In [1]:
```
# import the libraries
import numpy as np
import pandas as pd
from collections import Counter
import os
import glob
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/achyutganti/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[1]:
```
True
```

In [2]:
```
# String used to pre-process the dataset
sp = '!@#$%^&*(){}[]+=|:;,./?><\'\"-1234567890'
stop_words = set(stopwords.words("english"))
ps = PorterStemmer()
```

In [3]:
```
# This function calculates the feature vector, pre-processing like stemming,
# Also removes stop words and punctuations.
def feature_vector(path_func):
    len_func = 0
    func_vec_allfile = dict()
    for filename in glob.glob(os.path.join(path_func,'*.txt')):
        len_func+=1
        f = open(filename, 'r', encoding = 'utf8').read()
        for i in f:
            if i in sp:
                f = f.replace(i,'')
        f=f.lower()
        f=f.split()
        for words in f:
            if words not in stop_words:
                words_stem = ps.stem(words)
                if words_stem in func_vec_allfile:
                    func_vec_allfile[words_stem]+=1
                else:
                    func_vec_allfile[words_stem]=1
    return(func_vec_allfile,len_func)
```

In [4]:
```
# calculating the feature vector for entire document
f_t_allfile= dict()
path_all = '/Users/achyutganti/Downloads/aclImdb/train/allfiles'
all_file = feature_vector(path_all)
f_t_allfile = all_file[0]
len_allfile = all_file[1]
```

In [5]:
```
# Doing Dataset Statistics

# Sum of full dataset

sum(f_t_allfile.values())
```

Out[5]:
```
3075860
```

In [6]:
```
f_t_allfile
```

In [7]:
```
sum(f_t_allfile.values())
```

Out[7]:
```
3075860
```

In [8]:
```
all_file
```

In [ ]:
```
```

In [9]:
```
# calculating the feature vector for positive class
pos_vec_allfile = dict()
path_pos = '/Users/achyutganti/Downloads/aclImdb/train/pos'
pos_file = feature_vector(path_pos)
pos_vec_allfile = pos_file[0]
len_pos = pos_file[1]
```

In [10]:
```
sum(pos_vec_allfile.values())
```

Out[10]:
```
1562331
```

In [11]:
```
# calculating the feature vector for negative class
neg_vec_allfile = dict()
path_neg = '/Users/achyutganti/Downloads/aclImdb/train/neg'
neg_file = feature_vector(path_neg)
neg_vec_allfile = neg_file[0]
len_neg = neg_file[1]
```

In [12]:
```
sum(neg_vec_allfile.values())
```

Out[12]:
```
1513529
```

In [13]:
```
# calculates length of feature vector, total probabilities for both classes
feature_vec_len = len(f_t_allfile)
pos_prob = len_pos/len_allfile
neg_prob = len_neg/len_allfile
```

In [14]:
```
# function to calculate conditional probability of each word
def calculate_probability(class_prob):
    prob_both = dict()
    for i in f_t_allfile.keys():
        if i in class_prob:
            prob_both[i] = (class_prob[i]+1)/float(feature_vec_len+ sum(class_prob.values()))
        else:
            prob_both[i] = (1/float(feature_vec_len+ sum(class_prob.values())))
    return (prob_both)
```

In [15]:
```
# calling the function to calculate the probability in each class
prob_posit = dict()
prob_posit = calculate_probability(pos_vec_allfile)
prob_negat = dict()
prob_negat = calculate_probability(neg_vec_allfile)
```

In [16]:
```
#prob_posit
```

In [17]:
```
#prob_negat
```

In [18]:
```
prob_diff = {key: prob_posit[key] - prob_negat.get(key, 0) for key in prob_posit}
```

In [19]:
```
#prob_diff
```

In [20]:
```
#sorted(prob_diff.items(), key=lambda x:x[1], reverse = True)
```

In [21]:
```
#sorted(prob_diff.items(), key=lambda x:x[1])
```

Topic Modeling

In [22]:
```
#pip install wordcloud
```

In [23]:
```
#!pip install pyLDAvis
```

In [37]:
```
import pandas as pd
import numpy as np
import re
#from wordcloud import WordCloud
import gensim
from gensim.utils import simple_preprocess
from nltk.corpus import stopwords
import gensim.corpora as corpora
from pprint import pprint
import pyLDAvis.gensim_models
import pickle
import pyLDAvis
```

In [ ]:
```
'/Users/achyutganti/Downloads/aclImdb/train/allfiles'
```

In [52]:
```
import glob
import os

file_list = glob.glob(os.path.join(os.getcwd(), "/Users/achyutganti/Downloads/aclImdb/train/neg", "*.txt"))

corpus = []

for file_path in file_list:
    with open(file_path) as f_input:
        corpus.append(f_input.read())

print(corpus[0])
```

```
Working with one of the best Shakespeare sources, this film manages to be creditable to it's source, whils
t still appealing to a wider audience.<br /><br />Branagh steals the film from under Fishburne's nose, and
there's a talented cast on good form.
```

In [53]:
```
len(corpus)
```

Out[53]:
```
12500
```

In [54]:
```
import pandas as pd
df = pd.DataFrame(corpus, columns= {"Message"})
```

In [24]:
```
#df
```

In [56]:
```
df["Message_processed"] = df["Message"].map(lambda x: re.sub('[,\.!?]', '', str(x)))
```

```
<>:1: DeprecationWarning: invalid escape sequence \.
<>:1: DeprecationWarning: invalid escape sequence \.
/var/folders/m8/vl5k0yvx2rg3752txl5qj5x80000gn/T/ipykernel_98117/3870544856.py:1: DeprecationWarning: inva
lid escape sequence \.
  df["Message_processed"] = df["Message"].map(lambda x: re.sub('[,\.!?]', '', str(x)))
```

In [57]:
```
df['Message_processed'] = df['Message_processed'].map(lambda x: x.lower())
```

In [58]:
```
df['Message_processed'].head()
```

Out[58]:
```
0    working with one of the best shakespeare sourc...
1    welltremors i the original started off in 1990...
2    ouch this one was a bit painful to sit through...
3    i've seen some crappy movies in my life but th...
4    "carriers" follows the exploits of two guys an...
Name: Message_processed, dtype: object
```

In [59]:
```
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use','br','film','movie','like','one'])
def sent_to_words(sentences):
    for sentence in sentences:
        # deacc=True removes punctuations
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))

def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc))
             if word not in stop_words] for doc in texts]
data = df.Message_processed.values.tolist()
data_words = list(sent_to_words(data))# remove stop words
data_words = remove_stopwords(data_words)
```

In [60]:
```
id2word = corpora.Dictionary(data_words)
```

In [61]:
```
id2word
```

Out[61]:
```
<gensim.corpora.dictionary.Dictionary at 0x29246b700>
```

In [62]:
```
texts = data_words# Term Document Frequency
corpus = [id2word.doc2bow(text) for text in texts]# View
print(corpus[:1][0][:30])
```

```
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1)
, (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (18, 1), (19, 1)]
```

In [63]:
```
num_topics = 10# Build LDA model
lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                       id2word=id2word,
                                       num_topics=num_topics)# Print the Keyword in the 10 topics
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

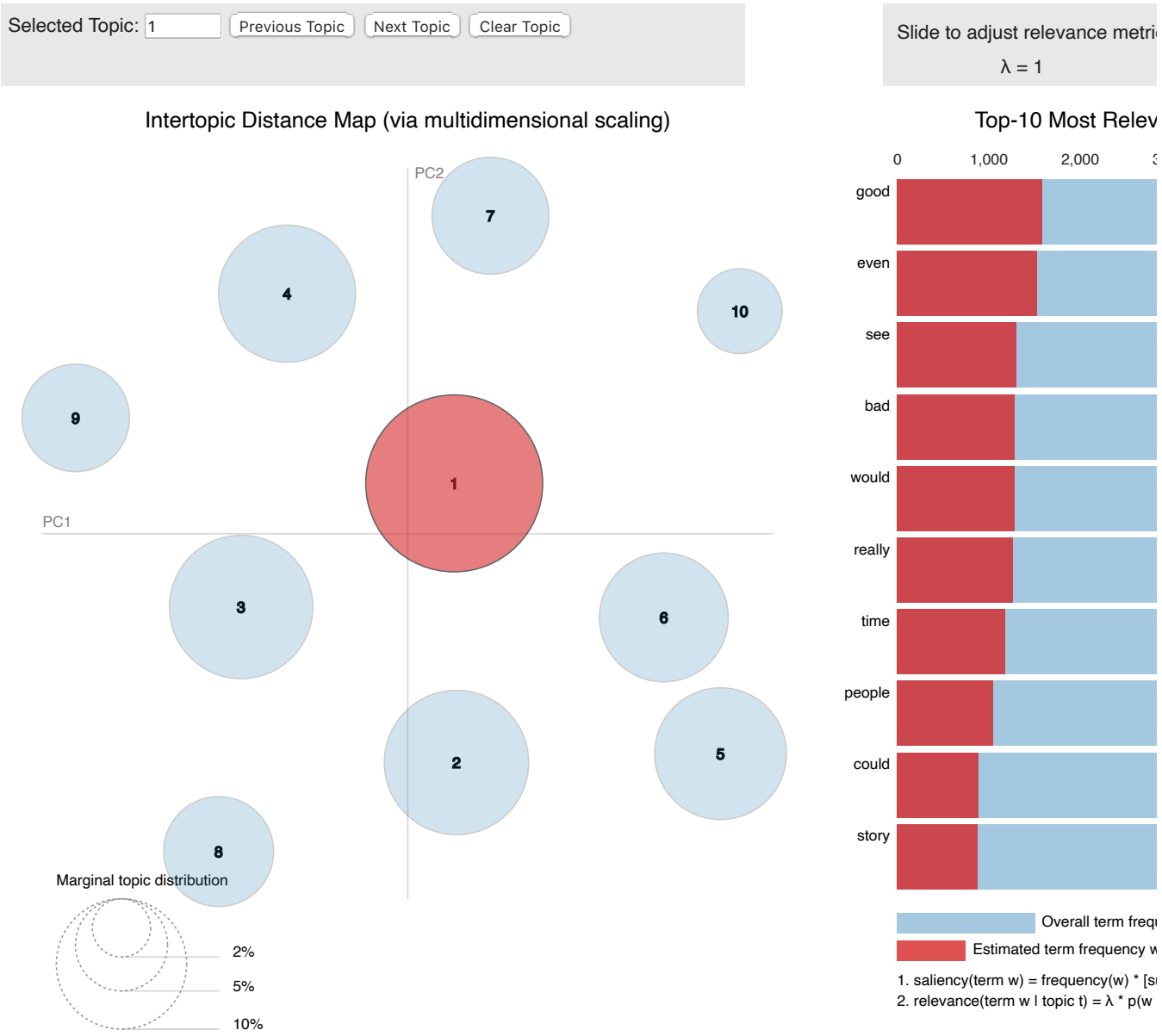
```
[(0,
  '0.006*"bad" + 0.006*"really" + 0.006*"good" + 0.005*"even" + 0.005*"story" '
  '+ 0.005*"get" + 0.005*"would" + 0.004*"acting" + 0.004*"see" + '
  '0.004*"much"'),
 (1,
  '0.005*"would" + 0.005*"see" + 0.004*"even" + 0.004*"good" + 0.004*"time" + '
  '0.004*"story" + 0.004*"bad" + 0.004*"movies" + 0.003*"first" + '
  '0.003*"made"'),
 (2,
  '0.006*"bad" + 0.006*"time" + 0.006*"would" + 0.005*"good" + 0.005*"even" + '
  '0.004*"movies" + 0.004*"ever" + 0.004*"people" + 0.004*"made" + '
  '0.004*"better"'),
 (3,
  '0.005*"good" + 0.005*"bad" + 0.005*"first" + 0.005*"would" + 0.004*"even" + '
  '0.004*"much" + 0.004*"made" + 0.004*"could" + 0.004*"people" + 0.004*"see"'),
 (4,
  '0.006*"would" + 0.006*"really" + 0.005*"bad" + 0.005*"time" + 0.005*"get" + '
  '0.005*"even" + 0.005*"good" + 0.004*"way" + 0.003*"plot" + '
  '0.003*"character"'),
 (5,
  '0.006*"good" + 0.006*"bad" + 0.006*"really" + 0.005*"even" + 0.004*"time" + '
  '0.004*"much" + 0.004*"story" + 0.004*"get" + 0.003*"acting" + 0.003*"make"'),
 (6,
  '0.006*"good" + 0.006*"even" + 0.005*"see" + 0.005*"bad" + 0.005*"would" + '
  '0.005*"really" + 0.005*"time" + 0.004*"people" + 0.003*"could" + '
  '0.003*"story"'),
 (7,
  '0.006*"would" + 0.005*"really" + 0.005*"much" + 0.004*"see" + 0.004*"get" + '
  '0.004*"could" + 0.004*"time" + 0.004*"story" + 0.003*"character" + '
  '0.003*"make"'),
 (8,
  '0.007*"even" + 0.006*"would" + 0.005*"time" + 0.005*"good" + 0.005*"make" + '
  '0.005*"made" + 0.004*"bad" + 0.004*"well" + 0.004*"people" + 0.004*"plot"'),
 (9,
  '0.006*"even" + 0.005*"bad" + 0.005*"good" + 0.004*"really" + 0.004*"could" '
  '+ 0.004*"time" + 0.004*"make" + 0.003*"much" + 0.003*"get" + 0.003*"story"')]
```

In [64]:
```
pyLDAvis.enable_notebook()
#LDAvis_data_filepath = os.path.join('./results/ldavis_prepared_'+str(num_topics))
```

In [66]:
```
vis = pyLDAvis.gensim_models.prepare(lda_model, corpus, id2word, mds="mmds", R=10)
vis
```

```
/Users/achyutganti/opt/anaconda3/lib/python3.9/site-packages/pyLDAvis/_prepare.py:243: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.
  default_term_info = default_term_info.sort_values(
```

Out[66]:


Selected Topic: 1
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metri (2)
λ = 1
Intertopic Distance Map (via multidimensional scaling)
PC2
PC1
7
4
10
9
1
3
6
2
5
8
Marginal topic distribution
2%
5%
10%
Top-10 Most Relev
0
1,000
2,000
good
even
see
bad
would
really
time
people
could
story
Overall term frequ
Estimated term frequency w
1. saliency(term w) = frequency(w) * [s
2. relevance(term w | topic t) = λ * p(w

In [ ]: