Major Project

An Analysis of Customer's Emotional-AI using Natural Language Processing

By

PRAPHUL SINHA

2K16/EMBA/522

Under Guidance of

DR. RAJAN YADAV

Professor, Delhi School of Management

Submitted in partial fulfillment of the requirement for award of

Master of Business Administration (EXECUTIVE)

Delhi School of Management

Delhi Technological University

Delhi-110042

Disclaimer

The views expressed in this project are personal and not of the organization and this project is done as a detailed study under the course from strategy perspective only.

DECLARATION

This is to certify that the project entitled ‘*An Analysis of Customer's Emotional-AI using Natural Language Processing*’ has been successfully completed by **Praphul Sinha** – **2K16/EMBA/522**.

This is further certified that this project work is a record of bonafide work done by me.

Place: **New Delhi** **Praphul Sinha**
Date: **22-May-2018** 2K16/EMBA/522

Certificate

This is to certify that the project entitled

‘***An Analysis of Customer's Emotional-AI using Natural Language Processing***’

has been successfully completed by **Praphul Sinha** – **2K16/EMBA/522**.

This is further certified that this project work is a record of

bonafide work done by him under my guidance.

The matter embodied in this report has not been submitted for award of any degree.

DR. RAJAN YADAV
Professor
Delhi School of Management
Delhi Technological University

DR. RAJAN YADAV
Professor
Delhi School of Management
Delhi Technological University

Acknowledgement

I Praphul Sinha, wish to extend my gratitude to **Dr. Rajan Yadav**, Professor, Delhi School of Management (DSM), Delhi Technological University; for giving me all the guidance and valuable insights to take up this Semester Project.

I also take this opportunity to convey sincere thanks to all the faculty members for directing and advising during the course.

ABSTRACT

It is imperative that today’s companies create compelling, differentiating customer experiences through marketing. Predictive marketing (**Artificial Intelligence**) changes the game for marketers: it gives firms the ability to anticipate consumers’ needs and interests, and more importantly, their likely reactions to marketing messages.

AI can be more focused on delivering ads to people, with less guesswork. AI predicts that some 55% of all digital advertising dollars will be driven by programmatic initiatives in 2015, as machine learning takes precedence over human analysis. The figure is predicted to rise to 63% by 2016, which represents $20 billion in programmatic ad buys. With AI, marketers will be able to understand consumers on an intricate level.1

The entire experience will be tailored to the recipient to ensure every ad is relevant, a far cry from the days of blanket coverage and disruptive marketing techniques. Just like billboards on roadsides, pre-AI ads relied on a disruptive advertising strategy, designed to distract and conquer. Unfortunately, these efforts translated to little more than guesses about what consumers may want to buy based on the few details the company could glean from their IP address. This approach to advertising, however, failed to understand the context of individuated customer searches, an issue AI overcomes through ‘intelligent learning’.

The proliferation of digital devices raises an opportunity and a threat for marketers: with more data than ever before, marketers can better understand their customers and attempt to serve them more relevant, tailored messages and suggestions. The flipside for many marketing organizations today, is that they are now faced with more than they can
handle. Too much data emerging from too many data sources present difficult challenges to customer intelligence and marketing teams’ ability to generate and act upon meaningful insights. Artificial intelligence becomes necessary to accelerate the loop from

insights to execution. Without it, marketers will struggle to achieve their objectives of delivering truly personalized 1:1 marketing interactions in real-time. Artificial intelligence will drive innovation in marketing, by helping marketers move from post campaign optimization to a more predictive execution and optimization of campaigns.

In February 2017, Rocket Fuel commissioned Forrester Consulting to explore predictive marketing and AI themes to better inform marketers and agencies about the benefits and challenges of its application.
The survey covered France, Germany, Italy, UK, US and Australia.

The survey found the following key findings:

KEY FINDINGS

1. **Five fundamentals of success will drive the future of predictive marketing**

In this brave new world, marketers must take key steps to build marketing capabilities that anticipate and address individual customers' needs and wants. Forrester recognizes this and ultimately found that in order for marketers to succeed in the digital age and overcome barriers, they must follow five core fundamental principles for success.
The five fundamentals are:

1. Customer recognition
2. Nonlinear customer journey
3. Real-time decision making
4. AI-based intelligent decision making
5. Using first-party data over external data

2. **AI-driven marketing promises to simplify, optimize, and streamline processes**

Not only does AI-driven marketing enable automation of processes, it makes businesses smarter — finding new insights that were previously not humanly possible — and enables them to act on these insights in real-time.

3. **Find the right partners with the skills and expertise to help transform marketing**

Navigating the complexity of predictive marketing and AI solutions can be daunting for any organization in any industry. Few organizations will manage to navigate the next era of predictive marketing without any assistance; it’s therefore paramount to work with partners that provide not only expertise but excellent customer services, too.

TABLE OF CONTENT

1

INTRODUCTION

Romance should never begin with sentiment.
It should begin with science and end with a settlement.
— ***Oscar Wilde****, An Ideal Husband*

The "microblogging" social networking website "Twitter" has a huge and quickly growing user base. Twitter are those wherein we can write short can write short 140 characters long or less named "tweets". Tweet is the process in which we follow the people and their message as 'tweets'. Owing to the increasing popularity of the website, "tweet" can be collected in the form of large data set from website. Twitter has special features that permits persons to carry their ideas and opinions openly about so forth subject, conversation topic or product that they are attracted in spreading their sentiments about twitter.

As a result, Twitter is a decent medium to examine on behalf of supposedly motivating tendencies about noticeable themes in the news or popular tradition. The application is built for the purpose of analyzing data, although RStudio performs well with the statistical problems and proceeds with a user friendly interface.

Twitter text analysis generate a means of following feelings and approaches on the web and determines if they are positively or negatively received by the followers. This project will develop and determine whether the tweet status updates (which is not more than 140 characters) returns a positive or negative opinion on the behalf of the one who tweeted which is done by the classification model.

Allowing the application of various data mining algorithms to explain the textual dataset, the determination of Text mining is method for unstructured (textual) information and to extract meaningful numeric indices from the text.

Twitter text analysis refers to the use of normal language processing, computational semantics and text analysis to recognize and specific information from the dataset.

Twitter has grown its value in recent years has been rising in political, industrial taking place to analyze the public's groups and interested Internet users have common emotion for their products and services from the twitter posts.

1.1 The Demand for Information on Opinions and Sentiment

“What other people think” has always been an important piece of information for most of us during the decision-making process. Long before awareness of the World Wide Web became widespread, many of us asked our friends to recommend an auto mechanic or to explain who they were planning to vote for in local elections, requested reference letters regarding job applicants from colleagues, or consulted Consumer Reports to decide what dishwasher to buy. But the Internet and the Web have now (among other things) made it possible to find out about the opinions and experiences of those in the vast pool of people that are neither our personal acquaintances nor well-known professional critics —that is, people we have never heard of. And conversely, more and more people are making their opinions available to strangers via the Internet.

Indeed, according to two surveys of more than 2000 American adults:

- 81% of Internet users (or 60% of Americans) have done online research on a product at least once;

- 20% (15% of all Americans) do so on a typical day;

- among readers of online reviews of restaurants, hotels, and various services (e.g., travel agencies or doctors), between 73% and 87% report that reviews had a significant influence on their purchase;

- consumers report being willing to pay from 20% to 99% more for a 5-star-rated item than a 4-star-rated item (the variance stems from what type of item or service is considered);

- 32% have provided a rating on a product, service, or person via an online ratings system, and 30% (including 18% of online senior citizens) have posted an online comment or review regarding a product or service.

We hasten to point out that consumption of goods and services is not the only motivation behind people’s seeking out or expressing opinions online. A need for political information is another important factor.

For example, in a survey of over 2500 American adults, Rainie and Horrigan [248] studied the 31% of Americans — over 60 million people — that were 2006 campaign internet users, defined as those who gathered information about the 2006 elections online and exchanged views via email.

Of these,

- 28% said that a major reason for these online activities was to get perspectives from within their community, and 34% said that a major reason was to get perspectives from outside their community;

- 27% had looked online for the endorsements or ratings of external organizations;

- 28% said that most of the sites they use share their point of view, but 29% said that most of the sites they use challenge their point of view, indicating that many people are not simply looking for validations of their pre-existing opinions; and

- 8% posted their own political commentary online.

The user hunger for and reliance upon online advice and recommendations that the data above reveals is merely one reason behind the surge of interest in new systems that deal directly with opinions as a first-class object. But, Horrigan [127] reports that while a majority of American internet users report positive experiences during online product research, at the same time, 58% also report that online information was missing, impossible to find, confusing, and/or overwhelming.
Thus, there is a clear need to aid consumers of products and of information by building better information-access systems than are currently in existence.

With the explosion of Web 2.0 platforms such as blogs,

discussion forums, peer-to-peer networks, and various
other types of social media . . . consumers have at their
disposal a soapbox of unprecedented reach and power
by which to share their brand experiences and opinions,
positive or negative, regarding any product or service.
As major companies are increasingly coming to realize,
these consumer voices can wield enormous influence in
shaping the opinions of other consumers — and, ultimately,
their brand loyalties, their purchase decisions, and their own brand advocacy . . .
Companies can respond to the consumer insights they generate through
social media monitoring and analysis by modifying their 4 Introduction marketing
messages, brand positioning, product development, and other activities accordingly.
— Zabin and Jefferies [327]

Marketers have always needed to monitor media for information related to their brands — whether it's for public relations activities, fraud violations, or competitive intelligence. But fragmenting media and changing consumer behavior have crippled traditional monitoring methods.

Technorati estimates that 75,000 new blogs are created daily, along with 1.2 million new posts each day, many discussing consumer opinions on products and services. Tactics [of the traditional sort] such as clipping services, field agents, and ad hoc research simply can't keep pace.

1.2 Early History

Although the area of sentiment analysis and opinion mining has recently enjoyed a huge burst of research activity, there has been a steady undercurrent of interest for quite a while. One could count early projects on beliefs as forerunners of the area. Later work focused mostly on interpretation of metaphor, narrative, point of view, affect, evidentially in text, and related.

The year 2001 or so seems to mark the beginning of widespread awareness of the research problems and opportunities that sentiment analysis and opinion mining raise, and subsequently there have been literally hundreds of papers published on the subject.

Factors behind this “land rush” include:

- the rise of machine learning methods in natural language processing and information retrieval;

- the availability of datasets for machine learning algorithms to be trained on, due to the blossoming of the World Wide Web and, specifically, the development of review-aggregation web-sites; and, of course

- realization of the fascinating intellectual challenges and commercial and intelligence applications that the area offers.

1.2 Terminology: Opinion Mining, Sentiment Analysis, Subjectivity

‘The beginning of wisdom is the definition of terms,’
wrote Socrates. The aphorism is highly applicable when
it comes to the world of social media monitoring and
analysis, where any semblance of universal agreement
on terminology is altogether lacking.
Today, vendors, practitioners, and the media alike call
this still-nascent arena everything from ‘brand monitoring,’
‘buzz monitoring’ and ‘online anthropology,’ to
‘market influence analytics,’ ‘conversation mining’ and
‘online consumer intelligence’.

In the end, the term
‘social media monitoring and analysis’ is itself a verbal
crutch. It is placeholder [sic], to be used until something
better (and shorter) takes hold in the English language
to describe the topic of this report.
— Zabin and Jefferies [327]

The above quotation highlights the problems that have arisen in trying to name a new area. The quotation is particularly apt in the context of this survey because the field of “social media monitoring and analysis” (or however one chooses to refer to it) is precisely one that the body of work we review is very relevant to. And indeed, there has been to date no uniform terminology established for the relatively young field we discuss in this survey. In this section, we simply mention
some of the terms that are currently in vogue, and attempt to indicate what these terms tend to mean in research papers that the interested reader may encounter.

The body of work we review is that which deals with the computational treatment of (in alphabetical order) ***opinion***, ***sentiment***, and ***subjectivity*** in text. Such work has come to be known as ***opinion mining***, ***sentiment analysis***, and/or ***subjectivity analysis***. The phrases review mining and appraisal extraction have been used, too, and there are some connections to ***affective computing***, where the goals include enabling computers to recognize and express emotions. This proliferation of terms reflects differences in the connotations that these terms carry, both in their original general-discourse usages and in the usages that have evolved in the technical literature of several communities.

Synonyms: *opinion*, *view, belief, conviction, persuasion, sentiment* mean a judgment one holds as true.

- ***Opinion*** implies a conclusion thought out yet open to dispute each expert seemed to have a different opinion.

- View suggests a ***subjective opinion*** very assertive in stating his views.

- **Belief** implies often deliberate acceptance and intellectual assent a firm belief in her party's platform.

- **Conviction** applies to a firmly and seriously held belief the conviction that animal life is as sacred as human.

- **Persuasion** suggests a belief grounded on assurance (as by evidence) of its truth was of the persuasion that everything changes.

- **Sentiment** suggests a settled opinion reflective of one's feelings her feminist sentiments are well-known.

2
Purpose of Study

2.1 Aim

The area of Text Mining for sentiment analysis proposes to understand the opinions are distributed among them into the different categories of emotion like positive, negative, neutral.

Generally, most of the sentiment analysis work has been done on review sites till now. Review sites deliver with the sentiments of for business to analyze stock or any match review, thus restricting the domain of application is for specially business and other prediction. Thus, the venture into the corpus of twitter API allows us to move into different extents and various applications.

The most common applications of sentiment analysis are to refine from data alike *emotion, attitudes and feelings on the internet*, especially for *tracking products, services, brands or even people*.

Twitter posts in the field of sentiment analysis, is the next step as tweets give us a better and more diverse data resource of opinions and sentiments that can be whatever like the latest phone they bought, movie they watched, political issues, religious views or the individuals state of mind by extract tweets from twitter API.

As we can imagine twitter is one of the most common applications is to track attitudes and feelings on the web for sentiment analysis, especially for tacking products, services, brands or even people Sentiment analysis is also stated as Text or Opinion Mining that implies by extracting opinions, emotions and sentiments in text. The main idea is to analyze whether they are viewed positively or negatively by a given viewer.

For analyzing of data or to perform text mining first we have to conduct any kind of construction for a suitable dataset we have to build API.

API of twitter is an interface between the twitter apps (created by user) and the RStudio integration by its package and function which is specific tool for loading tweets from twitter in RStudio.

To construct a database of tweets on the keywords like #IPL2015 and #Dhoni that is to be build using a Twitter API app is the aim of this paper.

Machine learning techniques will regulate the number and score of tweets which are positive and negative.

The result of those machine learning techniques is used to confirm text analysis of the dataset. The knowledge based techniques that will achieve a series of analysis on the data which uses a sentiment lexicon dictionary to regulate the number of positive and negative tweets that will be executed in RStudio.

2.2 Project description

Twitter now-a-days, is most popular than compare to any other social networking sites like, Facebook, LinkedIn and YouTube.

Current growth in wide-area network connectivity potential has enormously grown opportunities for resource sharing and collaboration.

Other social media have expanded so much popularity and we cannot refuse them.

Other social media also have become the most important applications on Web.

Twitter have shaped the creation of matchless public collection of sentiments about every worldwide entity that is of notice.

It is obvious that the arrival of these real-time information networking sites like twitter may provision for an exceptional system for decision creation and presentation, it also poses challenges and the process which is incomplete without expert tools for analyzing those sentiments to accelerate their implementation of ideas with newer and different opinions.

Apart from Twitter, we will have some brief look into the Facebook and other social networking site for analyzing the customer behavior.

To analysis tweets this project choose with the hash tag, for example: #Dhoni and #IPL2015 as these are relevant for data collection at that time.

Associated work defines the associated literature that is revised everywhere in the topic of text mining and sentiment analysis and different methodologies has been used. The overall layout of the project will be achieved by Design and Architecture that describes how the design of the project is based on Bayesian classifier which is described in this section.

In Architectures figure provides a pathway design of the different distribution systems that will be used and that how they are associated to other systems. Application provides a step by step guide through the processes of the project which contains the code that was used as input as well as outputs.

This section defines the datasets used in this project. It also describes in detail how the data was picked up and the basic report of the data. It refers to where it is kept and how it will be used to answer the aims of the project. Requirements provide the aspects of the project that must be measured such as Functional requirements, Data requirements, User requirements etc. Datasets.

3

Approaches for Sentiment Analysis

There exist two main approaches to the problem of extracting sentiment automatically.

The **lexicon-based approach** involves calculating orientation for a document from the semantic orientation of words or phrases in the document. The text classification approach involves building classifiers from labeled instances of texts or sentences, essentially a supervised classification task.

The latter approach could also be described as a **statistical or machine-learning approach**.

3.1 Lexicon-based Approach

Lexicon techniques make use of existing lexical resources is controlled by classification models. It is used for bag-of-words tactic in sentiment analysis.

In this tactic the collection of words where are associations between words that are not considered important.

The document is treated to determine the global sentiment.

Those values are combined in sentiments of every word that are given for value and using combination of functions where tuples are phrases having positive or negative feelings may be considered as adjectives or adverbs which the polarity of words are found as review was extracted from the review based in the average semantic orientation of tuples.

Semantic orientation (SO) is a measure of subjectivity and opinion in text. It usually captures an evaluative factor (positive or negative) and potency or strength (degree to which the word, phrase, sentence, or document in question is positive or negative) towards a subject topic, person, or idea. When used in the analysis of public opinion, such as the automated interpretation of on-line product reviews, semantic orientation can be extremely helpful in marketing, measures of popularity and success, and compiling reviews.

In this article, sentiment analysis refers to the general method to extract subjectivity and polarity from text (potentially also speech), and semantic orientation refers to the polarity and strength of words, phrases, or texts. Our concern is primarily with the semantic orientation of texts, but we extract the sentiment of words and phrases towards that goal.

Area where the lexicon-based model might be preferable to a classifier model is in simulating the effect of linguistic context. On reading any document, it becomes apparent that aspects of the local context of a word need to be taken into account in SO assessment, such as negation (e.g., not good) and intensification (e.g., very good), aspects that Polanyi and Zaenen (2006) named contextual valence shifters. Research by Kennedy and Inkpen (2006) concentrated on implementing those insights. They dealt with negation and intensification by creating separate features, namely, the appearance of good might be either **good** (*no modification*) **not good** (*negated good*), **int good** (*intensified good*), or **dim good** (*diminished good*). The classifier, however, cannot determine that these four types of good are in any way related, and so in order to train accurately there must be enough examples of all four in the training corpus.

In our approach, we seek methods that operate at a deep level of analysis, incorporating semantic orientation of individual words and contextual valence shifters, yet do not aim at a full linguistic analysis (one that involves analysis of word senses or argument structure), although further work in that direction is possible.

3.1.1 Adjectives

Much of the early research in sentiment focused on adjectives or adjective phrases as the primary source of subjective content in a document; adjectives and verbs; two-word phrases; the use of non-affective adjectives and adverbs; or rationales, words and phrases selected by human annotators.

In general, the SO of an entire document is the combined effect of the adjectives or relevant words found within, based upon a dictionary of word rankings (scores).

The dictionary can be created in different ways: manually, using existing dictionaries such as the General Inquirer (Stone et al. 1966), or semi-automatically, making use of resources like WordNet. The dictionary may also be produced automatically via association, where the score for each new adjective is calculated using the frequency of the proximity of that adjective with respect to one or more seed words.

Seed words are a small set of words with strong negative or positive associations, such as excellent or abysmal.

In principle, a positive adjective should occur more frequently alongside the positive seed words, and thus will obtain a positive score, whereas negative adjectives will occur most often in the vicinity of negative seed words, thus obtaining a negative score. The association is usually calculated following Turney's method for computing mutual information.

Classification of Adjectives:

Hatzivassiloglou and McKeown's (1997) **algorithm** classifies adjectives into those with positive or negative semantic orientation:

Semantic Polarity of an adjective:

Direction:

In which direction does the referent deviate from the norm in its semantic field?

Evaluative:

Is this good or bad?

If we know that two adjectives relate to the same property (e.g., hot and cold) but have different orientations they are usually antonyms.

Coordinated adjectives:
Extract from POS tagged WSJ (21 million words) adjective pairs coordinated by and, or, but, either-or, neither-nor. This results in 15048 adjective pairs (token); 9296 (type) Number of those where orientation of both partners is known (via gold standard): 4024 (token); 2748 (type)

Dictionaries:
Automatically or semi-automatically created dictionaries have some advantages.
We found many novel words in our initial Google-generated dictionary. For instance, unlistenable was tagged accurately as highly negative, an advantage that Velikovich et al. (2010) point out. However, in light of the lack of stability for automatically generated dictionaries, we decided to create manual ones. These were produced by hand-tagging all adjectives found in our development corpus, a 400-text corpus of reviews (see the following) on a scale ranging from −5 for extremely negative to +5 for extremely positive, where 0 indicates a neutral word (excluded from our dictionaries).

“Positive” and “negative” were decided on the basis of the word’s prior polarity, that is, its meaning in most contexts. We do not deal with word sense disambiguation but suspect that using even a simple method to disambiguate would be beneficial.

Some word sense ambiguities are addressed by taking part of speech into account.

For instance, plot is only negative when it is a verb, but should not be so in a noun dictionary; novel is a positive adjective, but a neutral noun.

3.1.2 Nouns, Verbs, and Adverbs

In the following example, we see that lexical items other than adjectives can carry important semantic polarity information.

Example (1)
a. The young man strolled **+** purposefully **+** through his neighborhood **+**.
b. The teenaged male strutted **–** cockily **–** through his turf **–**.

Although the sentences have comparable literal meanings, the plus-marked nouns,
verbs, and adverbs in Example (**1a**) ***indicate the positive orientation*** of the speaker
towards the situation, whereas the *minus-marked words* in Example (**1b**) have the
opposite effect. It is the combination of these words in each of the sentences that conveys the semantic orientation for the entire sentence.

In order to make use of this additional information, we created separate noun, verb, and adverb dictionaries, hand-ranked using the same **+5 to −5** scale as our adjective dictionary.

The enhanced dictionaries contain **2,252 adjective** entries, **1,142 nouns**, **903 verbs**, and **745 adverbs**.

The SO-carrying words in these dictionaries were taken from a variety of sources, the three largest being Epinions 1, the 400-text corpus described in the previous section; a 100-text subset of the 2,000 movie reviews in the Polarity Dataset; and positive and negative words from the General Inquirer dictionary (Stone et al. 1966; Stone 1997). The sources provide a fairly good range in terms of register: The Epinions and movie reviews represent informal language, with words such as ass-kicking and nifty; at the other end of the spectrum, the General Inquirer was clearly built from much more formal texts, and contributed words such as adroit and jubilant, which may be more useful in the processing of literary reviews or other more formal texts.

Each of the open-class words was assigned a hand-ranked SO value between 5 and −5 (neutral or zero-value words were excluded) by a native English speaker.

The numerical values were chosen to reflect both the prior polarity and the strength of the word, averaged across likely interpretations. The dictionaries were later reviewed by a committee of three other researchers in order to minimize the subjectivity of ranking SO by hand. Examples are shown in Table 1.

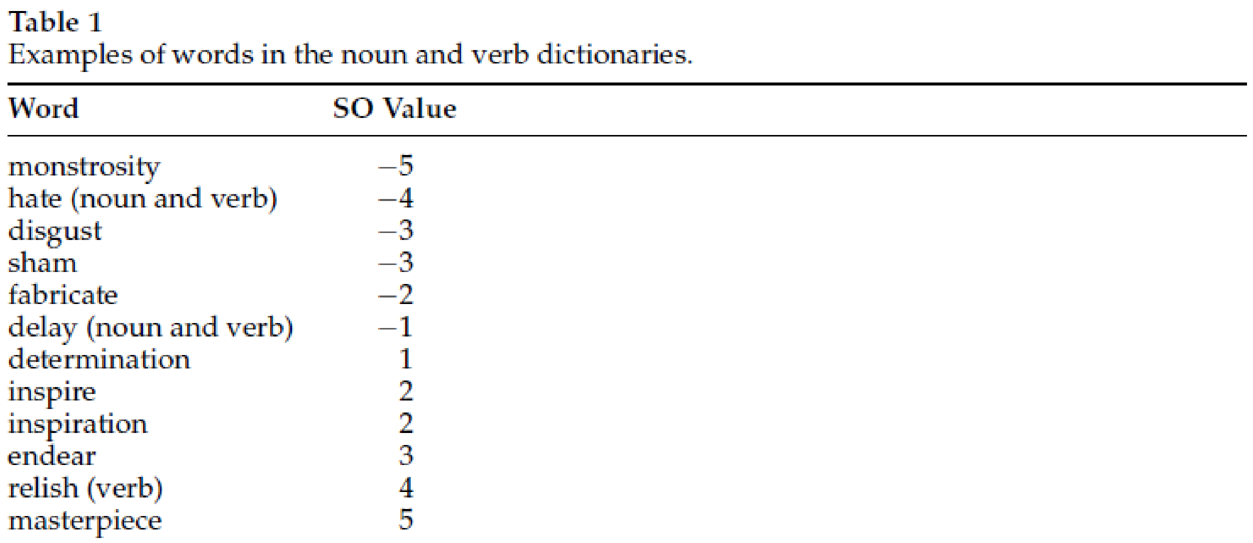


Table 1
Examples of words in the noun and verb dictionaries.

Word	SO Value
monstrosity	−5
hate (noun and verb)	−4
disgust	−3
sham	−3
fabricate	−2
delay (noun and verb)	−1
determination	1
inspire	2
inspiration	2
endear	3
relish (verb)	4
masterpiece	5

Except when one sense was very uncommon, the value chosen reflected an averaging across possible interpretations. In some cases, the verb and related noun have a different SO value.

For instance, **exaggerate is −1**, whereas **exaggeration is −2**, and the same values are applied to *complicate and complication*, respectively.

We find that grammatical metaphor, that is, the use of a noun to refer to an action, adds a more negative connotation to negative words.

All nouns and verbs encountered in the text are lemmatized, and the form (**singular or plural, past tense or present tense**) is **not taken into account in the calculation of SO value**.

As with the adjectives, there are more negative nouns and verbs than positive ones.

The adverb dictionary was built automatically using our adjective dictionary, by
matching adverbs ending in -ly to their potentially corresponding adjective, except
for a small selection of words that were added or modified by hand.

When SO-CAL encountered a word tagged as an adverb that was not already in its dictionary, it would stem the word and try to match it to an adjective in the main dictionary. This worked quite well for most adverbs, resulting in semantic orientation values that seem appropriate (see examples in Table 2).

Table 2
Examples from the adverb dictionary.

Word	SO Value
excruciatingly	−5
inexcusably	−3
foolishly	−2
satisfactorily	1
purposefully	2
hilariously	4

3.1.3 Intensification

Quirk et al. (1985) classify intensifiers into two major categories, depending on their polarity: **Amplifiers** (e.g., very) increase the semantic intensity of a neighboring lexical item, whereas **downtoners** (e.g., slightly) decrease it.

Some researchers in sentiment analysis have implemented intensifiers using simple addition and subtraction—that is, if a positive adjective has an SO value of 2, an amplified adjective would have an SO value of 3, and a down toned adjective an SO value of 1. One problem with this kind of approach is that it does not account for the wide range of intensifiers within the same subcategory.

Extraordinarily, for instance, is a much stronger amplifier than rather. Another concern is that the amplification of already "loud" items should involve a greater overall increase in intensity when compared to more subdued counterparts (compare truly fantastic with truly okay); in short, intensification should also depend on the item being intensified.

In SO, intensification is modeled using modifiers, with each intensifying word having a percentage associated with it; amplifiers are positive, whereas downtoners are negative, as shown in Table 3.

Table 3
Percentages for some intensifiers.

Intensifier	Modifier (%)
slightly	−50
somewhat	−30
pretty	−10
really	+15
very	+25
extraordinarily	+50
(the) most	+100

For example, if sleazy has an SO value of −3, somewhat sleazy would have an SO value of:

−3 × (100% − 30%) = −2.1. If excellent has a SO value of 5, most excellent would have an SO value of: 5 × (100% + 100%) = 10. Intensifiers are applied recursively starting from the closest to the SO-valued word: If good has an SO value of 3, then really very good has an SO value of (3 × [100% + 25%]) × (100% + 15%) = 4.3.

Because our intensifiers are implemented using a percentage scale, they are able to fully capture the variety of intensifying words as well as the SO value of the item being modified.

This scale can be applied to other parts of speech, given that adjectives, adverbs, and verbs use the same set of intensifiers, as seen in Example (2), where really modifies an **adjective (fantastic)**, an **adverb (well)**, and a **verb (enjoyed)**.

Example (2)

a. The performances were all really fantastic.

b. Zion and Planet Asai from the Cali Agents flow really well over this.

c. I really enjoyed most of this film.

Besides adverbs and adjectives, other intensifiers are quantifiers (a great deal of). We also included three other kinds of intensification that are common within our genre: the use of all capital letters, the use of exclamation marks, and the use of discourse connective but to indicate more salient information (e.g., ...but the movie was GREAT!).

In all, our intensifier dictionary contains 177 entries, some of them multi-word expressions.

3.1.3 Negation

The obvious approach to negation is **simply to reverse the polarity** of the lexical item next to a negator, **changing good (+3) into not good (−3)**. This we may refer to as switch negation. There are a number of subtleties related to negation that need to be taken into account, however. One is the fact that there are negators, including not, none, nobody, never, and nothing, and other words, such as without or lack (verb and noun), which have an equivalent effect, some of which might occur at a significant distance from the lexical item which they affect; a backwards search is required to find these negators, one that is tailored to the particular part of speech involved.

We assume that for adjectives and adverbs the negation is fairly local, though it is sometimes necessary to look past determiners, copulas, and certain verbs, as we see in Example (3).

Example (3)
a. Nobody gives a good performance in this movie. (nobody negates good)
b. Out of every one of the fourteen tracks, none of them approach being weak
and are all stellar. (none negates weak)
c. Just a V-5 engine, nothing spectacular. (nothing negates spectacular)

Negation search in SO includes two options:
Look backwards until a clause boundary marker is reached; or look backwards as long as the words/tags found are in a backward search skip list, with a different list for each part of speech. The first approach is fairly liberal, and allows us to capture the true effects of negation raising (Horn 1989), where the negator for a verb moves up and attaches to the verb in the matrix clause.

In the following examples the don’t that negates the verb think is actually negating the embedded clause.

Example (4)
I don’t wish to reveal much else about the plot because I don’t think it is worth
mentioning.

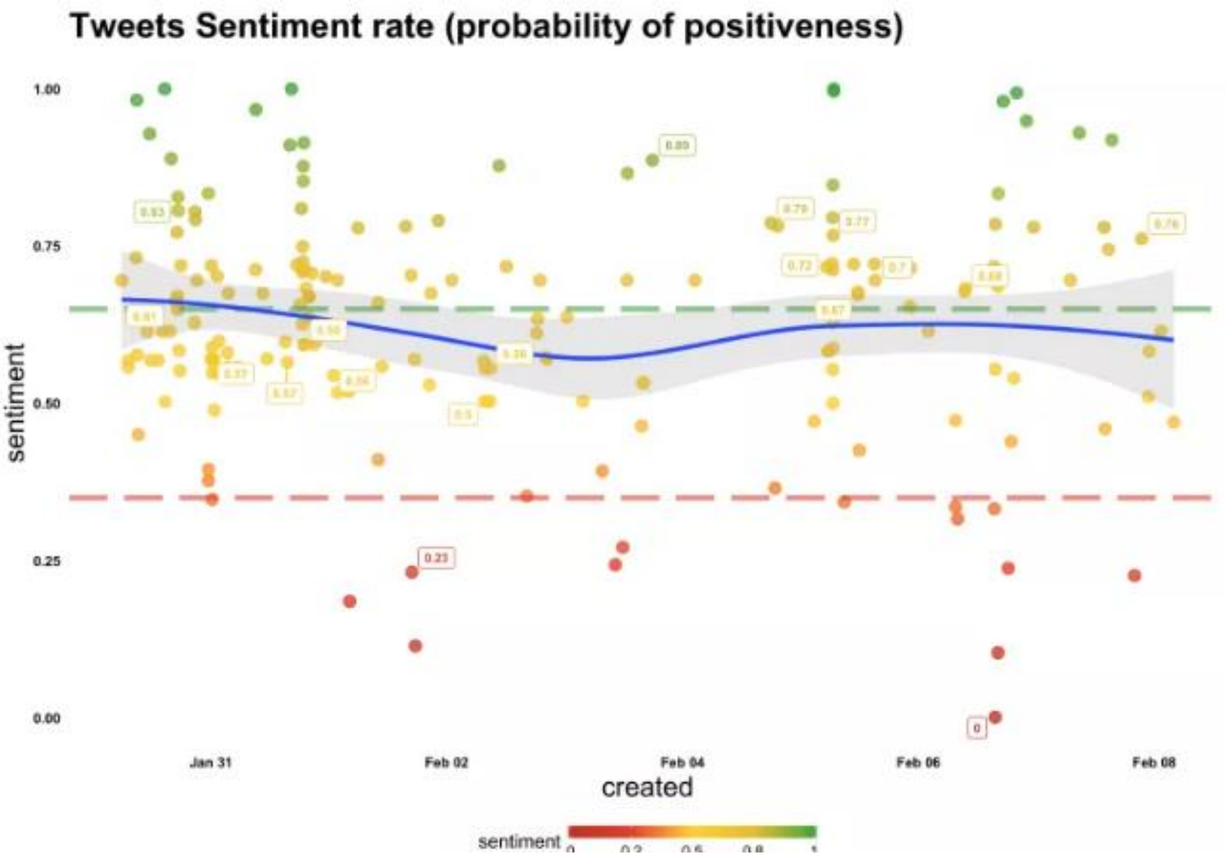
The second approach is more conservative. In Example (4), the search would only go as far as it, because adjectives (worth), copulas, determiners, and certain basic verbs are on the list of words to be skipped (allowing negation of adjectives within VPs and NPs), but pronouns are not. Similarly, verbs allow negation on the other side of to, and nouns look past adjectives as well as determiners and copulas. This conservative approach seems to work better, and is what we use in all the experiments in this article.

3.2 Machine Learning Approach

The Knowledge based methods are not depended on lexicon dataset for Machine Learning technique, in this methodology test set are in order to classify an employed, instead the use of a training set. In classification model it is developed by set of methodology using a training set, which tries to classify the input feature as vectors into matching class labels. The face of every changing social network language lexicons allows the algorithm to remain dynamic in nature.

The class labels of hidden feature vectors are drawn in the training set as test set is used to verify the models by predicting. **Support Vector Machines** (SVM) and **Bayesian** classifier are used to classify reviews into either positive or negative orientation in number of machine learning techniques.

Example:



The problem with the lexical-based method is that it just computes the number of positive and negative words and makes a conclusion based on their difference. Therefore, when using a simple vocabularies approach for a phrase “not bad” we’ll get a negative estimation.

But **doc2vec** is a deep learning algorithm that draws context from phrases. It’s currently one of the best ways of sentiment classification for movie reviews. You can use the following method to analyze feedbacks, reviews, comments, and so on. And you can expect better results comparing to tweets analysis because they usually include lots of misspelling.

3.2.1 Supervised Classification

- Choosing the correct class label for a given input.
- Classifier is supervised if it is build based on a training corpus containing the correct label for each input.
- Creating a classifier deciding which features of the input are relevant.

Generative vs. Conditional Classifiers

- Generative Classifiers predict **P (input, label)** (e.g. Naïve Bayes)
- Conditional Classifiers predict **P (label | input)** (e.g. Maximum Entropy)

Questions that these classifiers may answer

2 What is the most likely label for an input? (G,C)
3 How likely is a given label for a given input? (G,C)
4 What is the most likely input value? (G)
5 How likely is a given input value? (G)
6 How likely is a given input value with a given label? (G)
7 What is the most likely label for an input that might have one of two values, but we won't know? (G)

Evaluation - General

- We need to find a way to decide how well our classification model works
- Classic evaluation approach: training set, test set, evaluation set (distinct and non-overlapping)
- We know the correct labels and can assess performance of our classification model
- Evaluation set should be balanced as to which labels occur and quite general
- Size of evaluation set can be important as well
- Special cases: Cross validation and bootstrap sampling

Features

- Means something different than we would think
- We distinguish between the label l and property some *fi* of input *x*
- *fi* is called “feature” (e.g. end with letter” a”)
- We need to define a combination of labels and properties (“joint-feature”)

$$g(x, l) = \begin{cases} 1 & \text{if } f_i = a, (i = 1, \dots, k) \\ 0 & \text{otherwise} \end{cases}$$

3.2.2 Naive Bayes: Sentiment Classification

we introduce the **Multinomial Naive Bayes classifier**, so called because it is a **Bayesian** classifier that makes a simplifying (naive) assumption about how the features interact.

Naive Bayes is a probabilistic classifier, meaning that for a document ***d***, out of all classes ***c* ∈*C*** the classifier returns the class **c**ˆ which has the maximum posterior probability given the document. In following equation, we use the hat notation ˆ to mean “our estimate of the correct class”.

$$\hat{c} = \text{argmax } \mathbf{P}(c|d) \textbf{ for all } c \in C$$

This idea of Bayesian inference has been known since the work of Bayes (1763), and was first applied to text classification by Mosteller and Wallace (1964). The in- tuition of Bayesian classification is to use Bayes’ rule to transform Eq. 6.1 into other probabilities that have some useful properties. Bayes’ rule, it gives us a way to break down any conditional probability **P(x|y)** into three other probabilities:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

We can then substitute; we get:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)}$$

We can conveniently simplify by dropping the denominator P(d). This is possible because we will be computing $\frac{P(d|c)P(c)}{P(d)}$ P(d) for each possible class.

But P(d) doesn’t change for each class; we are always asking about the most likely class for the same document d, which must have the same probability P(d).

Thus, we can choose the class that maximizes this simpler formula:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)$$

We thus compute the most probable class cˆ given some document d by choosing the class which has the highest product of two probabilities: the prior probability of the class P(c) and the likelihood of the document P(d|c):

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \; \overbrace{P(d|c)}^{\text{likelihood}} \; \overbrace{P(c)}^{\text{prior}}$$

Without loss of generalization, we can represent a document *d* as a set of features *f*1, *f*2,…, *fn:*

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \overbrace{P(f_1, f_2, ..., f_n|c)}^{\text{likelihood}} \; \overbrace{P(c)}^{\text{prior}}$$

Unfortunately, the above equation is still too hard to compute directly: without some simplifying assumptions, estimating the probability of every possible combination of features (for example, every possible set of words and positions) would require huge numbers of parameters and impossibly large training sets. Naive Bayes classifiers therefore make two simplifying assumptions.

The first is the bag of words assumption discussed intuitively above: we assume position doesn't matter, and that the word "love" has the same effect on classification whether it occurs as the 1st, 20th, or last word in the document. Thus we assume that the features *f1, f2, ..., fn* only encode word identity and not position.

The second is commonly called the naive Bayes assumption: this is the conditional independence assumption that the probabilities **P (*fi* |*c*)** are independent given the class c and hence can be 'naively' multiplied as follows:

$$P(f_1, f_2, ..., f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot ... \cdot P(f_n|c)$$

The final equation for the class chosen by a naive Bayes classifier is thus:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f|c)$$

To apply the naive Bayes classifier to text, we need to consider word positions, by simply walking an index through every word position in the document:

$$\begin{aligned} \text{positions} &\leftarrow \text{all word positions in test document} \\ c_{NB} &= \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i \in positions} P(w_i|c) \end{aligned}$$

Naive Bayes calculations, like calculations for language modeling, are done in log space, to avoid underflow and increase speed. Thus above equation is generally expressed as:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

Linear Classifiers:

By considering features in log space above equation computes the predicted class as a linear function of input features. Classifiers that use a linear combination of the inputs to make a classification decision —like **naive Bayes** and also **logistic regression**— are called **linear classifiers**.

3.2.3 Training the Naive Bayes Classifier

How can we learn the probabilities P(*c*) and P (*fi* |*c*)? Let's first consider the maximum likelihood estimate. We'll simply use the frequencies in the data.
For the document prior P(c) we ask what percentage of the documents in our training set are in each **class c**. Let N_c be the number of documents in our training data with **class c** and N_{doc} be the total number of documents.

Then:

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

To learn the probability **P (*fi* |*c*)**, we'll assume a feature is just the existence of a word in the document's bag of words, and so we'll want **P (*wi* |*c*)**, which we compute as the fraction of times the word wi appears among all words in all documents of topic c. We first concatenate all documents with category c into one big "category c" text.

Then we use the frequency of wi in this concatenated document to give a maximum likelihood estimate of the probability:

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}$$

Here the vocabulary V consists of the union of all the word types in all classes, not just the words in one class c.

There is a problem, however, with maximum likelihood training. Imagine we are trying to estimate the likelihood of the word "fantastic" given class positive, but suppose there are no training documents that both contain the word "fantastic" and are classified as positive. Perhaps the word "fantastic" happens to occur (sarcastically?) in the class negative.

In such a case the probability for this feature will be zero:

$$\hat{P}(\text{“fantastic”}|\text{positive}) = \frac{count(\text{“fantastic”}, \text{positive})}{\sum_{w \in V} count(w, \text{positive})} = 0$$

But since naive Bayes naively multiplies all the feature likelihoods together, zero probabilities in the likelihood term for any class will cause the probability of the class to be zero, no matter the other evidence!

The simplest solution is the add-one (Laplace) smoothing. While Laplace smoothing is usually replaced by more sophisticated smoothing algorithms in language modeling, it is commonly used in naive Bayes text categorization:

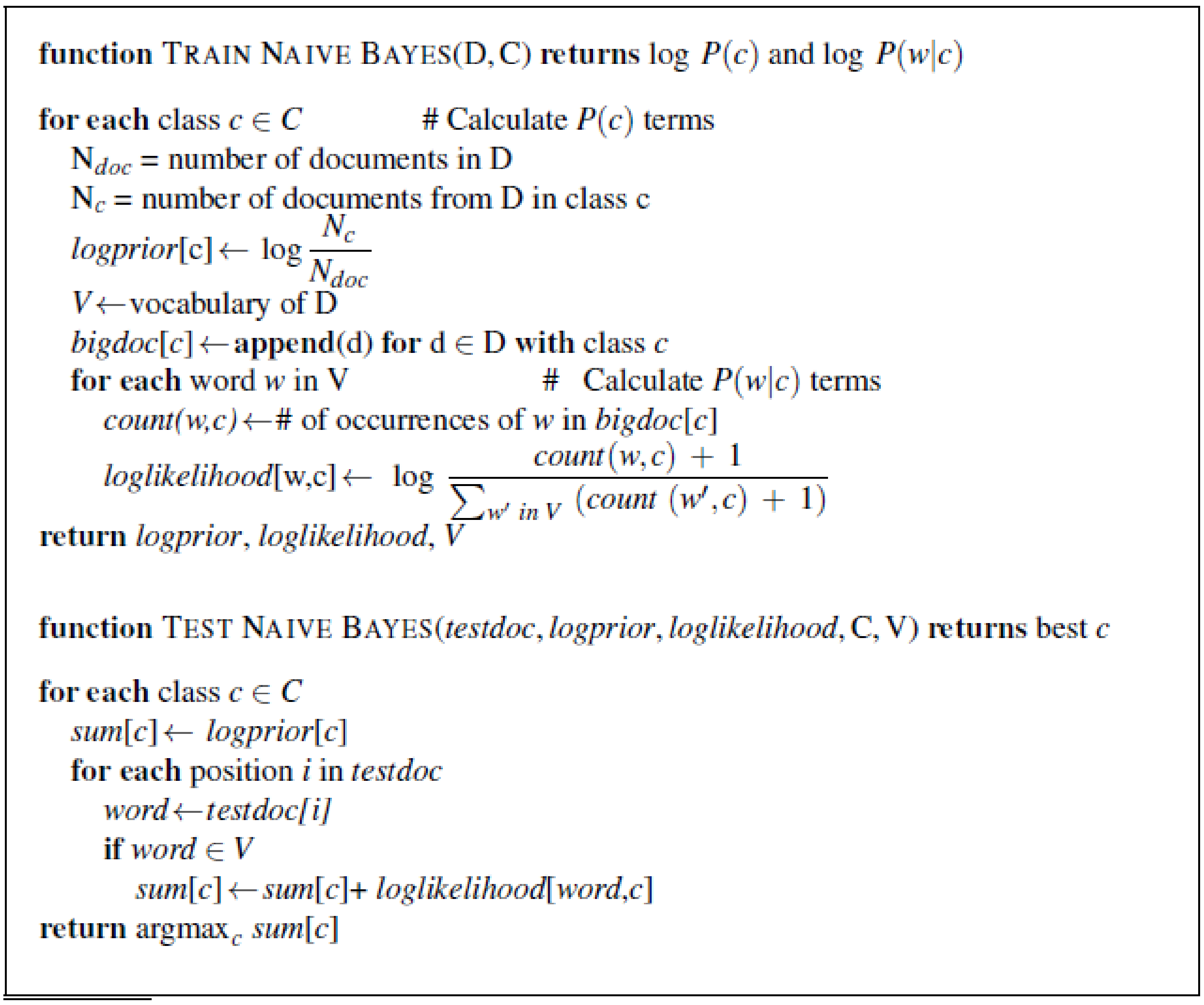
$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} (count(w, c) + 1)} = \frac{count(w_i, c) + 1}{\left(\sum_{w \in V} count(w, c)\right) + |V|}$$

Note once again that it is a crucial that the vocabulary **V** consists of the union of all the word types in all classes, not just the words in one **class c**. What do we do about words that occur in our test data but are not in our vocabulary at all because they did not occur in any training document in any class? The standard solution for such unknown words is to ignore such words—remove them from the test document and not include any probability for them at all.

Finally, some systems choose to completely ignore another class of words: stop words, very frequent words like *the* and *a*. This can be done by sorting the vocabulary by frequency in the training set, and defining the top 10–100 vocabulary entries as stop words, or alternatively by using one of the many pre-defined stop word list available online. Then every instance of these stop words are simply removed from both training and test documents as if they had never occurred.

In most text classification applications, however, using a stop word list doesn’t improve performance, and so it is more common to make use of the entire vocabulary and not use a stop word list.

FINAL ALGORITHM:



```
function TRAIN NAIVE BAYES(D, C) returns log $P(c)$ and log $P(w|c)$

for each class $c \in C$             # Calculate $P(c)$ terms
    $N_{doc}$ = number of documents in D
    $N_c$ = number of documents from D in class c
    $logprior[c] \leftarrow \log \frac{N_c}{N_{doc}}$
    $V \leftarrow$ vocabulary of D
    $bigdoc[c] \leftarrow$ append(d) for d $\in$ D with class $c$
    for each word $w$ in V                    #  Calculate $P(w|c)$ terms
        $count(w,c) \leftarrow$ # of occurrences of $w$ in $bigdoc[c]$
        $loglikelihood[w,c] \leftarrow \log \frac{count(w,c)\ +\ 1}{\sum_{w' \text{ in } V} (count\ (w',c)\ +\ 1)}$
return $logprior$, $loglikelihood$, $V$


function TEST NAIVE BAYES($testdoc$, $logprior$, $loglikelihood$, C, V) returns best $c$

for each class $c \in C$
    $sum[c] \leftarrow logprior[c]$
    for each position $i$ in $testdoc$
        $word \leftarrow testdoc[i]$
        if $word \in V$
            $sum[c] \leftarrow sum[c] + loglikelihood[word,c]$
return $\text{argmax}_c\ sum[c]$
```

3.2.2 Naive Bayes as a Language Model

Naive Bayes classifiers can use any sort of feature: dictionaries, URLs, email ad- dresses, network features, phrases, parse trees, and so on. But if, as in the previous section, we use only individual word features, and we use all of the words in the text (not a subset), then naive Bayes has an important similarity to language modeling. Specifically, a naive Bayes model can be viewed as a set of class-specific unigram language models, in which the model for each class instantiates a unigram language model.

Since the likelihood features from the naive Bayes model assign a probability to each word P (*word | c*), the model also assigns a probability to each sentence:

$$P(s|c) = \prod_{i \in positions} P(w_i|c)$$

Thus consider a naive Bayes model with the classes positive (+) and negative (-)
and the following model parameters:

w	P(w|+)	P(w|-)
I	0.1	0.2
love	0.1	0.001
this	0.01	0.01
fun	0.05	0.005
film	0.1	0.1
...	...	...

Each of the two columns above instantiates a language model that can assign a probability to the sentence “I love this fun film”:

P (” I love this fun film” |+) = 0.1 × 0.1 × 0.01 × 0.05 × 0.1 = 0.0000005
P (” I love this fun film” |−) = 0.2 × 0.001 × 0.01 × 0.005 × 0.1 = .0000000010

3.3 Hadoop Approach

Map Reduce Concept

Hadoop is processing system. Pig executes on Hadoop. The Hadoop Distributed File System makes use of both *HDFS* and, ***Map Reduce***.

A distributed file system that stores files on all of the nodes in a Hadoop cluster is said to be HDFS. The file are handles by distributing them across different machines by breaking it in the files into large blocks. It handles file including multiple copies of each block so that if any one of machine fails no data is lost. By default, Pig reads input files from HDFS and used HDFS to store intermediate data between Map Reduce jobs, and writes its output to HDFS. As we will see it can also read input from and write output to sources other than HDFS. It grants a POSIX-like interface to users in HDFS.

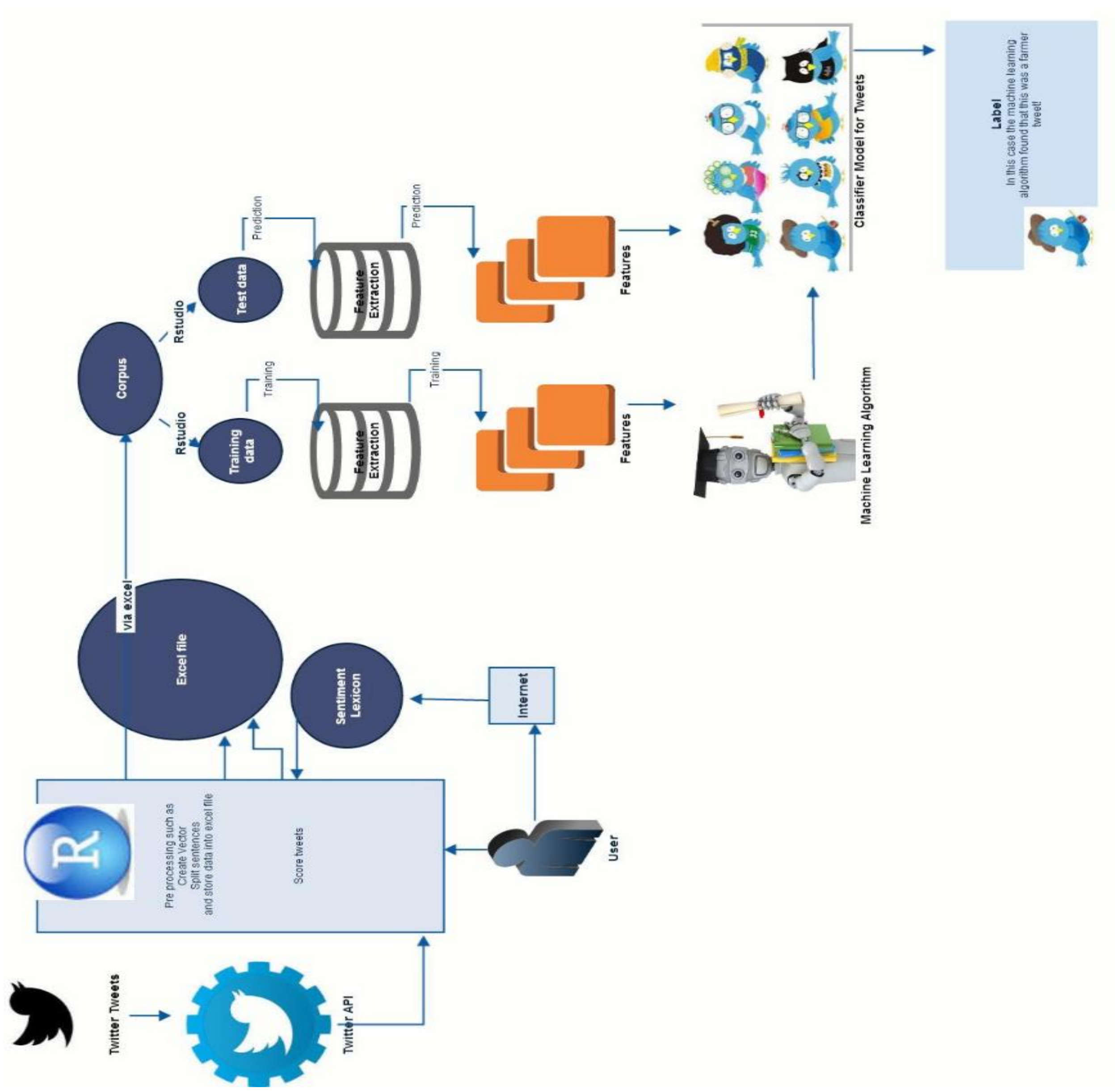
A simple but the powerful parallel data-processing paradigm is said to be Map Reduce. There are commonly three main stages: map, shuffle, and reduce in every job in Map Reduce which consists of data. The application has the prospect to work on each record in the input individually in the map phase. The map phase can generally be accomplished in one minute because if any maps are started at once it can input file size of gigabytes (GB) or terabytes (TB) in machines.

4

System Design and Datasets

4.1 Design and Architecture

The proposed architecture **using R** for this project is shown in below figure:



The sentiment orientation of the tweets is accessed through a Twitter API.

The knowledge based methodologies and machine learning based methodologies is used to analysis the two methods for suggesting hybrid tactics.

The words were scored from the internet and was openly available which was sourced into their relative sentiment orientation using a sentiment lexicon. The words which were extracted and stored in a feature vector.

Feature extraction is used by machine learning algorithm that was produced from the test set-feature extraction.

From the training set a Bayesian classifier was produced as in the form of model classifier. The associations between features such as emotional keywords and emoticons of Bayesian classifier does not consider in R. The collected reviews are used in R and it is saved to an excel file with .csv file and corpus was produced the excel file was then entered into the corpus was then separated into a training set and a test set and feature were extracted from each respectively. These features do not always relate with the use of a smiley emoticon at the end of a negative tweet. It is ideal for sentiment analysis as often to one another such as in Bayesian Classifier analyzes each of the features of the feature vector individually as it assumes that they are equally independent of each other.

The conditional probability for Bayesian can be defined as

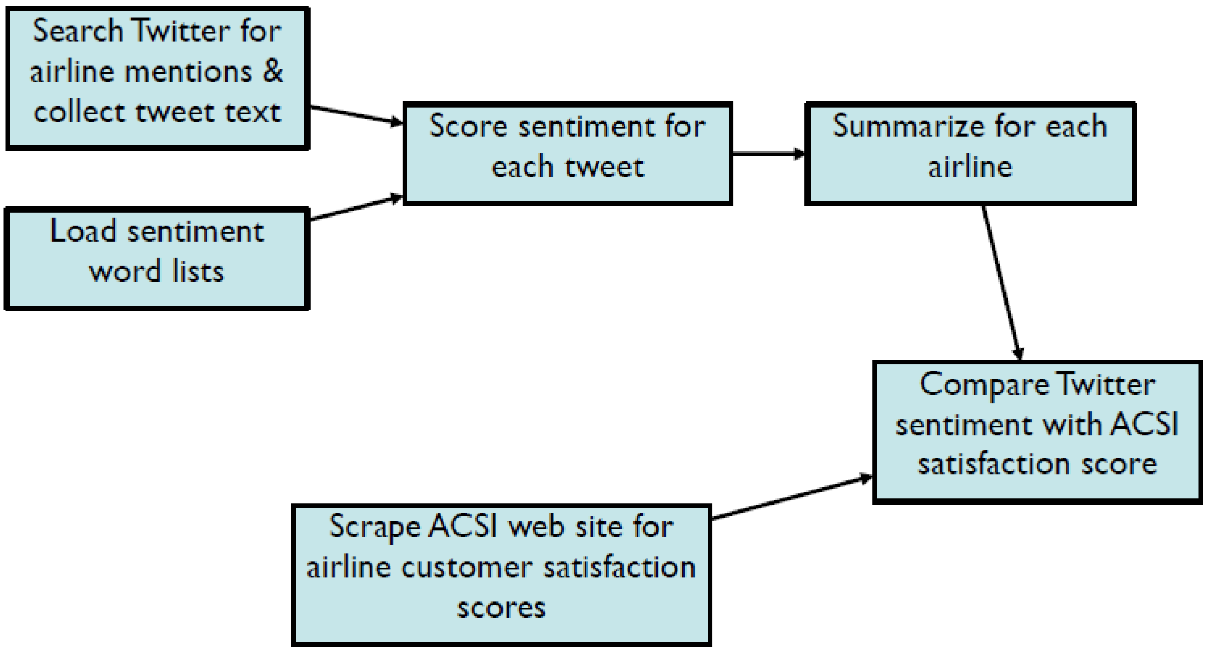
$$P (X \mid yj) = m$$

$$P (xi \mid yj)$$

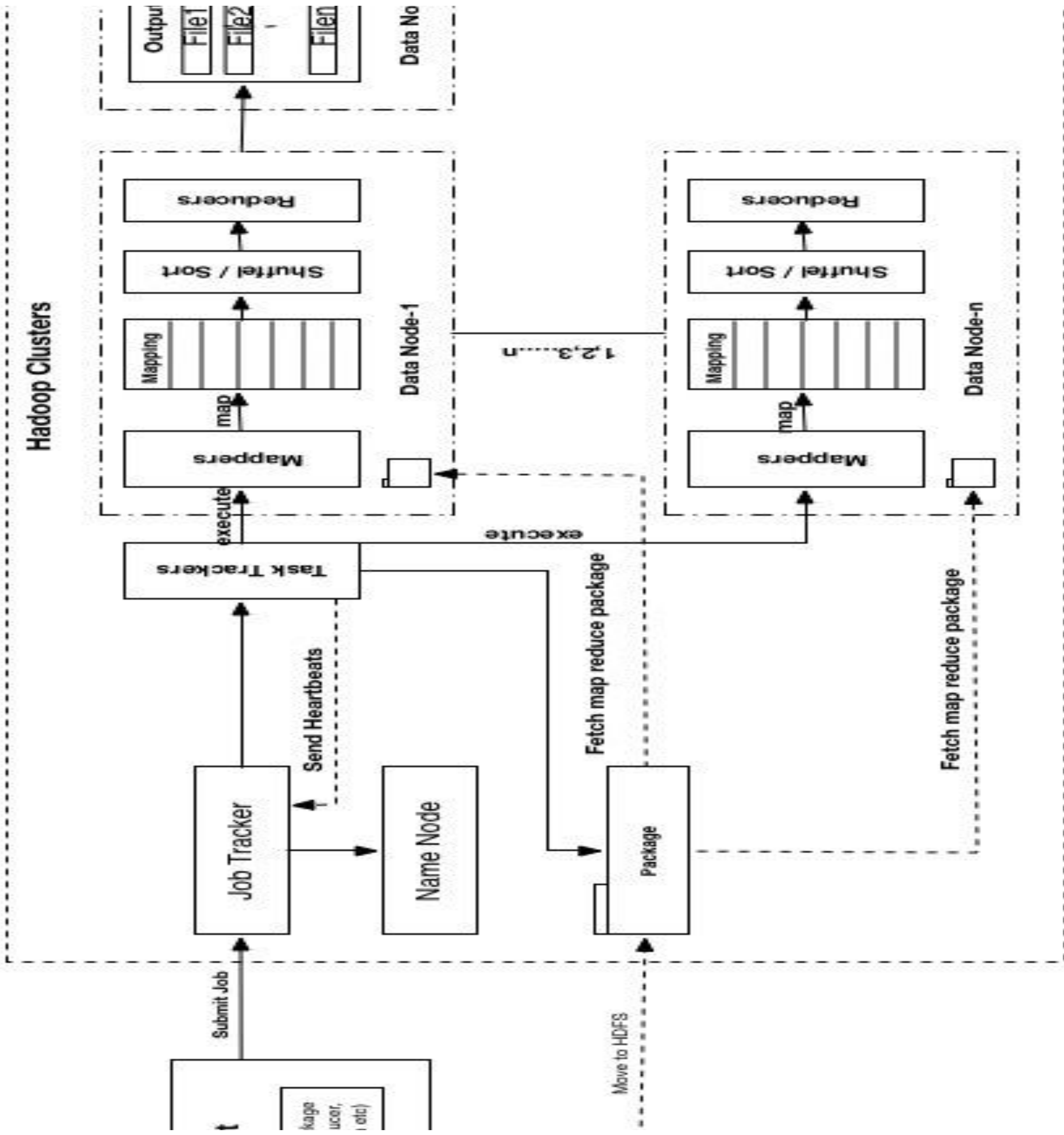
’X’ is the feature vector defined as X= {x1, x2, xm} and yj is the class label.

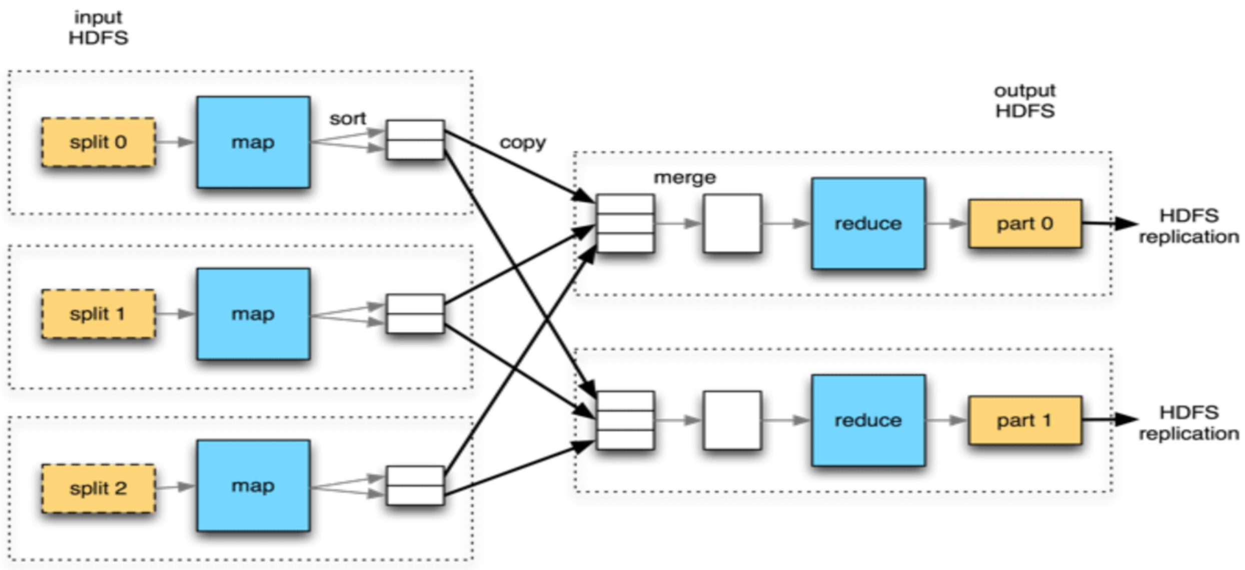
There are unlike many features such as emoticons, emotional keyword, which are treated as either positive or negative and utilized by Bayesian classifier for classification from the tweets that was collected for this project.

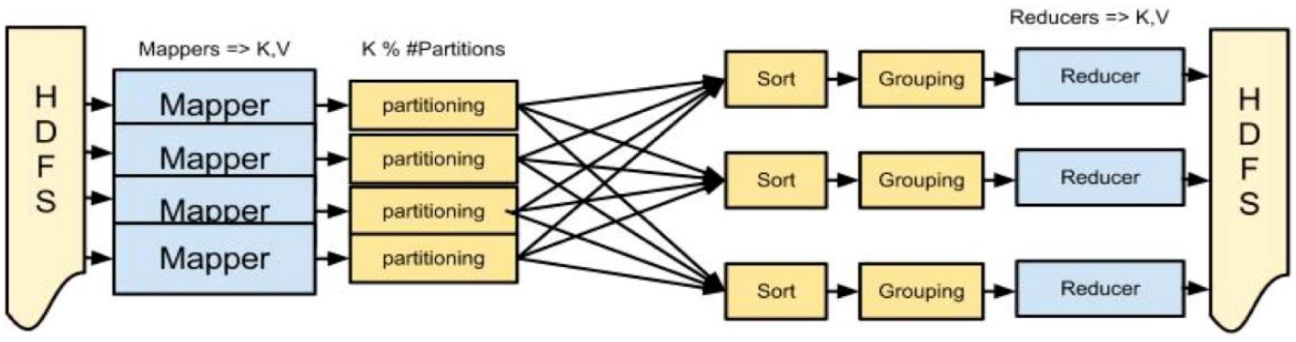
Game Plan


Search Twitter for airline mentions & collect tweet text
Load sentiment word lists
Score sentiment for each tweet
Summarize for each airline
Compare Twitter sentiment with ACSI satisfaction score
Scrape ACSI web site for airline customer satisfaction scores

Using Map Reduce:







The MapReduce Pipeline

A mapper receives (Key, Value) & outputs (Key, Value)
A reducer receives (Key, Iterable[Value]) and outputs (Key, Value)
Partitioning / Sorting / Grouping provides the Iterable[Value] & Scaling

4

Social Network Analytics

4.1 Online Social Networks: Research Issues

The field of online social networks has seen a rapid revival in recent years.
A key aspect of many of the online social networks is that they are rich in data,
and provide unprecedented challenges and opportunities from the perspective
of knowledge discovery and data mining.

There are two primary kinds of data which are often analyzed in the context of social networks:

Linkage-based and Structural Analysis:
In linkage-based and structural analysis, we construct an analysis of the linkage behavior of the network in order to determine important nodes, communities, links, and evolving regions of the network. Such analysis provides a good overview
of the global evolution behavior of the underlying network.

Adding Content-based Analysis:
Many social networks such as Flickr, Message Networks, and YouTube contain a tremendous amount of content which can be leveraged in order to improve the quality of the analysis.

For example, a photograph sharing site such as Flickr contains a tremendous
amount of text and image information in the form of user-tags and
images. Similarly, blog networks, email networks and message boards
contain text content which are linked to one another. It has been observed that combining content-based analysis with linkage-based analysis provides more effective results in a wide variety of applications.

For example, communities which are designed with text-content are much richer in terms of carrying information about the topical expertise of the underlying community.

The **other main differences** which arises in the context of **social network algorithms** is that between **dynamic analysis and static analysis**. In the case of **static analysis**, we assume that the social network changes slowly over time, and we perform an analysis of the whole network in batch mode over particular snapshots.

Dynamic networks also arise in the context of **mobile applications**, in which moving entities constantly interact with one another. Such dynamic networks
arise in the context of moving entities which interact with one another over time. For example, many mobile phones are equipped with GPS receivers, which are exploited by the applications on these phones. A classic example of such an application is the **Google Latitude** application which is capable of keeping track of the locations of different users, and issuing alerts when a given user is in the vicinity.

The most ***well-known structural problem in the context of social networks is that of community detection***. The problem of community detection is closely related to the ***problem of finding structurally related groups in the network***. These structurally related groups are referred to as **communities**.

Social networks can be viewed as a structure which enables the dissemination of information. This is direct corollary of its enabling of social interactions among individuals. The analysis of the dynamics of such interaction is a challenging problem in the field of social networks.

For example, important news is propagated through the network with the use of the interactions between the different entities. A well-known model for influence propagation may be found. The problem of influence analysis is very relevant in the context of social networks, especially in the context of determining the most influential members of the social network, who

are most likely to propagate their influence to other entities in the social network. The most influential members
in the social network may be determined using flow models, or by using page rank style methods which determine the most well connected entities in the social network.

Finally, an important class of techniques is that of **inferring links** which are not yet known in the social networks. **This problem is referred to as that of link inference**. The link prediction problem is useful for determining important future linkages in the underlying social network. Such future linkages provide an idea of the future relationships or missing relationships in the social network. Link prediction is also useful in a number of **adversarial applications** in which one does not fully know the linkages in an enemy or terrorist network, and uses automated data mining techniques in order to estimate the underlying links.

4.1.1 Statistical Properties of Social Networks

The questions of interest are:

What do social networks look like, on a large scale?
Do most nodes have few connections, with several “hubs” or is the distribution more stable? What sort of clustering behavior occurs?

How do networks behave over time?
Does the structure vary as the network grows? In what fashion do new entities enter a network? Does the network retain certain graph properties as it grows and evolves? Does the graph undergo a “phase transition", in which its behavior suddenly changes?

How do the non-giant weakly connected components behave over time?
One might argue that they grow, as new nodes are being added; and their size would probably remain a fixed fraction of the size of the GCC. Someone else might counter-argue that they shrink, and they eventually get absorbed into the GCC. What is happening, in real graphs?

What distributions and patterns do weighted graphs maintain?
How does the distribution of weights change over time– do we also observe a densification of weights as well as single-edges? How does the distribution of weights relate to the degree distribution? Is the addition of weight burst over time, or is it uniform?

Answering these questions is important to understand how natural graphs evolve, and to:

(a) spot anomalous graphs and sub-graphs;

(b) answer questions about entities in a network and what-if scenarios; and

(c) discard unrealistic graph generators.

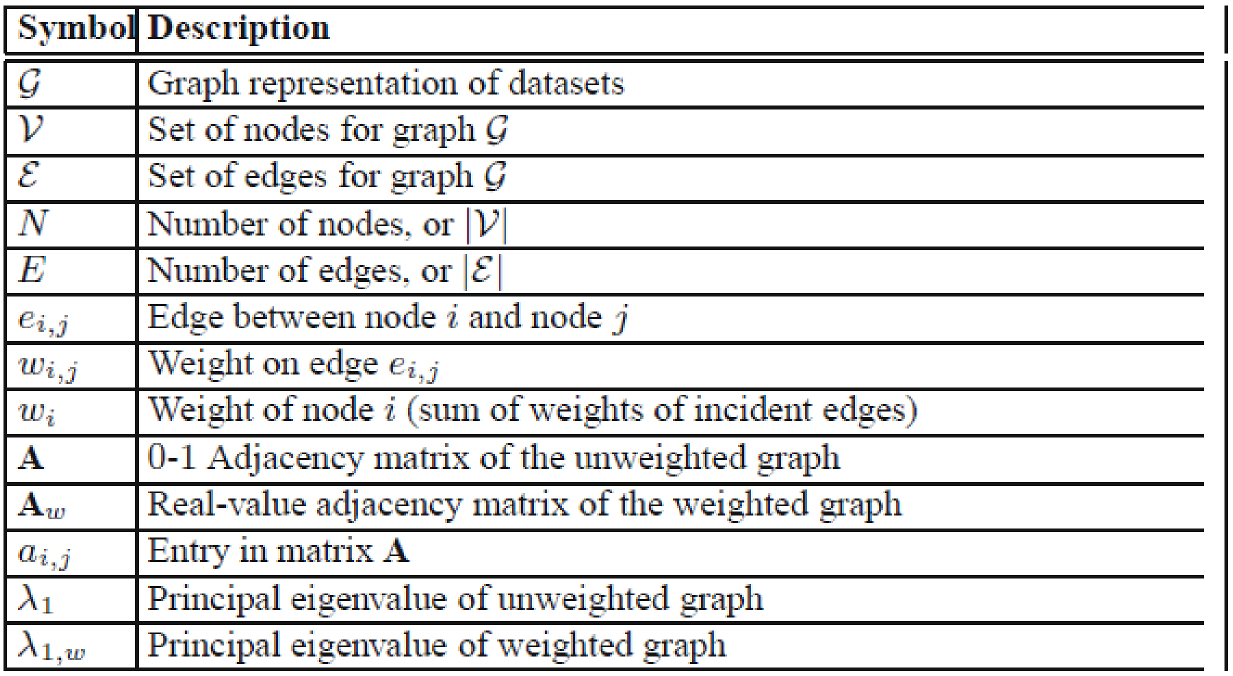
Let’s elaborate on each of the above applications:

Spotting anomalies is vital for determining abuse of social and computer networks, such as link-spamming in a web graph, fraudulent reputation building in e-auction systems, detection of dwindling/abnormal social sub-groups in a social-networking site like **Yahoo-360 (360.yahoo.com), Facebook (www.facebook.com)** and LinkedIn **(www.linkedin.com)**, and network intrusion detection.

Analyzing network properties are also useful for identifying authorities and search algorithms for discovering the "**network value**" of customers for using viral marketing or to improve recommendation systems. What-if scenarios are vital for extrapolation, provisioning and algorithm design:

For example, if we expect that the number of links will double within the next year, we should provision for the appropriate hardware to store and process the upcoming queries.

Table of Notations:



Symbol	Description
$\mathcal{G}$	Graph representation of datasets
$\mathcal{V}$	Set of nodes for graph $\mathcal{G}$
$\mathcal{E}$	Set of edges for graph $\mathcal{G}$
N	Number of nodes, or $\lvert\mathcal{V}\rvert$
E	Number of edges, or $\lvert\mathcal{E}\rvert$
$e_{i,j}$	Edge between node i and node j
$w_{i,j}$	Weight on edge $e_{i,j}$
w_i	Weight of node i (sum of weights of incident edges)
$\mathbf{A}$	0-1 Adjacency matrix of the unweighted graph
$\mathbf{A}_w$	Real-value adjacency matrix of the weighted graph
$a_{i,j}$	Entry in matrix $\mathbf{A}$
λ_1	Principal eigenvalue of unweighted graph
$\lambda_{1,w}$	Principal eigenvalue of weighted graph

4.2 Some Important Terms and Definitions

4.2.1 Graphs

We can represent a social network as a graph. A static, unweighted graph G consists of a set of **nodes V** and a **set of edges E: G = (V, E)**. We represent the sizes of *V* and *E* as *N* and *E*.

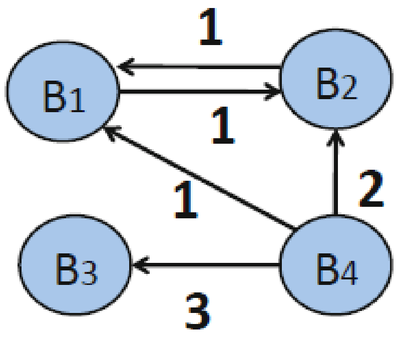
A graph may be directed or undirected– for instance, a phone call may be from one party to another, and will have a directed edge, or a mutual friendship may be represented as an undirected edge. Most properties we examine will be on undirected graphs.

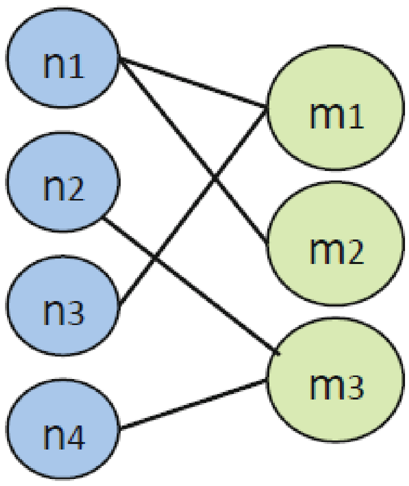
Graphs may also be weighted, where there may be multiple edges occurring between two nodes (e.g. repeated phone calls) or specific edge weights (e.g. monetary amounts for transactions). In a weighted graph G, let e_i, j be the edge between node i and node j.

We shall refer to these two nodes as the ʹneighboring nodes' or 'incident nodes' of edge e_i, j. Let w_i, j be the weight on edge e_i, j. The total weight *wi* of **node *i*** is defined as the sum of weights of all its incident edges, that is $w_i = \sum_{k=1}^{d_i} w_{i,k}$, where di denotes its degree. There is a relation between a given edge weight *wi, j* and the weights of its neighboring nodes *wi* and *wj.* Finally, graphs may be ***unipartite*** or ***multipartite***. Most social networks one thinks of are ***unipartite***– people in a group, papers in a citation network, etc. However, there may also be multipartite– that is, there are multiple classes of nodes and edges are only drawn between nodes of different classes. Bipartite graphs, like the movie-actor graph of IMDB, consist of disjoint sets of *nodes V1 and V2*, say, for authors and movies, with no edges among nodes of the same type.

We can represent a graph either visually, or with an adjacency matrix A, where nodes are in rows and columns, and numbers in the matrix indicate the existence of edges. For unweighted graphs, all entries are 0 or 1; for weighted graphs the adjacency matrix contains the values of the weights.

Below figure shows examples of graphs and their adjacency matrices. We next introduce other important concepts we use in analyzing these graphs.





	B_1	B_2	B_3	B_4
B_1	0	1	0	0
B_2	1	0	0	0
B_3	0	0	0	0
B_4	1	2	3	0

	n_1	n_2	n_3	n_4
m_1	1	0	1	0
m_2	1	0	0	0
m_3	0	1	0	1

4.2.2 Components

Another interesting property of a graph is its **component distribution**. We refer to a connected component in a graph **as a set of nodes and edges** where there exists a path between any two nodes in the set.

(For directed graphs, this would be a weakly connected component, where a strongly connected component requires a directed path between any given two nodes in a set.)

We find that in real graphs over time, a **giant connected component** (GCC) forms. However, it is also of interest to study the smaller components– when do they choose to join the GCC, and what size do they reach before doing so?

In our observations we will focus on the size of the second- and third-largest components. We will also look at the large scale distribution of all component sizes, and how that distribution changes over time. Not surprisingly, components of

***rank ≥ 2* form a power law**.

4.2.3 Diameter and Effective Diameter

We may want to answer the questions: How does the largest connected component of a real graph evolve over time? Do we start with one large CC, that keeps on growing? We propose to use the diameter-plot of the graph, that is, its diameter, over time, to answer these questions. For a given (static) graph, its diameter is defined as the maximum distance between any two nodes, where distance is the minimum number of hops (i.e., edges that must be traversed) on the path from one node to another, ignoring directionality. Calculating graph diameter is *O(N2)*.

Therefore, we choose to estimate the graph diameter by sampling nodes from the giant component. For *s = {1, 2, ..., S}*, we choose two nodes at random and calculate the distance (using breadth-first search). We then choose to record the 90 percentile value of distances, so we take the .9S largest recorded value.

The distance operation is *O(dk)*, where d is the graph diameter and k the maximum degree of any node– on average this is a much smaller cost.

Intuitively, the diameter represents how much of a *"small world"* the graph is– how quickly one can get from one "*end*" of the graph to another.

4.2.4 Heavy-tailed Distributions

While the Gaussian distribution is common in nature, there are many cases where the probability of events far to the right of the mean is significantly higher than in Gaussians.

In the Internet, for example, most routers have a very low degree (perhaps "home" routers), while a few routers have extremely high degree (perhaps the "core" routers of the Internet backbone) Heavy-tailed distributions attempt to model this. They are known as "**heavy-tailed**" because, while traditional exponential distributions have bounded variance *(large deviations from the mean become nearly impossible)*, **p(x)** decays polynomials quickly instead of exponentially as

$x \rightarrow \infty$, creating a "**fat tail**" for extreme values on the PDF plot.

One of the more well-known heavy-tailed distributions is the power law distribution. Two variables x and y are related by a power law when:

$$y(x) = Ax^{-\gamma}$$

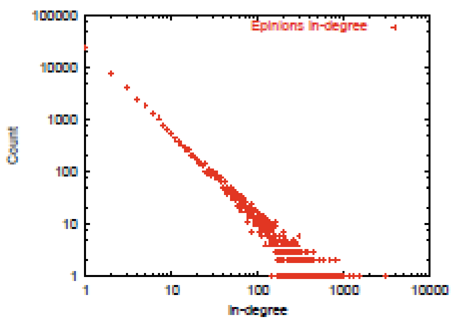
where **A and γ** are **positive constants**. The constant **γ** is often called the **power law exponent**.

A random variable is distributed according to a power law when the probability density function (pdf) is given by:

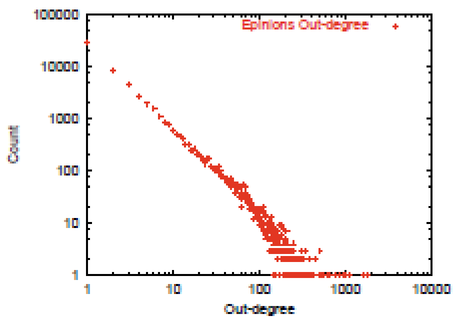
$$p(x) = Ax^{-\gamma}, \quad \gamma > 1, x \geq x_{min}$$

The **extra γ > 1 requirement ensures that p(x) can be normalized**. **Power laws with γ < 1 rarely occur in nature**.

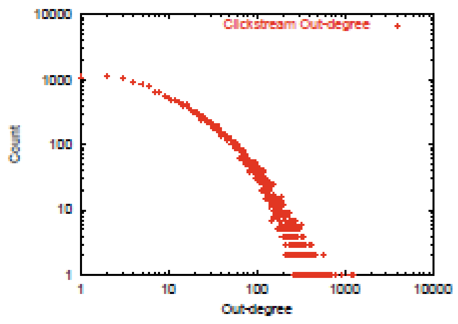
Skewed distributions, such as **power laws**, occur very often in real-world graphs, as shown in below graphs:



(a) Epinions In-degree



(b) Epinions Out-degree



(c) Clickstream Out-degree

While power laws appear in a large number of graphs, deviations from a pure power law are sometimes observed. Two of the more common deviations are exponential cutoffs and log normal.

Sometimes, the distribution looks like a *power law over the lower range of values along the x-axis*, but **decays very fast for higher values**. Often, **this decay is exponential, and this is usually called an exponential cutoff**:

$$y(x = k) \propto e^{-k/\kappa} k^{-\gamma}$$

where $e^{-k/\kappa}$ is the exponential cutoff term and $k^{-\gamma}$ is the power law term.

5

Implementation

5.1 R programming language

A software environment for statistical computing and graphics. For developing statistical software and data analysis the R language is broadly used among statisticians and data miners. R is an independent, open-source, and free implementation of the S programming language.

The source code for the R software environment is written primarily in C, FORTRAN, and R. It is a public-domain implementation of the widely regarded **S** statistical language, and the R/S platform is a de-facto standard among professional statisticians.

It is available for the Windows, Mac, and Linux operating systems. In addition to providing statistical operations, R is a general purpose programming language, so you can use it to automate analyses and create new functions that extend the existing language features. All of your basic data analysis: descriptive statistics, linear regressions, logic, probability, duration model, etc. Programming your own estimators and statistical models.

Tools and Packages used in R:

In this project I have used **RStudio GUI** and following **packages**:

- **twitteR:** The Twitter web API that provides an interface.
- **ROAuth**: It provides an interface to the OAuth 1.0 specification. This package allows users to connect the server of their choice and authenticate via OAuth package.
- **plyr**: It is a set of tools that solves a common set of problems: we need to break a big problem down into manageable pieces, operate on each pieces and then put all the pieces back together.
- **stringr:** It is a set of simple packages that make R's string functions more reliable, simpler and easier to use. It does this by confirming that: argument and names (and positions) are dependable, all functions deal with NA's and zero length character properly, and the

output data structures from each function matches the input data structures of other functions.

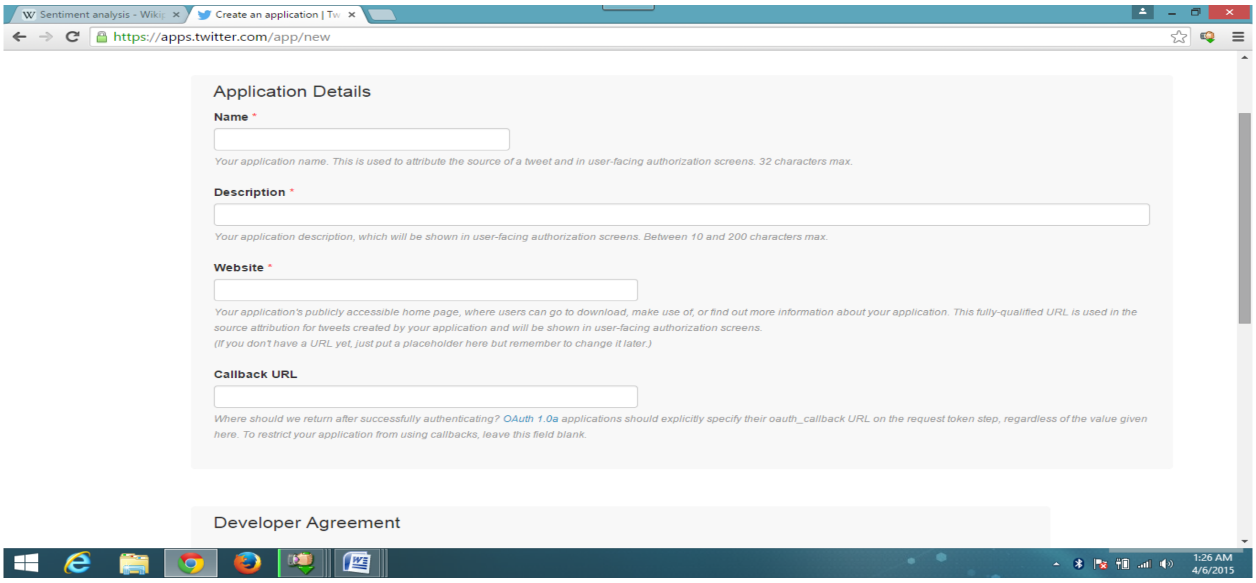
- **ggplot2:** It is an implementation of the grammar of graphics in R. We can build up a plot step by step from numerous data sources. It bind and gains both base and lattice graphics: conditioning and shared axes which are controlled automatically.
- **RColorBrewer**: This package offers palettes for sketching nice maps shaded according to a variable.
- **tm**: it is a framework for text mining applications inside R.
- **wordcloud***:* This package supports in creating pretty viewing word clouds in Text Mining.
- **sentiment***:* It is a R package with tools for sentiment analysis including Bayesian classifiers for positivity/negativity and emotion classification.

5.2 Creating a Twitter Application

First step to implement Twitter Analysis is to create a twitter application. This application will permit you to perform analysis by linking your R console to the twitter using the Twitter API. The steps for creating your twitter applications are:

By using this link ***https://dev.twitter.com*** and login by using your twitter account.

Then go to **My Applications** → **Create a new application**.



Sentiment analysis - Wiki
Create an application | Tw
https://apps.twitter.com/app/new

Application Details

Name *

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Developer Agreement

Steps to be followed:

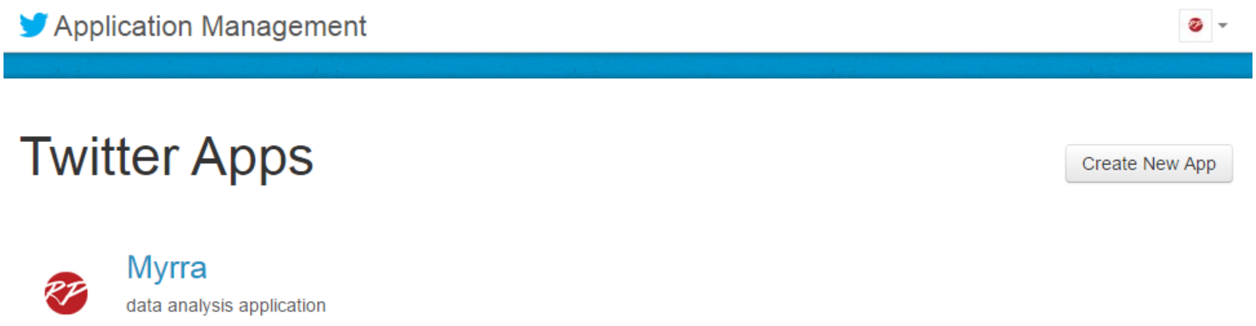
1. First give your application a name, description about your application in limited words not more than i.e. (10).
2. Name your website's URL or your blog address (in case you don't have any website).
3. No need to fill the Callback URL leave blank for now.
4. Complete other regulations and create your twitter application.

Once, all the steps are done, the created application will show as below. Please note the **Consumer key** and **Consumer Secret number** as they will be used in RStudio later.

Consumer Key Example: "**Llwxxxxxxxxxxxxxxxwl2n1oH**"

Consumer Secret Number Example: **"ShExxxxxxxxxxxxxxxxxxxxxxxxxxxxxx0ZEE"**

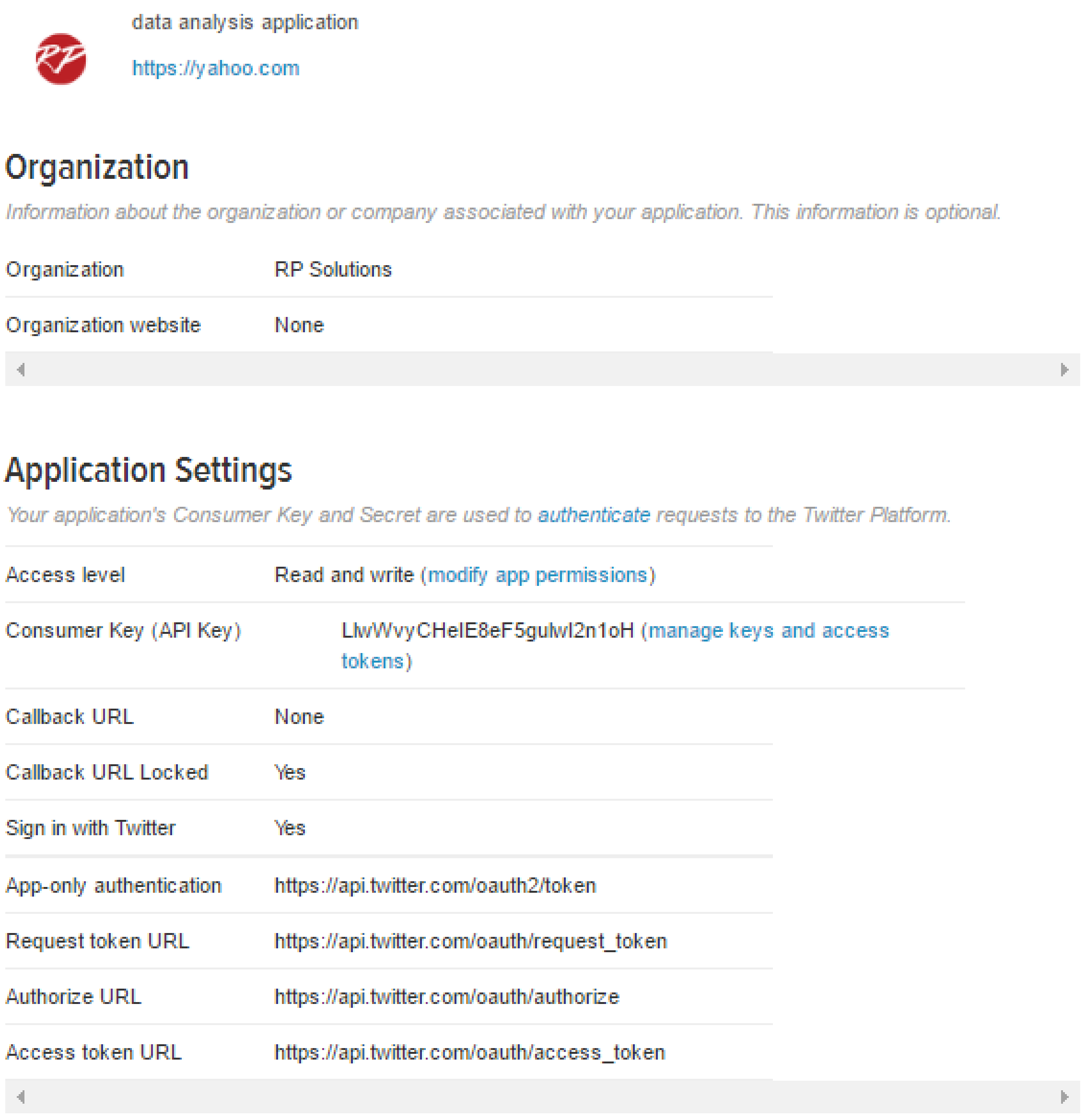
Like this your App will be created as shown below in pic.



Myrra

data analysis application

Now open the app and notice your app will have following token that are to be used later step.


data analysis application

https://yahoo.com

Organization

Information about the organization or company associated with your application. This information is optional.

Organization	RP Solutions
Organization website	None

Application Settings

Your application's Consumer Key and Secret are used to authenticate requests to the Twitter Platform.

Access level	Read and write (modify app permissions)
Consumer Key (API Key)	LlwWvyCHeIE8eF5gulwl2n1oH (manage keys and access tokens)
Callback URL	None
Callback URL Locked	Yes
Sign in with Twitter	Yes
App-only authentication	https://api.twitter.com/oauth2/token
Request token URL	https://api.twitter.com/oauth/request_token
Authorize URL	https://api.twitter.com/oauth/authorize
Access token URL	https://api.twitter.com/oauth/access_token

This step is done. Now let's go to RStudio for further process.

RStudio - Sentiment Analysis using Lexical based Approach:

Let’s start exploring the code base:

Step 1: Set up the working directory

R Code <...>

```
#
# Set the working directory
#
setwd("D:/DTU/EMBA Semester-2/Project/Project-Dir/Sentiment_Analysis")
```

This means that you set up the project’s folder in your computer as the working directory. This will help RStudio to get and set the dependencies without explicitly specifying the folder’s path all the time.

Step 2: Installing the packages

R Code <...>

```
#
# Installing the dependencies
#
install.packages('twitteR')
install.packages('RCurl')
install.packages('ROAuth')
install.packages('stringr')
install.packages("wordcloud")
install.packages("tm")
install.packages('plyr')
```

After executing the script section, console will look like this shown below:

OUTPUT:

```
> library(twitteR)
Warning message:
package 'twitteR' was built under R version 3.1.3
> library(RCurl)
Loading required package: bitops
Warning messages:
1: package 'RCurl' was built under R version 3.1.3
2: package 'bitops' was built under R version 3.1.3
> library(ROAuth)
Warning message:
package 'ROAuth' was built under R version 3.1.3
> library(stringr)
Warning message:
package 'stringr' was built under R version 3.1.3
> library(RJSONIO)
Warning message:
package 'RJSONIO' was built under R version 3.1.3
> library(plyr)

Attaching package: 'plyr'

The following object is masked from 'package:twitteR':

    id

Warning message:
package 'plyr' was built under R version 3.1.3
> library(bitops)
>
```

Step 3: Loading the positive and the negative word list

R Code <...>

```
#
# Load positive word list
# Location: \wordbanks\positive-words.txt
#
pos = scan('wordbanks/positive-words.txt', what='character', comment.char=';')

#
# load negative word list
# Location: \wordbanks\negative-words.txt
#
neg = scan('wordbanks/negative-words.txt', what='character', comment.char=';')
```

We are using scan () method to read the .txt files.

We are reading the positive-words.txt from the specified location and similarly reading the negative-words.txt.

Step 4: Creating word list

R Code <...>

```
# Create pos.words list
pos.words = c(pos)

# Create neg.words list
neg.words = c(neg)
```

In this step we are creating the word list using c () method.

Step 5: Setting up the Twitter URLs

R Code <...>

```
#
# TWITTER URLS
#
reqURL <- "https://api.twitter.com/oauth/request_token"
accessURL <- 'https://api.twitter.com/oauth/access_token'
authURL <- 'https://api.twitter.com/oauth/authorize'
```

In this step we are setting the required URLs to hit while doing authentication.

Step 6: Setting up the Twitter Authentication Keys

R Code <...>

```
#
# Authentication Keys
#
consumerKey <- "LlwWvyCHeIE8eF5gulwI2n1oH"
consumerSecret <- "ShEDczATYMFclRwAqnT4hbTCFydc8wKAHQU6wuyDHo9xKHOZEE"
```

Step 7: Setting up the Twitter Authentication Tokens

R Code <...>

```
#
# Authentication Tokens
#
access_token <- "827045622435950594-o8HJFkFjyeyjDl1jfLdzgwUHBhmH4nx"
access_token_secret <- "q8EpT3XTdgdVkEva77pPL3BPXQ426vnBmPx71CCL2KNbO"
```

Step 8: Creating twitCred instance for Authentication handshake

R Code <...>

```
#
# Creating twitCred instance  for Authentication handshake
#
twitCred <- OAuthFactory$new(consumerKey=consumerKey,
                             consumerSecret=consumerSecret,
                             requestURL=reqURL,
                             accessURL=accessURL,
                             authURL=authURL)
```

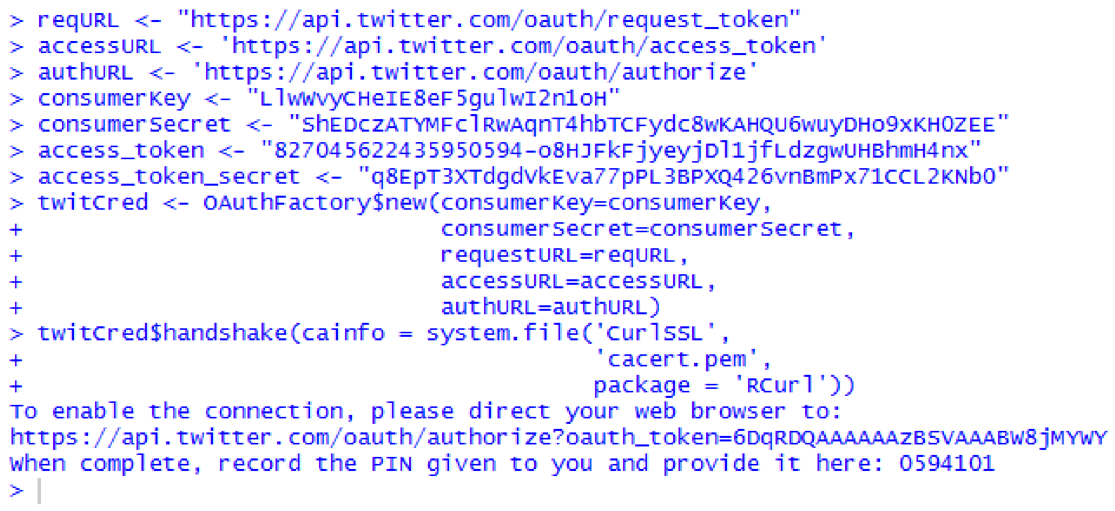
We are using OAuthFactory module to create a new instance named twitCred containing all the required information for authentication.

Step 9: Twitter Authentication handshaking

R Code <...>

```
#
# Command twitCred$handshake(cainfo="cacert.pem")
# will ask you to go a certain URL and
# entert he PIN you receive on this page
#
twitCred$handshake(cainfo = system.file('CurlSSL',
                                        'cacert.pem',
                                       package = 'RCurl'))
```

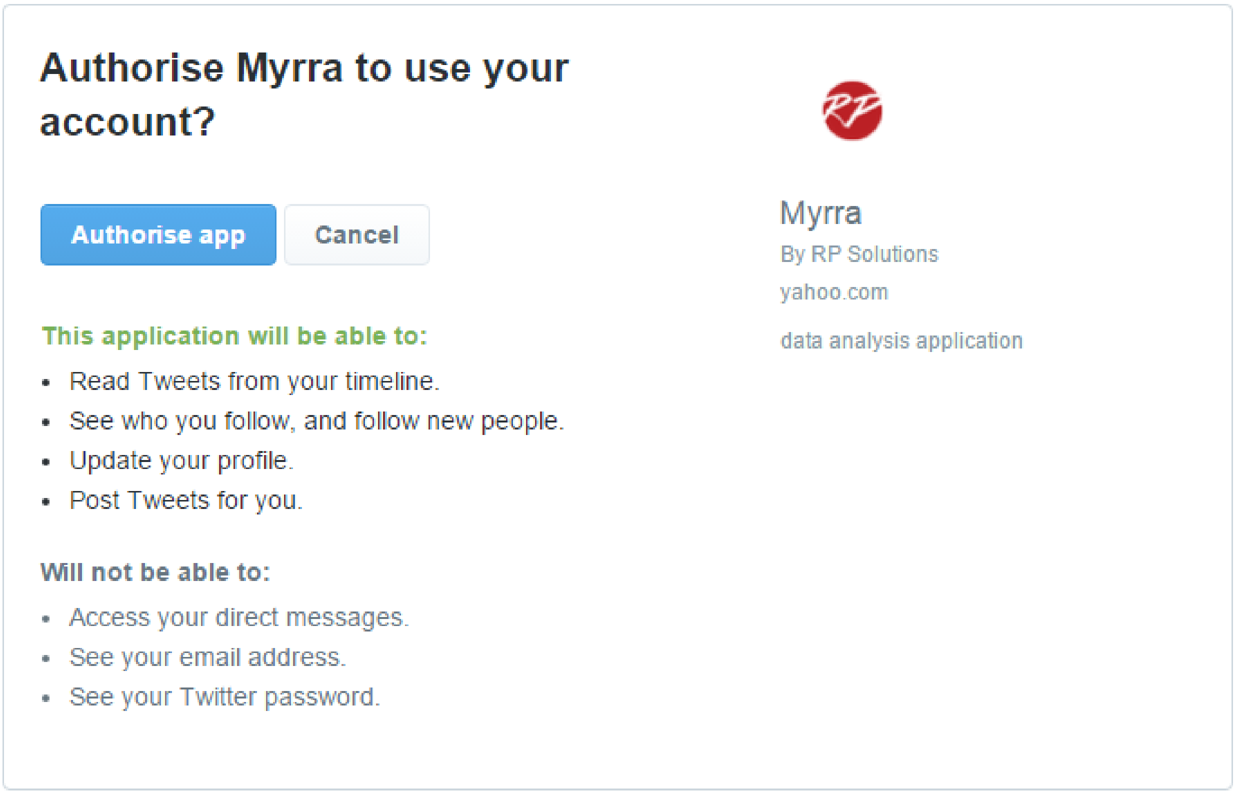
OUTPUT:



```
> reqURL <- "https://api.twitter.com/oauth/request_token"
> accessURL <- 'https://api.twitter.com/oauth/access_token'
> authURL <- 'https://api.twitter.com/oauth/authorize'
> consumerKey <- "LlwWvyCHeIE8eF5gulwI2n1oH"
> consumerSecret <- "ShEDczATYMFclRwAqnT4hbTCFydc8wKAHQU6wuyDHo9xKH0ZEE"
> access_token <- "827045622435950594-o8HJFkFjyeyjDl1jfLdzgwUHBhmH4nx"
> access_token_secret <- "q8EpT3XTdgdVkEva77pPL3BPXQ426vnBmPx71CCL2KNb0"
> twitCred <- OAuthFactory$new(consumerKey=consumerKey,
+                              consumerSecret=consumerSecret,
+                              requestURL=reqURL,
+                              accessURL=accessURL,
+                              authURL=authURL)
> twitCred$handshake(cainfo = system.file('CurlSSL',
+                                         'cacert.pem',
+                                         package = 'RCurl'))
To enable the connection, please direct your web browser to:
https://api.twitter.com/oauth/authorize?oauth_token=6DqRDQAAAAAAzBSVAAABW8jMYWY
When complete, record the PIN given to you and provide it here: 0594101
> 
```

Twitter Authentication Pages:

https://api.twitter.com/oauth/authorize?oauth_token=GDzWKAAAAAAAzBSVAAABW8jVE4c



Authorise Myrra to use your account?

This application will be able to:

- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.

Will not be able to:

- Access your direct messages.
- See your email address.
- See your Twitter password.

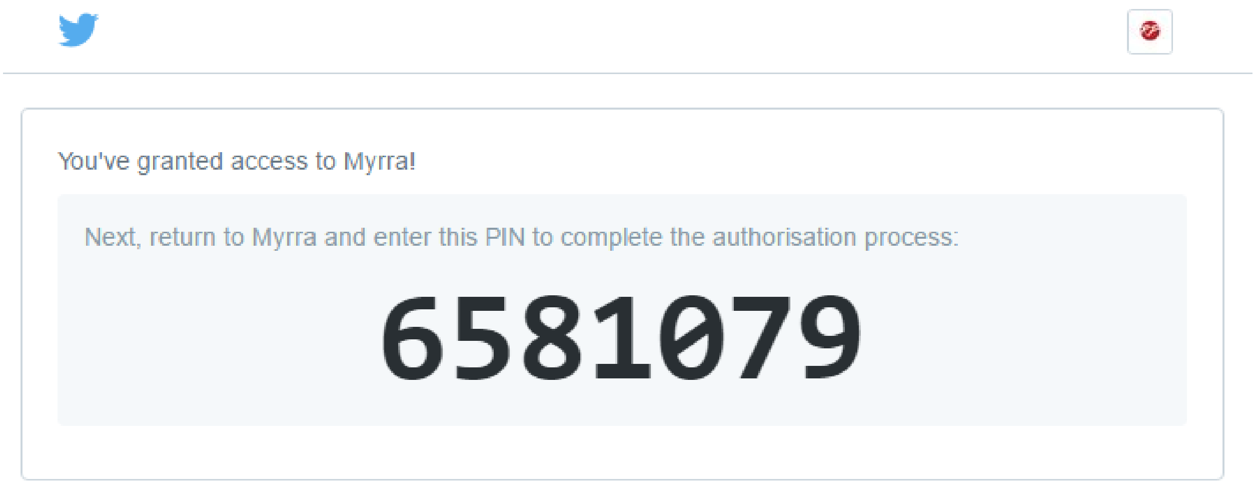
Myrra

By RP Solutions

yahoo.com

data analysis application

In the above page, click on Authorize App button in blue color. After clicking the button, below page will appear.



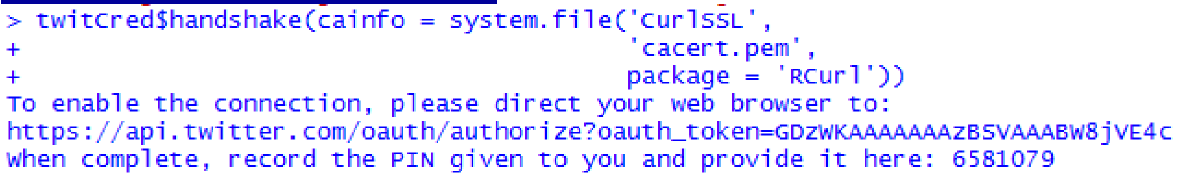
You've granted access to Myrra!

Next, return to Myrra and enter this PIN to complete the authorisation process:

6581079

We need to copy the number appearing in the page 2 and enter this number in RStudio for completing the authentication process.

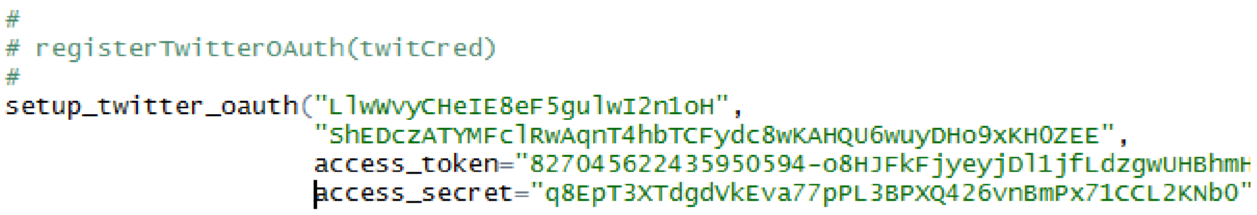
OUTPUT:



```
> twitCred$handshake(cainfo = system.file('curlSSL',
+                                          'cacert.pem',
+                                         package = 'RCurl'))
To enable the connection, please direct your web browser to:
https://api.twitter.com/oauth/authorize?oauth_token=GDzWKAAAAAAAzBSVAAABW8jVE4c
When complete, record the PIN given to you and provide it here: 6581079
```

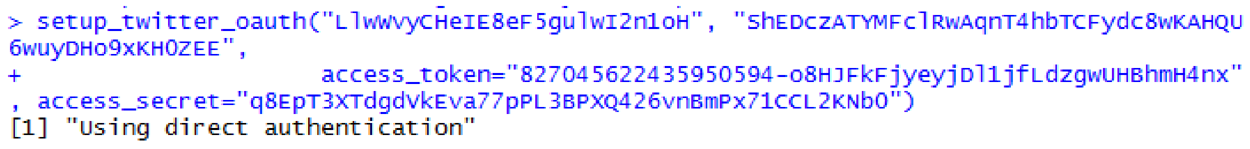
Step 10: Register Twitter Oath for communication

R Code <...>



```
#
# registerTwitterOAuth(twitCred)
#
setup_twitter_oauth("LlwWvyCHeIE8eF5gulwI2n1oH",
                    "ShEDczATYMFclRwAqnT4hbTCFydc8wKAHQU6wuyDHo9xKH0ZEE",
                    access_token="827045622435950594-o8HJFkFjyeyjDl1jfLdzgwUHBhmH
                    access_secret="q8EpT3XTdgdVkEva77pPL3BPXQ426vnBmPx71CCL2KNb0"
```

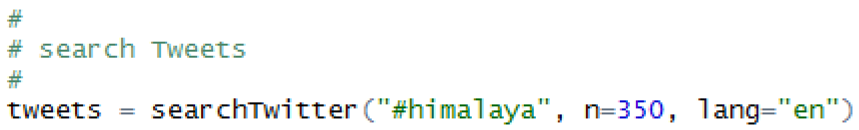
OUTPUT:



```
> setup_twitter_oauth("LlwWvyCHeIE8eF5gulwI2n1oH", "ShEDczATYMFclRwAqnT4hbTCFydc8wKAHQU
6wuyDHo9xKH0ZEE",
+                     access_token="827045622435950594-o8HJFkFjyeyjDl1jfLdzgwUHBhmH4nx"
, access_secret="q8EpT3XTdgdVkEva77pPL3BPXQ426vnBmPx71CCL2KNb0")
[1] "Using direct authentication"
```

Step 11: Search Tweets

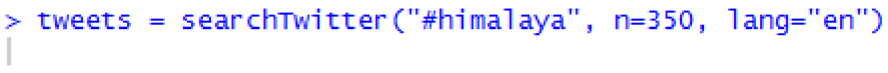
R Code <...>



```
#
# search Tweets
#
tweets = searchTwitter("#himalaya", n=350, lang="en")
```

In this case we are searching for **#himalaya**, language is **English** and **expecting 350 tweets**.

OUTPUT:



```
> tweets = searchTwitter("#himalaya", n=350, lang="en")
```

```
[[18]]
[1] "mollyrowanleach: RT @AlonzoLyons: #Adventure is calling from the #Himalaya--#Nepal
's Top 6 Peaks. Can you summit one, too? \n#IslandPeak #ImjaTse #trek\nhttps…"

[[19]]
[1] "nuralamsyah_bdg: RT @OraliaSotoRoman: The Monastery of Thiksey #Himalaya https://t
.co/OOatLPNM6M"

[[20]]
[1] "Sulzbach_City: https://t.co/Nv96Hj7iaa #Reishunger #Vollkorn #Basmati #Reis  #Bio
#Vollkorn Traditional #Basmati  #Indien… https://t.co/dYjNe1IARi"

[[21]]
[1] "RT_Himachal: RT @zonebxforbooks: A cool walk in the mall road of Simla with a book
 gives you pleasant memories #travel\n#Himalaya \n#Himalayas\n#retweet\nht…"

[[22]]
[1] "zonebxforbooks: A cool walk in the mall road of Simla with a book gives you pleasa
nt memories #travel\n#Himalaya \n#Himalayas\n#retweet\nhttps://t.co/x0A87cyx06"


...


[[344]]
[1] "southern_slut_: RT @AudreyAlpine: A beautiful sunset through the #clouds \xed��\x
ed�\u0084\xed��\xed�\u0083 on the #himalaya \xed��\xed�\u0089\xed��\xed�\u008b ht
tps://t.co/TGNfkdXE6k"

[[345]]
[1] "maniamannalai: RT @AudreyAlpine: A beautiful sunset through the #clouds \xed��\xe
d�\u0084\xed��\xed�\u0083 on the #himalaya \xed��\xed�\u0089\xed��\xed�\u008b htt
ps://t.co/TGNfkdXE6k"

[[346]]
[1] "binaloirinha: RT @AudreyAlpine: A beautiful sunset through the #clouds \xed��\xed
�\u0084\xed��\xed�\u0083 on the #himalaya \xed��\xed�\u0089\xed��\xed�\u008b http
s://t.co/TGNfkdXE6k"

[[347]]
[1] "juanrodriguezss: RT @AudreyAlpine: A beautiful sunset through the #clouds \xed��\
xed�\u0084\xed��\xed�\u0083 on the #himalaya \xed��\xed�\u0089\xed��\xed�\u008b h
ttps://t.co/TGNfkdXE6k"

[[348]]
[1] "Primev: RT @AudreyAlpine: A beautiful sunset through the #clouds \xed��\xed�\u00
84\xed��\xed�\u0083 on the #himalaya \xed��\xed�\u0089\xed��\xed�\u008b https://t.
co/TGNfkdXE6k"

[[349]]
[1] "rgjones16: RT @AudreyAlpine: A beautiful sunset through the #clouds \xed��\xed�\
u0084\xed��\xed�\u0083 on the #himalaya \xed��\xed�\u0089\xed��\xed�\u008b https:/
/t.co/TGNfkdXE6k"

[[350]]
[1] "leeway2k: RT @AudreyAlpine: A beautiful sunset through the #clouds \xed��\xed�\u
0084\xed��\xed�\u0083 on the #himalaya \xed��\xed�\u0089\xed��\xed�\u008b https://
t.co/TGNfkdXE6k"
```

Step 12: Convert the received tweets into vectors

R Code <...>

```r
#
# Covert to vector
#
tweet_text <- sapply(tweets, function(x) x$getText())
```

What is a vector?

A vector is a sequence of data elements of the same basic type. Members in a vector are officially called components. Nevertheless, we will just call them members in this site. Here is a vector containing three numeric values 2, 3 and 5. > c (2, 3, 5)

sapply () is used for traversing over the data in a vector and calling specified function for each item.

Step: 13: Convert the vector to corpus

R Code <...>

```r
#
# Corpus
#
tweet_corpus <- Corpus(VectorSource(tweet_text))
```

For checking the structure of the tweet_corpus, use:

R Code <...>

```r
str(tweet_corpus)
```

OUTPUT:

```
  .. ..- attr(*, "class")= chr "TextDocumentMeta"
  ..- attr(*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"
 $ 17 :List of 2
  ..$ content: chr "North Sikkim makes up for its small size with its vertical scale. C
louds part and big snow giants appear. Nov 2014.… https://t."| __truncated__
  ..$ meta   :List of 7
  .. ..$ author       : chr(0) 
  .. ..$ datetimestamp: POSIXlt[1:1], format: "2017-05-02 11:25:55"
  .. ..$ description  : chr(0) 
  .. ..$ heading      : chr(0) 
  .. ..$ id           : chr "17"
  .. ..$ language     : chr "en"
  .. ..$ origin       : chr(0) 
  .. ..- attr(*, "class")= chr "TextDocumentMeta"
  ..- attr(*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"
 $ 18 :List of 2
  ..$ content: chr "RT @Mercurytravels: Visit enchanting Kashmir with #MercuryTravels.
Experience the heritage of #India &amp; grandeur of #Himalay"| __truncated__
  ..$ meta   :List of 7
  .. ..$ author       : chr(0) 
  .. ..$ datetimestamp: POSIXlt[1:1], format: "2017-05-02 11:25:55"
  .. ..$ description  : chr(0) 
  .. ..$ heading      : chr(0) 
  .. ..$ id           : chr "18"
  .. ..$ language     : chr "en"
  .. ..$ origin       : chr(0) 
```

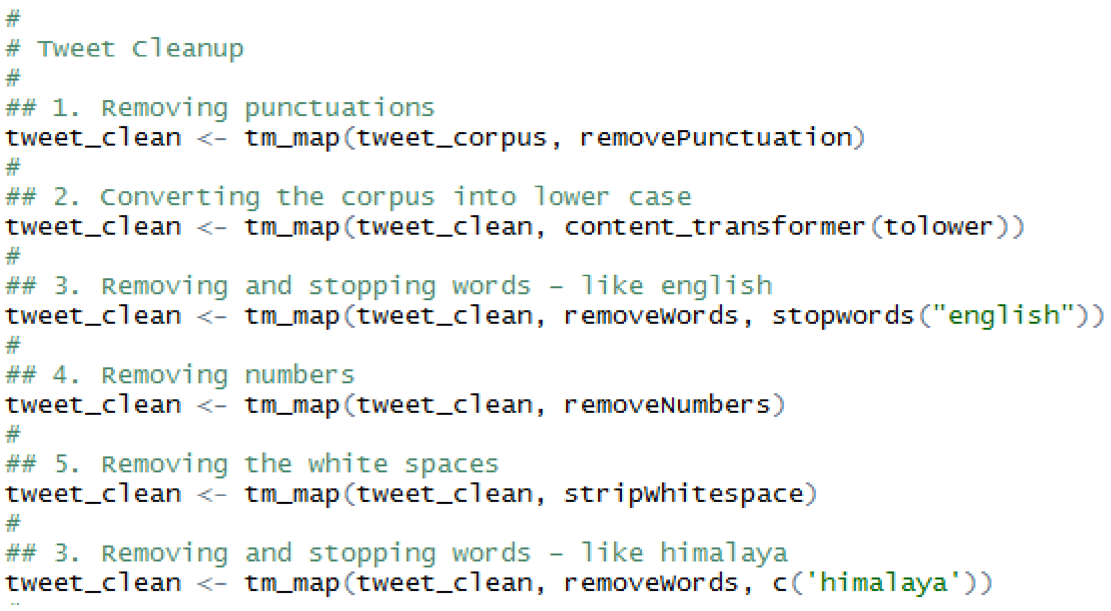
…

What is Corpus?

The main structure for managing documents in **tm** (text mining package) is a so-called Corpus, representing a collection of text documents. A corpus is an abstract concept, and there can exist several implementations in parallel.

Step 14: Cleaning up the tweets corpus

R Code <...>

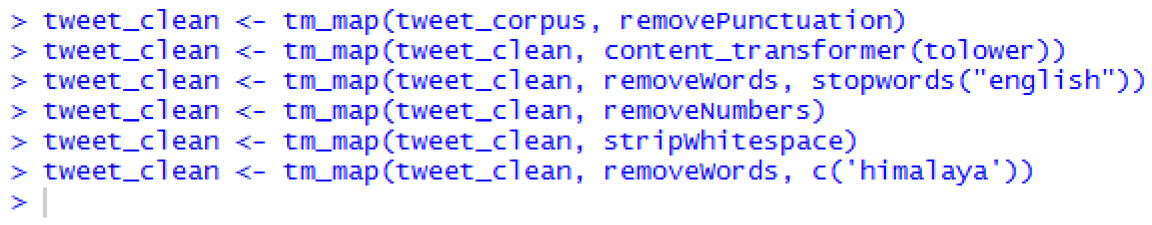


```
#
# Tweet Cleanup
#
## 1. Removing punctuations
tweet_clean <- tm_map(tweet_corpus, removePunctuation)
#
## 2. Converting the corpus into lower case
tweet_clean <- tm_map(tweet_clean, content_transformer(tolower))
#
## 3. Removing and stopping words - like english
tweet_clean <- tm_map(tweet_clean, removeWords, stopwords("english"))
#
## 4. Removing numbers
tweet_clean <- tm_map(tweet_clean, removeNumbers)
#
## 5. Removing the white spaces
tweet_clean <- tm_map(tweet_clean, stripWhitespace)
#
## 3. Removing and stopping words - like himalaya
tweet_clean <- tm_map(tweet_clean, removeWords, c('himalaya'))
```

Cleaning up the tweets corpus includes:

1. Removing punctuations
2. Converting the corpus into lower case
3. Removing and stopping words – like English and Himalaya
4. Removing numbers
5. Removing the white spaces

OUTPUT:



```
> tweet_clean <- tm_map(tweet_corpus, removePunctuation)
> tweet_clean <- tm_map(tweet_clean, content_transformer(tolower))
> tweet_clean <- tm_map(tweet_clean, removeWords, stopwords("english"))
> tweet_clean <- tm_map(tweet_clean, removeNumbers)
> tweet_clean <- tm_map(tweet_clean, stripWhitespace)
> tweet_clean <- tm_map(tweet_clean, removeWords, c('himalaya'))
> |
```

Step 14: Sentiment Score Algorithm

R Code <...>

```
#++++++++++++++++++++++++++++++
#
#   SENTIMENT SCORE function
#
#++++++++++++++++++++++++++++++
#
# Funtion Name: score.sentiment()
#
# Parameters
#   sentences: vector of text to score
#   pos.words: vector of words of positive sentiment
#   neg.words: vector of words of negative sentiment
#
score.sentiment = function(sentences, pos.words, neg.words)
{
  require(plyr);
  require(stringr);

  #
  # We got a vector of sentences. plyr will handle a list or
  # a vector as an "l" for us
  # We want a simple array of scores back, so we use "l" + "a" + "ply" = laply:
  # Objective: Create a simple array of scores with laply
  #
  scores = laply(sentences, function(sentence, pos.words, neg.words) {


    # clean up sentences with R's regex-driven global substitute, gsub():
    sentence = gsub('[[:punct:]]', '', sentence)
    sentence = gsub('[[:cntrl:]]', '', sentence)
    sentence = gsub('\\d+', '', sentence)

    # and convert to lower case:
    sentence = tolower(sentence)


    # split into words. str_split is in the stringr package
    word.list = str_split(sentence, '\\s+')

    # sometimes a list() is one level of hierarchy too much
    words = unlist(word.list)

    # compare our words to the dictionaries of positive & negative terms
    neg.matches = match(words, neg.words)
    pos.matches = match(words, pos.words)

    # match() returns the position of the matched term or NA
    # we just want a TRUE/FALSE:
    pos.matches = !is.na(pos.matches)
    neg.matches = !is.na(neg.matches)

    # and conveniently enough, TRUE/FALSE will be treated as 1/0 by sum():
    score = sum(pos.matches) - sum(neg.matches)

    return(score)
  }, pos.words, neg.words )

  scores.df = data.frame(score=scores, text=sentences)
  return(scores.df)
}
```

OUTPUT:

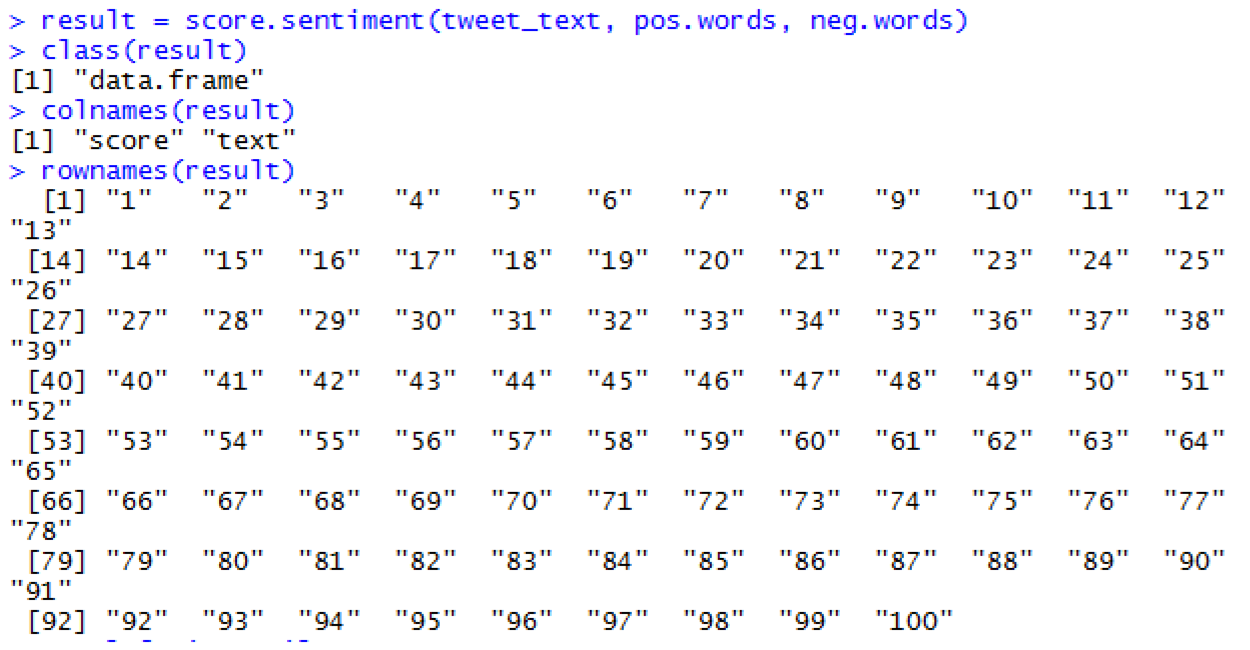
```
  scores = laply(sentences, function(sentence, pos.words, neg.words) {


    # clean up sentences with R's regex-driven global substitute, gsub():
    sentence = gsub('[[:punct:]]', '', sentence)
    sentence = gsub('[[:cntrl:]]', '', sentence)
    sentence = gsub('\\d+', '', sentence)

    # and convert to lower case:
    sentence = tolower(sentence)

    # split into words. str_split is in the stringr package
    word.list = str_split(sentence, '\\s+')

    # sometimes a list() is one level of hierarchy too much
    words = unlist(word.list)

    # compare our words to the dictionaries of positive & negative terms
    neg.matches = match(words, neg.words)
    pos.matches = match(words, pos.words)

    # match() returns the position of the matched term or NA
    # we just want a TRUE/FALSE:
    pos.matches = !is.na(pos.matches)
    neg.matches = !is.na(neg.matches)

    # and conveniently enough, TRUE/FALSE will be treated as 1/0 by sum():
    score = sum(pos.matches) - sum(neg.matches)

    return(score)
  }, pos.words, neg.words )

  scores.df = data.frame(score=scores, text=sentences)
  return(scores.df)
}
```

Step 15: Calculating Sentiment Score

R Code <...>

```
#
# Use score.sentiment() method to calculate the score of the reviews
#
# Calculating sentiment for the given reviews
#
result = score.sentiment(tweet_clean, pos.words, neg.words)
```

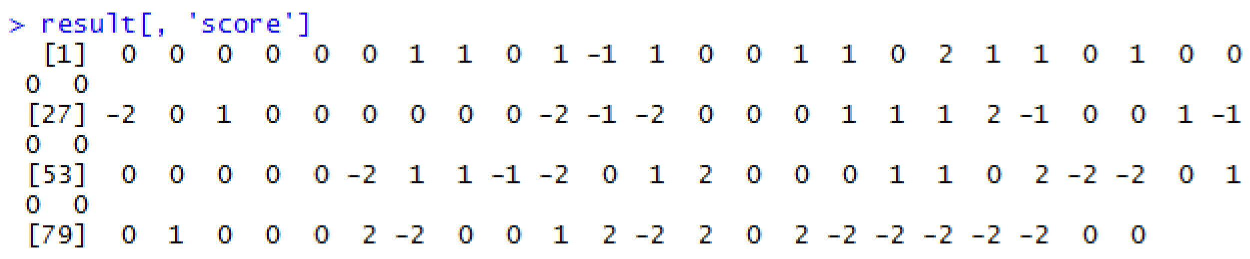
OUTPUT:



```
> result = score.sentiment(tweet_text, pos.words, neg.words)
> class(result)
[1] "data.frame"
> colnames(result)
[1] "score" "text"
> rownames(result)
  [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12"
"13"
 [14] "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"  "23"  "24"  "25"
"26"
 [27] "27"  "28"  "29"  "30"  "31"  "32"  "33"  "34"  "35"  "36"  "37"  "38"
"39"
 [40] "40"  "41"  "42"  "43"  "44"  "45"  "46"  "47"  "48"  "49"  "50"  "51"
"52"
 [53] "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"  "61"  "62"  "63"  "64"
"65"
 [66] "66"  "67"  "68"  "69"  "70"  "71"  "72"  "73"  "74"  "75"  "76"  "77"
"78"
 [79] "79"  "80"  "81"  "82"  "83"  "84"  "85"  "86"  "87"  "88"  "89"  "90"
"91"
 [92] "92"  "93"  "94"  "95"  "96"  "97"  "98"  "99"  "100"
```

Caluclated Sentiment Score

View 1:



```
> result[, 'score']
  [1]  0  0  0  0  0  0  1  1  0  1 -1  1  0  0  1  1  0  2  1  1  0  1  0  0
 0  0
 [27] -2  0  1  0  0  0  0  0  0 -2 -1 -2  0  0  0  1  1  1  2 -1  0  0  1 -1
 0  0
 [53]  0  0  0  0  0 -2  1  1 -1 -2  0  1  2  0  0  0  1  1  0  2 -2 -2  0  1
 0  0
 [79]  0  1  0  0  0  2 -2  0  0  1  2 -2  2  0  2 -2 -2 -2 -2 -2  0  0
```

View 2:

```
> result$score
  [1]  0  0  0  0  0  0  1  1  0  1 -1  1  0  0  1  1  0  2  1  1  0  1  0  0
 0  0
 [27] -2  0  1  0  0  0  0  0  0 -2 -1 -2  0  0  0  1  1  1  2 -1  0  0  1 -1
 0  0
 [53]  0  0  0  0  0 -2  1  1 -1 -2  0  1  2  0  0  0  1  1  0  2 -2 -2  0  1
 0  0
 [79]  0  1  0  0  0  2 -2  0  0  1  2 -2  2  0  2 -2 -2 -2 -2 -2  0  0
```

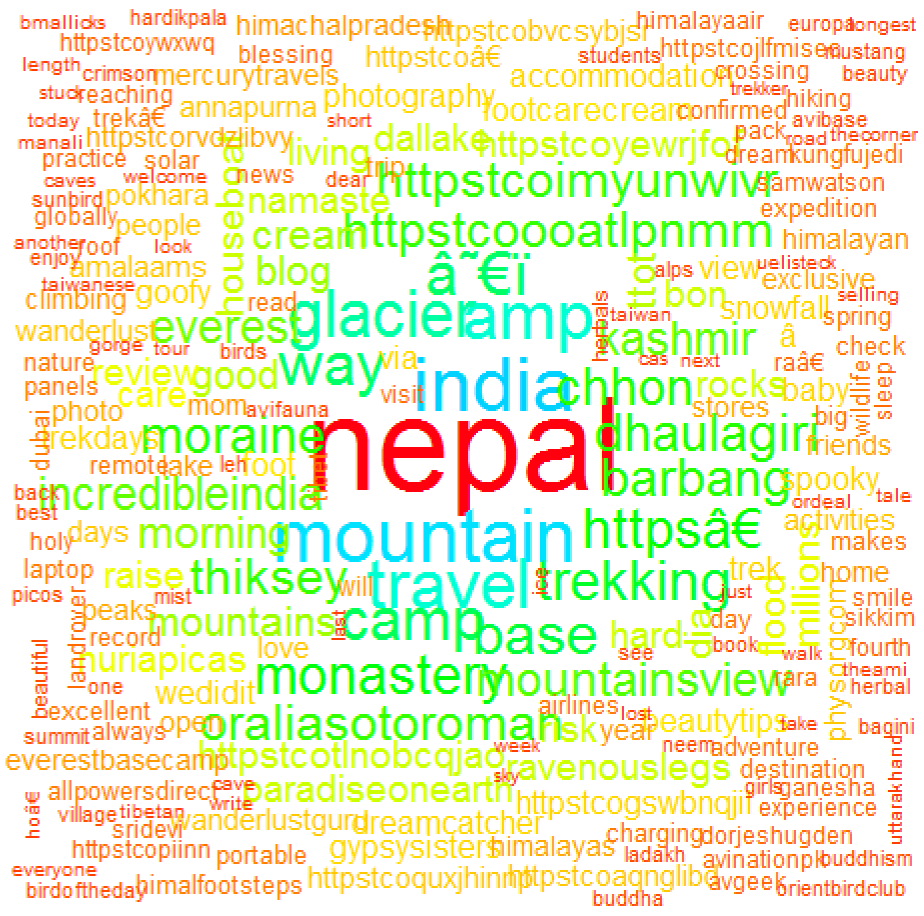
View 3:

```
> result[,1]
  [1]  0  0  0  0  0  0  1  1  0  1 -1  1  0  0  1  1  0  2  1  1  0  1  0  0
 0  0
 [27] -2  0  1  0  0  0  0  0  0 -2 -1 -2  0  0  0  1  1  1  2 -1  0  0  1 -1
 0  0
 [53]  0  0  0  0  0 -2  1  1 -1 -2  0  1  2  0  0  0  1  1  0  2 -2 -2  0  1
 0  0
 [79]  0  1  0  0  0  2 -2  0  0  1  2 -2  2  0  2 -2 -2 -2 -2 -2  0  0
```

As you can observe same result can be viewed using different command in R.

Visualization of the tweets

Word Cloud for #himalaya:



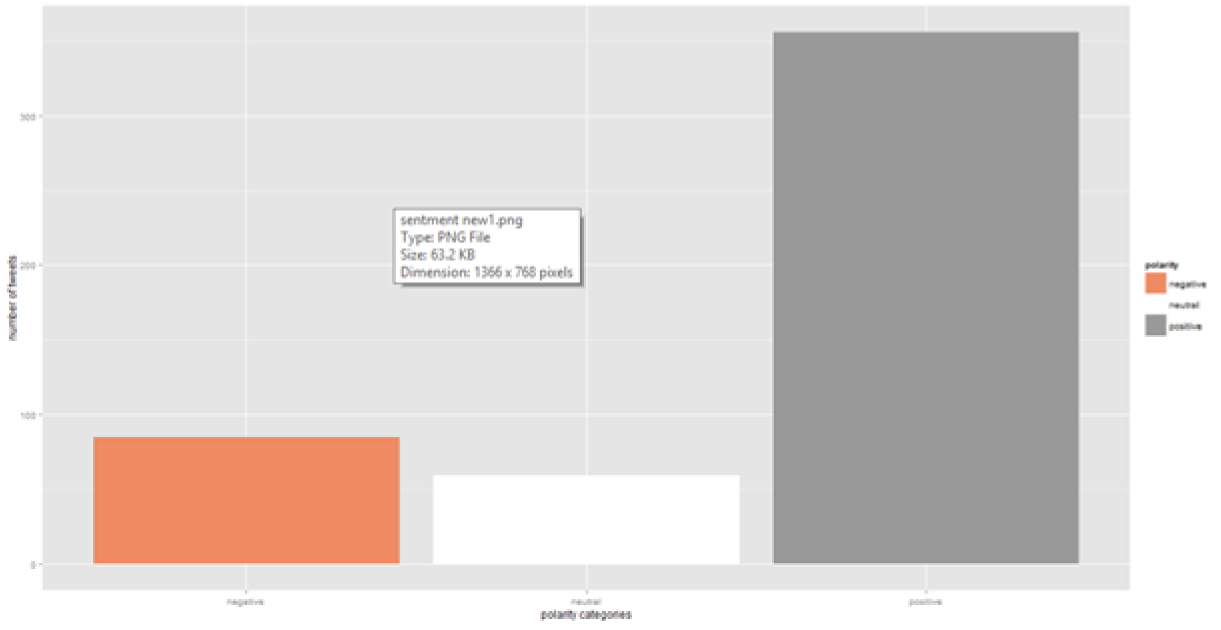
Step 15: Calculating Polarity

R Code <…>

```
# classify polarity
class_pol = classify_polarity(some_txt, algorithm="bayes")
# get polarity best fit
polarity = class_pol[,4]

# data frame with results
sent_df = data.frame(text=some_txt, emotion=emotion,
polarity=polarity, stringsAsFactors=FALSE)
```

OUTPUT:

Polarity:



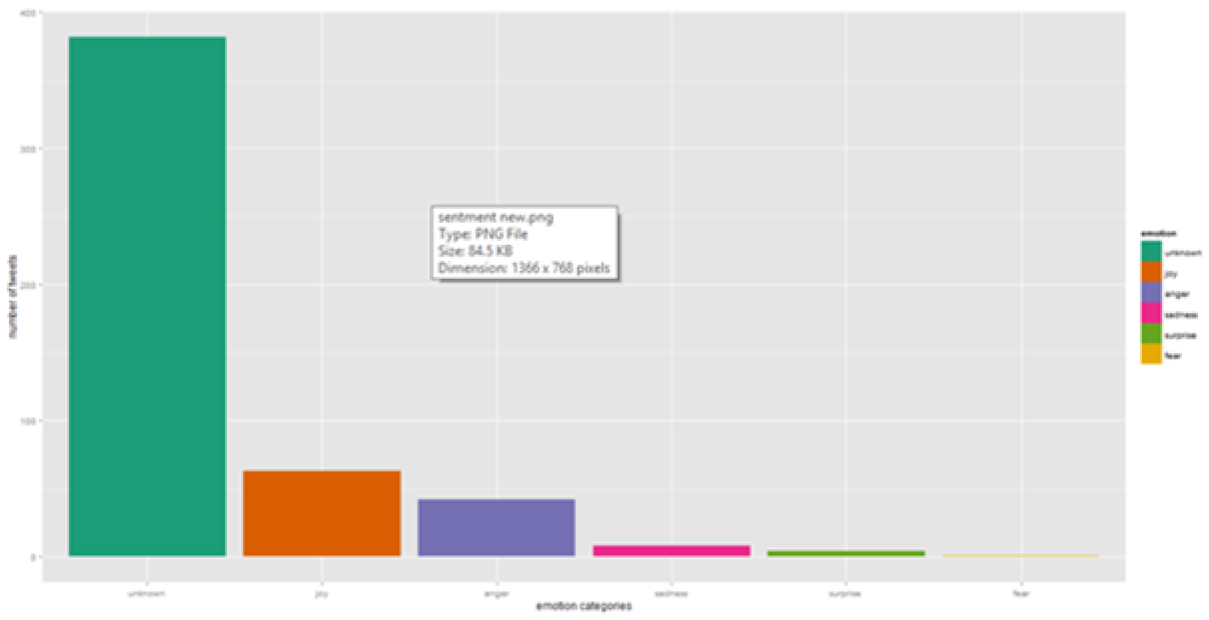
Step 15: Calculating Emotions

R Code <...>

```
# classify emotion
class_emo = classify_emotion(some_txt, algorithm="bayes", prior=1.0)
# get emotion best fit
emotion = class_emo[,7]
# substitute NA's by "unknown"
emotion[is.na(emotion)] = "unknown"
```

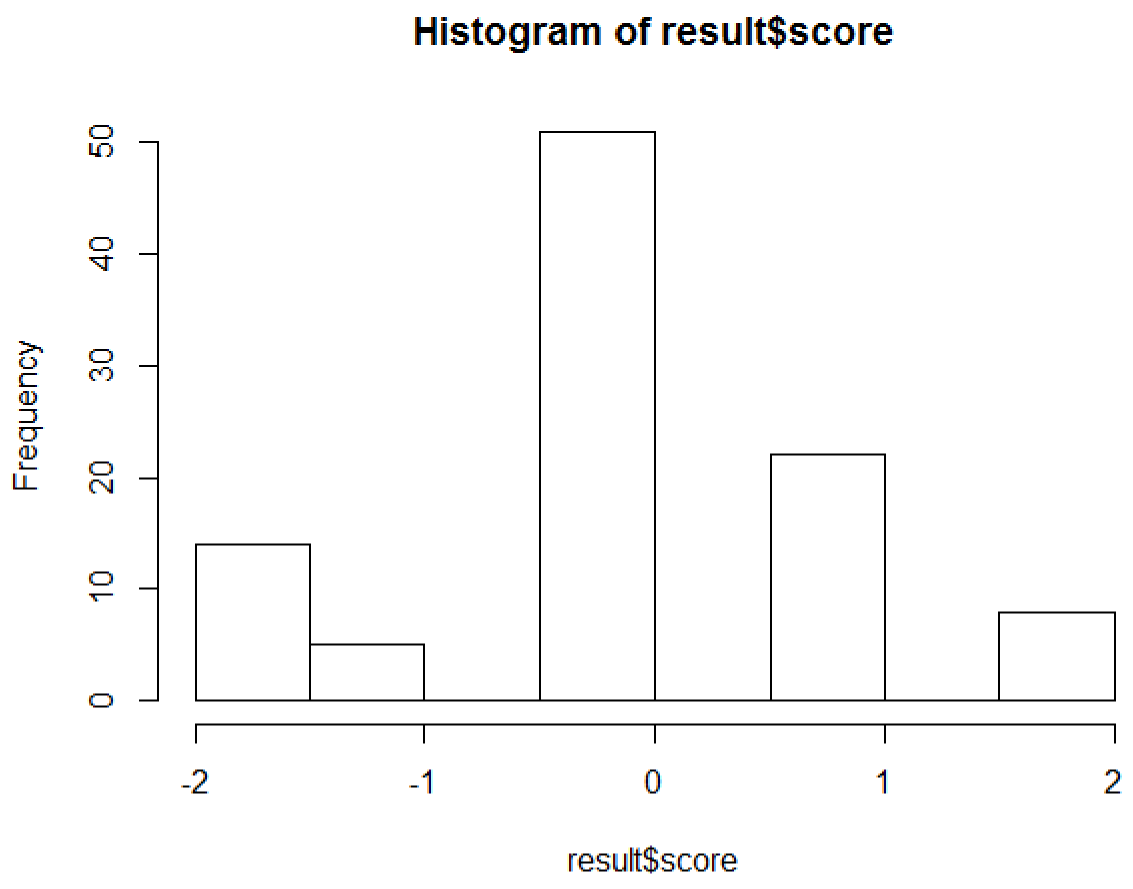
OUTPUT:

Emotions:



Histogram for the sentiment result

For **#himalaya**



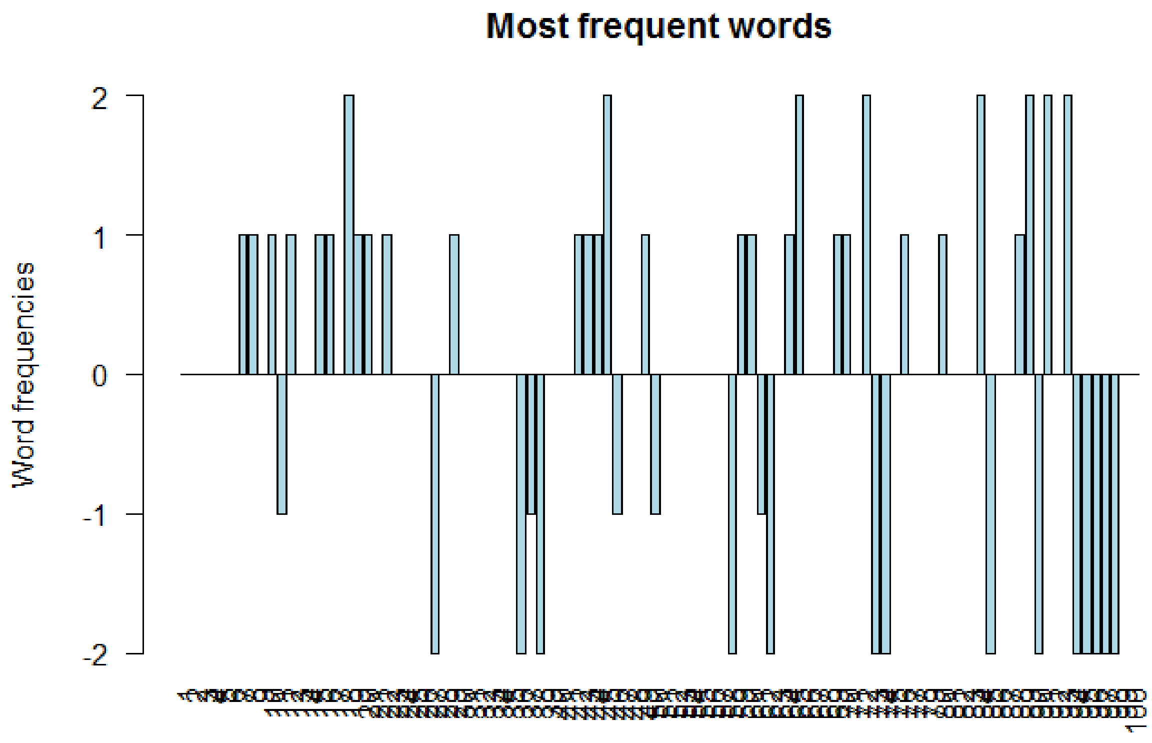
Analysis:

The sentiment score varies from -2 to +2.

X-axis has sentiment result; Y-axis has frequency of words.

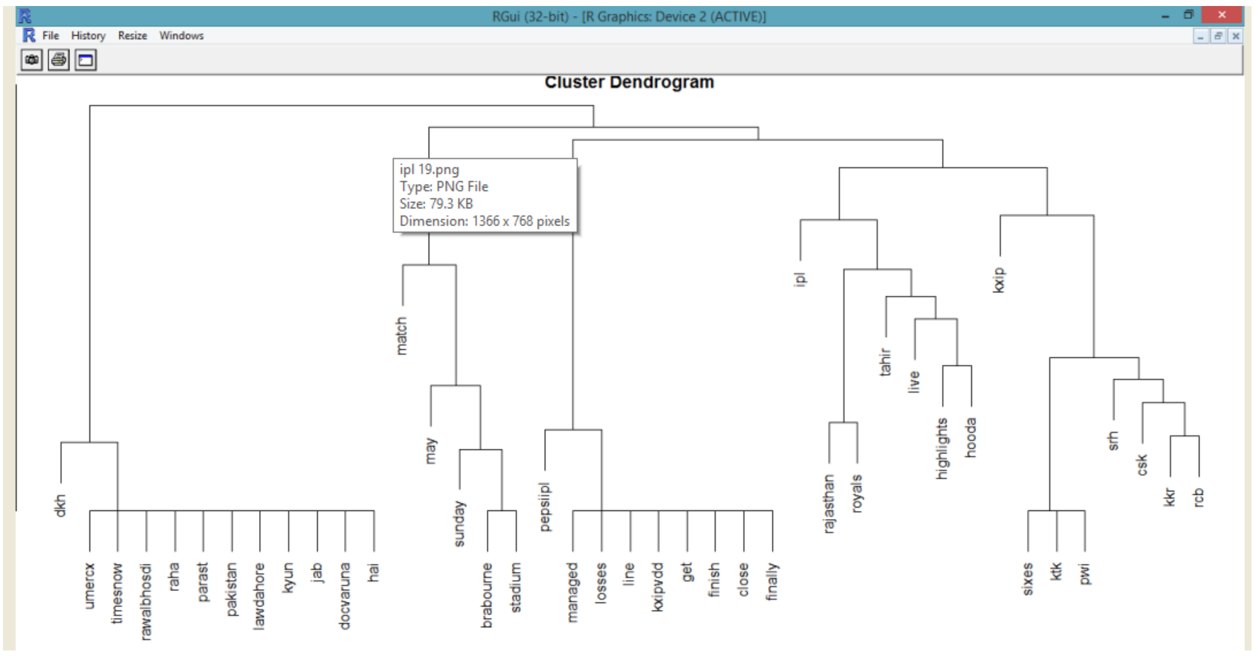
Most of the sentiment is **neutral** i.e. around score 0.

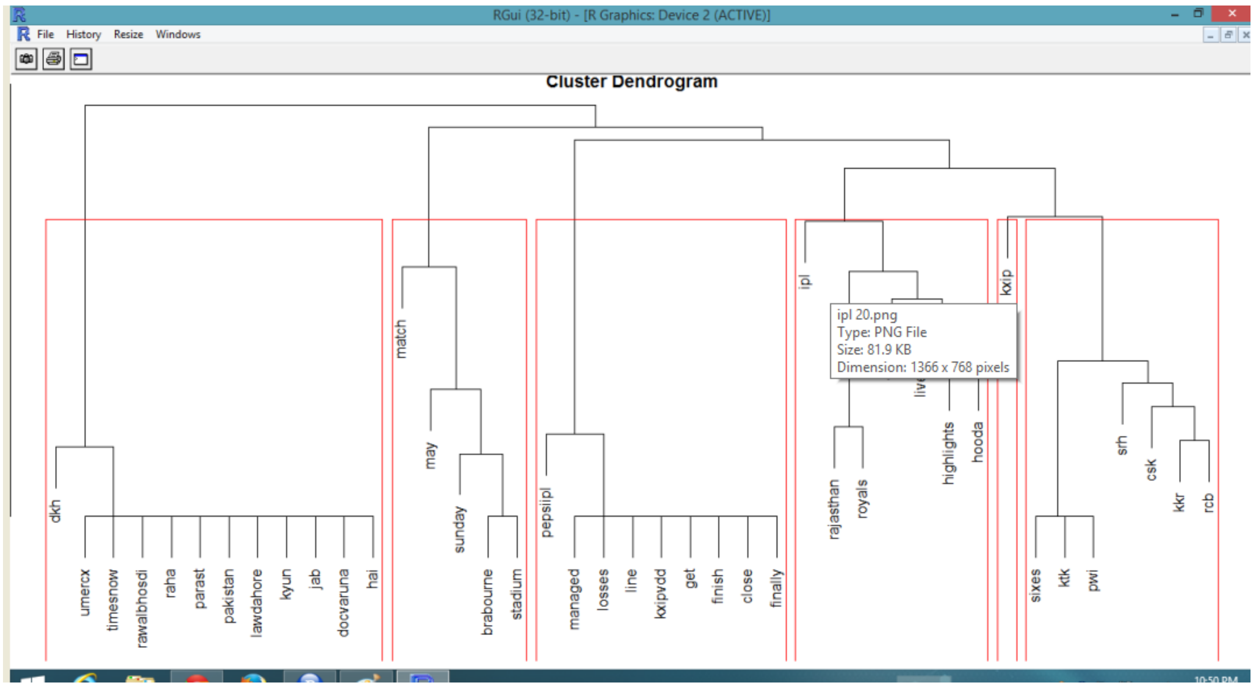
For #himalaya


Most frequent words
Word frequencies
2
1
0
-1
-2

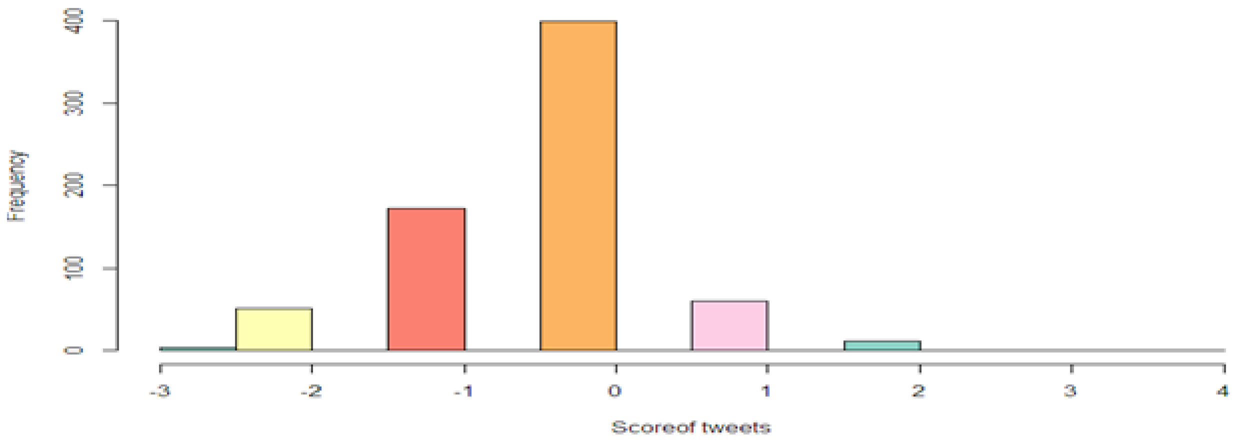
For **#ipl**

Cluster diagram:





Histogram for #kejeriwal:


Frequency
0
100
200
300
400
-3
-2
-1
0
1
2
3
4
Scoreof tweets

4. Natural Language Processing (NLP)

This section provides a brief history of NLP, introduces some of the main problems involved in extracting meaning from human languages and examines the kind of activities performed by NLP systems.

4.1. Background

Natural language processing systems take strings of words (sentences) as their input and produce structured representations capturing the meaning of those strings as their output. The nature of this output depends heavily on the task at hand. A natural language understanding system serving as an interface to a database might accept questions in English which relate to the kind of data held by the database. In this case the *meaning* of the input (the output of the system) might be expressed in terms of structured SQL queries which can be directly submitted to the database.

The first use of computers to manipulate natural languages was in the 1950s with attempts to automate translation between Russian and English [Locke & Booth]. These systems were spectacularly unsuccessful requiring human Russian-English translators to pre-edit the Russian and post-edit the English. Based on World War II code breaking techniques, they took individual words in isolation and checked their definition in a dictionary. They were of little practical use. Popular tales about these systems cite many mis-translations including the phrase "*hydraulic ram*" translated as "*water goat*".

In the 1960s natural language processing systems started to examine sentence structure but often in an ad hoc manner. These systems were based on pattern matching and few derived representations of meaning. The most well known of these is Eliza [Weisenbaum] though this system was not the most impressive in terms of its ability to extract meaning from language.

Serious developments in natural language processing took place in the early & mid 1970s as systems started to use more general approaches and attempt to formally describe the rules of the language they worked with. LUNAR [Woods 1973] provided an English interface to a database holding details of moon rock samples. SHRDLU [Winograd] interfaced with a virtual robot in a world of blocks, accepting English commands to move the blocks around and answer questions about the state of the world. Since that time there has been parallel development of ideas and technologies that provide the basis for modern natural language processing systems. Research in computer linguistics has provided greater knowledge of grammar construction [Gazdar] and Artificial Intelligence researchers have produced more effective mechanisms for parsing natural languages and for representing meanings [Allen]. Natural language processing systems now build on a solid base of linguistic study and use highly developed semantic representations.

Recently (during the 1990s) natural language systems have either focused on specific, limited domains with some success or attempted to provide general purpose language understanding ability with less success. A major goal in contemporary language processing research is to produce systems which work with complete threads of discourse (with human like abilities) rather than only with isolated sentences [Russell & Norvig(a)]. Successes in this area are currently limited.

4.2. Problems

Two problems in particular make the processing of natural languages difficult and cause different techniques to be used than those associated with the construction of compilers etc for processing artificial languages. These problems are (i) the level of ambiguity that exists in natural languages and (ii) the complexity of semantic information contained in even simple sentences.

Typically language processors deal with large numbers of words, many of which have alternative uses, and large grammars which allow different phrase types to be formed from the same string of words. Language processors are made more complex because of the irregularity of language and the different kinds of ambiguity which can occur. The groups of sentences below are used as examples to illustrate different issues faced by language processors. Each group is briefly discussed in the following section (in keeping with convention, ill-formed sentences are marked with an asterix).

1. The old man the boats.

2. Cats play with string.
   * Cat play with string.

3. I saw the racing pigeons flying to Paris. I saw the Eiffel Tower flying to Paris.

4. The boy kicked the ball under the tree. The boy kicked the wall under the tree.

1. In the sentence "The old man the boats" problems, such as they are, exist because the word "old" can be legitimately used as a noun (meaning a collection of old people) as well as an adjective, and the word "man" can be used as a verb (meaning take charge of) as well as a noun. This causes ambiguity which must be resolved during syntax analysis. This is done by considering all possible syntactic arrangements for phrases and sub-phrases when necessary.

   The implication here is that any parsing mechanism must be able to explore various syntactic arrangements for phrases and be able to backtrack and rearrange them whenever necessary.

2. The problem with the second sentence in this group is that there is no numeric agreement between the subject and the verb (one is a singular form, the other plural). Grammars must be expressive enough to specify checks for such anomalies and also specify actions which should take place if they occur. Mechanisms to signal failure in processing such cases are useful. For example, when combining semantics for "colourless" and "green" in a phrase like "the colourless green car" a signal of failure marks a sub-phrase as ill-formed and prevents it being considered any further. In the case of problems with the numeric agreement between subject and verb it may be more appropriate to signal a warning. A warning marks a sub-phrase as potentially-flawed but does not reject it outright.

3. Assuming these sentences are taken in isolation so there is no previous dialog which introduces a racing pigeon named "the Eiffel Tower"...

   The sensible way to interpret the meaning of the second sentence is "While I was flying to Paris I saw the Eiffel Tower in its usual position - firmly rooted to the ground". What prevents the second sentence being restructured in the same way as the first is the inconsistency with objects like the Eiffel Tower and activities like flight. Sensible semantic rules must detect this inconsistency and perhaps halt progress on any sub- phrase which implies the Eiffel Tower is involved in a flying activity.

   A semantic rule in this case might reason as follows:
   (i) the set of objects capable of flight is {humans, birds, bats, aeroplanes}
   (ii) the Eiffel Tower is defined as a structural-object
   (iii) structural-objects are not in the set of objects capable of flight so signal "ill-formed semantics" and halt progress on this rule.

4. The implication in the first sentence is that the activity performed by the boy causes the ball to move to a position which is under the tree. The kicking activity has a meaning of "move, using the boy's foot as an instrument to cause that movement". In the second sentence the wall is assumed not to have changed position. The activity which took place was one of "strike, using the foot as an instrument". The apparent disambiguation in this case can take place if it is known that balls are mobile objects (and are often moved using a foot as an instrument) and walls are static objects.

The purpose of examining example sentences like the four above are to introduce the reader to some of the complexity that occurs within natural language statements. The following subsections describe the type of activities carried out by systems which process natural language and therefore have to deal with problems similar to those illustrated above.

4.3. Natural Language Processing - the tasks involved

A simplified view of Natural Language Processing emphasises four distinct stages [Fig 2.1]. In real systems these stages rarely all occur as separated, sequential processes. In the overview that follows it is assumed that syntactic analysis and semantic analysis will be dealt with by the same mechanism - the parser. The rest of this section examines processes shown in the diagram.

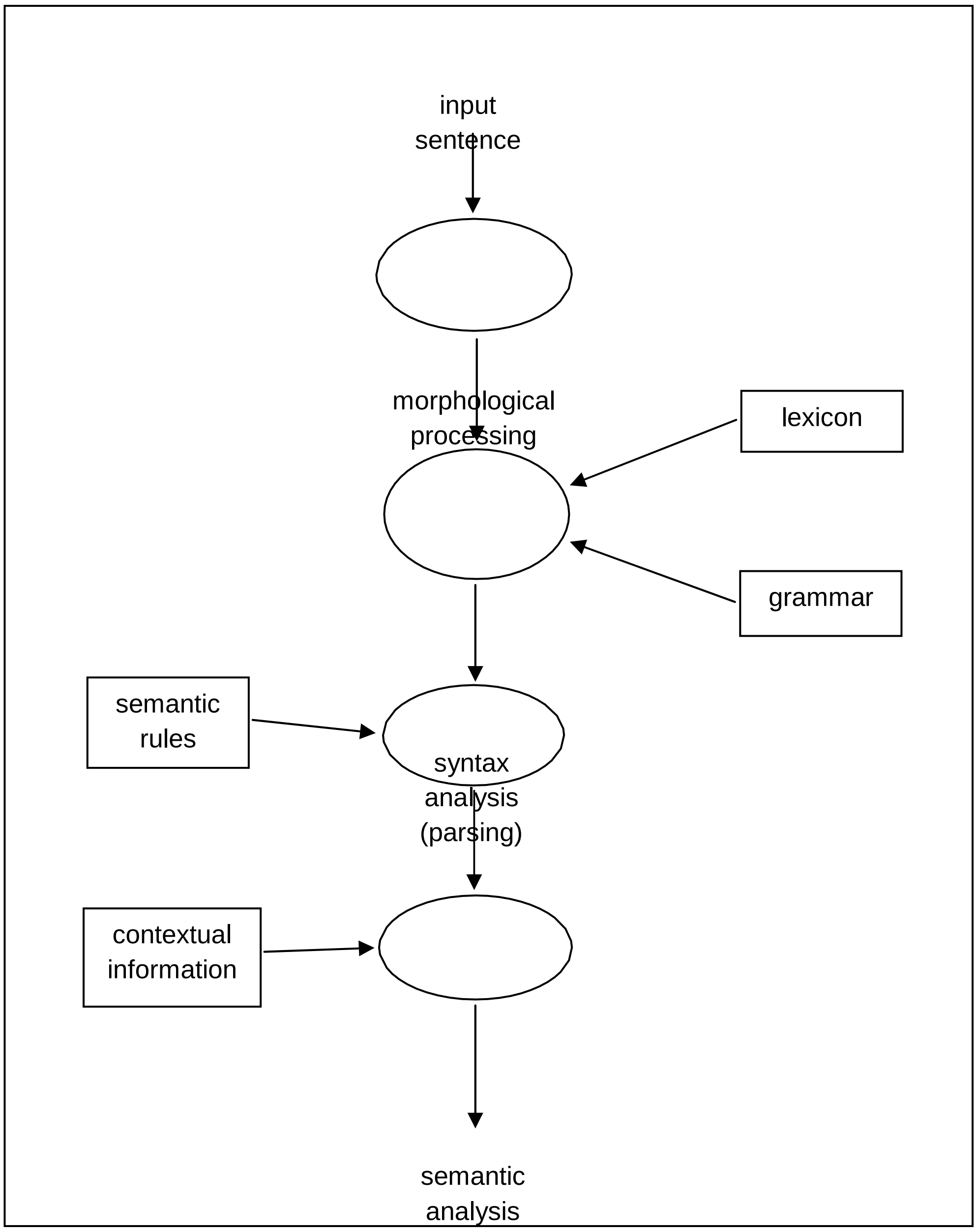


Fig. 2.1. The logical steps in Natural language Processing

4.3.1. Morphological Processing

The preliminary stage which takes place before syntax analysis is *morphological processing*.

The purpose of this stage of language processing is to break strings of language input into sets of tokens corresponding to discrete words, sub-words and punctuation forms. For example a word like "unhappily" can be broken into three sub-word tokens as:

un- happy -ly

Morphology is concerned primarily with recognising how base words have been modified to form other words with similar meanings but often with different syntactic categories. Modification typically occurs by the addition of prefixes and/or postfixes but other textual changes can also take place. In general there are three different cases of word form modification.

Inflection: textual representations of words change because of their syntactic roles. In English, for example, most plural nouns take -s as a suffix (and may require other modification), comparative and superlative forms of regular adjectives take *-er -est* suffixes.

Derivation: new words are derived from existing words. This manufacturing of words often occurs in a regular manner allowing following clear morphological rules. For example, in English some adjectives take -ness as a suffix when being used to create nouns (happy happiness). The same principles apply in most human languages though the rules are different[1].

Compounding: new words are formed by grouping existing words. This occurs infrequently in English (examples include "headache" and "toothpaste") but is widely used in other languages where it is morphologically possible to have infinitely long words.

The nature of morphological processing is heavily dependent on the language being analysed. In some languages single words (used as verbs) contain all the information about the tense, person and number of a sentence. In other languages this information may be spread across many words. For example the English sentence "I will have been walking..." where complex tense information is only available by examining the structure of auxiliary verbs. Some languages attach prefixes/suffixes to nouns to indicate their roles (see figure 2.2) others use word inflections to provide proximity information (see figures 2.3 & 2.4).

Noun + Suffix	Syntactic case	Meaning
Chennai-ukku	dative: destination	To Madras
Chennai-ukku-irundu	dative: source	From Madras
Chennai-le	containment	In Madras
Chennai-ai	object (formal)	Madras

Fig. 2.2. Suffix Attachment for Noun Cases (Formal Tamil) NB: the spellings used are the author's phonetic spellings.

Proximity	**Time**	**Things (inanimate)**
Near	i-ppa (this time: *now*)	i-ndtha (this thing: *this*)
Far	a-ppa (that time: *then*)	a-ndtha (that thing: *that*)
Question	e-ppa (what time: *when*)	e-ndtha (what thing: *which*)

Fig. 2.3. Proximity Information as Prefix Tags (Tamil)

Proximity	**Cow**	**Student**
Near Speaker	kgomo e (this cow)	mo-ithuti yo (this student)
Near Listener	kgomo e-o (that cow)	mo-ithuti yo-o (that student)
Far	kgomo e-le (that cow)	mo-ithuti yo-le (that student)

Fig. 2.4. Proximity Information by Demostrative Pronoun Inflection (Setswana)

As a language, English is more easy to tokenise and apply morphological analysis to than many. In some far-eastern languages words are not separated (by whitespace characters) in their written form (examples include Japanese and some Chinese languages). In many languages the morphology of words can be ambiguous in ways that can only be resolved by carrying out syntactic and/or semantic analysis on the input. Simple examples in English occur between plural nouns and singular verbs: "climbs" as in '*there are many climbs in the Alps*' or '*he climbs Everest in March*'. This example of ambiguity can be resolved by syntax analysis alone but other examples are more complex. "Undoable" could be analysed as ((un-do) -able) or as (un- (do-able)), ambiguity which cannot always be resolved at the syntax level alone.

The output from the morphological processing phase is a string of tokens which can then be used for lexicon lookup. These tokens may contain tense, number, gender and proximity information (depending on the language) and in some cases may also contain additional syntactic information for the parser. The next stage of processing is syntax analysis.

4.3.2.Syntax and Semantics

A language processor must carry out a number of different functions primarily based around syntax analysis and semantic analysis. The purpose of syntax analysis is two-fold: to check that a string of words (a sentence) is well-formed and to break it up into a structure that shows the syntactic relationships between the different words. A syntactic analyser (or parser) does this using a dictionary of word definitions (the lexicon) and a set of syntax rules (the grammar). A simple lexicon only contains the syntactic category of each word, a simple grammar describes rules which indicate only how syntactic categories can be combined to form phrases of different types.

Examples of a simple lexicon and grammar could be:

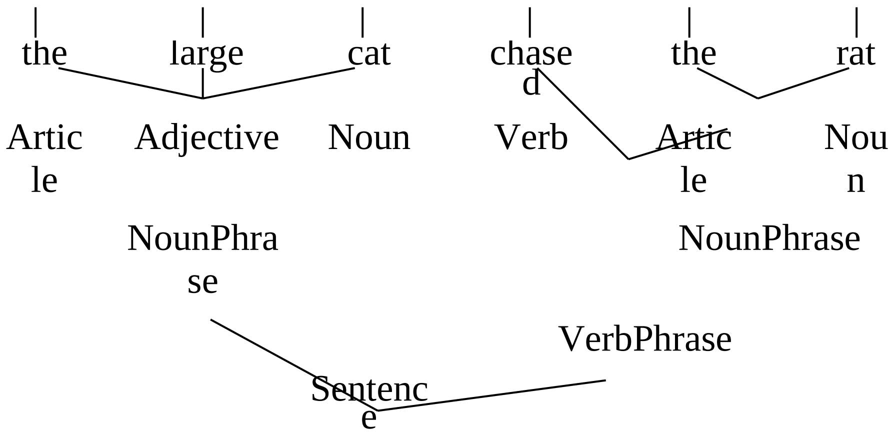
Lexicon

word	category
cat	Noun
chased	Verb
large	Adjective
rat	Noun
the	Article

Grammar

Sentence □ NounPhrase, VerbPhrase[2]
VerbPhrase □ Verb, NounPhrase
NounPhrase □ Article, Noun
NounPhrase □ Article, Adjective, Noun

This grammar-lexicon combination could destructure the sentence "*The large cat chased the rat*" as follows:



Often the task of a language processor is to analyse a sentence in a language like English and produce an expression in some formal notation which, as far as the computer system is concerned, concisely expresses the semantics of the sentence. An interface to a database might, for example, require a language processor to convert sentences in English or German into SQL queries. Semantic analysis is the term given to the production of this formalised semantic representation.

In order to carry out semantic analysis the lexicon must be expanded to include semantic definitions for each word it contains and the grammar must be extended to specify how the semantics of any phrase are formed from the semantics of its component parts. For example the grammar rule above *VerbPhrase* □ *Verb*, *NounPhrase* states how the syntactic group called *VerbPhrase* is formed from other syntactic groups but says nothing about the semantics of any resulting *VerbPhrase*. Using a simplified form of logic the grammar and lexicon can be expanded to capture some semantic information. This is illustrated in the following example.

Lexicon

word	category	semantics
cat	Noun	1x • feline(x)
chased	Verb	1xy • x L y L chased(x, y)
large	Adjective	1x • largesize(x)
rat	Noun	1x • rodent(x)
the	Article	E_1 „gensym0

NB: terms like "feline" and "chased" are used in this example as primitive semantic relations.

Grammar

Syntactic rule	Semantic rule
Sentence □ NounPhrase, VerbPhrase	apply VerbPhrase (NP)
VerbPhrase □ Verb, NounPhrase	apply Verb (NounPhrase)
NounPhrase □ Article, Noun	apply Noun (Article)
NounPhrase □ Article, Adjective, NounPhrase	apply Adjective (Article) L apply Noun (Article)

NB: the above is simplified for readability, apply is used to provide an argument to a 1 form so: apply 1x • rodent(x)(Ralf) □ rodent(Ralf)

Syntactic and semantic structures produced on analysing a simple sentence.

The example above demonstrates how grammar rules are used to specify the method for producing semantic forms. The rule describing the legal syntactic form for a *VerbPhrase*, for example, also describes how to create semantics for verb phrases. In this way semantics are formed and grouped into larger sub-phrases until, after applying all of the relevant rules, a semantic expression describing the whole sentence is produced.

4.3.3. Semantics and Pragmatics

After semantic analysis the next stage of processing deals with pragmatics. Unfortunately there is no universally agreed distinction between semantics and pragmatics. This document, in common with several other authors [Russel & Norvig(c)] makes the distinction as follows: semantic analysis associates meaning with isolated utterances/sentences; pragmatic analysis interprets the results of semantic analysis from the perspective of a specific context (the context of the dialogue or state of the world etc). This means that with a sentence like "*The large cat chased the rat*" semantic analysis can produce an expression which means *the large cat* but cannot carry out the further step of inference required to identify the large cat as Felix. This would be left up to pragmatic analysis. In some cases, like the example just described, pragmatic analysis simply fits actual objects/events which exist in a given context with object references obtained during semantic analysis. In other cases pragmatic analysis can disambiguate sentences which cannot be fully disambiguated during the syntax and semantic analysis phases. As an example consider the sentence "*Put the apple in the basket on the shelf*". There are two semantic interpretations for this sentence. Using a form of logic for the semantics:

1. put the apple which is currently in the basket onto the shelf

   (E$_1$ a : apple ÆE b : basket L inside(a, b)) L E$_1$ s : shelf fi puton(a, s)

2. put the apple into the basket which is currently on the shelf

   E$_1$ a : apple L (E$_1$ b : basket ÆE$_1$ s : shelf L on(b, s)) fi putin(a, b)

Pragmatic analysis, in consulting the current context, could choose between the two possibilities above based on the states and positions of objects in the world of discourse.

4.4. Section Summary

This section has provided examples of some of the problems associated with analysing human languages and has described the most important stages in Natural Language Processing. The next section examines the different approaches that can be used to specify grammars and lexicons.

The specification of a grammar and lexicon for even a small subset of a natural language is a non-trivial activity. The correctness/integrity of the grammar and lexicon are of great importance since their use underpins all syntactic and semantic analysis.

4.5 Specifying Grammars for Natural Languages

The single most important consideration for the design of Lkit concerned the types of grammar the toolkit would accept. This section gives a brief overview of the various types of grammar most commonly used to specify natural languages for computer-based analysis.

All grammars perform similar tasks, they specify a set of rules which work together to define legal sentences in their language. Context free grammars (like BNF) are the easiest for people to write and are easy to deal with computationally. However natural human languages are not context free. Most grammar formalisms draw a compromise by writing rules in a context free style then augmenting these rules to deal with the kind of complications discussed earlier (numeric agreement, handling semantics etc).

There are a variety of different approaches used for grammar construction and specification but it is useful to examine the use of any grammar in two ways:

1. as a conceptual design tool - a strategy for capturing and describing the complexity of a natural language;
2. as a formal notation for describing the syntactic (and possibly some semantic) characteristics of a language in a way that can be used by a parser.

The first of these categories is concerned primarily with the theory and structure of languages, the second is concerned with the description and implementation of grammars. The relationship between the conceptual design of a grammar and the notation used to specify/code it is similar to the relationship between the conceptual design of a program and the programming language used to write it. In this analogy the parser is similar to an interpreter or compiler that recognises the programming language

Unfortunately it is impossible to completely separate the theoretical/conceptual design of a grammar from issues of implementation. Different conceptual designs of grammar place different requirements on the notations that may be used to code them. Additionally different notations have different representational capabilities and may therefore restrict conceptual designs they can describe. Case Grammars, for example, are mainly concerned with the theoretical structure of a language and can be implemented by different parsers using different notations for grammar rules. However Case

Grammars influence the choice of notation used to describe them since some grammatical notations cannot adequately represent them. Conversely notations based on Context Free Grammars (like Recursive Transition Networks) impose severe constraints on the conceptual design of grammars which they describe.

The rest of this section describes various types of grammar indicating, where appropriate, their requirements and/or restrictions.

4.5.1. Case Grammars

Case Grammars [Fillmore] attempt to describe any given sentence in terms of a fixed frame of slots (called cases) which explicitly capture information about any activities described in the sentence, the instigators of those activities, positions, times, etc. Though there is no universally agreed set of cases or their names a common subset is outlined below.

Case	**Meaning**
Action	the action which takes place (usually related to the main verb of a sentence)
Actor	the instigator of the action (often an animate entity which/who *does* the action)
Object	the entity which is acted upon or is changed by the action
Source	the starting position for an entity (ie: its position before the action takes place)
Destination	the resulting position for an entity (after the action has completed)
Location	the location for the action
Instrument	an object/entity used in order to cause the event (eg: *key* in "*He unlocked the door with a key*"
Time	the time/date of the action

For example case analysis of the sentence "*Gary repaired the car in the garage on Sunday*" could generate the following case frame:

Action	repairs
Actor	Gary
Object	car
Location	garage
Time	Sunday

With Case Grammars rules describe syntactic constraints but also describe manipulations geared towards producing case frames (sets of case slots). These may be flat (as in the example above) or nested hierarchical structures which can form the basis of semantic representations.

To implement a Case Grammar the notation used to express rules must allow them to produce case frames (ie: structures without a predetermined form). Grammar notations whose only output reflects the syntactic structure of their input are not suitable since they cannot be used to construct case frames.

4.5.2. Semantic Grammars

Semantic Grammars (also called Domain Specific Grammars) [Rich & Knight] place no restrictions on the notations used to describe them but use rules built around domain specific phrase types rather than typical abstract categories. For example, in a medical domain the sentence "*there is a bleeding ulcer in the stomach towards the lesser curve*" the two phrases "*in the stomach*" and "*towards the lesser curve*" might be classified by a semantic grammar as a locator-phrase and a location-qualifier respectively. In a more general purpose grammar for English these two phrases would be of the same type: prepositional-phrase.

The motivation for doing this is that, in limited domains, grammars can be developed faster by people who are familiar with the domain but are not necessarily expert linguists. The problems of grammar construction are reduced by allowing it to focus on the domain and ignore features of language which are not used. Additionally, domain experts are more able to help describe (and debug) suitable grammar rules when the terminology is more natural to them.

Unfortunately it is generally recognised that semantic grammars are more difficult to extend in order to capture syntactic generalisations or to deal with increasingly complex/varied language as the domain expands. In short domain-specific/semantic grammars do not scale up well.

Semantic grammars can be constructed using various different grammar notations but typically the result of applying a semantic grammars reflects the semantic structure of its input rather than its syntactic structure. To achieve this semantic grammars associate semantic actions with each of their grammar rules, any grammar notation used with semantic grammars must support this.

4.5.3. Definitive Clause Grammars (DCGs)

Definitive Clause Grammars [Pereira & Warren] use rules which are based on logical implication expressions. These expressions are equivalent to Prolog clauses with the result that a Prolog interpreter can be used as a parser for DCGs. An unadapted Prolog interpreter would perform a top-down, depth first parse which is not particularly efficient.

Rules in DCGs are Horn clauses which may have multiple antecedents but only a single consequent. Non-terminal symbols in the grammar act as predicates, for example:

NounPhrase(NP) L VerbPhrase(VP) ﬁ Sentence(append(NP, VP))

In practice DCG rules are written like rules in other grammars (rather than as logical expressions) so the rule above could be written:

Sentence □ NounPhrase VerbPhrase.

Rules are augmented to allow context sensitive checks to be specified, for example to ensure numeric agreement between subject and verb thereby accepting sentences like "Cats$_{plur}$ play$_{plur}$ with string" while rejecting others like "Cat$_{sing}$ play$_{plur}$ with string". Typically such augmentations might be written:

Sentence □ NounPhrase(num (n)) VerbPhrase(num (n))

Additional augmentation allows semantics to be assembled for the target category of a rule

Sentence(semantics append(s1,s2))

NounPhrase(semantics (s1)) VerbPhrase(semantics (s2))

The appearance of such rules becomes more complex when different augmentations are combined, ie:

Sentence (semantics append(s1,s2))
 NounPhrase(num (n), semantics (s1)) VerbPhrase(num (n), semantics (s2))

Rules are complicated further when transformations & checks are specified for semantic structures. DCGs are popular because of their declarative style but are often associated with inefficient parsing strategies.

4.6. Lexical Functional Grammars (LFGs)

Lexical grammars present a different approach to grammar/lexicon construction by removing rules from the grammar and embedding them instead in the lexicon. The justification for this is that the valence of words (the number and type of other syntactic groups they associate with) is a feature associated with words rather than their meanings or syntactic classification. The valence of a word defines the legal structure of sentences it may occur in. For example, the verbs "dined", "ate" and "devoured" are all part-tense verb forms and all have similar meanings but (because of their transitivity) impose different restrictions on sentences they can be used in. The following sentences illustrate the point:

a. The guests devoured the meal.
b. * The guests devoured.
c. * The guests dined the meal.
d. The guests dined.
e. The guests ate the meal.
f. The guests ate.

A lexical rule for "ate" would contain the following kind of information:

ate	specifiers	$NP_{type = animate, participation = mandatory}$
	complements	$NP_{type = food\text{-}stuff, participation = optional}$

This rule states that the word "ate" is mandatorily specified (prefixed) by a noun phrase describing something animate and optionally complemented (followed) by a noun phrase describing something which is a food-stuff.

Because lexical grammars have rule like constructions occurring inside the lexicon their use imposes different sets of requirements on any syntax analyser/parser that implements them. This is also true because the structure of lexical rules tends to be different to the structure of rules used with other grammars, DCGs for example.

Proponents of LFGs cite their elegance in handling some complexities of language that other grammars struggle to represent and the way that specifier/complement notation can be uniformly applied to all parts of speech. Others claim that LFGs are less suitable for fast prototyping small grammars aimed at restricted language domains and that lexicons tend to contain even more duplication than usual.

As with many areas of linguistics the choice between an LFG approach and others is largely down to personal preference. Consider the use of prepositions with nouns [Murphy]. Different nouns take different pronouns, eg:

...demand for...

...cause of...

...increase in...

...solution to...

LFGs would handle this by embedding notions of legal pronoun use inside rules attached to the lexical entries for the nouns concerned. Other formalisms would attach type information to the lexical definitions of pronouns and sets of acceptable pronoun types to definitions of nouns. General purpose grammar rules would then be responsible for checking for legal noun- pronoun type correspondence.

4.7. Augmented Transition Networks (ATNs)

ATNs [Woods 1970] are usually described as a highly developed and extended form of State Transition Network (STN). A basic STN for natural language would have arcs labelled as some terminal syntactic category. Recursive Transition Networks (RTNs) are a development of STNs. They consist of collections of small STNs with arcs which can be labelled with names of other networks (non-terminals). ATNs are a further development of RTNs which allow arcs to be labelled with tests and actions which may bind or reference ATN variables. Fig 2.4 shows a simple ATN.

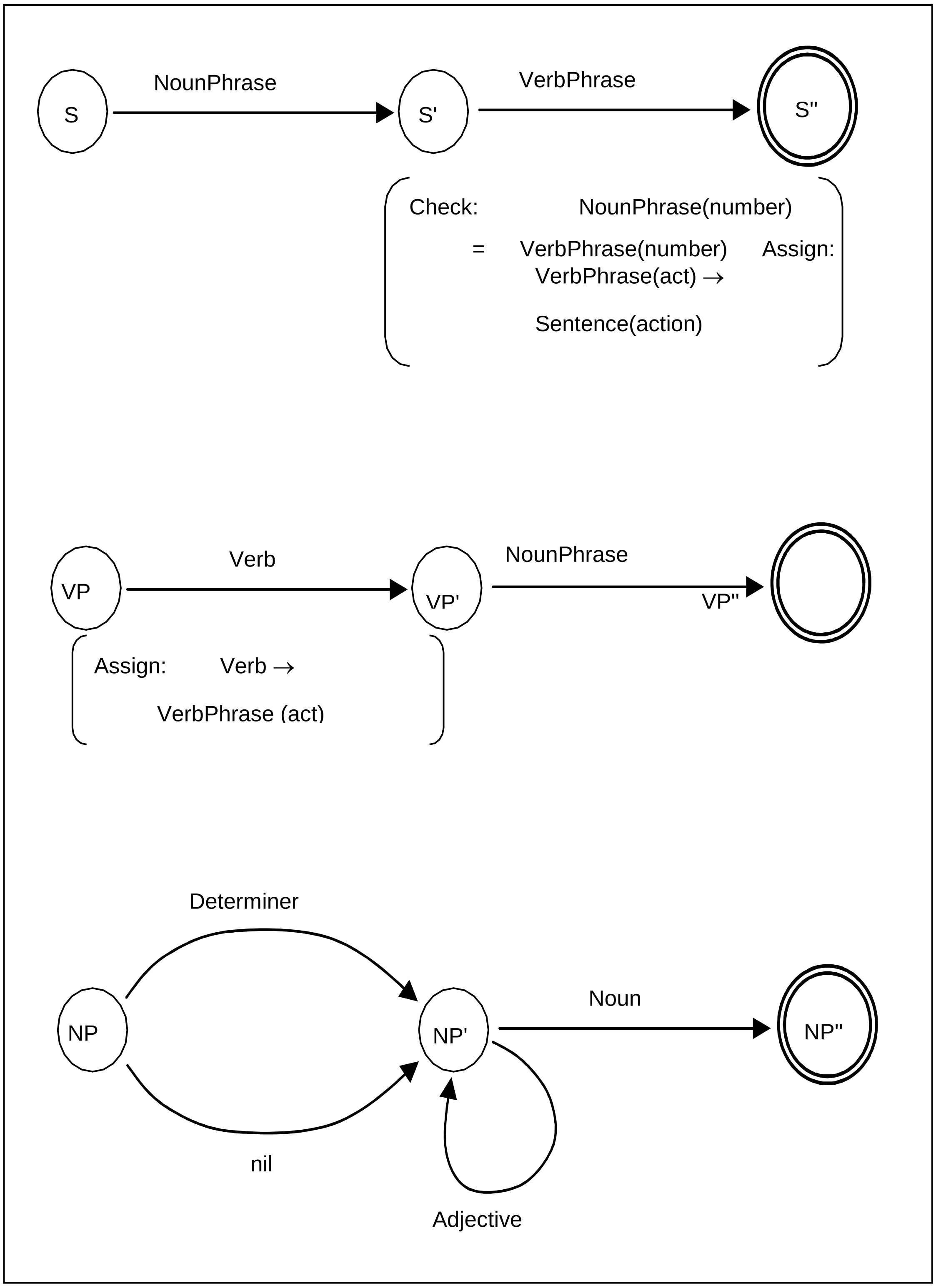


Fig 2.4. An example of a simple ATN.

ATNs are used by parsers which are flexible, typically top-down and can build structures of arbitrary complexity. They were probably the most widely used type of grammar in Natural Language Processing after the success of LUNAR [Woods 1973] until the mid 1980s. They lend themselves to the construction of powerful grammars which do not have to be described in a network-like formalism but none-the-less tend to be more procedural in nature than some other grammars. Since the 1980s ATN use has declined, this has been in part due to some researchers switching to DCGs which appear declarative in style and use augmentations based on unification rather than assignment (as in ATNs). Others researchers have moved away from ATN grammars in order to switch to parsers which can employ a greater variety of search strategy, in these cases there is often some similarity between the grammars they use and ATN grammars.

4.8. Semantics

The goal of most NLP[4] systems is to extract meaning from their language input. This *meaning* might ultimately be expressed as SQL for instance if the interface is for a database but in order to generate target representations NLP systems must first create intermediate representations to capture and refine meanings from their input. This intermediate representation that captures *meaning* is the semantic representation of the system.

In general, semantic representations need to capture details of objects and their relationships, events and the chains of causality that tie them together. A detailed discussion of semantic representations is beyond the scope of this document but the following section intends to highlight the issues of particular importance to semantic representations used specifically for NLP.

4.8.1. Forms of Semantic Representation

Any semantic representation can be viewed on three levels:

1. The conceptual level: what is represented and why. This level describes the representational capabilities. A simple representation may only capture details of objects and object-object relationships. A more sophisticated representation may capture details of events, their timings, possible outcomes and inter-event dependencies.

2. The abstract form: most representations can be described in abstract forms, using diagrams for example. These diagrams show what the representation makes explicit and what must be inferred. The abstract form shows the structure of the representation and the primitives used to build it. Abstract forms are not deterministically derived from the conceptual form. Even very simple concepts can be expressed by a variety of different abstract forms. Fig 4.1 & 4.2 show examples of this.

3. The physical realisation of the representation: the implementation form, data structures and inference mechanisms.

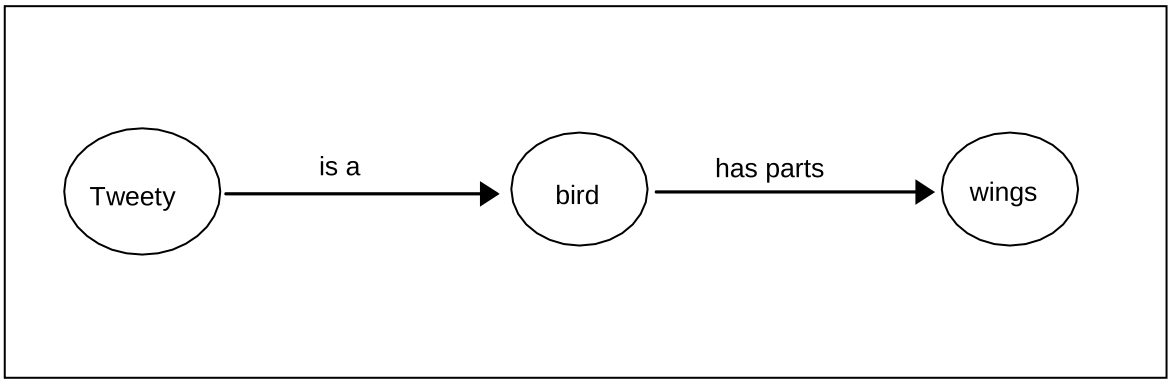


Fig. 4.1.. An abstract form capturing the relationship between *Tweetie* and *wings*

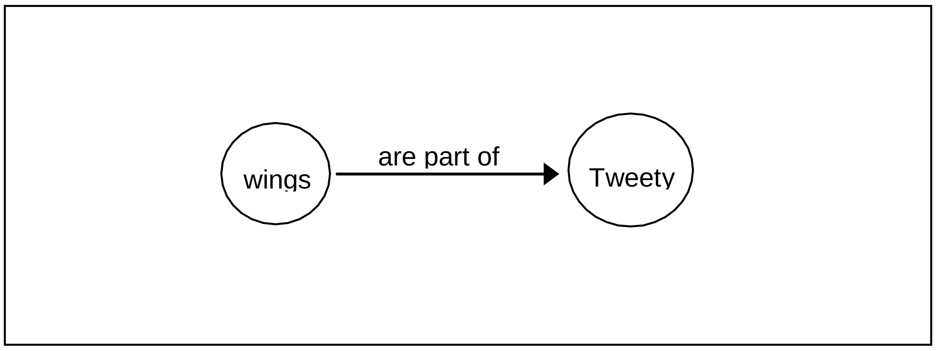


Fig. 4.2. An alternative abstract form capturing the relationship between *Tweetie* and *wings*

The choice of form for level 3 has impact on the capabilities of the abstract representation. A poor choice can limit its expressive power or make it clumsy to use. Some representations use simple logic for their implementation form. This has some advantages: it is well understood, it has precise, unambiguous semantics and it supports inference. However once this choice has been made it imposes restrictions at level 1 & 2 (it is difficult to deal with degrees of truth and the influence of time etc).

A major design effort in any Natural Language Processing system is the semantic representation. Pioneering work into suitable representations took place in the 1970s with Wilks's investigation into semantic primitives for his translation system [Wilks] and Schank's Conceptual Dependency Theory [Schank]. Wilks classified objects according to their semantic categories for example *PhysicalObjects*, *AnimateObjects* and *ManiplulableObjects* (those that could move or be moved). These classifications were used to guide his system in forming unambiguous descriptions of the language input. For example given the sentences:

1. The boy kicked the ball under the tree.
2. The boy kicked the wall under the tree.

The classification of "ball" as a *ManipulableObject* would cause the system to recognise that the activity described by sentence (1) was one of movement, while the classification of "wall" as a *StaticObject* in sentence (2) implied a *striking* activity.

Schank's Conceptual Dependency Theory reduced all verb forms to specialisations of a small number of primitive actions (eleven in the original work) which described activities like moving a physical object, applying a force to an object, ingesting foodstuff, etc. Each primitive action had a clearly defined set of legal uses, known preconditions and known results. For example an *Actor* could only move an *Object* if they could manipulate that object and they were in the same location. The result of moving the *Object* would be that both *Actor* and *Object* were at a new location.

In addition the representation described objects by a set of object states which took numeric values. For example the state *Health* had a range between +10 (perfect health) and -10 (dead). The physical condition of an object had a range from -10 (broken beyond repair), through -5 (damaged) to 0. The most interesting of these states relate to the mental & emotional states of humans and other intelligent entities, the state of *Joy* had a range from -10 to +10 with -5 indicating depression and +5 indicating happiness.

State values were associated with adjectives and adjectival phrases and were also generated by rules of inference. A rule of inference might state for example, the conditions under which a human could suffer a deterioration in their *Joy* value.

As well as specifying sets of primitive acts and states, Conceptual Dependency defined how these primitives could be combined to describe more complex concepts. In addition to other relationships, Conceptual Dependency explicitly represented causal relationships between one concept and another. A sentence like "*The boy ate a bad apple, it made him sick*" would map onto a form with two separate concepts: *eating-a-bad-apple* and *being-sick* with the second of these being (unintentionally but unavoidably) caused by the first.

The work of both Schank and Wilks, while dated in Artificial Intelligence research terms, introduced many of the ideas still considered significant when developing semantic representations. Their work identified the importance of developing useable semantic primitives and mechanisms for grouping these primitives into higher level forms. Designers of lexicons and grammars must consider both of these issues: the primitives and how they are grouped.

4.8.2. Building Semantic Representations

Semantic representations are built from semantic fragments attached to words in the lexicon. Semantic rules contained within grammar rules describe how these fragments are combined to form the semantics for larger phrases. For example if the target representation for a sentence like "*The boy eats an apple*" is some thing like INGESTS(young-male-human, fruit) and the relevant lexical entries are somewhat trivially:

eats INGEST
boy young-male-
human apple fruit

The (unlikely) grammar rule below would produce the required semantics:

Sentence □ NounPhrase1 Verb
NounPhrase2 Semantics:

$\text{Verb}_{\text{semantics}}(\text{NounPhrase1}_{\text{semantics}}, \text{NounPhrase2}_{\text{semantics}})$

This approach to deriving semantics by combining the action of a number of rules is known as "combinatorial semantics". Language processors can implement it two ways:

1. apply the relevant semantic processing each time a new phrase is formed;
2. apply semantic processing only when a complete and satisfactory parse has been found for an input sentence.

Each method has its own advantages. Strategy 1 has an obvious processing overhead but can generate useful information during the parse which helps to prevent exploration of semantically ill-formed phrase.