MODULE 1: MAKING SENSE OF UNSTRUCTURED DATA

CASE STUDY ACTIVITY TUTORIAL

CASE STUDY 2 – SPECTRAL CLUSTERING NEW STORIES

CASE STUDY ACTIVITY TUTORIAL

CASE STUDY 2 – SPECTRAL CLUSTERING NEW STORIES

Faculty: Stefanie Jegelka

In this document, we walk through some helpful tips to get you started with building your own application for automating the clustering of news stories using Spectral Clustering. In this tutorial, we provide examples and some pseudo-code for the following programming environment: **Python**. We cover the following:

Topics

Gathering News Articles

For this tutorial, it is assumed that you have access to electronic version of news stories as text, ideally all from a single day. It is suggested that you have at least 50 articles. You can extract the content of these articles, including their titles, manually or by using Python-based tools like Python-Goose (https://github.com/grangier/python-goose).

The rest of this tutorial assumes that you have access to the content of each of the news articles in the following format:

> Filename: **title-**<*article_numer*>**.txt**, example: title-1.txt
> Contents: **The title of the news story.**
>
> Filename: **article-**<*article_numer*>**.txt**, example: article-1.txt
> Contents: **The contents of the news story.**

In addition, it is also recommended that you tag each article with the actual "topic" which will help us evaluate the performance of the spectral clustering algorithm. This can be done by assigning the name of the hierarchical identifier for the news story on the website it is hosted on. For instance, news stories about "Brexit" were typically classified under the "Brexit" section in most online news websites. Stories about the "Middle East" are typically classified under "Middle East", and so on.

If this information is available, it can be made available in the following format:

Filename: **topic-**<*article_numer*>**.txt**, example: topic-1.txt
Contents: **The actual "topic" (section or sub-section) under which the news story was classified on the hosting website.**

Once, the original data is available in the text file format suggested here, we can then run the following lines of code, in Python, to have access to them for the rest of our script:

```
# total number of articles to process
N = 100

# in memory stores for the topics, titles and contents of the news stories
topics_array = []
titles_array = []
corpus = []

for i in range(0, N):

    # get the contents of the article
    with open("article-" + str(i) + ".txt", 'r') as myfile:
        d1=myfile.read().replace('\n', '')
        d1 = d1.lower()
        corpus.append(d1)

    #get the original topic of the article
    with open("topic-" + str(i) + ".txt",  'r') as myfile:
        to1=myfile.read().replace('\n', '')
        to1 = to1.lower()
        topics_array.append(to1)

    #get the title of the article
    with open("title-" + str(i) + ".txt",  'r') as myfile:
        ti1=myfile.read().replace('\n', '')
        ti1 = ti1.lower()
        titles_array.append(ti1)
```

Entity Extraction

We now want to represent each article's contents (corpus) as a "bag of entities". This simply means that we will look for the mentions of certain words, i.e. names of people, organizations, locations etc. The master list of such entities can be found using various publicly available libraries. One such Python-based library is called "MITIE" (https://github.com/mit-nlp/MITIE). You will need to install it, and make a note about the path to the library. This case be stored in the variable:

```
path_to_mitie_lib
```

MITIE provides several entity models by default. We will be using the entities in the model "NER" which can be found under the following path (under the MITIE source code/installation folder):

'/MITIE-models/english/ner_model.dat'

Store the path to the ner_mode.dat in the following variable:

path_to_ner_model

Now, add the path to the MITIE library to the sys.path in Python:

```
import sys
sys.path.append(path_to_mitie_lib)
```

You should now be able to import the MITIE library for use in the rest of this script:

```
from mitie import *
```

We are now ready to do the following:

1. Loop over all the article text corpuses to determine all the unique words used across our dataset.
2. Find the subset of the entities from the ner model that are among the unique words being used across the dataset (determined in step 1).

This goal can be achieved using the following lines of code:

```
# entity subset array
entity_text_array = []

for i in range(0, N):

    # Load the article contents text file and convert it into a list of words.
    tokens = tokenize(load_entire_file((“article-” + str(i) + “.txt”))

    # extract all entities known to the ner model mentioned in this article
    entities = ner.extract_entities(tokens)

    # extract the actual entity words and append to the array
    for e in entities:
        range_array = e[0]
        tag = e[1]
        score = e[2]
        score_text = "{:0.3f}".format(score)
        entity_text = " ".join(tokens[j] for j in range_array)
        entity_text_array.append(entity_text.lower())

    # remove duplicate entities detected
    entity_text_array = np.unique(entity_text_array)
```

TF-IDF

Now that we have the list of all entities used across our dataset, we can represent each article as a vector that contains the TF-IDF (https://en.wikipedia.org/wiki/Tf–idf) score for each entity stored in the *entity_text_array*. This task can easily be achieved by using the scikit-learn library (http://scikit-learn.org/stable/) for Python. Please ensure scikit-learn is installed and ready to use before

proceeding. The following lines of code can help represent each article in the dataset as a vector of TF-IDF values:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vect = TfidfVectorizer(sublinear_tf=True, max_df=0.5, analyzer='word',
        stop_words='english', vocabulary=entity_text_array)

corpus_tf_idf = vect.fit_transform(corpus)
```

Spectral Clustering

Now that we have the articles represented as vectors of their TF-IDF scores, we are ready to perform Spectral Clustering on the articles. We can use the scikit-learn library for this purpose as well. The following lines of code will cluster our articles in to 7 clusters. You can choose the number of clusters to suit your requirements:

```
from sklearn import cluster

# change n_clusters to equal the number of clusters desired
n_clusters = 7
n_components = n_clusters

#spectral clustering
spectral = cluster.SpectralClustering(n_clusters= n_clusters,
                                      eigen_solver='arpack',
                                      affinity="nearest_neighbors",
                                      n_neighbors = 10)

spectral.fit(corpus_tf_idf)
```

Output

We now have the spectral clustering model fitted to the dataset. The following lines of code will help us see the output in the following format (one line per article):

article_number, topic, spectral_clustering_cluster_number, article_title

```
if hasattr(spectral, 'labels_'):
   cluster_assignments = spectral.labels_.astype(np.int)

for i in range(0, len(cluster_assignments)):
   print (i, topics_array[i], cluster_assignments [i], titles_array[i])
```