CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

This project will be coded in python language.

First step in the program is to import the datasets first, and then preprocess the data.

```
1. jumlah_read_data = 330
2. data_read = 'Datasets/data' + str(jumlah_read_data) + '.csv'
3.
4. df = pd.read_csv(data_read, usecols=['Tweet','HS'])
```

At the first line, the writer specifies the amount of data that wanted ot be processed. The second line is to describes the location of the data is stored in the project workspace.

```
5. def preprocess(text):
6.     text = re.sub(r'URL\Z',' ', str(text))
7.     text = text.encode('ascii', 'replace').decode('ascii')
8.     text = re.sub('user', ' ', text)
9.     text = re.sub('rt', ' ', text)
10.      text = re.sub(r'\W', ' ', text)
11.      text = ' '.join(re.sub("([@#][A-Za-z0-9]+)|(\w+:\/\/\S+)"," ",
   text).split())
12.  text = text.replace('\\t'," ").replace('\\n'," ").replace('\\u'," 
   ").replace('\\',"")
13.      text = re.sub('\s+',' ',text)
14.  text = text.strip()
15.      text = stopwords.remove(text)
16.      text = stemmer.stem(text)
17.    text = ' '.join([alay_word_map[word] if word in alay_word_map
   else word for word in text.split(' ')])
18.
19.      return text
20.
21.  def wordtokenize(text):
22.      return word_tokenize(text)
23.
24.  df["Tweet"] = df["Tweet"].apply(preprocess)
25.  df["Tweet_List"] = df["Tweet"].apply(wordtokenize)
```

Because of several indonesian slang used in the data, dictionary of slang words must be included in the preprocessing section. Line 24 and 25 are for the implementation of the functions based on all rows of dataframe.

```
26.  def convert_to_list(text):
27.      new_text = ast.literal_eval(text.strip())
28.      return [text for text in new_text]
29.
30.  def hitung_tf(document):
31.      tf_dictionary = {}
32.
33.      # hitung frekuensi kata 'f' di seluruh dokumen
34.      for f in document:
```

```
35.          if f in tf_dictionary:
36.              tf_dictionary[f] += 1
37.          else:
38.              tf_dictionary[f] = 1
39.
40.      # hitung tf setiap kata
41.      for tf in tf_dictionary:
42.          tf_dictionary[tf] = tf_dictionary[tf] / len(document)
43.
44.      return tf_dictionary
45.
46.  def hitung_df(tf):
47.      jumlah_DF = {}
48.
49.      for document in tf:
50.          for x in document:
51.              if x in jumlah_DF:
52.                  jumlah_DF[x] += 1
53.              else:
54.                  jumlah_DF[x] = 1
55.
56.      return jumlah_DF
57.
58.  def hitung_idf(p_n_document, p_df):
59.      idf = {}
60.
61.      for x in p_df:
62.          idf[x] = np.log( p_n_document /  p_df[x])
63.
64.      return idf
65.
66.  def hitung_tfidf(tf):
67.      tfidf = {}
68.
69.      for x in tf:
70.          tfidf[x] = tf[x] * idf[x]
71.      return tfidf
```

Function line 26 is used to convert tweet in string type into a list of words from string type. Line 30 is to calculate the tf. Line 46 to calculate df values. Line 58 to calculate idf values based on total documents and df values that previously obtained. Line 66 is used to count the final tf-idf value for each documents.

```
72.  df["list"] = df["Tweet_List"].apply(convert_to_list)
73.  df["tf"] = df["list"].apply(hitung_tf)
74.  v_df = hitung_df(df['list'])
75.
76.  n_document = len(df)
77.
78.  idf = hitung_idf(n_document, v_df)
79.  df['tfidf'] = df['tf'].apply(hitung_tfidf)
```

Line 72 used to convert string type of data in the dataframe into list type of data. Line 73 used to calculate tf based on data inside the dataframe. Line 74 is to hold the return value of function *hitung_df* into a variable. Line 78 is to hold the return value of calculated idf in the function *hitung_idf*. Line 79 is to calculate tf-idf value and store it in the dataframe.

```
80.  view_tfidf = dict(df['tfidf'])
81.  weight = {}
82.
83.  for p_id, p_info in view_tfidf.items():
84.      for key in p_info:
85.          weight[p_id] = 0
86.          values = p_info.values()
87.          total = sum(values)
88.          if total != 0:
89.              weight[p_id] = total
```

From Line 83 to below is to calculate the weight of tf-idf value that been obtained from line 79.

```
90.  c1_awal = df.loc[df['Document'] == 0]
91.  c2_awal = df.loc[df['Document'] == 1]
92.  c1_awal = list(c1_awal['Value'])
93.  c2_awal = list(c2_awal['Value'])
```

Line 90 and 91 is to define the first centroid point. Here, the research is using 2 centroids. Therefore the first centroid point of c1 and c2 is defined by first and second data in the dataframe. Line 92 and 93 are to convert the variable that contains the data into a list data type.

```
94.  def hitung_euclidean_c1(parameter, c1):
95.      new_value_data_c1 = {}
96.
97.      new_value_data_c1 = parameter - c1
98.      new_value_data_c1 = np.power(new_value_data_c1,2)
99.      new_value_data_c1 = np.sqrt(new_value_data_c1)
100.
101.     return new_value_data_c1
102.
103. def hitung_euclidean_c2(parameter, c2):
104.     new_value_data_c2 = {}
105.
106.     new_value_data_c2 = parameter - c2
107.     new_value_data_c2 = np.power(new_value_data_c2,2)
108.     new_value_data_c2 = np.sqrt(new_value_data_c2)
109.
110.     return new_value_data_c2
```

Line 94 and 103 are the functions used to calculate the euclidean distance between each data and centroids. The data point that will be used here is the weight of each documents that have been calculated before.

```
111. data_iteration = pd.DataFrame()
112.
113. iteration_count = 1
114. lock_true = 1
115.
116. while (lock_true == 1):
117.
118.     # Iterasi
119.     # ----------------------------------
```

```
120.     data_iteration['C1']  =  df['Value'].apply(hitung_euclidean_c1,
   args=(c1_awal))
121.     data_iteration['C2']  =  df['Value'].apply(hitung_euclidean_c2,
   args=(c2_awal))
122.     data_iteration['Value'] = df['Value']
123.     data_iteration['HS'] = df['HS']
124.     comparison_column     =     np.where(data_iteration['C1']     <
   data_iteration['C2'], 'C1', 'C2')
125.     data_iteration['Cluster'] = comparison_column
126.     # -----------------------------------
127.     # Cari Centroid Baru
128.     # -----------------------------------
129.     # C1
130.
131.     # Mencari cluster c1 dan tampung weight nya
132.     data_cluster1                                                =
   data_iteration['Value'].loc[data_iteration['Cluster'] == 'C1']
133.
134.     # Menghitung banyaknya data cluster C1
135.     data_cluster1_total = len(data_cluster1)
136.
137.     # Menghitung summary dari data cluster C1
138.     data_cluster1                                                =
   data_iteration['Value'].loc[data_iteration['Cluster'] == 'C1'].sum()
139.
140.     c1_baru  =  float(data_cluster1  /  data_cluster1_total)  #  Titik
   Centroid baru
141.
142.     # C2
143.     data_cluster2                                                =
   data_iteration['Value'].loc[data_iteration['Cluster'] == 'C2']
144.     data_cluster2_total = len(data_cluster2)
145.     data_cluster2                                                =
   data_iteration['Value'].loc[data_iteration['Cluster'] == 'C2'].sum()
146.
147.     c2_baru = float(data_cluster2 / data_cluster2_total)
148.     # -----------------------------------
149.     iteration_count += 1
150.
151.     if c1_awal[0] == c1_baru and c2_awal[0] == c2_baru:
152.         print(iteration_count)
153.         break
154.     else:
155.         c1_awal[0] = c1_baru
156.         c2_awal[0] = c2_baru
```

From line 111 until 157 are the process of iteration in the Kmeans Algorithm. Line 111 is to create new dataframe to contain the values of iterations. Line 113 is to define the total amount of iterations. Line 114 is to lock the loop in Line 116 until indefinine times. Line 120 and 121 is to calculate the value of euclidean distance between each data and centroid points. Line 122 and 123 is to include the original data in every documents. Line 124 is to define the cluster which the data belongs to in the dataframe. Line 125 is to insert the value of Line 124 into the iteration dataframe. Line 133 is to define which documents is belongs to centroid point 1. Line 136 is to calculate total of data that belongs to centroid point 1.

Line 139 is to summarize the value of datas that belongs to centroid point 1. Line 141 is to divide the summarize in Line 139 and the value at Line 136. Line 141 also means that this line is to define the new centroid point 1 value. From line 144 until 148 is the same methods to define the new centroid point 2 value. Line 150 is to add 1 to existing iteration count. This is useful to calculate how much iteration has been processed. Line 152 is to check the if the new value of centroid 1 and centroid 2 is the same as the value before. If it is the same, then the iteration will be stopped. If it is not the same, the iteration continues with the old centroid point values replaced by the new centroid point values.

```
157. count_accuracy_c1 = ( (data_iteration['Cluster'] == 'C1') &
   (data_iteration['HS'] == 1) ) | ( (data_iteration['Cluster'] == 'C2')
   & (data_iteration['HS'] == 0) )
158.
159. jumlah_true = len(count_accuracy_c1.loc[count_accuracy_c1 == True])
160.
161. hasil_accuracy_c1 = (jumlah_true/jumlah_data) * 100
162.
163.
164. print(hasil_accuracy_c1)
```

Line 158 is to define which data is correctly clustered with the centroids and which is not. If the data is clustered to C1 and the indicator Hate Speech in the column HS is 1, or the data is clustered to C2 and the indicator Hate Speech in the column HS is 0, then it will return TRUE value. The variable will return a series of TRUE and FALSE list. Line 160 is to calculate the total of TRUE value in the series. Line 162 is to calculate the percentage accuracy of the clustering value. Line 165 is to print the accuracy of Line 162.

For example, by using 10 datas with final iteration table as below.

Table 1.7 : Example Table

Document	C1	C2	Value	HS	Cluster
0	0.226037606	0.348008678	1.817382067	1	C1
1	0.364667042	0.209379241	1.956011503	0	C2
2	0.185718494	0.388327789	1.777062955	0	C1
3	0.225185614	0.799231897	1.366158848	0	C1
4	0.503296478	0.070749805	2.094640939	1	C2
5	0.186570486	0.760616769	1.404773975	1	C1
6	0.711240632	0.137194349	2.302585093	0	C2
7	0.62083013	0.046783847	2.212174591	0	C2
8	0.533002785	0.041043498	2.124347247	0	C2
9	0.711240632	0.137194349	2.302585093	0	C2

C1 column is the data distance between C1 and the document. C2 column is the same as C1. Value column is the real data in the dataset. HS column is to indicates which is Hate Speech and not. Cluster column is the considered centroid point based on the least distance between C1 and C2 columns. Now base on Line 158 code, the variable will contain list value as below.

Table 1.8 : List Table

0	True
1	True
2	False
3	False
4	False
5	True
6	True
7	True
8	True
9	True

This means that the first document which is Document 0 has the requirement that Line 158 given which is Cluster column is C1 and HS column is 1. Document 1 has also fulfill the requirement too which is Cluster column is C2 and HS column is 0. This is because that Cluster is C1 and HS column 1 returns true, or Cluster column is C2 and HS column is 0 also returns true. Line 160 is to count the amount of True in the list. Line 162 is to count the percentage which is 7 data out of 10 total data is True, then divide 7 with 10, then multiply it to 100. And then the accuracy is contained in the named variable in Line 162.

5.2 Testing

For testing, start from 10 data to be processed. After going through preprocessing and got weight of tf-idf, then the algorithm will be tested. First the centroid point value will be set to document 0 and document 1. Document 0 is indicated as Hate Speech and document 1 is indicated as Not Hate Speech. The result coming from the algorithm is that from 10 data, we have 70% accuracy with 3 iterations.

Then the data will be added to 20. the centroid point values are still the same. From 20 datas the result is 60% with 4 iterations. With 30 datas are 66,7% accuracy and 3 iterations. This testing will be conducted until 100 datas tested as shown with below table.

Table 1.9 : Accuracy Table

Total Data	Accuracy	Centroid 1	Centroid 2	Iteration
10	70%	0	1	3
20	60%	0	1	4
30	66,7%	0	1	3
40	55%	0	1	5
50	54%	0	1	7
60	53,33%	0	1	6
70	58,57%	0	1	5
80	53,75%	0	1	6
90	54,44%	0	1	6
100	56%	0	1	5

With the results shown above, few of the results can be effectively boosted by changing centroid points to affect overall accuracy value. After the process, the results obtained are as follows.

Table 1.10 : Effective Accuracy Table

Total Data	Accuracy	Centroid 1	Centroid 2	Iteration
10	70%	0	1	3
20	60%	0	1	4
30	66,7%	0	1	3
40	55%	0	1	5
50	54%	0	1	7
60	53,33%	0	1	6
70	58,57%	0	1	5
80	55%	2	1	3
90	55,55%	0	8	7
100	56%	0	1	5

As the results above, centroid point shown that it will affect accuracy value. Because of the amount of time used when searching the effective centroid point based from the amount of data, 1000 data takes at least 7+ hours to calculate. With the default centroid point, 1000 data have 60,6% accuracy and have 10 iterations in total. 5000 data have 59,06% accuracy and 22 iterations. 10000 data have 59,3% accuracy and 19 iterations.

However by using N-Gram methods before TF-IDF, the results will be different. After using N-Gram then the next step is the same which is to calculate the weight of TF-IDF. Below are the results obtained.

Table 1.11 : N-Gram Results

Total Data	Uni-Gram	Bi-Gram	Tri-Gram
10	70%	80%	80%
20	60%	50%	50%
30	66,7%	43,33%	56,66%
40	55%	62,5%	62,5%
50	54%	46%	62%
60	53,33%	51,67%	58,33%
70	58,57%	54,28%	58,57%
80	55%	56,25%	60%
90	55,55%	54,44%	60%
100	56%	57%	60%

For the true false analysis, below is the formula and the data given below is for 100 data.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Accuracy = \frac{37 + 19}{37 + 23 + 21 + 19}$$

$$Accuracy = 56\%$$

Below is the confusion matrix of 100 data.

Table 1.12 : Precision Recall Table

	Predicted: Hate Speech	Predicted: Not Hate Speech
Actual: Hate Speech	TP=37	FN=21
Actual: Not Hate Speech	FP=23	TN=19