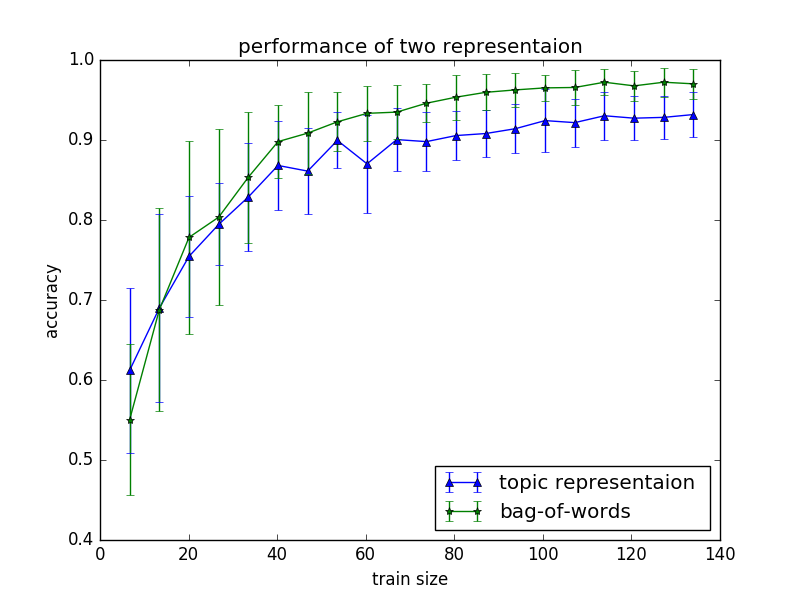
**COMP136 PP4 Report**

Beibei du

**Task 1: Gibbs Simpling**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| based | night | rights | light | sky |
| torque | sho | shifter | clutch | car |
| matter | mph | blah | mustang | diesels |
| drive | lights | used | service | oil |
| satellite | oort | point | system | writes |
| even | extra | cost | make | don |
| back | george | people | moon | bill |
| sci | world | long | nasa | space |
| washington | apr | article | writes | edu |
| temperature | zoo | spencer | toronto | henry |
| spacecraft | information | internet | mars | science |
| redesign | option | shuttle | station | launch |
| steering | turbo | such | power | engine |
| shuttle | pat | solar | hst | mission |
| time | article | day | etc | large |
| nice | probe | driving | ford | car |
| two | book | price | want | car |
| geico | good | article | edu | insurance |
| incoming | ics | uci | gif | edu |
| toyota | buy | manual | small | cars |

**Task2 : Classification**



From figure above, we can see that when train size is small, two representations are relative close, when train size is bigger, both of them have accuracy more than 90% when train size is big , bag of words representation performs better. But bag-of-words representation have more features (405 in this example) than topic representation, run time is longer than topic representation (20 in this example). LDA can reduce dimension and performance not so bad.

import numpy as np

import random

from numpy.linalg import inv

from random import shuffle

import matplotlib.pyplot as plt

from collections import OrderedDict

import csv

import math

alpha\_prime = 0.01

def read\_file(filename):

with open(filename, 'r') as f:

content = f.read().split()

return content

def read\_file\_r(filename):

result = []

with open(filename, 'r') as f:

for line in f:

line = line.strip('\n').split(',')

result.append(line[1])

return result

##############################################################################

# discrimitive

##############################################################################

def compute\_y (w0, phi):

a = np.dot(w0, phi)

return 1 / (1 + np.exp(-a))

# compute R

def compute\_R(y):

r = map(lambda x: x \* (1 - x), y)

return np.diagflat(r)

# compute w

def compute\_w(phi, phi\_tranpose, R, y, t, w0):

global alpha\_prime

N = len(phi[0])

p1 = inv(np.dot(alpha\_prime, np.identity(N)) + np.dot(np.dot(phi\_tranpose, R), phi))

p2 = np.dot(phi\_tranpose, (y-t)) + np.dot(alpha\_prime, w0)

w1 = w0 - np.dot(p1, p2)

return w1

def compute\_w\_sN(data, data\_label, tr\_index):

global alpha\_prime

phi = []

t = []

for i in tr\_index:

phi.append(data[i])

t.append(data\_label[i])

t = np.array(t).astype(float)

d = len(phi[0])

N = len(phi)

# add feature in the last row

phi\_p = np.ones((N, d+1))

phi\_p[:,:-1] = phi

phi = phi\_p

w0 = [0] \* (d + 1)

phi\_tranpose = np.transpose(phi)

y = compute\_y(w0, phi\_tranpose)

R = compute\_R(y)

# compute w1

w1 = compute\_w(phi, phi\_tranpose, R, y, t, w0)

sum1 = 1

n = 1

while n < 100 and sum1 >= 10 \*\* (-3):

w0 = w1

y = compute\_y(w0, phi\_tranpose)

R = compute\_R(y)

w1 = compute\_w(phi, phi\_tranpose, R, y, t, w0)

sum1 = sum(np.square(np.subtract(w1, w0))) / sum(np.square(w0))

n += 1

y = compute\_y(w1, phi\_tranpose)

R = compute\_R(y)

s = np.dot(np.dot(phi\_tranpose, R), phi)

s0 = inv(np.identity(d + 1) / alpha\_prime)

sN = inv(s0 + s)

return w1, sN

# compute error

def compute\_accu(data, data\_label, index\_test, w\_map, sN):

acc = 0

for te\_i in index\_test:

temp = data[te\_i].tolist()

temp.append(1)

d = np.array(temp)

ua = np.dot(w\_map, d)

sig\_s = np.dot(np.dot(d, sN), d)

a = ua / math.sqrt(1 + np.pi \* sig\_s / 8)

if a >= 0:

if data\_label[te\_i] == '1':

acc += 1

else:

if data\_label[te\_i] == '0':

acc += 1

return acc / float(len(index\_test))

#####################

def main():

filename = range(1, 201)

K = 20

N\_iters = 500

data = []

doc\_len = []

# array of document indices d\_n

d\_n = []

# array of initial topic indices z\_n

z\_n = []

for i in filename:

res = read\_file(str(i))

doc\_len.append(len(res))

temp = [str(i)] \* len(res)

d\_n += temp

data.append(res)

# array of words indices w(n)

words = [w for d in data for w in d]

print "number of words:"

print len(words)

vocab = list(set(words))

vocab = sorted(vocab)

print "vocab length"

print len(vocab)

word\_indices = OrderedDict()

for v in vocab:

word\_indices[v] = vocab.index(v)

#print word\_indices

w\_n = []

for word in words:

w\_n.append(word\_indices[word])

# total number of N\_words

N\_words = len(w\_n)

# array of initial topic indices

for i in range(N\_words):

z\_n.append(str(random.choice(range(1, K+1))))

V = len(vocab)

alpha = float(50) / float(K)

alpha\_1 = alpha \* np.ones(K)

beta = 0.1

beta\_1 = beta \* np.ones(V)

# random permutation of N\_words

pi\_n = np.random.permutation(range(0, N\_words))

# initialize a D \* K matrix C\_d

D = len(filename)

C\_d = []

i = 0

k = 0

while i < N\_words:

j = i

temp = [0] \* K

end = j + doc\_len[k]

while j < end:

temp[int(z\_n[j]) - 1] += 1

j += 1

C\_d.append(temp)

i = j

k += 1

# initialize a K \* V matrix C\_t

C\_t = []

for i in range(K):

temp = [0] \* V

C\_t.append(temp)

for i in range(N\_words):

topic = z\_n[i]

topic\_index = int(topic) - 1

word\_index = w\_n[i]

C\_t[topic\_index][word\_index] += 1

# initialize a 1 \* K array of probabilities P (to zero)

P = [0] \* K

# step 5

for i in range(N\_iters):

print i

for n in range(N\_words):

index = pi\_n[n]

word = w\_n[index]

topic = z\_n[index]

doc = d\_n[index]

C\_d[int(doc) - 1][int(topic) - 1] -= 1

C\_t[int(topic) - 1][word] -= 1

for k in range(K):

p\_1 = (C\_t[k][word] + beta) / (V \* beta + sum(C\_t[k]))

p\_2 = (C\_d[int(doc)-1][k] + alpha) / (K \* alpha + sum(C\_d[int(doc) - 1]))

P[k] = p\_1 \* p\_2

# normalize P

total = sum(P)

for k in range(K):

P[k] /= float(total)

#print C\_t

r = random.uniform(0,1)

for k in range(K):

if r >= sum(P[:k]) and r <= sum(P[:k+1]):

topic = str(k+1)

break

z\_n[index] = topic

C\_d[int(doc) - 1][int(topic) - 1] += 1

C\_t[int(topic) - 1][word] += 1

"""

print "z\_n"

print z\_n

"""

print "C\_d and C\_t"

print C\_d

#print "C\_t"

#print C\_t

fre\_word = []

for k in range(K):

fre = []

temp = sorted(range(len(C\_t[k])), key = lambda x: C\_t[k][x])

index\_list = temp[-5:]

for index in index\_list:

for key, val in word\_indices.iteritems():

if val == index:

fre.append(key)

fre\_word.append(fre)

print fre\_word

file = open("topicwords.csv", "w")

wr = csv.writer(file, dialect = 'excel')

wr.writerows(fre\_word)

file.close()

# Task2 Classification

# step1: prepare presentation

# topic representation

for d in range(D):

for k in range(K):

C\_d[d][k] = (alpha + C\_d[d][k]) / (K \* alpha + sum(C\_d[d]))

print "C\_d representation:"

print C\_d

# bag-of-words representation

C\_b = []

for d in data:

d\_v = []

order\_vocab = OrderedDict()

for v in vocab:

order\_vocab[v] = 0

for word in d:

order\_vocab[word] += 1

for v in vocab:

d\_v.append(order\_vocab[v] / float(len(d)))

C\_b.append(d\_v)

print "C\_b representation:"

print C\_b

data\_2 = []

data\_2.append(C\_d)

data\_2.append(C\_b)

# x, store length of train file for plot use

x = []

data\_label = read\_file\_r('index.csv')

# i = 0, topic representation; i = 1, bag-of-words

error\_list = []

for i in range(2):

data = np.array(data\_2[i])

N = len(data)

x.append(N-int(N/3))

error\_list\_rep = []

index\_N = range(N)

# run 30 times

for t in range(30):

# set up 1/3 data set index

te\_N = int(N/3)

index\_test = np.random.choice(N, te\_N, replace = False)

# remain 2/3 train set

index\_train = [v for j, v in enumerate(index\_N) if j not in index\_test]

tr\_N = len(index\_train)

# Record performance

splits = np.arange(0.05, 1.05, 0.05)

#splits = np.arange(0.2, 1.2, 0.2)

error\_rate = []

for s in splits:

# train set index

tr\_index = np.random.choice(index\_train, int(s\*tr\_N), replace = False)

tr\_label = []

for l in tr\_index:

tr\_label.append(data\_label[l])

while (len(set(tr\_label)) <= 1):

tr\_index = np.random.choice(index\_train, int(s\*tr\_N), replace = False)

tr\_label = []

for l in tr\_index:

tr\_label.append(data\_label[l])

# discrimitive

w\_map, sN = compute\_w\_sN(data, data\_label, tr\_index)

# test file

accu = compute\_accu(data, data\_label, index\_test, w\_map, sN)

error\_rate.append(accu)

error\_list\_rep.append(error\_rate)

error\_list.append(error\_list\_rep)

topic\_error\_list = error\_list[0]

print len(topic\_error\_list)

vocab\_error\_list = error\_list[1]

print "len vocab error list"

print len(vocab\_error\_list)

topic\_mean = np.mean(topic\_error\_list, axis = 0)

topic\_std = np.std(topic\_error\_list, axis = 0)

print "topic mean"

print topic\_mean

vocab\_mean = np.mean(vocab\_error\_list, axis = 0)

vocab\_std = np.std(vocab\_error\_list, axis = 0)

print "vocab mean"

print vocab\_mean

print "start to plot:"

splits = np.arange(0.05, 1.05, 0.05)

plt.figure(1)

x1 = x[0] \* splits

plt.errorbar(x1, topic\_mean, topic\_std, marker = '^')

plt.errorbar(x1, vocab\_mean, vocab\_std, marker = '\*')

plt.title('performance of two representaion')

plt.xlabel('train size')

plt.ylabel('accuracy')

plt.legend(['topic representaion ', 'bag-of-words'], loc = 0)

plt.savefig("task2\_3.png")

plt.clf()

main()