CSE250A Homework3 Answer

Yue Wang, A53102167

October 18, 2015

3.1 Inference

This model is a Markov chain, which means the distribution of each node only depends on the immediate previous node and the transmission probabilities. We can also view this in using conditional independence that every node conditional independent to its previous node except its immediate previous node given its immediate previous node.

(a)

In this question, I will prove it using induction.

If t = 1, we have: $P(X_2 = j|X_1 = i) = A^1_{ij}$, which is true from the definition of P.

Assume t = n, we have: $P(X_{n+1} = j|X_1 = i) = [A^n]_{ij}$

When t = n+1:

$P(X_{n+2} = j|X_1 = i) = \sum_{k=1}^{m} P(X_{n+2} = j, X_{n+1} = k|X_1 = i) = \sum_{k=1}^{m} P(X_{n+2} = j|X_{n+1} = k, X_1 = i)P(X_{n+1} = k|X_1 = i) = \sum_{k=1}^{m} P(X_{n+2} = j|X_{n+1} = k)P(X_{n+1} = k|X_1 = i) = \sum_{k=1}^{m} A_{kj} * [A^n]_{ik} = \sum_{k=1}^{m} [A^n]_{ik} * A_{kj} = [A]^{n+1}_{ij}$

(According to the definition of matrix production, if $C = A * B, C_{ij} = \sum_{k=1}^{m} A_{ik}B_{kj}$ where m is the number of columns of A and the number of rows of B)

(b)

When we do the inference, one possible way is to use get the A^t first which the complexity is $O(m^3t)$(When doing matrix multiplication, the plain way takes $O(m^3)$ and there are totally t matrix multiplications). Then use marginalization to get the $P(X_{t+1} = j)$, because $P(X_{t+1} = j) = \sum_{i=1}^{m} P(X_{t+1} = j, X_1 = i) = \sum_{i=1}^{m} P(X_{t+1} = j|X_1 = i)P(X_1 = i)$. And the time complexity is $O(m^3t + m) = O(m^3t)$

However there are more information than we need in A^t. So let's analyze our object again and look for a new efficient algorithm.

Let's define $V_{X_i} = [p_{X_{i1}}, p_{X_{i2}}, \cdots, p_{X_{im}}]$ as the probability vector which denotes the probability distribution of X_i. ($p_{X_{ik}}$ denotes the probability of $X_i = k$)

Then V_{X_1} is the initial probability distribution of the model.

Our objective is to get $V_{X_{t+1}}$, where $V_{X_{t+1}} = V_{X_t} * A$, where $V_{X_t} = V_{X_{t-1}} * A, \cdots, V_{X_2} = V_{X_1} * A$

There are totally t vector-matrix multiplications and each vector-matrix multiplication takes $O(m^2)$ steps. So the time complexity is $O(m^2t)$.

Proof of $V_{X_i} = V_{X_{i-1}} * A$:

$V_{X_ik} = p_{X_{ik}} = \sum_{j=1}^{m} P(X_i = k, X_{i-1} = j) = \sum_{j=1}^{m} P(X_i = k|X_{i-1} = j)P(X_{i-1} = j) = \sum_{j=1}^{m} A_{jk} *$
$V_{X_{i-1}j} = V_{X_{i-1}} * A_{*k}$ where A_{*k} denotes the kth column of A

So $V_{X_i} = V_{X_{i-1}} * A$

(c)

We know the complexity of $A * A$ is $O(m^3)$ and if we want to get A^{t+1} step by step, we will do t times matrix production where the time complexity is $O(m^3t)$

However, an alternate way is to calculate the matrix production using divide and conquer. We can calculate A^{t+1} in the following way:

$A^{t+1} = A^{\frac{t+1}{2}} * A^{\frac{t+1}{2}}$, if t+1 is even. $A^{t+1} = A^{\frac{t+1}{2}} * A^{\frac{t+1}{2}} * A$, if t+1 is odd

$A^{\frac{t+1}{2}} = A^{\frac{t+1}{4}} * A^{\frac{t+1}{4}}$, if $\frac{t+1}{2}$ is even. $A^{\frac{t+1}{2}} = A^{\frac{t+1}{4}} * A^{\frac{t+1}{4}}$, if $\frac{t+1}{2}$ is odd

...

Do this recursively until

$A^2 = A * A$ or $A^3 = A * A * A$

And there are totally $O(\log_2 t)$ steps, in each step, it takes $O(m^3)$ to do the matrix production. So the whole work can be done in $O(m^3 \log_2 t)$

(d)

Since the matrix is extremely sparse, we need to look for a new way to represent the transition matrix.

Let's define a new array S whose size is m. And each entry is a list. In the previous matrix, if A_{ij} is non-zero, we add a pair (j, A_{ij}) to the S[i]. Totally, there are O(sm) (j, A_{ij}) pairs.

If we want to get the distribution of X_{t+1}, we can calculate the distribution of $X_1, X_2, X_3, \cdots, X_t, X_{t+1}$ step by step, because X_{t+1} only depends on X_t

Assume we have the distribution of X_{t+1}, then

$P(X_{t+1} = k) = \sum_{i=1}^{m} P(X_{t+1} = k | X_t = i) P(X_t = i)$

So if $P(X_{t+1} = k | X_t = i) = 0$, there is no need to do the calculation $P(X_{t+1} = k | X_t = i) * P(X_t = i)$, so we can concentrate on that $P(X_{t+1} = k | X_t = i) \neq 0$, namely, $A_{ik} \neq 0$, which the number of is O(sm)

Let's define $V_{X_i} = [p_{X_{i1}}, p_{X_{i2}}, \cdots, p_{X_{im}}]$ as the probability vector which denotes the probability distribution of X_i. ($p_{X_{ik}}$ denotes the probability of $X_i = k$)

In our privious array S, there are totally O(sm) (k, A_{ik}) paris. We can loop over these paris and when we see a pair whose the first element is k, we can add $P(X_t = i) * A_{ik}$ to $p_{X_{(t+1)m}}$. And after we loop over all the pairs, which takes O(sm) time, we get the distribution of X_{t+1}. And from X_1 to X_{t+1}, there are t steps. So the time complexity is O(smt)

3.2 Stochastic simulation

(a)

Proof:

$\sum_z P(Z = z|B_1, B_2, \cdots, B_n) = \sum_z (\frac{1-\alpha}{1+\alpha})\alpha^{|z-f(B)|}$

Because f(B) is a constant number when the n is fixed, $z - f(B) \in [-\infty, +\infty]$ and it is an integer

Let $x = z - f(B)$ and $x \in [-\infty, +\infty]$ and x is an integer

$\sum_z P(Z = z|B_1, B_2, \cdots, B_n) = \sum_z (\frac{1-\alpha}{1+\alpha})\alpha^{|z-f(B)|} = \sum_x (\frac{1-\alpha}{1+\alpha})\alpha^{|x|}$ where $x \in [-\infty, +\infty]$ and x is an integer

$\sum_z P(Z = z|B_1, B_2, \cdots, B_n) = (\frac{1-\alpha}{1+\alpha})(\sum_x (2 * \alpha^x) - 1)$ where $x \in [0, +\infty]$ and X is an integer(Omitted in the following proof).

We can see α^x is a geometric progression. So:

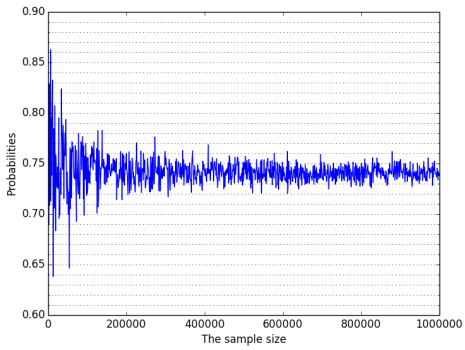
$\sum_{x=0}^{+\infty} (2 * \alpha^x) - 1 = 2 * \lim_{x\to\infty} \frac{1-\alpha^x}{1-\alpha} - 1 = \lim_{x\to\infty} \frac{2-2\alpha^x-1+\alpha}{1-\alpha} = \lim_{x\to\infty} \frac{1-2\alpha^x+\alpha}{1-\alpha} = \frac{1+\alpha}{1-\alpha}$

$\sum_z P(Z = z|B_1, B_2, \cdots, B_n) = (\frac{1-\alpha}{1+\alpha})(\frac{1+\alpha}{1-\alpha}) = 1$ where $z \in [-\infty, +\infty]$ and z is an integer

(b)

The probability $P(B_7 = 1|Z = 64)$ is roughly 0.74

(c)



(d)

```
import random
import math
import pylab
import matplotlib.ticker as ticker

#Calculate the posterior probability according to function f
def posterior(Z, B, alpha):
  fB = 0
  for i in xrange(10):
    fB = fB + (2**i)*B[i]
  #print fB
  return (1.0-alpha)*(alpha**math.fabs(Z-fB))/(1.0+alpha)

#A random sample
def sample(N):
  numerator = 0
  denominator = 0
  for i in xrange(N):
    B = []
    for j in xrange(10):
      B.append(random.randint(0, 1))

    postProb = posterior(64, B, 0.35)
    #print postProb
    if B[6]==1:
      numerator = numerator + postProb
    denominator = denominator + postProb
  return numerator/denominator

index = []
samples = []
#Generate more samples
for i in xrange(1000, 1000000, 1000):
  print i
```

```
  index.append(i)
  samples.append(sample(i))

#print last 10 probabilities, convergence on a value
print samples[-10:]

#Plot the probabilities as the function of N
pylab.figure()
ax = pylab.gca()
ax.yaxis.set_major_locator(ticker.MultipleLocator(0.05))
ax.yaxis.set_minor_locator(ticker.MultipleLocator(0.01))
ax.yaxis.grid(True, which='minor')
ax.yaxis.set_ticks_position('both')
plt = pylab.plot(index, samples)
pylab.ylabel('Probabilities')
pylab.xlabel('The sample size')
pylab.show()
```

3.3 Node clustering

Y_1	Y_2	Y_3	Y	$P(Y\|X=0)$	$P(Y\|X=1)$	$P(Z_1=1\|Y)$	$P(Z_2=1\|Y)$
0	0	0	1	0.09375	0.09375	0.9	0.1
1	0	0	2	0.28125	0.09375	0.8	0.2
0	1	0	3	0.09375	0.03125	0.7	0.3
0	0	1	4	0.03125	0.28125	0.6	0.4
1	1	0	5	0.28125	0.03125	0.5	0.5
1	0	1	6	0.09375	0.28125	0.4	0.6
0	1	1	7	0.03125	0.09375	0.3	0.7
1	1	1	8	0.09375	0.09375	0.2	0.8

3.4 Maximum likelihood estimation

a

From the lecture, we know the maximum likelihood solution is:

$P_{ML}(X_{i+1} = x_{i+1}|Pa_{i+1} = x_i) = P_{ML}(X_{i+1} = x_{i+1}|X_i = x_i) = \frac{\text{count}(X_{i+1}=x_{i+1}, X_i=x_i)}{\text{count}(X_i=x_i)}$

And in this question,

$\text{count}(X_{i+1} = x_{i+1}, X_i = x_i) = \text{count}_i(x_i, x_{i+1})$

$\text{count}(X_i = x_i) = \text{count}_i(x_i)$

So for the CPTs,

$P_{ML}(X_{i+1} = x_{i+1}|X_i = x_i) = \frac{\text{count}_i(x_i,x_{i+1})}{\text{count}_i(x_i)}$($x_i$ is the possible value for X_i)

b

$P_{ML}(X_{i-1} = x_{i-1}|X_i = x_i) = \frac{\text{count}(X_{i-1}=x_{i-1}, X_i=x_i)}{\text{count}(X_i=x_i)} = \frac{\text{count}_{i-1}(x_{i-1},x_i)}{\text{count}_i(x_i)}$

c

From the (a), the joint distribution of $\{X_1, X_2, \cdots, X_T\}$ is:

$P(X_1, X_2, \cdots, X_T) = P(X_T|X_{T-1})P(X_{T-1}|X_{T-2}) \cdots P(X_3|X_2)P(X_2|X_1)P(X_1)$ (because of the conditional independence)

$P(X_1, X_2, \cdots, X_T) = \frac{\text{count}_{T-1}(x_{T-1},x_T)}{\text{count}_{T-1}(x_{T-1})} \frac{\text{count}_{T-2}(x_{T-2},x_{T-1})}{\text{count}_{T-2}(x_{T-2})} \cdots \frac{\text{count}_2(x_2,x_3)}{\text{count}_2(x_2)} \frac{\text{count}_1(x_1,x_2)}{\text{count}_1(x_1)} \frac{\text{count}_1(x_1)}{\sum_{x_1} \text{count}_1(x_1)} =$

$\frac{\text{count}_{T-1}(x_{T-1},x_T)}{\text{count}_{T-1}(x_{T-1})} \frac{\text{count}_{T-2}(x_{T-2},x_{T-1})}{\text{count}_{T-2}(x_{T-2})} \cdots \frac{\text{count}_2(x_2,x_3)}{\text{count}_2(x_2)} \frac{\text{count}_1(x_1,x_2)}{m}$ (where m denotes the number of examples)

From the (b), the joint distribution of $\{X_1, X_2, \cdots, X_T\}$ is:

$P(X_1, X_2, \cdots, X_T) = P(X_1|X_2)P(X_2|X_3) \cdots P(X_{T-3}|X_{T-2})P(X_{T-2}|X_{T-1})P(X_T)$ (because of the conditional independence)

$P(X_1, X_2, \cdots, X_T) = \frac{\text{count}_1(x_1,x_2)}{\text{count}_2(x_2)} \frac{\text{count}_2(x_2,x_3)}{\text{count}_3(x_3)} \cdots \frac{\text{count}_{T-1}(x_{T-2},x_{T-1})}{\text{count}_{T-1}(x_{T-1})} \frac{\text{count}_{T-1}(x_{T-1},x_T)}{\text{count}_T(x_T)} \frac{\text{count}_T(x_T)}{\sum_{x_1} \text{count}_T(x_T)} =$

$\frac{\text{count}_1(x_1,x_2)}{\text{count}_2(x_2)} \frac{\text{count}_2(x_2,x_3)}{\text{count}_3(x_3)} \cdots \frac{\text{count}_{T-1}(x_{T-2},x_{T-1})}{\text{count}_{T-1}(x_{T-1})} \frac{\text{count}_{T-1}(x_{T-1},x_T)}{m}$

$= \frac{\text{count}_{T-1}(x_{T-1},x_T)}{\text{count}_{T-1}(x_{T-1})} \frac{\text{count}_{T-2}(x_{T-2},x_{T-1})}{\text{count}_{T-2}(x_{T-2})} \cdots \frac{\text{count}_2(x_2,x_3)}{\text{count}_2(x_2)} \frac{\text{count}_1(x_1,x_2)}{m}$ (which is equal to the distribu-

tion given by the maximum likelihood CPTs for G_1

So the maximum likelihood CPTs for G1 and G2 from this data set give rise to the same joint distribution over the nodes $\{X1, X2, ..., XT\}$

3.5 Statistical language modeling

(a)

The tokens start with the letter 'M' and their numerical unigram probabilities are:

```
1 : MILLION 0.00207275916815
2 : MORE 0.00170889899662
3 : MR. 0.00144160834928
4 : MOST 0.000787917303319
5 : MARKET 0.000780371280468
6 : MAY 0.000729897315629
7 : M. 0.000703406739462
8 : MANY 0.000696729059597
9 : MADE 0.000559861082734
10 : MUCH 0.000514597175811
11 : MAKE 0.000514462643799
12 : MONTH 0.000444909593632
13 : MONEY 0.00043710673694
14 : MONTHS 0.000405760778161
15 : MY 0.000400318346769
16 : MONDAY 0.000381985302598
17 : MAJOR 0.000370892526705
18 : MILITARY 0.000352045814852
19 : MEMBERS 0.000336060965798
20 : MIGHT 0.000273589191532
21 : MUST 0.000266507915631
22 : MEETING 0.000265737414108
23 : ME 0.000263572671735
24 : MARCH 0.000259793545218
25 : MAN 0.000252883491878
26 : MINISTER 0.000239772735806
27 : MS. 0.0002389900041
28 : MAKING 0.000211704466045
29 : MOVE 0.000209955549889
30 : MILES 0.000205968510263
```

(b)

The ten most likely words to follow the word "THE" and their numerical bigram probabilities are:

```
<UNK> 0.615019810006
U. 0.0133724994326
FIRST 0.011720260675
COMPANY 0.0116587880556
NEW 0.00945148007652
UNITED 0.00867230814123
GOVERNMENT 0.006803488636
NINETEEN 0.006650714911
SAME 0.00628706675745
TWO 0.00616074960283
```

(c)

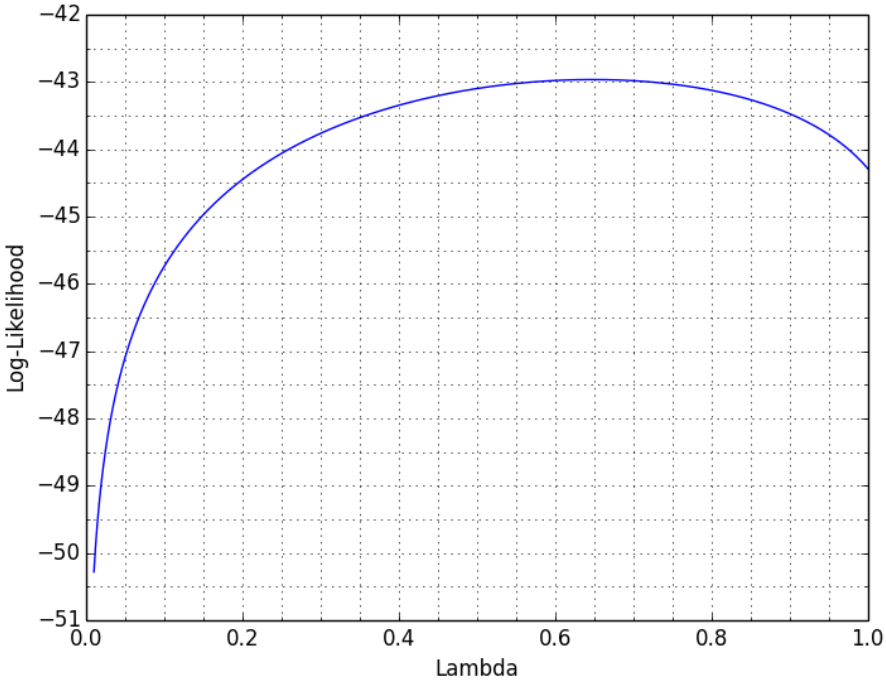
For this sentence, $\mathcal{L}_u = -64.5094403436$ while $\mathcal{L}_b = -40.9181321338$, so the bigram model yields the highest log-likelihood.

(d)

The pairs (sixteen, officials) and (sold, fire) are not observed in the training corpus, which causes the likelihood to be 0, namely, the log-likelihood to be $-\infty$.

(e)

The lambda corresponding to the maximum log-likelihood is roughly 0.6491
The plot is as follow:




```
import math
import sys
import numpy as np
import pylab
import matplotlib.ticker as ticker
vocabs = {}
rvocabs = {}
#Readin the vocabulary
with open('hw3_vocab.txt', 'r') as f_vocab:
  i = 1
  for line in f_vocab:
    line = line.strip()
    vocabs[i] = line
    rvocabs[line] = i
    i += 1
unigrams = {}
n = 0
#readin unigram
```

```
with open('hw3_unigram.txt', 'r') as f_uni:
  i = 1
  for line in f_uni:
    line = line.strip()
    unigrams[vocabs[i]] = int(line)
    i = i + 1
    n += int(line)
toprint = []
#calculate the unigram probabilities of words that start with 'M'
for key in unigrams.keys():
  #unigrams[key] = float(unigrams[key])/n
  if key[0] == 'M':
    toprint.append((key, float(unigrams[key])/n))
#output
c = 1
for x, y in sorted(toprint, key=lambda x: x[1],reverse=True):
  print c,':', x, y
  c = c+1
bigrams = {}
bigrams_all = {}
#readin the bigram and output the bigram probabilities of words that follow 'THE
                                                ,
with open('hw3_bigram.txt', 'r') as f_bi:
  for line in f_bi:
    line = line.strip().split('\t')
    bigrams_all[int(line[0])*1000+int(line[1])] = float(line[2])
    if int(line[0]) == rvocabs['THE']:
      bigrams[line[1]] = float(line[2])/unigrams['THE']
top10 = map(lambda x:(vocabs[int(x[0])], x[1]), sorted(bigrams.items(), key=
                                          lambda x: x[1], reverse=True)[0:10])
for x, y in top10:
    print x, y
#Calculate the unigram log-likelihood probabilities
def uniprob(sentence, unigrams):
  p_uni = 1.0
  for word in sentence.split(' '):
```

```
    p_uni *= (float(unigrams[word])/n)
  return math.log(p_uni)
#Calculate the bigram log-likelihood probabilities
def biprob(bi_sentence, unigrams, bigrams_all, rvocabs):
  bi_sentence = bi_sentence.split(' ')
  p_bi = 1.0
  isOb = False;
  for i in xrange(1, len(bi_sentence), 1):
    prev = bi_sentence[i-1]
    now = bi_sentence[i]
    if rvocabs[prev]*1000+rvocabs[now] not in bigrams_all.keys():
      print prev, now
      isOb = True;
    else:
      p_bi = p_bi*(bigrams_all[rvocabs[prev]*1000+rvocabs[now]]/unigrams[prev])
  if isOb:
    return math.log(sys.float_info.min)
  else:
    return math.log(p_bi)
#Calculate the log-likelihood probabilities using mix model
def mixprob(sentence, unigrams, bigrams_all, rvocabs, lam):
  bi_sentence = '<s> ' + sentence
  #sentence = sentence.split(' ')
  bi_sentence = bi_sentence.split(' ')
  p_mix = 0.0
  for i in xrange(1, len(bi_sentence), 1):
    prev = bi_sentence[i-1]
    now = bi_sentence[i]
    uni = float(unigrams[now])/n
    if rvocabs[prev]*1000+rvocabs[now] not in bigrams_all.keys():
      p_mix = p_mix + math.log(lam*uni)
    else:
      bi = (bigrams_all[rvocabs[prev]*1000+rvocabs[now]]/unigrams[prev])
      p_mix = p_mix + math.log(lam*uni+(1.0-lam)*bi)
  return p_mix
```

```
sentence = 'The stock market fell by one hundred points last week'
sentence = sentence.upper()
print uniprob(sentence, unigrams)
bi_sentence = '<s> ' + sentence
print biprob(bi_sentence, unigrams, bigrams_all, rvocabs)

sentence = 'The sixteen officials sold fire insurance'
sentence = sentence.upper()
print uniprob(sentence, unigrams)
bi_sentence = '<s> ' + sentence
print biprob(bi_sentence, unigrams, bigrams_all, rvocabs)

lams = []
probs = []
#Generate the function samples
for i in np.arange(0.00001, 1, 0.00001):
  lams.append(i)
  probs.append(mixprob(sentence, unigrams, bigrams_all, rvocabs, i))
  print i
#Plot the log-likelihood as the function of lambda
pylab.figure()
ax = pylab.gca()
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_minor_locator(ticker.MultipleLocator(0.5))
ax.yaxis.grid(True, which='minor')
ax.yaxis.set_ticks_position('both')
ax.xaxis.set_major_locator(ticker.MultipleLocator(0.2))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(0.05))
ax.xaxis.grid(True, which='minor')
plt = pylab.plot(lams[1000:], probs[1000:])
pylab.ylabel('Log-Likelihood')
pylab.xlabel('Lambda')
pylab.show()
print lams[probs.index(max(probs))]
```