江南大学

Curriculum Design of Programming
Final Report

School of Artificial Intelligence and Computer Science

Computer science and technology

Student Number: 5035190136

Student Name: Ejafa Bassam

Teacher Name: Wei Song

July 2020

Task: Inputting a given English document, and we have to design a clustering algorithm for clustering the sentences of the input document. Subsequently, selecting the most relevant sentence in each cluster to express the meaning of this cluster and a series of relevant sentences selected from different clusters can be used to represent the abstract of the document. Finally, we will output the generated document abstract, which can objectively reflect the content of the input document.

Introduction: With the growth of websites, the information in the internet has grown huge over the past few decades. So, the necessity of a good automatic summarization is increasing.

Automatic summarization is the process of shortening a set of data computationally, to create a subset (a summary) that represents the most important or relevant information within the original content.

There are two general approaches to automatic summarization:

- ***Extraction***
- ***Abstraction***

Extraction-based summarization:

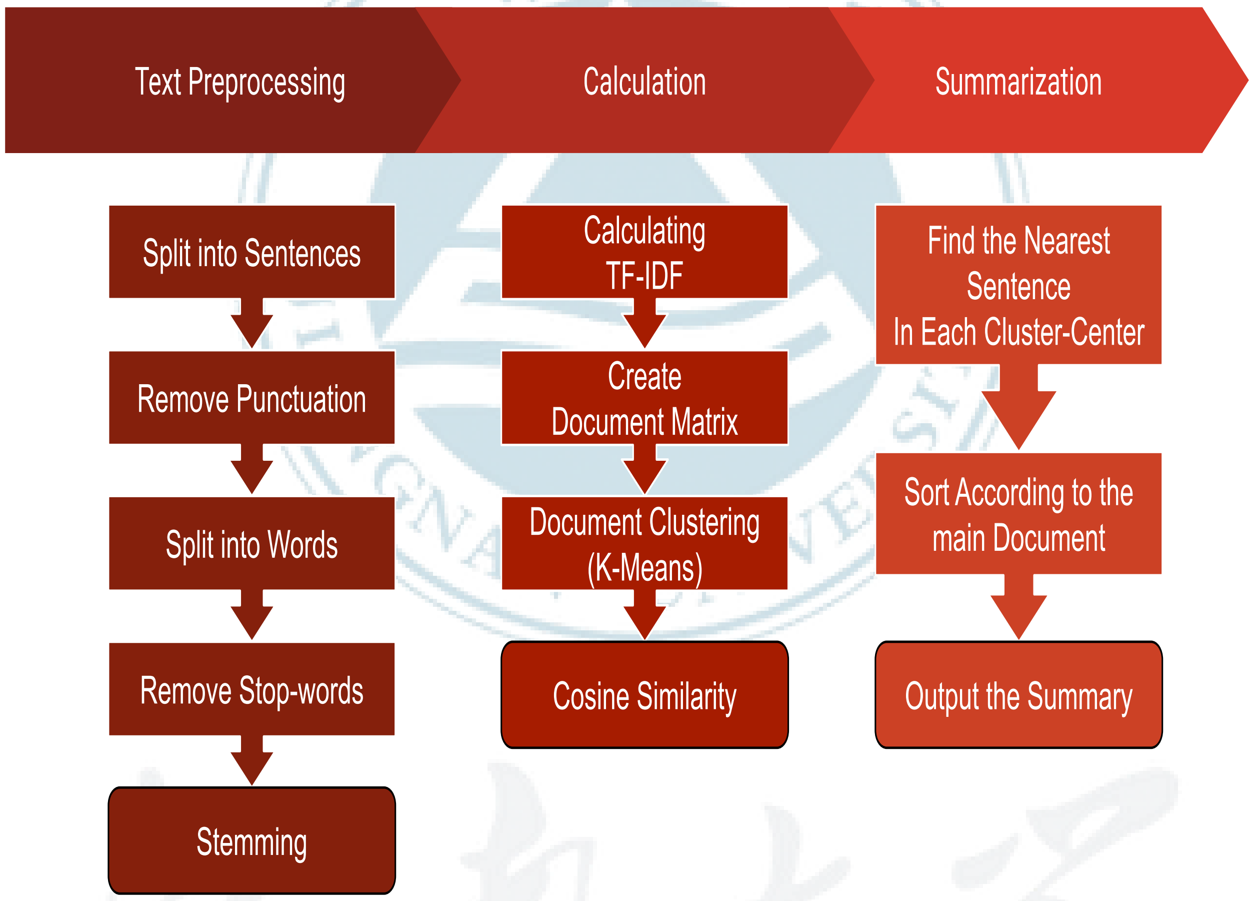
Here, content is extracted from the original data, but the extracted content is not modified in any way.

Abstraction-based summarization:

This has been applied mainly for text. Abstractive methods build an internal semantic representation of the original content, and then use this representation to create a summary that is closer to what a human might express.

For this programming design we will choose to work with ***Extraction-based summarization.***

Diagram:



Document Preprocessing: For preprocessing the document, we will follow the code.

First, we will download `nltk` and `sklearn` library and read the file:

```
import io
import os
os.system('pip install nltk')
os.system('pip install sklearn')

import nltk

nltk.download('stopwords')
nltk.download('punkt')

filename = input("name of file to summarize: ")
f = open(filename,"r+")
text = f.read()
```

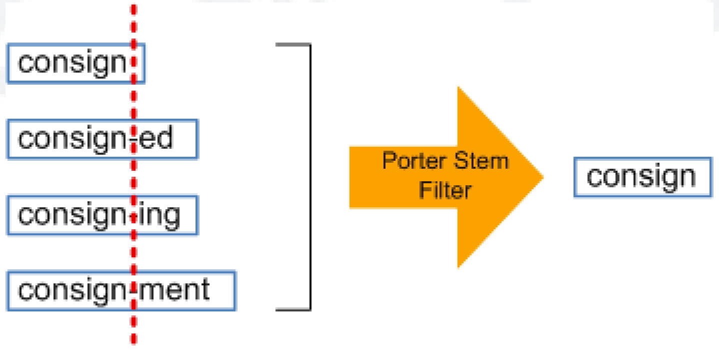
Then, we will Preprocess the document with `nltk` library:

```
# convert to lower case
text = text.lower()

# seperate sentences
from nltk import sent_tokenize
sentences = sent_tokenize(text)

# remove punctuation from each line
import string
sentences = [line.translate(str.maketrans('', '', string.pu
nctuation)) for line in sentences]

# split into words
from nltk.tokenize import word_tokenize
t_doc = [word_tokenize(line) for line in sentences]
```

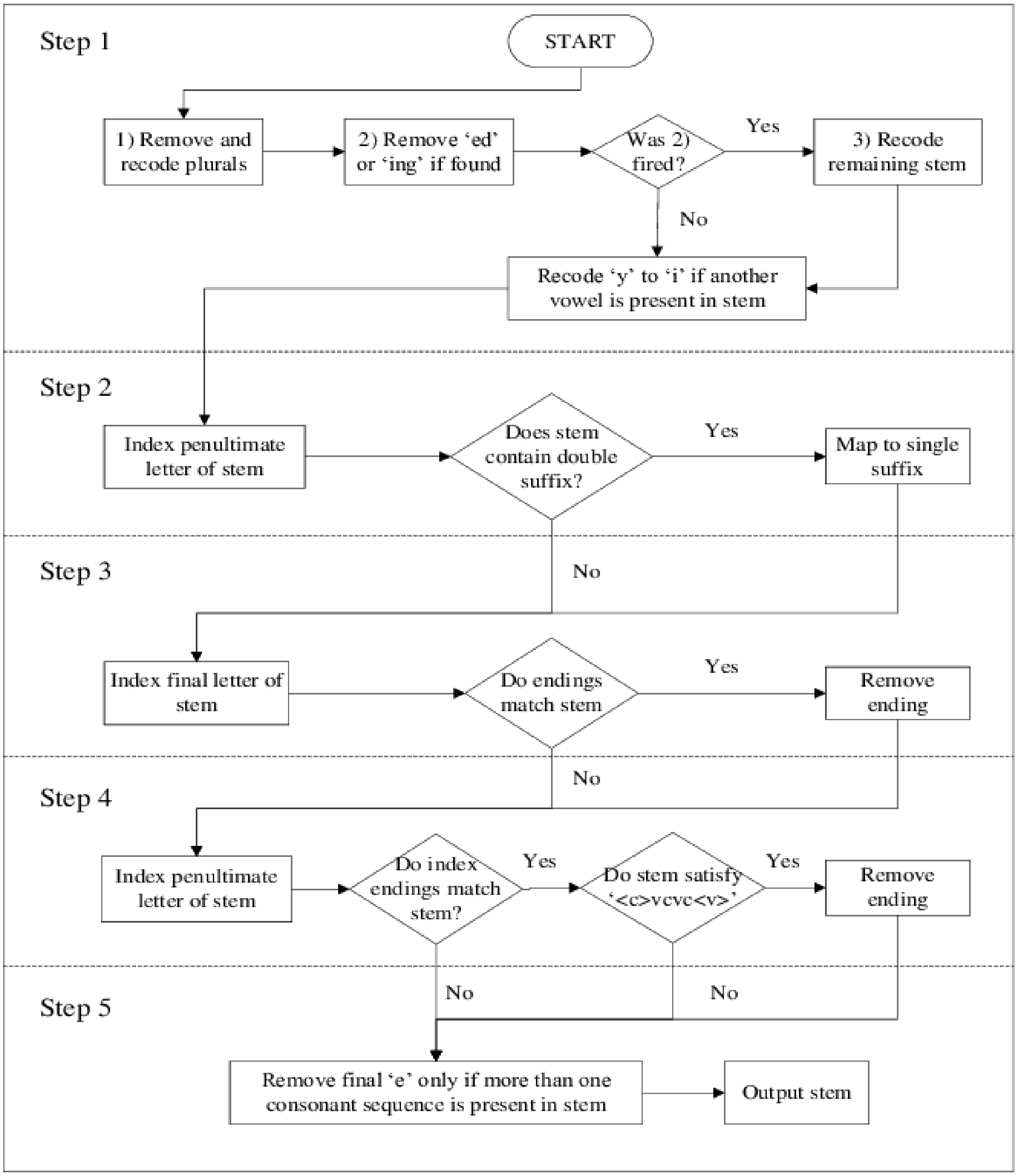
```
# filter out stop words
# and stemming using PorterStemmer

from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

stop_words = set(stopwords.words('english'))
porter = PorterStemmer()

for line in t_doc:
  for word in line:
    if word in stop_words:
      t_doc[t_doc.index(line)].remove(word)
    else: # using PorterStemer
      t_doc[t_doc.index(line)][line.index(word)] = porter.s
tem(word)
```

***Stop words**: stop words are words which are filtered out before or after processing of natural language data (text). such as he, she, the, is, at, which, and on.*

***Porter stemming**: The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and in flexional endings from words in English. Its main use is as part of a term normalization process that is usually done when setting up Information Retrieval systems.*



Flow-chart of Porter Stemmer:


Step 1
START
1) Remove and recode plurals
2) Remove 'ed' or 'ing' if found
Was 2) fired?
Yes
3) Recode remaining stem
No
Recode 'y' to 'i' if another vowel is present in stem
Step 2
Index penultimate letter of stem
Does stem contain double suffix?
Yes
Map to single suffix
Step 3
No
Index final letter of stem
Do endings match stem
Yes
Remove ending
No
Step 4
Index penultimate letter of stem
Do index endings match stem?
Yes
Do stem satisfy '<c>vcvc<v>'
Yes
Remove ending
Step 5
No
No
Remove final 'e' only if more than one consonant sequence is present in stem
Output stem

Now we will merge the stemmed words so that we can apply TF-IDF weighting function of `sklearn`

```
# creating stemmed sentences

t_doc_s = list()
for line in t_doc:
  t_doc_s.append(" ".join(line[:]))
```

Vector Space Model: Vector space model or term vector model is an algebraic model for representing text documents.

Definitions: *Documents and queries are represented as vectors.*

$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$$
$$q = (w_{1,q}, w_{2,q}, \ldots, w_{n,q})$$

Each dimension corresponds to a separate term. If a term occurs in the document, its value in the vector is non-zero. Several different ways of computing these values, also known as (term) weights, have been developed. One of the best-known schemes is TF-IDF weighting.

Term frequency-inverse document frequency weights*: In information retrieval,* `tf–idf` *or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.*

It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The `tf–idf` *value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.*

Term Frequency (tf): *gives us the frequency of the word in each document in the corpus. It is the ratio of number of times the word appears in a document compared to the total number of words in that document. It increases as the number of occurrences of that word within the document increases. Each document has its own tf.*

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Inverse Data Frequency (idf): *used to calculate the weight of rare words across all documents in the corpus. The words that occur rarely in the corpus have a high IDF score. It is given by the equation below.*

$$idf(w) = log(\frac{N}{df_t})$$

TF-IDF*: Combining these two we come up with the TF-IDF score (w) for a word in a document in the corpus. It is the product of tf and idf:*

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$$\begin{aligned} tf_{i,j} &= \text{number of occurrences of } i \text{ in } j \\ df_i &= \text{number of documents containing } i \\ N &= \text{total number of documents} \end{aligned}$$

The weight vector for document d is

$$\mathbf{v}_d = [w_{1,d}, w_{2,d}, \ldots, w_{N,d}]^T$$ *, where*

$$w_{t,d} = \mathrm{tf}_{t,d} \cdot \log \frac{|D|}{|\{d' \in D \,|\, t \in d'\}|}$$

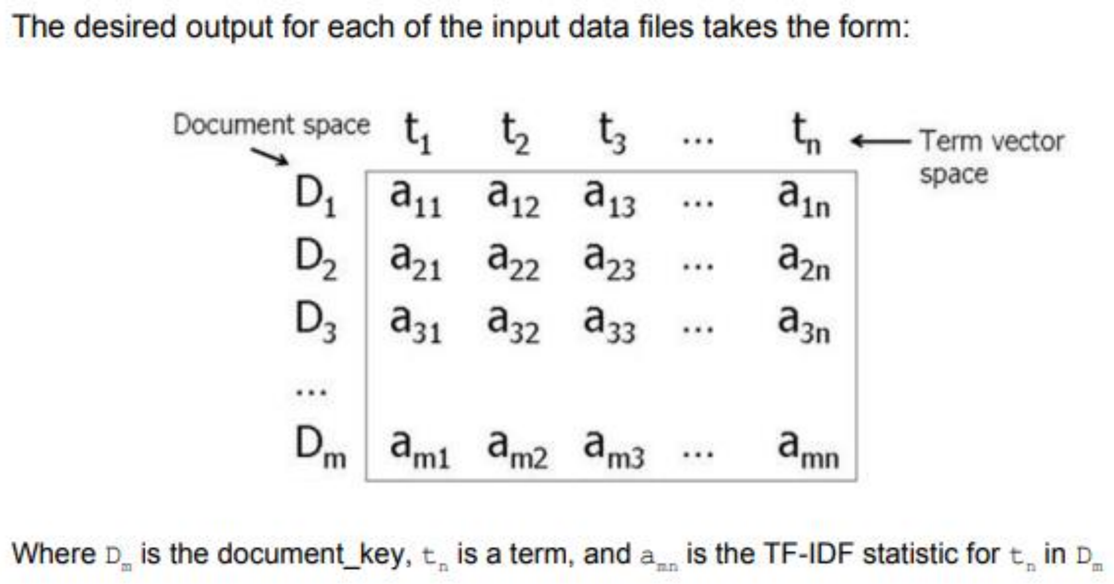
$\mathrm{tf}_{t,d}$ *is term frequency of term t in document d (a local parameter)*

$\log \frac{|D|}{|\{d' \in D \,|\, t \in d'\}|}$ *is inverse document frequency (a global parameter).*

$|D|$ *is the total number of documents in the document set;*

$|\{d' \in D \,|\, t \in d'\}|$ *is the number of documents containing the term t.*

After calculation of TF-IDF we will get a document matrix like this:



The desired output for each of the input data files takes the form:

$$\begin{array}{c|ccccc|}
 & t_1 & t_2 & t_3 & \cdots & t_n \\
\hline
D_1 & a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\
D_2 & a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\
D_3 & a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\
\cdots & & & & & \\
D_m & a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \\
\hline
\end{array}$$

Document space
Term vector space

Where D_m is the document_key, t_n is a term, and a_{mn} is the TF-IDF statistic for t_n in D_m

```python
# Calculating tf-idf and creating document matrix

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vector = vectorizer.fit_transform(t_doc_s)
```

In here, under the hood the code looks like this

```python
def fit(self, X, y=None):
    """Learn the idf vector (global term weights)
    Parameters
    ----------
    X : sparse matrix, [n_samples, n_features]
        a matrix of term/token counts
    """
    if not sp.issparse(X):
        X = sp.csc_matrix(X)
    if self.use_idf:
        n_samples, n_features = X.shape
        df = _document_frequency(X)

        # perform idf smoothing if required
        df += int(self.smooth_idf)
        n_samples += int(self.smooth_idf)

        # log+1 instead of log makes sure terms with zero i
df don't get
        # suppressed entirely.
        idf = np.log(float(n_samples) / df) + 1.0
        self._idf_diag = sp.spdiags(idf, diags=0, m=n_featu
res,n=n_features, format='csr')

    return self
```

```
def transform(self, X, copy=True):
    """Transform a count matrix to a tf or tf-
idf representation
    Parameters
    ----------
    X : sparse matrix, [n_samples, n_features]
        a matrix of term/token counts
    copy : boolean, default True
        Whether to copy X and operate on the copy or perfor
m in-place
        operations.
    Returns
    -------
    vectors : sparse matrix, [n_samples, n_features]
    """
    if hasattr(X, 'dtype') and np.issubdtype(X.dtype, np.fl
oating):
        # preserve float family dtype
        X = sp.csr_matrix(X, copy=copy)
    else:
        # convert counts or binary occurrences to floats
        X = sp.csr_matrix(X, dtype=np.float64, copy=copy)

    n_samples, n_features = X.shape

    if self.sublinear_tf:
        np.log(X.data, X.data)
        X.data += 1

    if self.use_idf:
        check_is_fitted(self, '_idf_diag', 'idf vector is n
ot fitted')

        expected_n_features = self._idf_diag.shape[0]
        if n_features != expected_n_features:
            raise ValueError("Input has n_features=%d while
 the model"
                                     " has been trained with n_
features=%d" % (n_features, expected_n_features))
            # *= doesn't work
        X = X * self._idf_diag

    if self.norm:
        X = normalize(X, norm=self.norm, copy=False)

    return X
```

Clustering: Clustering refers to dividing the data into some aggregation classes according to the intrinsic nature of the data. The elements in each aggregation class have the same characteristics as much as possible, and the differences in characteristics between different aggregation classes are as large as possible.

The purpose of cluster analysis is to analyze whether the data belongs to separate groups so that the members of a group are similar to each other and different from the members of other groups.

The general method of cluster analysis is to group data objects into multiple clusters. Objects in the same cluster have higher similarity, and objects in different clusters have greater differences.

Document Clustering: *Document clustering transforms some documents from original text information into mathematical information and presents them in the form of high-dimensional space points. These points are gathered into a cluster by calculating the distances of those points. The center of the cluster is called the cluster center. A good clustering should ensure that the distances within the cluster are as close as possible, but the points between clusters and clusters should be as far away as possible.*

Document clustering is mainly based on well-known clustering assumptions:

- *similar documents have a large degree of similarity,*
- *different types of documents have a low degree of similarity.*

k-means clustering: *k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.*

This results in a partitioning of the data space into Voronoi cells. It is popular for cluster analysis in data mining. k-means clustering minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult Weber problem: the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances.

K-means Document Clustering:

The K-means document clustering algorithm based on cosine similarity.

Input: choose K cluster centers randomly from the weight matrix corresponding to n documents

Output: get K kinds of documents, the content of the documents in each kind of document is similar

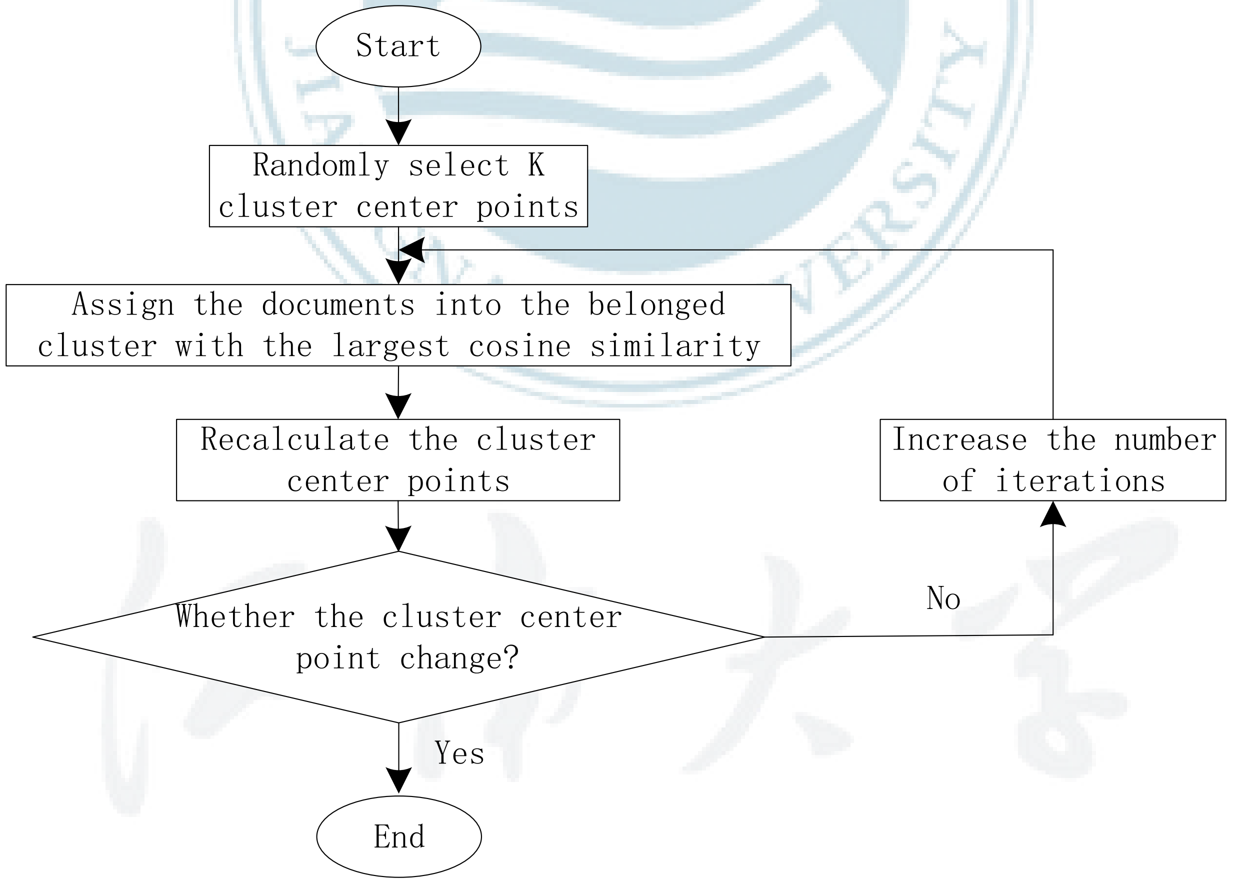
The steps of the K-means document clustering algorithm:

1. *according to the vector space model, n documents can be represented as a document matrix, and k documents are randomly selected from the document matrix as the initial cluster center.*
2. *repeat the following two steps until the cluster center point no longer changes.*
3. *calculate the cosine similarity between each document vector in the matrix and the center point of each cluster, and assign the documents into the belonged cluster with the largest cosine similarity.*
4. *recalculate cluster center points based on each type of partitioned object.*

$$C_k = (\sum_{i=1}^{m} d_j) / m$$

The k ranges from 1 to K, C_k denotes the k-th cluster center point, the d_j represents a document, $d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$, the m indicates the number of documents belonging to this class. Through the above formula can calculate K new cluster center points.

The flow chart of the K-means document clustering algorithm:


Start
Randomly select K cluster center points
Assign the documents into the belonged cluster with the largest cosine similarity
Recalculate the cluster center points
Whether the cluster center point change?
No
Increase the number of iterations
Yes
End

K-means with scikit learn: *for better result, we used scikit learn implementation of k-means clustering*

```
# k-means cluster
from sklearn.cluster import KMeans
import numpy as np

# asking for how many lines we need to summarize

n = int(input("how many output lines do you want for summar
ize: "))
kmeans = KMeans(n_clusters=n,max_iter=10000,n_init=1000).fi
t(vector)
```

As we want all the ideas from the document, we will select one sentence from each cluster. So, the clustering depends on user for the lines needed for summarization.

Information retrieval using cosine similarity: *Cosine similarity* can be seen as a method of *normalizing* document length during comparison. In the case of information retrieval, the cosine similarity of two documents will range from 0 to 1, since the term frequencies (using tf–idf weights) cannot be negative.

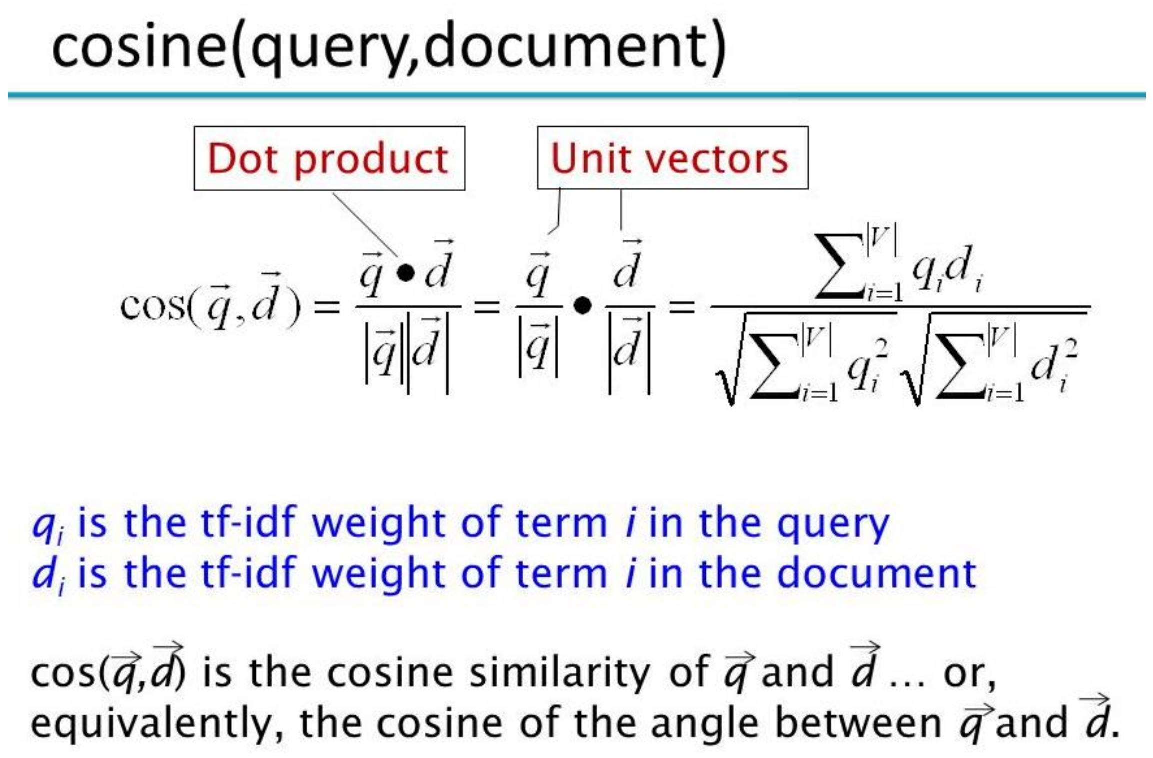
A similarity measure is a function that computes the degree of similarity between two vectors.

Using a similarity measure between the query and each document:

- It is possible to rank the retrieved documents in the order of presumed relevance.
- It is possible to enforce a certain threshold so that the size of the retrieved set can be controlled.

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined to equal the cosine of the angle between them, which is also the same as the inner product of the same vectors normalized to both have length 1.

We will match similarity with the center of the clusters from k-means cluster algorithm.


Dot product
Unit vectors

cosine(query,document)

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q} \bullet \vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|} \bullet \frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

q_i is the tf-idf weight of term i in the query
d_i is the tf-idf weight of term i in the document

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

Calculating cosine similarity:

```
# calculating cosine similarity

from sklearn.metrics.pairwise import cosine_similarity
values = cosine_similarity(kmeans.cluster_centers_,vector)
```

Combining Documents to create summarization: Now we will pick out the closest sentence near to cluster center from each cluster. Then we sort and join each sentence according to the real document.

```
# finding the nearest sentence for each cluster center with
 cosine similarity

line_no = list()
for i in values:
  line_no.append(list(i).index(i.max()))

# sorting to keep the flow of meaning
#print(values,line_no)
line_no.sort()
from nltk import sent_tokenize
s_sentences = sent_tokenize(document)
print("\nSummary: ")
for i in line_no:
  print(s_sentences[i])
```

Program input:

```
It's May, but Beijing feeling summer heat. By one
measure, season has changed; conditions could affect
wheat crop Beijing sweltered in summerlike heat on
Wednesday, with temperatures expected to remain high
until Friday night. Residents in cities from at least
eight northern provinces are also experiencing their
first heat wave of the year, the national weather
authority said on Wednesday. "They're the first really
high temperatures to hit northern China this year," said
```

Fang Chong, chief forecaster at the National Meteorological Center. "The high temperatures will continue to rise and peak on Thursday." Temperatures from Wednesday to Friday will climb as high as 40 C in parts of the Inner Mongolia autonomous region and reach 37 C in the vast regions in North China on Friday, she said. The center issued its first heat alert - a yellow warning, the third-highest - on Wednesday afternoon and advised residents to take precautions against heat stroke and fires, especially around midday. "It felt like I was walking in an oven in the strong sunshine. I had to wear sunscreen and use an umbrella, and I tried my best to stay indoors," said Wang Jing, who was working in Beijing on Wednesday. High-temperature days - referring to those that climb above 35 C - have arrived 20 days earlier than usual in Beijing, the center's data showed. Temperatures in the capital were higher than the average for the same period in recent years, but they're short of the record - 38.1 C on May 19, 2010. Though May is only half over, Beijing has entered summer, as determined by consecutive warm days. Tianjin, along with Shijiazhuang, Hebei province, and Zhengzhou, Henan province, were already into the summer, as they are ahead of the average of the past years, the national weather center said. "These regions, from Inner Mongolia to northern areas like Hebei and Henan, have been hit by high temperatures because of the west-to-east movement of a continental warm air mass, which brings sunny days with little rainfall," said He Lifu, another chief forecaster of the national weather authority. He added that it's not rare to see such dry and hot weather in northern China, as it happens every three to five years. Along with the discomfort caused by the heat, the ongoing dry and hot weather is bad for wheat growth. Seed weight will be reduced, which will affect the yield in some parts of Hebei and Henan provinces, the National Meteorological Center said.

<u>Program Output</u>:

```
it's may, but beijing feeling summer heat.

"they're the first really high temperatures to hit
northern china this year," said fang chong, chief
forecaster at the national meteorological center.

tianjin, along with shijiazhuang, hebei province, and
zhengzhou, henan province, were already into the summer,
as they are ahead of the average of the past years, the
national weather center said.
```

<u>By repetition of Algorithm</u>:

```
"they're the first really high temperatures to hit
northern china this year," said fang chong, chief
forecaster at the national meteorological center.

temperatures in the capital were higher than the average
for the same period in recent years, but they're short of
the record - 38.1 c on may 19, 2010. though may is only
half over, beijing has entered summer, as determined by
consecutive warm days.

tianjin, along with shijiazhuang, hebei province, and
zhengzhou, henan province, were already into the summer,
as they are ahead of the average of the past years, the
national weather center said.
```

Program:

```
import io
import os
os.system('pip install nltk')
os.system('pip install sklearn')
import nltk


nltk.download('stopwords')
nltk.download('punkt')

file = input("input file name: ")
f = open(file,"r+")
document = f.read()


# #convert to lower case
document = document.lower()

# seperate sentences
from nltk import sent_tokenize
sentences = sent_tokenize(document)


# remove punctuation from each line
import string
sentences = [line.translate(str.maketrans('', '', string.pu
nctuation)) for line in sentences]


# split into words
from nltk.tokenize import word_tokenize
t_doc = [word_tokenize(line) for line in sentences]
```

```
# filter out stop words
# and stemming using PorterStemmer

from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

stop_words = set(stopwords.words('english'))
porter = PorterStemmer()

for line in t_doc:
  for word in line:
    if word in stop_words:
      t_doc[t_doc.index(line)].remove(word)
    else: # using PorterStemer
      t_doc[t_doc.index(line)][line.index(word)] = porter.s
tem(word)

t_doc

# tf-idf using sklearn
# creating stemmed sentences

t_doc_s = list()
for line in t_doc:
  t_doc_s.append(" ".join(line[:]))
# print(t_doc_s)
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vector = vectorizer.fit_transform(t_doc_s)

vector.shape

# k-means cluster
from sklearn.cluster import KMeans
import numpy as np
```

```
# asking for how many lines we need to summarize

n = int(input("how many output lines do you want for summar
ize: "))
kmeans = KMeans(n_clusters=n,max_iter=10000,n_init=1000).fi
t(vector)
kmeans.cluster_centers_.shape

# calculating cosine similarity

from sklearn.metrics.pairwise import cosine_similarity
values = cosine_similarity(kmeans.cluster_centers_,vector)

values.shape

# finding the nearest sentence for each cluster center with
 cosine similarity
line_no = list()
for i in values:
  line_no.append(list(i).index(i.max()))

# sorting to keep the flow of meaning
#print(values,line_no)
line_no.sort()
from nltk import sent_tokenize
s_sentences = sent_tokenize(document)
print("\nSummary: ")
for i in line_no:
  print(s_sentences[i])
```

<u>Comparing Result with a count-based extraction summarization</u>:

```
It's May, but Beijing feeling summer heat.By one measure,
season has changed;conditions could affect wheat crop
Beijing sweltered in summerlike heat on Wednesday, with
temperatures expected to remain high until Friday night.
```

CONCLUSIONS AND FUTURE WORK: By comparison we can clearly see that the output result of the clustering program is more effective and expressive than the count-based extraction summarization. Through the course of evaluation, it is observed that k-means is a primitive clustering algorithm and advanced clustering algorithms are to be applied for superior results. Also, the process of sentence extraction from each cluster must not be a random process but requires devising a specific algorithm.