.txt', 'r
        for line in f:
            row = line.strip().split(',')
            if row[0] == 'CODE':
                code = row[1]
                total = int(row[2])
                data = map(float, row[3:])
                # normalize the values
                data = [i / total for i in data]
```

```
109                 centroids.append(data)
110     with open(CENTROIDFILE, 'w') as f:
111         print('{}'.format(k), file = f)
112         for tuple in centroids:
113             print(','.join(str(i) for i in tuple), file = f)
114     return centroids
115 
116 def getpurity(clusters):
117     majority = 0
118     for c in clusters:
119         counts = map(int, clusters[c]);
120         majority += max(counts)
121     return majority / float(1000)
122 
123 def go(centroid_points):
124     ''' submit the centroid calculation job;
125         follow that with the data labeling job
126     '''
127     mr_job = MRKmeans(args = ['normalized.txt'])
128     iteration = 0
129     while(1):
130         # save previous centroids to check convergency
131         centroid_points_old = centroid_points[:]
132         with mr_job.make_runner() as runner:
133             runner.run()
134             # capture reducer output
135             for line in runner.stream_output():
136                 key,value =  mr_job.parse_output_line(line)
137                 centroid_points[key] = value
138         iteration += 1
139         if stop_criterion(centroid_points_old, centroid_points, 0.
140             break
141     print('centroids converged: {} iterations'.format(iteration),
142     mr_job = MRLabeler(args = ['normalized.txt'])
143     labels = countlabels()
144     clusters = {}
145     with mr_job.make_runner() as runner:
146         runner.run()
147         for line in runner.stream_output():
148             centroid, value =  mr_job.parse_output_line(line)
149             label = int(value[0])
150             count = int(value[1])
151             if not centroid in clusters:
152                 clusters[centroid] = [0,0,0,0]
153             clusters[centroid][label] += count
154         for c in sorted(clusters):
155             results = map(float, clusters[c])
156             print('centroid {}: label 0: {:.4f} label 1: {:.4f} la
157                   .format(c, results[0] / labels['0'], results[1]
158                           results[2] / labels['2'], results[3] / l
159                   )
160             # print the actual counts
161 #             print('centroid {}: counts: {}'.format(c, clusters[c
162 #                   file = sys.stderr)
```

```
        print('purity: {:.4f}'.format(getpurity(clusters)))
        print('', file = sys.stderr)

print('k = 4, random initialization, v1', file = sys.stderr)
go(init_centroids_random_internal(4))
print('k = 4, random initialization, v2', file = sys.stderr)
go(init_centroids_random_external(4))
print('k = 2, perturbed initialization', file = sys.stderr)
go(init_centroids_perturbed(2))
print('k = 4, perturbed initialization', file = sys.stderr)
go(init_centroids_perturbed(4))
print('k = 4, trained initialization', file = sys.stderr)
go(init_centroids_trained(4))


```

```
Overwriting hw_4_5_driver.py
```

In [60]:
```
1 !python hw_4_5_driver.py
```

```
k = 4, random initialization, v1
centroids converged: 9 iterations
centroid 0: label 0: 0.0000 label 1: 0.1868 label 2: 0.2778 label 3:
0.0388
centroid 1: label 0: 0.0000 label 1: 0.5604 label 2: 0.0000 label 3:
0.0000
centroid 2: label 0: 0.9987 label 1: 0.0330 label 2: 0.0370 label 3:
0.9612
centroid 3: label 0: 0.0013 label 1: 0.2198 label 2: 0.6852 label 3:
0.0000
purity: 0.8560

k = 4, random initialization, v2
centroids converged: 3 iterations
centroid 0: label 0: 1.0000 label 1: 1.0000 label 2: 1.0000 label 3:
1.0000
purity: 0.7520

k = 2, perturbed initialization
centroids converged: 4 iterations
centroid 0: label 0: 0.9987 label 1: 0.0330 label 2: 0.2593 label 3:
0.9612
centroid 1: label 0: 0.0013 label 1: 0.9670 label 2: 0.7407 label 3:
0.0388
purity: 0.8390

k = 4, perturbed initialization
centroids converged: 4 iterations
centroid 0: label 0: 0.0000 label 1: 0.0000 label 2: 0.0370 label 3:
0.0000
centroid 1: label 0: 0.8816 label 1: 0.0110 label 2: 0.0000 label 3:
0.6214
centroid 2: label 0: 0.0013 label 1: 0.9670 label 2: 0.7037 label 3:
0.0388
centroid 3: label 0: 0.1170 label 1: 0.0220 label 2: 0.2593 label 3:
0.3398
purity: 0.8410

k = 4, trained initialization
centroids converged: 5 iterations
centroid 0: label 0: 0.9960 label 1: 0.0330 label 2: 0.2593 label 3:
0.3689
centroid 1: label 0: 0.0000 label 1: 0.5604 label 2: 0.0000 label 3:
0.0000
centroid 2: label 0: 0.0013 label 1: 0.4066 label 2: 0.7407 label 3:
0.0388
centroid 3: label 0: 0.0027 label 1: 0.0000 label 2: 0.0000 label 3:
0.5922
purity: 0.9010
```

Discussion: centroid initialization has a significant impact on the results. The purity scores of the five runs range from 0.75 to 0.90. Not surprisingly the best results came from the centroids initialized with the class-specific aggregate data, and the worst results came from using randomized synthetic data. Ideally the results would suggest a diagonal of populated cells with all others being zero. The final scenario above comes closest, but there is still room for improvement.

In [ ]:
```
1
```