Technical Report of the Document Clustering Project with Python

Student Name: Ramin Badri
Student ID: 9414163002
Supervisor: Dr. Amir Khani

Fall 2017

Table of Contents

Section 1: General Objectives of the Chapter

In this chapter, we aim to examine the application of data mining in textual datasets. By merging the concepts of textual data with clustering, we will arrive at a broader topic known as Document Clustering.
In fact, we will utilize data mining algorithms with the approach that our data -unlike usual- is text and, generally, documents. In text mining, we aim to extract hidden, valuable, and meaningful insights and relationships from seemingly ordinary and worthless textual data.

Section 2: Dataset characteristics

The data used in this project is inspired by a well-known travel agency company called TripAdvisor. This dataset includes 1,850 textual data entries (documents), each containing reviews and ratings provided by the site -based on the opinions of previous travelers- regarding each hotel. Since this number of documents is large for our work and slows down execution, the first 100 documents from this collection have been selected as the input and raw data. This selection does not follow any specific order and can be purely random as well.

Section 3: Programming Language and IDE Description

The programming languages R and Python are the most well-known and powerful languages for data science. In this project, we have used *Python 3.5*; there is a powerful distribution of Python called *Anaconda*, which has been specifically designed and developed for such projects. This distribution includes various packages and libraries for implementing data mining techniques.

Both the Anaconda environment and all Python IDEs can be used to develop and execute the program code. In this project, we utilized the popular Python programming IDE called *PyCharm Community*, version 2016.1.4.
The only remaining point is that we must first provide the path of the Anaconda interpreter to the software through *File → Settings → Project* Interpreter so that its libraries become operational and ready to import and use.

Section 4: Data Cleansing and Clustering Phase

In general, the program consists of 4 functions, which are illustrated in the figure below:

```
def tokenize_and_stem_and_removestopwords(input_text):...

def k_means(tfidfvector, numofclusters):...

def show_result(result_array, clustercounts):...

def draw_shape(a, b, c):...
```

Initially, the following code snippet is executed:

```
1   import pandas as pd
2   import nltk
3   from nltk.stem.porter import *
4   import string
5   from sklearn.feature_extraction.text import TfidfVectorizer
6   from sklearn.cluster import KMeans
7   import os
8   import matplotlib.pyplot as plt
9   from sklearn.manifold import MDS
10  from sklearn.metrics.pairwise import cosine_similarity
11  from scipy.cluster.hierarchy import ward, dendrogram
12
13  path = "C:/Users/vaio/Desktop/ramin-project/source/TripAdvisor_First_100_Hotels"
14  token_dict = {}
15  stopwords = nltk.corpus.stopwords.words('english')
16  stemmer = PorterStemmer()
17  document_names = []
```

In the above figure, all the necessary libraries and packages for the project are first imported. Then, in lines 12 to 17, we specify the path of the dataset and prepare the groundwork for the body of the codes.
Note: To run the program, you must change the given directory shown as the *"path"* variable to the existing path on your hard drive.

Next, the following section will be run and executed:

```
120     # iterate throw data collection
121     for subdir, dirs, files in os.walk(path):
122         for file in files:
123             file_path = subdir + os.path.sep + file
124             shakes = open(file_path, encoding="utf8")
125             new_path = file_path.replace(file_path, file_path[72:90])
126             document_names.append(new_path)
127             text = shakes.read()
128             lowers = text.lower()
129             no_punctuation = lowers.translate(string.punctuation)
130             token_dict[file] = no_punctuation
```

In this section, we read the files in the initial path one by one, convert the text inside them to lowercase, and store the result in *token_dict* which is a list object. Additionally, we will store a list of the names of each document in the *document_names* list object.

Then it's time to execute the following code snippet:

```
140 flag = True
141 ans_1 = input("Please enter number of clusters(k for k-means): ")
142 while flag:
143     try:
144         val = int(ans_1)
145         flag = False
146     except ValueError:
147         print("That's not an int!")
148 tfidf = TfidfVectorizer(tokenizer=tokenize_and_stem_and_removestopwords, stop_words='english', ngram_range=(1, 2))
149 tfs = tfidf.fit_transform(token_dict.values())
```

With the execution of the above code, the user is first asked about setting the parameter K, which represents the number of clusters. Then, using the definition made in line 133, the initial settings for creating the *tf-idf matrix* for the tokens are established. An important point is considering the parameter *ngram_range=(1,2)*, which indicates that we also want to obtain the 1-grams and 2-grams of each token (Following RapidMiner parameters). The tokenizer parameter specifies how the tokens are extracted, which is set to the function *tokeniz_and_stem_and_removestopwords()*, which we will explain further. Finally, we assign *token_dict* for execution to *tfidf* (line 135).

Now, the function *tokeniz_and_stem_and_removestopwords()* is executed. Its internal body is shown in the figure below. As the name suggests, this function takes text as input and performs tokenization, stemming, and the removal of stopwords in the English language. Ultimately, the output of this function consists of words ready for data mining. For this section, we have utilized the *nltk library*.

```
21  def tokenize_and_stem_and_removestopwords(input_text):  # Tokenize, Stem and remove stopwords
22      tokens = [word for sent in nltk.sent_tokenize(input_text) for word in nltk.word_tokenize(sent)]
23      filtered_stopwords = [w for w in tokens if w not in stopwords]
24      filtered_tokens = []
25      for token in filtered_stopwords:
26          if re.search('[a-zA-Z]', token):
27              filtered_tokens.append(token)
28      stems = [stemmer.stem(t) for t in filtered_tokens]
29      return stems
```

The *stopwords* and *stemmer* variables that were initialized earlier were used here (in the figure above). In this function, the operations of token extraction (line 22), removal of frequent words (line 23), removal of all characters except for the English alphabet (lines 25 to 27), and finally stemming (line 28) are performed. The output of this function is stored in the variable *tfs* after being converted to *tf-idf* numerical vectors.

Now the following snippet will be executed, The *k_means() function* is called with the parameters *tfs* and *ans_1* (which represents the number of clusters). In this function, the main operation of clustering the documents is performed. The input parameters of this function are the *tf-idf* matrix of the documents and the number of clusters. The internal body of this function is illustrated in the figure below.

```
136  result = k_means(tfs, ans_1)
```

```
32  def k_means(tfidfvector, numofclusters):
33      cluster_id = []
34      doc_id_per_cluster = []
35      km = KMeans(n_clusters=numofclusters)
36      km.fit(tfidfvector)
37      clusters = km.labels_.tolist()
38      for k in range(0, numofclusters):
39          cluster_id.append([])
40          doc_id_per_cluster.append([])
41      for k in range(0, len(clusters)):
42          for i in range(0, numofclusters):
43              if clusters[k] == i:
44                  cluster_id[i].append(clusters[k])
45                  doc_id_per_cluster[i].append(k)
46      return clusters, cluster_id, doc_id_per_cluster
```

Now it is time to display the results obtained from the clustering in the output. To do this, we pass the output of the *k_means* function to the *show_result()*(result, ans_1) function. Additionally, the number of clusters is also one of the input parameters of this function (line 151). The internal body of this function is shown below.

```
51  def show_result(result_array, clustercounts):
52      print("**There are ", len(result_array[0]), " documents in TripAdvisor data collection**")
53      print("**The data collection is clustered in ", clustercounts, "clusters as shown in below**")
54      print("###################")
55      for j in range(0, len(result_array[1])):
56          print("There are: ", len(result_array[1][j]), " documents in cluster: ", j)
57          print("Here we can see names of the documents:")
58          print()
59          for m in result_array[2][j]:
60              print(document_names[m])
61          print()
62          print("==================")
```

After executing the above function, we can observe the clustering broken down by the name and the number of documents present in each cluster. We will see this result in the results analysis section.

After executing the above function, most of the clustering work has been completed. The only remaining part is the section below, which begins with a question for the user.

```
152  ans_2 = input("Do you want to see clustering visual shapes? y or n: ")
153  if ans_2 == "y":
154      draw_shape(result[0], document_names, tfs, int(ans_1))
```

Two different types of charts have been prepared for the graphical representation of the document clustering. If the user wishes to view them, they should answer "y" to the above question and press the Enter key. In this case, the last function of this project, namely the *draw_shape()* function, will be called with the input parameters *result[0] = clusters*, *document_names*, *tfs*, and the number of clusters (*ans_1*). The first part of the internal body of this function is shown in the figure below.

```
65  def draw_shape(a, b, c, d):
66      dist = 1 - cosine_similarity(c)
67      mds = MDS(n_components=2, dissimilarity="precomputed", random_state=1)
68      pos = mds.fit_transform(dist)
69      xs, ys = pos[:, 0], pos[:, 1]
70      cluster_colors = []
71      cluster_names = []
72      prefix = "cluster_"
73      for i in range(0, d):
74          cluster_colors.append(colorize())
75          cluster_names.append(prefix+str(i))
76      df = pd.DataFrame(dict(x=xs, y=ys, label=a, title=b))
77      groups = df.groupby('label')
78      fig, ax = plt.subplots(figsize=(17, 9))  # set size
79      ax.margins(0.05)  # Optional, just adds 5% padding to the autoscaling
```

```
80      for name, group in groups:
81          ax.plot(group.x, group.y, marker='o', linestyle='', ms=12,
82                  label=cluster_names[name], color=cluster_colors[name], mec='none')
83          ax.set_aspect('auto')
84          ax.tick_params(
85              axis='x',  # changes apply to the x-axis
86              which='both',  # both major and minor ticks are affected
87              bottom='off',  # ticks along the bottom edge are off
88              top='off',  # ticks along the top edge are off
89              labelbottom='off')
90          ax.tick_params(
91              axis='y',  # changes apply to the y-axis
92              which='both',  # both major and minor ticks are affected
93              left='off',  # ticks along the bottom edge are off
94              top='off',  # ticks along the top edge are off
95              labelleft='off')
96      ax.legend(numpoints=1)  # show legend with only 1 point
97      # add label in x,y position with the label as the film title
98      for i in range(len(df)):
99          ax.text(df.ix[i]['x'], df.ix[i]['y'], df.ix[i]['title'], size=5)
100     plt.show()
101     plt.close()
```

By executing the above section and with the help of the *matplotlib.pyplot library*, which is specifically for drawing various charts, we can observe the clustering results graphically, differentiated by color (each document in a cluster has a unique color). An important point is that a different color and name are generated for the number of clusters that the user provided in the first step of the program (parameter k). Lines 70 to 75 contain the remaining lines of the program related to chart drawing settings, which we will skip explaining. In the next section, we will see this chart.

The second part of this function is related to drawing the *hierarchical clustering* chart of the documents, where we do not focus on the number of clusters but rather perform clustering based on the similarity obtained between the documents (line 66 – cosine similarity). The result of this section is the well-known Dendrogram chart. The code for this part is also shown in the figure below.

An important point in this section is the name in line 106, which is set for the parameter *labels = document_names*. This ensures that the names of the documents are displayed as labels on the leaves of the chart. This chart is also drawn with the help of the *matplotlib.pyplot library*. In the next section, we will also see the result of this part.

```
    # dendrogram plot
    linkage_matrix = ward(dist)  # define the linkage_matrix using ward clustering pre-computed distances
    fig, ax = plt.subplots(figsize=(15, 20))  # set size
    ax = dendrogram(linkage_matrix, orientation="top", labels=document_names, leaf_font_size=6)
    plt.tick_params(
        axis='x',  # changes apply to the x-axis
        which='both',  # both major and minor ticks are affected
        bottom='on',  # ticks along the bottom edge are off
        top='on',  # ticks along the top edge are off
        labelbottom='on')
    plt.show()
    plt.close()
    return True
```

Section Five: Displaying and Analyzing Results

Before starting this section, it is essential to mention that since the K-means algorithm operates randomly at its initial stage and selects starting points without a specific order, different output results may be observed each time the program is executed.

After executing the program, we can observe the following result, which is part of the output of this stage (here we consider K=5):

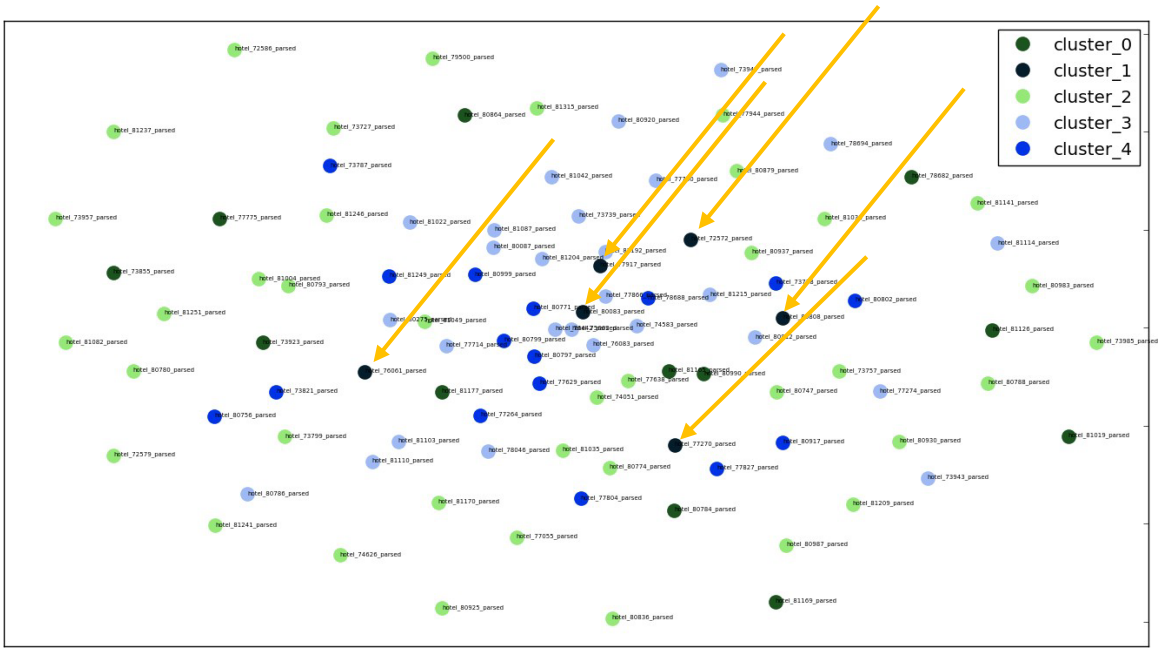
```
D:\Anaconda3\python.exe C:/Users/vaio/ramin/__init__.py
Please enter number of clusters(k for k-means): 5
**There are  100  documents in TripAdvisor data collection**
**The data collection is clustered in  5 clusters as shown in below**
##################
There are:  12  documents in cluster:  0
Here we can see names of the documents:

hotel_73855_parsed
hotel_73923_parsed
hotel_77775_parsed
hotel_78682_parsed
hotel_80784_parsed
hotel_80864_parsed
hotel_80990_parsed
hotel_81019_parsed
hotel_81126_parsed
hotel_81165_parsed
hotel_81169_parsed
hotel_81177_parsed

==================
There are:  6  documents in cluster:  1
Here we can see names of the documents:

hotel_72572_parsed
hotel_76061_parsed
hotel_77270_parsed
hotel_77917_parsed
hotel_80083_parsed
hotel_80808_parsed

==================
There are:  39  documents in cluster:  2
Here we can see names of the documents:

hotel_72579_parsed
hotel_72586_parsed
hotel_73727_parsed
```

Here, we can see the names of the documents within each cluster in a separate manner. For example, in cluster number 0, there are 12 documents, and their names are as follows from top to bottom:

```
hotel_73855_parsed
hotel_73923_parsed
hotel_77775_parsed
hotel_78682_parsed
hotel_80784_parsed
hotel_80864_parsed
hotel_80990_parsed
hotel_81019_parsed
hotel_81126_parsed
hotel_81165_parsed
hotel_81169_parsed
hotel_81177_parsed
```

For another example, we will focus on the documents within cluster number 1. This cluster contains 6 documents, which are highlighted in the first above figure. Now, let's move on to displaying the charts.

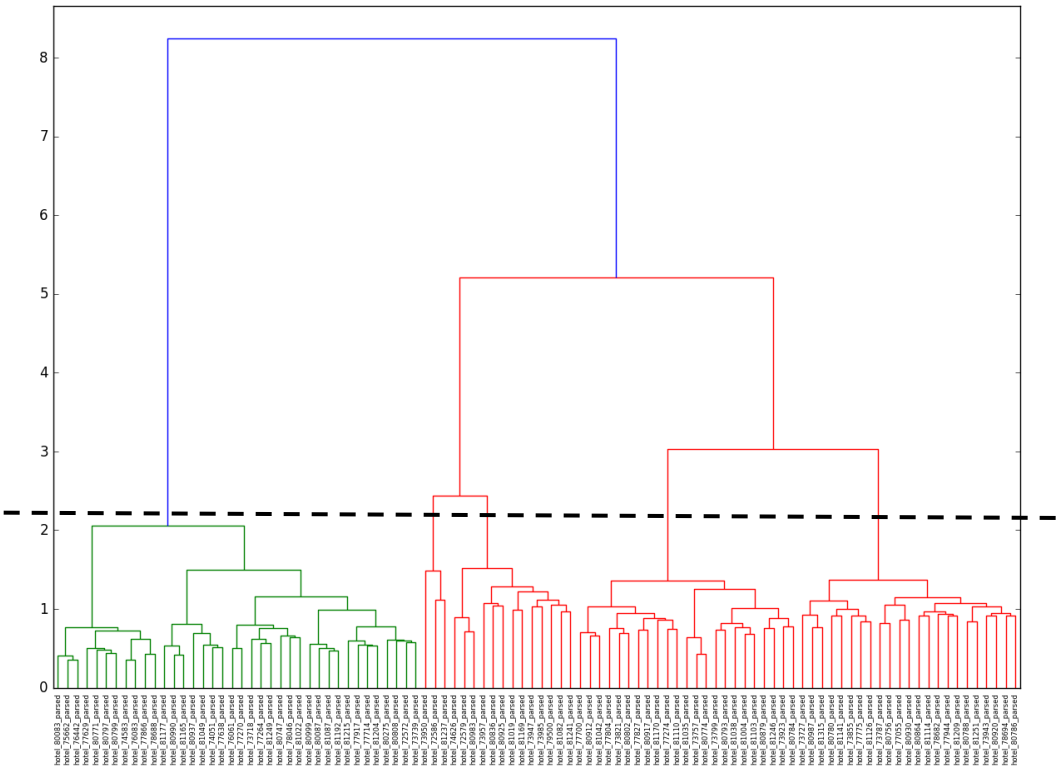
To view the charts, the user must respond with "y" to the question asked by the program. In this case, the chart will appear first. However, due to the large number of documents, it can be somewhat difficult to observe and read the names of the documents. In the chart below, we can see the same 6 documents related to cluster 1, which are shown in black.



The selection of colors is also random. After closing the above chart window, the Dendrogram related to this clustering will appear, which we can see in the figure below.

Unfortunately, even with zooming in on the figure, the names of the documents are not clearly visible due to the large number of leaves. Ultimately, this issue will be resolved when the user runs the program. It can be observed that, in general, the documents are divided into two clusters (at the highest level of clustering).

To achieve clustering with *k=5*, we need to cut the chart at the section indicated by the dashed line in the figure.



All the program codes along with their output are available in a separate file as plain text.