APPENDIX

CLASS L-LDA

```
'''
 LLDA.py

 Based on/Reference :
 https://github.com/JoeZJH/Labeled-LDA-Python
 https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-latent-dirichlet-allocation-437c81220158
'''

# Import libs
   import numpy as np
   from preprocess import Preprocess

   # Define class LLDA
   class LLDA :

      # Define constructor
      # @param float alpha : distribution var of document's topic distribution
      # @param float eta : distribution var of topic's term distribution
      def __init__(self, alpha, eta) :

         self.__alpha = alpha              # Variable distribution for controlling topic distribution in a document
         self.__eta = eta                  # Variable distribution for controlling word distribution in a topic

         self.__K = 0                      # Total topic (topic count)
         self.__M = 0                      # Total documents (document count)
         self.__T = 0                      # Total word (word count/term count)

         self.__V = []                     # Word dictionary
         self.__topic_V = []               # Topic dictionary

         self.__D = []                     # Document corpus, collection of documents

         self.__L = []                     # Lambda, shows the topic occurence inside documents
         self.__alpha_D = []               # alpha matrix, alpha x Lambda, used to constrain the doc topic distribution
         self.__eta_D = []                 # eta matrix

         self.__document_topic_matrix = []  # Document to topic matrix
```

```
38.             self.__topic_term_matrix = []       # Topic to term
  matrix
39.
40.             self.__preprocesser = Preprocess() # Define
  preprocessor object
41.
42.         # Create corpus
43.         # Used for creating corpus (D), collection of documents
44.         # @param array documents : Array of documents, in this
  case document is a text
45.         # @return array response
46.         def create_corpus(self, documents) :
47.
48.             # Create response for indicating if the process is
  ended successfully
49.             response = {
50.                 'error' : True,
51.                 'message' : ''
52.             }
53.
54.             # Try to create the corpus
55.             try :
56.
57.                 # Get the total of documents loaded
58.                 self.__M = len(documents)
59.
60.                 # Reset the T and K values to zero
61.                 self.__T = 0
62.                 self.__K = 0
63.
64.                 # Used to store document labels that will be
  used to set Lambda
65.                 d_labels = []
66.
67.                 # Creating dictionary (vocabulary) of word and
  topic
68.                 # and creating corpus
69.                 # For each document (d) in documents
70.                 for d in documents :
71.
72.                     # Preprocess document
73.                     preprocessed_d =
  self.__preprocesser.preprocess(d['text'])
74.                     d_label = []
75.                     d_words = []
76.
77.                     # For each term or word in the document
78.                     for t in preprocessed_d :
79.
80.                         # Check if the word or term is inside in
  the vocabulary
81.                         # If it is not, add them to the
  dictionary
82.                         if t not in self.__V :
```

```
83.
84.                             # Giving each term an id
85.                             self.__V.append(t)
86.                             self.__T += 1
87.
88.                         # Replace words or terms inside document
  by term id
89.                         d_words.append({
90.                             'term' : self.__V.index(t),
91.                             'topic' : 0
92.                         })
93.
94.                     # Add to the corpus
95.                     self.__D.append(d_words)
96.
97.                     # Adding common topic to every document
98.                     # Remove common if you want to do the other
  variant or approach (no common topic)
99.                     current_label = ['common']
100.                    # current_label = d['label']
101.
102.                    # For each label or topic inside documents
103.                    for label in current_label + d['label'] :
104.
105.                        # If label or topic is not inside the
  topic vocabulary
106.                        # add the label or topic
107.                        if label not in self.__topic_V :
108.
109.                            # Giving each label or topic an id
110.                            self.__topic_V.append(label)
111.                            self.__K += 1
112.
113.                        # Replace label or topic inside document
  by topic id
114.
  d_label.append(self.__topic_V.index(label))
115.
116.                    # Add them to d_labels that will later will
  be used to getting Lambda value
117.                    d_labels.append(d_label)
118.
119.                # Creating Lambda
120.                self.__L = np.zeros((self.__M, self.__K),
  dtype=int)
121.
122.
123.                # Updating the Lambda matrix values based from
  the loaded documents
124.                row = 0;
125.                for labels in d_labels :
126.
127.                    for label in labels :
128.
```

```
129.                         self.__L[row][label] = 1
130.
131.                     row += 1
132.
133.
134.                 # Lambda x alpha
135.                 self.__alpha_D = self.__L * self.__alpha
136.
137.                 # Creating eta_D
138.                 self.__eta_D = np.ones((self.__K, self.__T),
  dtype=float)
139.                 self.__eta_D = self.__eta_D * self.__eta
140.
141.                 # Create success message
142.                 response = {
143.                     'error' : False,
144.                     'message' : 'Model created successfully.'
145.                 }
146.             except Exception as e :
147.                 # If something went wrong
148.                 # Create error response
149.                 response = {
150.                     'error' : True,
151.                     'message' : 'Error : ' + str(e)
152.                 }
153.
154.             return response
155.
156.
157.         # Display attributes
158.         # Used to display the attributes of the model
159.         # @return void
160.         def display_attributes(self) :
161.
162.             print('/**')
163.             print(' Display Attributes')
164.             print(' Inside your current LLDA model : ')
165.             print()
166.             print(' alpha =', self.__alpha)
167.             print(' eta =', self.__eta)
168.             print()
169.             print(' K =', self.__K)
170.             print(' M =', self.__M)
171.             print(' T =', self.__T)
172.             print()
173.
174.             print(' D = ')
175.             print(' {:<8} {:<10} {:<10}'.format('doc_id',
  'term_id', 'topic_id'))
176.             i = 0
177.             for d in self.__D :
178.                 for w in d:
179.                     print(' {:<8} {:<10} {:<10}'.format((i + 1),
  w['term'], w['topic']))
```

```
180.                 i += 1
181.                 print()
182.
183.             print()
184.             print(' V =')
185.             print(' {:<8} {:<20}'.format('id','term'))
186.             for i in range(len(self.__V)) :
187.                 print(' {:<8} {:<20}'.format(i, self.__V[i]))
188.             print()
189.
190.             print(' topic_V = ')
191.             print(' {:<8} {:<20}'.format('id','topic'))
192.             for i in range(len(self.__topic_V)) :
193.                 print(' {:<8} {:<20}'.format(i,
  self.__topic_V[i]))
194.             print()
195.             print(' Lambda = ')
196.             print(self.__L)
197.             print()
198.             print(' alpha_D = ')
199.             print(self.__alpha_D)
200.             print(' eta_D = ')
201.             print(self.__eta_D)
202.             print('**/')
203.
204.         # Prepare training
205.         # 2 big idea of LDA (in this case LLDA)
206.         # 'Document consists of the distribution of topics'
207.         # 'Topic consists of the distribution of terms'
208.         # Preparing training the model by creating document
  topic matrix and topic term matrix
209.         # @return array response
210.         def __prepare_training(self) :
211.
212.             # Create response for indicating if the process is
  ended successfully
213.             response = {
214.                 'error' : True,
215.                 'message' : ''
216.             }
217.
218.             # Try to prepare the model for training
219.             try :
220.                 # Creating document to topic matrix, size M x K
221.                 self.__document_topic_matrix =
  np.zeros((self.__M, self.__K), dtype=int)
222.
223.                 # Creating topic to term matrix, size K x T
224.                 self.__topic_term_matrix = np.zeros((self.__K,
  self.__T), dtype=int)
225.
226.
227.                 # Add random topic to each term based on Lambda
228.                 for d in range(self.__M) :
```

```
229.
230.                 p = self.__L[d] / sum(self.__L[d])
231.                 i = 0
232.
233.                 for w in self.__D[d]:
234.
235.                     z = np.random.multinomial(1, p).argmax()
236.                     self.__D[d][i]['topic'] = z
237.                     i += 1
238.
239.
240.
241.             print(' D = ')
242.             print(' {:<8} {:<10} {:<10}'.format('doc_id',
  'term_id', 'topic_id'))
243.             i = 0
244.             for d in self.__D :
245.                 for w in d:
246.                     print(' {:<8} {:<10} {:<10}'.format((i +
  1), w['term'], w['topic']))
247.                 i += 1
248.                 print()
249.
250.             # Create the document topic matrix and topic
  term matrix
251.             i = 0
252.             for d in range(self.__M) :
253.
254.                 for j in range(len(self.__D[d])) :
255.
256.                     t = self.__D[d][j]['term']
257.                     k = self.__D[d][j]['topic']
258.
259.                     self.__document_topic_matrix[d, k] += 1
260.                     self.__topic_term_matrix[k, t] += 1
261.
262.                 i += 1
263.
264.             print(self.__document_topic_matrix)
265.             print(self.__topic_term_matrix)
266.
267.             # Create success message
268.             response = {
269.                 'error' : False,
270.                 'message' : 'Model is prepared for
  training!'
271.             }
272.
273.         except Exception as e :
274.             # If something went wrong
275.             # Create error response
276.             response = {
277.                 'error' : True,
278.                 'message' : 'Error : ' + str(e)
```

```
279.                }
280.
281.            return response
282.
283.
284.        # Begin training
285.        # Begin training the model
286.        # @param int times : How many times do you want to train
  the model
287.        # @return array response
288.        def begin_training(self, times) :
289.
290.            # Create response for indicating if the process is
  ended successfully
291.            response = {
292.                'error' : True,
293.                'message' : ''
294.            }
295.
296.            # Try to begin training the model
297.            try :
298.                # Prepare the training process
299.                preparation = self.__prepare_training()
300.
301.                if not preparation['error'] :
302.                    # If the preparation is successful
303.                    for i in range(times) :
304.
305.                        print('/**')
306.                        print(' * Iteration -', (i + 1))
307.                        print('**/')
308.                        print()
309.
310.                        # Train the model
311.                        self.__train_model()
312.
313.                        print(' D = ')
314.                        print(' {:<8} {:<10}
  {:<10}'.format('doc_id', 'term_id', 'topic_id'))
315.                        j = 0
316.                        for d in self.__D :
317.                            for w in d:
318.                                print(' {:<8} {:<10}
  {:<10}'.format((j + 1), w['term'], w['topic']))
319.                            j += 1
320.                            print()
321.
322.                    response = {
323.                        'error' : False,
324.                        'message' : 'Model is successfully
  trained!'
325.                    }
326.
327.                else :
```

```
328.                     # If something went wrong
329.                     # Create error response
330.                     response = {
331.                         'error' : True,
332.                         'message' : 'Error when try to train the
  model : ' + preparation['message']
333.                     }
334.
335.
336.             except Exception as e :
337.                 # If something went wrong
338.                 # Create error response
339.                 response = {
340.                     'error' : True,
341.                     'message' : 'Error : ' + str(e)
342.                 }
343.
344.             return response
345.
346.         # Train model
347.         # Train the model using gibbs sampling
348.         # @return void
349.         def __train_model(self) :
350.
351.             # For each document
352.             for d in range(self.__M) :
353.
354.
355.                 # We will change each z value on each word
  inside document d by using gibbs sampling
356.                 for j in range(len(self.__D[d])) :
357.
358.                     # Select the current term and topic assigned
  to the term or word
359.                     t = self.__D[d][j]['term']
360.                     k = self.__D[d][j]['topic']
361.
362.                     # decrement the count of topic k inside
  topic d and the count of term t in topic k
363.                     self.__document_topic_matrix[d, k] -= 1
364.                     self.__topic_term_matrix[k, t] -= 1
365.
366.                     # Using gibbs sampling to get a new topic
  value and then will be assigned to the term
367.                     left = (self.__topic_term_matrix[:, t] +
  self.__eta_D[k, t]) / (self.__topic_term_matrix +
  self.__eta_D[k]).sum(axis=1)
368.                     right = (self.__document_topic_matrix[d] +
  self.__alpha_D[d]) / (self.__document_topic_matrix +
  self.__alpha_D).sum(axis=1)[d]
369.                     p = (left * right/ sum(left * right))
370.                     z = np.random.multinomial(1, p).argmax()
371.
```

```
372.                    # Assign the new z (new topic) to the word
  or term
373.                    self.__D[d][j]['topic'] = z
374.
375.                    # Update the document topic matrix and topic
  term matrix
376.                    self.__document_topic_matrix[d, z] += 1
377.                    self.__topic_term_matrix[z, t] += 1
378.
379.        # Get word distributions (beta)
380.        # Get total word distributions inside all topics in
  corpus
381.        # @param boolean sort_result : True if you want to sort
  the result based from the weight
382.        # @return array response
383.        def get_word_distributions(self, sort_result) :
384.
385.            # Create response for indicating if the process is
  ended successfully
386.            response = {
387.                'error' : True,
388.                'message' : ''
389.            }
390.
391.            try:
392.
393.                # Get beta (word or term distributions inside
  topics)
394.                beta = (self.__topic_term_matrix + self.__eta_D)
  / (self.__topic_term_matrix +
  self.__eta_D).sum(axis=1).reshape(self.__K, 1)
395.
396.                # Create result array
397.                result = []
398.
399.                # Process the result after getting beta
400.                for k in range(len(beta)) :
401.
402.                    terms = []
403.
404.                    for t in range(len(beta[k])) :
405.
406.                        terms.append({
407.                            'term' : self.__V[t],
408.                            'weight' : beta[k][t]
409.                        })
410.
411.                    if sort_result:
412.                        terms = sorted(terms, key = lambda i:
  i['weight'], reverse = True)
413.
414.                    result.append({
415.                        'topic' : self.__topic_V[k],
416.                        'terms' : terms
```

```
                    })

                response = {
                    'error' : False,
                    'message' : '',
                    'data' : result
                }

            except Exception as e:
                # If something went wrong
                # Create error response
                response = {
                    'error' : True,
                    'message' : 'Error : ' + str(e)
                }

            return response

        # Get topic distributions (theta)
        # Get document topic distribution inside the corpus
        # @return array response
        def get_topic_distributions(self, sort_result) :

            # Create response for indicating if the process is
  ended successfully
            response = {
                'error' : True,
                'message' : ''
            }

            try:
                # Get theta (topic distributions in all
  documents inside corpus)
                theta = (self.__document_topic_matrix +
  (self.__alpha_D)) / (self.__document_topic_matrix +
  (self.__alpha_D)).sum(axis=1).reshape(self.__M, 1)

                # Create result array
                result = []

                # Process the result after getting theta
                for d in range(len(theta)) :

                    doc_no = (d + 1)
                    topics = []

                    for k in range(len(theta[d])) :

                        topics.append({
                            'topic' : self.__topic_V[k],
                            'weight' : theta[d][k]
                        })

                    if sort_result:
```

```
467.                     topics = sorted(topics, key = lambda i:
  i['weight'], reverse = True)
468.
469.                 result.append({
470.                     'doc' : doc_no,
471.                     'topics' : topics
472.                 })
473.
474.             response = {
475.                 'error' : False,
476.                 'message' : '',
477.                 'data' : result
478.             }
479.
480.
481.         except Exception as e:
482.             # If something went wrong
483.             # Create error response
484.             response = {
485.                 'error' : True,
486.                 'message' : 'Error : ' + str(e)
487.             }
488.
489.         return response
490.
491.     # Predict
492.     # Predict new document
493.     # @param string new_document
494.     # @return array response
495.     def predict(self, new_document) :
496.
497.         # Create response for indicating if the process is
  ended successfully
498.         response = {
499.             'error' : True,
500.             'message' : ''
501.         }
502.
503.         # Get word or term distributions
504.         word_distributions =
  self.get_word_distributions(False)
505.
506.         if not word_distributions['error'] :
507.             # If successful getting the word distribution
508.
509.             try:
510.
511.                 # Preprocess the new document
512.                 preprocessed_new_document =
  self.__preprocesser.preprocess(new_document)
513.
514.                 # Create result array
515.                 result = []
516.
```

```
517.                    # Predict new document based from the new
  document terms and word or term distribution inside model
518.                    for k in word_distributions['data'] :
519.
520.                        # If the topic is common, ignore it
521.                        if k['topic'] != 'common':
522.
523.                            print(k['topic'])
524.
525.                            total = 0
526.
527.                            # Check if term is appearing in the
  new document, multiply it with its weight
528.                            for t in k['terms'] :
529.
530.
531.                                if t['term'] in
  preprocessed_new_document :
532.                                    print(t)
533.
534.                                    total += (1 * t['weight'])
535.
536.
537.                            print()
538.
539.                            result.append({
540.                            'topic' : k['topic'],
541.                            'total' : total
542.                            })
543.
544.
545.                    # Check the result
546.                    max = {
547.                        'topic' : '',
548.                        'total' : 0
549.                    }
550.
551.                    print(result)
552.
553.                    # Getting the max total so we can conclude
  the prediction result
554.                    for re in result :
555.
556.                        if max['total'] < re['total']:
557.                            max = {
558.                                'topic' : re['topic'],
559.                                'total' : re['total']
560.                            }
561.
562.                    print(max)
563.                    print()
564.
565.                    response = {
566.                        'error' : False,
```

```
                    'message' : '',
                    'data' : max
                }

            except Exception as e:
                # If something went wrong
                # Create error response
                response = {
                    'error' : True,
                    'message' : 'Error : ' + str(e)
                }
        else :
            # If something went wrong
            # Create error response
            response = {
                'error' : True,
                'message' : word_distributions['message']
            }

        return response

    # Save model
    # Saving the model into npy file
    # @param string file_name : Define save file name
    # @return array response
    def save_model(self, file_name) :

        # Create response for indicating if the process is
ended successfully
        response = {
            'error' : True,
            'message' : ''
        }

        try:

            saved_attributes = {}

            saved_attributes['alpha'] = self.__alpha
            saved_attributes['eta'] = self.__eta
            saved_attributes['V'] = self.__V
            saved_attributes['topic_V'] = self.__topic_V
            saved_attributes['K'] = self.__K
            saved_attributes['M'] = self.__M
            saved_attributes['T'] = self.__T
            saved_attributes['L'] = self.__L
            saved_attributes['alpha_D'] = self.__alpha_D
            saved_attributes['eta_D'] = self.__eta_D
            saved_attributes['D'] = self.__D
            saved_attributes['document_topic_matrix']  =
self.__document_topic_matrix
            saved_attributes['topic_term_matrix'] =
self.__topic_term_matrix

```

```
618.                np.save('saved_model/' + file_name +
  '_saved_attributes.npy', saved_attributes)
619.
620.                response = {
621.                    'error' : False,
622.                    'message' : 'Model is saved!'
623.                }
624.            except Exception as e:
625.                # If something went wrong
626.                # Create error response
627.                response = {
628.                    'error' : True,
629.                    'message' : 'Error : ' + str(e)
630.                }
631.
632.
633.            return response
634.
635.        # Load model
636.        # Load model from the file
637.        # @param string file_name : Define loaded file name
638.        # @return array response
639.        def load_model(self, file_name) :
640.
641.            # Create response for indicating if the process is
  ended successfully
642.            response = {
643.                'error' : True,
644.                'message' : ''
645.            }
646.
647.            try:
648.
649.                saved_attributes = np.load('saved_model/' +
  file_name + '_saved_attributes.npy',
  allow_pickle='TRUE').item()
650.
651.                self.__alpha = saved_attributes['alpha']
652.                self.__eta = saved_attributes['eta']
653.                self.__V = saved_attributes['V']
654.                self.__topic_V = saved_attributes['topic_V']
655.                self.__K = saved_attributes['K']
656.                self.__M = saved_attributes['M']
657.                self.__T = saved_attributes['T']
658.                self.__L = saved_attributes['L']
659.                self.__alpha_D = saved_attributes['alpha_D']
660.                self.__eta_D = saved_attributes['eta_D']
661.                self.__D = saved_attributes['D']
662.                self.__document_topic_matrix =
  saved_attributes['document_topic_matrix']
663.                self.__topic_term_matrix =
  saved_attributes['topic_term_matrix']
664.
665.                response = {
```

```
                    'error' : False,
                    'message' : 'Model is loaded!'
                }
            except Exception as e:
                # If something went wrong
                # Create error response
                response = {
                    'error' : True,
                    'message' : 'Error : ' + str(e)
                }

            return response

        # Update model
        # Update model with new datas
        # @param array new_documents : Array of documents, in
  this case document is a text
        # @return array response
        def update_model(self, new_documents) :

            # Create response for indicating if the process is
  ended successfully
            response = {
                'error' : True,
                'message' : ''
            }

            # Try to update the corpus
            try :

                # Store old document length or total documents
                old_M = self.__M

                # Update current document length with new
  documents
                self.__M += len(new_documents)

                # Used to store document labels that will be
  used to set Lambda
                d_labels = []

                # Creating dictionary (vocabulary) of word and
  topic
                # and creating corpus
                # For each document (d) in documents
                for d in new_documents :

                    # Preprocess document
                    preprocessed_d =
  self.__preprocesser.preprocess(d['text'])
                    d_label = []
                    d_words = []

                    # For each term or word in the document
```

```
                    for t in preprocessed_d :

                        # Check if the word or term is inside in the vocabulary
                        # If it is not, add them to the dictionary
                        if t not in self.__V :

                            # Giving each term an id
                            self.__V.append(t)
                            self.__T += 1

                        # Replace words or terms inside document by term id
                        d_words.append({
                            'term' : self.__V.index(t),
                            'topic' : 0
                        })

                    # Add to the corpus
                    self.__D.append(d_words)

                    # Adding common topic to every document
                    current_label = ['common']
                    # current_label = d['label']

                    # For each label or topic inside documents
                    for label in current_label + d['label'] :

                        # If label or topic is not inside the topic vocabulary
                        # add the label or topic
                        if label not in self.__topic_V :

                            # Giving each label or topic an id
                            self.__topic_V.append(label)
                            self.__K += 1

                        # Replace label or topic inside document by topic id
                    d_label.append(self.__topic_V.index(label))

                    # Add them to d_labels that will later will be used to getting Lambda value
                    d_labels.append(d_label)

                # Creating Lambda
                old_L = self.__L
                self.__L = np.zeros((self.__M, self.__K), dtype=int)


```

```
759.                # Updating the Lambda matrix values based from
  the loaded documents
760.                row = 0
761.                for d in old_L:
762.                    # print(d)
763.                    col = 0
764.                    for k in d:
765.                        self.__L[row][col] = k
766.                        col += 1
767.
768.                    row += 1
769.
770.                row = old_M;
771.                for labels in d_labels :
772.
773.                    for label in labels :
774.
775.                        self.__L[row][label] = 1
776.
777.                    row += 1
778.
779.
780.                # Lambda x alpha
781.                self.__alpha_D = self.__L * self.__alpha
782.
783.                # Creating eta_D
784.                self.__eta_D = np.ones((self.__K, self.__T),
  dtype=float)
785.                self.__eta_D = self.__eta_D * self.__eta
786.
787.                # Creating document to topic matrix, size M x K
788.                self.__document_topic_matrix =
  np.zeros((self.__M, self.__K), dtype=int)
789.
790.                # Creating topic to term matrix, size K x T
791.                self.__topic_term_matrix = np.zeros((self.__K,
  self.__T), dtype=int)
792.
793.                # Add random topic to each term based on Lambda
794.                for d in range(old_M, self.__M) :
795.
796.                    p = self.__L[d] / sum(self.__L[d])
797.                    i = 0
798.
799.                    for w in self.__D[d]:
800.
801.                        z = np.random.multinomial(1, p).argmax()
802.                        self.__D[d][i]['topic'] = z
803.                        i += 1
804.
805.                print(' D = ')
806.                print(' {:<8} {:<10} {:<10}'.format('doc_id',
  'term_id', 'topic_id'))
807.                i = 0
```

```
808.                for d in self.__D :
809.                    for w in d:
810.                        print(' {:<8} {:<10} {:<10}'.format((i +
  1), w['term'], w['topic']))
811.                    i += 1
812.                    print()
813.
814.                # Create the document topic matrix and topic
  term matrix
815.                i = 0
816.                for d in range(self.__M) :
817.
818.                    for j in range(len(self.__D[d])) :
819.
820.                        t = self.__D[d][j]['term']
821.                        k = self.__D[d][j]['topic']
822.
823.                        self.__document_topic_matrix[d, k] += 1
824.                        self.__topic_term_matrix[k, t] += 1
825.
826.                    i += 1
827.
828.                print(self.__document_topic_matrix)
829.                print(self.__topic_term_matrix)
830.
831.
832.                # Create success message
833.                response = {
834.                    'error' : False,
835.                    'message' : 'Model created successfully.'
836.                }
837.            except Exception as e :
838.                # If something went wrong
839.                # Create error response
840.                response = {
841.                    'error' : True,
842.                    'message' : 'Error : ' + str(e)
843.                }
844.
845.            return response
```

TRAIN MODEL

```
1. from LLDA import LLDA
2. import csv
3.
4. # Documents
5. documents = []
6.
7. # Sample size
8. sample_size = 250
9.
10.    # Aspect name
```

```
11.   # Change if to other aspect if we want to change aspect
12.   # 3 Aspect (motivasi, semangat_kerja, kesadaran_diri)
13.   name = 'motivasi'
14.
15.   # Alpha & eta
16.   alpha = 10
17.   eta = 0.1
18.
19.   # Create LLDA object
20.   LLDA = LLDA(alpha, eta)
21.
22.   # Processing
23.   print('Process ' + name + '...')
24.   with open('./data/'+ name + '_documents.csv') as file:
25.       reader = csv.reader(file, delimiter=',')
26.       total_row = 0;
27.       for row in reader:
28.           total_row += 1
29.
30.           documents.append(
31.               {
32.                   'text' : row[0],
33.                   'label' : [row[1]]
34.               }
35.           )
36.
37.           if total_row == sample_size:
38.               break
39.
40.   response = LLDA.create_corpus(documents)
41.
42.   if response['error'] == False :
43.       print(response['message'])
44.       LLDA.display_attributes()
45.       LLDA.begin_training(100000)
46.
47.       save = LLDA.save_model(name)
48.       if save['error'] == False:
49.           print(' ' ,save['message'])
50.           print(' Training finished...');
51.   else :
52.       print(response['message'])
```

PREDICT OR CLASSIFY TEST DATA

```
1. from LLDA import LLDA
2. import csv
3.
4. # Create LLDA object
5. LLDA = LLDA(10, 0.1)
6.
7. # Aspect name
8. # Change if to other aspect if we want to change aspect
```

```
# 3 Aspect (motivasi, semangat_kerja, kesadaran_diri)
   name = 'motivasi'
   sample_size = 250

   load = LLDA.load_model(name)

   if load['error'] == False:
       LLDA.display_attributes()

       # Test prediction
       documents = []

       with open('./data/'+ name + '_documents.csv') as file:
           reader = csv.reader(file, delimiter=',')
           total_row = 0;
           for row in reader:
               total_row += 1

               if total_row > sample_size :

                   documents.append(
                       {
                           'text' : row[0],
                           'label' : row[1]
                       }
                   )


       # Check
       # print(len(documents))

       correct = 0
       incorrect = 0

       with open('./prediction_result/' + name + '_result.csv',
  mode='w') as file :

           writer = csv.writer(file, delimiter=',',
  quotechar='"', quoting=csv.QUOTE_MINIMAL)
           writer.writerow(['Document', 'Label', 'Prediction',
  'Hasil'])

           for i in range(len(documents)) :

               status = ''

               predict = LLDA.predict(documents[i]['text'])

               if predict['error'] == False:

                   if predict['data']['topic'] == documents[i]
  ['label'] :
                       correct += 1
```

```
                        status = 'Benar'
                    else :
                        incorrect += 1
                        status = 'Salah'

                    writer.writerow([documents[i]['text'],
  documents[i]['label'], predict['data']['topic'], status])

            # Count accuracy
            accuracy = (correct/(correct + incorrect))
            print(' Accuracy ', accuracy)

            writer.writerow(['total benar','', correct])
            writer.writerow(['accuracy','', accuracy])
```

PREPROCESS CLASS

```
'''
 preprocess.py
 Class used for preprocessing data
'''

# Import libraries
import re
from   Sastrawi.StopWordRemover.StopWordRemoverFactory   import
  StopWordRemoverFactory, StopWordRemover, ArrayDictionary
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

   class Preprocess :

       # Define constructor
       def __init__(self) :

           # Create stop word list
            self.__stopword_factory = ['yang', 'untuk', 'pada',
  'ke', 'para', 'namun', 'menurut', 'antara', 'dia', 'dua', 'ia',
  'seperti', 'jika', 'jika', 'sehingga', 'kembali', 'dan', 'ini',
  'karena',  'kepada',  'oleh',  'saat',  'harus',  'sementara',
  'setelah', 'belum', 'kami', 'sekitar', 'bagi', 'serta', 'di',
  'dari', 'telah', 'sebagai', 'masih', 'hal', 'ketika', 'adalah',
  'itu',  'dalam',  'bisa',  'bahwa',  'atau',  'hanya',  'kita',
  'dengan', 'akan', 'juga', 'ada', 'mereka', 'sudah', 'saya',
  'terhadap',  'secara',  'agar',  'lain',  'anda',  'begitu',
  'mengapa', 'kenapa', 'yaitu', 'yakni', 'daripada', 'itulah',
  'lagi', 'maka', 'tentang', 'demi', 'dimana', 'kemana', 'pula',
  'sambil', 'sebelum', 'sesudah', 'supaya', 'guna', 'kah', 'pun',
  'sampai',   'sedangkan',   'selagi',   'sementara',   'tetapi',
  'apakah', 'kecuali', 'sebab', 'selain', 'seolah', 'seraya',
  'seterusnya', 'tanpa', 'agak', 'boleh', 'dapat', 'dsb', 'dst',
  'dll', 'dahulu', 'dulunya', 'anu', 'demikian', 'tapi', 'ingin',
  'juga',   'nggak',   'mari',   'nanti',   'melainkan',   'oh',
  'seharusnya', 'sebetulnya', 'setiap', 'setidaknya', 'sesuatu',
  'pasti',  'saja',  'toh',  'ya',  'walau',  'tolong',  'tentu',
```

```
  'amat',   'apalagi',   'bagaimanapun',   'si',   'sih',   'aja',
  'pak','jadi','kalau']
18.
19.           stopwords = self.__stopword_factory
20.
21.           dictionary = ArrayDictionary(stopwords)
22.
23.           # Create stop word remover
24.                                     self.__stopword_remover    =
  StopWordRemover(dictionary)
25.
26.           # Create stemmer
27.           self.__stemmer_factory = StemmerFactory()
28.                                              self.__stemmer     =
  self.__stemmer_factory.create_stemmer()
29.
30.       # Preprocess
31.        # This method is used for preprocessing the document
  text
32.       def preprocess(self, text) :
33.
34.                                           preprocessed_text     =
  self.__remove_symbols(text.lower())
35.           # print(preprocessed_text)
36.                                           preprocessed_text     =
  self.__stopword_remover.remove(preprocessed_text)
37.           # print(preprocessed_text)
38.                                           preprocessed_text     =
  self.__stemmer.stem(preprocessed_text)
39.           # print(preprocessed_text)
40.           preprocessed_text = preprocessed_text.split()
41.           print(preprocessed_text)
42.
43.           return preprocessed_text
44.
45.       # Remove symbols
46.       # Used for removing symbols
47.       def __remove_symbols(self, text) :
48.
49.           symbols_re = '[!@$%^&*(){}~`\'"<>,./?:;\\-+_=]'
50.
51.           return re.sub(symbols_re, ' ', text)
```

CLASSIFICATION RESULT

Approach 1 :

Table 6.1: Result Table with Common Topic or Document Contains Neutral Words

Doc Test No	Motivation		Work Enthusiasm		Self-awareness	
	Manual Classification	Program Classification	Manual Classification	Program Classification	Manual Classification	Program Classification
1	1	4	5	5	3	3
2	4	4	5	5	4	4
3	4	2	5	5	4	4
4	1	4	3	3	4	4
5	5	1	3	5	3	3
6	1	2	5	5	1	4
7	4	4	5	5	4	4
8	4	3	3	3	2	2
9	4	4	5	5	3	3
10	1	1	2	2	2	4
11	4	3	2	3	4	4
12	1	1	5	5	3	3
13	1	1	5	5	4	4
14	5	4	3	5	3	3
15	4	3	4	5	4	4
16	4	3	5	2	2	2
17	4	4	1	1	1	1
18	3	4	5	5	2	2
19	2	4	4	5	3	3
20	4	2	5	5	4	4
21	5	3	1	2	2	2
22	4	2	5	5	5	5
23	4	4	2	2	4	5
24	3	3	3	3	4	4
25	2	2	2	2	4	4

26	1	2	2	2	3	4
27	4	4	5	5	3	3
28	2	3	3	5	4	4
29	4	4	4	5	3	2
30	4	4	4	5	1	1
31	4	4	5	5	2	2
32	4	4	3	1	4	4
33	4	4	1	2	3	3
34	4	4	4	4	4	4
35	1	1	5	5	3	3
36	4	4	5	5	5	5
37	4	3	1	1	3	2
38	4	4	2	5	2	4
39	4	4	5	5	5	5
40	4	4	3	1	3	1
41	1	4	3	3	3	2
42	4	4	5	5	1	1
43	4	4	5	5	2	2
44	2	3	1	5	3	3
45	4	4	5	3	2	3
46	4	3	1	5	3	4
47	3	3	2	5	3	1
48	3	3	1	1	2	4
49	4	4	3	3	5	5
50	1	1	4	5	1	1
51	2	2	5	5	4	4
52	2	2	2	5	4	4
53	4	4	5	5	2	4
54	2	4	3	5	4	5
55	3	4	3	3	2	1
56	4	4	5	3	2	4

57	4	4	2	2	2	2
58	4	1	2	2	1	1
59	1	1	2	2	3	3
60	2	4	3	3	3	3
61	4	4	3	3	4	4
62	2	3	2	2	3	1
63	4	4	3	3	3	3
64	4	4	2	2	3	4
65	4	4	4	5	3	2
66	4	4	5	5	4	4
67	1	1	4	5	3	3
68	4	2	3	3	2	2
69	2	1	4	5	3	4
70	4	4	2	5	2	2
71	4	4	5	5	2	4
72	1	1	5	5	3	3
73	4	4	5	5	5	4
74	2	3	3	3	2	4
75	4	4	5	5	4	4
76	1	1	5	3	3	3
77	3	3	2	1	2	2
78	1	1	1	2	3	3
79	1	3	5	5	4	4
80	2	4	5	5	2	2
81	4	4	5	5	1	3
82	2	2	5	5	4	4
83	2	4	1	5	4	5
84	2	2	5	5	2	4
85	4	5	3	3	3	3
86	5	5	1	1	4	4
87	4	3	5	5	2	2

Y

88	4	4	3	5	4	4
89	1	1	2	2	3	1
90	1	2	5	5	4	4
91	2	4	5	5	3	1
92	2	4	2	2	3	3
93	1	1	1	5	4	4
94	4	4	1	1	1	4
95	4	4	4	4	3	3
96	4	4	2	5	2	2
97	4	4	2	5	4	4
98	4	4	5	5	3	4
99	2	2	4	4	3	1
100	4	2	3	2	2	2
Correct Total	62		65		68	
Accuracy	62.00%		65.00%		68.00%	

Approach 2 :

Table 6.2: Result Table without Common Topic or Document Does not Contain Neutral Words

Doc Test No	Motivation		Work Enthusiasm		Self-awareness	
	Manual Classification	Program Classification	Manual Classification	Program Classification	Manual Classification	Program Classification
1	1	4	5	5	3	2
2	4	4	5	5	4	4
3	4	1	5	5	4	4
4	1	4	3	3	4	4
5	5	5	3	5	3	3
6	1	2	5	5	1	4
7	4	4	5	5	4	4
8	4	4	3	3	2	2
9	4	4	5	5	3	3
10	1	1	2	2	2	4

11	4	3	2	5	4	4
12	1	1	5	5	3	3
13	1	1	5	5	4	4
14	5	4	3	5	3	3
15	4	1	4	4	4	4
16	4	4	5	2	2	2
17	4	4	1	1	1	1
18	3	4	5	5	2	2
19	2	4	4	4	3	3
20	4	2	5	5	4	4
21	5	3	1	1	2	2
22	4	2	5	5	5	5
23	4	4	2	2	4	5
24	3	4	3	1	4	4
25	2	2	2	2	4	4
26	1	3	2	2	3	4
27	4	4	5	5	3	3
28	2	2	3	5	4	4
29	4	4	4	5	3	2
30	4	4	4	5	1	2
31	4	4	5	5	2	2
32	4	4	3	1	4	4
33	4	4	1	1	3	3
34	4	4	4	4	4	4
35	1	1	5	5	3	3
36	4	4	5	5	5	5
37	4	3	1	1	3	2
38	4	4	2	5	2	4
39	4	4	5	5	5	5
40	4	4	3	1	3	1
41	1	4	3	3	3	3

42	4	4	5	5	1	1
43	4	4	5	5	2	2
44	2	2	1	5	3	3
45	4	4	5	5	2	2
46	4	3	1	2	3	2
47	3	2	2	5	3	1
48	3	4	1	1	2	4
49	4	4	3	3	5	5
50	1	1	4	4	1	1
51	2	2	5	5	4	4
52	2	2	2	5	4	4
53	4	4	5	5	2	4
54	2	2	3	5	4	5
55	3	4	3	3	2	1
56	4	4	5	3	2	4
57	4	4	2	2	2	2
58	4	2	2	2	1	1
59	1	1	2	2	3	3
60	2	2	3	3	3	3
61	4	1	3	3	4	4
62	2	1	2	2	3	1
63	4	4	3	3	3	3
64	4	4	2	2	3	4
65	4	4	4	5	3	2
66	4	4	5	5	4	4
67	1	1	4	4	3	3
68	4	5	3	3	2	2
69	2	2	4	5	3	3
70	4	4	2	5	2	2
71	4	4	5	5	2	4
72	1	1	5	5	3	3

73	4	4	5	5	5	4
74	2	2	3	3	2	4
75	4	4	5	5	4	4
76	1	1	5	3	3	3
77	3	3	2	1	2	2
78	1	1	1	2	3	3
79	1	1	5	5	4	4
80	2	4	5	5	2	2
81	4	4	5	5	1	3
82	2	2	5	5	4	4
83	2	2	1	5	4	5
84	2	2	5	5	2	4
85	4	1	3	3	3	3
86	5	5	1	1	4	4
87	4	4	5	5	2	2
88	4	4	3	5	4	4
89	1	1	2	2	3	3
90	1	1	5	5	4	4
91	2	4	5	5	3	3
92	2	4	2	2	3	3
93	1	1	1	5	4	4
94	4	4	1	1	1	4
95	4	4	4	4	3	3
96	4	4	2	4	2	2
97	4	4	2	5	4	4
98	4	4	5	5	3	4
99	2	2	4	4	3	3
100	4	1	3	5	2	2
Correct Total	71		71		72	
Accuracy	71.00%		71.00%		72.00%	

AC

UNICHECK

Submission author:
16k10007 EVANDER REINHART M

Check ID:
15950399

Check date:
16.01.2020 09:31:42 GMT+0

Check type:
Doc vs Internet + Library

Report date:
17.01.2020 03:14:42 GMT+0

User ID:
29135

File name: **16.K1.0007_Evander Reinhart Mulyono.docx**

File ID: **20249317** Page count: **29** Word count: **10456** Character count: **60617** File size: **86.04 KB**

0.77% Matches

Highest match: **0.25%** with source **https://en.wikipedia.org/wiki/Pattern_mining**

0.7% Internet Matches	51	Page 31
0.08% Library matches	24	Page 31

5.63% Quotes

Quotes	8	Page 32

No references found

0% Exclusions

No exclusions found

Replacement

No replaced characters found