Review Project Analysis

Description:

Help a leading mobile brand understand the voice of the customer by analyzing the reviews of their product on Amazon

Problem Statement:

A popular mobile phone brand, Lenovo has launched their budget smartphone in the Indian market. The client wants to understand the VOC (voice of the customer) on the produ evaluate the current product, but to also get some direction for developing the product pipeline. The client is particularly interested in the different aspects that customers care ab on a leading e-commerce site should provide a good view.

Steps to perform:

Discover the topics in the reviews and present it to business in a consumable format. Employ techniques in syntactic processing and topic modeling.

Perform specific cleanup, POS tagging, and restricting to relevant POS tags, then, perform topic modeling using LDA (Latent Dirichlet Allocation). Finally, give business-friendly r for business.

Latent Dirichlet Allocation

In [11]:
```
import pandas as pd
import numpy as np

import nltk
from nltk.stem import WordNetLemmatizer

import string

import seaborn as sns
import matplotlib.pyplot as plt

import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.ERROR)

import warnings
warnings.filterwarnings('ignore')
```

Content:

Dataset: 'K8 Reviews v0.2.csv'

Columns: ['sentiment' , 'review']

1) Read the .csv file using Pandas. Take a look at the top few records.

In [12]:
```
df = pd.read_csv('K8 Reviews v0.2.csv')
df.head()
```

Out[12]:

	sentiment	review
0	1	Good but need updates and improvements
1	0	Worst mobile i have bought ever, Battery is dr...
2	1	when I will get my 10% cash back.... its alrea...
3	1	Good
4	0	The worst phone everThey have changed the last...

In [13]:
```
df.columns
```

Out[13]:
```
Index(['sentiment', 'review'], dtype='object')
```

In [14]:
```
df.sentiment.value_counts()
```

Out[14]:
```
0    7712
1    6963
Name: sentiment, dtype: int64
```

The sentiment against the review (4,5 star reviews are positive, 1,2 are negative)

- number of negative review: 7712
- number of positive review: 6963

2) Normalize casings for the review text and extract the text into a list for easier manipulation.

In [15]:
```
review_text = list(df['review'].values)
review_text = [item.lower() for item in review_text]
review_text[0]
```

Out[15]:
```
'good but need updates and improvements'
```

3) Tokenize the reviews using NLTKs word_tokenize function.

In [16]:
```
from nltk.tokenize import word_tokenize
review_tokens = [word_tokenize(review) for review in review_text]
review_tokens[0]
```

Out[16]:
```
['good', 'but', 'need', 'updates', 'and', 'improvements']
```

In [17]:
```
review_tokens[0]
```

Out[17]:
```
['good', 'but', 'need', 'updates', 'and', 'improvements']
```

In [18]:
```
clean_review_tokens = []
# remove remaining tokens that are not alphabetic
for list_token in review_tokens:
    clean_review_tokens.append([word for word in list_token if word.isalpha()])
```

In [19]:
```
clean_reviews = [[token for token in clean_review_token if len(token) > 1] for clean_review_token in clean_review_tokens]

#docs = [[token for token in doc if len(token) > 1] for doc in docs]
len(clean_reviews)
```

Out[19]:
```
14675
```

4) Perform parts-of-speech tagging on each sentence using the NLTK POS tagger.

In [20]:
```
all_nouns = []

review_pos = [nltk.pos_tag(tokens) for tokens in clean_reviews]
review0_nouns = [term for term,pos in review_pos[0] if pos.startswith("NN")]
```

5) For the topic model, we should want to include only nouns.

- Find out all the POS tags that correspond to nouns.
- Limit the data to only terms with these tags.

In [21]:
```
for sent in review_pos:
    all_nouns.append([term for term,pos in sent if pos.startswith("NN")])
```

In [22]:
```
review0_nouns
```

Out[22]:
```
['updates', 'improvements']
```

In [23]:
```
all_nouns[0:2]
```

Out[23]:
```
[['updates', 'improvements'],
 ['battery',
  'hell',
  'backup',
  'hours',
  'internet',
  'uses',
  'lie',
  'amazon',
  'lenove',
  'battery',
  'charger',
  'hours',
  'don']]
```

In [24]:
```
print(len(review_text))
print(len(all_nouns))
```

```
14675
14675
```

6) Lemmatize.

- Different forms of the terms need to be treated as one.
- No need to provide POS tag to lemmatizer for now.

In [25]:
```
lemmatizer = WordNetLemmatizer()

lemmatized_nouns = []
for nouns in all_nouns:
    lemmatized_nouns.append([lemmatizer.lemmatize(item) for item in nouns])
```

In [26]:
```
lemmatized_nouns[0:2]
```

Out[26]:
```
[['update', 'improvement'],
 ['battery',
  'hell',
  'backup',
  'hour',
  'internet',
  'us',
  'lie',
  'amazon',
  'lenove',
  'battery',
  'charger',
  'hour',
  'don']]
```

7) Remove stopwords and punctuation (if there are any).

In [27]:
```
from nltk.corpus import stopwords
from string import punctuation
```

In [28]:
```
stop_nltk = stopwords.words("english")
stop_punct = list(punctuation)
stop_final = stop_nltk + stop_punct
print(len(stop_final))

text_clean = []
for term in lemmatized_nouns:
    text_clean.append([item for item in term if item not in stop_final])

text_clean[0:2]
```

```
211
```

Out[28]:
```
[['update', 'improvement'],
 ['battery',
  'hell',
  'backup',
  'hour',
  'internet',
  'us',
  'lie',
  'amazon',
  'lenove',
  'battery',
  'charger',
  'hour']]
```

Latent Dirichlet Allocation (LDA)

- LDA is used to classify text in a document to a particular topic. It builds a topic per document model and words per topic model, modeled as Dirichlet distributions.
- Each document is modeled as a multinomial distribution of topics and each topic is modeled as a multinomial distribution of words.
- LDA assumes that the every chunk of text we feed into it will contain words that are somehow related. Therefore choosing the right corpus of data is crucial. It also assumes mixture of topics. Those topics then generate words based on their probability distribution.

8) Create a topic model using LDA on the cleaned-up data with 12 topics

- Print out the top terms for each topic.
- What is the coherence of the model with the c_v metric?

In [29]:
```
import gensim
from gensim.models import CoherenceModel

from tqdm import tqdm_notebook  as tqdm
```

In [30]:
```
dictionary = gensim.corpora.Dictionary(text_clean)

dictionary.filter_extremes(no_below=15, no_above=0.5)
print(dictionary)
```

```
Dictionary(517 unique tokens: ['improvement', 'update', 'amazon', 'backup', 'battery']...)
```

In [31]:
```
# Bag-of-words representation of the documents.
bow_corpus = [dictionary.doc2bow(doc) for doc in text_clean]

print(bow_corpus[0])
```

```
[(0, 1), (1, 1)]
```

The doc2bow() function converts dictionary into a bag-of-words

- In each document vector is a series of tuples
- The tuples are (term ID, term frequency) pairs
- This includes terms that actually occur - terms that do not occur in a document will not appear in that document's vector

In [32]:
```
# show the actual terms with the term frequency
[[(dictionary[id], freq) for id, freq in cp] for cp in bow_corpus[:1]]
```

Out[32]:
```
[[('improvement', 1), ('update', 1)]]
```

Hyperparameter Tuning for random_state

In [34]:
```
from tqdm import tqdm_notebook as tqdm

limit = 13
num_topics = 12
rand = list(np.arange(13,40,2))

b = 0.01
a = 0.61
for r in tqdm(rand):
    lm = gensim.models.LdaModel(corpus=bow_corpus, num_topics=num_topics, alpha=a, eta=b, random_state=r, id2word=dictionary)
    cm = CoherenceModel(model=lm, texts=text_clean, dictionary=dictionary,coherence='c_v')

    print("{} - {} ".format(cm.get_coherence(),r))
```

```
0.5157842940605378 - 13
0.5359865795903644 - 15
0.5018890695362971 - 17
0.5209738325472831 - 19
0.5578559179702802 - 21
0.5700638626880947 - 23
0.5494904611263187 - 25
0.5599743497333297 - 27
0.5429752957588752 - 29
0.525456132357209 - 31
0.542447018892657 - 33
0.5333646140457798 - 35
0.5270315689907091 - 37
0.5145870514419812 - 39
```

random_state with highest score: 23

Hyperparameter Tuning for Alpha and Eta

In [35]:
```
from tqdm import tqdm_notebook as tqdm

limit = 13
alpha=list(np.arange(0.01,1,0.3))
beta=list(np.arange(0.01,1,0.3))
num_topics = 12
r=25

for b in tqdm(beta):
    for a in alpha:
        lm = gensim.models.LdaModel(corpus=bow_corpus, num_topics=num_topics, alpha=a, eta=b, random_state=r, id2word=dictionary)
        cm = CoherenceModel(model=lm, texts=text_clean, dictionary=dictionary,coherence='c_v')
        print("{} - {} - {} - {}".format(cm.get_coherence(),b,a,r))
```

```
0.5492151627225282 - 0.01 - 0.01 - 23
0.5192889699429395 - 0.01 - 0.31 - 23
0.5700638626880947 - 0.01 - 0.61 - 23
0.49715101468090234 - 0.01 - 0.9099999999999999 - 23
0.5490656425555177 - 0.31 - 0.01 - 23
0.5154063763402096 - 0.31 - 0.31 - 23
0.5679656691829147 - 0.31 - 0.61 - 23
0.4971957150637602 - 0.31 - 0.9099999999999999 - 23
0.544147473127093 - 0.61 - 0.01 - 23
0.5196516638492322 - 0.61 - 0.31 - 23
0.578778811777873 - 0.61 - 0.61 - 23
0.49712731580181946 - 0.61 - 0.9099999999999999 - 23
0.5403136978460289 - 0.9099999999999999 - 0.01 - 23
0.5130125099027247 - 0.9099999999999999 - 0.31 - 23
0.582012823187397 - 0.9099999999999999 - 0.61 - 23
0.4985646456892578 - 0.9099999999999999 - 0.9099999999999999 - 23
```

Best value for Alpha and Eta: (b = 0.91, a = 0.61)

- Alpha = 0.61
- Eta = 0.91

Building the LDA model using the Hyperparameters found for 12 topics

In [67]:
```
b = 0.91
a = 0.61
r = 27
# Decided not to set the random_state and just use the default due to bias in the model
model = gensim.models.LdaModel(corpus=bow_corpus, num_topics=num_topics, alpha=a, eta=b, id2word=dictionary, passes=10)
```

In [68]:
```
for idx, topic in model.print_topics(-1):
    print("Topic: {} \nWords: {}".format(idx, topic))
    print("\n")
```

```
Topic: 0
Words: 0.644*"phone" + 0.092*"price" + 0.063*"feature" + 0.035*"range" + 0.024*"delivery" + 0.018*"budget" + 0.012*"smartphone" +
*"specification"


Topic: 1
Words: 0.390*"battery" + 0.075*"backup" + 0.060*"hour" + 0.058*"charger" + 0.044*"life" + 0.040*"charge" + 0.037*"day" + 0.028*"dr


Topic: 2
Words: 0.097*"day" + 0.083*"service" + 0.081*"amazon" + 0.044*"lenovo" + 0.040*"please" + 0.035*"customer" + 0.031*"return" + 0.02


Topic: 3
Words: 0.096*"screen" + 0.084*"lenovo" + 0.056*"sound" + 0.052*"processor" + 0.045*"experience" + 0.040*"thing" + 0.036*"work" + 0


Topic: 4
Words: 0.385*"camera" + 0.166*"quality" + 0.104*"performance" + 0.023*"speed" + 0.022*"superb" + 0.019*"mark" + 0.019*"picture" +
ok"


Topic: 5
Words: 0.231*"issue" + 0.157*"time" + 0.091*"network" + 0.082*"month" + 0.044*"lot" + 0.034*"use" + 0.025*"purchase" + 0.023*"repl


Topic: 6
Words: 0.155*"money" + 0.090*"heat" + 0.086*"display" + 0.072*"waste" + 0.063*"everything" + 0.059*"value" + 0.027*"worth" + 0.027


Topic: 7
Words: 0.079*"call" + 0.060*"option" + 0.059*"update" + 0.056*"software" + 0.056*"phone" + 0.047*"sim" + 0.030*"data" + 0.025*"cas


Topic: 8
Words: 0.286*"note" + 0.053*"hai" + 0.040*"video" + 0.032*"headphone" + 0.026*"lenovo" + 0.024*"ho" + 0.019*"mi" + 0.016*"way" + 0


Topic: 9
Words: 0.454*"product" + 0.288*"mobile" + 0.035*"system" + 0.016*"set" + 0.014*"awesome" + 0.009*"love" + 0.009*"offer" + 0.007*"f


Topic: 10
Words: 0.323*"problem" + 0.112*"heating" + 0.094*"device" + 0.045*"handset" + 0.027*"expectation" + 0.027*"review" + 0.021*"signal
15*"moto"


Topic: 11
Words: 0.058*"speaker" + 0.038*"music" + 0.037*"mode" + 0.032*"game" + 0.031*"app" + 0.030*"feature" + 0.029*"android" + 0.029*"ph
```

Coherence of model with c_v metric:

In [69]:
```
cm = CoherenceModel(model=model, texts=text_clean, dictionary=dictionary,coherence='c_v')
print("Coherence of model with c_v metric: {}".format(cm.get_coherence()))
```

```
Coherence of model with c_v metric: 0.5310035096340909
```

9) Analyze the topics through the business lens.

```
Determine which of the topics can be combined.
```

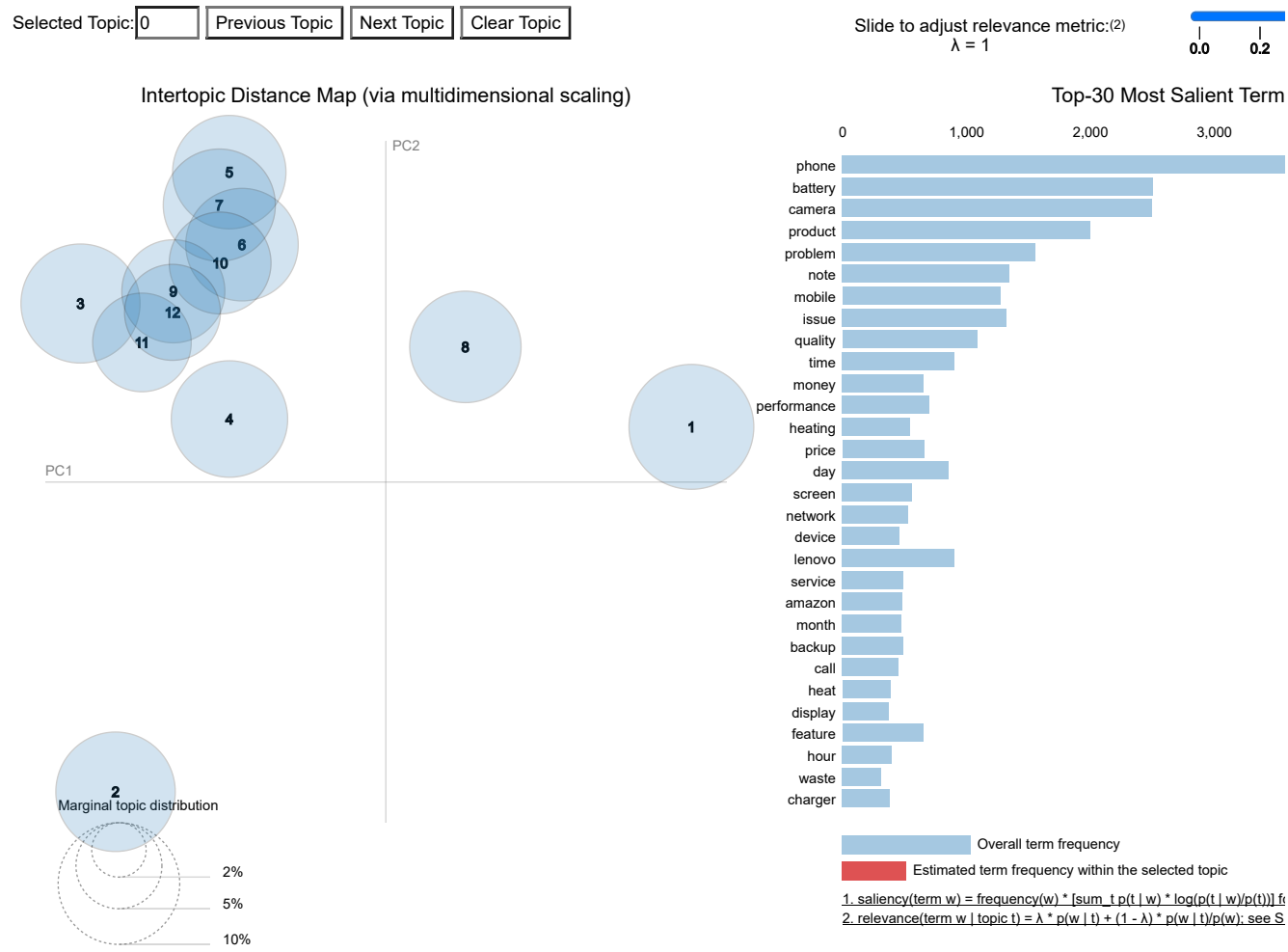
- pyLDAvis result as shown in the visualization below.
- The area of circle represents the importance of each topic over the entire corpus.
- The distance between the center of circles indicate the similarity between topics. For each topic.
- The histogram on the right side listed the top 30 most relevant terms.
- LDA extracted 12 main topics.

1 Topics 3 and 6 can be combined -- Features ###### 2 Topics 5 and 10 can be combined -- Problems and Issues ###### 3 Topics 3 and 11 can be combined -- me

In [70]:
```
import pyLDAvis
import pyLDAvis.gensim_models as gensimvis
pyLDAvis.enable_notebook()

# feed the LDA model into the pyLDAvis instance
lda_viz = gensimvis.prepare(model, bow_corpus, dictionary)
lda_viz
```

Out[70]:



Compute c_v coherence for various number of topics

In [71]:
```
def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
    """
    Compute c_v coherence for various number of topics
    Parameters:
    ----------
    dictionary : Gensim dictionary
    corpus : Gensim corpus
    texts : List of input texts
    limit : Max num of topics
    Returns:
    -------
    model_list : List of LDA topic models
    coherence_values : Coherence values corresponding to the LDA model with respective number of topics
    """
    # Set training parameters.
    b = 0.91
    a = 0.61
    r = 27
# Make a index to word dictionary.
    #temp = dictionary[0]  # This is only to "load" the dictionary.
    #id2word = dictionary.id2token
    coherence_values = []
    model_list = []
    for num_topics in tqdm(range(start, limit, step)):
        #model = gensim.models.wrappers.LdaMallet(mallet_path, corpus=corpus, num_topics=num_topics, id2word=id2word)
        #model = gensim.models.LdaModel(corpus=bow_corpus, num_topics=num_topics, alpha=a, eta=b,id2word=dictionary)
        model = gensim.models.LdaModel(corpus=bow_corpus, num_topics=num_topics, alpha=a, eta=b, random_state=r, id2word=dictiona
        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
```

Generate the list of LDA topics with each corresponding coherence_values

- model_list : List of LDA topic models
- coherence_values : Coherence values corresponding to the LDA model with respective number of topics

In [72]:
```
from tqdm import tqdm_notebook as tqdm
# Can take a long time to run.
model_list, coherence_values = compute_coherence_values(dictionary=dictionary, corpus=bow_corpus, texts=text_clean, start=2, limi
```

```
C:\Users\mglee\Anaconda3\lib\site-packages\ipykernel_launcher.py:24: TqdmDeprecationWarning: This function will be removed in tqdr
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
```

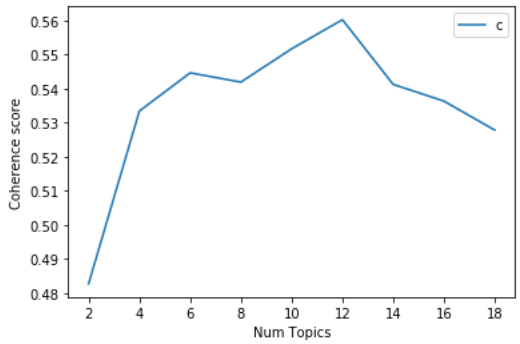
10) Create topic model using LDA with what you think is the optimal number of topics

- What is the coherence of the model?

made a plot showing: number of topics vs coherence score.

In [73]:
```
import matplotlib.pyplot as plt
%matplotlib inline

limit=20; start=2; step=2;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()
```



```
The plot shows the optimal number of topics.
Here we see 12 (Num Topics) have highest Coherence score.
12 topics is optimal number of topics.
```

11) The business should be able to interpret the topics.

- Name each of the identified topics.
- Create a table with the topic name and the top 10 terms in each to present to the business.

In [74]:
```
topics=model.show_topics(formatted=False)
```

In [75]:
```
topics = sorted(topics)
topics[1]
```

Out[75]:
```
(2,
 [('day', 0.096693695),
  ('service', 0.08329382),
  ('amazon', 0.08104251),
  ('lenovo', 0.043616433),
  ('please', 0.040068895),
  ('customer', 0.03503196),
  ('return', 0.03104863),
  ('buy', 0.026287062),
  ('box', 0.023697237),
  ('center', 0.023579126)])
```

In [76]:
```
top_terms=[]

top_id=[top for top,tl in topics]
terms =[tl for top,tl in topics]
#topic_df.insert(1,'Issues',issues)
for term in terms:
    top_terms.append([t for t,s in term])
print(top_terms)
```

```
[['battery', 'backup', 'hour', 'charger', 'life', 'charge', 'day', 'drain', 'turbo', 'usage'], ['day', 'service', 'amazon', 'lenov
'box', 'center'], ['screen', 'lenovo', 'sound', 'processor', 'experience', 'thing', 'work', 'glass', 'ram', 'dolby'], ['camera',
'mark', 'picture', 'clarity', 'front', 'look'], ['money', 'heat', 'display', 'waste', 'everything', 'value', 'worth', 'super', 'c
te', 'software', 'phone', 'sim', 'data', 'cast', 'jio', 'star'], ['note', 'hai', 'video', 'headphone', 'lenovo', 'ho', 'mi', 'way
'system', 'set', 'awesome', 'love', 'offer', 'friend', 'future', 'buying'], ['problem', 'heating', 'device', 'handset', 'expectati
y', 'moto'], ['speaker', 'music', 'mode', 'game', 'app', 'feature', 'android', 'photo', 'apps', 'stock']]
```

In [79]:
```
topic_names=['battery','amazon','latest features','performance','value','basic features','other','promotions','problems','apps']
topics = pd.DataFrame({'topic ID': top_id, 'topic': topic_names, 'top terms':top_terms})
topics.style.hide_index()
```

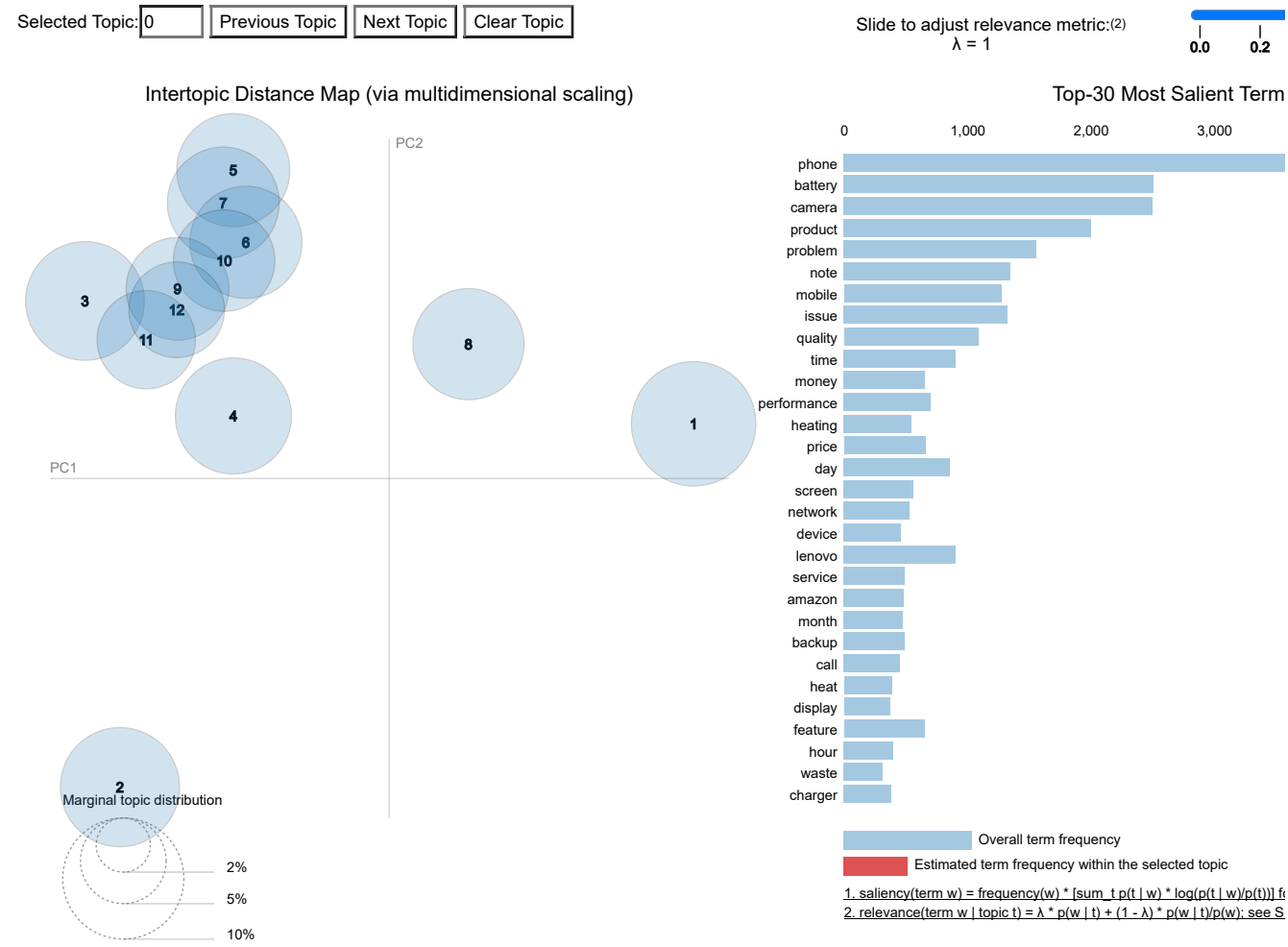
Out[79]:

topic ID	topic	top terms
1	battery	['battery', 'backup', 'hour', 'charger', 'life', 'charge', 'day', 'drain', 'turbo', 'usage']
2	amazon	['day', 'service', 'amazon', 'lenovo', 'please', 'customer', 'return', 'buy', 'box', 'center']
3	latest features	['screen', 'lenovo', 'sound', 'processor', 'experience', 'thing', 'work', 'glass', 'ram', 'dolby']
4	performance	['camera', 'quality', 'performance', 'speed', 'superb', 'mark', 'picture', 'clarity', 'front', 'look']
6	value	['money', 'heat', 'display', 'waste', 'everything', 'value', 'worth', 'super', 'condition', 'item']
7	basic features	['call', 'option', 'update', 'software', 'phone', 'sim', 'data', 'cast', 'jio', 'star']
8	other	['note', 'hai', 'video', 'headphone', 'lenovo', 'ho', 'mi', 'way', 'compare', 'hi']
9	promotions	['product', 'mobile', 'system', 'set', 'awesome', 'love', 'offer', 'friend', 'future', 'buying']
10	problems	['problem', 'heating', 'device', 'handset', 'expectation', 'review', 'signal', 'ok', 'connectivity', 'moto']
11	apps	['speaker', 'music', 'mode', 'game', 'app', 'feature', 'android', 'photo', 'apps', 'stock']

table with the topic name and the top 10 terms

In [80]:
```
lda_viz
```

Out[80]:



In [ ]:

In [ ]: