Movie Review Sentiment Analyser

A Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Technology
in
Computer Science & Engineering

by
Kritank Singh, Lakshay Saini & Avinash Sitaram

to the
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY
ALLAHABAD PRAYAGRAJ
July, 2020

UNDERTAKING

We declare that the work presented in this report titled "*Movie Review Sentiment Analyser*", submitted to the Computer Science and Engineering Department, Motilal Nehru National Institute of Technology Allahabad, Prayagraj, for the award of the ***Bachelor of Technology*** degree in ***Computer Science & Engineering***, is our original work. We have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, We accept that our degree may be unconditionally withdrawn.

July, 2020
Allahabad

(Kritank Singh, Lakshay Saini & Avinash Sitaram)

CERTIFICATE

Certified that the work contained in the report titled "*Movie Review Sentiment Analyser*", by *Kritank Singh, Lakshay Saini & Avinash Sitaram*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Prof. R.S. Yadav)
Computer Science and Engineering Dept.
M.N.N.I.T, Allahabad

July, 2020

Preface

As humans, we are able to classify text into positive/negative subconsciously. Today through this report one can get to know if such classification can be achieved through computers.
We will see various milestones that have been achieved in the field and what are the various challenges that one could face if he decide to implement such classifier. This report could be helpful for those individual who have a keen interest in the field of computer science and engineering.
Throughout the course of the report one could get lots on insights of the process of building a working sentiment analyser.

Acknowledgements

It is a great pleasure to thank the giants on whose shoulders we stand.
First of all, we would like to thank my supervisor Dr R.S Yadav (Faculty CSED,MNNIT Allahabad) for his constant guidance and support.
Secondly, all the teammates who have been together through thick and thin. We together have made this possible.
Lastly, All our friends and family without whose support & blessing nothing is possible.

Contents

Chapter 1

Introduction

Sentiment Analysis (SA) is one of the most widely studied applications of Natural Language Processing (NLP) and Machine Learning (ML). This field has grown tremendously with the advent of the Web 2.0. The Internet has provided a platform for people to express their views, emotions and sentiments towards products, people and life in general. Thus, the Internet is now a vast resource of opinion rich textual data.

Sentiment Analysis also known as "Opinion Mining", Sentiment Analysis refers to the use of Natural Language Processing to determine the attitude, opinions and emotions of a speaker, writer, or other subject within an online mention. Essentially, it is the process of determining whether a piece of writing is positive or negative. This is also called the Polarity of the content.

As humans, we are able to classify text into positive/negative subconsciously. For example, the sentence "The kid had a gorgeous smile on his face", will most likely give us a positive sentiment. In layman's terms, we kind of arrive at such a conclusion by examining the words and averaging out the positives and the negatives. For instance, the words "gorgeous" and "smile" are more likely to be positive, while words like "the", "kid" and "face" are really neutral. Therefore, the overall sentiment of the sentence is likely to be positive.

The goal of Sentiment Analysis is to harness this data in order to obtain important information regarding public opinion, that would help make smarter business decisions, political campaigns and better product consumption. Sentiment Analysis focuses on identifying whether a given piece of text is subjective or objective and if it is subjective, then whether it is negative or positive.

1.1 Motivation

According to Ramteke et al. (2012) motivation for Sentiment Analysis is two-fold. Both consumers and producers highly value "customer's opinion" about products and services. Thus, Sentiment Analysis has seen a considerable effort from industry as well as academia.

1.1.1 The Consumer's Perspective

While taking a decision it is very important for us to know the opinion of the people around us. Earlier this group used to be small, with a few trusted friends and family members. But, now with the advent of Internet we see people expressing their opinions in blogs and forums. These are now actively read by people who seek an opinion about a particular entity (product, movie etc.). Thus, there is a plethora of opinions available on the Internet.

From a consumers' point of view extracting opinions about a particular entity is very important. Trying to go through such a vast amount of information to understand the general opinion is impossible for users just by the sheer volume of this data. Hence, the need of a system that differentiates between good reviews and bad reviews. Further, labeling these documents with their sentiment would provide a succinct summary to the readers about the general opinion regarding an entity.

1.1.2 The Producer's Perspective

With the explosion of Web 2.0 platforms such as blogs, discussion forums, etc., consumers have at their disposal, a platform to share their brand experiences and opinions, positive or negative regarding any product or service. According to Pang and Lee (2008) these consumer voices can wield enormous influence in shaping the opinions of other consumers and, ultimately, their brand loyalties, their purchase decisions, and their own brand advocacy.

Since the consumers have started using the power of the Internet to expand their horizons, there has been a surge of review sites and blogs, where users can perceive a product's or service's advantages and faults. These opinions thus shape the future of the product or the service. The vendors need a system that can identify trends in customer reviews and use them to improve their product or service and also identify the requirements of the future.

1.1.3 The Societies' Perspective

Recently, certain events, which affected Government, have been triggered using the Internet. The social networks are being used to bring together people so as to organize mass gatherings and oppose oppression. On the darker side, the social networks are being used to insinuate people against an ethnic group or class of people, which has resulted in a serious loss of life. Thus, there is a need for Sentiment Analysis systems that can identify such phenomena and curtail them if needed.

1.2 Applications

- **Real time Prediction:** Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- **Multi class Prediction:** This algorithm is also well known for multi class prediction features. Here we can predict the probability of multiple classes of target variables.

- **Text classification/ Spam Filtering/ Sentiment Analysis:** Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam email) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)

- **Recommendation System:** Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

Chapter 2

Related Work

For the past decade or so, there has been a lot of research that has taken place in Sentiment Analysis. We will discuss some of those works in this chapter.

2.1 Lexical Resources

There are various lexical resources in use, for Sentiment Analysis. We discuss dictionary and SentiWordNet in this section.

2.1.1 Dictionary

All sentiment analysis tools require a list of words or phrases with positive and negative connotation, and such a list of words is referred to as a dictionary. Dictionary is an important lexical resource for Sentiment Analysis.

A single dictionary for all the domains, is difficult to generate. This happens because of the domain specificity of words. Certain words convey different sentiments in different domains. For example:

- Word like "fingerprints" conveys a major breakthrough in a criminal investigation whereas it will be negative for smartphone manufacturers.
- "Freezing" is good for a refrigerator but pretty bad for software applications.

- We want the movie to be “unpredictable” but not our cell phones.

A few popular dictionaries are discussed in the following sections.

- The Loughran and McDonald Financial Sentiment Dictionary:
  Loughran and McDonald (2011) show how, applying a general sentiment word list to accounting and financial topics, will lead to a high rate of misclassification. They found that around three-fourths of the negative word in a general sentiment dictionary were not negative in the financial domain. So they created the dictionary, “The Loughran and McDonald Financial Sentiment Dictionary”. It is a publicly available domain-specific dictionary and it contains custom lists of positive and negative words specific to the accounting and financial domain.

- Lexicoder Sentiment Dictionary (LSD):
  Lexicoder Sentiment Dictionary (LSD) is also a domain-specific dictionary. It expands the score of coverage of existing sentiment dictionaries, by removing neutral and ambiguous words and then extracting the most frequent ones. Some important features of this dictionary are the implementation of basic word sense disambiguation with the use of phrases, truncation and preprocessing, as well as the effort to deal with negations.

- WordStat Sentiment Dictionary:
  The WordStat Sentiment Dictionary was formed by combining words from the Harvard IV dictionary, the Regressive Imagery dictionary (Martindale, 2003) and the Linguistic and Word Count dictionary (Pennebaker, 2007). It contains a list of more than 4733 negative and 2428 positive word patterns. Sentiment is not predicted by these word patterns but by a set of rules that take into account negations.

2.1.2 Sentiwordnet

SentiwordNet is a lexical resource in which each wordnet synset ‘s’ is associated to three numerical scores Obj(s), Pos(s) and Neg(s), which describe how objective,

positive and negative the terms contained in the synset are. Each of the three scores range from 0.0 to 1.0, and their sum is 1.0 for each synset. A graded evaluation of opinion, as opposed to hard evaluation, proves to be helpful in the development of opinion mining applications.

2.2 Feature Engineering

The efficiency of classifiers depends upon the selection of features. Under feature engineering, we discuss feature selection in Sentiment Analysis.

- Sense based feature: The traditional approaches to sentiment analysis have been using lexeme and syntax based features. Balamurali et al. (2011) focus on a new approach to sentiment analysis by using "word senses" as "semantic features" for sentiment classification. In his paper, he used WordNet 2.1 (Fellbaum, 1998) as the sense repository. Each word is mapped to a synset based on its sense.

- Term Presence vs. Term Frequency: Traditionally term frequency was used as a feature in all the sentiment classification tasks. Later, Pang et al. (2002) showed that term presence is more important than term frequency. Term presence is a binary-valued feature which shows whether a term is present or not, unlike the term frequency feature which kept a count of the terms. This has been proved experimentally that term presence gives better results than term frequency

- Term Position as feature: An observation that has been made is that term position plays an important role in determining the sentiment. For example, in movie reviews, the review might begin with some sentiment, discuss about the movie and at the end summarize it with the author's view. We see that the sentiment is mainly present in the concluding sentences. Hence, we conclude that term position is very important. The sentiment of the initial sentences might not be the sentiment of the whole review. So term position is also included as a feature.

2.3 Machine Learning techniques

Using movie reviews as data, Pang et al. (2002) show that standard Machine Learning techniques outperform human-produced baselines. The movie-review domain has been chosen because there are large on-line collections of such reviews, and reviewers often summarize their overall sentiment using stars ratings etc., which are easily extractable by machine. We discuss a few of the Machine Learning Approaches.

- Na¨ıve Bayes: A Na¨ıve Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (na¨ıve) independence assumptions.1 The Bayes' rule is given below: P(c—d) = P (c)P (d—c) P (d) Na¨ıve Bayes holds a strong independence assumption which states that features are independent of each other. By applying conditional independence assumption of features we get: PNB(c—d) = P (c) QN i=1 P (fi—c) ni (d) P (d) Where, P(c—d): probability of data d belonging to class c. P(d): probability of data. P(fi —c): probability of the ith feature belonging to class c. So, we see that the probability is expressed in terms of product of features. The conditional independence assumption may or may not hold in every situation. Yet, Na¨ıve Bayes has proven to give good results.

- Maximum Entropy: Maximum entropy classifiers are commonly used as alternatives to Na¨ıve Bayes classifiers because they do not assume statistical independence of the independent variables (commonly known as features).2 (Nigam, 1999) shows that it sometimes, but not always, outperforms Na¨ıve Bayes classifier. Here is the exponential form taken by Maximum Entropy classifier: PME(c—d) = 1 Z(d) exp(P i iFi,c(d, c)) Where, Z(d) is a normalization function. Fi,c is a feature /class function for fi and class c, defined as follows: Fi,c(d, c0) = 1 : ni(d) ¿ 0 and c0 = c 0 : otherwise The 0 i,cs are feature-weight parameters. Its large value shows that fi is a strong indicator of the class c. Maximum Entropy classifier focuses on choosing parameters which maximize the the distribution. It does not make any assumptions about the relationships between features and so performs better than Na¨ıve Bayes when conditional independence assumptions are not met

2.4 Sarcasm

Sarcasm is the use of positive words to express a negative opinion about some target. In this section, we look at the different features that can be used in sarcasm detection.

- Intensifiers as features:
  Liebrecht et al. (2013) introduce a sarcasm detection system for tweets, messages on the microblogging service offered by Twitter. In micro-blogging sites such as Twitter, tweets are often explicitly marked with the sarcasm hashtag to indicate that it is sarcastic. Research has shown that sarcasm is often signaled by hyperbole, using intensifiers and exclamations. In contrast to this, non-hyperbolic sarcastic messages often have an explicit marker. Unlike a simple negation, a sarcastic message conveys a negative opinion using only positive words or intensified positive words. According to Gibbs and Izett (2005), sarcasm divides its addressees into two groups; a group of people who understand sarcasm (the so-called group of wolves) and a group of people who do not understand sarcasm (the so-called group of sheep). On Twitter, the senders use the hashtag in order to ensure that the addressees detect the sarcasm in their utterance.

- Lexical and pragmatic features:
  Gonzalez-Ibanez et al. (2011) throw light upon the impact of lexical and pragmatic factors for effectively identifying sarcastic utterances in Twitter. He also compares the performance of machine learning techniques and human judges on this task. Sarcastic tweets were collected using the sarcasm hashtag and automatic filtering was done to remove retweets, duplicates, quotes, spam, tweets written in languages other than English, and tweets with URLs. Since hashtagged tweets are noisy, all the tweets were filtered where the hashtags of interest were not located at the very end of the message.

- Pattern-based features:
  Davidov et al. (2010) make use of pattern-based features. Words are divided into two categories, high frequency words(HFW) and content words(CW). A

word whose corpus frequency is higher than FH is HFW and a word whose corpus frequency is lower than FC is a content word. A pattern is an ordered sequence of high frequency words and slots for content words. After extracting patterns, hundreds of patterns are obtained out of which some may be very specific and some may be very general. In order to reduce the feature space, pattern selection is done. Then for each pattern, feature value is calculated.

2.5 Hyperbole

Hyperbole is the use of exaggeration as a rhetorical device or figure of speech. Colston and Keller (1998) compare irony with hyperbole and the extent to which they express surprise. The authors test the inflation hypothesis which states that hyperbole is understood because it inflates the discrepancy between the expected and ensued situation. If hyperbole is understood because of this inflation then it should not matter that a speaker's surprise is obvious when explicitly stated expectations are violated. Inflating that discrepancy should still express surprise.
The author tests whether or not hyperbole expresses surprise when a speaker's expectations are explicitly known. This is done by combining irony and hyperbole to see how they together express surprise. For example:

- Hyperbole: I see we got 10 feet of snow last night
- Irony: I see we got some slight flurries last night
- Combination of irony and hyperbole: I see we didn't get any snow at all last night

The latter utterance is both ironic and hyperbolic because it goes back to what was expected (a slight amount of snow) and because it inflates the discrepancy between what was expected and what ensued.

2.6 Thwarting

Thwarting is the phenomenon in which a minority of sentences decide the polarity of the whole piece of text or document. There are two approaches to detect whether a document is thwarted or not. One is rule-based approach and the other is statistical approach. Ramteke et al. (2011) describe both the approaches. The author has made use of domain ontology to handle thwarting. Domain ontology comprises of features and entities from the domain and the relationships between them depicted by a hierarchy. For building the ontology the features and entities need to be identified and then linked in the form of an hierarchy. The author has built the domain ontology manually.

The rule-based approach makes use of a domain ontology . An ontology gives weightings to entities related to a domain. Ramteke et al.(2012) built up a system using the domain ontology for camera review domain. The word polarities were found using four different lexicons namely SentiWordNet, Inquirer, BL Lexicon and Taboada. The entity specific polarities were found by considering the dependencies found using Stanford Dependency parser. The weighing scheme gave a weight of 1 to the leaf nodes and then subsequently increased the weights by 1 for the higher levels. A review is said to be thwarted, if the root node has a different polarity from its leaf nodes.

The drawback of rule-based approach is that it gives equal weightage to all the features in the domain ontology. This drawback is overcome by assigning weights to features. This approach also makes use of the domain ontology . It aims at finding the features and their weights that are used for training the classifier, from the domain ontology. The review is represented as a sequence of weighted polarities. The review is linearly scanned and if a word is encountered belonging to the ontology, its polarity and weight are extracted using the corresponding node in the ontology. The sequence of occurrence of words is maintained since position is vital to determining thwarting. The features are extracted from the sequence and fed to the classifier, which classifies the review to be thwarted or not.

Chapter 3

Proposed Work

We need to train a model to recognise the emotion of the movie review and classify them as either positive or negative. It is a binary classification problem.
The data set used here is known as 'Polarity Data set'.The data set used in our project has been taken from **Stanford**.
This is a data set for binary sentiment classification containing substantially more data than previous benchmark data sets.

This data set has following properties:

- They provide a set of 25,000 highly polar movie reviews for training, and 25,000 for testing.
- It only comprises of English reviews.
- There is white space around punctuation like periods, commas, and brackets.
- There is additional unlabeled data for use as well.

This problem is approached in three stages:

1. **Data Prepossessing**
   This stage involves the following steps:
   (a) Loading the data

(b) Loading the data

(c) Analysis of data

(d) Cleaning the data

(e) Defining a vocabulary

(f) Creating the feature list (words that would be considered in the sentiment analysis)

2. **Bag Of Words Representation**

   This stage focuses on preparing the data for the training model. It involves following steps:

   - Converting reviews to lines of tokens.
   - Encoding reviews with a bag-of-words model representation.

3. **Sentiment Analysis Model**

   In this project we develop a **Naive Bayes Model** to predict the sentiment/emotion of the reviews.

Chapter 4

Experimental Setup and Results Analysis

The various steps taken and their results in the process:

4.1 Road Map

1. **Data Prepossessing**

   (a) **Training and Testing set** The data set contains positive and negative reviews. Among them, some positive and negative reviews are used as training sets and the remaining positive and negative reviews are used as test data.

   (b) **Load the data** The load method helps to load the data set into the memory.

   (c) **Analysis of the data**

   - Size of the dataset
   - Total Positive Sentiment in the dataset
   - Total Negatives Sentiment in the dataset
   - Average Positive review length

- Average Negative review length
- Train Data size
- Test Data size

```
Data Annalysis:
Data size: 4000
Total Positive in dataset: 2000
Total Negative in dataset: 2000
Average postive review Length: 240.4
Average negative review Length: 229.26
Train Data size: 3000
Test Data size: 1000
```

2. **Analysis of the data**

   (a) **Define a Vocabulary**

   i. **Cleaning the data:** This step helps to turn data into clean tokens. The following methods are employed in this project to generate clean tokens.

   - Remove all the punctuations.
   - Remove all stop words.
   - Remove all words with small length (length ¡= 1 character).

   What are Stopwords?

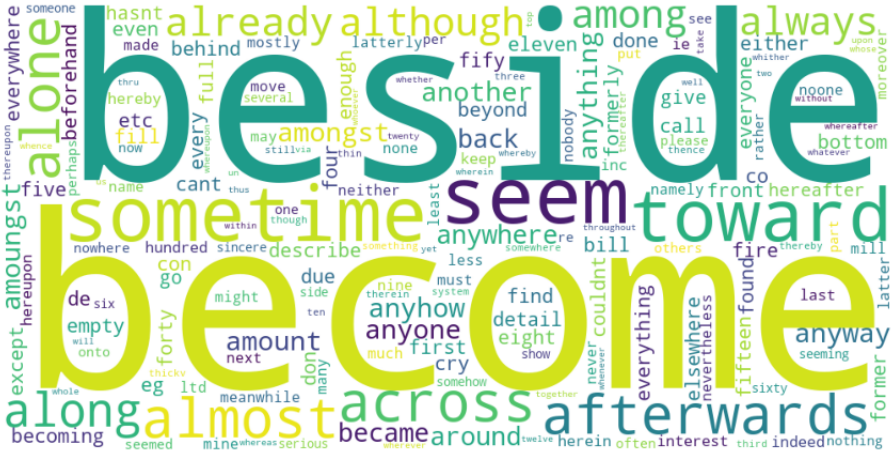
   Stop words are words which are filtered out before or after processing of natural language data. Though "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list.

   Stopwords Example:

   'a', 'about', 'above', 'across', 'after', 'afterwards', 'again', 'against', 'all', 'almost', 'alone','along', 'already', 'also', 'although', 'always', 'am', 'among', 'amongst', 'amongst', 'amount', 'an', 'and', 'another',

'any', 'anyhow', 'anyone', 'anything', 'anyway', 'anywhere', 'are', 'around','as', 'at', 'back', 'be', 'became', 'because', 'become', 'becomes', 'becoming', 'been', 'before', 'beforehand', 'behind', 'being', 'below', 'beside', 'besides', 'between', 'beyond', 'bill', 'both', 'bottom', 'but', 'by', 'call', 'can', 'cannot', 'cant', 'co', 'con', 'could', 'couldn't', 'cry', 'de', 'describe', 'detail', 'did', 'do', 'does', 'doing', 'don', 'done','down', 'due', 'during', 'each', 'eg', 'eight', 'either', 'eleven', 'else' .....(and many more)

Stopword wordcloud :



ii. **Build Vocabulary:** It is necessary to define a vocabulary of words to vectorise the document of reviews. The below method defines a vocabulary of words and maintains the count of the occurrence of each word in the vocabulary.

The **cleanbuildVocabulary** function of the **Features** class performs cleaning and building the vocabulary

iii. **Build Features:** From the vocabulary created above features are created on the basis of these features our reviews are classified as Positive or Negative.

The **buildFeatures** function of the **Features** class performs this task.

(b) **Reviews to lines of tokens (Reviews →Tokens)**
Here, each review is loaded to memory and cleaned for tokens. The obtained tokens are further cleaned by retaining only those tokens that are also in features we defined previously.

(c) **Reviews to Bag-Of-Words Vectors (Reviews→Tokens →Vectors)**
The data that is fed to the training model should be encoded into numerical values and all the training examples should be of uniform length. So far what we have is the training and test data in the textual form and of non-uniform length. We use the Bag-Of-Words model to encode the data in order to make it suitable for training/learning.
In this model each review is transformed to an encoded vector where each word is assigned a score. The length of the vector corresponds to the length of the vocabulary. There are different methods for scoring the words. In this project we use **'count'** for scoring the words. Let's understand how it works.

Example
Let's assume that
Features = this, that, is, mine, not, cat, dog
text = "This this is mine".
Then encoded text using the 'count' scoring method is
[2, 0, 1, 1, 0, 0, 0]

Explanation:

- The number of words in the text is 4.
- The length of the vector = length of the features = 7
- Score of 'this' = (total occurence of 'this' in the text) = 2
- The score of 'this' is stored in the index corresponding to 'this' in the vector(for now, we can assume it to be in the same index as in vocabulary).

- The words 'that', 'not', 'cat', 'dog' do not occur in the text so their corresponding indexes in the vector are assigned a score of zero

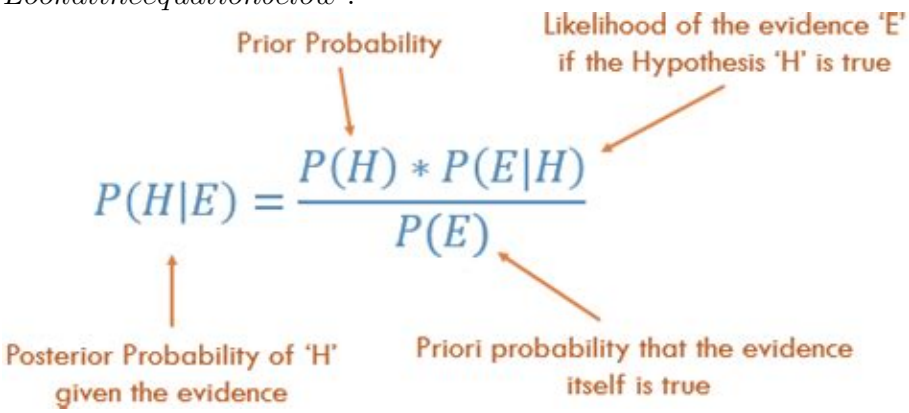
The process of **getWordVectorCount** of the **NaiveBayes** performs the complete task of review →token →count vector.

This count vector is fed into the model to train/predict the class to which the review belongs.

3. SENTIMENT ANALYSIS MODEL

(a) **Building a Classifier Model** There are different types of models that we can use. Here we have used Naive Bayes Classifier.

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other features. Bayes theorem provides a way of calculating posterior probability $P(\frac{H}{E})$
$fromP(H), P(E)andP(\frac{E}{H})$.
$Lookattheequationbelow$:



Above,

(b) P(H—E) is the posterior probability of class (H, target) given predictor (E, attributes).

(c) P(H) is the prior probability of class.

(d) P(E—H) is the likelihood which is the probability of a predictor given class.

(e) P(E) is the prior probability of the predictor

How does Naive Bayes algorithm work?

Let's understand it using an example. Below I have a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather conditions. Let's follow the below steps to perform it.

Step 1: Convert the data set into a frequency table

Step 2: Create a Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

Step 3: Now, use the Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Problem: Players will play if the weather is sunny. Is this statement correct? We can solve it using the above discussed method of posterior probability.

$$P(\frac{Yes}{Sunny}) \;=\; P(\frac{Sunny}{Yes}) \;\text{*P(Yes)/ p(Sunny)}$$

Here we have P (Sunny /Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64 Now, P (Yes / Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different classes based on various attributes. This algorithm is mostly used in **text classification** and with problems having **multiple classes**.

4.2 Implementation

We have made two classes a **Feature** class and **NaiveBase(Multinomial)** class and a helper function to load the dataset.

4.2.1 Load Data

```
# Helper Functions
def load(directory):
    X = [] # an element of X is represented as (filename,text)
    Y = [] # an element of Y represents the sentiment category of the corresponding X element
    for category in os.listdir(directory):
        for document in os.listdir(directory+'/'+category):
            with open(directory+'/'+category+'/'+document, "r", encoding="utf8") as f:
                X.append((document,f.read()))
                Y.append(category)
    return X,Y;



#Setting the data directory
#train_directory = 'Training_Data'
train_directory = 'train'
#test_directory = 'Test_Data'
test_directory = 'test'

#Loading the dataset
print("Loading Data!")
X_train,Y_train = load(train_directory)
X_test,Y_test = load(test_directory)
print("Data Loaded!")
print()
```

4.2.2 Data Analysis

```
#Annalysis of Data
print("Data Annalysis:")
#display size of data
print("Data size:",len(X_train)+len(X_test))

#displaying postive and negative review count
pos_count=0
neg_count=0
for i in range(len(Y_train)):
    if(Y_train[i]== "Positive"):
        pos_count+=1
    else : neg_count+=1
for i in range(len(Y_test)):
    if(Y_test[i]== "Positive"):
        pos_count+=1
    else : neg_count+=1

print("Total Positive in dataset:",pos_count)
print("Total Negative in dataset:",neg_count)

#displaying average postive and negative review length
pos_len=0
neg_len=0

for i in range(len(Y_train)):
    if(Y_train[i]== "Positive"):
        pos_len+=len(X_train[i][1].split())
    else : neg_len+=len(X_train[i][1].split())
for i in range(len(Y_test)):
    if(Y_test[i]== "Positive"):
        pos_len+=len(X_test[i][1].split())
    else : neg_len+=len(X_test[i][1].split())

print("Average postive review Length:",round(pos_len/pos_count,2))
print("Average negative review Length:",round(neg_len/neg_count,2))

#Display test and train data sizes
print("Train Data size:",len(X_train))
print("Test Data size:",len(X_test))
print()
```

4.2.3 Feature Class

```
class Features:
    def __init__(self):
        self.vocab = {}
        self.features = []
        self.num_words =[]
        self.cutoff_freq = 0

    def cleanBuildVocabulary(self,X_train):
        for i in range(len(X_train)):

            for word in X_train[i][1].split():
                word_new  = word.strip(string.punctuation).lower()
                if (len(word_new)>2)  and (word_new not in stopwords):
                    if word_new in self.vocab:
                        self.vocab[word_new]+=1
                    else:
                        self.vocab[word_new]=1

    def plotVocabulary(self):
        self.num_words = [0 for i in range(max(self.vocab.values())+1)]
        freq = [i for i in range(max(self.vocab.values())+1)]
        for key in self.vocab:
            self.num_words[self.vocab[key]]+=1
        plt.plot(freq,self.num_words)
        plt.axis([1,10, 0, 20000])
        plt.xlabel("Frequency")
        plt.ylabel("No of words")
        plt.grid()
        plt.show()

    def setCutOff(self,num):
        self.cutoff_freq = num
        # For deciding cutoff frequency
        num_words_above_cutoff = len(self.vocab)-sum(self.num_words[0:self.cutoff_freq])
        print("Number of words with frequency higher than cutoff frequency({}) :".format(self.cutoff_freq),num_words_above_cutoff)

    def buildFeatures(self):
        for key in self.vocab:
            if self.vocab[key] >=self.cutoff_freq:
                self.features.append(key)
```

4.2.4 Naive Base(Multinomial) Class

```
# Implementing Naive Bayes from scratch
class NaiveBayes:

    def __init__(self,feature):
        # count is a dictionary which stores several dictionaries corresponding to each sentiment category
        # each value in the subdictionary represents the freq of the key corresponding to that setiment category
        self.count = {}
        # classes represents the different sentiment categories
        self.classes = None
        self.feature=feature

    def fit(self,X_train,Y_train):
        # This can take some time to complete
        self.classes = set(Y_train)
        for class_ in self.classes:
            self.count[class_] = {}
            for i in range(len(X_train[0])):
                self.count[class_][i] = 0
            self.count[class_]['total'] = 0
            self.count[class_]['total_points'] = 0
        self.count['total_points'] = len(X_train)

        for i in range(len(X_train)):
            for j in range(len(X_train[0])):
                self.count[Y_train[i]][j]+=X_train[i][j]
                self.count[Y_train[i]]['total']+=X_train[i][j]
            self.count[Y_train[i]]['total_points']+=1

    def __probability(self,test_point,class_):

        log_prob = np.log(self.count[class_]['total_points']) - np.log(self.count['total_points'])
        total_words = len(test_point)
        for i in range(len(test_point)):
            current_word_prob = test_point[i]*(np.log(self.count[class_][i]+1)-np.log(self.count[class_]['total']+total_words))
            log_prob += current_word_prob

        return log_prob
    def __predictSinglePoint(self,test_point):

        best_class = None
        best_prob = None
        first_run = True

        for class_ in self.classes:
            log_probability_current_class = self.__probability(test_point,class_)
            if (first_run) or (log_probability_current_class > best_prob) :
                best_class = class_
                best_prob = log_probability_current_class
                first_run = False

        return best_class


    def predict(self,X_test):
        # This can take some time to complete
        Y_pred = []
        for i in range(len(X_test)):
        # print(i) # Uncomment to see progress
            Y_pred.append( self.__predictSinglePoint(X_test[i]) )

        return Y_pred

    def accuracy(self,Pos_true_count,Neg_true_count,Pos_false_count,Neg_false_count):
        # returns the mean accuracy
        return (Pos_true_count+Neg_true_count)/(Pos_true_count+Neg_true_count+Pos_false_count+Neg_false_count

    def recall(self,Pos_true_count,Neg_true_count,Pos_false_count,Neg_false_count):
        #returns recall
        return Pos_true_count/(Pos_true_count+Pos_false_count)

    def precision(self,Pos_true_count,Neg_true_count,Pos_false_count,Neg_false_count):
        #returns precision
        return Pos_true_count/(Pos_true_count+Neg_false_count)

    def fmeasure(self,recall,precision):
        #returns fmeasure
        return (2*recall*precision)/(recall+precision)
```

```
def generateReport(self,Y_pred,Y_true):
    #generate report
    Pos_true_count  = 0
    Neg_true_count  = 0
    Pos_false_count = 0
    Neg_false_count = 0
    for i in range(len(Y_pred)):
        if Y_pred[i]==Y_true[i] and Y_true[i] == "Positive":
            Pos_true_count+=1
        elif Y_pred[i]==Y_true[i] and Y_true[i] =="Negative":
            Neg_true_count+=1
        elif Y_pred[i]!=Y_true[i] and Y_true[i] =="Positive":
            Pos_false_count+=1
        elif Y_pred[i]!=Y_true[i] and Y_true[i] =="Negative":
            Neg_false_count+=1;
    print("Report:")
    print("Accuracy:",round(self.accuracy(Pos_true_count,Neg_true_count,Pos_false_count,Neg_false_count),2))
    recall=self.recall(Pos_true_count,Neg_true_count,Pos_false_count,Neg_false_count)
    print("Recall:",round(recall,2))
    precision=self.precision(Pos_true_count,Neg_true_count,Pos_false_count,Neg_false_count)
    print("Precision:",round(precision,2))
    print("Fmeasure:",round(self.fmeasure(recall,precision),2))


def getWordVectorCount(self,X):
    # To represent test data as word vector counts
    X_dataset = np.zeros((len(X),len(feature.features)))
    # This can take some time to complete
    for i in range(len(X)):
        # print(i) # Uncomment to see progress
        word_list = [ word.strip(string.punctuation).lower() for word in X[i][1].split()]
        for word in word_list:
            if word in feature.features:
                X_dataset[i][feature.features.index(word)] += 1
    return X_dataset

def getCleanedWordVectorCount(self,check):
    X_check = np.zeros((len(check),len(feature.features)))
    # This can take some time to complete
    for i in range(len(check)):
        # print(i) # Uncomment to see progress
        word_list = [ word.strip(string.punctuation).lower() for word in check[i].split()]
        word_list = [word for word in word_list if word not in stopwords]
        for word in word_list:
            if word in feature.features:
                X_check[i][feature.features.index(word)] += 1
    return X_check
```

```
def predictClass(self,check):
    X_check=self.getCleanedWordVectorCount(check)
    Y_pred=self.predict(X_check)
    return Y_pred
```

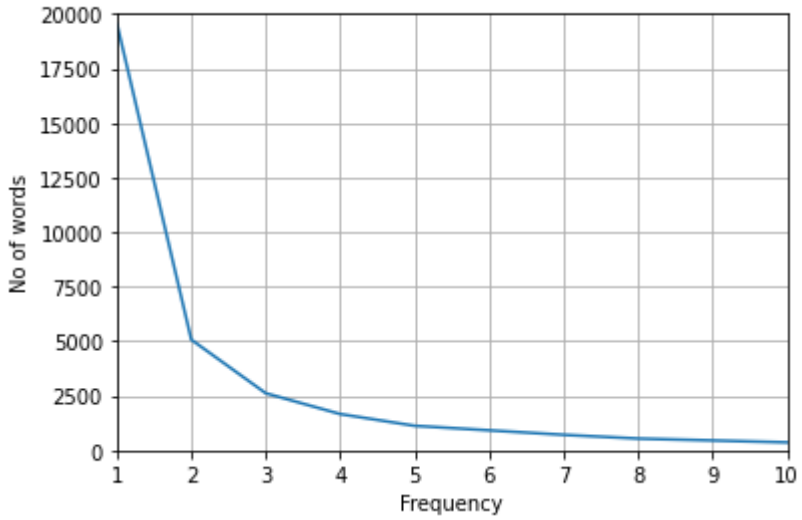
4.2.5 Building vocab

```
print("Creating Vocabulary!")
feature= Features()
feature.cleanBuildVocabulary(X_train) #building vocabulary
print("Vocabulary Created!")
print()

#feature.plotVocabulary()
feature.setCutOff(70)
print()

print("Creating Features!")
feature.buildFeatures()          #building features
print("Features Created!")
print()
```

4.2.6 Building & Testing Model

```
#building model
print("Creating model!")
clf2 = NaiveBayes(feature)
X_train_dataset=clf2.getWordVectorCount(X_train)
X_test_dataset=clf2.getWordVectorCount(X_test)
clf2.fit(X_train_dataset,Y_train)      #training model
print("Model Created!")
print()

print("Testing model!")
Y_test_pred = clf2.predict(X_test_dataset)
print("Model tested!")
print()

clf2.generateReport(Y_test_pred,Y_test)
print()


while True:
    check = []
    replay = input("Want to test on custom reviews? ").lower()
    if replay in ("yes", "y"):
        review = input("Enter a movie Review : ")
        check.append(review)
        print()
        print(clf2.predictClass(check))
    elif replay in ("no", "n"):
        break
    else:
        print ("Sorry, I didn't understand that.")

print("done!")
```

4.3 Results




```
In [19]: runfile('C:/Users/KRITANK SINGH/
Desktop/MINI_PROJECT/untitled2.py',
wdir='C:/Users/KRITANK SINGH/Desktop/
MINI_PROJECT')
Data size: 4000
Train Data size: 3000
Test Data size: 1000
Number of words with frequency higher than
cutoff frequency(10) : 4323
Report:
Accuracy: 0.78
Recall: 0.68
Precision: 0.85
Fmeasure: 0.75

Want to test on custom reviews? n
done!
```

```
In [7]: runfile('C:/Users/KRITANK SINGH/
Desktop/MINI_PROJECT/untitled2.py',
wdir='C:/Users/KRITANK SINGH/Desktop/
MINI_PROJECT')
Data size: 50000
Train Data size: 25000
Test Data size: 25000
Number of words with frequency higher than
cutoff frequency(80) : 3962
Report:
Accuracy: 0.8
Recall: 0.77
Precision: 0.81
Fmeasure: 0.79
```

```
wdir='C:/Users/KRITANK SINGH/Desktop/
MINI_PROJECT')
Data size: 4000
Train Data size: 3000
Test Data size: 1000
Number of words with frequency higher than
cutoff frequency(70) : 761
Report:
Accuracy: 0.84
Recall: 0.83
Precision: 0.84
Fmeasure: 0.83

Want to test on custom reviews? n
done!
```

We can supply custom reviews to be classified by our model

```
Want to test on custom reviews? y
Enter a movie Review : bad
['Negative']
Want to test on custom reviews? y
Enter a movie Review : good
['Positive']
Want to test on custom reviews? y
Enter a movie Review : wonderful
['Positive']
```

Chapter 5

Conclusion and Future Work

Using the above Multinomial Naive Bayes model on the **Stanford** dataset we achived a **maximum**:

1. accuracy of 0.84
2. recall of 0.83
3. precision of 0.84
4. f-measure of 0.83

We achived above figures on taking cut-off frequency = 70 (experimentally)

5.1 Source of Improvements

We can further improve on the accuracy by taking due considerations for hyperbole, sarcasm and thrawting. Our current model does not take into account for such factors thus its accuracy is limited at 0.84.

Appendix A

Some Complex Proofs and simple Results

A.1 Multinomial naïve Bayes

This is the event model typically used for document classification, with events representing the occurrence of a word in a single document (see bag of words assumption). The likelihood of observing a histogram x is given by

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i {p_{ki}}^{x_i}$$

The multinomial naïve Bayes classifier becomes a linear classifier when expressed in log-space:

$$\begin{aligned}\log p(C_k \mid \mathbf{x}) &\propto \log\left(p(C_k)\prod_{i=1}^{n} {p_{ki}}^{x_i}\right)\\ &= \log p(C_k) + \sum_{i=1}^{n} x_i \cdot \log p_{ki}\\ &= b + \mathbf{w}_k^\top \mathbf{x}\end{aligned}$$

where $b = \log p(C_k) b = \log p(C_k) and w_{ki} = \log p_{ki} w_{ki} = \log p_{ki}$.