Structuring Customer Complaints

By Yogesh H. Kulkarni

# Introduction

In past, if you were not particularly happy with a service or a product, you would go to the service provider or the shop and lodge a complaint. With services-businesses going online and due to enormous scale, lodging complaints in-person may not be always possible. Electronic ways such as emails, social media and particularly websites like [www.consumercomplaints.in](http://www.consumercomplaints.in) focusing on such issues, are widely used platforms to vent out the anger as well as publicising the issue in expectance of quick actions. Keeping a close watch on complaints on such sites has become imperative for businesses such as banks. This article looks at ways to structure these unstructured complaints in actionable form.

In a typical case, a bank may be interested in looking at classification of the complaints into various categories such as “Loans”, “Fixed Deposits”, “Credit Cards”, etc. so that they can be forwarded to respective departments. An important feature would be to summarize long complaints so that further actions can be formulated quickly. Sentiment analysis of such complaints is typically not very useful as most of them would be highly negative anyways. This article proposes a way to classify and summarise customer complaints seen on websites like http://www.consumercomplaints.in/.

# Proposed Method

The Natural Language Programming pipeline is utilized to structure the text with stages such as:

* **Input**: Acquisition of text-corpus.
* **Core Process**: Extraction of features (including summarization).
* **Output**: Generation of Insights, Actionable tasks.

Following sections describe all these stages, with more elaboration of one the core process of feature extraction, the summarization.

## Input

Various tools-libraries can be used to scrape reviews from websites. Python has libraries such as “requests”, “BeautifulSoup” to take care of these tasks. Useful tutorials can be found at [Tutorial1](https://www.analyticsvidhya.com/blog/2015/10/beginner-guide-web-scraping-beautiful-soup-python/) and [Tutorial2](http://www.anthonydebarros.com/2013/07/04/python-twitter-facebook-api-script-sqlite/). Output of this stage is a set of text files with one complaint in each. A sample complaint looks as follows (some of the text has been masked, for the sake of confidentiality):

PIN not received.

17 Reviews

<CUSTOMER USER NAME>

Hello,  
  
I have issued a new Debit/ATM card for my account no. 039309999999. This charged around Rs 240 (so unreasonable amount) for it on my account and Bank officer tells this is service charge. The card was delivered to my home however pin did not come to home.

:

:

:

BANK charges so much for such small things with pathetic service.  
Even an account statement incurs around Rs. 150 or 200 (i don't remember exactly).  
Is there anybody from BANK who can take responsibility and look into this matter ?  
  
regards,

XXXX

16 Comments

Updated: Mar 19, 2010

## Extraction of Features

Apart from core text of the complaint, useful features are :

* Subject e.g. “PIN not received”.
* Number of reviews e.g. “17 Reviews”
* Customer user name
* Account numbers e.g. “039309999999”
* Amount involved, e.g. “Rs 240”
* Number of comments e.g. “16 Comments”
* Last update e.g. “March 19, 2010”.

Subject being the first line of the complaint is easy to extract. It forms a one line gist of the issue.

Other features such as Number of reviews, comments, etc. are typically in a fixed format and can be extracted by regular expressions. Tutorials like [this](https://www.analyticsvidhya.com/blog/2015/06/regular-expression-python/) can be used to extract them.

Extraction of known categories such as “Loans”, “Fixed Deposits”, “Credit Cards” can also be done using matching pre-defined keywords by regular expressions.

Customer complaints could be very long and with such large volume, it is manually impossible to read through all of them. Effective summarization is a way of compressing the text into few meaningful lines. Following section elaborates one of the ways of summarization for customer complaints.

# Summarization

Text summaries can be Abstractive or Extractive. In Abstractive, the summary is constructed by employing words and phrases which are (typically) NOT in the original text, whereas in Extractive, few of the highly representative sentences are picked from the original text and ordered to form the summary. The proposed method is of Extractive type.

## Problem Definition

Given a document *D*, having sentences (return *Y,* which is a set of *K* important statements from *D*. Thus, Extractive text summarization is a binary classification model, where, out of *n* sentences, *K* sentences are labelled as *True* (meaning, they are part of the summary) or *False* if otherwise. So, the problem boils down the determining if a sentence () is labelled as True or *False*.

Decision of labelling depends on various factors, called as Summary features. In the overall process, for each statement, these features are computed. Their weighted sum is ranked. Top *K* ranked sentences are chosen as set *Y*, representing the summary.

## Summary Features

In the current method, following features are incorporated to arrive at the rank of a sentence:

1. **TF-ISF (Term Frequency – Inverse Sentence Frequency)**: For each word, its number of occurrences are computed, called TF. ISF denotes uniqueness of that word across all sentences. TF-ISF of a sentence is summation of TF-ISF of each word in it.

**def** tf(word, doc):  
 count = doc.count(word)  
 total = len(doc)  
 tf\_score = count / float(total)  
 **return** tf\_score  
  
**def** n\_containing(word, docs):  
 count = 0  
 **for** doc **in** docs:  
 **if** doc.count(word) > 0:  
 count += 1  
 **return** count  
  
**def** isf(word, docs):  
 doc\_count = n\_containing(word, docs)  
 ratio = len(docs) / float(1 + doc\_count )  
 **return** math.log(ratio)  
  
**def** tfidf(word, doc, docs):  
 tf\_score = tf(word, doc)  
 isf\_score = isf(word, docs)  
 **return** tf\_score \* isf\_score

**def** compute\_tfisf\_scores(sentences):  
 tfisf\_scores = []  
 **for** sent **in** sentences:  
 sentence\_score = 0  
 **for** word **in** sent:  
 sentence\_score += tfisf(word,sent,sentences)  
 sentence\_score /= float(len(sent))  
 tfisf\_scores.append(sentence\_score)  
 **return** normalize(tfisf\_scores)

1. **Length**: Number of words in the sentence can dictate importance of it. Shorter ones are less important as they may not represent the gist of the whole text.
2. **Position**: Sentences occurring initially and towards end carry more meaning than the middle ones. The first sentence is of utmost importance.
3. **Proper Nouns**: The sentences which contain Named Entity called Proper Nouns (“NNP”) are important ones as they contain names of the places, persons, etc.
4. **Cue Words**: Domain specific words such as “Undelivered”, ‘Fraud”, etc. suggest important sentences. So, sentences having more such words are given more weightage.
5. **Topic Words**: Topic words are arrived as central words of the whole text. It could be words such as “Debit”, “Loan”, etc. Sentences aligned more with them are central to the text and thus are more eligible to be part of the summary.

**def** identify\_lda\_topics(cleaned\_sentences):  
 dictionary = corpora.Dictionary(cleaned\_sentences)  
 doc\_term\_matrix = [dictionary.doc2bow(doc) **for** doc **in** cleaned\_sentences]  
 ldamodel = models.ldamodel.LdaModel(doc\_term\_matrix, num\_topics=6, id2word=dictionary, passes=5)  
 topics = ldamodel.show\_topics(num\_topics=1, formatted=**False**, num\_words=6)  
topic\_names = []  
 **for** topic **in** ldamodel.show\_topics(num\_topics=6, formatted=**False**, num\_words=6):  
 *# print("Topic {}: Words: ".format(topic[0]))* topicwords = [w **for** (w, val) **in** topic[1]]  
 topic\_names += topicwords  
 **return** list(set(topic\_names))

Each feature values are normalized to lie in range 0 to 1.

## Computation

Rank is computed for each sentence as weighted sum of the features. Values of the weights can either be derived empirically of by employing Machine/Deep Learning algorithms such as Naïve Bayes, Logistic Regression, Support Vector Machine, etc. The current method computes Rank as:

rank\_scores = weight\_TfIsf \* df[**'TfIsf'**] + \  
 weight\_Length \* df[**'Length'**] + \  
 weight\_Position \* df[**'Position'**] + \  
 weight\_ProperNouns \* df[**"ProperNouns"**] + \  
 weight\_TopicWords \* df[**'TopicWords'**] + \  
 weight\_CueWords \* df[**'CueWords'**]

A data frame is populated with the summary features as below:



The data frame is then sorted based on the “Rank” and fed for summary generation.

## Summary Generation

While collecting K top ranked sentences, care is taken that sentences “similar” to already selected sentences are not added to the set Y. This avoids getting almost duplicate sentences in the summary. Similarity measure used in the method is based on TF-ISF and similarity as shown below.

**def** compute\_similarity(sent1, sent2):  
 sentences = [sent1, sent2]  
  
 **from** sklearn.feature\_extraction.text **import** CountVectorizer  
 c = CountVectorizer()  
 bow\_matrix = c.fit\_transform(sentences)**from** sklearn.feature\_extraction.text **import** TfidfTransformer  
 normalized\_matrix = TfidfTransformer().fit\_transform(bow\_matrix)  
 similarity\_graph = normalized\_matrix \* normalized\_matrix.T  
 **return** 1 - (similarity\_graph[0,1])

And the resultant 3-line (*K* = 3) summary (*Y*) is:

PIN not received.

This charged around Rs 240 (so unreasonable amount) for it on my account and Bank officer tells this is service charge.

The card was delivered to my home however pin did not come to home.

# Post Processing

Each complaint is ready with features such as its Summary and classification category for identifying the department to which it can be forwarded. The concerned department can sort complaints based on the amount involved, number of reviews and/or comments and start addressing the issues by reading the summary. If more details are needed, then original complaints can be considered.

# Conclusion

The current article presents an automatic summarization method. It extracts features from sentences and picks top ranked sentences as the summary. Features such as ``Cue Words'' give flexibility for customization specific to the given domain. Topic words used are the centroids of the clusters of words. Sentences having such central words form gist of the original text. Overall rank of the sentence captures effect of all these features in relative importance. The proposed method can be further developed to incorporate additional features. Use of machine/deep learning algorithms can derive more accurate weights used for ranking the sentences.

Yogesh H. Kulkarni is currently pursuing full-time PhD in the field of Geometric modeling, after working in the same domain for more than 16 years. He is also keenly interested in data sciences, especially Natural Language Processing, Machine Learning and wishes to pursue further career in these fields. His full-bio is at <https://in.linkedin.com/in/yogeshkulkarni> and he can be reached at [yogeshkulkarni@yahoo.com](mailto:yogeshkulkarni@yahoo.com)