```
from numpy import zeros, int8, log
from pylab import random
import sys
#import jieba
import nltk
from nltk.tokenize import word_tokenize
import re
import time
import codecs
# N is # of of document
# K is # of topic
# M is # of word
# beta is probablity of word given a topic
# theta is probablity of a topic given a document
# document- word matrix, N x M : word count in a document
class PLSA(object):
    def initialize(self, N, K, M, word2id, id2word, X):
        self.word2id, self.id2word, self.X = word2id, id2word, X
        self.N, self.K, self.M = N, K, M
        # theta[i, j] : p(zj|di): 2-D matrix
        self.theta = random([N, K])
        # beta[i, j] : p(wj|zi): 2-D matrix
        self.beta = random([K, M])
        # p[i, j, k] : p(zk|di,wj): 3-D tensor
        self.p = zeros([N, M, K])
        for i in range(0, N):
            normalization = sum(self.theta[i, :])
            for j in range(0, K):
                self.theta[i, j] /= normalization;

        for i in range(0, K):
            normalization = sum(self.beta[i, :])
            for j in range(0, M):
                self.beta[i, j] /= normalization;


    def EStep(self):
        for i in range(0, self.N):
            for j in range(0, self.M):
                ## ================== YOUR CODE HERE
==========================
                ###  for each word in each document, calculate its
                ###  conditional probability belonging to each topic (update
p)
                denominator = 0
                for k in range(0, self.K):
                    self.p[i, j, k] = self.theta[i, k] * self.beta[k, j]
                    denominator += self.p[i, j, k]
                for k in range(0, self.K):
                    self.p[i, j, k] /= denominator
                #
============================================================


    def MStep(self):
        # update beta
        for k in range(0, self.K):
            # ================== YOUR CODE HERE ==========================
            ###  Implement M step 1: given the conditional distribution
            ###  find the parameters that can maximize the expected
```

```
57              denominator = 0
58              for m in range(0, self.M):
59                  self.beta[k, m] = 0
60                  for n in range(0, self.N):
61                      self.beta[k, m] += self.X[n, m] * self.p[n, m, k]
62                  denominator += self.beta[k, m]
63              for m in range(0, self.M):
64                  self.beta[k, m] /= denominator
65              # ============================================================
66
67          # update theta
68          for i in range(0, self.N):
69              # ================== YOUR CODE HERE ==========================
70              ###  Implement M step 2: given the conditional distribution
71              ###  find the parameters that can maximize the expected
    likelihood (update theta)
72              for k in range(0, self.K):
73                  self.theta[i, k] = 0
74                  denominator = 0
75                  for m in range(0, self.M):
76                      self.theta[i, k] += self.X[i, m] * self.p[i, m, k]
77                      denominator += self.X[i, m]
78                  self.theta[i, k] /= denominator
79              # ============================================================
80
81
82      # calculate the log likelihood
83      def LogLikelihood(self):
84          loglikelihood = 0
85          for i in range(0, self.N):
86              for j in range(0, self.M):
87                  # ================== YOUR CODE HERE
    ==========================
88                  ###  Calculate likelihood function
89                  temp = 0
90                  for k in range(0, self.K):
91                      temp += self.theta[i, k] * self.beta[k, j]
92                  if temp > 0:
93                      loglikelihood += self.X[i, j] * log(second_term)
94                  #
    ============================================================
95          return loglikelihood
96
97      # output the params of model and top words of topics to files
98      def output(self, docTopicDist, topicWordDist, dictionary, topicWords,
    topicWordsNum):
99          # document-topic distribution
100         file = codecs.open(docTopicDist,'w','utf-8')
101         for i in range(0, self.N):
102             tmp = ''
103             for j in range(0, self.K):
104                 tmp += str(self.theta[i, j]) + ' '
105             file.write(tmp + '\n')
106         file.close()
107
108         # topic-word distribution
109         file = codecs.open(topicWordDist,'w','utf-8')
110         for i in range(0, self.K):
111             tmp = ''
```

```
113                 tmp += str(self.beta[i, j]) + ' '
114             file.write(tmp + '\n')
115         file.close()
116 
117         # dictionary
118         file = codecs.open(dictionary,'w','utf-8')
119         for i in range(0, self.M):
120             file.write(self.id2word[i] + '\n')
121         file.close()
122 
123         # top words of each topic
124         file = codecs.open(topicWords,'w','utf-8')
125         for i in range(0, self.K):
126             topicword = []
127             ids = self.beta[i, :].argsort()
128             for j in ids:
129                 topicword.insert(0, self.id2word[j])
130             tmp = ''
131             for word in topicword[0:min(topicWordsNum, len(topicword))]:
132                 tmp += word + ' '
133             file.write(tmp + '\n')
134         file.close()
```