Nick Hamlin

W261 Midterm

3/2/2016

Supporting code for Questions 6-8: KL Divergence

In [1]:
```
%%writefile kltext.txt
1.Data Science is an interdisciplinary field about processes and system
2.Machine learning is a subfield of computer science[1] that evolved fr
```

```
Writing kltext.txt
```

In [3]:
```
#Use this to make sure we reload the MrJob code when we make changes
%load_ext autoreload
%autoreload 2
```

In [2]:
```
import numpy as np
np.log(3)
```

Out[2]:
```
1.0986122886681098
```

In [33]:

```
%%writefile kldivergence.py
from __future__ import division
from mrjob.job import MRJob
from mrjob.step import MRStep
import re
import numpy as np

class kldivergence(MRJob):
    def mapper1(self, _, line):
        """create inverted index of letted docs"""
        index = int(line.split('.',1)[0])
        letter_list = re.sub(r"[^A-Za-z]+", '', line).lower()
        count = {}
        for l in letter_list:
            if count.has_key(l):
                count[l] += 1
            else:
                count[l] = 1
        for key in count:
            #Yields letter, (origin doc,% of the doc represented by the
            #yield key, [index, count[key]*1.0/len(letter_list)]

            #THIS VERSION IMPLEMENTS SMOOTHING FOR QUESTION 8
            yield key, [index, (count[key]+1)/(len(letter_list)+24)]

    def reducer1(self, key, values):
        """For each letter, aggregate data from each input line
        Using this inverted index, calculate p*log(p/q) for each letter
        Then emit these results to the second reducer for summation
        """
        letter=key
        print letter #Use this to show unique letters so we can answer
        for doc in values:
            doc_id,letter_prob=doc[:]
            #Split results into elements of P and Q for clarity
            if doc_id==1:
                p_i=letter_prob
            if doc_id==2:
                q_i=letter_prob
        #Once we've loaded the results for both documents, calculate th
        output=p_i*np.log(p_i/q_i)
        yield None,output

    def reducer2(self, key, values):
        kl_sum = 0
        for value in values:
            kl_sum = kl_sum + value
        yield None, kl_sum

    def steps(self):
        return [MRStep(mapper=self.mapper1,
                       reducer=self.reducer1),
                MRStep(reducer=self.reducer2)
               ]
```

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

```
Overwriting kldivergence.py
```

In [35]:
```
from kldivergence import kldivergence
mr_job = kldivergence(args=['kltext.txt','--no-strict-protocols'])
with mr_job.make_runner() as runner:
    runner.run()
    # stream_output: get access of the output
    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

```
a
b
c
d
e
f
g
h
i
k
l
m
n
o
p
r
s
t
u
v
w
x
y
(None, 0.06726997279170038)
```

Supporting code for Questions 10-12: Weighted K-Means

Weight each example as follows using the inverse vector length (Euclidean norm):

weight(X)= 1/||X||,

where ||X|| = SQRT(X.X)= SQRT(X1^2 + X2^2)

Here X is vector made up of X1 and X2.

Z(J) = Sum (all X(I) in cluster J) W(I) * X(I) / Sum (all X(I) in cluster J) W(I).

In [77]:

```python
%%writefile Kmeans.py
from __future__ import division
from numpy import argmin, array, random
from mrjob.job import MRJob
from mrjob.step import MRStep
from math import sqrt
from itertools import chain

#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**2

    distances = (diffsq.sum(axis = 1))**0.5
    # Get the nearest centroid for each instance
    min_idx = argmin(distances)
    return min_idx

#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_init=self.reducer_init,
                reducer=self.reducer)
               ]

    #load centroids info from file
    def mapper_init(self):
        self.centroid_points = [map(float,s.split('\n')[0].split(','))
        #open('Centroids.txt', 'w').close()

    #load data and output the nearest centroid index and data point
    def mapper(self, _, line):
```

```
    def mapper(self, _, line):
        D = (map(float,line.split(',')))
        idx = MinDist(D,self.centroid_points)
        yield int(idx), (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:
            #weight=1/(sqrt(x**2+y**2))
            num = num + n
            sumx = sumx + x#*weight
            sumy = sumy + y#*weight
        yield int(idx),(sumx,sumy,num)

    #load centroids info from file
    def reducer_init(self):
        self.centroid_points = [map(float,s.split('\n')[0].split(','))
        open('Centroids.txt', 'w').close()


    #Aggregate sum for each cluster and then calculate the new centroid
    def reducer(self, idx, inputdata):
        centroids = []
        num = [0]*self.k
        distances = 0
        running_weight_sum=0
        running_weighted_distance_sum=0
        for i in range(self.k):
            centroids.append([0,0])
        for x, y, n in inputdata:
            #Here's where we're adding the weights

            #Calculate distances between x and y coordinates
            #of each point and the previous location of the centroid of
            #its current cluster assignment
            delta_x=self.centroid_points[idx][0]-x
            delta_y=self.centroid_points[idx][1]-y
            weight=1/(sqrt(delta_x**2+delta_y**2))
            running_weight_sum+=weight
            num[idx] = num[idx] + n
            #Weights get applied to each component of the centroid here
            centroids[idx][0] = centroids[idx][0] + x*weight
            centroids[idx][1] = centroids[idx][1] + y*weight
            running_weighted_distance_sum+=sqrt((x*weight)**2+(y*weight


        #For Q10:
        print running_weighted_distance_sum/running_weight_sum

        #make sure we also apply average weights to the denominator her
        #Otherwise, we'll distort our results
        average_weight=running_weight_sum/num[idx]
        centroids[idx][0] = centroids[idx][0]/(num[idx]*average_weight)
```

```
        centroids[idx][1] = centroids[idx][1]/(num[idx]*average_weight)

        #Overwrite old centroid locations with new ones

        with open('Centroids.txt', 'a') as f:
            f.writelines(str(centroids[idx][0]) + ',' + str(centroids[i
        yield idx,(centroids[idx][0],centroids[idx][1])

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

```
Overwriting Kmeans.py
```

In [78]:
```
from numpy import random, array
from Kmeans import MRKmeans, stop_criterion
mr_job = MRKmeans(args=['Kmeandata.csv','--file','Centroids.txt','--no-

#Generate initial centroids
centroid_points = [[0,0],[6,3],[3,6]]
k = 3
with open('Centroids.txt', 'w+') as f:
        f.writelines(','.join(str(j) for j in i) + '\n' for i in centro

# Update centroids iteratively
for i in range(10):
    # save previous centoids to check convergency
    centroid_points_old = centroid_points[:]
    print "iteration"+str(i+1)+":"
    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
    print "\n"
    i = i + 1
print "Centroids\n"
for centroid in centroid_points:
    print centroid
```

```
iteration1:
4.33525090907
5.63618341048
5.62378330985
0 [-2.6816121341554244, 0.4387800225117985]
1 [5.431259098350165, 0.7531374418947868]
2 [0.7963174910876522, 5.419446653714617]


iteration2:
4.38942551908
5.30689875626
5.30937092936
0 [-4.219544623788974, 0.209058167211663]
1 [5.1958522411319015, 0.3334853533723542]
2 [0.40235363459492957, 5.203065613832178]


iteration3:
4.69625408185
5.18765172565
5.18288471468
0 [-4.6058853595554945, 0.1078428968944301]
1 [5.10139048325138, 0.15156599354661596]
2 [0.23172595157895318, 5.096489533741727]


iteration4:
4.89069054631
5.13590004411
5.1241245473
0 [-4.801190218234985, 0.05878082502005945]
1 [5.055563555258459, 0.06290904872145056]
2 [0.14952380182408184, 5.04128846202037]


iteration5:
4.99282774516
5.11371221408
5.09344277395
0 [-4.906463425879866, 0.032759981077283826]
1 [5.034173461891218, 0.021068662996156867]
2 [0.10500938054864571, 5.008476237362788]


iteration6:
5.03416502007
5.10765375134
5.07486089641
0 [-4.954102811929994, 0.02331959334603343]
1 [5.025125550625107, -0.001618434529619084]
2 [0.08397362923855911, 4.992421086785056]
```

```
iteration7:
5.05821350844
5.10483668637
5.06978329177
0 [-4.977997339660123, 0.019146098700403916]
1 [5.021081792321543, -0.014110744528659413]
2 [0.0726763911253696, 4.984432926373196]


iteration8:
5.07003770339
5.10342525924
5.06680272175
0 [-4.990012319882803, 0.01740513922914574]
1 [5.01919583889221, -0.020930543933152697]
2 [0.06693097575785635, 4.980690463262181]


iteration9:
5.07661833597
5.10272360361
5.06548643421
0 [-4.99624302220268, 0.016933860461069136]
1 [5.018289897210405, -0.024641826168726056]
2 [0.06399549557045879, 4.97898189238902]


iteration10:
5.08010791946
5.10237412977
5.06491066536
0 [-4.999537542091526, 0.016916506997156085]
1 [5.01784542233342, -0.026658865151699053]
2 [0.062489726282723417, 4.978209955370474]


Centroids

[-4.999537542091526, 0.016916506997156085]
[5.01784542233342, -0.026658865151699053]
[0.062489726282723417, 4.978209955370474]
```

Things I tried for Question 10

My stream-of-consciousness thoughts on what's going on in this problem

- Initially, I tried adding weights in both the combiner and reducer (since at scale we can't count on whether or not the combiner actually ran. This gave me centroids all near zero, which seemed wrong.
- I wondered if adding the weights in both the combiner and the reducer would cause things to double count, so I tried just adding them in the reducer and got similar

results.
- Then, I thought that maybe I was defining the weights incorrectly. Initially, I'd calculated the weight for each point based on its distance from the origin, but why would we care about this in the context of clustering. What might make more sense would be to think of the weights based on the distance of a point to it's current centroid assignment. That way, points that are closer to the cluster centroid will have more of an influence over where the centroids move in the next iteration than points that are farther away. This makes much more intuitive sense than what I was initially doing, but I'm still getting centroids that are close to zero.
- The reason the centroids are so close to zero is that at this point, I'm multiplying each point by the inverse vector length (which is the same as dividing by the vector length). This total then gets divided AGAIN by the total number of points in the cluster. To fix this, we'd need to ALSO normalize the denominator, which is what I ultimately decided to do.

In [ ]:
```

```