

Performing Sentiment Analysis Using Natural Language Processing

A

Major Project submitted in partial fulfillment of the
requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE ENGINEERING

By

ADARSH PERI (16841A0565)

YUGENDER CHAUHAN (16841A05C9)

GOWTHAM VARUN (16841A05A8)

Under the esteemed guidance of

MS. K. PRANUSHA

(Assistant Professor, Dept. of CSE)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AURORA'S TECHNOLOGICAL AND RESEARCH INSTITUTE

(Affiliated to Jawaharlal Nehru Technological University, Hyderabad)

Parvathapur, Uppal, Hyderabad-500 039

(2019-20)



Aurora's Technological And Research Institute
(Affiliated to Jawaharlal Nehru Technological University, Hyderabad)
Parvathapur, Uppal, Hyderabad-500 039
(2019-20)

DECLARATION

We hereby declare that the work described in this project, entitled '**Natural Language Processing**' which is being submitted by us in partial fulfillment for the award of Bachelor of Technology in Computer Science and Engineering to **Jawaharlal Nehru Technological University** is a record of bonafide work carried by us under the guidance of **Ms. K.Pranusha, Assistant Professor, Dept. of CSE.**

The results embodied in this major project report have not been submitted to any other University or Institute for the award of any Degree.

Place: Hyderabad

Date:

ADARSH PERI
16841A0565

YOUGENDER CHAUHAN
16841A05C9

GOWTHAM VARUN
16841A05A8



Aurora's Technological And Research Institute
(Affiliated to Jawaharlal Nehru Technological University, Hyderabad)
Parvathapur, Uppal, Hyderabad-500 039
(2019-20)

CERTIFICATE

This is to certify that the major project report entitled “**Natural Language Processing**” that is being submitted by **Adarsh Peri, Yougender Chauhan**, and **Gowtham Varun** bearing Roll No.s **16841A0565, 16841A05C9**, and **16841A05A8**, in partial fulfillment for the award of the Degree of Bachelor of Technology in **Computer Science and Engineering** to the Jawaharlal Nehru Technological University is a record of bonafide work carried out by them under my guidance and supervision.

The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma.

Date:

Internal Guide

Ms. K.Pranusha
Assistant Professor
Dept. of CSE

Head of the department

Ms. A. Durga Pavani
Dept. of CSE

Project Coordinator

Dr. S. Venkatesan
Professor
Dept. of CSE

Director

Mr. Srikanth Jatla

External Examiner

ACKNOWLEDGEMENT

This work has been done during the project period and it was a very good opportunity to put theoretical knowledge into planned exercise with an aim to solve a real time problem and also to develop a confidence to face various practical situations.

We would also like to express our gratitude to **Mr. Srikanth Jatla, Director, Aurora's Technological and Research Institute** for providing us congenial atmosphere and encouragement.

We express our sincere thanks to Head of the Department **Ms. A. Durga Pavani** for giving us the support and her kind attention and valuable guidance to us throughout this course.

We express our sincere thanks to Project Coordinator **Dr S. Venkatesan** for helping us complete our project work by giving valuable suggestions.

We convey thanks to our project guide **Ms. K.Pranusha**, Department of Computer Science and Engineering, for providing encouragement, constant support and guidance which was of a great help to complete this project successfully.

ASBTRACT

Modern organizations work with huge amounts of data. That data can come in a variety of different forms including documents, spreadsheets, audio recordings, emails, JSON, and so many, many more. One of the most common ways that such data is recorded is via text. That text is usually quite similar to the natural language that we use from day-to-day.

Natural Language Processing(NLP) is the study of programming computers to process and analyze large amounts of natural textual data. Knowledge of NLP is essential for Data Scientists since text is such an easy to use and common container for storing data.

Faced with the task of performing analysis and building models from textual data, one must know how to perform the basic Data Science tasks. That includes cleaning, formatting, parsing, analyzing, visualizing, and modelling the text data. It'll all require a few extra steps in addition to the usual way these tasks are done when the data is made up of raw numbers.

Natural Language Processing (NLP) is the capacity of a computer to "understand" natural language text at a level that allows meaningful interaction between the computer and a person working in a particular application domain.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
1	INTRODUCTION	
	1.1 Data Science	1
	1.2 Deep Learning	2
	1.3 Natural Language Processing	3
	1.4 Machine Learning	4
2	REQUIREMENTS SPECIFICATION	
	2.1 Software Requirements	11
	2.2 Hardware Requirements	11
3	LITERATURE SURVEY	
	3.1 Problem Identification and Solutions	12
4	SOFTWARE REQUIREMENTS ANALYSIS	
	4.1 Problem Definition	15
	4.2 Modules	16
5	SOFTWARE DESIGN	
	5.1 System Architecture	18
6	CODE TEMPLATES	
	6.1 Creating Feature sets and Classifiers	30
	6.2 Voter Classifier	39
	6.3 Sentiment Analysis	43
7	TESTING	
	7.1 Testing methodologies	45
	7.2 Test Cases	49
8	OUTPUT SCREENS	54
9	CONCLUSION	56
10	BIBLIOGRAPHY	57

LIST OF FIGURES

FIG.NO.	FIGURE NAME	PAGE NO.
1.1	Different Processes in Data Science	1
1.2	Linear Regression	4
1.3	Using Naive Bayes	6
1.4	Logistic Regression	7
1.5	K-Means Algorithm	9
5.1	System Architecture	18
5.2	NLP Pipeline	18
5.3	Parts of Speech Recognition	21
5.4	After processing POS	21
5.5	Lemmatization	22
5.6	Stop Words Identification	23
5.7	Dependency Parsing part 1	24
5.8	Dependency Parsing part 2	24
5.9	Noun Phrases part 1	26
5.10	Noun Phrases part 2	50
5.11	Nouns	27
5.12	Entities	27
5.13	Coreference	28
8.1	Accuracy Scores of Classifiers	54
8.2	Sentiment Analysis Part 1	54
8.3	Sentiment Analysis Part 2	55
8.4	Sentiment Analysis Part 3	55

1.INTRODUCTION

1.1 Data Science

Data is the new Oil. This statement shows how every modern IT system is driven by capturing, storing and analyzing data for various needs. Be it about making decision for business, forecasting weather, studying protein structures in biology or designing a marketing campaign. All of these scenarios involve a multidisciplinary approach of using mathematical models, statistics, graphs, databases and of course the business or scientific logic behind the data analysis. So, we need a programming language which can cater to all these diverse needs of data science. Python shines bright as one such language as it has numerous libraries and built in features which makes it easy to tackle the needs of Data science.

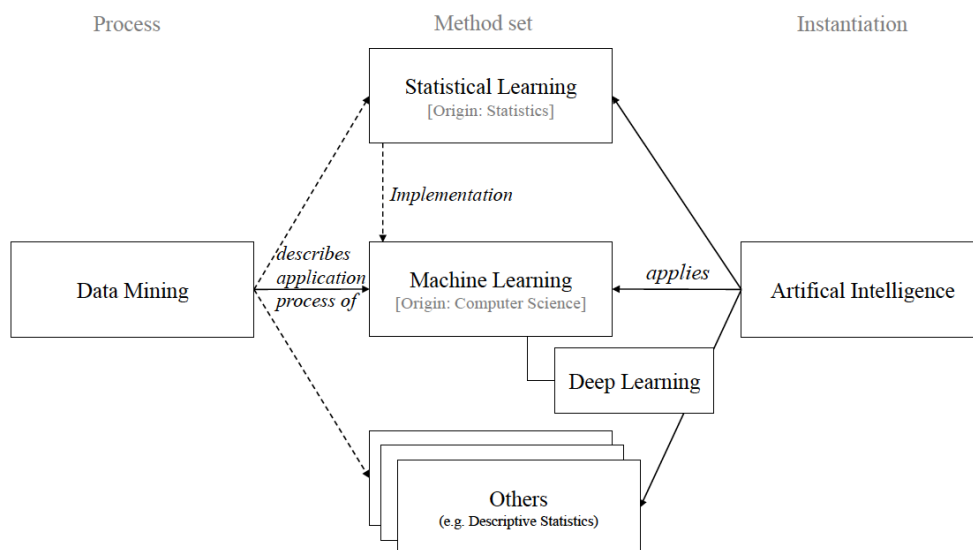


Fig 1.1 Different Processes involving Data Science

1.2 Deep Learning

Deep learning is a form of machine learning that can utilize either supervised or unsupervised algorithms, or both. While it's not necessarily new, deep learning has recently seen a surge in popularity as a way to accelerate the solution of certain types of difficult computer problems, most notably in the computer vision and natural language processing (NLP) fields.

Neural networks can have many hidden layers. Deep learning is based on the representation learning (or feature learning) branch of machine learning theory.

By extracting high-level, complex abstractions as data representations through a hierarchical learning process, deep learning models yield results more quickly than standard machine learning approaches. In plain English, a deep learning model will learn the features that are important by itself, instead of requiring the data scientist to manually select the pertinent features, such as the pointedness of ears found in cat pictures.

The “deep” in deep learning comes from the many layers that are built into the deep learning models, which are typically neural networks. A convolutional neural network (CNN) can be made up of many, many layers of models, where each layer takes input from the previous layer, processes it, and outputs it to the next layer, in a daisy-chain fashion.

It was a CNN developed by Google's DeepMind team that famously beat the human world champion of the ancient Chinese game of Go, which many saw as a sign of deep learning's ascendance.

1.3 Natural Language Processing

Modern organizations work with huge amounts of data. That data can come in a variety of different forms including documents, spreadsheets, audio recordings, emails, JSON, and so many, many more. One of the most common ways that such data is recorded is via text. That text is usually quite similar to the natural language that we use from day-to-day. Natural Language Processing(NLP) is the study of programming computers to process and analyze large amounts of natural textual data. Knowledge of NLP is essential for Data Scientists since text is such an easy to use

and common container for storing data. Faced with the task of performing analysis and building models from textual data, one must know how to perform the basic Data Science tasks. That includes cleaning, formatting, parsing, analyzing, visualizing, and modelling the text data. It'll all require a few extra steps in addition to the usual way these tasks are done when the data is made up of raw numbers. Natural Language Processing (NLP) is the capacity of a computer to "understand" natural language text at a level that allows meaningful interaction between the computer and a person working in a particular application domain.

1.4 Machine Learning

At its most basic level, machine learning refers to any type of computer program that can “learn” by itself without having to be explicitly programmed by a human. The phrase (and its underlying idea) has its origins decades ago – all the way to Alan Turing’s seminal 1950 paper “Computing Machinery and Intelligence,” which featured a section on his famous “Learning Machine” that could fool a human into believing that it’s real. Today, machine learning is a widely used term that encompasses many types of programs that you’ll run across in big data analytics and data mining. At the end of the day, the “brains” actually powering most predictive programs – including spam filters, product recommenders, and fraud detectors — are machine learning algorithms.

1.4.1 Types of ML

1.4.1.1 Supervised Learning

In **supervised** learning, the user trains the program to generate an answer based on a known and labeled data set. Classification and regression algorithms, including random forests, decision trees, and support vector machines, are commonly used for supervised learning tasks.

1.4.1.1.1 Regression Analysis

Regression Analysis is a statistical process for estimating the relationships between the dependent variables or criterion variables and one or more independent variables or predictors. Regression analysis explains the changes in criteria in relation to changes in select predictors. The conditional expectation of the criteria based on predictors where the average value of the dependent variables is given when the independent variables are changed. Three major uses for regression analysis are determining the strength of predictors, forecasting an effect, and trend forecasting.

Different Types of Regressions are

1.4.1.1.1.1 Simple Linear Regression

In machine learning, we have a set of input variables (x) that are used to determine an output variable (y). A relationship exists between the input variables and the output variable. The goal of ML is to quantify this relationship.

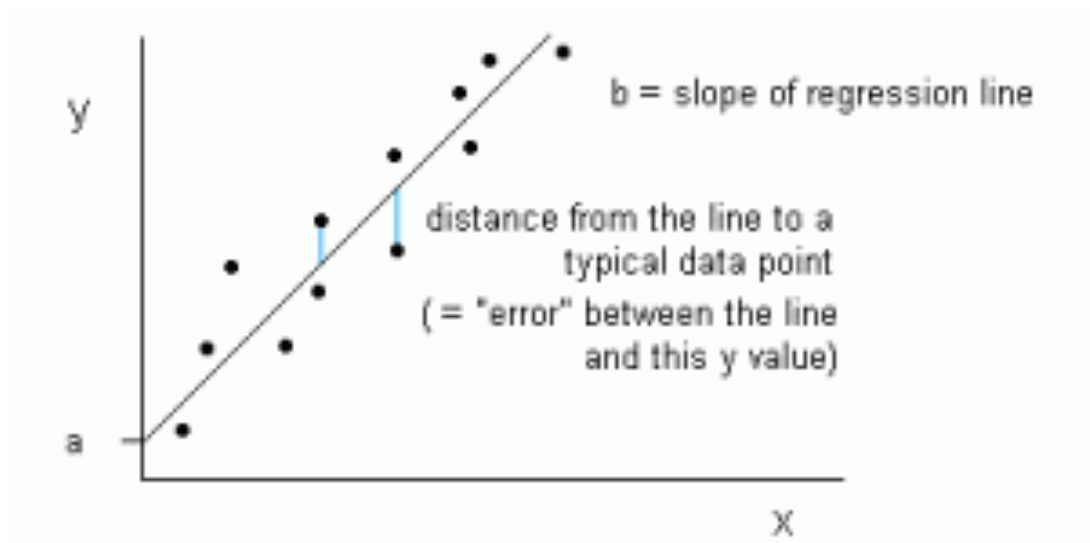


Figure 1.2 Linear Regression is represented as a line in the form of $y = a + bx$.

In Linear Regression, the relationship between the input variables (x) and output variable (y) is expressed as an equation of the form $y = a + bx$. Thus, the goal of linear regression is to find out the values of coefficients a and b . Here, a is the intercept and b is the slope of the line.

Figure 1 shows the plotted x and y values for a data set. The goal is to fit a line that is nearest to most of the points. This would reduce the distance ('error') between the y value of a data point and the line.

1.4.1.1.2 Multiple Linear Regression

Multiple Linear Regression is the most common form of linear regression analysis. As a predictive analysis, the multiple linear regression is used to explain the relationship between one continuous dependent variable and two or more independent variables. The independent variables can be continuous or categorical (dummy coded as appropriate).

1.4.1.1.2 Classification

Classification is a data-mining technique that assigns categories to a collection of data to aid in more accurate predictions and analysis. Classification is one of several methods intended to make the analysis of very large datasets effective.

Different types of classifiers are

1.4.1.1.2.1 Naive Bayes

To calculate the probability that an event will occur, given that another event has already occurred, we use Bayes's Theorem. To calculate the probability of hypothesis(h) being true, given our prior knowledge(d), we use Bayes's Theorem as follows:

$$P(h|d) = (P(d|h) P(h)) / P(d)$$

where:

- $P(h|d)$ = Posterior probability. The probability of hypothesis h being true, given the data d, where $P(h|d) = P(d_1|h) P(d_2|h) \dots P(d_n|h) P(d)$

- $P(d|h)$ = Likelihood. The probability of data d given that the hypothesis h was true.
- $P(h)$ = Class prior probability. The probability of hypothesis h being true (irrespective of the data)
- $P(d)$ = Predictor prior probability. Probability of the data (irrespective of the hypothesis)

This algorithm is called 'naive' because it assumes that all the variables are independent of each other, which is a naive assumption to make in real-world examples.

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

Figure 1.3: Using Naive Bayes to predict the status of 'play' using the variable 'weather'.

Using Figure 1.3 as an example, what is the outcome if weather = 'sunny'?

To determine the outcome play = 'yes' or 'no' given the value of variable weather = 'sunny', calculate $P(\text{yes}|\text{sunny})$ and $P(\text{no}|\text{sunny})$ and choose the outcome with higher probability.

$$\rightarrow P(\text{yes}|\text{sunny}) = (P(\text{sunny}|\text{yes}) * P(\text{yes})) / P(\text{sunny}) = (3/9 * 9/14) / (5/14) = 0.60$$

$$\rightarrow P(\text{no}|\text{sunny}) = (P(\text{sunny}|\text{no}) * P(\text{no})) / P(\text{sunny}) = (2/5 * 5/14) / (5/14) = 0.40$$

Thus, if the weather = 'sunny', the outcome is play = 'yes'.

1.4.1.1.2.2 Logistic Regression

Linear regression predictions are continuous values (i.e., rainfall in cm), logistic regression predictions are discrete values (i.e., whether a student passed/failed) after applying a transformation function. Logistic regression is best suited for binary classification: data sets where $y = 0$ or 1 , where 1 denotes the default class. For example, in predicting whether an event will occur or not. Logistic regression is named after the transformation function it uses, which is called the logistic function $h(x) = 1 / (1 + e^{-x})$. This forms an S-shaped curve.

In logistic regression, the output takes the form of probabilities of the default class (unlike linear regression, where the output is directly produced). As it is a probability, the output lies in the range of 0-1. So, for example, if we're trying to predict whether patients are sick, we already know that sick patients are denoted as 1 , so if our algorithm assigns the score of 0.98 to a patient, it thinks that patient is quite likely to be sick. This output (y-value) is generated by log transforming the x-value, using the logistic function $h(x) = 1 / (1 + e^{-x})$. A threshold is then applied to force this probability into a binary classification.

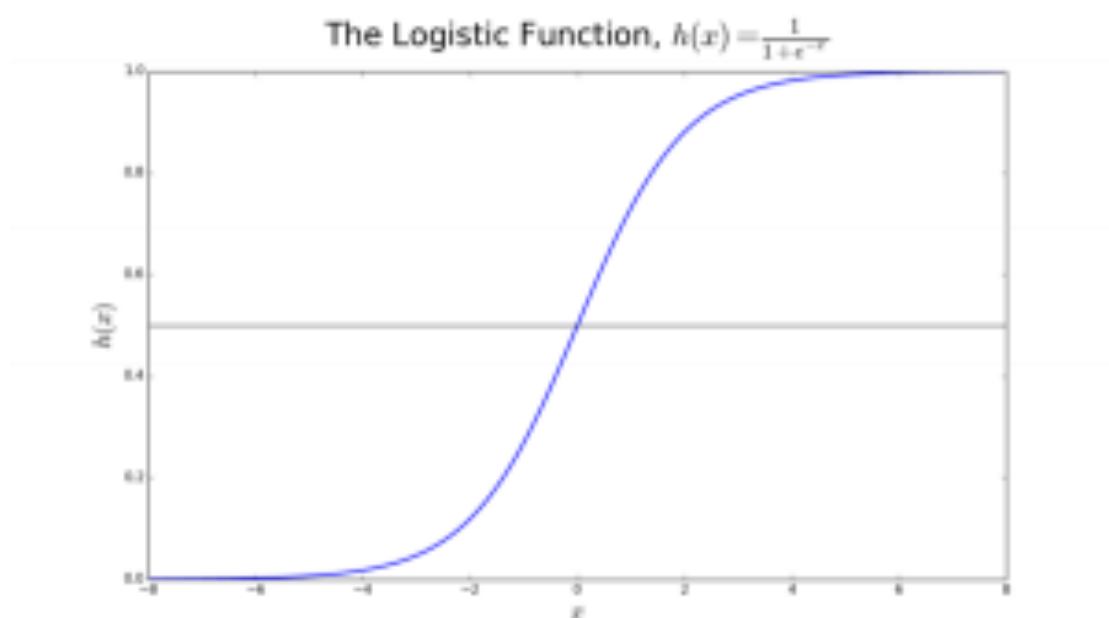


Figure 1.4: Logistic Regression to determine if a tumour is malignant or benign. Classified as malignant if the probability $h(x) \geq 0.5$.

In Figure 1.4, to determine whether a tumour is malignant or not, the default variable is $y = 1$ (tumour = malignant). The x variable could be a measurement of the tumour, such as the size of the tumour. As shown in the figure, the logistic function transforms the x -value of the various instances of the data set, into the range of 0 to 1. If the probability crosses the threshold of 0.5 (shown by the horizontal line), the tumour is classified as malignant.

The logistic regression equation $P(x) = e^{(b_0 + b_1x)} / (1 + e^{(b_0 + b_1x)})$ can be transformed into $\ln(p(x) / 1-p(x)) = b_0 + b_1x$.

1.4.1.2 Unsupervised

In **unsupervised** machine learning, the algorithms generate answers on unknown and unlabeled data. Data scientists commonly use unsupervised techniques for discovering patterns in new data sets. Clustering algorithms, such as K-means, are often used in unsupervised machine learning.

Data scientists can program machine learning algorithms using a range of technologies and languages, including Java, Python, Scala, and others. They can also use pre-built machine learning frameworks to accelerate the process; Mahout is an example of a machine learning framework that was popular on Apache Hadoop, while Apache Spark's MLlib library today has become a standard.

Data scientists are expected to be familiar with the differences between supervised machine learning and unsupervised machine learning — as well as ensemble modelling, which uses a combination of approaches techniques, and semi-supervised learning, which combines supervised and unsupervised approaches.

1.4.1.2.1 K-means

K-means is an iterative algorithm that groups similar data into clusters. It calculates the centroids of k clusters and assigns a data point to that cluster having least distance between its centroid and the data point.

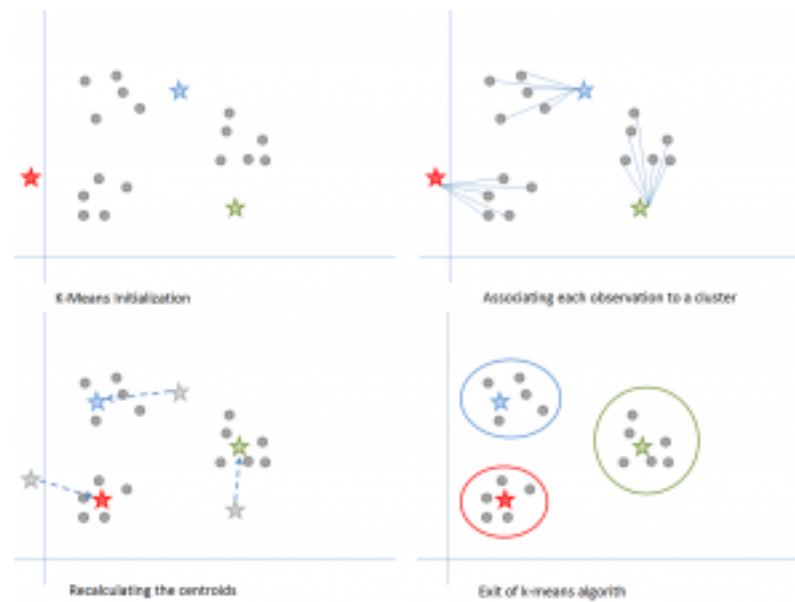


Figure 1.5: Steps of the K-means algorithm.

Here's how it works:

We start by choosing a value of k . Here, let us say $k = 3$. Then, we randomly assign each data point to any of the 3 clusters. Compute cluster centroid for each of the clusters. The red, blue and green stars denote the centroids for each of the 3 clusters.

Next, reassign each point to the closest cluster centroid. In the figure above, the upper 5 points got assigned to the cluster with the blue centroid. Follow the same procedure to assign points to the clusters containing the red and green centroids. Then, calculate centroids for the new clusters. The old centroids are gray stars; the new centroids are the red, green, and blue stars.

Finally, repeat steps 2-3 until there is no switching of points from one cluster to another. Once there is no switching for 2 consecutive steps, exit the K-means algorithm.

1.4.1.2.2 Time series Classification

Time series classification deals with classifying the data points over the time based on its' behavior. There can be data sets which behave in an abnormal manner when comparing with other data sets. Identifying unusual and anomalous time series is becoming increasingly common for organizations. It is a must for an organization to identify abnormal behaviours in order to make strong business decisions and market predictions. As an example, huge business industries such as Yahoo monitor their mail servers over time in order to detect anomalies and malicious time series. In this case, Feature Extraction can be used as a methodology for time series classification.

2. REQUIREMENTS SPECIFICATION

2.1 Software Requirements

Programming language	Python 3.5
Operating System	Mac, Linux, Windows

2.2 Hardware Requirements

Processor	Intel i5 Processor
RAM	4GB
Disk Space	128GB

All the above requirements are the bare minimum requirements to run the project ideally.

3. LITERATURE SURVEY

Sarcasm occurring due to the presence of numerical portions in text has been quoted as an error made by automatic sarcasm detection approaches in the past. The paper **“Having 2 hours to write a paper is fun”: Detecting sarcasm in Numerical Portions of Text** by Lakshya Kumar, Arpan Somani, Pushpak Bhattacharya(IIT Bombay) presents a first study in detecting sarcasm in numbers, as in the case of the sentence ‘Love waking up at 4 am’. We analyze the challenges of the problem, and present Rule- based, Machine Learning and Deep Learning approaches to detect sarcasm in numerical portions of text. The proposed Deep Learning approach outperforms four past works for sarcasm detection and Rule-based and Machine learning approaches on a dataset of tweets, obtaining an F1-score of 0.93. This shows that special attention to text containing numbers may be useful to improve state-of-the-art in sarcasm detection.

Precise semantic representation of a sentence and definitive information extraction are key steps in the accurate processing of sentence meaning, especially for figurative phenomena such as sarcasm, Irony, and metaphor cause literal meanings to be discounted and secondary or extended meanings to be intentionally profiled. Semantic modelling faces a new challenge in social media, because grammatical inaccuracy is commonplace yet many previous state-of-the-art methods exploit grammatical structure. For sarcasm detection over social media content, researchers so far have counted on Bag-of-Words(BOW), N-grams etc. In **Fracking Sarcasm using Neural Networks** - Aniruddha Ghosh, Tony Veale (Uni. of Dublin) a neural network semantic model for the task of sarcasm detection is proposed to deal with the same. The proposed neural network model composed of Convolution Neural Network(CNN) and followed by a Long short term memory (LSTM) network and finally a Deep neural network(DNN). The proposed model outperforms state-of-the-art text- based methods for sarcasm detection, yielding an F-score of .92.

Sentiment analysis is the process to study of people opinion, emotion and way of considering a matter and take decision into different categorizes like positive, negative and neutral in data mining. The sentiment analysis is used to find out negation within the text using Natural Language Processing rules. The aim of **Effect of negation in sentiment analysis - Wareesa Sharif, Rashid Naseem (IEEE)** is to detect negation affect on consumer reviews which looks like positive but exactly negative in sense. A number of different approaches have been used, but these approaches do not provide efficient and appropriate way of calculating negation sense in sentiment analysis. The proposed modified negation approach presents a way of calculating negation identification and is helpful to improve classification accuracy. Main achievement of this approach is that it is helpful for calculating the negation in sentiment analysis without the words not, no, n't, never etc. This method produced a significant result for review classification by accuracy, precision and recall.

Sentiment analysis is an important task in natural language understanding and has a wide range of real-world applications. The typical sentiment analysis focus on predicting the positive or negative polarity of the given sentence(s). This task works in the setting that the given text has only one aspect and polarity. A more general and complicated task would be to predict the aspects mentioned in a sentence and the sentiments associated with each one of them. This generalized task is called aspect-based sentiment analysis (ABSA). In the annual SemEval competition, an ABSA task has been added since 2014.

Riding on the recent trends of deep learning, **Deep Learning for aspect-based analysis - Bo Wang, Min Liu (Stanford University)** applies deep neural nets to solve this task. The authors design a combined model with aspect prediction and sentiment prediction. For both predictions, we achieve better than or close to state-of-the-art performance using deep learning models. The paper also proposes a new method to combine the syntactic structure and convolutional neural nets to directly match aspects and corresponding polarities.

3.1 Problem Identification and Existing Solutions

3.1.1 Dealing with multipolarity

“The audio quality of my new laptop is so cool but the display colours are not too good.” Some sentiment analysis models will assign a negative or a neutral polarity to this sentence. To deal with such situations, a sentiment analysis model must assign a polarity to each aspect in the sentence; here, “audio” is an aspect assigned a positive polarity and “display” is a separate aspect with a negative polarity.

3.1.2 Dealing with Negations

The simplest approach for dealing with negation in a sentence, which is used in most state-of-the-art sentiment analysis techniques, is marking as negated all the words from a negation cue to the next punctuation token. The effectiveness of the negation model can be changed because of the specific construction of language in different contexts.

3.1.3 Dealing with Numerical Sarcasm

Ghosh and Veale in their 2016 paper use a combination of a convolutional neural network, a long short-term memory (LSTM) network, and a DNN. They compare their approach against recursive support vector machines (SVMs) and conclude that the deep learning architecture is an improvement over such approaches.

4. SOFTWARE REQUIREMENT ANALYSIS

4.1 Problem Definition

Natural Language Processing(NLP) is the study of programming computers to process and analyze large amounts of natural textual data. Knowledge of NLP is essential for Data Scientists since text is such an easy to use and common container for storing data.

Faced with the task of performing analysis and building models from textual data, one must know how to perform the basic Data Science tasks. That includes cleaning, formatting, parsing, analyzing, visualizing, and modelling the text data. It'll all require a few extra steps in addition to the usual way these tasks are done when the data is made up of raw numbers.

Natural Language Processing (NLP) is the capacity of a computer to "understand" natural language text at a level that allows meaningful interaction between the computer and a person working in a particular application domain. Modern organizations work with huge amounts of data. That data can come in a variety of different forms including documents, spreadsheets, audio recordings, emails, JSON, and so many, many more. One of the most common ways that such data is recorded is via text. That text is usually quite similar to the natural language that we use from day-to-day.

Given a dataset of movie reviews, our aim is to use an unsupervised machine learning algorithm, NLP(Natural Language Processing), alongside a supervised ML algorithm, Naive Bayes, to classify whether the given review is good or bad.

For example: The food is very good. (O/P: 1 (good))

I would not visit this place again. (O/P: 0(bad))

4.2 Modules

4.2.1 NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

4.2.2 Pickle

Python pickle module is used for serializing and de-serializing a Python object structure. Any object in Python can be pickled so that it can be saved on disk. What pickle does is that it “serializes” the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.

4.2.3 NLTK

The NLTK module is a massive tool kit, aimed at helping you with the entire Natural Language Processing (NLP) methodology. NLTK will aid you with everything from splitting sentences from paragraphs, splitting up words, recognizing the part of speech of those words, highlighting the main subjects, and then even with helping your machine to understand what the text is all about.

4.2.4 RE

A *regular expression* is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are widely used in UNIX world.

4.2.5 Sklearn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. The library is built upon the SciPy. The library is focused on modelling data. It is not focused on loading, manipulating and summarizing data. For these features we use NumPy and Pandas.

4.2.6 Random

This module implements pseudo-random number generators for various distributions. For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.

5. SOFTWARE DESIGN

5.1 System Architecture

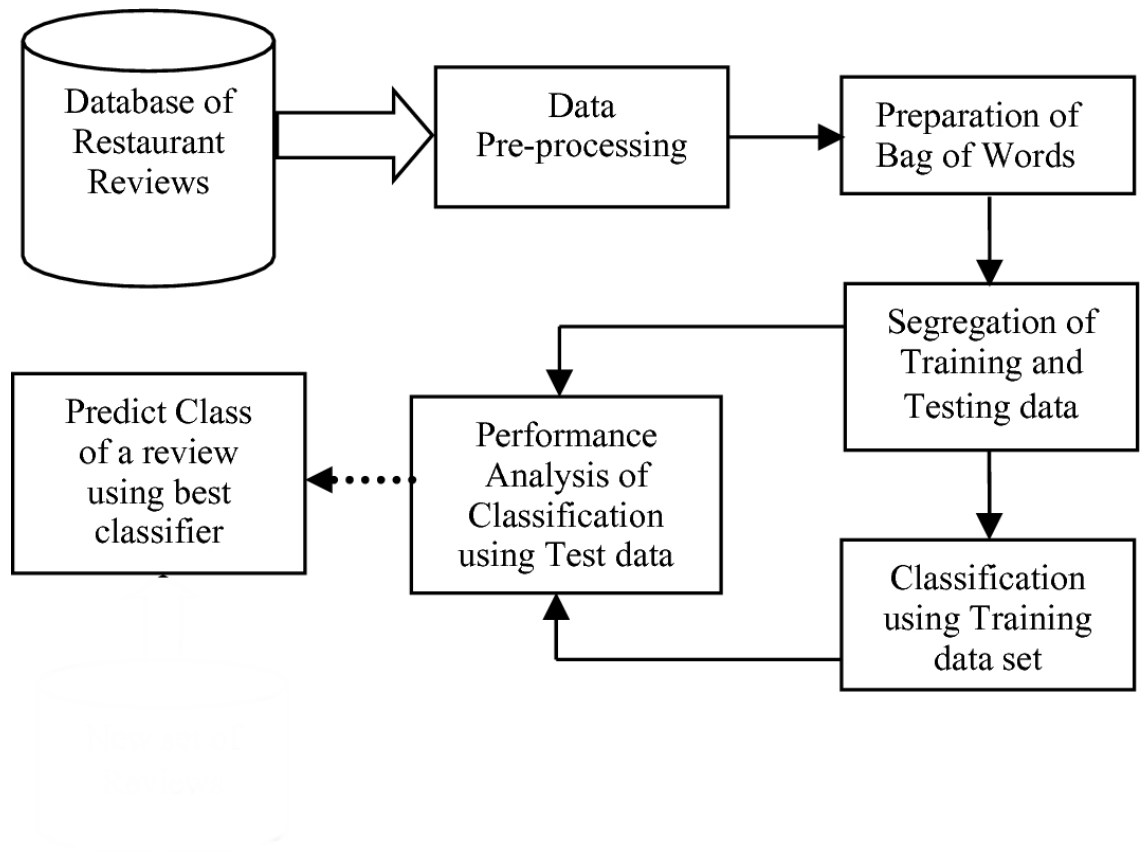


Fig 5.1 System Architecture

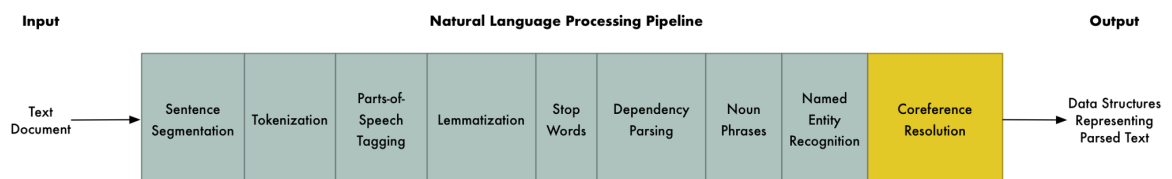


Fig 5.2 NLP Pipeline

5.1.1 NLP Pipeline

Let's look at a piece of text from Wikipedia

London is the capital and most populous city of England and the United Kingdom. Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia. It was founded by the Romans, who named it Londinium.

This paragraph contains several useful facts. It would be great if a computer could read this text and understand that London is a city, London is located in England, London was settled by Romans and so on. But to get there, we have to first teach our computer the most basic concepts of written language and then move up from there.

5.1.1.1 Sentence Segmentation

The first step in the pipeline is to break the text apart into separate sentences. That gives us this

“London is the capital and most populous city of England and the United Kingdom.”

“Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia.”

“It was founded by the Romans, who named it Londinium.”

We can assume that each sentence in English is a separate thought or idea. It will be a lot easier to write a program to understand a single sentence than to understand a whole paragraph.

Coding a Sentence Segmentation model can be as simple as splitting apart sentences whenever you see a punctuation mark. But modern NLP pipelines often use more complex techniques that work even when a document isn't formatted cleanly.

5.1.1.2 Word Tokenization

Now that we've split our document into sentences, we can process them one at a time. Let's start with the first sentence from our document:

"London is the capital and most populous city of England and the United Kingdom."

The next step in our pipeline is to break this sentence into separate words or *tokens*. This is called *tokenization*. This is the result:

"London", "is", "the", "capital", "and", "most", "populous", "city", "of", "England", "and", "the", "United", "Kingdom", "."

Tokenization is easy to do in English. We'll just split apart words whenever there's a space between them. And we'll also treat punctuation marks as separate tokens since punctuation also has meaning.

5.1.1.3 Predicting Parts of Speech for Each Token

Next, we'll look at each token and try to guess its part of speech — whether it is a noun, a verb, an adjective and so on. Knowing the role of each word in the sentence will help us start to figure out what the sentence is talking about.

We can do this by feeding each word (and some extra words around it for context) into a pre-trained part-of-speech classification model:

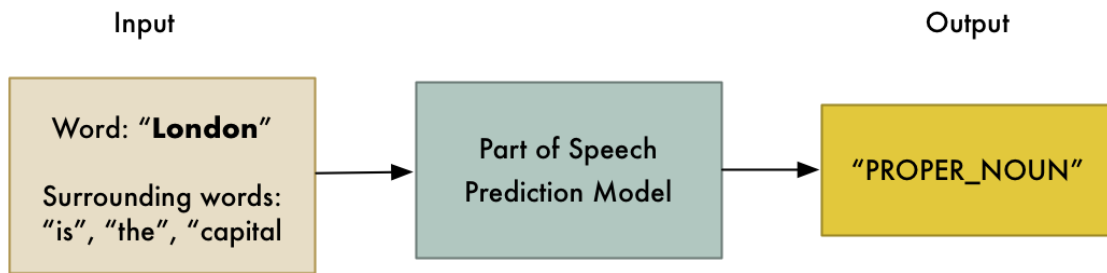


Fig 5.3 Parts of Speech Recognition

The part-of-speech model was originally trained by feeding it millions of English sentences with each word’s part of speech already tagged and having it learn to replicate that behavior.

Keep in mind that the model is completely based on statistics — it doesn’t actually understand what the words mean in the same way that humans do. It just knows how to guess a part of speech based on similar sentences and words it has seen before.

After processing the whole sentence, we’ll have a result like this:



Fig 5.4 After processing POS

With this information, we can already start to glean some very basic meaning. For example, we can see that the nouns in the sentence include “London” and “capital”, so the sentence is probably talking about London.

5.1.1.4 Text Lemmatization

In English (and most languages), words appear in different forms. Look at these two sentences:

I had a **pony**.

I had two **ponies**.

Both sentences talk about the noun **pony**, but they are using different inflections. When working with text in a computer, it is helpful to know the base form of each word so that you know that both sentences are talking about the same concept. Otherwise the strings “pony” and “ponies” look like two totally different words to a computer.

In NLP, we call finding this process *lemmatization* — figuring out the most basic form or *lemma* of each word in the sentence. The same thing applies to verbs. We can also lemmatize verbs by finding their root, unconjugated form. So “**I had two ponies**” becomes “**I [have] two [pony]**.”

Lemmatization is typically done by having a look-up table of the lemma forms of words based on their part of speech and possibly having some custom rules to handle words that you’ve never seen before.

Here’s what our sentence looks like after lemmatization adds in the root form of our verb:

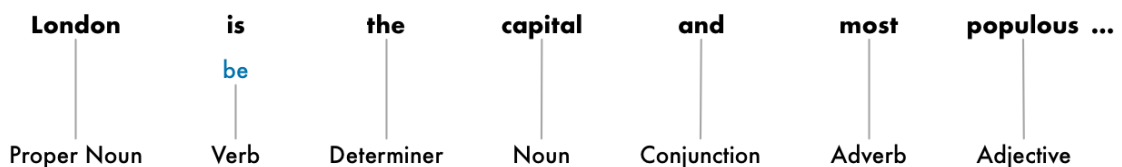


Fig 5.5 Lemmatization

5.1.1.5 Identifying Stop Words

Next, we want to consider the importance of each word in the sentence. English has a lot of filler words that appear very frequently like “and”, “the”, and “a”. When doing statistics on text, these words introduce a lot of noise since they appear way more frequently than other words. Some NLP pipelines will flag them as **stop words** —that is, words that you might want to filter out before doing any statistical analysis.

Here’s how our sentence looks with the stop words greyed out:

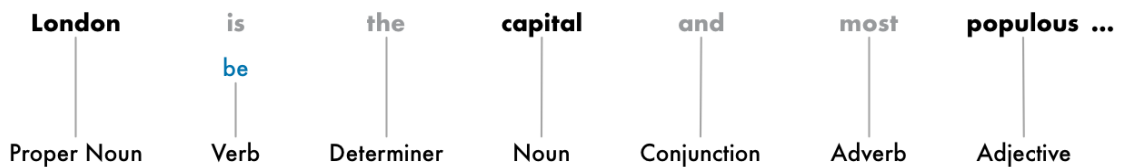


Fig 5.6 Stop Words Identification

Stop words are usually identified by just by checking a hardcoded list of known stop words. But there’s no standard list of stop words that is appropriate for all applications. The list of words to ignore can vary depending on your application. For example if you are building a rock band search engine, you want to make sure you don’t ignore the word “The”. Because not only does the word “The” appear in a lot of band names, there’s a famous 1980’s rock band called *The The*!

5.1.1.6 Dependency Parsing

The next step is to figure out how all the words in our sentence relate to each other. This is called *dependency parsing*.

The goal is to build a tree that assigns a single **parent** word to each word in the sentence. The root of the tree will be the main verb in the sentence. Here's what the beginning of the parse tree will look like for our sentence:

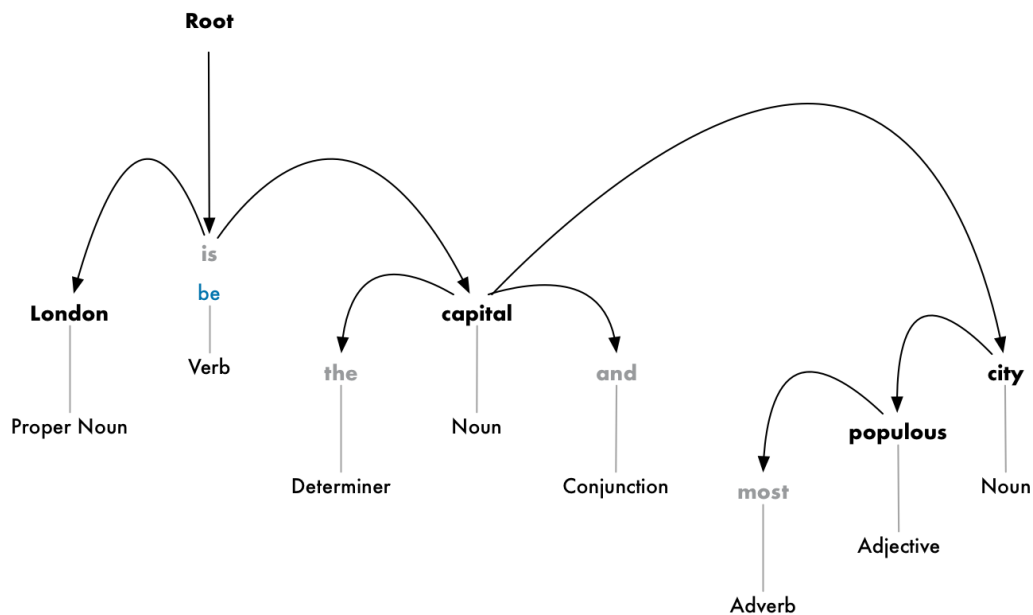


Fig 5.7 Dependency parsing part 1

But we can go one step further. In addition to identifying the parent word of each word, we can also predict the type of relationship that exists between those two words

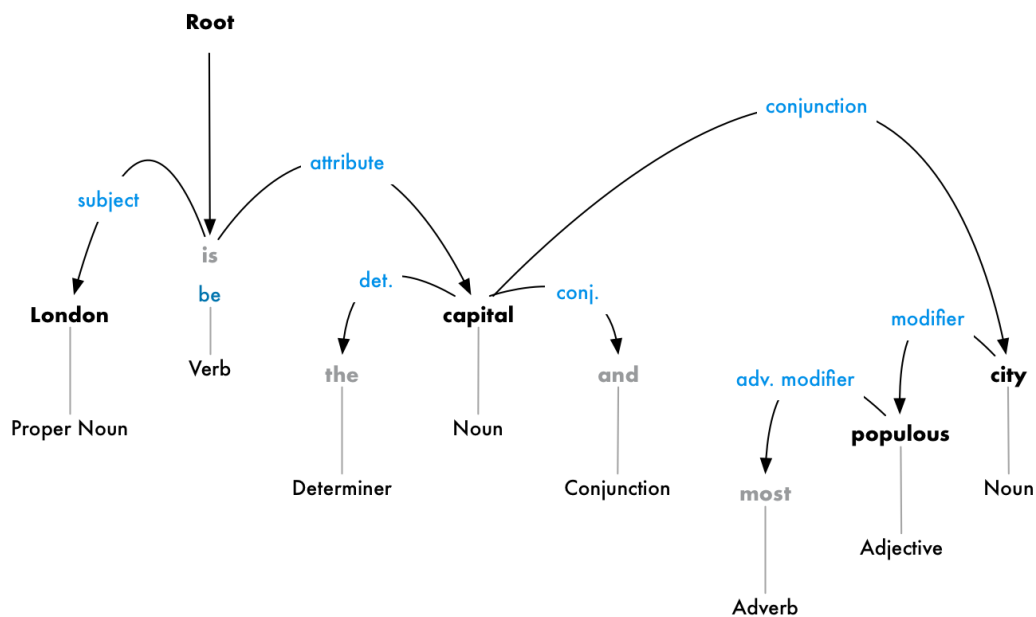


Fig 5.8 Dependency parsing part 2

This parse tree shows us that the subject of the sentence is the noun “*London*” and it has a “*be*” relationship with “*capital*”. We finally know something useful — *London* is a *capital*! And if we followed the complete parse tree for the sentence (beyond what is shown), we would even found out that London is the capital of the *United Kingdom*.

Just like how we predicted parts of speech earlier using a machine learning model, dependency parsing also works by feeding words into a machine learning model and outputting a result. But parsing word dependencies is particularly complex task and would require an entire article to explain in any detail. If you are curious how it works, a great place to start reading is Matthew Honnibal’s excellent article “*Parsing English in 500 Lines of Python*”.

But despite a note from the author in 2015 saying that this approach is now standard, it’s actually out of date and not even used by the author anymore. In 2016, Google released a new dependency parser called *Parsey McParseface* which outperformed previous benchmarks using a new deep learning approach which quickly spread throughout the industry. Then a year later, they released an even newer model called *ParseySaurus* which improved things further. In other words, parsing techniques are still an active area of research and constantly changing and improving.

It’s also important to remember that many English sentences are ambiguous and just really hard to parse. In those cases, the model will make a guess based on what parsed version of the sentence seems most likely but it’s not perfect and sometimes the model will be embarrassingly wrong. But over time our NLP models will continue to get better at parsing text in a sensible way.

5.1.1.6.1 Finding Noun Phrases

So far, we've treated every word in our sentence as a separate entity. But sometimes it makes more sense to group together the words that represent a single idea or thing. We can use the information from the dependency parse tree to automatically group together words that are all talking about the same thing.

For example, instead of this:

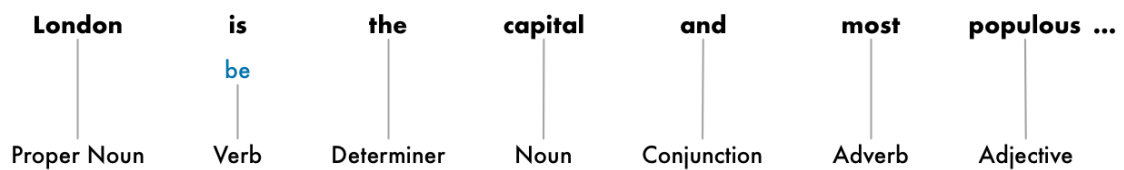


Fig 5.9 Noun Phrases part 1

We can group the noun phrases to generate this:

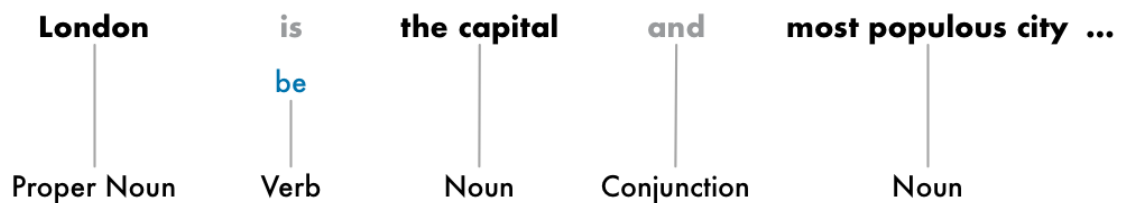


Fig 5.10 Noun Phrases part 2

Whether or not we do this step depends on our end goal. But it's often a quick and easy way to simplify the sentence if we don't need extra detail about which words are adjectives and instead care more about extracting complete ideas.

5.1.1.7 Named Entity Recognition (NER)

Now that we've done all that hard work, we can finally move beyond grade-school grammar and start actually extracting ideas.

In our sentence, we have the following nouns:

London is the **capital** and most populous **city** of **England** and the **United Kingdom**.

Fig 5.11 Nouns

Some of these nouns present real things in the world. For example, “*London*”, “*England*” and “*United Kingdom*” represent physical places on a map. It would be nice to be able to detect that! With that information, we could automatically extract a list of real-world places mentioned in a document using NLP.

The goal of *Named Entity Recognition*, or *NER*, is to detect and label these nouns with the real-world concepts that they represent. Here's what our sentence looks like after running each token through our NER tagging model:

London is the capital and most populous city of **England** and the **United Kingdom**.

Geographic Entity		Geographic Entity		Geographic Entity
----------------------	--	----------------------	--	----------------------

Fig 5.12 Entities

But NER systems aren't just doing a simple dictionary lookup. Instead, they are using the context of how a word appears in the sentence and a statistical model to guess which type of noun a word represents. A good NER system can tell the difference between “*Brooklyn Decker*” the person and the place “*Brooklyn*” using context clues.

5.1.1.8 Coreference Resolution

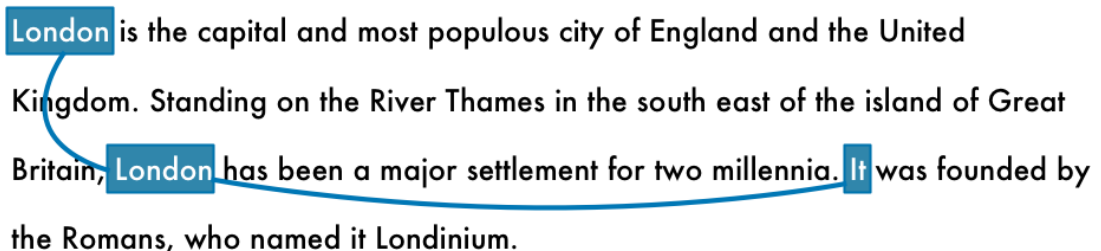
At this point, we already have a useful representation of our sentence. We know the parts of speech for each word, how the words relate to each other and which words are talking about named entities. However, we still have one big problem. English is full of pronouns — words like *he*, *she*, and *it*. These are shortcuts that we use instead of writing out names over and over in each sentence. Humans can keep track of what these words represent based on context. But our NLP model doesn't know what pronouns mean because it only examines one sentence at a time.

Let's look at the third sentence in our document:

“It was founded by the Romans, who named it Londinium.”

If we parse this with our NLP pipeline, we'll know that “it” was founded by Romans. But it's a lot more useful to know that “London” was founded by Romans. As a human reading this sentence, you can easily figure out that “it” means “London”. The goal of coreference resolution is to figure out this same mapping by tracking pronouns across sentences. We want to figure out all the words that are referring to the same entity.

Here's the result of running coreference resolution on our document for the word “London”:



London is the capital and most populous city of England and the United Kingdom. Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia. It was founded by the Romans, who named it Londinium.

Fig 5.13 Coreference

6. CODE TEMPLATES

6.1 Creating feature sets and classifying processed data using multiple algorithms

```
import nltk
import random
import re
import os

from nltk.classify.scikitlearn import SklearnClassifier
import pickle
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from nltk.classify import ClassifierI
from statistics import mode
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

from nltk.stem import WordNetLemmatizer

class VoteClassifier(ClassifierI):
    def __init__(self, *classifiers):
        self._classifiers = classifiers

    def classify(self, features):
        votes = []
        for c in self._classifiers:
```

```

        v = c.classify(features)
        votes.append(v)
    return mode(votes)

def confidence(self, features):
    votes = []
    for c in self._classifiers:
        v = c.classify(features)
        votes.append(v)

    choice_votes = votes.count(mode(votes))
    conf = choice_votes / len(votes)
    return conf

files_pos = os.listdir('train/pos')
files_pos = [open('train/pos/'+f, 'r').read() for f in files_pos]
files_neg = os.listdir('train/neg')
files_neg = [open('train/neg/'+f, 'r').read() for f in files_neg]

all_words = []
documents = []

# j is adjct, r is adverb, and v is verb
allowed_word_types = ["J", "V", "R"]

for p in files_pos:
    documents.append( (p, "pos") )
    cleaned = re.sub(r'^[a-zA-Z]\s', "", p)
    words = word_tokenize(cleaned)

    lm = WordNetLemmatizer()

```

```

words = [lm.lemmatize(w) for w in words if not w in stop_words]
pos = nltk.pos_tag(words)
for w in pos:
    if w[1][0] in allowed_word_types:
        all_words.append(w[0].lower())

```

```

for p in files_neg:
    documents.append( (p, "neg") )
    cleaned = re.sub(r'[^(a-zA-Z)\s]', "", p)
    words = word_tokenize(cleaned)

```

```

lm = WordNetLemmatizer()
words = [lm.lemmatize(w) for w in words if not w in stop_words]
pos = nltk.pos_tag(words)
for w in pos:
    if w[1][0] in allowed_word_types:
        all_words.append(w[0].lower())

```

```

save_documents = open("documents.pickle","wb")
pickle.dump(documents, save_documents)
save_documents.close()

```

```

all_words = nltk.FreqDist(all_words)

```

```

word_features = list(all_words.keys())[:5000]

```

```

save_word_features = open("word_features5k.pickle","wb")
pickle.dump(word_features, save_word_features)
save_word_features.close()

```

```

def find_features(document):
    words = word_tokenize(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)

    return features

featuresets = [(find_features(rev), category) for (rev, category) in documents]

random.shuffle(featuresets)
#print(len(featuresets))

featuresets_f = open("featuresets.pickle", "wb")
pickle.dump(featuresets, featuresets_f)
featuresets_f.close()

testing_set = featuresets[20000:]
training_set = featuresets[:20000]

classifier = nltk.NaiveBayesClassifier.train(training_set)
#print("Original Naive Bayes Algo accuracy percent:",
      (nltk.classify.accuracy(classifier, testing_set))*100)
#classifier.show_most_informative_features(15)

#####

save_classifier = open("originalnaivebayes5k.pickle","wb")
pickle.dump(classifier, save_classifier)
save_classifier.close()

```

```
MNB_classifier = SklearnClassifier(MultinomialNB())
MNB_classifier.train(training_set)
#print("MNB_classifier accuracy percent:", (nltk.classify.accuracy(MNB_classifier,
testing_set))*100)
```

```
save_classifier = open("MNB_classifier5k.pickle","wb")
pickle.dump(MNB_classifier, save_classifier)
save_classifier.close()
```

```
BernoulliNB_classifier = SklearnClassifier(BernoulliNB())
BernoulliNB_classifier.train(training_set)
#print("BernoulliNB_classifier accuracy percent:",
(nltk.classify.accuracy(BernoulliNB_classifier, testing_set))*100)
```

```
save_classifier = open("BernoulliNB_classifier5k.pickle","wb")
pickle.dump(BernoulliNB_classifier, save_classifier)
save_classifier.close()
```

```
LogisticRegression_classifier = SklearnClassifier(LogisticRegression(solver='lbfgs'))
LogisticRegression_classifier.train(training_set)
#print("LogisticRegression_classifier accuracy percent:",
(nltk.classify.accuracy(LogisticRegression_classifier, testing_set))*100)
```

```
save_classifier = open("LogisticRegression_classifier5k.pickle","wb")
pickle.dump(LogisticRegression_classifier, save_classifier)
save_classifier.close()
```

```
LinearSVC_classifier = SklearnClassifier(LinearSVC())
LinearSVC_classifier.train(training_set)
#print("LinearSVC_classifier accuracy percent:",
(nltk.classify.accuracy(LinearSVC_classifier, testing_set))*100)
```



```

save_classifier = open("LinearSVC_classifier5k.pickle","wb")
pickle.dump(LinearSVC_classifier, save_classifier)
save_classifier.close()

classifiers_dict = {'ONB': [classifier, 'originalnaivebayes5k.pickle'],
                    'MNB': [MNB_classifier, 'MNB_classifier5k.pickle'],
                    'BNB': [BernoulliNB_classifier, 'BernoulliNB_classifier5k.pickle'],
                    'LogReg': [LogisticRegression_classifier,
                    'LogisticRegression_classifier5k.pickle'],

                    'SVC': [LinearSVC_classifier, 'LinearSVC_classifier5k.pickle']}

from sklearn.metrics import f1_score, accuracy_score
ground_truth = [r[1] for r in testing_set]
predictions = {}
f1_scores = {}
for clf, listy in classifiers_dict.items():
    # getting predictions for the testing set by looping over each reviews featureset
    tuple
    # The first elemnt of the tuple is the feature set and the second element is the label
    predictions[clf] = [listy[0].classify(r[0]) for r in testing_set]
    f1_scores[clf] = f1_score(ground_truth, predictions[clf], labels = ['neg', 'pos'],
    average = 'micro')
    print(f'f1_score {clf}: {f1_scores[clf]}')

acc_scores = {}
for clf, listy in classifiers_dict.items():
    # getting predictions for the testing set by looping over each reviews featureset
    tuple
    # The first elemnt of the tuple is the feature set and the second element is the label
    acc_scores[clf] = accuracy_score(ground_truth, predictions[clf])
    print(f'Accuracy_score {clf}: {acc_scores[clf]}')

```

6.1.1 The Algorithms used are as follows

6.1.1.1 Naive bayes

Naive Bayes is a supervised learning algorithm used for classification tasks. Hence, it is also called Naive Bayes Classifier.

Naive bayes is a supervised learning algorithm for classification so the task is to find the class of observation (data point) given the values of features. Naive bayes classifier calculates the probability of a class a set of feature values (i.e. $p(y_i | x_1, x_2, \dots, x_n)$). Input this into Bayes' theorem:

$$p(y_i | x_1, x_2, \dots, x_n) = \frac{p(x_1, x_2, \dots, x_n | y_i) \cdot p(y_i)}{p(x_1, x_2, \dots, x_n)}$$

$p(x_1, x_2, \dots, x_n | y_i)$ means the probability of a specific combination of features given a class label. To be able to calculate this, we need extremely large datasets to have an estimate on the probability distribution for all different combinations of feature values. To overcome this issue, **naive bayes algorithm assumes that all features are independent of each other**. Furthermore, denominator ($p(x_1, x_2, \dots, x_n)$) can be removed to simplify the equation because it only normalizes the value of conditional probability of a class given an observation ($p(y_i | x_1, x_2, \dots, x_n)$).

The probability of a class ($p(y_i)$) is very simple to calculate:

$$p(y_i) = \frac{\text{number of observations with class } y_i}{\text{number of all observations}}$$

Under the assumption of features being independent, $p(\mathbf{x1}, \mathbf{x2}, \dots, \mathbf{xn} | \mathbf{yi})$ can be written as:

$$p(\mathbf{x1}, \mathbf{x2}, \dots, \mathbf{xn} | \mathbf{yi}) = p(\mathbf{x1} | \mathbf{yi}) \cdot p(\mathbf{x2} | \mathbf{yi}) \cdot \dots \cdot p(\mathbf{xn} | \mathbf{yi})$$

6.1.1.2 Multinomial naive bayes

You probably have more than two words in the vocabulary. Multinomial distribution is just a generalization of the binomial distribution derived using a couple combinatorics concepts: Dixon's identity, and the number of ways to form a combination.

$$m!x_1! \cdots x_n! q_1^{x_1} \cdots q_n^{x_n} n! m! x_1! \cdots x_n! q_1^{x_1} \cdots$$

$q_1^{x_1} \cdots q_n^{x_n}$ where $m = \sum_{i=1}^n x_i$ and $n = \sum_{i=1}^n q_i$ (e.g. m is number of words in observed document)

$$p(\mathbf{x} | \mathbf{Ck}) = \frac{(\sum_{i=1}^n x_i)!}{\prod_{i=1}^n x_i! \prod_{k=1}^n q_k^{x_k}} p(\mathbf{x} | \mathbf{Ck}) = \frac{(\sum_{i=1}^n x_i)!}{\prod_{i=1}^n x_i! \prod_{k=1}^n q_k^{x_k}} \implies p(\mathbf{Ck} | \mathbf{x}) = p(\mathbf{Ck}) p(\mathbf{x})$$

$$\frac{(\sum_{i=1}^n x_i)!}{\prod_{i=1}^n x_i! \prod_{k=1}^n q_k^{x_k}} \implies p(\mathbf{Ck} | \mathbf{x}) = p(\mathbf{Ck}) p(\mathbf{x}) \frac{(\sum_{i=1}^n x_i)!}{\prod_{i=1}^n x_i! \prod_{k=1}^n q_k^{x_k}} \text{ (from substituting into the general Naive Bayes equation above)}$$

For practicality, you may only care about relative probabilities (take away the need to compute $p(\mathbf{x})$ and replace "=" with " \propto ")

6.1.1.3 Bernoulli naive bayes

For a Bernoulli trial with two outcomes and probability q for one of those outcomes, the probability of this outcome occurring exactly k out of m times is

$$p(k, m) = \binom{m}{k} q^k (1-q)^{m-k} \quad p(k, m) = \binom{m}{k} q^k (1-q)^{m-k}$$

The analogy with the document/word example above, is having two possible words, one of them occurring with proportion q in the corpus, and computing the probability that it occurs k times in a document with m words.

6.1.1.4 Logistic regression

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).

In logistic regression, the dependent variable is binary or dichotomous, i.e. it only contains data coded as 1 (TRUE, success, pregnant, etc.) or 0 (FALSE, failure, non-pregnant, etc.).

The goal of logistic regression is to find the best fitting (yet biologically reasonable) model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a log-it transformation of the probability of presence of the characteristic of interest:

$$\text{logit}(p) = b_0 + b_1 X_1 + b_2 X_2 + b_3 X_3 + \dots + b_k X_k$$

where p is the probability of presence of the characteristic of interest. The logit transformation is defined as the logged odds:

$$\text{odds} = \frac{p}{1-p} = \frac{\text{probability of presence of characteristic}}{\text{probability of absence of characteristic}}$$

and

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right)$$

Rather than choosing parameters that minimize the sum of squared errors (like in ordinary regression), estimation in logistic regression chooses parameters that maximize the likelihood of observing the sample value.

6.1.1.5 LinearSVC

SVC uses the Support Vector Domain Description (SVDD) to delineate the region in data space where the input examples are concentrated. SVDD belongs to the general category of kernel based learning. In its "linear" version SVDD looks for the smallest sphere that encloses the data. When used in conjunction with a kernel function, it looks for the smallest enclosing sphere in the feature space defined by the kernel function. While in feature space the data is described by a sphere, when mapped back to data-space the sphere is transformed into a set of non-linear contours that enclose the data (see Figure 2). SVDD provides a decision function that tells whether a given input is inside the feature-space sphere or not, indicating whether a given point belongs to the support of the distribution. More specifically, it is the radius-squared of the feature-space sphere minus the distance-squared of the image of a data point \mathbf{x} from the center of the feature-space sphere. This function, denoted by $f(\mathbf{x})$ returns a value greater than 0 if \mathbf{x} is inside the feature space sphere and negative otherwise. For more details on SVDD the reader is referred to the SVDD article.

The contours where $f(\mathbf{x})=0$ are then interpreted as cluster boundaries. An example of such contours is shown in Figure 2. However, these boundaries define the clusters implicitly, and an additional step is required to "tease" the cluster membership out of the SVDD.

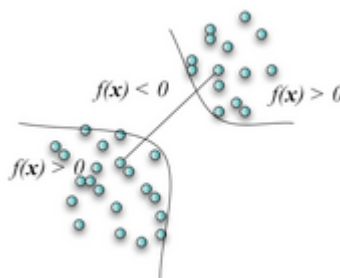


Figure 6.1 Low density SVDD Region

The key geometrical observation that enables to infer clusters out of the SVDD is that given a pair of data points that belong to different components (clusters), the line segment that connects them must go through a region in data space which is part of a "valley" in the probability density of the data, i.e. does not belong to

the support of the distribution. Such a line must then go outside the feature-space sphere, and therefore have a segment with points that return a negative value when tested with the SVDD decision function (see Figure 1). This observation leads to the definition of an adjacency matrix A between pairs of points in our dataset. For a given pair of points \mathbf{x}_i and \mathbf{x}_j the i,j element of A is given by

$$A_{ij} = \begin{cases} 1, & \text{if } f(\mathbf{x}) > 0 \text{ for all } \mathbf{x} \text{ on the line segment connecting } \mathbf{x}_i \text{ and } \mathbf{x}_j \\ 0, & \text{otherwise.} \end{cases}$$

Clusters are now defined as the connected components of the graph induced by A . Checking the line segment is implemented by sampling a number of points (20 points were used in numerical experiments).

6.2 Creating a Voter Classifier and calculating confidence to return sentiment analysis

```
import nltk
import random
#from nltk.corpus import movie_reviews
from nltk.classify.scikitlearn import SklearnClassifier
import pickle
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC
from nltk.classify import ClassifierI
from statistics import mode
from nltk.tokenize import word_tokenize
```

```
class VoteClassifier(ClassifierI):
    def __init__(self, *classifiers):
        self._classifiers = classifiers
```

```
def classify(self, features):
    votes = []
    for c in self._classifiers:
        v = c.classify(features)
        votes.append(v)
    return mode(votes)
```

```
def confidence(self, features):
    votes = []
    for c in self._classifiers:
        v = c.classify(features)
        votes.append(v)
```

```
choice_votes = votes.count(mode(votes))
conf = choice_votes / len(votes)
return conf
```

```
documents_f = open("documents.pickle", "rb")
documents = pickle.load(documents_f)
documents_f.close()
```

```
word_features5k_f = open("word_features5k.pickle", "rb")
word_features = pickle.load(word_features5k_f)
word_features5k_f.close()
```

```
def find_features(document):
    words = word_tokenize(document)
    features = {}
```

```

    for w in word_features:
        features[w] = (w in words)

    return features

featuresets_f = open("featuresets.pickle", "rb")
featuresets = pickle.load(featuresets_f)
featuresets_f.close()

random.shuffle(featuresets)
#print(len(featuresets))

testing_set = featuresets[10000:]
training_set = featuresets[:10000]

open_file = open("originalnaivebayes5k.pickle", "rb")
classifier = pickle.load(open_file)
open_file.close()

open_file = open("MNB_classifier5k.pickle", "rb")
MNB_classifier = pickle.load(open_file)
open_file.close()

open_file = open("BernoulliNB_classifier5k.pickle", "rb")
BernoulliNB_classifier = pickle.load(open_file)
open_file.close()

open_file = open("LogisticRegression_classifier5k.pickle", "rb")
LogisticRegression_classifier = pickle.load(open_file)
open_file.close()

```



```
open_file = open("LinearSVC_classifier5k.pickle", "rb")
LinearSVC_classifier = pickle.load(open_file)
open_file.close()
```

```
open_file = open("SGDC_classifier5k.pickle", "rb")
SGDC_classifier = pickle.load(open_file)
open_file.close()
```

```
voted_classifier = VoteClassifier(
    classifier,
    LinearSVC_classifier,
    MNB_classifier,
    BernoulliNB_classifier,
    LogisticRegression_classifier)
```

```
def sentiment(text):
    feats = find_features(text)
    print(text)
    return voted_classifier.classify(feats), voted_classifier.confidence(feats)
```

6.3 Testing our trained sentiment analysis module with multiple examples

```
import sentiment_mod as s
```

```
print(s.sentiment("This movie was awesome! The acting was great, plot was  
wonderful, and there were pythons...so yea!"))  
print("pos")  
print()
```

```
print(s.sentiment("This movie was utter junk. There were absolutely 0 pythons. I don't  
see what the point was at all. Horrible movie, 0/10"))  
print("neg")  
print()
```

```
print(s.sentiment("Barely any good"))  
print("neg")  
print()
```

```
print(s.sentiment("It was a great movie"))  
print("neg")  
print()
```

```
print(s.sentiment("emerges as something rare , an issue movie that's so honest and  
keenly observed that it doesn't feel like one"))  
print("pos")  
print()
```

```
print(s.sentiment("it's so laddish and juvenile , only teenage boys could possibly find  
it funny"))  
print("neg")  
print()
```

```
print(s.sentiment("My experience has been okay I guess"))  
print("neg")  
print()
```

```
print(s.sentiment("The movie was fantastic"))  
print("pos")  
print()
```

```
print(s.sentiment("Your support team is useless"))  
print("neg")  
print()
```

```
print(s.sentiment("It was an entertaining movie "))
```

7. TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

Software testing is the process of executing a program with the intention of finding errors in the code. It is a process of evolution of system or its parts by manual or automatic means to verify that it is satisfying specified or requirements or not. Generally, no system is perfect due to communication problems between user and developer, time constraints, or conceptual mistakes by developer.

Testing is not a distinct phase in system development life cycle but should be applicable throughout all phases i.e. design development and maintenance phase. Testing is used to show incorrectness and considered to success when an error is detected.

7.1 Testing Methodologies

7.1.1 *Software testing*

Software testing is a critical element of software quality and assurance and represents the ultimate review of specifications, design and coding. Testing is an exposure of the system to trial input to see whether it produces correct output. Software testing includes the following

Test activities are determined and test data selected.

The test is conducted and test results are compared with the expected results.

7.1.2 Unit Testing

Unit testing focuses on the building blocks of the software system, that is, objects and subsystems. There are three motivations behind focusing on components. First, unit testing reduces the complexity of the overall test activities, allowing us to focus on smaller units of the system. Unit testing makes it easier to pinpoint and correct faults given that few computers are involved in this test. Unit testing allows parallelism in the testing activities; that is each component can be tested independently of one another.

The specific candidates for unit testing are chosen from the object model and the system decomposition of the system. In principle, all the objects developed during the development process should be tested. Which is often not feasible because of time and budget? Many unit testing techniques have been devised. The unit testing used in this project is Unit Testing.

7.1.3 Equivalence Testing

It is a black box testing technique that minimizes the number of test cases. The possible inputs are partitioned into equivalence classes, and a test case is selected for each class. The assumption of equivalence testing is that the system usually behaves in similar ways for all members of a class. To test the behavior associated with an equivalence class, we only need to test one member of the class. Equivalence testing consists of two steps: identification of the equivalence classes and selection of the test inputs.

7.1.4 Boundary Testing

Boundary testing is a special case of equivalence testing and focuses on the conditions at the boundary of the equivalence classes. Rather than selecting any element in the equivalence class, boundary testing requires that the element be selected from the “edges” of the equivalence class.

7.1.5 Path testing

Path testing is a white box testing technique that identifies faults in the implementation of the component. The assumption behind path is that, by exercising all possible paths through the code at least once, most faults will trigger failures. The identification of paths requires knowledge of the source code and data structures.

7.1.6 State testing

Object oriented languages introduce the opportunity for new types of faults in object oriented systems. State based testing is a recent testing technique, which focuses on object-oriented systems. Most testing technique, which focuses on selecting a number of test inputs for a given state of the system, exercising a component or a system, and comparing the observed outputs with java. State based testing focuses on comparing the resulting state of the system with the expected state. In the context of a class, state-based testing consists of deriving test cases from UML state chart diagram for the class.

7.1.7 Integration testing

It detects faults that have not been detected during unit testing, by focusing on small group of components.

7.1.8 Test Case Design

The design of tests for software and other engineering products can be as challenging as the initial design of the product. Test case methods provide the developer with a systematic approach to testing. Moreover, these methods provide a mechanism that can help to ensure the completeness of tests and provide the highest like hood for uncovering errors in software.

Any Engineered product can be tested in either of the two ways:

Knowing the specified function that a product has been designed to perform, tests can be conducted. These tests demonstrate whether each function is fully operational and at the same time searches for errors in each function.

Knowing the internal workings of a product, tests can be conducted to ensure that internal operations are performed according to specifications and all internal components hence been adequately exercised.

Test case design methods are divided into two types

7.1.8.1 White-box testing

White –box testing, sometimes called glass-box testing is a test case design method that uses the control structure of the procedural design to derive test cases. Using white-box testing methods, the s/w engineer can derive test cases that guarantee that all independent paths within a module have been exercised at least once. Exercise all logical decisions on their true and false sides. Execute all loops at their boundaries and within their operational bounds. Exercise internal data structures to ensure their validity. Basis path testing is a white-box testing technique. The basis path method enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set are guaranteed to exercise every statement in the program at least one time during testing.

7.1.8.2 Black-box testing

Black-box testing, also called behavioural testing, focuses on the functional requirements of the s/w. Black-box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements of a program. It is a complementary approach that is likely to uncover a different class of errors that white-box methods could not. Black-box testing attempted to find errors in the following categories:

Incorrect or missing functions.

Interface errors

Errors in data structures or external data access.

Behaviour or performance errors.

Initialization and termination errors.

Black-box testing purposely disregards control structure; attention is focused on information domain. By applying black-box techniques, we derive a set of cases that satisfies the criteria test cases that reduce, by a count that is greater than one, the number of additional test cases that must be designed to achieve reasonable testing. Test cases that tell us something about the presence or absence of classes of errors, rather than an error associated only with the specified test.

7.2 Test Cases

For the given project, we are going to test the calculated accuracy score over multiple examples, to check whether there is consistency in the calculated scores.

7.2.1 *Testing accuracy with 10 new reviews*

For the first scenario, we use ten different movie reviews and check whether the predicted sentiment is right or wrong.

This movie was awesome! The acting was great, plot was wonderful, and there were pythons...so yea!

('pos', 1.0)

pos

This movie was utter junk. There were absolutely 0 pythons. I don't see what the point was at all. Horrible movie, 0/10

('neg', 1.0)

neg

Barely any good

('pos', 0.8)

neg

It was a great movie

('pos', 1.0)

pos

emerges as something rare , an issue movie that's so honest and keenly observed that it doesn't feel like one

('pos', 1.0)

pos

it's so laddish and juvenile , only teenage boys could possibly find it funny

('neg', 0.6)

neg

My experience has been okay I guess

('neg', 0.6)

neg

The movie was fantastic

('pos', 1.0)

pos

Movie is not good

('pos', 0.8)

neg

It was an entertaining movie

('pos', 1.0)

pos

We see that 8/10 reviews are analyzed correctly. Therefore the 83% accuracy rate is maintained across ten reviews which are not in the dataset used to train the algorithm.

7.2.2 Testing accuracy with 15 more reviews

We will try the same over 15 reviews to find out if there is consistency in the accuracy rate. If we get 12 or more correct, then our sentiment analysis using NLP has been successful.

Whether or not this is the best film Marvel Studios has made to date-and it is clearly in the discussion-it is by far the most thought-provoking.

('pos', 1.0)

pos

The identity politics provide a fresh spin to the genre's increasingly tedious narrative formula.

('pos', 0.6)

pos

It is epic in so many ways and important for all of us to experience, start discussions about, and champion.

('pos', 1.0)

pos

It serves as an incredibly important cultural message arriving at a crucial time, complete with stunning visuals, unforgettable and epic action-sequences.

('pos', 1.0)

pos

The only complaint about Avengers: Endgame is that it raises the bar so high that there may well never be a superhero movie to match it.

('pos', 1.0)

pos

The MCU will go on and on, but this chapter - and the American pragmatism vs. American ideals bromance that drove it - have well and truly come to their Excelsior! Nuff said! moment.

('pos', 1.0)

pos

I'm not naïve enough to suppose that anything I can say here is likely to change anyone's plans about going or not going, so I'll simply say that I enjoyed this silly but thrilling superhero free-for-all.

('neg', 1.0)

neg

Pixar by the numbers, this third and largely unnecessary sequel to Toy Story delivers everything you'd expect from the animation studio, minus the warmth, wit, and dread.

('pos', 1.0)

neg

Toy Story 4 is so good it's criminal. The legislation it flouts is the law of diminishing returns which governs movies with numbers after their names.

('pos', 1.0)

pos

Rare is a character who has both a one-note comedy beat (executed beautifully) and an extreme amount of emotional depth. What makes Forky so compelling is not his existential crisis, but his deep empathy for others...

('pos', 1.0)

pos

The Last Airbender squanders its popular source material with incomprehensible plotting, horrible acting, and detached joyless direction.

('neg', 1.0)

neg

Bizarre and clumsily plotted, Gigli is a mess. As for its stars, Affleck and Lopez lack chemistry.

('neg', 1.0)

neg

Preposterous and predictable, The Perfect Man manages few laughs with its poorly paced sitcom script, cookie-cutter characters and contrived plotting.

('neg', 1.0)

neg

Clichéd and unoriginal, Paranoia is a middling techno-thriller with indifferent performances and a shortage of thrills.

('neg', 0.6)

neg

A poorly constructed, derivative sci-fi stinker with a weak script and poor action sequences.

('neg', 1.0)

neg

We get 14 out of 15 correct, and therefore come to the conclusion that our training has been successful.

8. OUTPUT SCREENS

```
Accuracy_score ONB: 0.8392
Accuracy_score MNB: 0.8386
Accuracy_score BNB: 0.839
Accuracy_score LogReg: 0.8424
Accuracy_score SVC: 0.8226
```

Fig 8.1 Accuracy scores of classifiers

This movie was awesome! The acting was great, plot was wonderful, and there were pythons...so yea!

('pos', 1.0)
pos

This movie was utter junk. There were absolutely 0 pythons. I don't see what the point was at all. Horrible movie, 0/10

('neg', 1.0)
neg

Barely any good

('pos', 0.8)
neg

It was a great movie

('pos', 1.0)
pos

emerges as something rare , an issue movie that's so honest and keenly observed that it doesn't feel like one

('pos', 1.0)
pos

Fig 8.2 Sentiment Analysis part 1

It is epic in so many ways and important for all of us to experience,
start discussions about, and champion.
(`'pos', 1.0`)
pos

It serves as an incredibly important cultural message arriving at a
crucial time, complete with stunning visuals, unforgettable and epic
action-sequences.
(`'pos', 1.0`)
pos

The only complaint about Avengers: Endgame is that it raises the bar so
high that there may well never be a superhero movie to match it.
(`'pos', 1.0`)
pos

The MCU will go on and on, but this chapter – and the American pragmatism
vs. American ideals bromance that drove it – have well and truly come to
their Excelsior! Nuff said! moment.
(`'pos', 1.0`)
pos

Fig 8.3 Sentiment Analysis part 2

Bizarre and clumsily plotted, Gigli is a mess. As for its stars, Affleck
and Lopez lack chemistry.
(`'neg', 1.0`)
neg

Preposterous and predictable, The Perfect Man manages few laughs with its
poorly paced sitcom script, cookie-cutter characters and contrived
plotting.
(`'neg', 1.0`)
neg

Clichéd and unoriginal, Paranoia is a middling techno-thriller with
indifferent performances and a shortage of thrills.
(`'neg', 0.6`)
neg

A poorly constructed, derivative sci-fi stinker with a weak script and
poor action sequences.
(`'neg', 1.0`)
neg

Fig 8.4 Sentiment Analysis part 3

9. CONCLUSION

Here's what sentiment analysis is: it's a tremendously difficult task even for human beings. That being said, sentiment analysis classifiers might not be as precise as other types of classifiers.

One might be asking, is it worth the effort? The answer is simple: it sure is worth it! Chances are that sentiment analysis predictions will be wrong from time to time, but by using sentiment analysis you will get the opportunity to get it right about 70-80% of the times you submit your texts for classification.

Although our model gives us an accuracy score of about 84%, the model is bound to fail when it comes across sentences with sarcasm or multiple negations involved. Sometimes, there also might be an issue with some phrases being removed from the sentences (because of them being a part of the set of stop words). However, we consider this to be a brilliant project, given the fact that we were able to achieve over 80% accuracy in most cases, which is an incredible feat in itself.

This project has also served as an excellent teacher of various kinds of Machine learning algorithms, and has helped us understand how computers are becoming smarter by the day, and are soon to achieve the intelligence that a human being possesses.

10. BIBLIOGRAPHY

Below are the various sources and literature/research papers that we had reviewed and studied to come up with this project.

Arpan Somani, Lakshya Kumar, Pushpak Bhattacharya(IIT Bombay). Having 2 hours to write a paper is fun: Detecting sarcasm in Numerical Portions of Text
<https://arxiv.org/pdf/1709.01950.pdf>

Aniruddha Ghosh, Tony Veale (Uni. of Dublin). Fracking Sarcasm using Neural Networks
<https://www.aclweb.org/anthology/W16-0425.pdf>

Wareesa Sharif, Rashid Naseem (IEEE). Effect of negation in sentiment analysis
<https://ieeexplore.ieee.org/document/7845119?section=abstract>

Bo Wang, Min Liu (Stanford University). Deep Learning for aspect-based analysis
<https://cs224d.stanford.edu/reports/WangBo.pdf>

<https://monkeylearn.com/sentiment-analysis/>

<https://towardsdatascience.com/basic-binary-sentiment-analysis-using-nltk-c94ba17ae386>

<http://breakthroughanalysis.com/2012/09/10/typesofsentimentanalysis/>