Latent Dirichlet Allocation(LDA) for Topic Modelling (Python)

Using the Amazon fine food reviews dataset

Link:https://www.kaggle.com/snap/amazon-fine-food-reviews

For performing LDA based topic modelling , I will be using the gensim package for LDA topic modelling & pyLDAvis for visualization of LDA topic model

1. Import Packages

The core packages used in this tutorial are `re` , `gensim` , `spacy` and `pyLDAvis` . Besides this we will also using `matplotlib` , `numpy` and `pandas` for data handling and visualization. Let's import them.

In [2]:
```
import pandas as pd
import numpy as np

import re
import string

import spacy

import nltk
# nltk.download('stopwords')

import gensim
from gensim import corpora

# libraries for visualization
import pyLDAvis
import pyLDAvis.gensim
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

2. Reading the data

In [3]:
```
review_data= pd.read_csv("Reviews.csv")

# print(review_data.head(2))

review_data.head()
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all

3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If ing
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Gre at Th

In [4]:
```
print("Length of the data is :" , len(review_data))

print('Unique Products : ' , len(review_data.groupby('ProductId')))


print('Unique Users : ', len(review_data.groupby('UserId')))
```

```
Length of the data is : 568454
Unique Products :  74258
Unique Users :  256059
```

3. Cleaning Text

In [5]:
```
def clean_text(text ):
    delete_dict = {sp_character: '' for sp_character in string.punctuation}
    delete_dict[' '] = ' '
    table = str.maketrans(delete_dict)
    text1 = text.translate(table)
    #print('cleaned:'+text1)
    textArr= text1.split()
    text2 = ' '.join([w for w in textArr if ( not w.isdigit() and  ( not w.isdigit() and len(w)>3))])

    return text2.lower()
```

In [7]:
```
review_data.dropna(axis = 0, how ='any',inplace=True)


review_data['Text'] = review_data['Text'].apply(clean_text)
review_data['Num_words_text'] = review_data['Text'].apply(lambda x:len(str(x).split()))

print('-------Dataset --------')

print(review_data['Score'].value_counts())
print(len(review_data))

print('-------------------------')
max_review_data_sentence_length  = review_data['Num_words_text'].max()

mask = (review_data['Num_words_text'] < 100) & (review_data['Num_words_text'] >=20)
df_short_reviews = review_data[mask]
df_sampled = df_short_reviews.groupby('Score').apply(lambda x: x.sample(n=20000)).reset_index(drop = True)

print('No of Short reviews')
print(len(df_short_reviews))
```

```
-------Dataset --------
Score
5    363102
4     80654
1     52264
3     42638
2     29743
Name: count, dtype: int64
568401
-------------------------
No of Short reviews
373279
```

4. Pre-Process the Text

In [8]:
```
from nltk.corpus import stopwords

stop_words = stopwords.words('english')

# function to remove stopwords
def remove_stopwords(text):
    textArr = text.split(' ')
    rem_text = " ".join([i for i in textArr if i not in stop_words])
    return rem_text

# remove stopwords from the text
df_sampled['Text']=df_sampled['Text'].apply(remove_stopwords)
```

5. Lemmetization

In [9]:
```
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])

def lemmatization(texts,allowed_postags=['NOUN', 'ADJ']):
       output = []
       for sent in texts:
             doc = nlp(sent)
             output.append([token.lemma_ for token in doc if token.pos_ in allowed_postags ])
       return output
```

In [10]:
```
text_list=df_sampled['Text'].tolist()
print(text_list[1])

print('-'*60)

tokenized_reviews = lemmatization(text_list)
print(tokenized_reviews[1])
```

```
dark roast coffee notice buying like dark roast coffee seems burned strong like starbucks coffee probably
like coffee took taste dumped entire smell strong reminds nasty cigar
------------------------------------------------------------
['coffee', 'notice', 'buying', 'dark', 'roast', 'coffee', 'strong', 'starbuck', 'coffee', 'coffee', 'tast
e', 'entire', 'smell', 'strong', 'nasty', 'cigar']
```

6. Create vocabulary dictionary and document term matrix

In [11]:
```
dictionary = corpora.Dictionary(tokenized_reviews)

doc_term_matrix = [dictionary.doc2bow(rev) for rev in tokenized_reviews]
```

7. LemmetizationCreating the object for LDA model using gensim library

In [12]:
```
LDA = gensim.models.ldamodel.LdaModel

# Build LDA model
lda_model = LDA(corpus=doc_term_matrix,
                id2word=dictionary,
                num_topics=10,
                random_state=100,
                chunksize=1000,
                passes=50,
                iterations=100)
```

In [13]:
```
lda_model.print_topics()
```

Out[13]:
```
[(0,
  '0.051*"butter" + 0.035*"bread" + 0.031*"peanut" + 0.026*"recipe" + 0.023*"soup" + 0.021*"can" + 0.017
*"bone" + 0.012*"mint" + 0.012*"coat" + 0.012*"restaurant"'),
 (1,
  '0.050*"bar" + 0.045*"vanilla" + 0.039*"smooth" + 0.036*"tea" + 0.035*"green" + 0.033*"blend" + 0.032*"p
otato" + 0.024*"breakfast" + 0.019*"black" + 0.015*"brew"'),
```

```
 (2,
  '0.034*"taste" + 0.033*"flavor" + 0.031*"good" + 0.031*"sugar" + 0.030*"sweet" + 0.028*"water" + 0.024
*"product" + 0.019*"drink" + 0.019*"great" + 0.018*"calorie"'),
 (3,
  '0.038*"organic" + 0.032*"fruit" + 0.028*"cereal" + 0.024*"rice" + 0.023*"free" + 0.021*"honey" + 0.020
*"cracker" + 0.020*"healthy" + 0.020*"corn" + 0.019*"whole"'),
 (4,
  '0.076*"food" + 0.047*"product" + 0.018*"good" + 0.018*"year" + 0.018*"time" + 0.015*"great" + 0.014*"mo
nth" + 0.014*"cat" + 0.012*"happy" + 0.011*"problem"'),
 (5,
  '0.070*"chocolate" + 0.041*"cookie" + 0.028*"snack" + 0.027*"candy" + 0.026*"great" + 0.020*"good" + 0.0
18*"popcorn" + 0.017*"gift" + 0.017*"kid" + 0.017*"love"'),
 (6,
  '0.112*"coffee" + 0.038*"flavor" + 0.030*"good" + 0.022*"strong" + 0.019*"taste" + 0.015*"great" + 0.013
*"bean" + 0.011*"kcup" + 0.011*"morning" + 0.010*"nice"'),
 (7,
  '0.057*"flavor" + 0.039*"good" + 0.035*"chip" + 0.026*"great" + 0.024*"salt" + 0.021*"taste" + 0.017*"sa
uce" + 0.013*"snack" + 0.013*"little" + 0.013*"favorite"'),
 (8,
  '0.052*"treat" + 0.026*"dog" + 0.025*"time" + 0.025*"small" + 0.019*"size" + 0.016*"little" + 0.015*"gre
at" + 0.014*"minute" + 0.013*"large" + 0.013*"piece"'),
 (9,
  '0.056*"price" + 0.051*"store" + 0.032*"good" + 0.030*"product" + 0.029*"great" + 0.027*"amazon" + 0.023
*"local" + 0.018*"grocery" + 0.014*"order" + 0.013*"time"')]
```

8. Visualize the topics

In [14]:

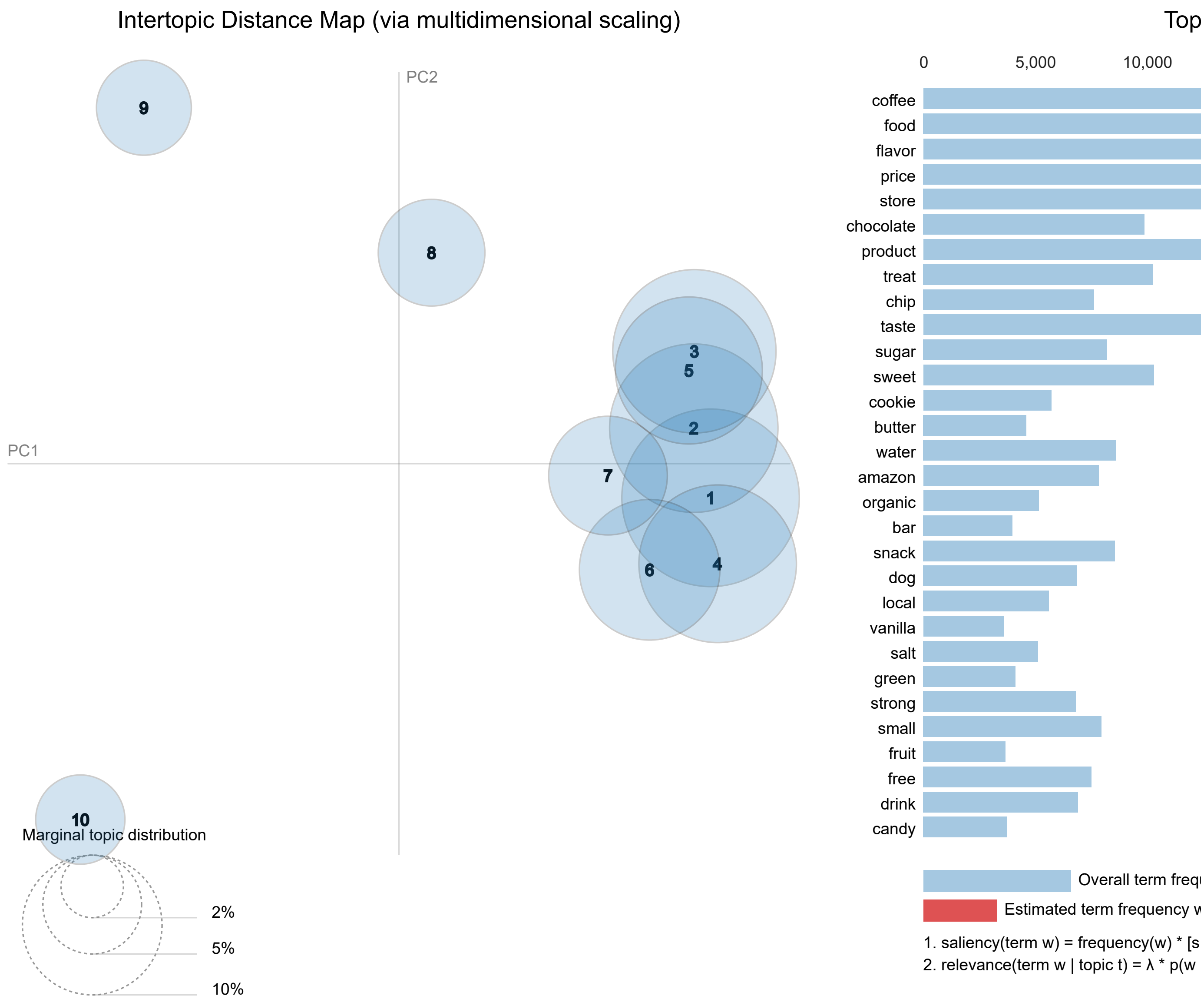
```
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, doc_term_matrix, dictionary)
vis
```

Out[14]:

Selected Topic: 0 Previous Topic Next Topic Clear Topic

Slide to adjust relevance metri (2)

λ = 1



9. Measuring how good the model is. lower the better.

In [ ]:
```
print('\nPerplexity: ', lda_model.log_perplexity(doc_term_matrix,total_docs=10000))
```

Computing Coherence Score

10. Computing Coherence Score

In [16]:
```
from gensim.models.coherencemodel import CoherenceModel

coherence_model_lda = CoherenceModel(model=lda_model, texts=tokenized_reviews, dictionary=dictionary , cohe
coherence_lda = coherence_model_lda.get_coherence()

print('\nCoherence Score: ', coherence_lda)
```

```
Coherence Score:  0.42386054991557176
```

11. Method to find optimal number of topics

In [17]:
```
def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
```

```
    coherence_values = []
    model_list = []

    for num_topics in range(start, limit, step):
        model = gensim.models.ldamodel.LdaModel(corpus=corpus, num_topics=num_topics, id2word=dictionary)
        model_list.append(model)

        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values
```

In [18]:

```
model_list, coherence_values = compute_coherence_values(dictionary=dictionary, corpus=doc_term_matrix, text
```

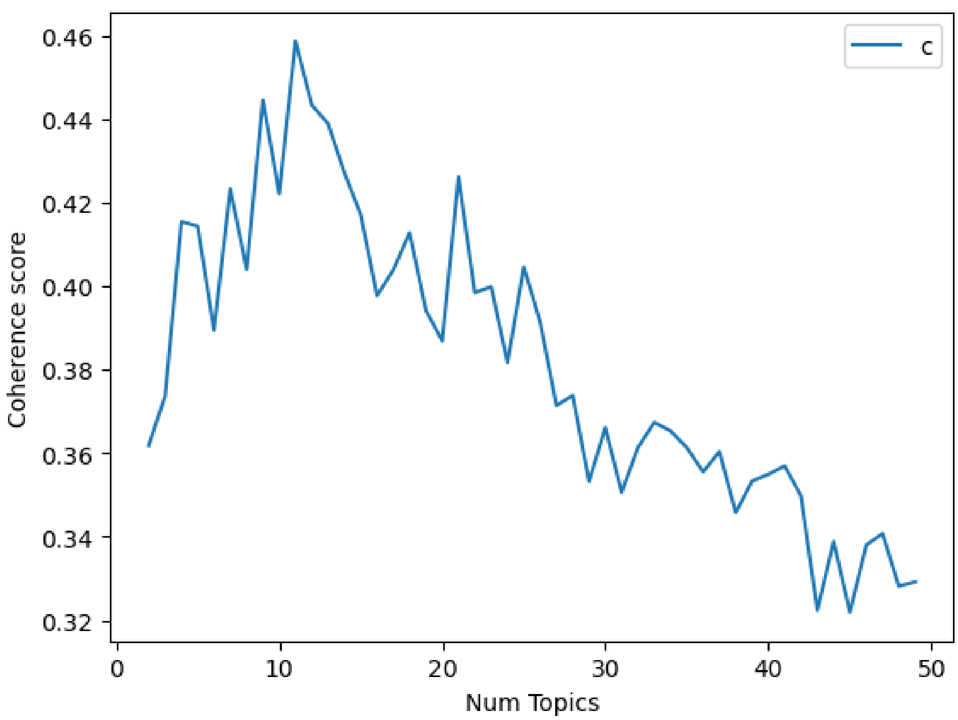
12. Viualizatoin with Graph

In [19]:

```
limit=50; start=2; step=1;
x = range(start, limit, step)

plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')

# Print the coherence scores
plt.show()
```



13.Printing the coherence scores

In [20]:

```
for m, cv in zip(x, coherence_values):
    print("Num Topics =", m, " has Coherence Value of", round(cv, 4))
```

```
Num Topics = 2  has Coherence Value of 0.3618
Num Topics = 3  has Coherence Value of 0.3736
Num Topics = 4  has Coherence Value of 0.4154
Num Topics = 5  has Coherence Value of 0.4143
Num Topics = 6  has Coherence Value of 0.3894
Num Topics = 7  has Coherence Value of 0.4232
Num Topics = 8  has Coherence Value of 0.404
Num Topics = 9  has Coherence Value of 0.4445
Num Topics = 10  has Coherence Value of 0.4221
Num Topics = 11  has Coherence Value of 0.4586
Num Topics = 12  has Coherence Value of 0.4433
Num Topics = 13  has Coherence Value of 0.4388
Num Topics = 14  has Coherence Value of 0.4271
```

```
Num Topics = 15  has Coherence Value of 0.4171
Num Topics = 16  has Coherence Value of 0.3977
Num Topics = 17  has Coherence Value of 0.4038
Num Topics = 18  has Coherence Value of 0.4128
Num Topics = 19  has Coherence Value of 0.3941
Num Topics = 20  has Coherence Value of 0.3869
Num Topics = 21  has Coherence Value of 0.4261
Num Topics = 22  has Coherence Value of 0.3984
Num Topics = 23  has Coherence Value of 0.3998
Num Topics = 24  has Coherence Value of 0.3816
Num Topics = 25  has Coherence Value of 0.4046
Num Topics = 26  has Coherence Value of 0.3914
Num Topics = 27  has Coherence Value of 0.3714
Num Topics = 28  has Coherence Value of 0.3738
Num Topics = 29  has Coherence Value of 0.3532
Num Topics = 30  has Coherence Value of 0.3661
Num Topics = 31  has Coherence Value of 0.3506
Num Topics = 32  has Coherence Value of 0.3613
Num Topics = 33  has Coherence Value of 0.3673
Num Topics = 34  has Coherence Value of 0.3653
Num Topics = 35  has Coherence Value of 0.3613
Num Topics = 36  has Coherence Value of 0.3555
Num Topics = 37  has Coherence Value of 0.3604
Num Topics = 38  has Coherence Value of 0.3457
Num Topics = 39  has Coherence Value of 0.3533
Num Topics = 40  has Coherence Value of 0.3549
Num Topics = 41  has Coherence Value of 0.3569
Num Topics = 42  has Coherence Value of 0.3498
Num Topics = 43  has Coherence Value of 0.3223
Num Topics = 44  has Coherence Value of 0.3388
Num Topics = 45  has Coherence Value of 0.3219
Num Topics = 46  has Coherence Value of 0.338
Num Topics = 47  has Coherence Value of 0.3407
Num Topics = 48  has Coherence Value of 0.3281
Num Topics = 49  has Coherence Value of 0.3292
```

14. Selecting the model and printing the topics

In [21]:
```
optimal_model = model_list[7]

model_topics = optimal_model.show_topics(formatted=False)

optimal_model.print_topics(num_words=10)
```

Out[21]:
```
[(0,
  '0.029*"good" + 0.029*"great" + 0.022*"product" + 0.019*"flavor" + 0.014*"taste" + 0.014*"energy" + 0.01
3*"popcorn" + 0.013*"cheese" + 0.012*"cracker" + 0.012*"snack"'),
 (1,
  '0.121*"coffee" + 0.032*"flavor" + 0.029*"good" + 0.023*"strong" + 0.019*"taste" + 0.014*"bean" + 0.013
*"great" + 0.012*"green" + 0.011*"smooth" + 0.010*"blend"'),
 (2,
  '0.039*"product" + 0.027*"time" + 0.022*"year" + 0.018*"order" + 0.018*"good" + 0.018*"month" + 0.015*"g
reat" + 0.015*"amazon" + 0.013*"store" + 0.011*"week"'),
 (3,
  '0.037*"price" + 0.031*"store" + 0.026*"great" + 0.025*"good" + 0.016*"amazon" + 0.014*"grocery" + 0.014
*"local" + 0.014*"product" + 0.012*"pack" + 0.011*"time"'),
 (4,
  '0.037*"flavor" + 0.033*"good" + 0.022*"sauce" + 0.022*"candy" + 0.021*"bar" + 0.019*"sweet" + 0.019*"gr
eat" + 0.019*"taste" + 0.015*"texture" + 0.015*"little"'),
 (5,
  '0.054*"treat" + 0.045*"chip" + 0.029*"cookie" + 0.027*"dog" + 0.020*"salt" + 0.020*"small" + 0.017*"gre
at" + 0.016*"size" + 0.016*"good" + 0.013*"love"'),
 (6,
  '0.047*"flavor" + 0.034*"chocolate" + 0.032*"sugar" + 0.028*"sweet" + 0.026*"good" + 0.025*"taste" + 0.0
20*"drink" + 0.013*"milk" + 0.011*"syrup" + 0.011*"great"'),
 (7,
  '0.031*"water" + 0.028*"butter" + 0.019*"bread" + 0.018*"great" + 0.018*"milk" + 0.018*"protein" + 0.018
*"product" + 0.017*"peanut" + 0.017*"good" + 0.016*"free"'),
 (8,
  '0.082*"food" + 0.022*"good" + 0.018*"healthy" + 0.015*"cat" + 0.014*"product" + 0.012*"ingredient" + 0.
012*"organic" + 0.012*"great" + 0.012*"brand" + 0.012*"rice"')]
```

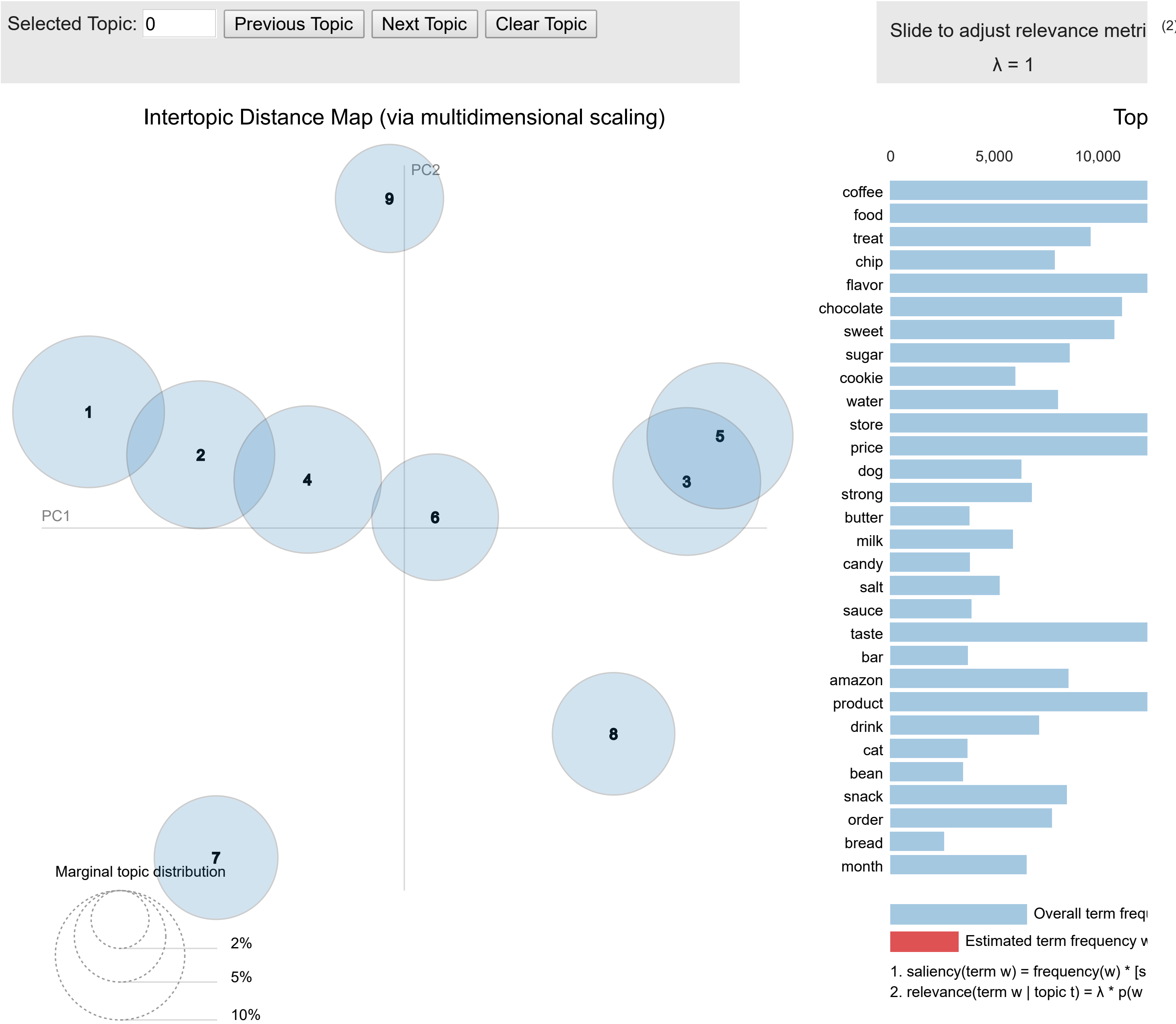
15. Visualize the topics

In [22]:
```
pyLDAvis.enable_notebook()

vis = pyLDAvis.gensim.prepare(optimal_model, doc_term_matrix, dictionary)

vis
```

Out[22]:



16. Saving to PDF

In [26]:
```
!jupyter nbconvert --to webpdf --allow-chromium-download Topic_Modeling_Using_LDA.ipynb
```

```
[NbConvertApp] Converting notebook Topic_Modeling_Using_LDA.ipynb to webpdf
[NbConvertApp] Building PDF
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 387666 bytes to Topic_Modeling_Using_LDA.pdf
```