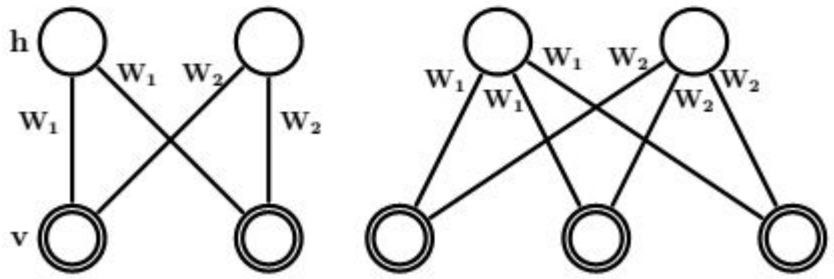
Tutorial on Replicated Softmax Model

Wen Tang (wtang6@ncsu.edu)

Introduce of RSM

Replicated Softmax Model is a two-layer architecture network. One is visible layer and another is hidden layer. The visible layer is like a restricted Boltzmann machine. It is a binary matrix, with v=1 if visible unit is on. The hidden unit is the binary stochastic hidden unit. The connections only exist between visible layer and hidden layers, which looks as follows:



Since v and h are binary units, we define the energy on the state of $\{\mathbf{V}, \mathbf{h}\}$ as follows:

$$E(\mathbf{V}, \mathbf{h}) = -\sum_{i=1}^{D}\sum_{j=1}^{F}\sum_{k=1}^{K} W_{ij}^{k} h_j v_i^k - \sum_{i=1}^{D}\sum_{k=1}^{K} v_i^k b_i^k - \sum_{j=1}^{F} h_j a_j,$$

Where w is the weights between these two layers, a is the bias of hidden layer and b is the bias of visible layer.

As we need to calculate the probabilities of the v and h, we could use exponent to them to keep the value being positive and then normalize them by the sum of all exponent values. Thus we could get the probabilities as follows:

$$P(\mathbf{V}) = \frac{1}{\mathcal{Z}}\sum_{\mathbf{h}} \exp\left(-E(\mathbf{V}, \mathbf{h})\right), \quad \mathcal{Z} = \sum_{\mathbf{V}}\sum_{\mathbf{h}} \exp\left(-E(\mathbf{V}, \mathbf{h})\right),$$

$$p(v_i^k = 1|\mathbf{h}) = \frac{\exp\left(b_i^k + \sum_{j=1}^{F} h_j W_{ij}^k\right)}{\sum_{q=1}^{K} \exp\left(b_i^q + \sum_{j=1}^{F} h_j W_{ij}^q\right)}$$

$$p(h_j = 1|\mathbf{V}) = \sigma\left(a_j + \sum_{i=1}^{D}\sum_{k=1}^{K} v_i^k W_{ij}^k\right),$$

$$\sigma(x) = 1/(1+\exp(-x))$$

For one visible unit we have above formulas, if we share the same set of weights for several visible units, we can change the energy of the state of $\{\mathbf{V}, \mathbf{h}\}$ to be:

$$E(\mathbf{V},\mathbf{h}) = -\sum_{j=1}^{F}\sum_{k=1}^{K} W_j^k h_j \hat{v}^k - \sum_{k=1}^{K} \hat{v}^k b^k - D\sum_{j=1}^{F} h_j a_j,$$

$$\hat{v}^k = \sum_{i=1}^{D} v_i^k$$

In order to get the max probability of visible vectors, the target of RSM is to maximize the log-likelihood of P(Vn). So we take the derivative of log-likelihood with the respect to parameters W:

$$\frac{1}{N}\sum_{n=1}^{N} \frac{\partial \log P(\mathbf{V_n})}{\partial W_j^k} = \mathrm{E}_{P_{\mathrm{data}}}\left[\hat{v}^k h_j\right] - \mathrm{E}_{P_{\mathrm{Model}}}\left[\hat{v}^k h_j\right],$$

where E[.] is the expectation, Pdata is the distribution from the empirical distribution, when Pmodel is the distribution defined by RSM. To avoid learn the model, learning is done by following an approximation to the gradient, called “Contrastive Divergence”(CD):

$$\Delta W_j^k = \alpha\left(\mathrm{E}_{P_{\mathrm{data}}}\left[\hat{v}^k h_j\right] - \mathrm{E}_{P_T}\left[\hat{v}^k h_j\right]\right),$$

where α is the learning rate and P_T represents a distribution defined by running the Gibbs chain.

CD-t is update W with repeat update {v,h} for t times. Usually, CD1 could already get a good result. But sometimes CD1 will stuck on local minimum. So we could train the model by add it to CD5 to go to a better optimal result. The theory is that in a lower temperature, the probability of particle going into a lower minimum place is much higher than the probability of particle going from a lower minimum place to a higher minimum place. The lower the temperature is, the more times update on CD.

Experiments on RSM

The codes of each functions such as preprocess, experiment comparison, etc. are in other file. Here just explain the idea of the procedures of using the codes to experiment on RSM, not details of the functions.

Mode set:
Here we could regard v as a D x K matrix. D is the document size, K is the dictionary size. h is the latent topic in the topic model.

Data: 20-newsgroup-bydate
Training Data: 11314 documents, Test Data: 7532 documents

1. RSM train and test

```
#import the libraries you need
import rsm_numpy
import preprocess as ps
import ppl
import dsl
```

1.1 Read the documents from file and tokenize them. Remove the stopwords and punctuations in them and get the stemmed tokens. Also return the labels of each document.

```
#path of trainning data
path_train='20news-bydate-train'

#preprocess the data
#tokenize, stemming, remove stop words and punctuations
text,train_label=ps.data_perprocess(path_train)
```

1.2 Choose the K most frequent words to represent the document as the doc-word matrix.

```
#top_k is the first k highest frequency tokens which would be chosen as the word vectors
top_k=2000
#get the frequent part of the data
text_high=ps.frequent_part(text,top_k)

#get the document-word frequency matrix
dictionary,token_id=ps.dictionary_count(text_high)
corpus=ps.corpus(dictionary,text)
train=ps.word_document(corpus,token_id)
```

1.3 Set the parameters and train RSM.

```
# hyperparameters
hiddens = 50
batch = 100
epochs = 1000
rate = 0.0001
iter=1

#train the RSM model by iter=1
RSM = rsm_numpy.RSM()
result = RSM.train(train, hiddens, epochs, iter, lr=rate, btsz=batch)
#save the result of the RSM_CD1
dsl.save(result,'result/rsm_result_1')
```

1.4 Change the iterations of updating the parameters, i.e, CD-X.

```
#set iterations=5, i.e., CD-5
iter=5

#train the RSM model by iter=5
RSM = rsm_numpy.RSM()
result = RSM.train(train, hiddens, epochs, iter, lr=rate, btsz=batch)
dsl.save(result,'result/rsm_result_5')
```

1.5 Preprocess on test data and use trained model to calculate the perplexity.

```
#path of test data
path_test='20news-bydate-test'

#perprocess the test data
test,test_label=ps.data_perprocess(path_test)
#get test word-document matrix
corpus_test=ps.corpus(dictionary,test)
test=ps.word_document(corpus_test,token_id)

#calculate ppl for test
#load the rsm_1 model from disk
result=dsl.load('result/rsm_result_5')
w_vh=result['w_vh']
w_v=result['w_v']
w_h=result['w_h']
# return the perplexity which is to assess the topic model
Eppl_CD5=ppl.rsmppl(w_v,w_h,w_vh,test)
print("Eppl_CD5=",Eppl_CD5)

#load the rsm_1 model from disk
result=dsl.load('result/rsm_result_1')
w_vh=result['w_vh']
w_v=result['w_v']
w_h=result['w_h']
Eppl_CD1=ppl.rsmppl(w_v,w_h,w_vh,test)
print("Eppl_CD1=",Eppl_CD1)
```

The result is:

E(ppl_CD1)= 613.37307070024747

E(ppl_CD5)= 642.94004342193978

Perplexity Formula:

$$\exp\left(-1/N \sum_{n=1}^{N} 1/D_n \log p(\mathbf{v}_n)\right)$$

2. Compare Perplexity of LDA and RSM

```
#import the libraries you need
from __future__ import division
from gensim import corpora, models, similarities
import logging
logging.basicConfig(format='%(asctime)s:%(levelname)s:%(message)s',level=log
ging.INFO)
import dsl,ppl
import perprocess as ps
import matplotlib.pyplot as plt
import numpy as np
import lda
import analysis as ana
```

```
# When use LDA we need change the matrix type as int64
train=np.int64(train)
test=np.int64(test)
```

2.1 Train the LDA model

```
#train the LDA model
print("-------------------LDA GET Training--------------------")
model=lda.LDA(n_topics=50,n_iter=2000,random_state=1)
model.fit(train)
#get the topic_word distribution and doc_topic distribution.
topic_word=model.components_
doc_topic=model.doc_topic_
#save the these data
dsl.save(topic_word,'result/topic_word')
dsl.save(model,'result/lda_model')
dsl.save(doc_topic,'result/doc_topic')
print("-------------------LDA Model Has Been Saved--------------------")
```

2.2 Sample 50 held out documents from test set

```
#sample the held document from the test data
sample=50
sample_id=np.random.randint(7532,size=(50,sample))
```

2.3 Calculate the Perplexity of LDA model

```
#Since the doc-topic distribution is different for each document, we need to
#calculate it for each test document

#calculte the ppl of lda model
ppl_lda=[]
for i in xrange(sample):
    test_sample=test[sample_id[i]]
    doc_topic_test=model.transform(test_sample,max_iter=1000)
    pdf=np.dot(doc_topic_test,topic_word)
    z = np.nansum(test_sample * np.log(pdf))
    s = np.sum(test_sample)
    pplt = np.exp(- z / s)
    ppl_lda.append(pplt)
```

2.4 Load trained RSM model and return the Perplexity

```
#load the rsm_5 model from disk
result=dsl.load('result/rsm_result_5')
w_vh=result['w_vh']
w_v=result['w_v']
w_h=result['w_h']

#calculate the ppl of rsm_5 model
```

```
ppl_rsm_5=[]
for i in xrange(sample):
    test_sample=test[sample_id[i]]
    pplt=ppl.rsmppl(w_v,w_h,w_vh,test_sample)
    ppl_rsm_5.append(pplt)

ppl_rsm_5=np.array(ppl_rsm_5)
dsl.save(ppl_lda,'result/ppl_rsm_5')


#load the rsm_1 model from disk
result=dsl.load('result/rsm_result_1')
w_vh=result['w_vh']
w_v=result['w_v']
w_h=result['w_h']


#calculate the ppl of rsm_1 model
ppl_rsm_1=[]
for i in xrange(sample):
    test_sample=test[sample_id[i]]
    pplt=ppl.rsmppl(w_v,w_h,w_vh,test_sample)
    ppl_rsm_1.append(pplt)
ppl_rsm_1=np.array(ppl_rsm_1)
dsl.save(ppl_lda,'result/ppl_rsm_1')
```

2.5 Draw the figures and save it to the disk

```
#plot the figures of compare the ppl of rsm and lda
plt.plot(ppl_rsm_5,ppl_lda,'ro')
plt.plot(ppl_rsm_1,ppl_lda,'bo')
x_min=np.floor(np.min([ppl_rsm_5.min(),ppl_rsm_1.min(),ppl_lda.min()])*0.9)
x_max=np.ceil(np.max([ppl_rsm_5.max(),ppl_rsm_1.max(),ppl_lda.max()])*1.1)
x_axis=np.linspace(x_min,x_max,1000)
plt.plot(x_axis,x_axis,'k')
plt.axis([x_min,x_max,x_min,x_max])
plt.xlabel('RSM')
plt.ylabel('LDA')
plt.title('20 news perplexity')
plt.savefig('result/perplexity.png')
```

The result is :

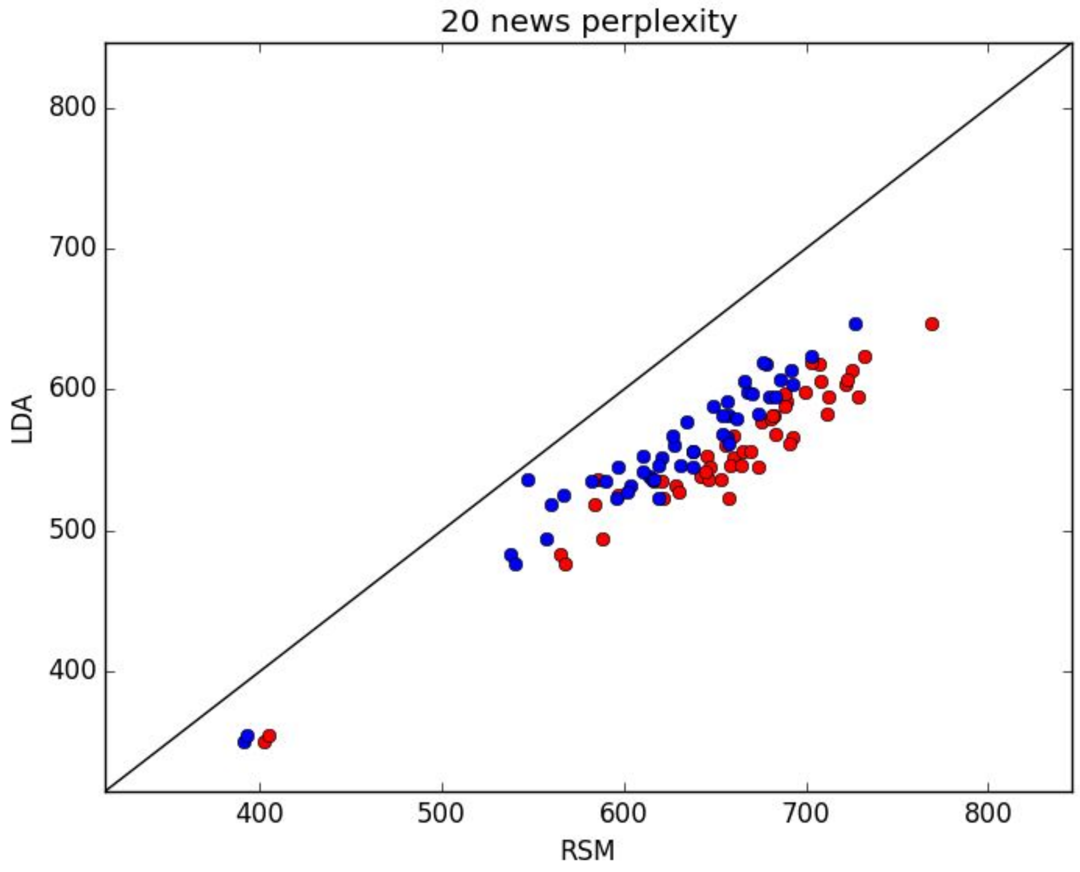


Figure 1. Perplexity of RSM and LDA
The blue point is update RSM by CD1, and the red point is CD5.

Analysis:

Perplexity is one way to evaluate the ability of the topic modeling. From the Figure 1, we can see LDA is still a few better than RSM, but it depends on three main reasons to compare the perplexity. First of all, the parameters of two different models. It is hard to recover the exact the experiments in the paper, because we do not know the parameters in the LDA. Gibbs in LDA is also a sampling way. So different iterations to sample the Gibbs will also affect the result of the doc-topic distribution, which is direct value used in computing the perplexity. Second, in the paper, they use the LDA model by the Gibbs sampling implementation of the Matlab Topic Modeling Toolbox. But here we use the Python LDA library instead. And the formula of perplexity in their paper is unknown, maybe it is much different with the one I used. Last, it is because the iterations of RSM model is different with them used in the paper. They used a changeable way to change the CD from 1 to 5. However, we just fixed CD with 1 or 5. That’s why we may not obtain such a good result with RSM.

3. Query a Document and Retrieve the similar documents by LDA and RSM

3.1 Get the labels and threshold of retrieving

```
# retrieve K similar documents from database
# k starts from 1 ends in the length of training data, step=50
k=np.hstack((np.arange(1,100,10),np.arange(101,train_label.shape[0]-1,50)))
k=np.hstack((k,train_label.shape[0]-1))
```

3.2 Map the doc-word matrix into LDA space and use the latent topics to calculate the cosine similarities. Obtain the precision and recall of LDA. Here reduce the word-vectors from 2000 to 50 dimension.

```
#use topics to represent the documents
#here in LDA is 50 topics
train_topics=doc_topic
test_topics=model.transform(test,max_iter=1000)
dsl.save(test_topics,'result/test_topics')
#train_topics=dsl.load('doc_topic')
#test_topics=dsl.load('test_topics')
train_topics_sort=np.argsort(train_topics)
test_topics_sort=np.argsort(test_topics)

#calculate the cosines between each query documents and training data
#get the predict label of the query documents
lda_perdict_label,lda_cosine=ana.perdict_label(train_topics_sort,test_topics_sort,t
rain_label)

#return the precision and recalls for retrieving k similar documents
lda_p,lda_r=ana.precision_recall(lda_perdict_label,train_label,test_label,k)

#save the data
dsl.save(lda_cosine,'result/lda_cosine')
dsl.save(lda_p,'result/lda_precision')
dsl.save(lda_r,'result/lda_recall')
```

3.3 Project the doc-word matrix into RSM space and get the hidden vectors as the latent topics to calculate the cosine similarities. Return the precision and recall of RSM. Here reduce the word vectors from 2000 to 50 dimension.

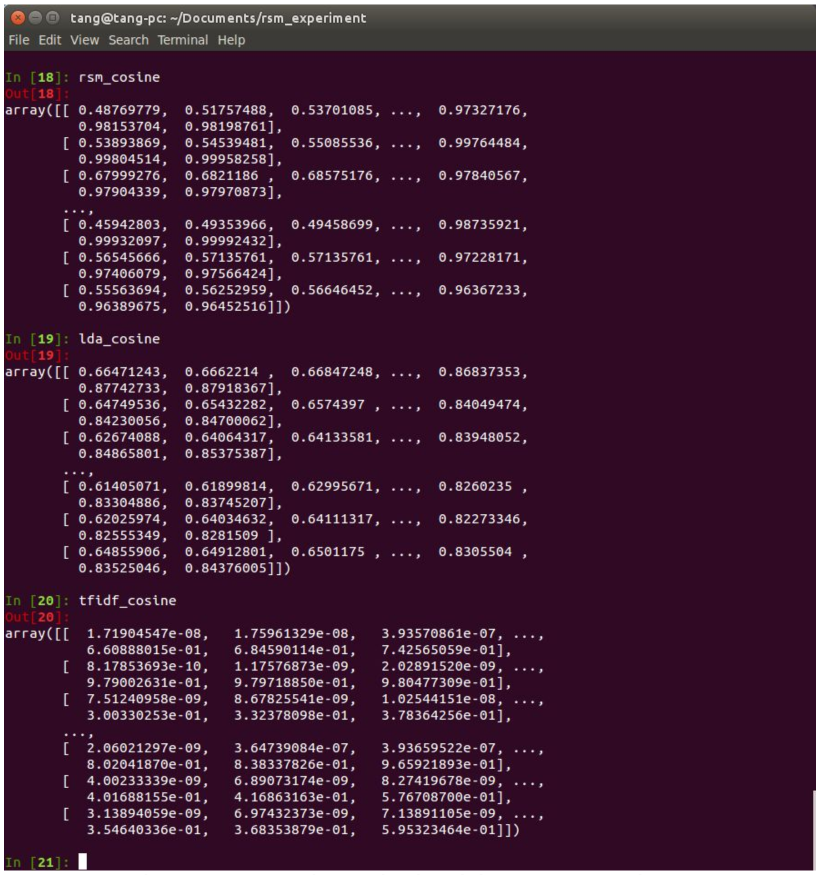
```
# use the hidden vectors to reperesent the documents
# here in RSM is 50 dimensions
h_train=ppl.rsm_hidden(w_v,w_h,w_vh,train)
```

```
h_test=ppl.rsm_hidden(w_v,w_h,w_vh,test)

# calculate the similarity, precision and recall of RSM
rsm_perdict_label, rsm_cosine=ana.perdict_label(h_train,h_test,train_label)
rsm_p,rsm_r=ana.precision_recall(rsm_perdict_label,train_label,test_label,k)

dsl.save(rsm_cosine,'result/rsm_cosine')
dsl.save(rsm_p,'result/rsm_precision')
dsl.save(rsm_r,'result/rsm_recall')
```

3.4 Calculate the TF-IDF of the matrix to change the weights of each word. Get the cosine similarities to retrieve the documents and compute the precision and recall of TF-IDF. Here the dimension of word vector is still 2000.

```
# use tfidf to represent the documents
# here still keep 2000 verbs
# calculate the similarity, precision and recall of tfidf
train_tfidf=ps.tfidf(train)
test_tfidf=ps.tfidf(test)
tfidf_perdict_label, tfidf_cosine=ana.perdict_label(train_tfidf,test_tfidf,train_label)
tfidf_p,tfidf_r=ana.precision_recall(tfidf_perdict_label,train_label,test_label,k)

dsl.save(tfidf_cosine,'result/tfidf_cosine')
dsl.save(tfidf_p,'result/tfidf_precision')
dsl.save(tfidf_r,'result/tfidf_recall')
```

The result is:


tang@tang-pc: ~/Documents/rsm_experiment
File Edit View Search Terminal Help

```
In [18]: rsm_cosine
Out[18]:
array([[ 0.48769779,  0.51757488,  0.53701085, ...,  0.97327176,
         0.98153704,  0.98198761],
       [ 0.53893869,  0.54539481,  0.55085536, ...,  0.99764484,
         0.99804514,  0.99958258],
       [ 0.67999276,  0.6821186 ,  0.68575176, ...,  0.97840567,
         0.97904339,  0.97970873],
       ...,
       [ 0.45942803,  0.49353966,  0.49458699, ...,  0.98735921,
         0.99932097,  0.99992432],
       [ 0.56545666,  0.57135761,  0.57135761, ...,  0.97228171,
         0.97406079,  0.97566424],
       [ 0.55563694,  0.56252959,  0.56646452, ...,  0.96367233,
         0.96389675,  0.96452516]])

In [19]: lda_cosine
Out[19]:
array([[ 0.66471243,  0.6662214 ,  0.66847248, ...,  0.86837353,
         0.87742733,  0.87918367],
       [ 0.64749536,  0.65432282,  0.6574397 , ...,  0.84049474,
         0.84230056,  0.84700062],
       [ 0.62674088,  0.64064317,  0.64133581, ...,  0.83948052,
         0.84865801,  0.85375387],
       ...,
       [ 0.61405071,  0.61899814,  0.62995671, ...,  0.8260235 ,
         0.83304886,  0.83745207],
       [ 0.62025974,  0.64034632,  0.64111317, ...,  0.82273346,
         0.82555349,  0.8281509 ],
       [ 0.64855906,  0.64912801,  0.6501175 , ...,  0.8305504 ,
         0.83525046,  0.84376005]])

In [20]: tfidf_cosine
Out[20]:
array([[  1.71904547e-08,   1.75961329e-08,   3.93570861e-07, ...,
          6.60888015e-01,   6.84590114e-01,   7.42565059e-01],
       [  8.17853693e-10,   1.17576873e-09,   2.02891520e-09, ...,
          9.79002631e-01,   9.79718850e-01,   9.80477309e-01],
       [  7.51240958e-09,   8.67825541e-09,   1.02544151e-08, ...,
          3.00330253e-01,   3.32378098e-01,   3.78364256e-01],
       ...,
       [  2.06021297e-09,   3.64739084e-07,   3.93659522e-07, ...,
          8.02041870e-01,   8.38337826e-01,   9.65921893e-01],
       [  4.00233339e-09,   6.89073174e-09,   8.27419678e-09, ...,
          4.01688155e-01,   4.16863163e-01,   5.76708700e-01],
       [  3.13894059e-09,   6.97432373e-09,   7.13891105e-09, ...,
          3.54640336e-01,   3.68353879e-01,   5.95323464e-01]])

In [21]:
```

Figure 2: Cosine Similarities Comparison

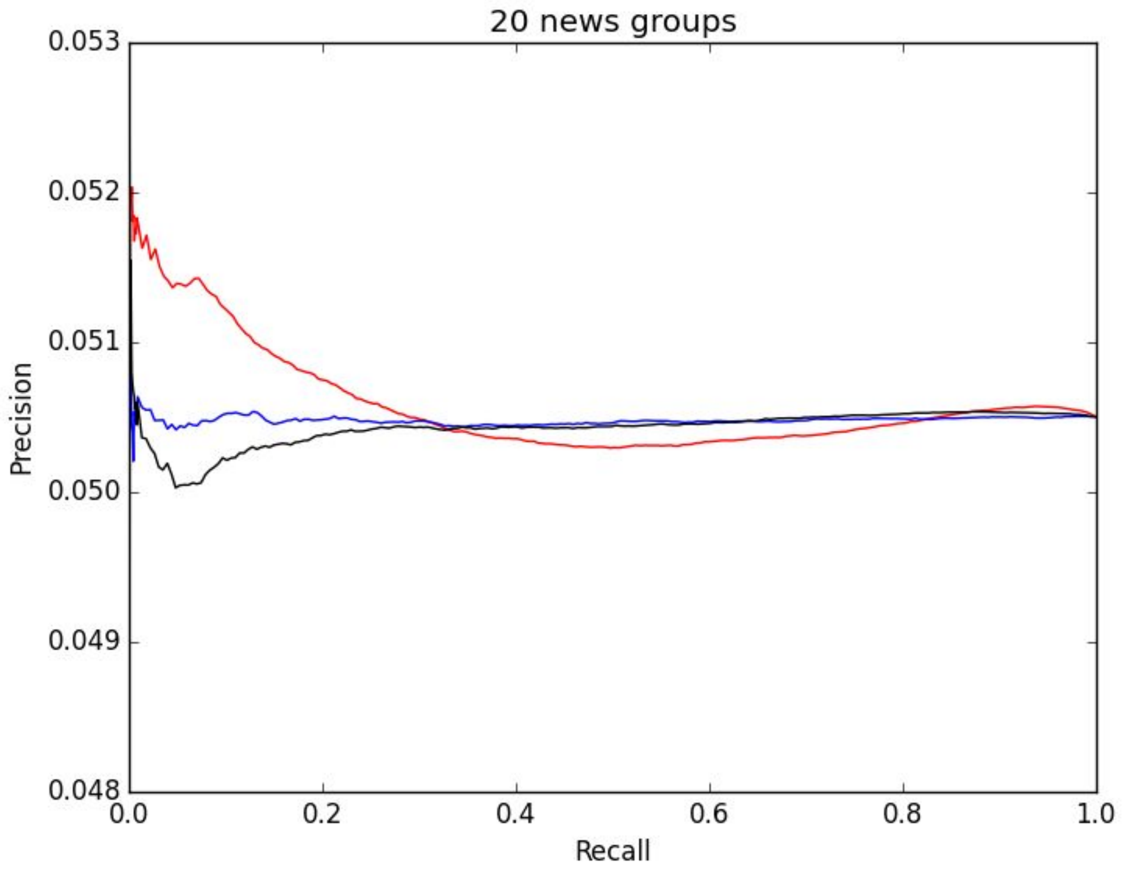


Figure 3: Precision and Recall of RSM, LDA, TFIDF
The red line is RSM, the blue one is LDA, and the black one is TF-IDF

<u>Analysis:</u>

When querying a document, in Figure 2, we could see the cosine of RSM is higher than LDA, and we notice the range of cosine in the RSM is lager than the LDA. That means, in the RSM space, the similar vectors are more closer to each other, while the dissimilar vectors have a longer distances to others. So the precision is obviously higher than LDA, which we can see in Figure 3. Both of LDA and RSM comparing to the TF-IDF have much higher precision, because the cosine similarities is very low in TF-IDF.

The precision and recall is average over all the 7352 test documents. It is much different with the results in paper. Perhaps, first, the preprocssing procedures are different, especially in the stemming step. And then, the sort algorithm is used to pick the most k frequent tokens. There are several tokens have the same frequency, so different sort algorithms will pick different tokens to be a part of the word vectors.

Reference

[1] Ruslan Salahutdinov, Geoffrey Hinton, “Replicated Softmax: an Undirected Topic Model”, 2009
[2] Geoffery Hinton, “Neural Network for Machine Learning”, Lecture 1-2, 11-12, 2012. (http://class.coursera.org/neuralnets-2012-001/lecture)
[3] http://www.52nlp.cn/
[4] http://pythonhosted.org/lda/
[5] http://docs.scipy.org/doc/numpy/reference/
[6] http://matplotlib.org/users/pyplot_tutorial.html