W261-2-midterm-krishnaswami

March 2, 2016

MIDS Machine Learning at Scale MidTerm exam
Week 8
Spring, 2016
Exam location is at:
https://www.dropbox.com/s/jdkkttnwd88uxkl/MIDS-MLS-MidTerm-2016-Spring-Live.txt?dl=0

1 Instructions for midterm

Instructions:

1. Please acknowledge receipt of exam by sending a quick reply to the instructors
2. Review the submission form first to scope it out (it will take a 5-10 minutes to input your answers and other information into this form)
3. Please keep all your work and responses in ONE (1) notebook only (and submit via the form)
4. Please make sure that the NBViewer link for your Submission notebook works
5. Please do NOT discuss this exam with anyone (including your class mates) until after 8AM (West coast time) Friday, March 4, 2016

Please use your live session time from week 8 to complete this mid term (plus an additional 30 minutes if you need it). This is an open book exam meaning you can consult webpages and textbooks (but not each other or other people). Please complete this exam by yourself.
Please submit your solutions and notebook via the following form:
http://goo.gl/forms/ggNYfRXz0t

2 Exam durations (All times are in California Time)

Live Session Group #4 4:00 PM - 6:00 PM (Tuesday)
Live Session Group #2 4:00 PM - 6:00 PM (Wednesday)
Live Session Group #3 6:30 PM - 8:30 PM (Wednesday)

3 Exam questions begins here

3.1 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, d

3.2 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: a – The key should be sorted prior to the "real" keys corresponding to data

3.3 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: c – if any subset appears fewer than N times, the whole itemset cannot appear N times.

3.4 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 – it effectively adds a tiny amount for words not present in the corpus ($\frac{\alpha}{N+\alpha}$)

3.5 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: b – bias tends to drop off with model complexity. However, alternate samples of data tend to fit vastly different models since the complex model is more apt to overfit its training data.

3.6 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

3.7 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

$$\text{KLDistance}(P, Q) = \sum_i P(i) \log \left(\frac{P(i)}{Q(i)} \right)$$

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

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%20divergence-MIDS-Midterm.ipynb https://www.dropbox.com/s/zr9xfhwakrxz9hc/Kullback%E2%80%93Leibler%20divergence-MIDS-Midterm.ipynb?dl=0

In [4]:
```
%%writefile kltext.txt
1.Data Science is an interdisciplinary field about processes and systems to extract knowledge o
2.Machine learning is a subfield of computer science[1] that evolved from the study of pattern
```

```
Writing kltext.txt
```

In [10]:
```
%load_ext autoreload
```

In [108]:
```
%%writefile kldivergence.py
#!/opt/anaconda/bin/python
from collections import Counter
from mrjob.job import MRJob
from mrjob.step import MRStep
from mrjob.protocol import JSONProtocol
import numpy as np
import re
import sys

class kldivergence(MRJob):
    INTERNAL_PROTOCOL=JSONProtocol
    def mapper1(self, _, line):
        index = line.split('.',1)[0]
        letter_list = re.sub(r"[^A-Za-z]+", '', line).lower()
        count = Counter()
        for l in letter_list:
            count[l] += 1
        # change for MT8
        for key in count:
            yield key, {index: (1.0+count[key])/(24.0+len(letter_list))}
        # prev:
        # for key in count:
        #     yield key, {index: 1.0*count[key]/len(letter_list)}

    def reducer1(self, key, values):
        # input is inverted index posting w frequency
        row={}
        for val in values:
            row.update(val)
        p_i=row.get('1')
        q_i=row.get('2')
        yield key, p_i*np.log(p_i/q_i)

    def collect(self, key, value):
        ## Print letters for MT7
        #print "Discarding key", key
        yield None, value
    def reducer2(self, key, values):
        yield None, sum(values)

    def steps(self):
        return [MRStep(mapper=self.mapper1,
```

```
                                reducer=self.reducer1),
                        MRStep(mapper=self.collect,
                                reducer=self.reducer2)]

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

```
Overwriting kldivergence.py
```

```
In [100]: %autoreload 2
    from kldivergence import kldivergence
    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)
```

```
Discarding key a
Discarding key b
Discarding key c
Discarding key d
Discarding key e
Discarding key f
Discarding key g
Discarding key h
Discarding key i
Discarding key k
Discarding key l
Discarding key m
Discarding key n
Discarding key o
Discarding key p
Discarding key r
Discarding key s
Discarding key t
Discarding key u
Discarding key v
Discarding key w
Discarding key x
Discarding key y
(None, 0.08088278445318145)
```

3.7.1 Questions

MT6. Which number below is the closest to the result you get for KLD($Line_1 \,||\, line_2$)?

(a) 0.7

(b) 0.5

(c) 0.2

(d) 0.1

Answer: a

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

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:

$$\frac{n_i + 1}{n + 24}$$

where n_i 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($line_1 \,||\, line_2$)??

(a) 0.08

(b) 0.71

(c) 0.02

(d) 0.11

```
In [107]: %autoreload 2
          from kldivergence import kldivergence
          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)
```

```
(None, 0.06759173231460627)
```

Answer: b

3.8 Gradient descent

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

(a) To make gradient descent converge, we must slowly decrease α 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 α is kept fixed. (But α 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, d

3.9 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 https://www.dropbox.com/s/kjtdyi10nwmk4ko/MrJobKmeans-MIDS-Midterm.ipynb?dl=0
Weight each example as follows using the inverse vector length (Euclidean norm):

$$\text{weight}(\mathbf{X}) = \|\mathbf{X}\|^{-1}$$

,
where

$$\|\mathbf{X}\| = \sqrt{\mathbf{X} \cdot \mathbf{X}} = \sqrt{X_1^2 + X_2^2}$$

Here $\mathbf{X}$ is vector made up of X_1 and X_2.
Using the following data answer the following questions:
https://www.dropbox.com/s/ai1uc3q2ucverly/Kmeandata.csv?dl=0

In [120]:
```
!wget -q -O Kmeandata.csv https://www.dropbox.com/s/ai1uc3q2ucverly/Kmeandata.csv?dl=1
```

In [144]:
```
%%writefile Kmeans.py
from itertools import chain
from mrjob.job import MRJob
from mrjob.step import MRJobStep
from numpy import argmin, array, random, sqrt
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**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.mapper,combiner = self.combi
                       ]
            #load centroids info from file
            def mapper_init(self):
                with open("Centroids.txt") as cent:
                    self.centroid_points = [map(float,s.split('\n')[0].split(',')) for s in cent]
                with open("Centroids.txt", 'w') as cent:
                    pass
            #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)
                w=1.0/sqrt(D[0]**2+D[1]**2)
                yield int(idx), (w*D[0],w*D[1],w)
            #Combine sum of data points locally
            def combiner(self, idx, inputdata):
                sumx = sumy = wgt = 0
                for x,y,w in inputdata:
                    wgt = wgt + w
                    sumx = sumx + x
                    sumy = sumy + y
                yield int(idx),(sumx,sumy,wgt)
            #Aggregate sum for each cluster and then calculate the new centroids
            def reducer(self, idx, inputdata):
                centroids = [[0,0]]*self.k
                wgt = [0]*self.k
                distances = 0
                for x, y, w in inputdata:
                    wgt[idx] += w
                    centroids[idx][0] += x
                    centroids[idx][1] += y
                centroids[idx][0] /= wgt[idx]
                centroids[idx][1] /= wgt[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()
```

```
Overwriting Kmeans.py
```

```
In [145]: %autoreload 2
          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_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 [-2.6816121341554244, 0.4387800225117981]
1 [5.203939274722273, 0.18108381085421293]
2 [0.2798236662882328, 5.147133354098043]


iteration2:
0 [-4.499453073691768, 0.1017143951710932]
1 [4.7342756092123475, -0.035081051175915486]
2 [0.10883719601553689, 4.724161916864905]


iteration3:
0 [-4.618233072986696, 0.01209570625589213]
1 [4.7342756092123475, -0.035081051175915486]
2 [0.05163332299537063, 4.637075828035132]


iteration4:
0 [-4.618233072986696, 0.01209570625589213]
1 [4.7342756092123475, -0.035081051175915486]
2 [0.05163332299537063, 4.637075828035132]


iteration5:
0 [-4.618233072986696, 0.01209570625589213]
1 [4.7342756092123475, -0.035081051175915486]
2 [0.05163332299537063, 4.637075828035132]


iteration6:
0 [-4.618233072986696, 0.01209570625589213]
1 [4.7342756092123475, -0.035081051175915486]
2 [0.05163332299537063, 4.637075828035132]
```

```
iteration7:
0 [-4.618233072986696, 0.01209570625589213]
1 [4.7342756092123475, -0.035081051175915486]
2 [0.05163332299537063, 4.637075828035132]


iteration8:
0 [-4.618233072986696, 0.01209570625589213]
1 [4.7342756092123475, -0.035081051175915486]
2 [0.05163332299537063, 4.637075828035132]


iteration9:
0 [-4.618233072986696, 0.01209570625589213]
1 [4.7342756092123475, -0.035081051175915486]
2 [0.05163332299537063, 4.637075828035132]


iteration10:
0 [-4.618233072986696, 0.01209570625589213]
1 [4.7342756092123475, -0.035081051175915486]
2 [0.05163332299537063, 4.637075828035132]


Centroids

[[-4.618233072986696, 0.01209570625589213], [4.7342756092123475, -0.035081051175915486], [0.0516333229
```

3.9.1 Questions:

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: b

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

$$\frac{\sum_i \text{weighted\_distance}_i}{\sum_i \text{weight}_i}$$

(a) 2.5

(b) 1.5

(c) 0.5

(d) 4.0

```
In [160]: import numpy as np
          def weight(X):
              return 1.0/np.sqrt(np.linalg.norm(X))

          def dist(X,Y):
              return np.sqrt(np.linalg.norm(X-Y))

          cp=np.array(centroid_points)
          def classify(X):
              return np.argmin(np.array([ dist(X, centroid) for centroid in cp[:,:]]))

          with open('Kmeandata.csv') as f:
              num=0.0
              den=0.0
              for line in f:
                  X=np.array(line.strip().split(','), dtype=float)
                  centroid=cp[classify(X),:]
                  w=weight(X)
                  num += w*dist(X, centroid)
                  den += w
              print num/den
```

```
1.2174445577
```

Answer: b

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: c, d