Document Clustering and Topic Modeling

In this project, I used unsupervised learning models to cluster unlabeled documents into different groups, visualized the results, and identified their latent topics/structures.

Contents

Part 1: Load Data

In [1]:
```
# Load libraries
import numpy as np
import pandas as pd
import nltk
import random
import re
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

from nltk.stem.snowball import SnowballStemmer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.cluster import KMeans
from yellowbrick.cluster import SilhouetteVisualizer
from sklearn.decomposition import PCA, KernelPCA
from sklearn.decomposition import LatentDirichletAllocation

random.seed(20202200)
```

In [2]:
```
# Load data into dataframe
df = pd.read_csv("Review_data.csv", sep=',', header=0)
```

In [3]:
```
df.head()
```

Out[3]:

	review_body	star_rating
0	Good luck finding a reasonably priced band rep...	1
1	No i get dark on the first week with me!! I wi...	1
2	I dont know if mine was a mistake but it clear...	1
3	The rod that holds the bracelet broke several ...	1
4	I bought 2 watches , one watch doesnot work at...	1

In [4]:
```
# Cheching if there is any missing value
df.isnull().sum()
```

Out[4]:
```
review_body    0
star_rating    0
dtype: int64
```

In [5]:
```
df.star_rating.value_counts()
```

Out[5]:
```
1    3000
2    3000
3    3000
4    3000
5    3000
Name: star_rating, dtype: int64
```

In [6]:
```
# Take only the review_body column for unsupervised learning task

data = df.loc[:, 'review_body'].tolist()
print(type(data))
print(len(data))
```

```
<class 'list'>
15000
```

In [7]:
```
# Take a look at some of the reviews
for _ in range(5):
    print(data[_],"\n")
```

```
Good luck finding a reasonably priced band replacement. I ordered the band fr
om the dealer who sold it to me (no one else in town could get one) and Skage
n sent the wrong one.  I guess I'll try again, but not allowing anyone else t
o make bands for your unique watch design seems stupid. I will certainly neve
r buy one again.

No i get dark on the first week with me!! I will never buy this item and i ha
d buy 5 of them

I dont know if mine was a mistake but it clearly states aqua so im confused w
hy mine is lime green. I hate lime green and am very irritated. This is why p
eople hate ordering on amazon. Ive spent 100s of dollars on here latey and th
is one will make me not want to order.  At least its not much money. Just ann
oying thinking u ordered something and get something else.  Well its going in
the trash...

The rod that holds the bracelet broke several times and the company do not fi
x it, it is sitting on the drawer so I can come to see the Jeweler to try to
fix one more time. Don't buy it. Really. Don't buy it, It is headache.

I bought 2 watches , one watch doesnot work at all, other watch  runs, its ti
me slows down 5-10 minutes backward. Outwardly the watches look beautiful, it
doesnot show time . I don't know why you are selling these kind of watches on
line. It is a waste my money and time that I bought these watches.
```

In [8]:
```
import nltk
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.
xml (https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml)
```

Out[8]:
```
True
```

Part 2: Tokenizing and Stemming

In [9]:
```
# Use nltk's English stopwords.
stopwords = stopwords.words('english')

print("We use " + str(len(stopwords)) + " stop-words from nltk library.")
```

```
We use 179 stop-words from nltk library.
```

In [ ]:
```

```

Use our defined functions to analyze (i.e. tokenize, stem) our reviews.

In [10]:
```
def tokenization_and_stemming(text):
    '''
    INPUT
    text - string
    OUTPUT
    clean_tokens - a list of words
    This function processes the input using the following steps :
    1. Remove punctuation characters
    2. Tokenize text into list
    3. Stem, Normalize and Strip each word
    4. Remove stop words
    '''
    # Remove punctuation characters and numbers
    text = re.sub(r"[^a-zA-Z]", " ", text)

    # Tokenize text
    tokens = word_tokenize(text)

    # Create a instance of stem class
    stemmer = SnowballStemmer("english")

    clean_tokens = []
    for word in tokens:
        clean_tok = stemmer.stem(word).lower().strip()
        if clean_tok not in stopwords:
            clean_tokens.append(clean_tok)

    return clean_tokens
```

In [11]:
```
tokenization_and_stemming(data[42])
```

Out[11]:
```
['warranti', 'card', 'box']
```

Part 3: TF-IDF:Term Frequency-Inverse Document Frequency

In this part, I use the TfidfVectorizer() from the sklearn library to create the tf-idf matrix

In [12]:
```
tfidf_model = TfidfVectorizer(
    max_df=0.99, # max_df : maximum document frequency for the given word
    max_features=1000, # max_features: maximum number of words
    min_df=0.01, # min_df : minimum document frequency for the given word
    use_idf=True, # use_idf: if not true, we only calculate tf
    tokenizer=tokenization_and_stemming,
    ngram_range=(1,1) # ngram_range: (min, max), eg. (1, 2) including 1-gram,
)


# Fit the TfidfVectorizer to our data
tfidf_matrix = tfidf_model.fit_transform(data)


print("In total, there are {} reviews and {} terms.".format(
    str(tfidf_matrix.shape[0]), str(tfidf_matrix.shape[1])
))
```

```
E:\ProgramData\anaconda3\lib\site-packages\sklearn\feature_extraction\text.p
y:528: UserWarning: The parameter 'token_pattern' will not be used since 'tok
enizer' is not None'
  warnings.warn(

In total, there are 15000 reviews and 445 terms.
```

In [13]:
```
# Check the parameters
tfidf_model.get_params()
```

Out[13]:
```
{'analyzer': 'word',
 'binary': False,
 'decode_error': 'strict',
 'dtype': numpy.float64,
 'encoding': 'utf-8',
 'input': 'content',
 'lowercase': True,
 'max_df': 0.99,
 'max_features': 1000,
 'min_df': 0.01,
 'ngram_range': (1, 1),
 'norm': 'l2',
 'preprocessor': None,
 'smooth_idf': True,
 'stop_words': None,
 'strip_accents': None,
 'sublinear_tf': False,
 'token_pattern': '(?u)\\b\\w\\w+\\b',
 'tokenizer': <function __main__.tokenization_and_stemming(text)>,
 'use_idf': True,
 'vocabulary': None}
```

Save the terms identified by TF-IDF.

In [15]:
```
# Words
tf_selected_words = tfidf_model.get_feature_names_out()
```

In [16]:
```
tfidf_matrix
```

Out[16]:
```
<15000x445 sparse matrix of type '<class 'numpy.float64'>'
        with 238491 stored elements in Compressed Sparse Row format>
```

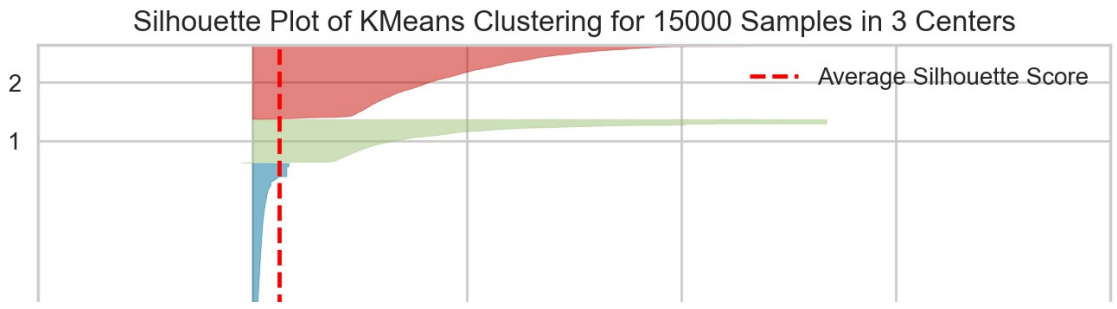
Part 4: K-means clustering

In this part, I perform the K-means algorithm to find out possible clusters in our reviews dataset.

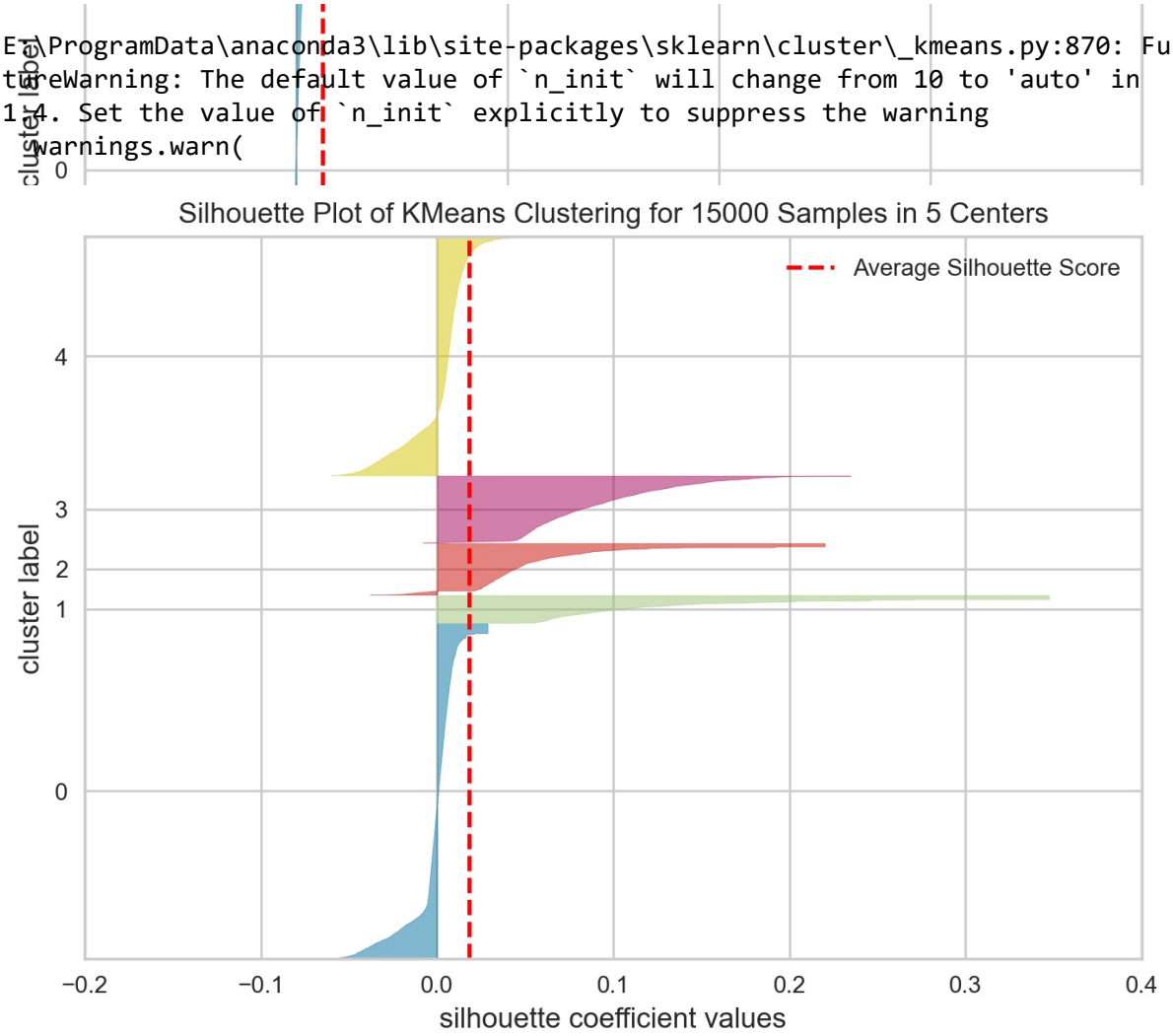
In [17]:
```
# Number of clusters
num_clusters = (2,3,5)

for num in num_clusters:
    kmeans = KMeans(n_clusters=num)

    model_km = SilhouetteVisualizer(kmeans, colors='yellowbrick')
    model_km.fit(tfidf_matrix) # Fit the data to the visualizer

    model_km.show()
```

```
E:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fu
tureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```


```
E:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fu
tureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



```
E:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fu
tureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



From the silhouette plot above, we can see the average silhouette coefficients are very similar among different solutions. There is no significant preference over one solution. As we know our dataset contains the product reviews, the reviews probably would fall into one of positive, neutral or negative clusters. So I decide to use 3 as the number of clusters in the kmeans.

In [18]:
```
kmeans_model = KMeans(n_clusters=3)

kmeans_model.fit(tfidf_matrix) # Fit the data
```

```
E:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fu
tureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

Out[18]:

```
KMeans
KMeans(n_clusters=3)
```

4.1. Analyze K-means Result

In [19]:
```
kmeans_results = df.rename({'review_body':'review'})
clusters = kmeans_model.labels_.tolist()
kmeans_results['cluster'] = clusters
```

In [20]:
```
kmeans_results.head(10)
```

Out[20]:

	review_body	star_rating	cluster
0	Good luck finding a reasonably priced band rep...	1	0
1	No i get dark on the first week with me!! I wi...	1	0
2	I dont know if mine was a mistake but it clear...	1	0
3	The rod that holds the bracelet broke several ...	1	0
4	I bought 2 watches , one watch doesnot work at...	1	0
5	This watch would have been fantastic, if it ha...	1	0
6	I have this watch. It looks and feels heavy du...	1	0
7	What the hell! I just got the watch today but ...	1	0
8	I am mechanically inclined but cannot get this...	1	0
9	It didnt work right out from the box. I had to...	1	0

In [21]:
```
print ("Number of reviews included in each cluster:")
cluster_size = kmeans_results['cluster'].value_counts().to_frame()
cluster_size
```

```
Number of reviews included in each cluster:
```

Out[21]:

	cluster
0	12612
2	1500
1	888

In [22]:
```
kmeans_results.groupby('cluster')['star_rating'].value_counts()
```

Out[22]:
```
cluster  star_rating
0        1              2632
         2              2626
         5              2523
         3              2490
         4              2341
1        4               316
         5               238
         3               181
         2                78
         1                75
2        4               343
         3               329
         2               296
         1               293
         5               239
Name: star_rating, dtype: int64
```

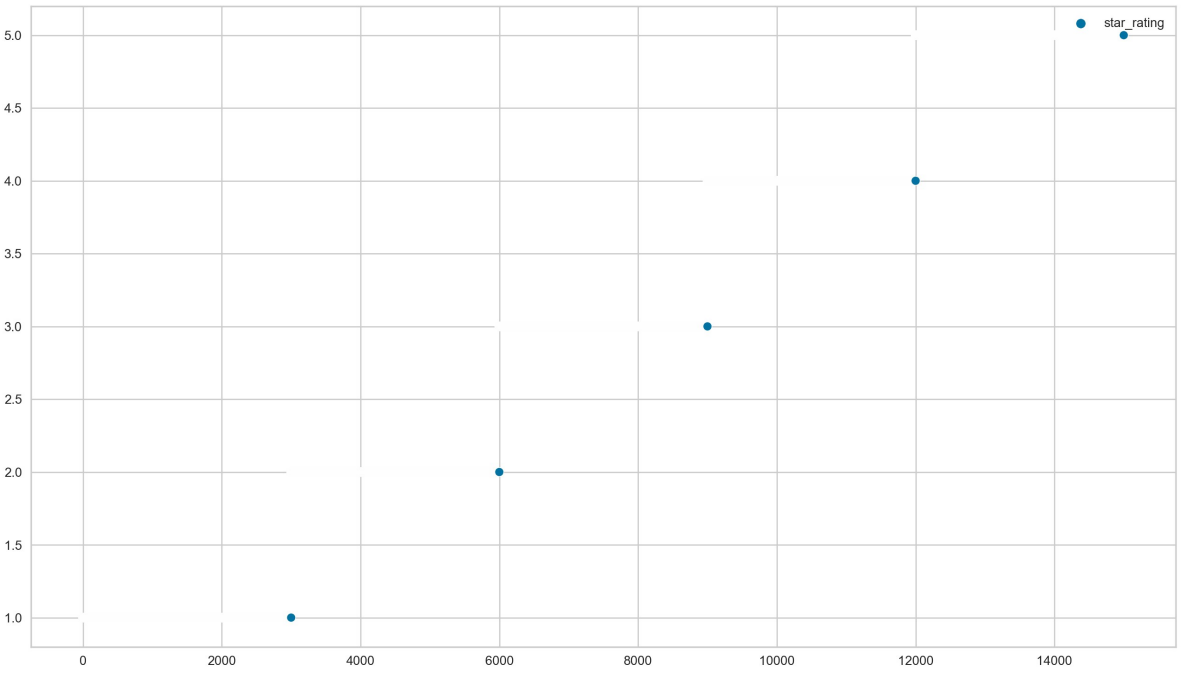
We can see that cluster 0 contains more negative reviews while cluster 2 contains more positive reviews. The reviews in cluster 1 are more neutral. And we can also see that the sizes are quite different among groups.

4.2. Plot the kmeans result

In [23]:
```
pca = KernelPCA(n_components=2)
tfidf_matrix_np=tfidf_matrix.toarray()
X = pca.fit_transform(tfidf_matrix_np)

xs, ys = X[:, 0], X[:,1]
```

In [25]:
```
pca_df = pd.DataFrame(dict(x = xs, y = ys, Cluster = clusters ))
plt.subplots(figsize=(16,9))
sns.scatterplot(data=df)
```

Out[25]: <Axes: >



Part 5: Topic Modeling - Latent Dirichlet Allocation

In [26]:
```
# Use LDA for clustering
LDA = LatentDirichletAllocation(n_components=3)
```

In [27]:
```
# Term frequency for LDA model
tf_lda = CountVectorizer(
    max_df=0.99,
    max_features=500,
    min_df=0.01,
    tokenizer=tokenization_and_stemming,
    ngram_range=(1,1))


tf_matrix_lda = tf_lda.fit_transform(data)


print ("In total, there are {} reviews and {} terms.".format(
    str(tf_matrix_lda.shape[0]), str(tf_matrix_lda.shape[1])
))
```

```
E:\ProgramData\anaconda3\lib\site-packages\sklearn\feature_extraction\text.p
y:528: UserWarning: The parameter 'token_pattern' will not be used since 'tok
enizer' is not None'
  warnings.warn(

In total, there are 15000 reviews and 445 terms.
```

In [28]:
```
print(tf_matrix_lda.shape)
```

```
(15000, 445)
```

In [30]:
```
# Feature names
lda_feature_name = tf_lda.get_feature_names_out()
```

In [31]:
```
# Document topic matrix for tf_matrix_lda
lda_output = LDA.fit_transform(tf_matrix_lda)
print(lda_output.shape)
```

```
(15000, 3)
```

In [32]:
```
# Topics and words matrix
# Components_[i, j] can be viewed as pseudocount that represents the number of
topic_word = LDA.components_
print(topic_word.shape)
```

```
(3, 445)
```

In [33]:
```
# Column names
topic_names = ["Topic" + str(i) for i in range(LDA.n_components)]

# Index names
doc_names = ["Doc" + str(i) for i in range(len(data))]

df_document_topic = pd.DataFrame(np.round(lda_output, 2), columns=topic_names,

# Get dominant topic for each document
topic = np.argmax(df_document_topic.values, axis=1)
df_document_topic['topic'] = topic

df_document_topic.head(10)
```

Out[33]:

	Topic0	Topic1	Topic2	topic
Doc0	0.01	0.59	0.39	1
Doc1	0.19	0.77	0.04	1
Doc2	0.21	0.78	0.02	1
Doc3	0.32	0.66	0.02	1
Doc4	0.02	0.97	0.02	1
Doc5	0.59	0.40	0.01	0
Doc6	0.65	0.25	0.10	0
Doc7	0.57	0.40	0.03	0
Doc8	0.73	0.25	0.02	0
Doc9	0.03	0.93	0.03	1

In [34]:
```
df_document_topic['topic'].value_counts().to_frame()
```

Out[34]:

	topic
2	6351
1	5266
0	3383

The cluster size is more even in this case.

In [36]:
```
# Topic-word matrix
df_topic_words = pd.DataFrame(LDA.components_)

# Column and index
df_topic_words.columns = tf_lda.get_feature_names_out()
df_topic_words.index = topic_names

df_topic_words.head()
```

Out[36]:

	abl	absolut	accur	actual	adjust	advertis	ago	
Topic0	182.067081	17.530736	334.093979	183.076160	445.145973	91.307843	32.930609	434.
Topic1	82.448989	118.679910	7.718927	83.271724	1.275352	69.096551	250.336612	0.
Topic2	78.483930	73.789354	65.187094	309.652116	209.578675	70.595606	13.732779	0.

3 rows × 445 columns

In [38]:
```
# Print top n keywords for each topic
def print_topic_words(tfidf_model, lda_model, n_words):
    words = np.array(tfidf_model.get_feature_names_out())
    topic_words = []
    # For each topic, we have words weight
    for topic_words_weights in lda_model.components_:
        top_words = topic_words_weights.argsort()[::-1][:n_words]
        topic_words.append(words.take(top_words))
    return topic_words

topic_keywords = print_topic_words(tfidf_model=tf_lda, lda_model=LDA, n_words=

df_topic_words = pd.DataFrame(topic_keywords)
df_topic_words.columns = ['Word '+str(i) for i in range(df_topic_words.shape[1
df_topic_words.index = ['Topic '+str(i) for i in range(df_topic_words.shape[0]
df_topic_words
```

Out[38]:

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9	Word 10	Word 11	Wo
Topic 0	br	watch	time	use	set	hand	day	one	work	onli	light	veri	rea
Topic 1	watch	work	one	batteri	time	love	veri	product	bought	great	good	buy	loc
Topic 2	watch	look	band	like	veri	nice	wrist	face	wear	great	good	would	b

In [39]:
```
df_document_topic["star_rating"] = df.star_rating.values
```

In [40]:
```
df_document_topic.groupby('topic')['star_rating'].value_counts()
```

Out[40]:
```
topic  star_rating
0      3              837
       2              800
       4              673
       1              627
       5              446
1      1             1731
       2             1059
       5             1031
       3              757
       4              688
2      4             1639
       5             1523
       3             1406
       2             1141
       1              642
Name: star_rating, dtype: int64
```

From the result above, it seems that topic 0 is more like a negative review topic. We also see words like "replace" and "return" score very high in the output. Both topic 1 and 2 seem to be kind of positive.

Part 6: Discussion

In this project, I tried out both K-means and LDA as examples of clustering and topic modeling.

K-means has some limitations. It is very sensitive to outliers. As you may have already seen, it can produce very small clusters corresponding to outliers. And K-means also has difficulties with clusters of different sizes and densities. That is why I randomly select equal size subsets of different star ratings reviews from the whole dataset.

Latent Dirichlet allocation (LDA) is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. The LDA model is highly modular and can, therefore, be easily extended. The main field of interest is modeling relations between topics. In this task, LDA did a better job of clustering the reviews.

In [ ]:

In [ ]: