Product Recommendation and Analysis

By
Kolapalli Venkata Murali Krishna - 20186027
Subham Sahu - 20186051

Project Submitted to
MSIT, IIIT
HYDERABAD

Approved by **Rehana Shahi** (Assistant Mentor, MSIT)

Acknowledgments:

We would like to specifically thank the following people:

Rehana Shahi(Assistant Mentor): For being a brilliant project guide, that has provided constant support and reassurance throughout the entire project.

Rajesh Kumar Kakumanu (Assistant Mentor) : For being available all the time and regularly updating the status of the project by encouraging us to do better throughout the project.

All the knowledgeable contributors: We would like to acknowledge the following resources which helped us to get a better understanding of the project and give us a proper direction in developing the project. The online web resources are Coursera, Google ,machinelearningplus.com, towardsdatascience.com and peers.

TABLE OF CONTENTS

ABSTRACT

With the development of e-commerce, shopping online is becoming more popular. The convenience of new web technologies enables us to freely express our opinions and reviews for various products purchased online. Therefore, consumer reviews, opinions and shared experiences in the use of the product is a powerful source of information about consumer preferences that can be used in recommending products based on the reviews. Despite the importance and value of such information, there is no comprehensive mechanism that formalizes the opinions by selecting and retrieving the opinions and then displaying the key strengths and weaknesses of a product. In this project, we propose a system where the product reviews from e-commerce platforms are taken into consideration and are then analyzed and topic mining is performed on these customer reviews. The reviews are then categorized into positive and negative for a product which helps the manufacturer and the seller. There might be a few aspects of a product where the feature might be good and the same can be said for the negative aspects as well. Hence, topic mining helps in categorizing a product into a positive and a negative one which helps the manufacturer to understand the key strengths of a specific product and in turn helps the e-commerce platform to understand popular products that can be hosted on the website. This categorization of negative and positive reviews for a product makes it easy for a buyer to narrow down their choices while browsing for a product from a specific seller.

1.Introduction:

The development of e-commerce, shopping online is becoming more popular and consumers are freely expressing their opinions on a particular product. These consumer reviews can be put to better use by analyzing the text reviews and categorizing the specific features of a product based on their likeness.

2. Goals of the project:

In this project, we propose a system where the product reviews from e-commerce platforms are taken into consideration and are then analyzed and topic mining is performed on these customer reviews. The reviews are then categorized into positive and negative for a product which helps the manufacturer and the seller as well as the customer too by knowing which features of a product are much talked about.

3. Functionality:

The fundamental step of the project is to first take two sets of data, one which is the supervised learning on which the sentimental learning is being performed. The second one is an unsupervised learning which builds the generative models to develop the topics. Based on these models we can visualize which features of a product are much talked about.

4. Constraints:

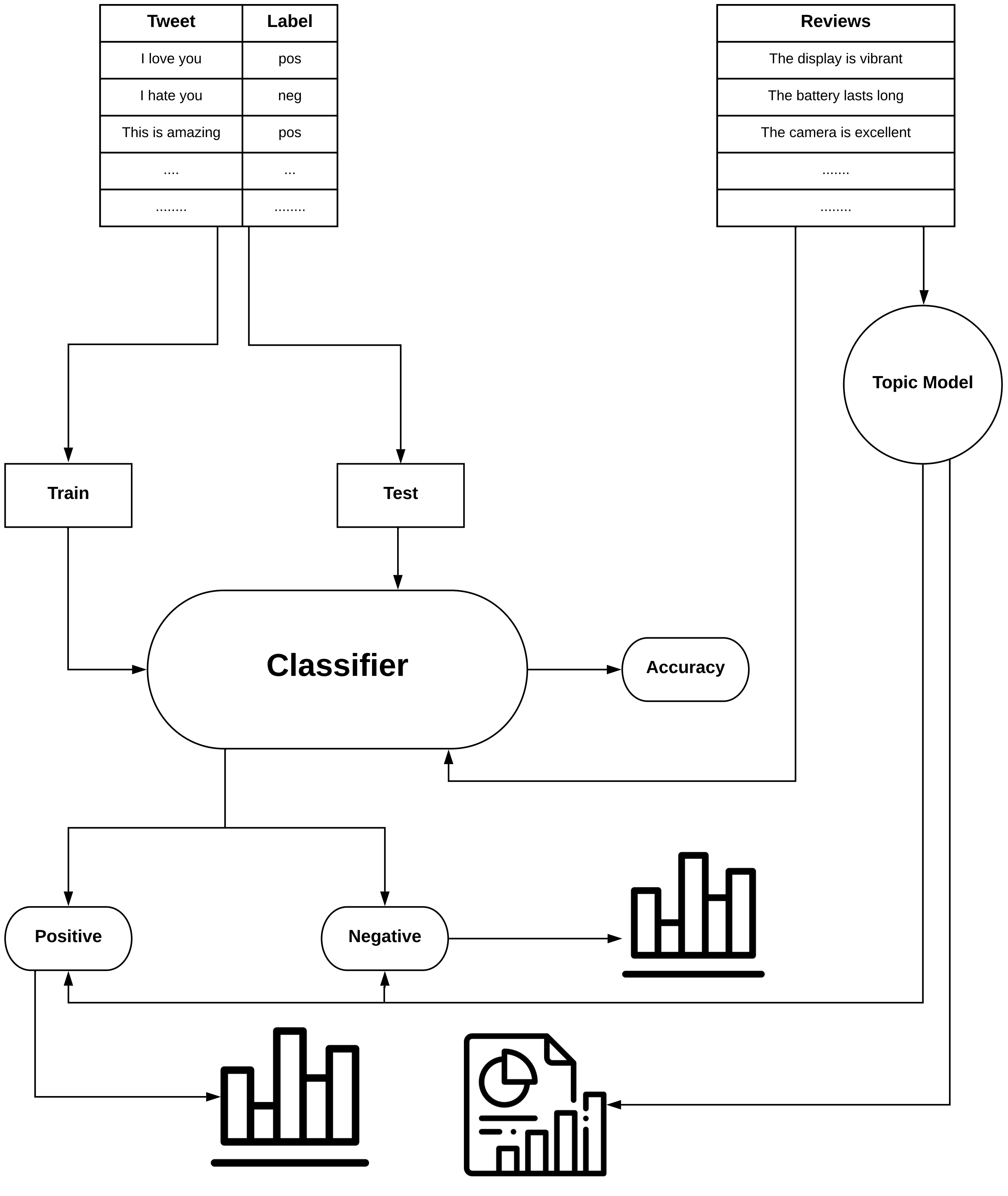
Extracting the data for building classifier and also sample sets from the original datasets was an initial challenge. Building a model with a lesser coherence value was hard to achieve in the first few trials.

5. Overview of the Project :

The project is specifically developed by keeping the consumer, manufacturer and seller in mind. The system takes the text reviews of the consumer and then analyzes the text by applying LDA (Latent Dirichlet allocation) and Naive Bayes. Several packages like numpy, pandas, matplotlib and gensim framework have been implemented in order to achieve the desired result. Finally, the text is been analyzed, processed and is then categorized into positive and negative aspects for a particular feature of a product.

Product Recommendation and Analysis

using Text Classification and Text Mining



UML Diagram

6. Coding and implementation:

The packages used during the development of the project are :

Packages used :
numpy
pandas
gensim
matplotlib
NLTK
Word cloud
pickle
IPython
Textblob from NLTK

Tools used :
Jupyter Notebook

Technologies used :
Python and R

7. Outputs and ScreenShots.

Importing the required packages.

In [1]:
```
import pandas as pd
import numpy as np
from textblob import TextBlob
from textblob.classifiers import NaiveBayesClassifier
import pickle

import ssl
try:
        _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
        pass
else:
        ssl._create_default_https_context = _create_unverified_https_context

import nltk
nltk.download('punkt')
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.stem.porter import *

import gensim
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS
from gensim import models
from gensim.models.coherencemodel import CoherenceModel

#Representing wordcloud.
from wordcloud import WordCloud, STOPWORDS

# Plotting tools
import pyLDAvis
import pyLDAvis.gensim  # don't skip this
import matplotlib.pyplot as plt
%matplotlib inline
from IPython import get_ipython

#Ignoring warnings.
import warnings
warnings.filterwarnings('ignore')
warnings.filterwarnings('ignore',category = DeprecationWarning)
np.random.seed(2019)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/Muralikrishna/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/Muralikrishna/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

List for dividing Positive and Negative sentences.

In [3]:
```
positive_sen = []
negative_sen = []
neutral_sen = []
```

Building the classifier using NAIVEBAYES from TWITTER data.

In [4]:
```
with open('train.json', 'r') as fp:
  cl = NaiveBayesClassifier(fp, format="json")
print("Sucessfully built the classifier ....")
```

```
Sucessfully built the classifier ....
```

Accuracy of the Classifier.

```
In [9]: print("calculating the accuracy of classifier...")
        with open('test.json', 'r') as test:
          print("classifier accuracy:")
          print(cl.accuracy(test,format="json"))
```

```
calculating the accuracy of classifier...
classifier accuracy:
0.7008
```

Saving the Classifier using PICKLE.

```
In [10]: sentiment_classifier = open("classifier.pickle","wb")
         print("Generating pickle file....")

         print("Dumping pickle....")
         pickle.dump(cl, sentiment_classifier)

         sentiment_classifier.close()
         print("Pickle file created.....")
```

```
Generating pickle file....
Dumping pickle....
Pickle file created.....
```

Building the classifier using PICKLE file.

```
In [11]: with open("classifier.pickle", "rb") as classifier_f:
                 cl = pickle.load(classifier_f)
         classifier_f.close()
         print("output for sample doc:")
         print(cl.classify("i think he doesnt like me, but there is a chance that he can love me"))
```

```
output for sample doc:
neg
```

"Classify" function will CATEGORIZE the Positive and Negative text.

```
In [18]: def classify(review):
                 blob = TextBlob(review, classifier=cl)
                 blob.lower()
                 blob.correct()
                 for sentence in blob.sentences:
                         if sentence.classify() == "neg" and len(str(sentence)) > 3:
                                 negative_sen.append(str(sentence))
                         elif sentence.classify() == "pos" and len(str(sentence)) > 3:
                                 positive_sen.append(str(sentence))
                         else:
                                 if len(str(sentence)) > 3:
                                         neutral_sen.append(str(sentence))
```

Loading Reviews from topic.csv (reviews file).

```
In [2]: print("loading reviews")
        data = pd.read_csv('topic.csv', error_bad_lines=False);
        data_text = data[['Reviews']]
        data_text['index'] = data_text.index
        documents = data_text
        total_docs = len(documents)
        print("Number of Reviews: " + str(total_docs))
```

```
loading reviews
Number of Reviews: 604
```

NOTE: *We will consider each review as each document and will build the topic model*.

CLASSIFYING reviews with respect to sentences.

In [19]:
```
print("classifying reviews...")
documents['Reviews'].map(classify)
print("reviews classified....")
```

```
classifying reviews...
reviews classified....
```

STORING positive text in positive.txt file.

In [21]:
```
print("Number of positive sentences: "+str(len(positive_sen)))
pos_str = ("").join(positive_sen)
file1 = open("positive.txt","w")
file1.writelines(pos_str)
file1.close()
print("positive text file created....")
```

```
Number of positive sentences: 984
positive text file created....
```

STORING Negative text in Negative.txt file.

In [22]:
```
print("Number of negative sentences: "+str(len(negative_sen)))
neg_str = ("").join(negative_sen)
file2 = open("negative.txt","w")
file2.writelines(neg_str)
file2.close()
print("negative text file created....")
```

```
Number of negative sentences: 1481
negative text file created....
```

STORING Neutral text in Neutral.txt file.

In [23]:
```
print("Number of neutral sentences: "+str(len(neutral_sen)))
neu_str = ("").join(neutral_sen)
file2 = open("neutral.txt","w")
file2.writelines(neu_str)
file2.close()
print("neutral text file created....")
```

```
Number of neutral sentences: 0
neutral text file created....
```

"get_classfied_files" function READS the file and returns whole text as ONE STRING.

In [3]:
```
def get_classfied_files(filename):
    unseen_document = ""
    for line in open(filename, 'r'):
        unseen_document = unseen_document+""+line
    return unseen_document
```

LOADING Positive and Negative text.

In [4]:
```
pos_str = get_classfied_files("positive.txt")
neg_str = get_classfied_files("negative.txt")
print("loaded files successfully.. ")
```

```
loaded files successfully..
```

"preprocess" function, removes STOPWORDS, PUNCTUATION, STEMMING the words, LEMMATIZING the words less than 3 (Length of the word) from the reviews.|

In [5]:
```
def preprocess(text):
        result = []
        for token in gensim.utils.simple_preprocess(text):
                if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) > 3:
                        result.append(lemmatize_stemming(token))
        return result
```

LEMMATIZING and STEMMING the word

In [6]:
```
def lemmatize_stemming(text):
        stemmer = PorterStemmer()
        return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='a'))
```

PREPROCESSING all reviews.

In [7]:
```
processed_docs = documents['Reviews'].map(preprocess)
processed_docs[0:5]
```

Out[7]:
```
0               [like, phone, unfortun, tmobil, return]
1                   [love, fast, slim, sleek, slipperi]
2    [product, defect, plug, charg, thought, gonna,...
3                [happi, cell, phone, beauti, friendli]
4    [phone, month, decid, complet, quit, work, mid...
Name: Reviews, dtype: object
```

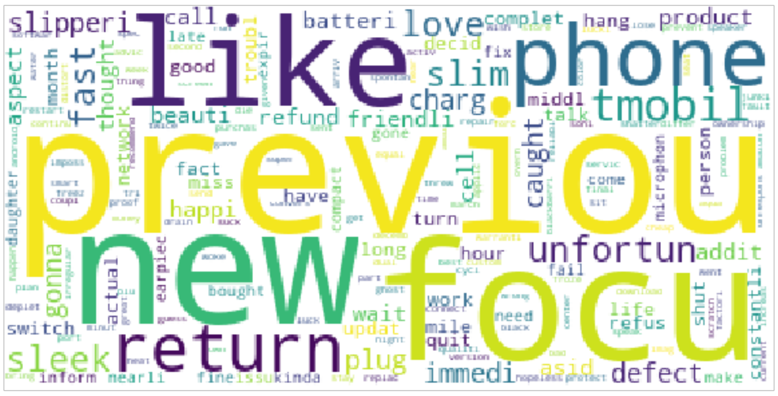
Creating dictionary from PREPROCESSED documents.

In [52]:
```
dictionary = gensim.corpora.Dictionary(processed_docs)
print("Unique words are generated from the Preprocess Documents..")
print("Number of unique words: "+str(len(dictionary)))
```

```
Unique words are generated from the Preprocess Documents..
Number of unique words: 1744
```

Creating the WORDCLOUD to remove unnecessary words.

In [34]:
```
words = ' '
stopwords =set(STOPWORDS)
stopwords = []
for k,v in dictionary.iteritems():
        words = words + v + ' '
wordcloud = WordCloud(width = 400, height = 200,background_color ='white',stopwords = stopwords,min_font_size = 5).gener
        #plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



LISTING the word ID's of unnecessary words.

In [54]:
```
del_list = ["galaxi","iphon","samsung","soni","xperia","month","love","time","problem","like","good","great"]
del_ids = [k for k,w in dictionary.items() if w in del_list]
print("Number of unique words after ID filtering: "+str(len(dictionary)))
```

```
Number of unique words after ID filtering: 9
```

FILTERING the dictionary using ID's as well as coverage of words in the documents (REVIEWS).

In [55]:
```
dictionary.filter_tokens(bad_ids=del_ids)
dictionary.filter_extremes(no_below=70, no_above=0.2, keep_n=1700)
print("Final  of unique words: "+str(len(dictionary)))
print()
print("Words after filtering:")
print()
for k, v in dictionary.iteritems():
        print(k, v)
print()
```

```
Final  of unique words: 4

Words after filtering:

0 batteri
1 screen
2 camera
3 price
```

NOTE: Word should cover minimum of 70 docs (reviews) and should not be above 340(0.2*1700) docs.

Creating the (bagofwords) bow_corpus.

In [56]:
```
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
print(bow_corpus[0:20])
```

```
[[], [], [], [], [(0, 2)], [(1, 2)], [], [], [(0, 1), (1, 2)], [], [], [], [], [(0, 1)], [(1, 2)], [], [], [(0, 1), (1,
```

NOTE: Converting words into numbers based on id's. [review] -> [(word1, word1_frequency), (word2, word2_frequency) ...]

Building the LDA model using bow_corpus

In [57]:
```
lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=4, id2word=dictionary, passes=2, workers=3)
print("bag of words is built...")
```

```
bag of words is built...
```

Printing the BOW_MODEL topics.

In [58]:
```
print()
for idx, topic in lda_model.print_topics(-1):
        print('Topic {} : {}'.format(idx+1, topic))
        print()
```

```
Topic 1 : 0.692*"price" + 0.140*"screen" + 0.117*"camera" + 0.051*"batteri"

Topic 2 : 0.627*"camera" + 0.212*"screen" + 0.137*"batteri" + 0.025*"price"

Topic 3 : 0.554*"batteri" + 0.353*"screen" + 0.081*"price" + 0.011*"camera"

Topic 4 : 0.547*"price" + 0.262*"camera" + 0.118*"batteri" + 0.072*"screen"
```

Coherence Value of the BOW_MODEL (Indicates the distance between topics).

In [59]:
```
coherence_model_lda = CoherenceModel(model=lda_model, texts=processed_docs, dictionary=dictionary, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score_lda: ', coherence_lda)
print()
```

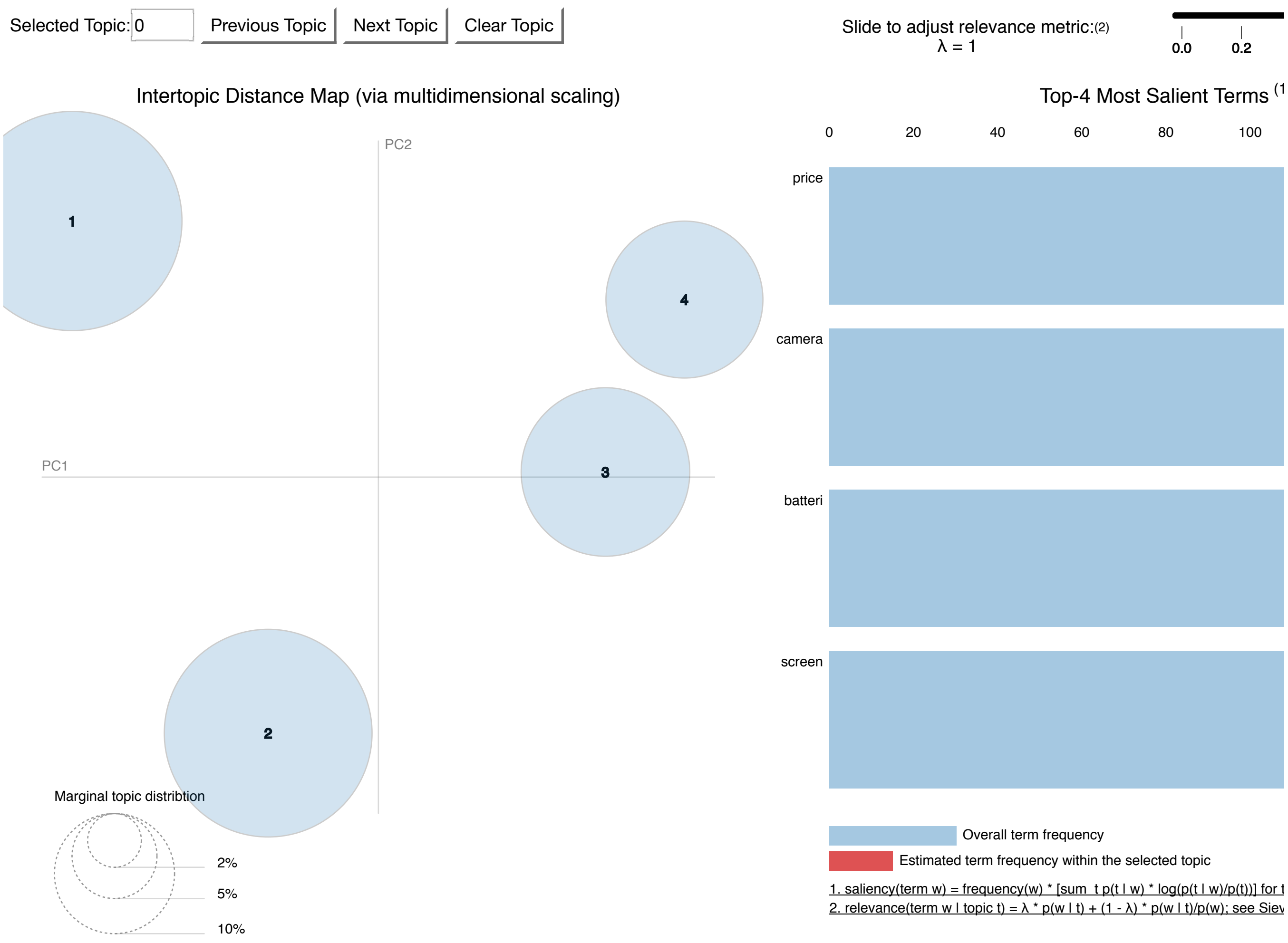
```
Coherence Score_lda:  0.5876214365561421
```

Printing the Visualisation of Bag of Word Topics.

In [60]:
```
pyLDAvis.enable_notebook()
visualisation = pyLDAvis.gensim.prepare(lda_model, bow_corpus, dictionary)
```

In [61]:
```
visualisation
```

Out[61]:



Creating the TF-IDF corpus.

In [62]:
```
tfidf = models.TfidfModel(bow_corpus)
corpus_tfidf = tfidf[bow_corpus]
```

NOTE: Converting words into numbers based on id's. [review] -> [(word1, word1_frequency*word_weight), (word2, word2_freque
word_weight = log((1+M) / K) || M - coverage of word in all documents, K - total number of documents.

Building the LDA model using TF-IDF_corpus.

In [65]:
```
lda_model_tfidf = gensim.models.LdaMulticore(corpus_tfidf, num_topics=4, id2word=dictionary, passes=2, workers=3)
print("TF-IDF is built...")
```

```
TF-IDF is built...
```

Printing the TF-IDF MODEL Topics.

In [66]:
```
print()
for idx, topic in lda_model_tfidf.print_topics(-1):
        print('Topic {} : {}'.format(idx+1, topic))
        print()
```

```
Topic 1 : 0.875*"price" + 0.061*"batteri" + 0.042*"camera" + 0.023*"screen"

Topic 2 : 0.873*"camera" + 0.096*"price" + 0.017*"screen" + 0.015*"batteri"

Topic 3 : 0.731*"batteri" + 0.179*"camera" + 0.063*"screen" + 0.026*"price"

Topic 4 : 0.697*"screen" + 0.142*"price" + 0.130*"camera" + 0.031*"batteri"
```

Coherence Value of the TF-IDF MODEL (Indicates the distance between topics).

In [67]:
```
coherence_model_lda = CoherenceModel(model=lda_model_tfidf, texts=processed_docs, dictionary=dictionary, coherence='c_v'
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score_tdif: ', coherence_lda)
print()
```

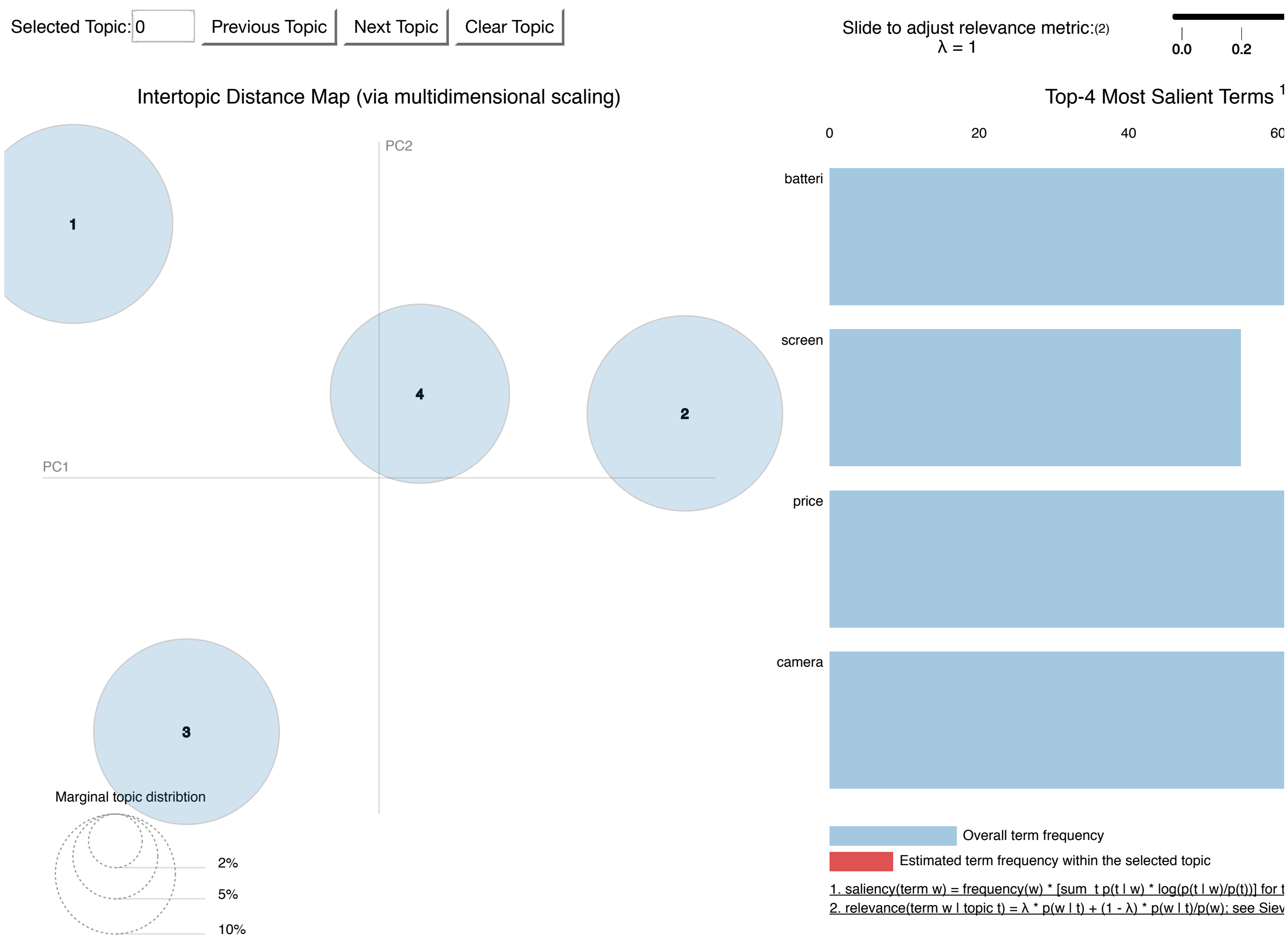
```
Coherence Score_tdif:  0.5876214365561422
```

Printing the Visualisation of TF-IDF Topics.

In [68]:
```
pyLDAvis.enable_notebook()
visualisation = pyLDAvis.gensim.prepare(lda_model_tfidf, corpus_tfidf, dictionary)
```

In [69]:
```
visualisation
```

Out[69]:



Coverage of Topics in Positive text.

In [82]:

```
no_pos = 984
no_neg = 1481
total = no_pos+no_neg
p_reviews = total_docs*(no_pos/total)
n_reviews = total_docs*(no_neg/total)
print("Number of positive reviews :" +str(round(p_reviews)))
print("Number of negative reviews :" +str(round(n_reviews)))
```

```
Number of positive reviews :241
Number of negative reviews :363
```

In [101]:

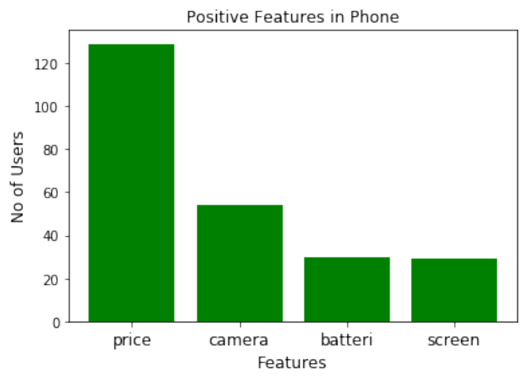
```
labels_p= []
scores_p= []
labels_n= []
scores_n= []
```

In [102]:
```
bow_vector = dictionary.doc2bow(preprocess(pos_str))
for index, score in sorted(lda_model_tfidf[bow_vector], key=lambda tup: -1*tup[1]):
    #print("Score: {}\t Topic: {}".format(round(score*100), lda_model_tfidf.print_topic(index, 4)))
    a = dictionary.get(lda_model_tfidf.get_topic_terms(index, 4)[0][0])
    b = round(score*p_reviews)
    print("Topic: {}\t Score: {}".format(a,b))
    labels_p.append(a)
    scores_p.append(b)
print()
```

```
Topic: price     Score: 129.0
Topic: camera    Score: 54.0
Topic: batteri   Score: 30.0
Topic: screen    Score: 29.0
```

In [103]:
```
index = np.arange(len(labels_p))
def plot_bar_p():
    # this is for plotting purpose
    index = np.arange(len(labels_p))
    plt.bar(index, scores_p,color="green")
    plt.xlabel('Features', fontsize=12)
    plt.ylabel('No of Users', fontsize=12)
    plt.xticks(index, labels_p, fontsize=12, rotation=0)
    plt.title('Positive Features in Phone')
    plt.show()
```

In [104]:
```
plot_bar_p()
```



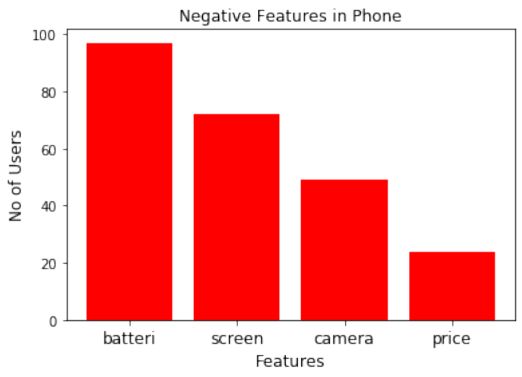
Coverage of Topics in Negative text.

In [105]:
```
bow_vector = dictionary.doc2bow(preprocess(neg_str))
for index, score in sorted(lda_model_tfidf[bow_vector], key=lambda tup: -1*tup[1]):
    #print("Score: {}\t Topic: {}".format(round(score*100), lda_model_tfidf.print_topic(index, 4)))
    a = dictionary.get(lda_model_tfidf.get_topic_terms(index, 4)[0][0])
    b = round(score*p_reviews)
    print("Topic: {}\t Score: {}".format(a,b))
    labels_n.append(a)
    scores_n.append(b)
print()
```

```
Topic: batteri   Score: 97.0
Topic: screen    Score: 72.0
Topic: camera    Score: 49.0
Topic: price     Score: 24.0
```

In [110]:
```
index = np.arange(len(labels_n))
def plot_bar_n():
    # this is for plotting purpose
    index = np.arange(len(labels_n))
    plt.bar(index, scores_n,color="red")
    plt.xlabel('Features', fontsize=12)
    plt.ylabel('No of Users', fontsize=12)
    plt.xticks(index, labels_n, fontsize=12, rotation=0)
    plt.title('Negative Features in Phone')
    plt.show()
```

```
In [111]: plot_bar_n()
```



_END__

8.Conclusion

The aim of the project was met successfully by reading the input data sets and then training the data by both means of supervised and unsupervised learning. The required models were then generated and reviews were categorized into positive and negative aspects. Finally, visualization of the text was achieved by displaying the likeness of specific features of a product.

9. References :

- https://www.coursera.org/learn/python-text-mining
- https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24
-
- https://towardsdatascience.com/topic-mining-on-amazon-reviews-ae76fc286c61
-
- https://datascienceplus.com/evaluation-of-topic-modeling-topic-coherence/
-
- https://textblob.readthedocs.io/en/dev/quickstart.html
-
- https://www.machinelearningplus.com/nlp/topic-modeling-visualization-how-to-present-results-lda-models/