In [ ]:
```
## W261 Section 3
#### Safyre Anderson
```

===Map-Reduce===

MT0. Which of the following statememts about map-reduce are true? Check all that apply.

(a) If you only have 1 computer with 1 computing core, then map-reduce is unlikely to help

(b) If we run map-reduce using N computers, then we will always get at least an N-Fold speedup compared to using 1 computer

(c) Because of network latency and other overhead associated with map-reduce, if we run map-reduce using N computers, then we will get less than N-Fold speedup compared to using 1 computer

(d) When using map-reduce with gradient descent, we usually use a single machine that accumulates the gradients from each of the map-reduce machines, in order to compute the paramter update for the iterion

Answer: (a),(c), and (d)

===Order inversion===

MT1. Suppose you wish to write a MapReduce job that creates normalized word co-occurrence data form a large input text. To ensure that all (potentially many) reducers receive appropriate normalization factors (denominators) in the correct order in their input streams (so as to minimize memory overhead), the mapper should emit according to which pattern:

(a) emit (*,word) count

(b) There is no need to use order inversion here

(c) emit (word,*) count

(d) None of the above

Answer: (c)

===Apriori principle===

MT2. When searching for frequent itemsets with the Apriori algorithm (using a threshold, N), the Apriori principle allows us to avoid tracking the occurrences of the itemset {A,B,C} provided

(a) all subsets of {A,B,C} occur less than N times.

(b) any pair of {A,B,C} occurs less than N times.

(c) any subset of {A,B,C} occurs less than N times.

(d) All of the above

Answer: (d)

===Bayesian document classification===

MT3. When building a Bayesian document classifier, Laplace smoothing serves what purpose?

(a) It allows you to use your training data as your validation data.

(b) It prevents zero-products in the posterior distribution.

(c) It accounts for words that were missed by regular expressions.

(d) None of the above

Answer: (b)

===Bias-variance tradeoff===

MT4. By increasing the complexity of a model regressed on some samples of data, it is likely that the ensemble will exhibit which of the following?

(a) Increased variance and bias

(b) Increased variance and decreased bias

(c) Decreased variance and bias

(d) Decreased variance and increased bias

Answer: (d)

===Combiners===

MT5. Combiners can be integral to the successful utilization of the Hadoop shuffle. This utility is as a result of

(a) minimization of reducer workload

(b) both (a) and (c)

(c) minimization of network traffic

(d) none of the above

Answer: (b)

===Pairwise similarity using K-L divergence===

In probability theory and information theory, the Kullback–Leibler divergence (also information divergence, information gain, relative entropy, KLIC, or KL divergence) is a non-symmetric measure of the difference between two probability distributions P and Q. Specifically, the Kullback–Leibler divergence of Q from P, denoted DKL(P‖Q), is a measure of the information lost when Q is used to approximate P:

For discrete probability distributions P and Q, the Kullback–Leibler divergence of Q from P is defined to be

KLDistance(P, Q) = Sum over i (P(i) log (P(i) / Q(i))

In the extreme cases, the KL Divergence is 1 when P and Q are maximally different and is 0 when the two distributions are exactly the same (follow the same distribution).

For more information on K-L Divergence see:

https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence (https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence)

For the next three question we will use an MRjob class for calculating pairwise similarity using K-L Divergence as the similarity measure:

Job 1: create inverted index (assume just two objects) Job 2: calculate/accumulate the similarity of each pair of objects using K-L Divergence

Download the following notebook and then fill in the code for the first reducer to calculate the K-L divergence of objects (letter documents) in line1 and line2, i.e., KLD(Line1||line2).

Here we ignore characters which are not alphabetical. And all alphabetical characters are lower-cased in the first mapper.

http://nbviewer.ipython.org/urls/dl.dropbox.com/s/9onx4c2dujtkgd7/Kullback%E2%80%93Leibler%20diverge MIDS-Midterm.ipynb
(http://nbviewer.ipython.org/urls/dl.dropbox.com/s/9onx4c2dujtkgd7/Kullback%E2%80%93Leibler%20diverg MIDS-Midterm.ipynb)
https://www.dropbox.com/s/zr9xfhwakrxz9hc/Kullback%E2%80%93Leibler%20divergence-MIDS-Midterm.ipynb?dl=0
(https://www.dropbox.com/s/zr9xfhwakrxz9hc/Kullback%E2%80%93Leibler%20divergence-MIDS-Midterm.ipynb?dl=0)

In [1]:
```
%%writefile kldivergence.py
#!/usr/bin/env python
from mrjob.job import MRJob
from mrjob.step import MRStep
import re
```

```
import numpy as np

class kldivergence(MRJob):
    ## line 1 = P
    ## line 2 = Q

    def mapper1(self, _, line):
        index = int(line.split('.',1)[0])
        ## replaces everything that is not a letter into nothing
        # all letters get smushed into one big line
        letter_list = re.sub(r"[^A-Za-z]+", '', line).lower()
        count = {}
        # Count occurances of each character
        for l in letter_list:
            if count.has_key(l):
                count[l] += 1
            # still counts 1, so no log(0)
            else:
                count[l] = 1
        for key in count:
            yield key, [index, count[key]*1.0/len(letter_list)]


    def reducer1(self, letter, index_prior_pair):
        #Fill in your code
        # emit partial sums of KLD for each i
        # where i is a letter
        P_dict = {}
        Q_dict = {}

        for index, prior in index_prior_pair:
            if index == 1:
                P_dict.setdefault(letter,1)
                P_dict[letter] = float(prior)

            if index == 2:
                Q_dict.setdefault(letter,1)
                Q_dict[letter] = float(prior)

            # partial sum can also be written as:
            # p(i)*log(p(i) - p(i)*log(q(i))
            # --> if q(i) is 0, second term is basically 0

            for key in P_dict.keys():
                term1 = P_dict[key] * np.log(P_dict[key])
                try:
                    term2 = P_dict[key] * np.log(Q_dict[key])
                    partial_sum = term1 - term2
                    yield key, partial_sum
                except KeyError:
```

```
                    partial_sum = 0
                    yield key, partial_sum


    def combiner2(self, key, values):
        kl_sum = 0
        for value in values:
            kl_sum = kl_sum + value
        yield None, kl_sum

    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(combiner = self.combiner2,
                       reducer=self.reducer2)]

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

```
Writing kldivergence.py
```

In [4]:

```
from kldivergence import kldivergence
mr_job = kldivergence(args=['/Users/Safyre/Documents/W261_Midterm_Prep
/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)
```

```
(None, 0.08088278445318145)
```

MT6. Which number below is the closest to the result you get for KLD(Line1||line2)? (a) 0.7 (b) 0.5 (c) 0.2 (d) 0.1

Answer: (d) 0.1

MT7. Which of the following letters are missing from these character vectors? (a) p and t

(b) k and q

(c) j and q

(d) j and f

In [5]:

```
## Part 7:
import string, re
alphabet=string.ascii_lowercase

str1 = "Data Science is an interdisciplinary field about processes and
systems to extract knowledge or insights from large volumes of data in
various forms (data in various forms, data in various forms, data in v
arious forms), either structured or unstructured,[1][2] which is a con
tinuation of some of the data analysis fields such as statistics, data
mining and predictive analytics, as well as Knowledge Discovery in Dat
abases"
str2 = "Machine learning is a subfield of computer science[1] that evo
lved from the study of pattern recognition and computational learning
theory in artificial intelligence.[1] Machine learning explores the st
udy and construction of algorithms that can learn from and make predic
tions on data.[2] Such algorithms operate by building a model from exa
mple inputs in order to make data-driven predictions or decisions,[3]:
2 rather than following strictly static program instructions"

tot_str = str1+str2

tot_str = re.sub(r"[^A-Za-z]+", '', tot_str).lower()

#print set(tot_str)
for i in alphabet:
    if i not in tot_str:
        print i
```

```
j
q
z
```

Answer: (c)

MT8. The KL divergence on multinomials is defined only when they have nonzero entries. For zero entries, we have to smooth distributions. Suppose we smooth in this way:

(ni+1)/(n+24)

where ni is the count for letter i and n is the total count of all letters.

After smoothing, which number below is the closest to the result you get for KLD(Line1||line2)??

(a) 0.08

(b) 0.71

(c) 0.02

(d) 0.11

In [6]:

```
%%writefile kldivergencesmooth.py
#!/usr/bin/env python
from mrjob.job import MRJob
from mrjob.step import MRStep
import re
import numpy as np

class kldivergencesmooth(MRJob):
    ## line 1 = P
    ## line 2 = Q

    def mapper1(self, _, line):
        index = int(line.split('.',1)[0])
        ## replaces everything that is not a letter into nothing
        # all letters get smushed into one big line
        letter_list = re.sub(r"[^A-Za-z]+", '', line).lower()
        count = {}
        # Count occurances of each character
        for l in letter_list:
            if count.has_key(l):
                count[l] += 1
            # still counts 1, so no log(0)
            else:
                count[l] = 1
        for key in count:
            yield key, [index, (count[key]*1.0+1)/(len(letter_list)+24
)]


    def reducer1(self, letter, index_prior_pair):
        #Fill in your code
```

```
        # emit partial sums of KLD for each i
        # where i is a letter
        P_dict = {}
        Q_dict = {}

        for index, prior in index_prior_pair:
            if index == 1:
                P_dict.setdefault(letter,1)
                P_dict[letter] = float(prior)

            if index == 2:
                Q_dict.setdefault(letter,1)
                Q_dict[letter] = float(prior)

            # partial sum can also be written as:
            # p(i)*log(p(i) - p(i)*log(q(i))
            # --> if q(i) is 0, second term is basically 0

            for key in P_dict.keys():
                term1 = P_dict[key] * np.log(P_dict[key])
                try:
                    term2 = P_dict[key] * np.log(Q_dict[key])
                    partial_sum = term1 - term2
                    yield key, partial_sum
                except KeyError:
                    partial_sum = 0
                    yield key, partial_sum


    def combiner2(self, key, values):
        kl_sum = 0
        for value in values:
            kl_sum = kl_sum + value
        yield None, kl_sum

    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(combiner = self.combiner2,
                      reducer=self.reducer2)]

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

```
Writing kldivergencesmooth.py
```

In [7]:
```
from kldivergencesmooth import kldivergencesmooth
mr_job = kldivergencesmooth(args=['/Users/Safyre/Documents/W261_Midterm_Prep/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)
```

```
(None, 0.06726997279170045)
```

In [ ]:
```
Answer: (a)
```

In [ ]:

===Gradient descent===

MT9. Which of the following are true statements with respect to gradient descent for machine learning, where alpha is the learning rate. Select all that apply

(a) To make gradient descent converge, we must slowly decrease alpha over time and use a combiner in the context of Hadoop.

(b) Gradient descent is guaranteed to find the global minimum for any function J() regardless of using a combiner or not in the context of Hadoop

(c) Gradient descent can converge even if alpha is kept fixed. (But alpha cannot be too large, or else it may fail to converge.) Combiners will help speed up the process.

(d) For the specific choice of cost function J() used in linear regression, there is no local optima (other than the global optimum).

Answer: (c) and (d)

===Weighted K-means===

Write a MapReduce job in MRJob to do the training at scale of a weighted K-means algorithm.

You can write your own code or you can use most of the code from the following notebook:

http://nbviewer.ipython.org/urls/dl.dropbox.com/s/kjtdyi10nwmk4ko/MrJobKmeans-MIDS-Midterm.ipynb (http://nbviewer.ipython.org/urls/dl.dropbox.com/s/kjtdyi10nwmk4ko/MrJobKmeans-MIDS-Midterm.ipynb) https://www.dropbox.com/s/kjtdyi10nwmk4ko/MrJobKmeans-MIDS-Midterm.ipynb?dl=0 (https://www.dropbox.com/s/kjtdyi10nwmk4ko/MrJobKmeans-MIDS-Midterm.ipynb?dl=0)

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.

Using the following data answer the following questions:

https://www.dropbox.com/s/ai1uc3q2ucverly/Kmeandata.csv?dl=0 (https://www.dropbox.com/s/ai1uc3q2ucverly/Kmeandata.csv?dl=0)

In [8]:

```
%%writefile Kmeans.py
import numpy as np
from numpy import argmin, array, random
from mrjob.job import MRJob
from mrjob.step import MRJobStep
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 [
            MRJobStep(mapper_init = self.mapper_init, mapper=self.mapp
er,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 in open("Centroids.txt").readlines()]
        open('Centroids.txt', 'w').close()
    #load data and output the nearest centroid index and data point
    def mapper(self, _, line):
        D = (map(float,line.split(',')))
        idx = MinDist(D,self.centroid_points)
        norm = np.sqrt((D[0]**2)+(D[1]**2))
        yield int(idx), (norm**-1, D[0],D[1],1)

    #Combine sum of data points locally
    def combiner(self, idx, inputdata):
        sumx = sumy = num = 0
        for weight, x, y, n in inputdata:
            num = num + n
            sumx = sumx + x*weight
            sumy = sumy + y*weight
        yield int(idx),(sumx,sumy,num)
    #Aggregate sum for each cluster and then calculate the new centroi
ds
    def reducer(self, idx, inputdata):
        centroids = []
        num = [0]*self.k
        distances = 0
        for i in range(self.k):
            centroids.append([0,0])
        for x, y, n in inputdata:
            num[idx] = num[idx] + n
            centroids[idx][0] = centroids[idx][0] + x
            centroids[idx][1] = centroids[idx][1] + y
        centroids[idx][0] = centroids[idx][0]/num[idx]
```

```
        centroids[idx][1] = centroids[idx][1]/num[idx]
        with open('Centroids.txt', 'a') as f:
            f.writelines(str(centroids[idx][0]) + ',' + str(centroids[
idx][1]) + '\n')
        yield idx,(centroids[idx][0],centroids[idx][1])

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

```
Writing Kmeans.py
```

In [10]:

```
%reload_ext autoreload
%autoreload 2
from numpy import random, array
from Kmeans import MRKmeans, stop_criterion
mr_job = MRKmeans(args=['/Users/Safyre/Documents/W261_Midterm_Prep/Kme
andata.csv', '--file=/Users/Safyre/Documents/W261_Midterm_Prep/Centroi
ds.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 centr
oid_points)

# 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:
0 [-0.9679507185802757, 0.0025351790126423666]
1 [0.9667327574053561, -0.007163503804041002]
2 [0.010756330127372938, 0.9660024871240357]


iteration2:
```

```
0 [-0.9679507185802757, 0.0025351790126423666]
1 [0.9667327574053561, -0.007163503804041002]
2 [0.010756330127372938, 0.9660024871240357]


iteration3:
0 [-0.9679507185802757, 0.0025351790126423666]
1 [0.9667327574053561, -0.007163503804041002]
2 [0.010756330127372938, 0.9660024871240357]


iteration4:
0 [-0.9679507185802757, 0.0025351790126423666]
1 [0.9667327574053561, -0.007163503804041002]
2 [0.010756330127372938, 0.9660024871240357]


iteration5:
0 [-0.9679507185802757, 0.0025351790126423666]
1 [0.9667327574053561, -0.007163503804041002]
2 [0.010756330127372938, 0.9660024871240357]


iteration6:
0 [-0.9679507185802757, 0.0025351790126423666]
1 [0.9667327574053561, -0.007163503804041002]
2 [0.010756330127372938, 0.9660024871240357]


iteration7:
0 [-0.9679507185802757, 0.0025351790126423666]
1 [0.9667327574053561, -0.007163503804041002]
2 [0.010756330127372938, 0.9660024871240357]


iteration8:
0 [-0.9679507185802757, 0.0025351790126423666]
1 [0.9667327574053561, -0.007163503804041002]
2 [0.010756330127372938, 0.9660024871240357]


iteration9:
0 [-0.9679507185802757, 0.0025351790126423666]
1 [0.9667327574053561, -0.007163503804041002]
2 [0.010756330127372938, 0.9660024871240357]


iteration10:
0 [-0.9679507185802757, 0.0025351790126423666]
1 [0.9667327574053561, -0.007163503804041002]
```

```
2 [0.010756330127372938, 0.9660024871240357]


Centroids

[[-0.9679507185802757, 0.0025351790126423666], [0.9667327574053561,
-0.007163503804041002], [0.010756330127372938, 0.9660024871240357]]
```

MT10. Which result below is the closest to the centroids you got after running your weighted K-means code for 10 iterations?

(a) (-4.0,0.0), (4.0,0.0), (6.0,6.0)

(b) (-4.5,0.0), (4.5,0.0), (0.0,4.5)

(c) (-5.5,0.0), (0.0,0.0), (3.0,3.0)

(d) (-4.5,0.0), (-4.0,0.0), (0.0,4.5)

Answer: (c)

MT11. Using the result of the previous question, which number below is the closest to the average weighted distance between each example and its assigned (closest) centroid?

The average weighted distance is defined as sum over i (weighted_distance_i) / sum over i (weight_i)

(a) 2.5 (b) 1.5 (c) 0.5 (d) 4.0

In [ ]:
```
Answer (c)
```

MT12. Which of the following statements are true? Select all that apply. a) Since K-Means is an unsupervised learning algorithm, it cannot overfit the data, and thus it is always better to have as large a number of clusters as is computationally feasible. b) The standard way of initializing K-means is setting $\mu_1=\cdots=\mu_k$ to be equal to a vector of zeros. c) For some datasets, the "right" or "correct" value of K (the number of clusters) can be ambiguous, and hard even for a human expert looking carefully at the data to decide. d) A good way to initialize K-means is to select K (distinct) examples from the training set and set the cluster centroids equal to these selected examples.

Answer: (b), (c), (d)

In [ ]:

```

```