Sentiment Analysis and Classification Of written reviews using Naïve Bayes machine learning model

Submitted By

Abhinav Srivastav
(Admission No. 14je000189)
Semester: VIII

Under the Guidance of
Dr. Haider Banka
Associate Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY (ISM), DHANBAD – 826004

May 09,2018

INDIAN INSTITUTE OF TECHNOLOGY(ISM), DHANBAD

(Declared as Deemed-to-be-University U/S 3 of the UGC Act, 1956
Vide Notification No. F11-4/67-U3, dated 18.9.1967 of Govt. of India)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled "**Sentiment Analysis and Classification Of written reviews using Naïve Bayes machine learning model**" is a record of project successfully carried out by **Abhinav Srivastav**, Admission No. 14je000189 , under the guidance of Dr. Haider Banka.

This report is submitted as an authentic record and as a requirement for the award of degree of **Bachelor of Technology (B.Tech) in Computer Science and Engineering** during the academic session 2017-18 **to Indian Institute of technology (Indian School of Mines), Dhanbad.**

The result embodied in this report has not been submitted for award of any other degree to any other university or institute.

Prof. P.K Jana

Professor and Head
Department of Computer-
Science and Engineering
Indian Institute of Technology
(ISM) Dhanbad

Dr. Haider Banka

Associate Professor
Department of Computer-
Science and Engineering
Indian Institute of Technology
(ISM) Dhanbad

Acknowledgement

I express my deep sense of gratitude and respect towards my project guide, Dr. Haider Banka, Assosiate Professor, Department of Computer Science and Engineering, Indian Institute of Technology(ISM) Dhanbad. I am very grateful for the generosity, expertise and guidance I have received from him while working on the project. I have been amazingly fortunate to have an advisor who gave me the freedom to explore on my own and at the same time the provide guidance to recover when my steps faltered. Expressing my sincere thanks to my mentor I would like to convey my sincere regards and gratitude for encouraging me to undertake this project and to work on it at time and again, it would not have been possible without him.

Abhinav Srivastav

14je000189

B.Tech, Computer Science and Engineering

Indian Institute and Engineering (ISM) Dhanbad

Abstract

An important part of our information-gathering behaviour has always been to figure out what other people think. With the growing availability and popularity of opinion-rich resources such as online review sites and personal blogs, new opportunities and challenges arise as people now can, and do, actively use information technologies to seek out and understand the opinions of others. The sudden eruption of activity in the area of opinion mining and sentiment analysis, which deals with the computational treatment of opinion, sentiment, and subjectivity in text, has thus occurred at least in part as a direct response to the surge of interest in new systems that deal directly with opinions as a first-class object.

Generally, there is a great deal of information that can be extracted from the attitude of a speaker, writer, or other subject with respect to some topic or the overall contextual polarity or emotional reaction to a document, interaction, or event. The attitude may be a judgment or evaluation, affective state, or the intended emotional communication (that is to say, the emotional effect intended by the author or interlocutor). With the growing area of net connectivity, there are overflowing emotions present in form of blogs, articles etc. There is also a need to identify the antisocial elements and restrict their flow in the society.

This report presents a Sentiment Analysis model to analyse movie reviews rating them as positive and negative. Combining Machine Learning and Baseline algorithm of NLP we can actually read people's mind , though at a very small scale .

Index

Contents

1 Introduction

Sentiment analysis (sometimes known as opinion mining or emotion AI) refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine. Sentiment analysis deals with people's opinion, sentiments, evaluations, appraisals, attitudes and emotions towards the entities such as products, services, organisations, topics etc.

Although linguistics and NLP have a very long history, little research had been done about people's opinion and sentiments before 2000. Since then the field has become a wide area for research. There are several reasons for this, First, it had wide range of applications, Second proliferation of commercial applications. This provides a strong motivation for research. Third reason is the internet boom, which enables us to get large amount of the data regarding opinions of people. Then we have the social media which coincide with the inception and rapid growth of sentiments. The origin of sentiment analysis can be traced to the 1950s, when sentiment analysis was primarily used on written paper documents. Today, however, sentiment analysis is widely used to mine subjective information from content on the Internet, including texts, tweets, blogs, social media, news articles, reviews, and comments. This is done using a variety of die rent techniques, including NLP, statistics, and machine learning methods. Organizations then use the information mined to identify new opportunities and better target their message toward their target demographics. The White House Administration even uses sentiment analysis to predict public response to its policy announcements.

In the last decade, sentiment analysis (SA), also known as opinion mining, has attracted an increasing interest. It is a hard challenge for language technologies, and achieving good results is much more difficult than some people think. The task of automatically classifying a text written in a natural language into a positive or negative feeling, opinion or subjectivity (Pang and Lee, 2008), is sometimes so complicated that even different human annotators disagree on the classification to be assigned to a given text. Personal interpretation by an individual is different from others, and this is also affected by cultural factors and each person's experience. And the shorter the text, and the worse written, the more difficult the task becomes, as in the case of messages on social networks like Twitter or Facebook.

2 Approach

Semantic analysis

Semantic approaches are characterized by the use of dictionaries of words (lexicons) with semantic orientation of polarity or opinion. Systems typically pre-process the text and divide it into words, with proper removal of stop words and a linguistic normalization with stemming or lemmatization, and then check the presence or absence of each term of the lexicon, using the sum of the polarity values of the terms for assigning the global polarity value of the text. Typically, systems also include i) a more or less advanced treatment of modifier terms (such as very, too, little) that increase or decrease the polarity of the accompanying terms; and ii) inversion terms or negations (such as no, never), which reverse the polarity of the terms to which they affect. Moreover, the learning-based approaches consist on training a classifier using any supervised learning algorithm from a collection of annotated texts, where each text is usually represented by a vector of words (bag of words), n-grams or skip-grams, in combination with other types of semantic features that attempt to model the syntactic structure of sentences, intensification, negation, subjectivity or irony. Systems use different techniques, but the most popular are classifiers based on SVM (Support Vector Machines), Naive Bayes and KNN (K-Nearest Neighbour). More advanced techniques appear in the most recent investigations, such as LSA (Latent Semantic Analysis) and Deep Learning.

First the pre-processing is done

1. The text is first segmented into words
2. Then the words lowered
3. Stemming is done to create the bag of words
4. The classification is done.
5. Naïve Bayes classifier is used

This approach in which we use a dictionary to train the classifier is called the semantic approach. The approach employs major part of the information retrieval concepts.

Having distinguished whether a sentence is a fact or opinion, we separate positive, negative into two classes. We base this decision on the number and strength of semantically oriented words (either positive or negative) in the sentence. We first discuss how such words are automatically found by our system, and then describe the method by which we aggregate this information across the sentence.

To determine which words are semantically oriented, in what direction, and the strength of their orientation, co-occurrence with words from a known seed set of semantically oriented words. The approach is based on the hypothesis that positive words co-occur more than expected by chance, and so do negative words; this hypothesis was validated, at least for strong positive/negative words. 1, 20, 100 and over 600 positive and negative pairs of adjectives. For a given seed set size, we denote the set of positive seeds as ADJ p and the set of negative seeds as ADJ n. calculate a modified log-likelihood ratio L(Wi, POSj) for a Wi with part of speech POSj (j can be adjective, adverb, noun or verb) as the ratio of its collocation frequency with ADJp and ADJ n within sentence, Where Freq(Wall, POSj, ADJp) represents the collocation frequency of all words Wall of part of speech POSj with ADJp and is a smoothing constant.

Feature Engineering

In order to perform machine learning, it is necessary to extract certain clues from the text that may lead to an effective correct classification. Clues about the original data are usually stored in the form of a feature vector, $F=(f1,f2.....fn)$. Each coordinates of a feature vector represents one clue, also called a feature, "fi" of the original text. On setting out to classify a document, we generally start with depicting a very large number of words that need to be considered, even though very few of the words in the corpus are actually expressing sentiment. These extra features have two clear drawbacks that need to be eliminated. The first is that they show down the process of document classification since there are far more words than needed. The second is that they can actually reduce accuracy since the classifier is obliged to consider these words when classifying a document. As the name suggests, feature selection is a process through which we run across the corpus before the classifier has been trained and remove any features that seem unnecessary. This allows the classifier to fit a model to the problem more quickly as there will be less information to consider, thus allowing it to classify items faster.

Part of Speech Features

Parts of Speech information is most commonly exploited in all NLP tasks. One of the most important reasons is that they provide a crude form of word sense disambiguation. Since the language used in Twitter is generally informal, part of speech tagging isn't very accurate for tweets. We used both NLTK Part of Speech tagger and Open NLP Part Of Speech tagger along with a heuristic that adjectives and/or adverbs, JJ, JJR, JJS, RB, RBR and RBS in the Penn Tree bank target, are generally used to articulate opinions in natural language. So, we further process the data by excluding all data except the entity, adjectives, adverbs and words such as not, couldn't etc which generally indicate a reversal of sentiment. After using part of speech taggers, experimented with unigrams, bigrams and combination of unigrams and bigrams.

N-Gram Features

N-grams are capable of capturing context to some extent and are widely used in Natural Language Processing tasks. Whether higher order n-grams are useful is a matter of debate. The processed data is used to extract features that will be used to train our classifier. We have experimented with unigrams, bigrams, trigrams and combination of unigrams and bigrams. The data was tokenized by spaces using NLTK and these tokens were subject to NLTK to generate n-grams.

Unigram
For text classification purpose, the unigram model was used which selects individual words from the data. Apple is opening up the iPhone SDK. Imstoked!, for instance, contains following unigrams: Apple, opening, iPhone, SDK, stoked ,etc.

Bigram
In bigram model, a pair of words is extracted from data. The tweet, Apple is opening up the iPhone SDK. Imstoked!, for instance, contains bigrams like: (Apple, opening), (opening, iPhone) etc .

Unigram + Bigram
In this model, a combination of unigram as well as bigram model is used to extract words from the data.

Model Building

Training
For training purpose, the polarity labelled data from corpus is first parsed and relevant features are extracted from it to build the feature vector. This vector is used to create a Feature List which is a list of all the features of all the data items in dataset used for training, this list is stored in a text file on secondary memory for further use in both the training and classification.

Naive Bayesian Classifier
Naive Bayesian Text Classification algorithm is used for the purpose of classification of given trained model. It is the probabilistic approach to the text classification. Here the class labels are known and the goal is to create probabilistic models, which can be used to classify new texts. It is specifically formulated for text and makes use of text specific characteristics. The Naïve Bayesian classifier treats each document as a bag of words and the generative model makes the following assumptions: firstly, words of a document are generated independently of context, and, secondly, the probability of the word is independent of its position. This is why the name naive was used for this algorithm. In real text documents the words often correlate with each other and the position of the word in text may play role.

Naive Bayes has been studied extensively since the 1950s. It was introduced under a different name into the text retrieval community in the early 1960s, and remains a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (such as spam or legitimate, sports or politics, etc.) with word frequencies as the features. With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines. It also finds application in automatic medical diagnosis.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

In the statistics and computer science literature, naive Bayes models are known under a variety of names, including simple Bayes and independence Bayes. All these names reference the use of Bayes' theorem in the classifier's decision rule, but naive Bayes is not (necessarily) a Bayesian method.

Classification
For classification purpose, the test data is pre-processed and feature vector of test data is formed. This test data is then fed into Naive Bayes algorithm along with the training data to calculate the probability using the Naïve Bayes conditional probability formula to get polarity of the highest probability.

3 Algorithm

1. **Pre-Processing**
2. **Training**
3. **Testing**

Pre-Processing

1. Collect the Passage to be indexed.
2. Tokenize the text.
3. Do linguistic pre-processing of tokens.
4. Index the Passage that each term occurs in.

Tokenisation
Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called *tokens*, perhaps at the same time throwing away certain characters, such as punctuation. Here is an example of tokenization:

Input: Friends, Romans, Countrymen, lend me your ears;

Output: Friends | Romans | Countrymen | lend | me | your | ears

These tokens are often loosely referred to as terms or words, but it is some token times important to make a type/token distinction. A *token* is an instance of a sequence of characters in some particular document that are grouped to type together as a useful semantic unit for processing. A *type* is the class of all tokens term containing the same character sequence. A *term* is a (perhaps normalized) type that is included in the IR system's dictionary. The set of index terms could be entirely distinct from the tokens, for instance, they could be semantic identifiers in a taxonomy, but in practice in modern IR systems they are strongly related to the tokens in the document. However, rather than being exactly the tokens that appear in the document, they are usually derived from them by various normalization processes. For example, if the document to be indexed is *to sleep perchance to dream*, then there are five tokens, but only four types (because there are two instances of *to*). However, if *to* is omitted from the index (as a stop word; That is, as defined here, tokens that are not indexed (stop words) are not terms, and if multiple tokens are collapsed together via normalization, they are indexed as one term, under the normalized form. However, we later relax this definition when discussing classification, where there is no index. Here we drop the requirement of inclusion in the dictionary. A *term* means a normalized word.

It looks fairly trivial: chop on whitespace and throw away punctuation characters. This is a starting point, but even for English there are a number of tricky cases.

Stop Words
Sometimes, some extremely common words that would appear to be of little value in helping select documents matching a user need are excluded from stop words the vocabulary entirely. These words are called *stop words*. The general strat collection egy for determining a stop list is to sort the terms by *collection frequency* (the frequency total number of times each term appears in the

document collection), and then to take the most frequent terms, often hand-filtered for their semantic stop list content relative to the domain of the documents being indexed, as a *stop list*, the members of which are then discarded during indexing. An example of a stop list is shown in. Using a stop list significantly reduces the number of postings that a system has to store. And a lot of the time not indexing stop words does little harm: keyword searches with terms like the and by don't seem very useful. However, this is not true for phrase searches. The phrase query "President of the United States," which contains two stop words, is more precise than President AND "United States." The meaning of lights to London is likely to be lost if the word to is stopped out. A search for Vannevar Bush's article *As we may think* will be difficult if the first three words are stopped out, and the system searches simply for documents containing the word think. Some special query types are disproportionately affected. Some song titles and well-known pieces of verse consist entirely of words that are commonly on stop lists.

Normalisation

Having broken up our documents (and also our query) into tokens, the easy case is if tokens in the query just match tokens in the token list of the document. However, there are many cases when two character sequences are not quite the same but you would like a match to occur. For instance, if you search for *USA*, you might hope to also match documents containing *U.S.A.* token *Token normalization* is the process of canonicalizing tokens so that matches normalization occur despite superficial differences in the character sequences of the to equivalence tokens. The most standard way to normalize is to implicitly create *equivalence* classes *classes*, which are normally named after one member of the set. For instance, if the tokens *anti-discriminatory* and *anti discriminatory* are both mapped onto the term anti discriminatory, in both the document text and queries, then searches for one term will retrieve documents that contain either. The advantage of just using mapping rules that remove characters like hyphens is that the equivalence classing to be done is implicit, rather than being fully calculated in advance: the terms that happen to become identical as the result of these rules are the equivalence classes. It is only easy to write rules of this sort that remove characters.

The usual way is to index unnormalized tokens and to maintain a query expansion list of multiple vocabulary entries to consider for a certain query term. A query term is then effectively a disjunction of several postings lists. The alternative is to perform the expansion during index construction. When the document contains automobile, we index it under car as well (and, usually, also vice versa), it is not obvious when you might want to add characters. For instance, it would be hard to know to turn *anti discriminatory* into *anti-discriminatory*.

Stemming & Lemmatization

For grammatical reasons, documents are going to use different forms of a word, such as *organize*, *organizes*, and *organizing*. Additionally, there are families of derivationally related words with similar meanings, such as *democracy*, *democratic*, and *democratization*. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set. The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

am, are, is⇒be

car, cars, car's, cars'⇒car

The result of this mapping of text will be something like:

the boy's cars are different colors⇒

the boy car be differ color

However, the two words differ in their flavor. *Stemming* usually refers to a crude heuristic process that chops off the ends of words in the hope of *2.2 Determining the vocabulary of terms* 31 achieving this goal correctly most of the time, and often includes the removal of derivational affixes. *Lemmatization* usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return

the base lemma or dictionary form of a word, which is known as the *lemma*. If confronted with the token *saw*, stemming might return just *s*, whereas lemmatization would attempt to return either *see* or *saw*, depending on whether the use of the token was as a verb or a noun. The two may also differ in that stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma. Linguistic processing for stemming or lemmatization is often done by an additional plug-in component to the indexing process, and a number of such components exist, both commercial and open source.

Training

Once the '*Cleaning'* process of the review passage is done, then the created bag of words of the passage is then used to train the model of the classifiers.

The classifier used here is Multinomial Naïve Bayes. We first understand what is Naïve Bayes.

Naïve Bayes

we introduce is the *multinomial* Naive Bayes *Naive Bayes* or *multinomial NB* model, a probabilistic learning method. The probability of a document *d* being in class *c* is computed as

$$P(c|d) \propto P(c) \prod_{1 \le k \le n_d} P(t_k|c)$$

where $P(tk|c)$ is the conditional probability of term *tk* occurring in a document of class *c*. We interpret $P(tk|c)$ as a measure of how much evidence *tk* contributes that *c* is the correct class. $P(c)$ is the prior probability of a document occurring in class *c*. If a document's terms do not provide clear evidence for one class versus another, we choose the one that has a higher prior probability. $\{t1, t2, \ldots, tnd\}$ are the tokens in *d* that are part of the vocabulary we use for classification and *nd* is the number of such tokens in *d*.
For example, $\{t1, t2, \ldots, tnd\}$
for the one-sentence document *Beijing and Taipei join the WTO* might be
{Beijing , Taipei, join , WTO}, with $nd = 4$, if we treat the terms and the as stop words.
In text classification, our goal is to find the *best* class for the document. The maximum a best class in NB classification is the most likely or *maximum a posteriori* (MAP) posteriori

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg\max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \le k \le n_d} \hat{P}(t_k|c).$$

.We write $\hat{P}$ for P because we do not know the true values of the parameters $P(c)$ and $P(tk|c)$, but estimate them from the training set as we will see in a moment.
In Equation , many conditional probabilities are multiplied, one for each position $1 \le k \le nd$. This can result in a floating point underflow. It is therefore better to perform the computation by adding logarithms of probabilities instead of multiplying probabilities. The class with the highest log probability score is still the most probable;

$\log(xy) = \log(x) + \log(y)$

and the logarithm function is monotonic. Hence, the maximization that is actually done in most implementations of NB is:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \le k \le n_d} \log \hat{P}(t_k|c)].$$

Equation has a simple interpretation. Each conditional parameter log $\hat{P}(tk|c)$ is a weight that indicates how good an indicator *tk* is for *c*. Similarly, the prior log $\hat{P}(c)$ is a weight that indicates the relative frequency of *c*. More frequent classes are more likely to be the correct class than infrequent classes. The sum of log prior and term weights is then a measure of how much evidence there is for the document being in the class, and Equation selects the class for which we have the most evidence.

How do we estimate the parameters $\hat{P}(c)$ and $\hat{P}(tk|c)$?

We first try the maximum likelihood estimate, which is simply the relative frequency and corresponds to the most likely value of each parameter given the training data. For the priors this estimate is:

$$\hat{P}(c) = \frac{N_c}{N},$$

where *Nc* is the number of documents in class *c* and *N* is the total number of documents.

We estimate the conditional probability $\hat{P}(t|c)$ as the relative frequency of term *t* in documents belonging to class *c*:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}},$$

where *Tct* is the number of occurrences of *t* in training documents from class *c*, including multiple occurrences of a term in a document.We have made the *positional independence assumption* here, which we will discuss in more detail in the next section: *Tct* is a count of occurrences in all positions *k* in the documents in the training set. Thus, we do not compute different estimates for different positions and, for example, if a word occurs twice in a document, in positions *k*1 and *k*2, then

$$\hat{P}(t_{k_1}|c) = \hat{P}(t_{k_2}|c).$$

The problem with the MLE estimate is that it is zero for a term–class combination that did not occur in the training data. If the term WTO in the training data only occurred in *China* documents, then the MLE estimates for the other classes, for example *UK*, will be zero:

$$\hat{P}(\text{WTO}|UK) = 0.$$

Now, the one-sentence document *Britain is a member of the WTO* will get a conditional probability of zero for *UK* because we're multiplying the conditional probabilities for all terms in Equation. Clearly, the model should assign a high probability to the *UK* class because the term Britain occurs. The problem is that the zero probability for WTO cannot be "conditioned away," no matter how strong the evidence for the class *UK* from other features. The sparseness estimate is 0 because of *sparseness*: The training data are never large enough to represent the frequency of rare events adequately, for example, the frequency of WTO occuring in *UK* documents. add-one To eliminate zeros, we use *add-one* or *Laplace smoothing*, which simply adds smoothing one to each count:

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V}(T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B},$$

where $B = |V|$ is the number of terms in the vocabulary. Add-one smoothing can be interpreted as a uniform prior (each term occurs once for each class) that is then updated as evidence from the training data comes in.

TrainMultinomialNB(C, D)
1 $V \leftarrow$ ExtractVocabulary(D)
2 $N \leftarrow$ CountDocs(D)
3 **for each** $c \in$ C
4 **do** $Nc \leftarrow$ CountDocsInClass(D, c)
5 *prior* [c] $\leftarrow$ Nc/N
6 *textc* $\leftarrow$ ConcatenateTextOfAllDocsInClass(D, c)
7 **for each** $t \in V$
8 **do** *Tct* $\leftarrow$ CountTokensOfTerm(*textc, t*)
9 **for each** $t \in V$
10 **do** *condprob*[t][c] $\leftarrow$ _ *Tct*+1
t_ (*Tct*_+1)
11 **return** *V, prior, condprob*

ApplyMultinomialNB(C, *V, prior, condprob, d*)
1 $W \leftarrow$ ExtractTokensFromDoc(*V, d*)
2 **for each** $c \in$ C
3 **do** *score*[c] $\leftarrow$ log *prior* [c]
4 **for each** $t \in W$
5 **do** *score*[c] += log *condprob*[t][c]
6 **return** arg max$c$∈C *score*[c]

The Package used for the pre-processing of the Passages is '**nltk'** package for python.
For implementing the Multinomial Naïve Bayes algorithm, '**sklearn'** package was used.

Testing

After the training is done, the test set is run on the algorithm. The accuracy of the test set is 84%. The test set was made of size 100 units of reviews.

4 Test Results

- The training accuracy obtained was 96.90%
- The testing accuracy obtained was 84.15%

```
Creating train features.........

Training  features created in 6.000066518783569 s

TRAINING DATA.............
TRAINING COMPLETED IN 1.6875112056732178 s


TRAIN ACCURACY=96.90309690309691

GETTING AND CLEANING TEST DATA NOW.............
1%0%
TEST DATA CLEANED IN 8.283565044403076 s

TESTING AVAILABLE TEST DATA NOW

Test ACCURACY=84.15841584158416
```

5 Applications and Challenges

Applications

- One of the possibility is as an augmentation to recommendation systems, since it is feasible for a system to not recommend items that receive a lot of negative feedback
- Online systems that display ads as sidebars, it is helpful to detect webpages that contain sensitive content inappropriate for ads placement.
- Question answering is another area where sentiment analysis can prove useful.
- Sentiment analysis has specifically been proposed as a key enabling technology in e-Rulemaking, allowing the automatic analysis of the opinions that people submit about pending policy or government-regulation proposals.
- Prevent Hate speeches from circulating online
- As a feedback system from the people.

Challenges

- The sentiment analysis itself is a very tough problem.
- There are millions of variation in the language in which the review is provided.
- Language specific
- The pre-processing techniques affect the accuracy to a great extent.

6 Code

Code for pre-processing the Data

```
#----------------code-------------------#
from nltk.corpus import stopwords
import re
import nltk
from bs4 import BeautifulSoup
import nltk.stem
global i
i=1
from sys import stdout
#---------------Function 1-----------------#
def complete(m):
    global i
    if m==0:
        return
    print('\r'+str(int((i/m)*100))+'%',end='')
    stdout.flush()
    if i==m:
        i=1
    i=i+1
#----------------Function 2----------------#
def clean(file,m):
    soup=BeautifulSoup(file,'lxml')
    #stop=set([word for word in (stopwords.words('english'))])
    s=nltk.stem.SnowballStemmer('english')

    file=soup.get_text().lower()
    tokens=[re.match('[a-z]*',s.stem(word)).group() for word in nltk.word_tokenize(file)]

    k=nltk.pos_tag([word for word in tokens if word!='' ])
```

```
    #words=set([word for word,tag in k if (tag.startswith("NN") or tag.startswith("JJ") or
tag.startswith("RB") or tag.startswith("VB") ) ])


    string=" ";
    string=string.join(tokens)
    complete(m)


    return string
#-----------------------End of code---------------------#
```

Code for Model training

```
#-----------------------Begin code----------------------#

import pandas as pd
import numpy as np
import tokenise as cln
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
import time

data=pd.DataFrame.from_csv('labeledtrainData.tsv',sep='\t')
data.index=np.array(range(len(data)))

train_data=data.loc[:1000]
#train_data=data;
m=len(train_data);
test_data=data.loc[1500:1600];
n=len(test_data)
#test_data=pd.DataFrame.from_csv('testData.tsv',sep='\t')
test_data.index=np.array(range(len(test_data)))
```

```
print('CLEANING TRAIN DATA....')
t=time.time()
i=1;
clean_train_review=[cln.clean(rev,m) for rev in train_data.review]
print('\nTrain Data Has been cleaned in '+str(time.time()-t)+' s')

print('EXAMPLE OF CLEAN REVIEW BY ALGORITHM=\n')
print('GIVEN TRAIN REVIEW----------  \n'+str(train_data.review.loc[1]))
print('\nCLEAN REVIEW ----------\n'+str(clean_train_review[1]))

vectorizer=CountVectorizer(ngram_range=(1,3),analyzer='word',max_features=8000)

print('\nCreating train features.....')
t=time.time();
train_data_features=vectorizer.fit_transform(clean_train_review).toarray()
print('\nTraining  features created in '+str(time.time()-t)+' s')

vocab=vectorizer.get_feature_names()
train_target=np.array(train_data.sentiment).astype(int)

print('\nTRAINING DATA.....')
t=time.time();
clf=MultinomialNB()
classes=[0,1]
clf=clf.partial_fit(train_data_features,train_target,classes)
print('TRAINING COMPLETED IN '+str(time.time()-t)+' s')

pred=clf.predict(train_data_features)

print()
print('\nTRAIN ACCURACY='+str(np.mean(pred==train_data.sentiment)*100));
```

```
print('\nGETTING AND CLEANING TEST DATA NOW.....')
t=time.time();
clean_test_review=[cln.clean(rev,n) for rev in test_data.review]
print('\nTEST DATA CLEANED IN '+str(time.time()-t)+' s')

test_data_features=vectorizer.transform(clean_test_review).toarray()

print("\nTESTING AVAILABLE TEST DATA NOW");
pred=clf.predict(test_data_features)

print()
print('Test ACCURACY='+str(np.mean(pred==test_data.sentiment)*100));

feedback_pos= np.sum((pred==1)&(test_data.sentiment==1))/np.sum(test_data.sentiment==1)
feedback_neg= np.sum((pred==0)&(test_data.sentiment==0))/np.sum(test_data.sentiment==0)

ch=1;

wrong_count=0
feed_train=[]
feed_target=[]
while ch!=0:
   test_string=input('ENTER UR MOVIE REVIEW TO TEST SENTIMENT.....\n')
   cln_string=[cln.clean(test_string,0)]

   print('Your cleaned review is ='+str(cln_string))
   my_data=vectorizer.transform(cln_string).toarray()
   pred=clf.predict(my_data)
   if pred==1:
      print('YOUR REVIEW IS POSITIVE !!')
   else:
```

```
        print('YOUR REVIEW IS NEGATIVE !!')
#    feed=int(input('IS THAT CORRECT? PLEASE ENTER YOUR FEEDBACK REVIEW(1 for pos  0 for neg )'))
#    if feed!=pred:
#        feed_train.append(str(cln_string))
#        feed_target.append(feed)
#        wrong_count=wrong_count+1
#        print('WRONG COUNT='+str(wrong_count))
#    if wrong_count==5:
#        X=vectorizer.transform(feed_train).toarray()
#        Y=np.array(feed_target)
#        print('fitting feedback data')
#        clf.partial_fit(X,Y)
#        feed_train=[]
#        feed_target=[]
#        wrong_count=0
    ch=(input('TO EXIT PRESS 0 \n'))
    if ch!='0':
        ch=1
    else:
        ch=0


    #-----------------------End of code----------------------#
```

7 Output

Output 1

```
wdir='C:/Users/Akshit Verma/Desktop/Project/Program')
CLEANING TRAIN DATA....
100%
Train Data Has been cleaned in 59.08996295928955 s
EXAMPLE OF CLEAN REVIEW BY ALGORITHM=

GIVEN TRAIN REVIEW----------
\The Classic War of the Worlds\" by Timothy Hines is a very entertaining film that obviously
goes to great effort and lengths to faithfully recreate H. G. Wells' classic book. Mr. Hines
succeeds in doing so. I, and those who watched his film with me, appreciated the fact that it
was not the standard, predictable Hollywood fare that comes out every year, e.g. the
Spielberg version with Tom Cruise that had only the slightest resemblance to the book.
Obviously, everyone looks for different things in a movie. Those who envision themselves as
amateur \"critics\" look only to criticize everything they can. Others rate a movie on more
important bases,like being entertained, which is why most people never agree with the
\"critics\". We enjoyed the effort Mr. Hines put into being faithful to H.G. Wells' classic
novel, and we found it to be very entertaining. This made it easy to overlook what the
\"critics\" perceive to be its shortcomings."

CLEAN REVIEW --------------
 classic war of the worlds  by timothi hine is a veri entertain film that obvious goe to
great effort and length to faith recreat h g well  classic book  mr hine succeed in do so  i
and those who watch his film with me  appreci the fact that it was not the standard  predict
hollywood fare that come out everi year  e  the spielberg version with tom cruis that had
onli the slightest resembl to the book  obvious  everyon look for differ thing in a movi
those who envis themselv as amateur  critics  look onli to critic everyth they can  other
rate a movi on more import base  like be entertain  which is whi most peopl never agre with
the  critics  we enjoy the effort mr hine put into be faith to h  well  classic novel  and
we found it to be veri entertain  this made it easi to overlook what the  critics  perceiv
to be it shortcom

Creating train features..........

Training  features created in 5.216703176498413 s

TRAINING DATA.............
TRAINING COMPLETED IN 0.1511516571044922 s


TRAIN ACCURACY=92.7036481759
```

Output 2

```
TRAIN ACCURACY=92.7036481759

GETTING AND CLEANING TEST DATA NOW.............
1%0%
TEST DATA CLEANED IN 2.8746891021728516 s

TESTING AVAILABLE TEST DATA NOW

Test ACCURACY=85.1485148515

ENTER UR MOVIE REVIEW TO TEST SENTIMENT.............
This film should be brilliant.  It sounds like a great plot, the actors are first grade, and
the supporting cast is good as well, and Stallone is attempting to deliver a good
performance. However, it can't hold up
Your cleaned review is =['this film should be brilliant  it sound like a great plot  the
actor are first grade  and the support cast is good as well  and stallon is attempt to deliv
a good perform  howev  it can hold up']
YOUR REVIEW IS POSITIVE !!

TO EXIT PRESS 0


ENTER UR MOVIE REVIEW TO TEST SENTIMENT.............
Full of zany characters and richly applied satire, and some great plot twists
Your cleaned review is =['full of zani charact and rich appli satir  and some great plot
twist']
YOUR REVIEW IS POSITIVE !!

TO EXIT PRESS 0


ENTER UR MOVIE REVIEW TO TEST SENTIMENT.............
It was pathetic. The worst part about it was the boxing scenes.
Your cleaned review is =['it was pathet  the worst part about it was the box scene ']
YOUR REVIEW IS NEGATIVE !!

TO EXIT PRESS 0
```

8 Referances

1 Bo Pang and Lillian Lee, Opinion Mining and Sentiment Analysis

2 Dhiraj Gurkhe , Niraj Pal and Rishit Bhatia ,Effective Sentiment Analysis of Social Media Datasets using Naive Bayesian Classification.

3 Russell, Stuart; Norvig, Peter (2003) [1995]. Artificial Intelligence: A Modern Approach (2nd ed.). Prentice Hall. *ISBN 978-0137903955*.