08-clustering-for-text-similarity

October 21, 2021

1 Clustering for Text Similarity

[1]:
```
# Python Regular Expression (RegEx)
import re
# Operating System Module
import os
# numpy library
import numpy as np
# pandas library
import pandas as pd
# matplotlib library
import matplotlib.pyplot as plt
# if uising a Jupyter notebook, include:
%matplotlib inline
# Natural Language Toolkit
import nltk
# The BeautifulSoup Library for WEB Scraping
from bs4 import BeautifulSoup

import codecs
from sklearn import feature_extraction
```

1.1 Example 1 - Clustering Film Reviews

In the next example we will use a large, freely available database to present a simple clustering example.

References : *Brandon Rose, "Top 100 Films of all Time", see here for the original post and here for the full functional notebook*

1.1.1 Import Data

This work starts from a list of the top 100 films of all time from an IMDB user list called Top 100 Greatest Movies of All Time (The Ultimate List) by ChrisWalczyk55. IMDb (an acronym for Internet Movie Database) is an online database of information related to films, television programs, home videos, video games, and streaming content online – including cast, production crew and personal biographies, plot summaries, trivia, ratings, and fan and critical reviews. Originally a fan-operated website, the database is now owned and operated by IMDb.com, Inc., a subsidiary of Amazon.

To this list, the author adds two more file of synopses gathered from IMDB and Wikipedia. All the files in txt format can be downloaded from the github repository cited in the references box above.

The goal is to identify the latent structures within the synopses of the top 100 films of all time.

Altogether we will work with 4 text files:

- `title_list.txt` : the titles of the films in their rank order, this file is mostly used for labeling purposes;
- `synopses_list_imdb`: the synopses of the films matched to the 'titles' order (from imdb)
- `synopses_list_wiki`: the synopses of the films matched to the 'titles' order (from wikipedia)
- `genres_list_wiki` : the genre of the films matched to the 'titles' order (from wikipedia)

We are going to use the library *BeautifulSoup* to clean text from unwanted html tag and stuff like that.

[2]:
```
#import three lists: titles, imdb and wikipedia synopses
path = './corpus/'
```

[3]:
```
#
# title-list as the name suggests contains a simple list of the
# 100 top films of all time according to IMDB
#
titles = open(path + 'title_list.txt').read().split('\n')
#ensures that only the first 100 are read in
titles = titles[:100]
```

[4]:
```
print(titles[0:10])
```

```
['The Godfather', 'The Shawshank Redemption', "Schindler's List", 'Raging Bull',
'Casablanca', "One Flew Over the Cuckoo's Nest", 'Gone with the Wind', 'Citizen
Kane', 'The Wizard of Oz', 'Titanic']
```

[5]:
```
#
# read wikipedia synopses
#
synopses_wiki = open(path + 'synopses_list_wiki.txt', encoding="utf8").read().
 ↪split('\n BREAKS HERE')
synopses_wiki = synopses_wiki[:100]
#
# strips html formatting and converts to unicode
#
synopses_clean_wiki = []
for text in synopses_wiki:
    text = BeautifulSoup(text, 'html.parser').getText()
    synopses_clean_wiki.append(text)
synopses_wiki = synopses_clean_wiki
```

[6]:
```
print(synopses_wiki[0][:200]) #first 200 characters in first synopses (for 'The␣
↪Godfather')
```

```
 Plot  [edit]  [  [  edit  edit  ]  ]
  On the day of his only daughter's wedding, Vito Corleone hears requests in his
role as the Godfather, the Don of a New York crime family. Vito's youngest son,
```

[7]:
```
#
# read imdb synopses
#
synopses_imdb = open(path + 'synopses_list_imdb.txt').read().split('\n BREAKS␣
↪HERE')
synopses_imdb = synopses_imdb[:100]

synopses_clean_imdb = []
#
# strips html formatting and converts to unicode
#
for text in synopses_imdb:
    text = BeautifulSoup(text, 'html.parser').getText()
    synopses_clean_imdb.append(text)

synopses_imdb = synopses_clean_imdb
```

[8]:
```
genres = open(path + 'genres_list.txt').read().split('\n')
genres = genres[:100]
```

[9]:
```
print(str(len(titles)) + ' titles')
print(str(len(synopses_wiki)) + ' synopses wiki')
print(str(len(synopses_imdb)) + ' synopses imdb')
print(str(len(genres)) + ' genres')
```

```
100 titles
100 synopses wiki
100 synopses imdb
100 genres
```

We merge synopses from wikipedia and imdb in order to have more words to work with

[10]:
```
synopses = []

for i in range(len(synopses_wiki)):
    item = synopses_wiki[i] + synopses_imdb[i]
    synopses.append(item)
```

[11]:
```
# generates index for each item in the corpora (in this case it's
# just rank)
ranks = []
```

```
for i in range(0,len(titles)):
    ranks.append(i)
```

1.1.2 Tokenization and Stemming

Below we define two functions:

- *tokenize_and_stem*: tokenizes (splits the synopsis into a list of its respective words (or tokens) and also stems each token
- *tokenize_only*: tokenizes the synopsis only

[12]:
```
from nltk.stem import WordNetLemmatizer, PorterStemmer, SnowballStemmer
from nltk.corpus import stopwords
```

[17]:
```
stop_words = stopwords.words('english')
stemmer    = SnowballStemmer("english")

def tokenize_only(text):
    tokens = [word.lower() for word in nltk.word_tokenize(text)]
    filtered_tokens = []
    # filter out any tokens not containing letters (e.g., numeric tokens, raw␣
 ↪punctuation)
    for token in tokens:
        if token not in stop_words:
            if re.search('[a-zA-Z]', token):
                filtered_tokens.append(token)
    return filtered_tokens

def tokenize_and_stem(text):
    filtered_tokens = tokenize_only(text)
    stems = [stemmer.stem(t) for t in filtered_tokens]
    return stems
```

[18]:
```
tokenize_only("this is, a sentence. And this is another sentence! This is A␣
 ↪SENTENCE which contains NUMBERS: 123456")
```

[18]:
```
['sentence', 'another', 'sentence', 'sentence', 'contains', 'numbers']
```

[19]:
```
tokenize_and_stem("tHis is, a sentence. And this is another sentence! This is A␣
 ↪SENTENCE which contains NUMBERS: 123456")
```

[19]:
```
['sentenc', 'anoth', 'sentenc', 'sentenc', 'contain', 'number']
```

Below we use stemming/tokenizing and tokenizing functions to iterate over the list of synopses to create two vocabularies: one stemmed and one only tokenized.

[20]:
```
totalvocab_stemmed   = []
totalvocab_tokenized = []
```

```python
for text in synopses:
    allwords_stemmed = tokenize_and_stem(text)
    totalvocab_stemmed.extend(allwords_stemmed)

    allwords_tokenized = tokenize_only(text)
    totalvocab_tokenized.extend(allwords_tokenized)
```

Using these two lists, we create a pandas DataFrame with the stemmed vocabulary as the index and the tokenized words as the column. The benefit of this is it provides an efficient way to look up a stem and return a full token. The downside here is that stems to tokens are one to many: the stem 'run' could be associated with 'ran', 'runs', 'running', etc. For our purposes in this very simple example this is not a problem.

[22]:
```python
vocab_frame = pd.DataFrame({'words': totalvocab_tokenized}, index =␣
↪totalvocab_stemmed)
vocab_frame.head()
```

[22]:

	words
plot	plot
edit	edit
edit	edit
edit	edit
day	day

1.1.3 Feature Extraction

We are going to use the cosine similarity as a metric of documents similarity that we can load from the metrics module of sklear:

[23]:
```python
from sklearn.metrics.pairwise import cosine_similarity
```

To vectorize text we shall use the usual Tf-idf transformation (help page for the TfidfVectorizer function):

[24]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(  max_df       = 0.8
                                   , min_df       = 0.2
                                   , max_features = 200000
                                   , use_idf      = True
                                   , tokenizer    = tokenize_and_stem
                                   , ngram_range  = (1,3))

# In the next line we have an example of a useful magic function: %time, which␣
↪will automatically
# determine the execution time of the single-line Python statement that follows␣
↪it. More about
# the use of magic commands in jupyter notebook here:
```

```
# https://towardsdatascience.com/useful-ipython-magic-commands-245e6c024711

%time tfidf_matrix = tfidf_vectorizer.fit_transform(synopses)

print(tfidf_matrix.shape)
```

```
Wall time: 8.58 s
(100, 611)
```

[25]:
```
dist = 1 - cosine_similarity(tfidf_matrix)
```

[26]:
```
terms = tfidf_vectorizer.get_feature_names()
```

[27]:
```
terms[:10]
```

[27]:
```
["'d",
 "'s death",
 "'s father",
 "'s friend",
 "'s hous",
 "'s mother",
 'abandon',
 'abl',
 'accept',
 'accid']
```

1.1.4 K-means clustering

Using the tf-idf matrix, we can try to use the k-means clustering algorithms to to find the hidden structure within the synopses. K-means initializes with a pre-determined number of clusters (here we chose 5). Each observation is assigned to a cluster (cluster assignment) so as to minimize the within cluster sum of squares. Next, the mean of the clustered observations is calculated and used as the new cluster centroid. Then, observations are reassigned to clusters and centroids recalculated in an iterative process until the algorithm reaches convergence.

It took several runs for the algorithm to converge a global optimum as k-means is susceptible to reaching local optima.

[48]:
```
from sklearn.cluster import KMeans

num_clusters = 5

km = KMeans(n_clusters = num_clusters, init='k-means++', max_iter=100000,␣
 ↪n_init=200)

%time km.fit(tfidf_matrix)

clusters = km.labels_.tolist()
```

```
Wall time: 1.81 s
```

[49]:
```python
order_centroids = km.cluster_centers_
```

[50]:
```python
order_centroids
```

[50]:
```
array([[0.01133982, 0.01143624, 0.01625907, …, 0.05884046, 0.03594936,
         0.00997995],
        [0.01208379, 0.01155751, 0.00693076, …, 0.01855676, 0.01847054,
         0.00893591],
        [0.00655417, 0.0158397 , 0.0042834 , …, 0.00842836, 0.02645245,
         0.00366542],
        [0.00521074, 0.01129072, 0.00633069, …, 0.00395351, 0.02246614,
         0.00213132],
        [0.00086107, 0.00571096, 0.03174532, …, 0.02991187, 0.00823433,
         0.00485025]])
```

[51]:
```python
order_centroids = order_centroids.argsort()
```

[52]:
```python
order_centroids
```

[52]:
```
array([[496, 419,  40, …, 368, 216, 214],
       [291, 422, 579, …, 303,  98, 408],
       [145, 102, 474, …, 488, 491, 579],
       [389, 137, 391, …, 303,  46, 509],
       [305, 209,  65, …, 336, 299, 242]], dtype=int64)
```

[53]:
```python
order_centroids[:, ::-1]
```

[53]:
```
array([[214, 216, 368, …,  40, 419, 496],
       [408,  98, 303, …, 579, 422, 291],
       [579, 491, 488, …, 474, 102, 145],
       [509,  46, 303, …, 391, 137, 389],
       [242, 299, 336, …,  65, 209, 305]], dtype=int64)
```

[54]:
```python
import pandas as pd

films = { 'title': titles, 'rank': ranks, 'synopsis': synopses, 'cluster':␣
↪clusters, 'genre': genres }

frame = pd.DataFrame(films, index = [clusters] , columns = ['rank', 'title',␣
↪'cluster', 'genre'])
frame.head()
```

[54]:
```
   rank                     title  cluster  \
0     0             The Godfather        0
1     1  The Shawshank Redemption        1
```

```
3     2          Schindler's List        3
0     3               Raging Bull        0
1     4                Casablanca        1

                                      genre
0                     [u' Crime', u' Drama']
1                     [u' Crime', u' Drama']
3  [u' Biography', u' Drama', u' History']
0    [u' Biography', u' Drama', u' Sport']
1        [u' Drama', u' Romance', u' War']
```

[55]:
```
frame['cluster'].value_counts()
```

[55]:
```
1    39
0    30
3    16
2     9
4     6
Name: cluster, dtype: int64
```

[56]:
```
print("Top terms per cluster:")
print()
cluster_names = {}
for i in range(num_clusters):
    print("Cluster %d words:" % i, end='')
    cluster_name = ''
    for ind in order_centroids[i, :6]:
        cluster_name = cluster_name + ' ' + (vocab_frame.loc[terms[ind].split('␣
↪')].values.tolist()[0][0].encode('utf-8', 'ignore')).decode('utf-8')
        print(' %s' % vocab_frame.loc[terms[ind].split(' ')].values.
↪tolist()[0][0].encode('utf-8', 'ignore'), end=',')
    cluster_names[i] = cluster_name.strip()
    print()
    print()

    print("Cluster %d titles:" % i, end='')
    for title in frame.loc[i]['title'].values.tolist():
        print(' %s,' % title, end='')
    print()
    print()
```

```
Top terms per cluster:

Cluster 0 words: b'shouted', b'proceeds', b'apparently', b'storms', b'next',
b'blow',

Cluster 0 titles: The Godfather, Raging Bull, Gone with the Wind, Citizen Kane,
```

```
The Godfather: Part II, On the Waterfront, The Sound of Music, Amadeus, A
Streetcar Named Desire, To Kill a Mockingbird, The Best Years of Our Lives, My
Fair Lady, Ben-Hur, Doctor Zhivago, The Good, the Bad and the Ugly, The
Apartment, High Noon, The Pianist, Goodfellas, The Exorcist, Midnight Cowboy,
Mr. Smith Goes to Washington, Rain Man, Annie Hall, Out of Africa, Terms of
Endearment, Tootsie, Giant, The Grapes of Wrath, Yankee Doodle Dandy,

Cluster 1 words: b'intent', b'protection', b'water', b'dangerous', b'support',
b'soldiers',

Cluster 1 titles: The Shawshank Redemption, Casablanca, One Flew Over the
Cuckoo's Nest, Psycho, Sunset Blvd., Vertigo, Forrest Gump, West Side Story,
E.T. the Extra-Terrestrial, The Silence of the Lambs, Singin' in the Rain, Some
Like It Hot, 12 Angry Men, Gandhi, Unforgiven, Rocky, An American in Paris,
Butch Cassidy and the Sundance Kid, The French Connection, City Lights, It
Happened One Night, Good Will Hunting, Fargo, Shane, Close Encounters of the
Third Kind, Network, Nashville, The Graduate, American Graffiti, Pulp Fiction,
The Maltese Falcon, A Clockwork Orange, Taxi Driver, Wuthering Heights, Double
Indemnity, Rebel Without a Cause, Rear Window, The Third Man, North by
Northwest,

Cluster 2 words: b'dance', b'case', b'school', b'army', b'ii', b'heard',

Cluster 2 titles: The Wizard of Oz, Titanic, Star Wars, 2001: A Space Odyssey,
Chinatown, Jaws, The Treasure of the Sierra Madre, The African Queen, Mutiny on
the Bounty,

Cluster 3 words: b'owned', b'couple', b'parents', b'boss', b'date', b'new',

Cluster 3 titles: Schindler's List, Lawrence of Arabia, The Bridge on the River
Kwai, Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb,
Apocalypse Now, The Lord of the Rings: The Return of the King, Gladiator, From
Here to Eternity, Saving Private Ryan, Raiders of the Lost Ark, Patton,
Braveheart, Platoon, Dances with Wolves, The Deer Hunter, All Quiet on the
Western Front,

Cluster 4 words: b'knocks', b'eye', b'barely', b'figure', b'flee', b'fly',

Cluster 4 titles: It's a Wonderful Life, The Philadelphia Story, The King's
Speech, A Place in the Sun, The Green Mile, Stagecoach,
```

[57]:
```python
cluster_names
```

[57]:
```
{0: 'shouted proceeds apparently storms next blow',
 1: 'intent protection water dangerous support soldiers',
 2: 'dance case school army ii heard',
```

```
3: 'owned couple parents boss date new',
4: 'knocks eye barely figure flee fly'}
```

1.1.5 Dimensionality Reduction and Manifold Learning

Dimensionality reduction, or dimension reduction, is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension. Working in high-dimensional spaces can be undesirable for many reasons; raw data are often sparse as a consequence of the curse of dimensionality, and analyzing the data is usually computationally intractable. Dimensionality Reduction is a very complex problem in machine learning and we cannot dive into it in this simple introduction.

Apart from simplifying data, dimensionality reduction has other uses as well. Let's consider the **visualization** process for a minute here. If the data lies in a 100-dimensional space, we cannot get an intuitive feel for what the data looks like. We can barely manage to imagine the 4th dimension, let alone visualizing the 100th! However, if a meaningful two or three dimensional representation of the data can be found, then it is possible to visualize it. Though this may seem like a trivial point, many statistical and machine learning algorithms have very poor optimality guarantees, so the ability to actually see the data and the output of an algorithm is of great practical interest.

Multidimensional Scaling

[58]:
```
from sklearn.manifold import MDS
#
# chosen parameters:
#
# n_components = 2              - two components as we're plotting points in a␣
↪two-dimensional plane
# dissimilarity="precomputed" - "precomputed" because we provide a distance␣
↪matrix
# we will also specify `random_state` so the plot is reproducible.

mds = MDS(  n_components = 2
          , dissimilarity="precomputed"
          , random_state=1)
#
# dist is the distance matrix we have computed before with cosine similarity
#
pos = mds.fit_transform(dist)  # shape (n_components, n_samples)
xs, ys = pos[:, 0], pos[:, 1]
```

1.1.6 Visualizing document clusters

[59]:
```
#set up colors per clusters using a dict
cluster_colors = {0: '#1b9e77', 1: '#d95f02', 2: '#7570b3', 3: '#e7298a', 4:␣
↪'#66a61e'}
```

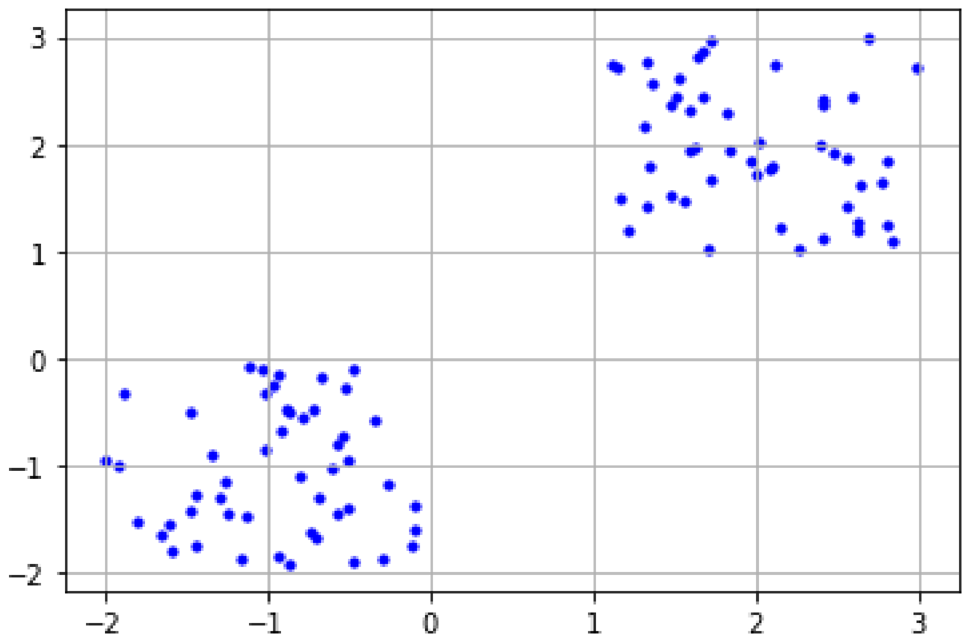
[60]:

```python
import matplotlib.pyplot as plt

#create data frame that has the result of the MDS plus the cluster numbers and␣
↪titles
df = pd.DataFrame(dict(x=xs, y=ys, label=clusters, title=titles))

#group by cluster
groups = df.groupby('label')



# set up plot
fig, ax = plt.subplots(figsize=(17, 9)) # set size
ax.margins(0.05) # Optional, just adds 5% padding to the autoscaling

#iterate through groups to layer the plot
#note that I use the cluster_name and cluster_color dicts with the 'name'␣
↪lookup to return the appropriate color/label
for name, group in groups:
    ax.plot(group.x, group.y, marker='o', linestyle='', ms=12,␣
↪label=cluster_names[name], color=cluster_colors[name], mec='none')
    ax.set_aspect('auto')
    ax.tick_params(\
        axis= 'x',          # changes apply to the x-axis
        which='both',      # both major and minor ticks are affected
        bottom='off',      # ticks along the bottom edge are off
        top='off',         # ticks along the top edge are off
        labelbottom='off')
    ax.tick_params(\
        axis= 'y',         # changes apply to the y-axis
        which='both',      # both major and minor ticks are affected
        left='off',      # ticks along the bottom edge are off
        top='off',         # ticks along the top edge are off
        labelleft='off')

ax.legend(numpoints=1)  #show legend with only 1 point

#add label in x,y position with the label as the film title
for i in range(len(df)):
    ax.text(df.loc[i]['x'], df.loc[i]['y'], df.loc[i]['title'], size=8)




plt.show() #show the plot

#uncomment the below to save the plot if need be
#plt.savefig('clusters_small_noaxes.png', dpi=200)
```

1.1.7 Hierarchical document clustering

Let's try another clustering algorithm for example the Ward clustering algorithm because it offers hierarchical clustering. Ward clustering is an agglomerative clustering method, meaning that at each stage, the pair of clusters with minimum between-cluster distance are merged. I used the precomputed cosine distance matrix (dist) to calclate a linkage_matrix, which I then plot as a dendrogram.

[61]:
```
from scipy.cluster.hierarchy import ward, dendrogram

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="right", labels=titles);

plt.tick_params(\
    axis= 'x',          # changes apply to the x-axis
    which='both',      # both major and minor ticks are affected
    bottom='off',      # ticks along the bottom edge are off
    top='off',         # ticks along the top edge are off
    labelbottom='off')

plt.tight_layout() #show plot with tight layout

#uncomment below to save figure
plt.savefig('ward_clusters.png', dpi=200) #save figure as ward_clusters
```



1.2 Example 2 - Clustering Wikipedia

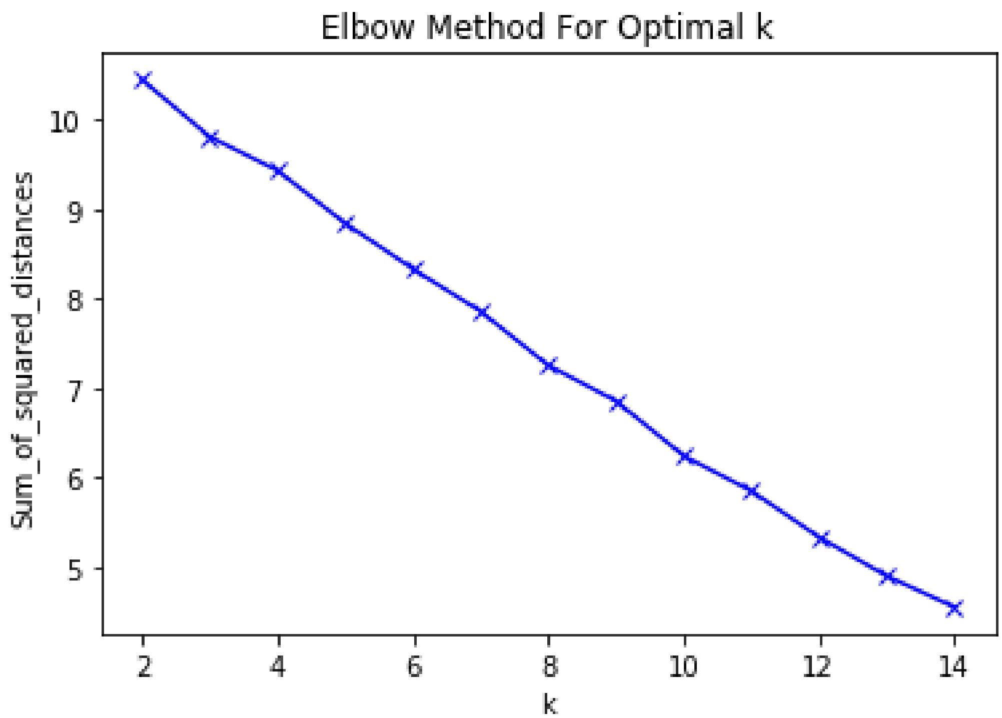
[62]:
```
import pandas as pd
import wikipedia

articles=["Ingmar  Bergman"
,"Sarah  Bernhardt"
,"Charlie  Chaplin"
,"Marlene  Dietrich"
,"Walt  Disney"
,"Sergei  Eisenstein"
,"Federico  Fellini"
,"Alfred  Hitchcock"
,"Stanley  Kubrick"
,"Akira  Kurosawa"
,"Marie  Curie"
,"Charles  Darwin"
,"Thomas  Edison"
,"Albert  Einstein"
,"Leonhard  Euler"
,"Michael  Faraday"
,"Enrico  Fermi"
,"Carl Friedrich  Gauss"
,"David  Hilbert"
,"James Clerk  Maxwell"
```

```
,"Sir Isaac  Newton"
,"Max  Planck"
,"Ernest  Rutherford"
,"Erwin  Schrödinger"
,"Nikola  Tesla"
,"Alan  Turing"]
wiki_lst=[]
title=[]
for article in articles:
   print("loading content: ",article)
   wiki_lst.append(wikipedia.page(article).content)
   title.append(article)
print("examine content")
```

```
loading content:  Ingmar  Bergman
loading content:  Sarah  Bernhardt
loading content:  Charlie  Chaplin
loading content:  Marlene  Dietrich
loading content:  Walt  Disney
loading content:  Sergei  Eisenstein
loading content:  Federico  Fellini
loading content:  Alfred  Hitchcock
loading content:  Stanley  Kubrick
loading content:  Akira  Kurosawa
loading content:  Marie  Curie
loading content:  Charles  Darwin
loading content:  Thomas  Edison
loading content:  Albert  Einstein
loading content:  Leonhard  Euler
loading content:  Michael  Faraday
loading content:  Enrico  Fermi
loading content:  Carl Friedrich  Gauss
loading content:  David  Hilbert
loading content:  James Clerk  Maxwell
loading content:  Sir Isaac  Newton
loading content:  Max  Planck
loading content:  Ernest  Rutherford
loading content:  Erwin  Schrödinger
loading content:  Nikola  Tesla
loading content:  Alan  Turing
examine content
```

[63]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words={'english'})
X = vectorizer.fit_transform(wiki_lst)
```

[64]:
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
Sum_of_squared_distances = []
K = range(2,15)
for k in K:
   km = KMeans(n_clusters=k, max_iter=1000, n_init=10)
   km = km.fit(X)
   Sum_of_squared_distances.append(km.inertia_)
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



[65]:
```
true_k = 3
model = KMeans(n_clusters=true_k, init='k-means++', max_iter=100000,␣
 ↪n_init=200, random_state=1)
model.fit(X)
labels=model.labels_
wiki_cl=pd.DataFrame(list(zip(title,labels)),columns=['title','cluster'])
print(wiki_cl.sort_values(by=['cluster']))
```

```
                   title  cluster
0         Ingmar  Bergman        0
```

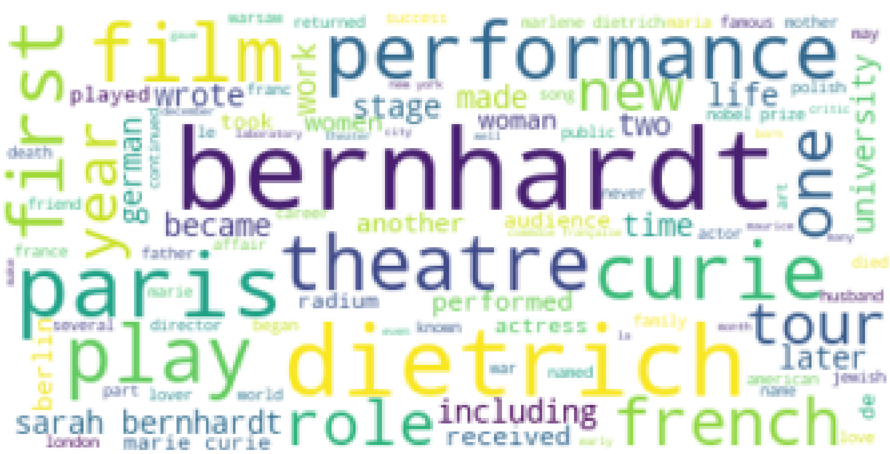
```
9          Akira  Kurosawa        0
7        Alfred  Hitchcock        0
6        Federico  Fellini        0
5       Sergei  Eisenstein        0
8         Stanley  Kubrick        0
2         Charlie  Chaplin        0
4             Walt  Disney        0
3        Marlene  Dietrich        1
1         Sarah  Bernhardt        1
10            Marie  Curie        1
19   James Clerk  Maxwell        2
23      Erwin  Schrödinger        2
22      Ernest  Rutherford        2
21             Max  Planck        2
20       Sir Isaac  Newton        2
18          David  Hilbert        2
12          Thomas  Edison        2
16           Enrico  Fermi        2
15        Michael  Faraday        2
14          Leonhard  Euler        2
13        Albert  Einstein        2
24           Nikola  Tesla        2
11         Charles  Darwin        2
17  Carl Friedrich  Gauss        2
25            Alan  Turing        2
```

[154]:
```python
from wordcloud import WordCloud
result={'cluster':labels,'wiki':wiki_lst}
result=pd.DataFrame(result)
for k in range(0,true_k):
   s=result[result.cluster==k]
   text=s['wiki'].str.cat(sep=' ')
   text=text.lower()
   text=' '.join([word for word in text.split()])
   wordcloud = WordCloud(max_font_size=50, max_words=100,␣
 ↪background_color="white").generate(text)
   print('Cluster: {}'.format(k))
   print('Titles')
   titles=wiki_cl[wiki_cl.cluster==k]['title']
   print(titles.to_string(index=False))
   plt.figure()
   plt.imshow(wordcloud, interpolation="bilinear")
   plt.axis("off")
   plt.show()
```

```
Cluster: 0
Titles
    Ingmar  Bergman
```

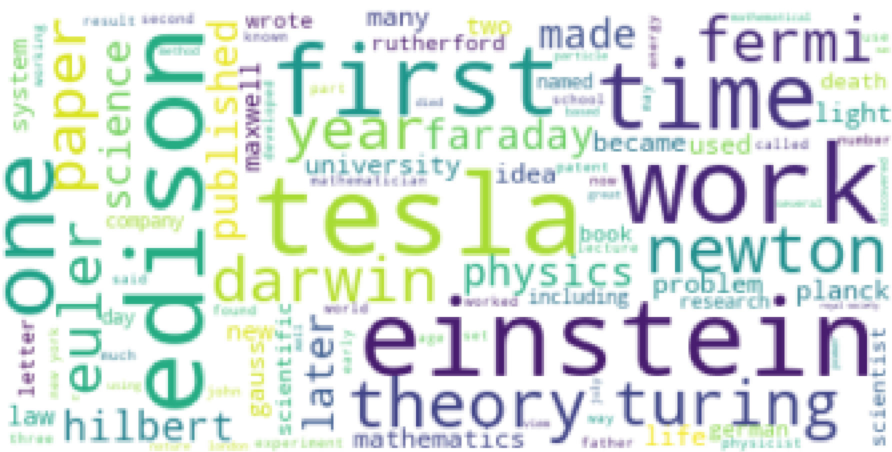
```
   Charlie  Chaplin
       Walt  Disney
 Sergei  Eisenstein
  Federico  Fellini
  Alfred  Hitchcock
   Stanley  Kubrick
     Akira  Kurosawa
```


```
Cluster: 1
Titles
  Sarah  Bernhardt
 Marlene  Dietrich
      Marie  Curie
```




```
Cluster: 2
```

```
Titles
       Charles  Darwin
        Thomas  Edison
      Albert  Einstein
       Leonhard  Euler
      Michael  Faraday
         Enrico  Fermi
Carl Friedrich  Gauss
        David  Hilbert
  James Clerk  Maxwell
     Sir Isaac  Newton
           Max  Planck
    Ernest  Rutherford
    Erwin  Schrödinger
         Nikola  Tesla
           Alan  Turing
```



1.3 References and Credits

Rose B., "*Top 100 Films of all Time*", see here for the original post and here for the full functional notebook*