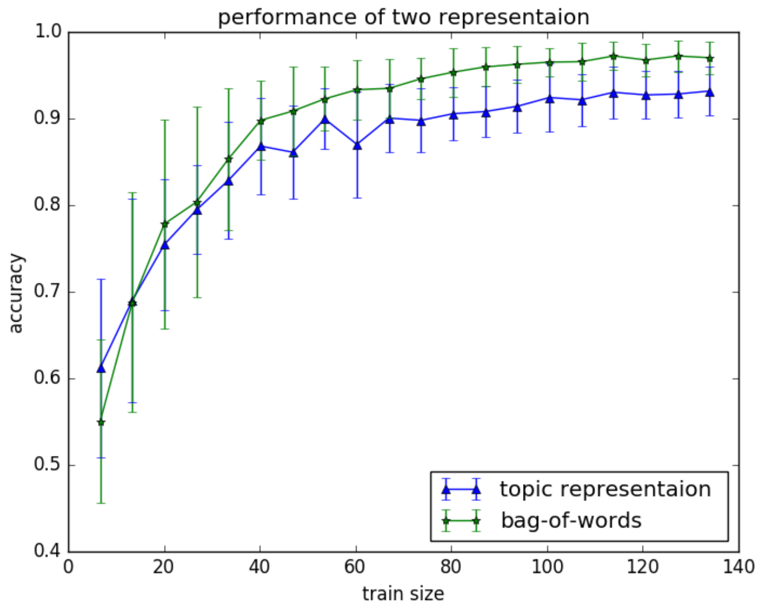
COMP136 PP4 Report

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()
```