**Sentiment Analysis of IMDB dataset**

**1) What is Sentiment Analysis**

A natural language processing tool used to describe and identify emotions in subjective data is sentiment analysis (or opinion mining). To identify sentiment in emails, survey responses, social media data, and beyond, sentiment analysis is also performed on textual data. Sentiment analysis is the method in which positive or negative emotions are detected in the text. Companies also use it to identify sentiment in social data, gauge brand credibility, and recognize clients.

Since customers express their thoughts and feelings more openly than ever before,

analysis sentiments is becoming an important instrument for monitoring and understanding feelings. Tracking consumer feedback automatically, such as impressions of survey response and interactions with social media helps advertisers to understand what makes consumers happy or unhappy, so that they can tailor products and services to meet their customers' needs.

**2) Types of Sentiment Analysis**

Sentiment analysis models focus on polarity (positive, negative, neutral) but also on feelings and emotions (angry, happy, sad, etc), urgency (urgent, not urgent) and even intentions (interested v. not interested). most popular types of sentiment analysis:

***Fine-grained Sentiment Analysis***

If polarity precision is important to business, we might consider expanding polarity categories to include:

• Very positive

• Positive

• Neutral

• Negative

• Very negative

This is usually referred to as fine-grained sentiment analysis, and could be used to interpret 5-star ratings in a review, for example:

• Very Positive = 5 stars

• Very Negative = 1 star

***Emotion detection***

This form of study of feelings attempts to detect feelings, such as happiness, annoyance, rage, sorrow, and so on. Lexicons (i.e. lists of words and the emotions they convey) or complicated machine learning algorithms are used by many emotion detection systems. One of the downsides of using lexicons is that individuals communicate emotions in various ways. Some words that usually express frustration, such as bad or kill (for example, your product is so bad or I'm killed by your customer support) can also express happiness.

Aspect-based Sentiment Analysis

Typically, when evaluating texts' emotions, let's say product reviews, we would want to know what basic things or attributes people discuss in a positive, neutral or negative way. For example, in this text: "The battery life of this camera is too short" an aspect-based classifier will be able to decide that the sentence expresses a negative opinion about the battery life function.

***.Multilingual sentiment analysis***

Analysis of multilingual opinion can be hard. It requires a lot of energy and preprocessing. Some of these tools (e.g. emotion lexicons) are accessible online, although some need to be developed (e.g. translated companies or algorithms for noise detection), but you'll need to know how to use them with code.

language of your choice.

**3) Benefits of Sentiment Analysis**

Sorting Data at Scale

Can you imagine manually sorting through thousands of tweets, customer support conversations, or surveys? There’s just too much business data to process manually. Sentiment analysis helps businesses process huge amounts of data in an efficient and cost-effective way.

Real-Time Analysis

Sentiment analysis can identify critical issues in real-time, for example is a PR crisis on social media escalating? Is an angry customer about to churn? Sentiment analysis models can help you immediately identify these kinds of situations, so you can take action right away.

Consistent criteria

It’s estimated that people only agree around 60-65% of the time when determining the sentiment of a particular text. Tagging text by sentiment is highly subjective, influenced by personal experiences, thoughts, and beliefs. By using a centralized sentiment analysis system, companies can apply the same criteria to all of their data, helping them improve accuracy and gain better insights.

**4) How does it work ?**

Natural Language Processing (NLP) and machine learning algorithms (basically rules) are the driving forces behind sentiment analysis.

Sentiment analysis algorithms fall into one of three buckets:

• Rule-based: these systems automatically perform sentiment analysis based on a set of manually crafted rules.

• Automatic: systems rely on machine learning techniques to learn from data.

• Hybrid systems combine both rule-based and automatic approaches.

***Rule Based Approach***

To help define subjectivity, polarity, or the subject of an opinion, a rulebased system typically uses a collection of humancrafted laws. Different NLP techniques developed in computer linguistics may include these laws, such as:

Stemming, tokenization, marking and parsing of part-of-speech.

Lexicons (i.e. lists of words and expressions).

Rulebased systems are very naive, as the way terms are mixed in a series is not taken into account. Of course it is possible to use more sophisticated processing methods and add new rules to support new phrases and vocabulary. Adding new rules, however can influence previous outcomes, nd the whole system may become very complex. Since rulebased systems also need finetuning and maintenance, frequent investment will also be required.

***Automatic Approach***

Automatic approaches do not rely on manually designed rules, but on machine learning techniques, compared to rule based systems. As a classification problem, a sentiment analysis task is typically modeled, whereby a classifier is fed a text and returns a category, e.g. positive, negative or neutral.

***Hybrid Approach***

Hybrid systems incorporate into one method the desirable elements of rule-based and automated techniques. The tremendous advantage of these methods is that the effects are always more precise.

**5) How Sentiment Analysis works?**

Graphical user interface, diagram

Description automatically generated

In the training phase (a), our model learns to connect a specific input (i.e.

a text) to the corresponding output (tag) based on the test samples used for training.

The extractor of features transfers the text input into a vector of features.

For creating a model, pairs of feature vectors and tags (e.g. positive, negative or neutral) are fed into the machine learning algorithm.

The function extractor is used in the prediction method

(b) to convert unseen text inputs into vectors of features.

 Then these function vectors are fed into the model, creating predicted tags

(again, positive, negative, or neutral).

#### Feature Extraction from Text

The first step in a text classifier for machine learning is to transform the text

extraction or vectorization of text, and with its frequency, the classical approach was bag-of-words or bag-of-ngrams.

More recently, new feature extraction techniques have been applied based on word embeddings (also known as *word vectors*). This kind of representations makes it possible for words with similar meaning to have a similar representation, which can improve the performance of classifiers.

Diagram

Description automatically generated

Data Cleaning : The data sometimes contains symbols, emoticons, short-forms typos etc. It is important to address such issues and clean our data before feeding it to the classifiers as it is a well-known Machine Learning saying if you feed garbage to models we will receive garbage as output

Tokenization :

* Assembly language (the language of Machines) is a language of numbers and is very different from human languages. To analyze texts written in human languages it is important to represent them in computer-recognizable form i.e. numbers. The computer further counts these numbers.  The process of grouping characters as tokens initiate the process of counting is known as Tokenization. Thus, tokenization prepares the data for counting. ​
* Consider the text :"Have a good day" the tokens for this text is ['Have','a','good','day']​

Vectorization:

* The processes of representing all the characters of a text document in the form of vectors is known as vectorization.Vectorization takes care of how we should proceed with counting the data.​
* Vectorization creates a vocabulary of all the tokens present in set of documents and further represents text as  numeric vector of all documents.​
* Types of Vectorizations:​
  + - Boolean Vectorizer​: The boolean vectorizer represents just the presence or absence of a word from vocabulary in the document​
    - Count Vectorizer​ : The count vectorizer represents the number of times a word from vocabulary appears in the document​
* TfIDF Vectorizer​: The TFIDF(term frequency-inverse document frequency) vectorizer stands for is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.​
  + - This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents​
    - TF-IDF (term frequency-inverse document frequency) was invented for document search and information retrieval. It works by increasing proportionally to the number of times a word appears in a document, but is offset by the number of documents that contain the word.​
    - So, words that are common in every document, such as this, what, and if, rank low even though they may appear many times, since they don’t mean much to that document in particular.​

Classification Algorithms

The classification step usually involves a statistical model like Naïve Bayes, Logistic Regression, Support Vector Machines, or Neural Networks:

* Naïve Bayes: a family of probabilistic algorithms that uses Bayes’s Theorem to predict the category of a text.
* Linear Regression: a very well-known algorithm in statistics used to predict some value (Y) given a set of features (X).
* Support Vector Machines: a non-probabilistic model which uses a representation of text examples as points in a multidimensional space. Examples of different categories (sentiments) are mapped to distinct regions within that space. Then, new texts are assigned a category based on similarities with existing texts and the regions they’re mapped to.
* Deep Learning: a diverse set of algorithms that attempt to mimic the human brain, by employing artificial neural networks to process data.

**6) Analysis on IMDB dataset**

About the data

* The data used for this task is IMDB movie review dataset with 45000 rows as training data and 4995 rows in the training dataset with two columns text and label.​
* The target label contains values 0 and 1 represents negative sentiment while 1 represents positive sentiment ​
* The data is then split in 7/3 ratio for training and validation .i.e.  31500 rows for training and 13500 rows for testing ​
* 50% of data contained label 0 and 50% contained label 1.

Distribution of number of words complete data set

Chart, histogram

Description automatically generated

Distribution of number of words in Train split dataset

Chart, histogram

Description automatically generated

Distribution of number of words in Validation split dataset

Chart, histogram

Description automatically generated

We can see the distribution maintains it shape throughout the split and thus, we can say we have covered all types of data in training our models and also validating them,

***Classification Algorithms and the results***

*Naïve Bayes Algorithm*

**Multinomial Naïve Bayes**

Multinomial Naïve Bayes is one of the methods to train models for text categorization/ classification. It is based on the Bayes theorem which helps us compute the conditional probabilities of a word in text. It is referred to as Naïve as it assumes that the occurrence of each word in the text document is independent of the occurrence of other words, which is not really possible. But still this approach seems to work quite well. The multinomial naïve Bayes calculates the prior probabilities of each class and conditional probabilities of each word given each class to determine the posterior probabilities. The general formulas for these probabilities are as follows:

Prior Probability: Probability(class)

Conditional Probability (for Multinomial Naïve Bayes): P(w|class) = nw +1 / n + v

Representations of variables:

w- represents a word from the text in the document

class – the class of the given text in the document

nw – The total number of times the word w occurs in this class of text

n – Total number of words in the class of text

v – vocabulary(number of distinct words in the whole document irrespective of their class)

**Benoulli Naïve Bayes**

As the name suggests it is another method based on Bayes theorem to train models for text categorization/classification. The Benoulli naïve Bayes calculates the prior probabilities of each class and conditional probabilities of each word given each class to determine the posterior probabilities. The general formulas for these probabilities are as follows:

Prior Probability: Probability(class)

Conditional Probability (for Benoulli Naïve Bayes): P(w|class) = nw +1 / n + cn

Suppose the vocabulary has words w1, w2,w3, w4 and we want to classify a sentence with words w2 and w3.

Posterior Probability = Prior Probability \* P(w2|class1) \*P(w3|class1) \*P(1- w1|class1) \*P(1- w4|class1)

nw – The total number of texts in this class that contain word w

n – Total number of texts in this class

cn – Total number of classes.

As we can see unlike multinomial model this model uses the presence/absence of word while calculating conditional probabilities.

**Input to the two models**

The Benoulli Naïve Bayes accepts only the Boolean representation of the vectorized text documents. While on the other hand Multinomial Naïve Bayes accepts word frequencies or tfidf..

***Multinomial Naïve Bayes***

The data was split in 70:30 ratio for training the model and testing it. The training and testing set of input string data was vectorize using term frequency vectorizer, Frequency vectorizer with n-gram tokenizer with max range 2 and min range 1 and tfidf vectorizer. It is important to note here that all the stop words were removed. In python this was achieved through using CountVectorizer() and TfidfVectorizer() objects. The reason for avoiding the use of Boolean Vectorizer is the approach that Multinomial Naïve Bayes algorithm uses to calculate posterior probabilities. Multinomial Naïve Bayes estimates the probability of the event that one of the N unique words occurs in a position and thus, we don’t use the Boolean Vectorizer as it woudn’t be a good fit.

***Benoulli Naïve Bayes***

The data was split in 70:30ratio for training the model and testing it. The training and testing set of input string data was vectorized using Boolean Vectorizer and n-gram tokenizer without stop words and also using Boolean Vectorizer and word tokenizer without and with stopwords. The Benoulli Naïve Bayes calculates the probabilities based on the presence and absence of words in a class and thus we use Boolean vectorizer for this model.

For all the classification reports shared below the boolen vectorizer represents the use of Bernoulli Naïve Bayes while count and TFIDF Vectorizer represent the use of Multinomial Naïve Bayes.

Vectorizer: Boolean Vectorizer

Tokenizer: unigram

Classification Report :

Table

Description automatically generated

Vectorizer: Count Vectorizer

Tokenizer: unigram

Classification Report :

Table

Description automatically generated

Vectorizer: TFIDF Vectorizer

Tokenizer: unigram

Classification Report :

Table

Description automatically generated

We can see the best performing model with unigram tokenizer was Frequency(Count) vectorizer with accuracy of 86% and precision of almost 88% for label 1 and 84% for label 0

Vectorizer: Boolean Vectorizer

Tokenizer: biigram

Classification Report :

Table

Description automatically generated

Vectorizer: Count Vectorizer

Tokenizer: biigram

Classification Report :

Table

Description automatically generated

Vectorizer: TFIDF Vectorizer

Tokenizer: biigram

Classification Report :

Table

Description automatically generated

We can see introducing bigrams improves model accuracy and precision.

However, the best performing model for Naïve Bayes turns out to be Bernoulli Naïve Bayes with bigram Tokenizer and Boolean Vectorizer. It gives us accuracy of 88%and precision of almost 88% for both classes.

**SVM Classifier**

Usually, Linear SVM is used to do text based classifications. For this project I have experimented with different values of C represents the regularization parameter greater the value of C less the strength of regularization. It should strictly be positive.

Tokenizer: unigram

C : 0.05

Classification Report :

A picture containing text, receipt

Description automatically generated

Tokenizer: unigram

C : 0.1

Classification Report :

A picture containing text, receipt

Description automatically generated

Tokenizer: unigram

C : 0.5

Classification Report :

A picture containing text, receipt

Description automatically generated

Tokenizer: unigram

C : 1

Classification Report :

A picture containing text, receipt

Description automatically generated

We can see for all values of C TFIDF Vectorizer does a better job than the Frequency/Count Vectorizer. The highest accuracy with unigram tokenizer was obtained with C=0.1 and C=1 which was approximately 90%. The precision for label 1 by both the classifiers was same 89% however, c=0.1 did a better a slightly better job with precision for label 0 and thus, it can be seen as the best model for unigram tokenizer.

Tokenizer: bigram

C : 0.05

Classification Report :

A picture containing text, receipt

Description automatically generated

Tokenizer: bigram

C : 0.1

Classification Report :

A picture containing text, receipt

Description automatically generated

Tokenizer: bigram

C : 0.5

Classification Report :

A picture containing text, receipt

Description automatically generated

Tokenizer: bigram

C : 1

Classification Report :

A picture containing text, receipt

Description automatically generated

For bigram tokenizer almost all models did gave accuracies around 90% and precision around 90 % for both labels. However, the precision and accuracy of best performing model for bigram (C=1,TFIDF vectorizer) was same as the best performing model for unigram vectorizer. Additionally, bigrams are computationally expensive as they increase the number of features. **Thus overall best performing model remains SVM with unigram tokenizer and with C=0.1.**

**Word Clouds**

**Positive Sentences**

A picture containing text, newspaper

Description automatically generated

As we know the dataset is IMDB movie review dataset the most used word in positive sentences is words related to movie time, story etc. However, we can see use of some positive words such as great, love, good, well etc

**Negative Sentences**

A picture containing text, newspaper, accessory

Description automatically generated

Again for negative sentences the most used words were the ones related to movies however there are not many negative words use. This tells us that absence of positive words has been marked as negative sentiment in our dataset.