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
        self.__topic_term_matrix = []      # Topic to term matrix

        self.__preprocesser = Preprocess() # Define preprocessor object

    # Create corpus
    # Used for creating corpus (D), collection of documents
    # @param array documents : Array of documents, in this case document is a text
    # @return array response
    def create_corpus(self, documents) :

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

        # Try to create the corpus
        try :

            # Get the total of documents loaded
            self.__M = len(documents)

            # Reset the T and K values to zero
            self.__T = 0
            self.__K = 0

            # 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 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
            # Remove common if you want to do the other variant or approach (no common topic)
            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
        self.__L = np.zeros((self.__M, self.__K), dtype=int)


        # Updating the Lambda matrix values based from the loaded documents
        row = 0;
        for labels in d_labels :

            for label in labels :

                self.__L[row][label] = 1

            row += 1


        # Lambda x alpha
        self.__alpha_D = self.__L * self.__alpha

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

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

    return response


# Display attributes
# Used to display the attributes of the model
# @return void
def display_attributes(self) :

    print('/**')
    print(' Display Attributes')
    print(' Inside your current LLDA model : ')
    print()
    print(' alpha =', self.__alpha)
    print(' eta =', self.__eta)
    print()
    print(' K =', self.__K)
    print(' M =', self.__M)
    print(' T =', self.__T)
    print()

    print(' D = ')
    print(' {:<8} {:<10} {:<10}'.format('doc_id', 'term_id', 'topic_id'))
    i = 0
    for d in self.__D :
        for w in d:
            print(' {:<8} {:<10} {:<10}'.format((i + 1), w['term'], w['topic']))
        i += 1
        print()

    print()
    print(' V =')
    print(' {:<8} {:<20}'.format('id','term'))
    for i in range(len(self.__V)) :
        print(' {:<8} {:<20}'.format(i, self.__V[i]))
    print()

    print(' topic_V = ')
    print(' {:<8} {:<20}'.format('id','topic'))
    for i in range(len(self.__topic_V)) :
        print(' {:<8} {:<20}'.format(i, self.__topic_V[i]))
    print()
    print(' Lambda = ')
    print(self.__L)
    print()
    print(' alpha_D = ')
    print(self.__alpha_D)
    print(' eta_D = ')
    print(self.__eta_D)
    print('**/')

# Prepare training
# 2 big idea of LDA (in this case LLDA)
# 'Document consists of the distribution of topics'
# 'Topic consists of the distribution of terms'
# Preparing training the model by creating document topic matrix and topic term matrix
# @return array response
def __prepare_training(self) :

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

    # Try to prepare the model for training
    try :
        # Creating document to topic matrix, size M x K
        self.__document_topic_matrix = np.zeros((self.__M, self.__K), dtype=int)

        # Creating topic to term matrix, size K x T
        self.__topic_term_matrix = np.zeros((self.__K, self.__T), dtype=int)


        # Add random topic to each term based on Lambda
        for d in range(self.__M) :

            p = self.__L[d] / sum(self.__L[d])
            i = 0

            for w in self.__D[d]:

                z = np.random.multinomial(1, p).argmax()
                self.__D[d][i]['topic'] = z
                i += 1



        print(' D = ')
        print(' {:<8} {:<10} {:<10}'.format('doc_id', 'term_id', 'topic_id'))
        i = 0
        for d in self.__D :
            for w in d:
                print(' {:<8} {:<10} {:<10}'.format((i + 1), w['term'], w['topic']))
            i += 1
            print()

        # Create the document topic matrix and topic term matrix
        i = 0
        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

            i += 1

        print(self.__document_topic_matrix)
        print(self.__topic_term_matrix)

        # Create success message
        response = {
            'error' : False,
            'message' : 'Model is prepared for training!'
        }

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

    return response


# Begin training
# Begin training the model
# @param int times : How many times do you want to train the model
# @return array response
def begin_training(self, times) :

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

    # Try to begin training the model
    try :
        # Prepare the training process
        preparation = self.__prepare_training()

        if not preparation['error'] :
            # If the preparation is successful
            for i in range(times) :

                print('/**')
                print(' * Iteration -', (i + 1))
                print('**/')
                print()

                # Train the model
                self.__train_model()

                print(' D = ')
                print(' {:<8} {:<10} {:<10}'.format('doc_id', 'term_id', 'topic_id'))
                j = 0
                for d in self.__D :
                    for w in d:
                        print(' {:<8} {:<10} {:<10}'.format((j + 1), w['term'], w['topic']))
                    j += 1
                    print()

            response = {
                'error' : False,
                'message' : 'Model is successfully trained!'
            }

        else :
            # If something went wrong
            # Create error response
            response = {
                'error' : True,
                'message' : 'Error when try to train the model : ' + preparation['message']
            }


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

    return response

# Train model
# Train the model using gibbs sampling
# @return void
def __train_model(self) :

    # For each document
    for d in range(self.__M) :


        # We will change each z value on each word inside document d by using gibbs sampling
        for j in range(len(self.__D[d])) :

            # Select the current term and topic assigned to the term or word
            t = self.__D[d][j]['term']
            k = self.__D[d][j]['topic']

            # decrement the count of topic k inside topic d and the count of term t in topic k
            self.__document_topic_matrix[d, k] -= 1
            self.__topic_term_matrix[k, t] -= 1

            # Using gibbs sampling to get a new topic value and then will be assigned to the term
            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()

            # Assign the new z (new topic) to the word or term
            self.__D[d][j]['topic'] = z

            # Update the document topic matrix and topic term matrix
            self.__document_topic_matrix[d, z] += 1
            self.__topic_term_matrix[z, t] += 1

# Get word distributions (beta)
# Get total word distributions inside all topics in corpus
# @param boolean sort_result : True if you want to sort the result based from the weight
# @return array response
def get_word_distributions(self, sort_result) :

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

    try:

        # Get beta (word or term distributions inside topics)
        beta = (self.__topic_term_matrix + self.__eta_D) / (self.__topic_term_matrix + self.__eta_D).sum(axis=1).reshape(self.__K, 1)

        # Create result array
        result = []

        # Process the result after getting beta
        for k in range(len(beta)) :

            terms = []

            for t in range(len(beta[k])) :

                terms.append({
                    'term' : self.__V[t],
                    'weight' : beta[k][t]
                })

            if sort_result:
                terms = sorted(terms, key = lambda i: i['weight'], reverse = True)

            result.append({
                'topic' : self.__topic_V[k],
                '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:
                topics = sorted(topics, key = lambda i: i['weight'], reverse = True)

            result.append({
                'doc' : doc_no,
                'topics' : topics
            })

        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

# Predict
# Predict new document
# @param string new_document
# @return array response
def predict(self, new_document) :

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

    # Get word or term distributions
    word_distributions = self.get_word_distributions(False)

    if not word_distributions['error'] :
        # If successful getting the word distribution

        try:

            # Preprocess the new document
            preprocessed_new_document = self.__preprocesser.preprocess(new_document)

            # Create result array
            result = []

            # Predict new document based from the new document terms and word or term distribution inside model
            for k in word_distributions['data'] :

                # If the topic is common, ignore it
                if k['topic'] != 'common':

                    print(k['topic'])

                    total = 0

                    # Check if term is appearing in the new document, multiply it with its weight
                    for t in k['terms'] :


                        if t['term'] in preprocessed_new_document :
                            print(t)

                            total += (1 * t['weight'])


                    print()

                    result.append({
                    'topic' : k['topic'],
                    'total' : total
                    })


            # Check the result
            max = {
                'topic' : '',
                'total' : 0
            }

            print(result)

            # Getting the max total so we can conclude the prediction result
            for re in result :

                if max['total'] < re['total']:
                    max = {
                        'topic' : re['topic'],
                        'total' : re['total']
                    }

            print(max)
            print()

            response = {
                '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

        np.save('saved_model/' + file_name + '_saved_attributes.npy', saved_attributes)

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


      return response

    # Load model
    # Load model from the file
    # @param string file_name : Define loaded file name
    # @return array response
    def load_model(self, file_name) :

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

      try:

        saved_attributes = np.load('saved_model/' + file_name + '_saved_attributes.npy', allow_pickle='TRUE').item()

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

        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)


        # Updating the Lambda matrix values based from the loaded documents
        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


        # Lambda x alpha
        self.__alpha_D = self.__L * self.__alpha

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

        # Creating document to topic matrix, size M x K
        self.__document_topic_matrix = np.zeros((self.__M, self.__K), dtype=int)

        # Creating topic to term matrix, size K x T
        self.__topic_term_matrix = np.zeros((self.__K, self.__T), dtype=int)

        # Add random topic to each term based on Lambda
        for d in range(old_M, self.__M) :

            p = self.__L[d] / sum(self.__L[d])
            i = 0

            for w in self.__D[d]:

                z = np.random.multinomial(1, p).argmax()
                self.__D[d][i]['topic'] = z
                i += 1

        print(' D = ')
        print(' {:<8} {:<10} {:<10}'.format('doc_id', 'term_id', 'topic_id'))
        i = 0
```

```
            for d in self.__D :
                for w in d:
                    print(' {:<8} {:<10} {:<10}'.format((i + 1), w['term'], w['topic']))
                i += 1
                print()

            # Create the document topic matrix and topic term matrix
            i = 0
            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

                i += 1

            print(self.__document_topic_matrix)
            print(self.__topic_term_matrix)


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

        return response
```

TRAIN MODEL

```
from LLDA import LLDA
import csv

# Documents
documents = []

# Sample size
sample_size = 250

# Aspect name
```

```
    # Change if to other aspect if we want to change aspect
    # 3 Aspect (motivasi, semangat_kerja, kesadaran_diri)
    name = 'motivasi'

    # Alpha & eta
    alpha = 10
    eta = 0.1

    # Create LLDA object
    LLDA = LLDA(alpha, eta)

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

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

            if total_row == sample_size:
                break

    response = LLDA.create_corpus(documents)

    if response['error'] == False :
        print(response['message'])
        LLDA.display_attributes()
        LLDA.begin_training(100000)

        save = LLDA.save_model(name)
        if save['error'] == False:
            print(' ' ,save['message'])
            print(' Training finished...');
    else :
        print(response['message'])
```

PREDICT OR CLASSIFY TEST DATA

```
from LLDA import LLDA
import csv

# Create LLDA object
LLDA = LLDA(10, 0.1)

# Aspect name
# 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']

            stopwords = self.__stopword_factory

            dictionary = ArrayDictionary(stopwords)

            # Create stop word remover
            self.__stopword_remover = StopWordRemover(dictionary)

            # Create stemmer
            self.__stemmer_factory = StemmerFactory()
            self.__stemmer = self.__stemmer_factory.create_stemmer()

        # Preprocess
         # This method is used for preprocessing the document text
        def preprocess(self, text) :

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

            return preprocessed_text

        # Remove symbols
        # Used for removing symbols
        def __remove_symbols(self, text) :

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

            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

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%	

UNICHECK

Submission author:
16k10007 EVANDER REINHART M

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

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

Check ID:
15950399

Check type:
Doc vs Internet + Library

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