CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

In the implementation step, we will create a program using the Python programming language and then we will create a class called L-LDA. The code written is based on Jiahong Zhou implementation of L-LDA algorithm [18]. The code will only focus on classification with L-LDA. Additional libraries used to assist in the process are numpy and Sastrawi. Numpy is used to help the process of calculating arrays because later 2 matrices will be created as explained in chapter 4 earlier (Document Topic Matrix and Topic Term Matrix) also helps in the process of taking a random sampling from a probability and Sastrawi is used to help the preprocessing stage, in the stemming process which means changing words into standard forms.

So, the first step is creating a L-LDA class and then we will create a constructor to define some properties. Those will be used later in creating the model.

```
1. def __init__(self, alpha, eta) :
2.
3.         self.__alpha = alpha
4.         self.__eta = eta
5.         self.__K = 0
6.         self.__M = 0
7.         self.__T = 0
8.         self.__V = []
9.         self.__topic_V = []
10.         self.__D = []
11.         self.__L = []
12.         self.__alpha_D = []
13.         self.__eta_D = []
14.         self.__document_topic_matrix = []
15.         self.__topic_term_matrix = []
16.         self.__preprocesser = Preprocess()
```

After that we need to create the corpus. To create the corpus, first we will load all the documents and then those documents will be preprocessed first by using a preprocesser object. So we need to create the preprocesser object by

creating a Preprocess class. Also we are going to create our own stop words library because in this case, some stop words have a meaning (Example : ok, siap, etc).

```
class Preprocess :
    def __init__(self) :
        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']
        stopwords = self.__stopword_factory
        dictionary = ArrayDictionary(stopwords)
        self.__stopword_remover = StopWordRemover(dictionary)
        self.__stemmer_factory = StemmerFactory()
        self.__stemmer =
  self.__stemmer_factory.create_stemmer()
```

Then to preprocess a document, we will remove symbols from the document and then use our own stop words library and Sastrawi stemmer.

```
    def preprocess(self, text) :

        preprocessed_text = self.__remove_symbols(text.lower())
        preprocessed_text =
  self.__stopword_remover.remove(preprocessed_text)
        preprocessed_text =
  self.__stemmer.stem(preprocessed_text)
        preprocessed_text = preprocessed_text.split()
        print(preprocessed_text)

        return preprocessed_text

   def __remove_symbols(self, text) :
        symbols_re = '[!@$%^&*(){}~`\'"<>,./?:;\\-+_=]'
        return re.sub(symbols_re, ' ', text)
```

Then we will insert some values into array D. the row index represents document id or number and then the contents are the words in each document according to the document number (document id). But the words will be replaced with each id of the word that belongs to that word. So like in chapter 4, we will create a word vocabulary dictionary and then also create a topic vocabulary dictionary.

Creating a word vocabulary dictionary :

```
1.  for d in documents :
2.          preprocessed_d =
  self.__preprocesser.preprocess(d['text'])
3.          d_label = []
4.          d_words = []
5.
6.          for t in preprocessed_d :
7.
8.            if t not in self.__V :
9.
10.                 self.__V.append(t)
11.                 self.__T += 1
12.
13.             d_words.append({
14.                 'term' : self.__V.index(t),
15.                 'topic' : 0
16.             })
17.
18.         self.__D.append(d_words)
19.
20.         current_label = ['common']
21.
22.         for label in current_label + d['label'] :
23.
24.             if label not in self.__topic_V :
25.
26.                 self.__topic_V.append(label)
27.                 self.__K += 1
28.
29.             d_label.append(self.__topic_V.index(label))
30.
31.         d_labels.append(d_label)
```

After that we will create a Lambda value like in chapter 4. To create it, we will check on d_labels array because that array is used to describe the label or topic in each document.

```
1. self.__L = np.zeros((self.__M, self.__K), dtype=int)
2.
```

```
3. row = 0;
4. for labels in d_labels :
5.
6.     for label in labels :
7.
8.         self.__L[row][label] = 1
9.
10.    row += 1
```

Then we will create α matrix and η matrix and also create a $\alpha^{(d)}$

```
1. self.__alpha_D = self.__L * self.__alpha
2.
3. self.__eta_D = np.ones((self.__K, self.__T), dtype=float)
4. self.__eta_D = self.__eta_D * self.__eta
```

After the corpus is created, we will give a topic to each of the words in each document in the corpus which later indicates that the word shows or illustrates a particular topic based on the Lambda that has been set. It is the same steps that has already been explained in chapter 4.

```
1. for d in range(self.__M) :
2.     p = self.__L[d] / sum(self.__L[d])
3.     i = 0
4.
5.     for w in self.__D[d]:
6.
7.         z = np.random.multinomial(1, p).argmax()
8.         self.__D[d][i]['topic'] = z
9.         i += 1
```

Then we can create the Document Topic Matrix and Topic Term Matrix like explained in chapter 4.

```
1. self.__document_topic_matrix = np.zeros((self.__M, self.__K),
   dtype=int)
2. self.__topic_term_matrix = np.zeros((self.__K, self.__T),
   dtype=int)
3. i = 0
4. for d in range(self.__M) :
5.
6.     for j in range(len(self.__D[d])) :
7.
8.         t = self.__D[d][j]['term']
9.         k = self.__D[d][j]['topic']
10.
11.        self.__document_topic_matrix[d, k] += 1
12.        self.__topic_term_matrix[k, t] += 1
13.
14.     i += 1
```

After that, the next step is creating the model or training the model. We will use collapsed Gibbs Sampling that is already been explained before in chapter 4. We will slowly the topic of every word in every document in the corpus D.

```
for d in range(self.__M) :

    for j in range(len(self.__D[d])) :

        t = self.__D[d][j]['term']
        k = self.__D[d][j]['topic']

        self.__document_topic_matrix[d, k] -= 1
        self.__topic_term_matrix[k, t] -= 1

       left = (self.__topic_term_matrix[:, t] + self.__eta_D[k,
  t]) / (self.__topic_term_matrix + self.__eta_D[k]).sum(axis=1)
        right = (self.__document_topic_matrix[d] +
  self.__alpha_D[d]) / (self.__document_topic_matrix +
  self.__alpha_D).sum(axis=1)[d]
        p = (left * right/ sum(left * right))
        z = np.random.multinomial(1, p).argmax()

        self.__D[d][j]['topic'] = z

        self.__document_topic_matrix[d, z] += 1
        self.__topic_term_matrix[z, t] += 1
```

After training process, we can get word distributions in all topics by using :

```
beta = (self.__topic_term_matrix + self.__eta_D) /
  (self.__topic_term_matrix +
  self.__eta_D).sum(axis=1).reshape(self.__K, 1)
```

And topic distributions in all documents by using :

```
theta = (self.__document_topic_matrix + (self.__alpha_D)) /
  (self.__document_topic_matrix +
  (self.__alpha_D)).sum(axis=1).reshape(self.__M, 1)
```

Then to predict or classify a document, we will use the same method like explained in chapter 4.

```
    word_distributions = self.get_word_distributions(False)

    preprocessed_new_document =
  self.__preprocesser.preprocess(new_document)

    result = []

    for k in word_distributions['data'] :

       if k['topic'] != 'common':
```

```
10.
11.             print(k['topic'])
12.
13.             total = 0
14.
15.             for t in k['terms'] :
16.
17.                 if t['term'] in preprocessed_new_document :
18.                     print(t)
19.
20.                     total += (1 * t['weight'])
21.
22.             result.append({
23.             'topic' : k['topic'],
24.             'total' : total
25.             })
26.
27.     max = {
28.         'topic' : '',
29.         'total' : 0
30.     }
31.
32.     for re in result :
33.
34.         if max['total'] < re['total']:
35.             max = {
36.                 'topic' : re['topic'],
37.                 'total' : re['total']
38.             }
```

When there are new data that needs to be inserted, we need to update our model. Check if there is a new topic or label in the new data and check if there is a new word in the new data. Then we can update the model current word vocabulary dictionary and topic vocabulary dictionary

```
1. d_labels = []
2. for d in new_documents :
3.
4.         preprocessed_d =
   self.__preprocesser.preprocess(d['text'])
5.         d_label = []
6.         d_words = []
7.
8.         for t in preprocessed_d :
9.
10.            if t not in self.__V :
11.
12.                self.__V.append(t)
13.                self.__T += 1
14.
15.            d_words.append({
16.                'term' : self.__V.index(t),
```

```
                'topic' : 0
            })

        self.__D.append(d_words)

        current_label = ['common']

        for label in current_label + d['label'] :

            if label not in self.__topic_V :

                self.__topic_V.append(label)
                self.__K += 1

            d_label.append(self.__topic_V.index(label))

        d_labels.append(d_label)
```

Then we need to update the Lambda value.

```
    old_L = self.__L
    self.__L = np.zeros((self.__M, self.__K), dtype=int)

    row = 0
    for d in old_L:
        # print(d)
        col = 0
        for k in d:
            self.__L[row][col] = k
            col += 1

        row += 1

    row = old_M;
    for labels in d_labels :

        for label in labels :

            self.__L[row][label] = 1

        row += 1
```

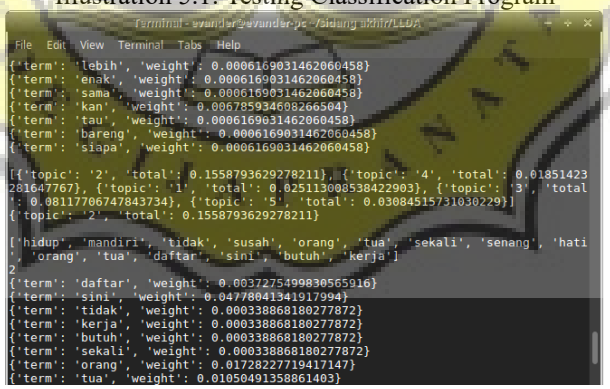
Do not forget to update the Document Topic Matrix and Topic Term Matrix and update other properties inside the model.

```
    self.__alpha_D = self.__L * self.__alpha

    self.__eta_D = np.ones((self.__K, self.__T), dtype=float)
    self.__eta_D = self.__eta_D * self.__eta

    self.__document_topic_matrix = np.zeros((self.__M,
  self.__K), dtype=int)
```

```
8.     self.__topic_term_matrix = np.zeros((self.__K, self.__T),
   dtype=int)
9.
10.    for d in range(old_M, self.__M) :
11.
12.        p = self.__L[d] / sum(self.__L[d])
13.        i = 0
14.
15.        for w in self.__D[d]:
16.
17.            z = np.random.multinomial(1, p).argmax()
18.            self.__D[d][i]['topic'] = z
19.            i += 1
20.
21.    i = 0
22.    for d in range(self.__M) :
23.
24.        for j in range(len(self.__D[d])) :
25.
26.            t = self.__D[d][j]['term']
27.            k = self.__D[d][j]['topic']
28.
29.            self.__document_topic_matrix[d, k] += 1
30.            self.__topic_term_matrix[k, t] += 1
31.
32.        i += 1
```

5.2 Testing

Illustration 5.1: Testing Classification Program



For testing, we will compare the results of classification tests from manual classification and classification with the program. From the results of the classification of the program itself will be obtained from the two approaches that were explained earlier. Later the results will be compared which results are much better.

Approach 1 :

Table 5.1: Result Table with Common Topic or Document Contains Neutral Words

Aspect	Motivation	Work Enthusiasm	Self-awareness
Correct Total	62	65	68
Accuracy	62.00%	65.00%	68.00%

Approach 2 :

Table 5.2: Result Table without Common Topic or Document Does not Contain Neutral Words

Aspect	Motivation	Work Enthusiasm	Self-awareness
Correct Total	71	71	72
Accuracy	71.00%	71.00%	72.00%

The results show that the second approach is far better than the first approach because the accuracy obtained from the classification of each aspect is much better than the first approach. Why is that?

This is caused by the laying of words that do not describe any topic or are common to the wrong words. In grouping data, if in a text document consists of so many variations of words, it will be very difficult to find words that are included in the common topic or which means the word does not refer anywhere or the word that is general in nature which means the word also can sometimes appear frequently in each document. If inside the corpus or text data used has several words that are the same then the results will be definitely much better and words that are common will certainly be detected as common words or words that do not refer to any topic KS, K, C, B, and BS. Because in every document that word is almost appearing all the time. It would be much different if in the data set the words used are vary, this can lead to inaccurate results if you want to also search for words that are common or neutral. Therefore the second approach is much better because all data is directly grouped into that category / addressed to the topic.