W261 Fall, 2016, Midterm

Name: Shih Yu Chang

Email: sychang@ischool.berkeley.edu

In [18]:
```
import numpy as np
from __future__ import division

%reload_ext autoreload
%autoreload 2
```

In [19]:
```
%%writefile kltext.txt
1.Data Science is an interdisciplinary field about processes and systems t
2.Machine learning is a subfield of computer science[1] that evolved from
```

```
Writing kltext.txt
```

In [20]:
```
import numpy as np
np.log(3)
```

Out[20]:
```
1.0986122886681098
```

Pairwise similarity using K-L divergence

In [21]:
```
!cat kltext.txt
```

```
1.Data Science is an interdisciplinary field about processes and system
s to extract knowledge or insights from large volumes of data in variou
s forms (data in various forms, data in various forms, data in various
forms), either structured or unstructured,[1][2] which is a continuatio
n of some of the data analysis fields such as statistics, data mining a
nd predictive analytics, as well as Knowledge Discovery in Databases.
2.Machine learning is a subfield of computer science[1] that evolved fr
om the study of pattern recognition and computational learning theory i
n artificial intelligence.[1] Machine learning explores the study and c
onstruction of algorithms that can learn from and make predictions on d
ata.[2] Such algorithms operate by building a model from example inputs
 in order to make data-driven predictions or decisions,[3]:2 rather tha
n following strictly static program instructions.
```

In [26]:

```
%%writefile kldivergence.py
from __future__ import division
from mrjob.job import MRJob
import re
import numpy as np
class kldivergence(MRJob):

    # process each string character by character
    # the relative frequency of each character emitting Pr(character|str)
    # for input record 1.abcbe
    # emit "a"    [1, 0.2]
    # emit "b"    [1, 0.4] etc...
    def mapper1(self, _, line):
        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:
            yield key, [index, count[key]*1.0/len(letter_list)]

    # on a component i calculate (e.g., "b")

    #  (P(i) log (P(i) / Q(i))
    #
    def reducer1(self, key, values):
        p = 0
        q = 0
        for v in values:
            if v[0] == 1:  #String 1
                p = v[1]
            else:          # String 2
                q = v[1]
        sim = np.log(p/q) * p

        yield None, sim


    #Aggegate components
    def reducer2(self, key, values):
        kl_sum = 0
        for value in values:
            kl_sum = kl_sum + value
        yield "KLDivergence", kl_sum

    def steps(self):
        return [self.mr(mapper=self.mapper1,
                        reducer=self.reducer1),

                self.mr(reducer=self.reducer2)

               ]
```

In [27]:

```
%reload_ext autoreload
%autoreload 2
from mrjob.job import MRJob
from kldivergence import kldivergence

#dont forget to save kltext.txt (see earlier cell)
mr_job = kldivergence(args=['kltext.txt'])
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)
```

```
('KLDivergence', 0.08088278445318145)
```

In [30]:

```
%%writefile kldivergence_smooth.py
from __future__ import division
from mrjob.job import MRJob
import re
import numpy as np
class kldivergence_smooth(MRJob):

    # process each string character by character
    # the relative frequency of each character emitting Pr(character|str)
    # for input record 1.abcbe
    # emit "a"    [1, (1+1)/(5+24)]
    # emit "b"    [1, (2+1)/(5+24) etc...
    def mapper1(self, _, line):
        index = int(line.split('.',1)[0])
        letter_list = re.sub(r"[^A-Za-z]+", '', line).lower()
        count = {}

        # (ni+1)/(n+24)

        for l in letter_list:
            if count.has_key(l):
                count[l] += 1
            else:
                count[l] = 1
        for key in count:
            yield key, [index, (count[key] + 1) * 1.0 / (len(letter_list)


    def reducer1(self, key, values):
        p = 0
        q = 0
        for v in values:
            if v[0] == 1:
                p = v[1]
            else:
                q = v[1]

        sim = np.log(p/q) * p

        yield None, sim

    # Aggregate components
    def reducer2(self, key, values):
        kl_sum = 0
        for value in values:
            kl_sum = kl_sum + value
        yield "KLDivergence", kl_sum

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

               ]

if __name__ == '__main__':
```

In [31]:

```
%reload_ext autoreload
%autoreload 2

from kldivergence_smooth import kldivergence_smooth
mr_job = kldivergence_smooth(args=['kltext.txt'])
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)
```

```
('KLDivergence', 0.06726997279170038)
```

Weighted K-means

In [59]:

```
%%writefile Kmeans.py
from numpy import argmin, array, random
from mrjob.job import MRJob
from mrjob.step import MRStep
from itertools import chain
import os

#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,comb
               ]
    #load centroids info from file
    def mapper_init(self):
        print "Current path:", os.path.dirname(os.path.realpath(__file__))

        self.centroid_points = [map(float,s.split('\n')[0].split(',')) for
        #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
```

In [60]:

```
from numpy import random, array
from Kmeans import MRKmeans, stop_criterion
mr_job = MRKmeans(args=['Kmeandata.csv', '--file', 'Centroids.txt'])

#Geneate 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 centroid_

# 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"
print centroid_points
```

```
iteration1:
Current path: /tmp/Kmeans.cloudera.20161020.003230.821415/job_local_dir
/0/mapper/0
Centroids:  [[0.0, 0.0], [6.0, 3.0], [3.0, 6.0]]
Current path: /tmp/Kmeans.cloudera.20161020.003230.821415/job_local_dir
/0/mapper/1
Centroids:  [[0.0, 0.0], [6.0, 3.0], [3.0, 6.0]]
0 [-3.344726378997632, 0.3375985510805805]
1 [5.379067911319121, 0.15446805295171434]
2 [0.24288276270220563, 5.350519186138149]


iteration2:
Current path: /tmp/Kmeans.cloudera.20161020.003231.043216/job_local_dir
/0/mapper/0
Centroids:  [[0.0, 0.0], [6.0, 3.0], [3.0, 6.0]]
Current path: /tmp/Kmeans.cloudera.20161020.003231.043216/job_local_dir
/0/mapper/1
Centroids:  [[0.0, 0.0], [6.0, 3.0], [3.0, 6.0]]
0 [-3.344726378997632, 0.3375985510805805]
```

In [61]:

```
import csv
from numpy import argmin, array, random

#Euclidean norm
def norm(x):
    return (x[0]**2 + x[1]**2)**0.5

#Calculate find the nearest centroid for data point
def smallestDist(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 distances[min_idx]

data = []

centroids = [[-4.5,0.0],[4.5,0.0],[0.0,4.5]]

num = 0.0
den = 0.0

with open('Kmeandata.csv', 'r') as infile:
    for line in csv.reader(infile):
        point = [float(line[0]), float(line[1])]
        weight = 1/norm(point)
        num += smallestDist(point, centroids) * weight
        den += weight

print num / den
```

```
1.5932559652
```

In [37]:

```
!pwd
```

```
/home/cloudera
```

In [ ]: