Computational Humour: Analysis and Generation of Humorous Texts in Hindi

Thesis submitted in partial fulfillment
of the requirements for the degree of

Masters of Science
in
Computational Linguistics by Research

by

Srishti Aggarwal
201325049
srishti.aggarwal@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
October 2018

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Computational Humour: Analysis and Generation of Humorous Texts in Hindi" by Srishti Aggarwal, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Prof. Radhika Mamidi

To My Parents

Acknowledgments

I am grateful to all the wonderful people in my life for their encouragement and support for my research work. First and foremost, I thank my parents for their love, support and faith in me. Thank you for being there for me, and for always getting me last minute tickets whenever I felt like coming home. I thank my sister for always wishing me well and being my silent supporter. I thank my family - my grandparents, uncles, aunts, cousins - for celebrating the smallest of victories, encouraging me in the biggest of defeats, and for keeping a joke running about the topic of my research.

I would like to express my gratitude towards my advisor Prof. Radhika Mamidi who guided and supported me at each step of this work, right from when I showed interest in this topic more than two years ago. She showed great patience in giving me the freedom to work on my own pace, while also keeping me grounded and on track. I am thankful to Kritik Mathur, who worked with me on this project for a semester and co-authored two submissions with me. All our discussions and the many ideas we tried out really helped shape this work.

My sincere gratitutde towards Harshit Harchani, Nisarg Jhaveri and Madhuri Tummalapalli for supporting and guiding me throughout my research work, for always lending a patient ear whenever I had something to discuss, or even just wanted to talk out loud, and for giving me company all through this last year. I thank Mohit Goel for the many pep talks he gave me, at all the right times. A special thanks to Arpit Merchant for being there for me and always helping me out whenever and wherever possible. I also want to thank my other friends - Smriti Singh, Vivek Ghaisas, Sanjana Sharma, Anvesh Rao, Kaveri Anuranjana for all the fun times together that made my time at IIIT Hyderabad memorable.

Abstract

Humour is an intricate part of human behaviour and a valued quality when developing interpersonal relationships. It is an interesting and challenging domain to work in owing to its subjectivity. Humour is one of the most sophisticated forms of human intelligence and creativity, intimately related with factors such as world knowledge, reasoning and perception. With our AI systems becoming increasingly intelligent, there is now a demand for systems which are more human-like than ever before. No AI system could be complete without the ability to to process and use an integral human phenomenon like humour. Humour is often expressed through language in various forms such as funny stories/anecdotes, jokes, witty sayings, puns etc. This verbally expressed humour is of interest to NLP researchers because it deals with spoken and textual language. The subfield of Computational Linguistics and Artificial Intelligence which aims to develop systems which can use and understand humour like humans do is called Computational Humour.

A complete computational humour system has to contain two main modules - a humour interpreter, and a humour generator. Both these modules could have their independent uses in a variety of applications, but when it comes to interactive and comprehensive AI systems, both are equally important. In this thesis, we present a first ever work for computational humour for Hindi. We consider both subfields in computational humour - Humour Understanding and Humour Generation, and work on specific applications in both.

We study Hindi-English code mixed puns by focusing on automatic recovery of the target words/phrases in them. We define code-mixed puns, analyse the type of structures present in them, and classify them into two main categories - intra-sentential and intra-word code-mixed puns. We then propose a four-step methodology to recover intra-sentential code-mixed puns. On evaluating our model on a test dataset of such puns, we observe that we are able to successfully recover 67%. We recognise several limitations of our system, like the inability to deal with highly unusual and creative language use.

For the problem of computational generation of humour, we consider two tasks. First, we build a prototype system which automatically generates a simple form of *Dur-se-Dekha* jokes in Hindi using a small lexicon and handcrafted rules. Second, we design an algorithm which can induce humour in a short non-humorous text by performing single word subsitutions on the basis of phonetic similarity

and sentiment opposition. We perform human evaluation on the results of both and observe that we are successfully able to generate funny instances in both these cases.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Philosophers and researchers since the times of Aristotle [45] and Plato [58] have been trying to understand and define the concept of humour. There are dozens of different definitions of humour. The Oxford English Dictionary defines humour as “that quality of action, speech, or writing which excites amusement; oddity, jocularity, facetiousness, comicality, fun" [78]. According to other researchers, humour can been seen as “any object or event that elicits laughter, amuses or is felt to be funny" [73], or a “non-serious social incongruity" [7], or even an individual trait or a characterictic of a person(rather than a statement/event) [41, 42]. Definitions of humour have evolved from those which relied on laughter into a multi-disciplinary topic of research and spans over the fields of sociology, physiology, psychology, phiosophy, literature and linguistics. Some researchers agree that there cannot be a single definition that covers all aspects of humour, that humour is impossible to define [73].

Despite the lack of an all-encompassing definition, humour is a phenomenon we come across and recognise every day. It is an intricate part of human behaviour, communications and a valued quality when developing interpersonal relationships. We are amused by various things every single day - when a friend falls down but is not actually hurt (physical slapstick), funny sound effects in a movie, unexpected twists in situation comedies, visual gags and so on [69].

A very common way to express humour is through language in the form of funny stories/anecdotes, jokes, witty sayings, puns etc. This is called “Verbally Expressed Humour". Ritchie [65] defines verbally expressed humour as “humour conveyed in language, as opposed to physical or visual humour, but not necessarily playing on the form of the language". As language and text are the central elements of this type of humour, this is of most interest to linguists, and also the main focus for NLP researchers.

Humour scholars dating back to Cicero and Ouentilian identify two main subclasses within verbally expressed humour - Referential and Verbal humour [73], where humour created from the language/text form itself if called verbal humour and humour merely conveyed by the language form is called referential or situational humour. Verbal humour is dependent on the form of the language used and its

linguistic features while referential humour is dependent on content of those words. Cicero suggested translation as a test to assign one of these categories to an instance of humour - verbal humour will lose it's funniness when translated while referential humour will retain it [73].

Verbally expressed humour can be in the form of a contextually independent joke, such as those found in joke books, which are repeated verbatim or with minor changes by a speaker in order to ellicit humorous response from the listeners (also called canned jokes) [73], or, it can be in the form of spontaeneously created conversational humour in the form of puns, witticisms, retorts, etc [20]. One of the very common techniques used to produce conversational humour is wordplay. Wordplay jokes depend on using words that sound similar, but have different and contrasting meanings. The contrast in meaning is what creates conflicts or violates expectations and as a result, produces humour [88]. Puns are the most common form of wordplay jokes we come across in everyday lives.

Humour is an interesting and challenging domain to work in. It is a fundamentally existing part of human nature, which is not only difficult to define, but also hugely subjective at the same time. The same situation/joke could be funny to one person, but offensive to another. It can amuse a person at one time but annoy the same person at another. The sense of humour of a person varies across space, time, cultures, individual characterictics and even the moods of a person. It is one of the most sophisticated forms of human intelligence and creativity, intimately related with factors such as world knowledge, reasoning and perceptiom.

1.1 Computational Humour

Computational humour is a branch of computational linguistics (CL) and artificial intelligence (AI) which aims to develop systems which can use and understand humour like humans do. As Stock [80] puts it - "a computational humour system should be able to: recognize situations appropriate for humour; choose a suitable kind of humour for the situation; generate an appropriately humorous output; and, if there is some form of interaction or control, evaluate the feedback." Thus, a complete computational humour system has to contain two main modules - a humour detector and interpreter, and a humour generator. Both these modules have their independent uses in a variety of applications, but when it comes to interactive and comprehensive AI systems, both are equally important.

Most of the existing research in computational humour has been focused on verbally expressed humour. Dealing with just text (rather than physical movements, images, videos or audio) removes one level of complexity because even though computers can't actually understand text, they can easily read and reproduce it. The second reason for this focus is that there is significant work done by linguists in developing terminology and concepts for atleast partially defining aspects like grammar and semantics of a text [69, 66]. This gives a computational humour system some basic tools and models to build upon.

1.2 Motivation

The problem of computational humour is deemed to be AI complete [80, 13], i.e, the difficulty of solving it is equivalent to that of solving the problem of artificial general intelligence - making computers behave as intelligently as people, or to enable computers to interact with humans as other humans do. Humour appreciation and usage comes so naturally to humans that they are unaware of the complex cognitive steps, world knowledge, human intelligence and creativity behind it. Encoding all of this such that a computer system is able to behave as humans do is a difficult task.

Natural language understanding in itself is a hard problem which hasn't been solved yet. Understanding humour becomes even more difficult given its complex and subjective nature. Humour understanding requires the computer to have a deep understanding of the context, world knowledge and the situation. Similarly, creating humour requires the sytem to not only construct a perceivably funny text that fits in the context of the situation, but to also judge the situation as joke appropriate.

Past works in this domain have failed to define a single all-encompassing framework which can act as the theoretical foundation for computational humour. Such a theory does not seem possible to develop given the complexity, subjectivity and incompleteness of the existing theories and approaches [13]. We posit that making incremental advances by targeting limited humour settings can help achieve the broad goal of a generally funny computer system.

Most of the related literature focuses on the English language, one of the most common languages spoken around the world with a vast amount of natural language processing (NLP) resources available. In this thesis, we attempt to prototype computational humour systems for Hindi or code-mixed Hindi language. To the best of our knowledge, this is one of the first works for computational humour in Hindi.

One of the most obvious as well as ambitious applications of computational humour is in Human Computer Interaction (HCI). The aim is to develop AI systems which can act as social actors or companinions [66] [54]. Just as humour is an integral part of natural human interactions, any AI system is incomplete without the capability to process and use humour. For example, one of the most comprehensive AI system we have seen ideated has been TARS, that tactical robot in the 2014 movie, Interstellar. TARS has witty, sarcastic, and humorous traits programmed into him to make him a better suited companion. There is even an option to change these settings. Other, real world AI systems, like Siri have some humorous traits built in too, although those might be a little too scripted. This is because the creators of these systems see the importance of dealing with humour if these AI systems are to seem human-like.

A little less ambitiously, advances in computational humour can make computer interfaces better. According to Binsted [12] humour modeling using even some basic observtions can help make "clarifi-

cation queries less repetitive, statements of ignorance more acceptable, and error messages less patronising" and, overall, make a computational agent seem more human.

Other than AI interfaces, computational humour can have applications in other domains as well. Humour generation systems could be used to generate funny news headlines or creative advertisements which can catch audience's attention [13]. Companies spend a lot of time and manpower to come up with these; computational humour generation systems can help make these tasks easier. Such a system could also be used by ventures which need a steady supply of jokes, for example, greeting card makers [66]. Verbal humour generation and understanding systems can also find aplications in language educational systems, to help users understand multiple possible usages of language and help improve text comprehension.

1.3 Our Contributions

In this thesis, we present, to the best of our knowledge, first ever work which deals with computational humour for the Hindi language, or looks into code-mixed humour. We consider both the subproblems in computational humour - understanding and creation, and work on specific applications in both as defined below.

- **Humour Understanding**

  We study Hindi-English code mixed puns by focusing on automatic recovery of the target words/phrases in them. We define code-mixed puns, analyse the type of structures present in them, and classify them into two main categories - intra-sentential and intra-word code mixed puns. We then propose a methodology to recover intra-sentential code-mixed puns. With our model recovering around 67% of the jokes accurately, we observe encouraging results on testing the model on a limited dataset. We also perform an error analysis on the results to better understand the limitations of our proposed approach, and gain added insights into the complexity of the problem.

- **Humour Generation**

  We attempt two tasks in the case of computational generation of humour. First, we consider the case of *Dur-se-Dekha* jokes in Hindi. We study the structure of these jokes, and using toy data, design a method to automatically generate a particular subtype of these jokes. We test the computer-generated jokes against human-created ones to prove that it is indeed possible to generate similarly funny jokes using an automated system.

  Second, we consider the task of designing small humorous texts in Hindi formed by making single word substitutions in non-humorous texts based on certain hueristics. We compare the results of our method with the adapted version of a previous work done for the English language to show that our sentiment-based substitution approach results in more desirable humorous outputs.

1.4 Thesis Organisation

This thesis is organised into 5 chapters. The current chapter defines our problem statement and places it in the context of the field of computational humour. The next chapter (Chapter 2), gives a background into the various perspectives researchers have used to understand/theorize humour, and a summary of the important works previously attempted in the domain of computational humour. Chapter 3 describes our work in automatic target recovery of Hindi-English code-mixed puns while Chapter 4 decribes our attempts at automatic generation of humour, where we work to tackle two subproblems namely - generation of *Dur-se-Dekha* jokes in Hindi, and induction of humour in short texts using word substitutions. We conclude in Chapter 5, also giving possible directions for future extensions of our work.

Chapter 2

Background and Related Work

2.1 Theories of Humour

2.1.1 Conventional Theories of Humour

Researchers have grouped theories of humour into three main groups - superiority theories (also called disparagement-based thoery), relief or release-based theories and incongruity-based theories.

1. **Superiority Theory**
   Disparagement, or Superiority theories are based on “hostility, superiority, malice, agression, decision or disparagement" [59]. It is based on the observation that people laugh at other people’s infirmities, especially if they are enemies [84]. This class of theories is among the oldest ideas in humour theories [41, 74], dating all the way back to Plato (Philebus) and Aristotle (Poetics). Aristotle, for example, considered laughter to be intimately related to ungliness and debasement whereas Plato treated it as an emotion that overrides rational self-control [51].

   Thomas Hobbes (1588-1679), who was probably the originator of the “superiority theory" in the 17th century, regarded laughter as a result of sudden access to self-esteem (“sudden-glory") when we realize that our own situations compare favourably with the misfortunes or infirmities of others. He also found that we laugh at our own past follies, provided we are conscious of having surmounted them or at unexpected successes [50, 41, 73].

   Because this theory is based on the interpersonal and social aspect of humour, superiority/ disparagement theory is mainly of interest to sociolinguists of humour, but of limited application elsewhere [73].

2. **Relief Theory**
   Relief/Release-based theories are what connect humour to laughter. The principle behind relief theory of humour is that “humour releases tensions, psychic energy, or that humour releaes one from inhibitions, conventions and laws" [73]. Sigmund Frued (1856-1938), probably the most influential proponent of this theory [73], stated that all laugh-producing situations are pleasurable

because they save psychic energy. It brings to the person pleasure because it spares expenditure of feeling, comedy because it spares expenditure of ideas and joking because it spares expenditure of inhibition [22].

One particular instance where we see this theory of humour in action is in movies. In thriller and adventure movies, in scenes where audience would experience high tensions and suspense, the plot would often include comic relief in the form of a side comment to bring down the tensions slightly, allowing the audience a temporary relief from the high tension emotions before continuing the sequence further.

Relief theory of humour is also used to account for wordplay-based jokes like puns, or humour created due to the infractions of Grice's cooperative principle[1] [26] as "liberation" from the rules of language [73].

3. **Incongruity Theory**
   Incongruity theories focus on the cognitive elements of humour [41]. These theories suggest that humour arises when something unexpected takes place, and hence it's effect is more powerful when "it brings to light a real connection between two things normally regarded with quite different attitudes or when it forces on us a complete reversal of values" [50].

   The surprise element in a joke has been emphasised by incongruity theorists too. They speak of the set-up and the punch line [77]. The set-up is the first part of the joke which sets the expectation. The punch line is the last part that violates that expectation. In the language of Incongruity Theory, the joke's ending is incongruous with the beginning. There has been some debate among cognitively-oriented humour theorists [41] as to whether incongruity alone is a necessary and sufficient condition for humour or whether its resolution is also important (**Incongruity-Resolution Theory**) [64, 84].

   Incongruity theory is now the most widely accepted and the dominant theory of humour in philosophy and psychology and even linguistics [51].

We find that none of these theories is adequate by itself to explain every type of humour. Each is valid for certain kinds of humour only. For example, the superiority theory would explain ridicules, situation jokes as well as satires well, whereas the incongruity theory finds it difficult to explain the same but gives a satisfactory explanation when dealing with puns, riddles, and conundrums which relief theory cannot account for [50].

[1] Grice's cooperative principle assumes that when two parties communicate, both implicitly assume conversational cooperation - that both parties cooperate to achieve mutual conversational ends. Grice says that this cooperation manifests itself in the form of four "maxims" - That of quality (give the most helpful amount of information), quantity (do not say what you believe to be false), relation (be relevant) and manner (put what you say in the clearest, briefest, and most orderly manner).

2.1.2 Linguistic Theories of Humour

1. **Script-based Semantic Theory of Humour (SSTH)**
   Victor Raskin's Script-based Semantic Theory of Humour (SSTH) [59] is one of the first formal theories [71] for verbal humour. It is not based on any one particular group of theories mentioned in the previous subsection, and is compatible with all.

   A "script" in Script-based Semantic Theory of Humour is defined as "an enriched, structured chunk of semantic information, associated with word meaning and evoked by specific words" [59]. This theory mentions two necessary and sufficient conditions that all joke-bearing texts must satisfy [60, 71]:

   (a) "The text is compatible, fully or in part, with two different scripts". These two compatible scripts overlap fully or in part in the text.

   (b) "The two scripts with which the text is compatible are opposite" (i,e. negation of each other [71]), following a list of some basic oppositions like real/unreal, normal/abnormal, life/death etc [72, 59, 60].

   Joke : "Is the doctor at home? " the patient asked in his bronchial whisper. "No," the doctor's young and pretty wife whispered in reply. "Come right in".

   This joke, given by Raskin [59], is compatible with two scripts - DOCTOR and LOVER, which are opposite on the sex vs non-sex basis. It is important to note that the unexpectedness of this ambiguity in scripts of the text is as important as its presence.

   The main limitation of this theory is that it is based only on the semantics, and so, while it can detect whether a short text is a joke, it cannot tell how similar two jokes are.

2. **General Theory of Verbal Humour (GTVH)**
   The General Theory of Verbal humour (GTVH), is an improvement on SSTH [4]. It defines each joke in terms of six "Knowledge Resources" rather than just scripts. These knowledge resources are defined as below [72]:

   - Script Opposition (SO): deals with script opposition (same as presented in SSTH).
   - Logical Mechanism (LM): represents the mechanism in which the two senses (scripts) in the joke are brought together.
   - Situation (SI): the setting of the jokes, including objects, participants, surroundings, actvities etc.
   - Target (TA): the individual or group which forms the butt of the joke. Target is the only optional parameter among the six KRs since non-aggressive jokes do not have a target.
   - Narrative Strategy (NS): The narrative organisation of the joke, such as a riddle, question-answer, dialogue etc.

- Language (LA): The actual linguistic components (lexical, syntactic, phonological etc.) that form the text of the joke.

According to GTVH, each joke is a six-tuple, with an instance of each of the knowledge resources as parameters (Joke:{SO,LM,SI,TA,NS,LA}). Two jokes are different if at least one of these parameters is different.

GTVH gives a hierarchal ordering for the six knowledge resoures: SO,LM,SI,TA,NS,LA ordering from less similar and less determined to most similar and most determined. A study conducted by Ruch, et al [72] showed that there is a linear increase in similarity between pairs of jokes selected along this heirarchy (with the exception of LM Knowledge resource). This means that two jokes that differ only at the Script Opposition are less similar than jokes that differ only on the Situation parameter. Also, the more Knowledge Resources two jokes have in common, the more similar they are.

2.2 Computational Humour

2.2.1 Humour Detection and Understanding

Short texts such as tweets, one-liner jokes, and puns have received most attention in research on computational understanding and detection of humour. A brief overview of these works is given below.

1. **Knock Knock jokes**
   Julia Taylor's [87] work was one of the first investigations into computational recognition of humour. This work focused on a particular type of wordplay jokes - the "Knock Knock" jokes. A typical Knock Knock (KK) joke is a dialogue between two people that uses wordplay in the punchline according to the template given below:

Template:

$Line_1$: "Knock, Knock"
$Line_2$: "Who is there?"
$Line_3$: any phrase
$Line_4$: $Line_3$ followed by "who?"
$Line_5$: One or several sentences containing one of the following:
- $Type_1$: $Line_3$
- $Type_2$: a wordplay on $Line_3$
- $Type_3$: a meaningful response to $Line_3$.

Example : Knock, Knock
Who is there?
Ashley
Ashley who?
Actually, I don't know.

Based on Raskin's SSTH, the basic assumption is that the original phrase and the complimentary wordplay have two different scripts that overlap in the setup of the joke. Recognizing humour in a KK joke arises from recognizing the wordplay. The algorithm learns n-grams from a dataset of knock knock jokes and provides a heuristic location of where wordplay may or may not occur. It uses a wordplay generator to produce an utterance that is similar in pronunciation to a given word, and the wordplay recognizer determines if the utterance is valid. Once a possible wordplay is discovered, a joke recognizer determines if a found wordplay transforms the text into a joke [88].

This work was successful in recognising wordplay, but not in recognising the joke. This was mainly due to small training dataset used to build the n-gram dataset, a limited number of wordplay examples that unless it was an exact match, the joke recogniser did not work.

2. **Detection, Identification and Understanding of Puns**
Hempelmann [28] studied target recoverability of imperfect puns, arguing that a good model for it provides necessary groundwork for effective automatic pun generation. He worked on a theory which models prominent factors in punning such as phonological similarity and studied how these measures could be used to evaluate possible imperfect puns given an input word and a set of target words.

Miller and Gurevych [48] described a Word Sense Disambiguation (WSD) model for target recovery of homographic or perfect puns. This system works by running each pun through a variation of the Lesk algorithm [37] that scores each candidate sense according to the lexical overlap with the pun's context. Jaech, et al [32] developed a computational model for target recovery of paronomasic puns using techniques for automatic speech recognition, and learned phone edit probabilities in puns which give insight into how people perceive sounds and into what makes a good pun.

Miller, et al. 2017 [49] describes the various systems developed as a part of the 2017 SemEval task on detection and interpretation of english puns which had three subtasks - Pun detection (For each context in the data set, the system had to decide whether or not it contains a pun), Pun location (for any or all of the contexts, systems had to make a single guess as to which word is the pun) and pun interpretation (for any or all of the contexts, systems had to annotate the two meanings of the given pun by reference to WordNet sense keys). Ten teams participated in the challenge and proposed rule-based approaches ([93]), WSD-based approaches [55, 56], and supervised classification approaches using features like Word2Vec (w2v) embeddings, semantic distance based on wordnet, phonetic distance or edit distance [95, 30, 31, 19, 89].

Yokogawa, 2002 [97] analyzed ungrammatical Japanese puns and generated target candidates by replacing ungrammatical parts of the sentence by similar expressions. The system was based on the hypothesis that the similarity of articulation matches similarity of sounds.

3. **Computational Models for Humour Recognition**
   Mihalcea and Strapparava [46] were one of the first to build a computational humour recognition model for distinguishing between humorous and non-humorous one-liners. For this model, they investigated humour specific stylistic features (alliteration, antonymy, adult slang) and content-based features through experiments where the humour recognition task is formulated as a traditional text classification problem. They found alliteration to be one of the most discriminative stylistic features for humour. They also observed that humorous texts tend to have a more human-centric vocabulary, more negatives and words having negative connotation, mentions of professional communities and human weaknesses (for e.g. stupidity, ignorance, alcohol,lie etc.)

   Yang et al. [96] explored latent structures behind humour on the same dataset - Incongruity (semantic distances between words), Ambiguity (possible senses of the words), Interpersonal effect (polarity and subjectivity) and Phonetic style (alliteration and rhyme), as possible features for a humour detection model. Their experiments showed that Incongruity features perform the best among all latent semantic structures when distinguishing between puns and other non-humorous text, while both Ambiguity and Phonetic features contributed to recognition performance on the one-liner jokes dataset. This demonstrated that humour characteristics are expressed differently in different contexts and datasets.

   There have also been studieson detecting humour on social media like Twitter, which have not only attempted to diffenrentiate humorous tweets from non-humorous ones, but also humour from other figurative language styles like irony [63, 8, 98].

2.2.2 Humour Generation

Over the years, various researchers have worked on giving computers the ability to generate humour. While various techniques and approaches have been used, one thing we see in common is that all of the humour generation systems have been built for classes of jokes having a standard template or fixed structure. We discuss some of these systems below.

1. **Light-Bulb Joke Generator (LIBOJ)**
   Created by Raskin and Attardo in 1994 [61], LIBOJ is one of the first ever computational humour generation systems. LIBOJ is a simple template based system, which picks a stereotyped entry from its lexicon and substitutes it in the template to create a jokes. For example, the famous light-bulb jokes "How many Poles does it take to change a light bulb? Five. One to hold the light bulb and four to turn the table he's standing on." is obtained by combining Template_1 and Entry_1

Template$_1$: How many (group name) does it take to screw in a light bulb?
(Number$_X$). One to (activity$_1$) and (number$_Y$) to (activity$_2$).
Condition: X = Y + 1

Entry$_1$: (Poles ((activity$_1$ hold the light bulb)
(number$_X$ 1)
(acrivity$_2$ turn the table he is standing on)
(number$_Y$ 4)))

This system has only a limited mechanism for joke generation and does not use any other lexicon or linguistic knowledge.

2. **Joke Analysis and Production Engine (JAPE)**
JAPE is a computational model to generate simple punning riddles, formalised by Kim Binstead and Graeme Ritchie in the mid 1990s [14, 11, 15, 67]. The model's main punning mechanism, homophone substitution, generates only a subset of punning riddles i.e. those with noun phrase punchlines. The components that make up JAPE - the lexicon, consisting of fifty-nine words and twenty-one NPs, a homophone base containing pairs of phonologically identical lexemes, six schematas, fourteen templates, and a post-production checker to sift out more of the obvious non-jokes all contain general humour-independent information, which is one of the main features of this system. Each produced joke has a setup and a punchline. Some of the examples of the jokes produced by JAPE are:

Example$_1$: "What do you call a quirky quantifier?
An odd number"

Example$_2$: "What's the difference between money and a bottom?
One you spare and bank, the other you bare and spank."

Although this system was never analysed fully owing to the lack of enough data for a statistically sound analysis, authors observed that it suffered from "joke fatigue" because of the system's tendency to generate a lot of similar type of jokes, and a general lack of enthusiasm for this type of jokes.

Later, Dan Lehr also integrated JAPE with his natural language robot named **Elmo** [39] where the user could ask Elmo to "tell me a riddle", or Elmo could produce humour relevant to the user input. While the integration of the pun generator with natural language interface was successful, it failed to produce relevant humour smoothly on arbitrary user input.

3. **Tom Swifties**
"Tom Swifties are pun-like utterances ascribed to the character 'Tom', in which (typically) a

manner adverb enters into a formal and semantic relation with other elements in the sentence," [38]. For example:

Example$_3$: "I hate seafood", said Tom crabbily.

Everything produced by this generator is in the form of Template$_3$:

Template$_2$: "SENTENCE" said Tom ADV[manner]

The adverb (ADV) in the Template$_3$ must have a phonetic link to the meaning of at least one word in the SENTENCE, and be semantically related to it. Thus, in Example$_6$, there is a semantic relationship between the words "seafood" and "crab" and a phonetic link between the words "crab" and "crabbily" [38].

4. **Homonym Common Phrase Pun (HCPP) Generater**
   Homonym Common Phrase Pun Generator was developed by Christopher Venour in 1999 [94]. The idea was to take "commmon phrases" like idioms (e.g. "kick the habit", "jump ship" etc) and commonly connected words ("knead the dough", "serial killer", etc), and create puns on these "common phrases" using word sense and spelling ambiguity in their homonyms. Using common phrases simplifies the generation of puns since of the meanings is already built in into the cliched meaning of the phrase, so, only one sentence articulating the other meaning of this word in the common phrase needs to be generated from scratch. The minimum requirement considered for generating homonym puns is a semantic relation between a homonym of the base word ("common phrases" in this case) and a word or phrase in the target sentence. The system uses different algorithms to deal with each syntax, a set of 36 common phrases, and a humour-independent lexicon 240 words contains semantic relations between nouns, noun phrases, verbs, verb phrases and adjectives. Given below are examples of the jokes generated by the HCPP generator [94]

   Example$_4$: The pheasant breathes oxygen. Fowl air.

   Example$_5$: The sailor bears a stress. Pier pressure.

   It was noticed that a lot of the force of a HCPP joke depended on subverting the familiar, hence, the more familiar the phrase, the higher the impact.

5. **Witty Idiomatic Sentence Creation Revealing Ambiguity in Context (WISCRAIC)**
   WISCRAIC [44] is a program which generates witticisms that use phonological ambiguity to create jokes based around idioms as well as the explanations for the produced jokes. This system, which tested with 84% success in generating explanations for WISCRAIC produced jokes, can be used as an aid for teaching English idioms to non-native speakers [44]. The program consists of three modules [44]:

- "Joke Constructer – the module that contains information about the required elements and relationships in a joke. This module uses dictionary of idioms, dictionary of professions, general dictionary, and lexicon.
- Surface-form Generator – the module that uses grammar to convert an input from Joke Constructor into a joke.
- Explanation Generator – this module takes the elements provided by Joke Constructer and uses grammar to generate an explanation of relations between the elements."

An example of a WISCRAIC joke and explanation given below, is taken from [44]:

Example$_6$: "The friendly gardener had thyme for the woman!""

Explanation: The word time, which is part of the idiom [have, time, for, someone]
is a homonym of the word thyme.
A HOMONYM is a word that sounds like another word.
—
| LINK | between thyme and gardener:
———————————
| thyme is a type of plant
| a gardener works with plants
'friendly', which is associated with the idiom [have, time, for, someone]
was selected from other adjectives as it has the highest imagability score:439"

6. **HAHAcronym: Humorous Agents for humorous Acronyms**
   As the first computational humour project sponsored by the European Commission, HAHAronym: Humorous Agents for Humorous Acronyms, aimed to to realize a working prototype of an acronym ironic-reanalyser and generator [81, 82, 83]. A central part of this system is an incongruity detector/generator to detect semantic mistached between expected sentence meanings and other meaning.

   The project takes an existing acronym as input and after anylysing the actual meaning and context, comes up with a fun parody of it using a word substitution algorithm onn the actual meaning using humour-independdnt resources and features like - wordnet domains, wordent relations, semantic opposition, rhyme and extreme-sounding words. Some of the examples of the acronym reanalysis are [83]:

Example$_7$: MIT - Massachusetts Institute of Technology
Mythical Institute of Theology

Example$_8$: FBI - Federal Bureau of Investigation
Fantastic Bureau of Intimidation

7. **The Mnemonic Sentence Generator (MSG)**
Developed in 2001, the main aim for MSG [43] was to build a system which helps the user remember an alphanumeric password string by producing a possibly funny sentence corresponding to the password. The system waord on 8 character aphanumeric string, generating a sentence consisting of 2 clauses, which in turn consist of 4 words each, following Template$_3$.

Template$_3$: (W1= Person Name) + (W2= Positive-Verb) + (W3=Person Name + "s") + (W4= Common Noun) + ", while" + (W5= Person Name) + (W6= Negative-Verb) + (W7= Person-Name + "s") + (W8= Common Noun)

An example of what the program can do is the following sentence, generated from password "AjQA3Jtv": "Arafat joined Quayle's Ant,while TARAR Jeopardized thurmond's vase" [43].

8. **The FEVer Generator**
The FEVer program is designed to generate simple one line puns obtained through "Familiar Expression Variations" (FEVs) [90]. The variations are generated through replacement of a single word in the original expression (e.g. proverbs, movie titles etc) within certain morphological (the substituted word must have the same Part-of-speech tag), phonetic (the substituted word should be phonetically similar to the original word) and semantic constraints (taboo word vs non-taboo word).

Example$_9$: Woman:Impossible ([90], non-taboo)

Example$_{10}$: Lust Busters ([90], taboo)

A similar methodolgy was used on a SMS corpus by [92] to generate reinforcement based "adult humour", by ensuring that the substituted word is a "taboo word". [91] showed that the tabooness of the substituted word was one of the most determinate features in the perception of the joke.

Example$_{11}$: Remember to get the book from fat person.

9. **Unsupervised joke generation from big data**
Petrovic and Matthews were the first people who attempted the task of joke generation without using labeled training data or handcoded rules, but instead using large quantities of unannotated data [57]. They worked on generating jokes having the structure as given in Template$_4$.

Template$_4$: I like my X like I like my Y, Z. (X, Y, Z are variables to be filled in, where X and Y are nouns and Z is typically an attribute that describes X and Y.)

They generated humour by performing fully unsupervised content selection to fill in the given selection. They used Google n-gram model [25], English Wordnet [33] and encoded four of their own assumptions about what kind of relationships (noun attribute similarity, attrribute surprisal etc) between the three variables X, Y and Z would make a joke funny. The system was able to generate jokes such as the ones in the examples below, which were considered funny by human raters 16% of the time as compared to 33% when they were created by humans.

Example$_{12}$: I like my coffee like I like my war, cold.

Example$_{13}$: I like my relationships like I like my source, open.

10. ***Libitum***

Libitum is an algorithm that helps humans generate humour in a MadLib, a popular fill-in-the-blank game [29]. Mad Libs is a phrasal template word game where one player prompts others for a list of words to substitute for blanks in a story, before reading the – often comical or nonsensical – story aloud [2]. The algorithm generates candidate words, and its core component is a machine-trained classifier that can assess whether a potential fill-in-the-blank word is funny, based on several features like hint type, length o the word, the candidate probabilities in language model and the blank's surrounding context. The human then has to decide which of these candidate words to fill in the blank with.

Example$_{14}$: Cats are the most *barbaric* (adjective) pets in the world.
They were probably first kept because they ate *humans* (animal).
Later cats were *bullied* (verb [past]) because they are *corrupt* (adjective)
. . . Cats are active carnivores, meaning they hunt *online* (adjective) prey.

Their testing shows that Libitum successfully aids humans in filling in Mad Libs that are usually judged funnier than those filled in by humans with no computerized help.

[2] https://en.wikipedia.org/wiki/Mad_Libs

Chapter 3

Understanding Code-Mixed Puns

In this chapter, we consider a subtype of humour - puns, which are a common type of wordplay-based jokes. In particular, we consider code-mixed puns which have become increasingly mainstream on social media, in informal conversations and advertisements and aim to build a system which can automatically identify the pun location and recover the target of such puns. We first study and classify code-mixed puns into two categories namely intra-sentential and intra-word, and then propose a four-step algorithm to recover the pun targets for puns belonging to the intra-sentential category. Our algorithm uses language models, and phonetic similarity-based features to get the desired results. We test our approach on a small set of code-mixed punning advertisements, and observe that our system is successfully able to recover the targets for 67% of the puns.

3.1 Introduction

Puns are one of the simplest and most common forms of wordplay humour in the English language. They are also one of the most widespread forms of spontaneous humour [68] and have found their place in casual conversations, literature, online comments, tweets and advertisements [40, 86]. Puns are a hugely versatile and commonly used literary device and it is essential to include them in any comprehensive approach to computational humour.

We consider Hindi-English code-mixed puns and aim to automatically recover their targets. The **target** of a pun is its phonologically similar counterpart, the relationship to which and whose resolution (recovery) in the mind of the listener/hearer induces humour. For example, in the pun “The life of a patient of hypertension is always at steak." the word “steak" is the pun with target “stake".

With India being a diverse linguistic region, there is an increase in the use of code-mixed Hindi-English language (along with various others) because bilingualism and even multilingualism are quite common. Consequently, we have also seen an increase in the usage of code-mixed language in online forums, advertisements etc. Code-mixed humour, especially puns have become increasingly popular

because being able to use the same punning techniques but with two languages in play has opened up numerous avenues for new and interesting wordplays. With the increasing popularity and acceptance for the usage of code-mixed language, it has become important that computers are also able to process it and even decipher complex phenomena like code-mixed humour.

Traditional Word Sense Disambiguation (WSD) based methods [49, 55, 56]cannot be used in target recovery of code-mixed puns, because they are no longer about multiple senses of a single word but about two words from two different languages. Code-switching comes with no markers, and the punning word may not even be a word in either of the languages being used. Sometimes words from the two languages can be combined to form a word which only a bilingual speaker would understand. Hence, this task on such data calls for a different set of strategies altogether.

The rest of the chapter is organised as follows: We define our problem statement and approach in Section 3.2. Section 3.3 gives a brief background into linguistic studies for puns, and works dealing with code-mixed data, Section 3.4 discusses our classification of code-mixed puns, Section 3.5 describes the dataset we use to test our approach and Section 3.6 describes our proposed model for the task of automatic target recovery of Hindi-English code-mixed puns. In Section 3.7, we analyse the performance of our model on a set of puns, and discuss the various error cases. Finally, we outline future work for this task in Section 3.8 and conclude in Section 3.9 with a review of our research contributions.

3.2 Our contribution

We approach this problem in two parts. First, we analyze the types of structures in code-mixed puns and classify them into two categories, namely intra-sentential and intra-word code-mixed puns. Second, we develop a four step pipeline to achieve our goal - Language Identification, Pun Candidate Identification, Context Lookup and Phonetic Distance Minimization. We then test our approach on a small dataset and note that our method is successfully able to recover targets for a majority of the puns. To the best of our knowledge, this is a first work dealing with code-mixed puns.

3.3 Background and Related Work

3.3.1 Linguistic Studies on Puns

Puns are a form of wordplay jokes in which one sign (e.g. a word or a phrase) suggests two or more meanings by exploiting polysemy, homonymy, or phonological similarity with another sign, for an intended humorous or rhetorical effect [1]. Puns where the two meanings share the same pronunciation are known as *homophonic* or *perfect*[1] puns, while those relying on similar but non-identical sounding

[1]Example of a perfect pun: “The tallest building in town is the library — it has thousands of stories!"

words are known as *heterophonic* [28] or *imperfect*[2] puns [99]. In this chapter, we study automatic target recoverability of Hindi-English code mixed puns - which are more commonly imperfect puns, but may also be perfect puns in some cases.

Zwicky and Zwicky [99], Sobkowiak [79] extensively studied various phonological variations in imperfect puns such as strong asymmetry in phoneme substitution. They note that puns show more frequent changes in vowels than in consonants because of their smaller role in target recoverability.

3.3.2 Code-Mixing

Code-mixing is the mixing of two or more languages or language varieties in an utterance. Code-mixing is now recognized as a natural part of bilingual and multilingual language use. Significant linguistic efforts have been made to understand the sociological and conversational necessity behind code-switching [5]; for example, to understand whether it is an act of identity in a social group, or a consequence of a lack of competence in either of the languages. These papers distinguish between inter-sentence, intra-sentence and intra-word code mixing.

Different types of language mixing phenomena have been discussed and defined by several linguists, with some making clear distinctions between phenomena based on certain criteria, while others use 'code-mixing' or 'code-switching' as umbrella terms to include any type of language mixing [52, 23]. In this paper, we use both these terms 'code-mixing' and 'code-switching' interchangeably.

Coming to the work on automatic analysis of code-mixed languages, there have been studies on detecting code-mixing in spoken language as well as different types of short texts, such as information retrieval queries [24], SMS messages [21, 70], social media data [9] and online conversations [53]. These scholars have carried out experiments for the task of language identification using language models, dictionaries, logistic regression classification, Conditional Random Fields, SVMs, and noted that approaches using contextual knowledge were most robust. King and Abney [34] used weakly semi-supervised methods to perform word-level language identification.

We however, use a dictionary based approach for the language identification task. While working with puns, ambiguity in language identification can be an important marker for identifying the pun, so it is more important for us to recognize all possible ambiguities rather than picking just one depending on probabilities. This ability to recognize ambiguities, and the simplicity of a dictionary-based language identification model makes it suited for this task.

[2] An example of an imperfect puns: "with fronds like these, who needs anemones" [99]

3.4 Classification of Code-Mixed Puns

We focus on the task of automatically disambiguating or recovering Hindi-English code mixed puns. For this purpose, it is first necessary to understand what these puns are.

For the purposes of this research, we only consider puns where the ambiguity or the wordplay lies in the code-switching i.e, the pun word and its target are from different languages. For example the pun "Rivers can't hear because *woh behri hoti hai*." is a sentence with the pun being *behri* (meaning deaf) and its target being *beh rahi* (meaning flowing). Here, while the sentence is code-mixed, the pun word and the target both belong to the same language. We do not consider such puns for the present study, since existing works on english puns (see Section 2.2.1) can be adapted for this type of puns.

We analyze the structure of code-mixed puns with the pun word and its target belonging to different languages and propose two broad categories to classify them in - puns where the code-mixing is intra-sentential and the other where it is intra-word. Both these categories are explained below, while we currently develop a model only for the former category.

3.4.1 Intra-Sentential Code-Mixed Puns

Intra-sentential code-mixing is when code changes at the word-level within a sentence. Here, the language varies at the word level. Also, each word of the sentence belongs to one or the other language. Table 3.1 gives examples of puns belonging to this category.

Pun$_1$	:	Grand *Salaam*
Translation	:	Grand Salute
Pun Location	:	*Salaam*
Target	:	Grand Slam
Pun$_2$	:	*Phir bhi* zeal *hai Hindustani*
Translation	:	The zeal is still Indian
Pun Location	:	zeal
Target	:	*Phir bhi dil hai Hindustani*
Translation (Target)	:	The heart is still Indian.

Table 3.1: Examples of intra-sentential code-mixed puns

3.4.2 Intra-Word Code Mixed Puns

Pun$_3$	:	Face *bhukh* with Amul, Mark
Translation	:	Face hunger with Amul, Mark
Pun Location	:	Face*bhukh*
Target	:	Facebook with Amul, Mark
Pun$_4$	:	*Rajnitea?*
Translation	:	Rajni, tea?
Pun Location	:	*Rajnitea*
Target	:	*Rajneeti*
Translation(target)	:	Politics

Table 3.2: Examples of intra-word code-mixed puns

In this category, code mixing is present within a word. New words are formed using Portmanteau[3] or Blending where two or more syllables/phonemes from different languages are blended together to form a single word, resulting in a word which is phonetically similar to the target word. Table 3.2 illustrates examples of intra-word code-mixed puns.

3.5 Test dataset

Most puns we hear or use in everyday conversations are rarely recorded. One of the most common resources to find recorded puns are advertisements, for example, the highly creative and frequently released Amul advertisements in India [40]. Most of these are *contextually integrated* [68] with an image. While such puns may lose their humour out of context, it is still possible to recover their targets without the context image. Figure 3.1 depicts an example of such an advertisement.

To create a dataset to test our model on, we collected 518 advertisements released by Amul in the years 2014, 2015, 2017 and 2018, from their official web page[4]. Of these, 333 were puns, including 121 code-mixed puns as defined in Section 3.1. Contextual information was used to differentiate puns from other ambiguous but non-punning words. We extracted the text of these 121 code-mixed puns and asked 3 annotators to disambiguate them, given just the advertisement text. All three annotators were university students in 22-23 years age group, native Hindi speakers with bilingual fluency in English.

[3]https://en.wikipedia.org/wiki/Portmanteau
[4]http://www.amul.com/m/amul-hits

Figure 3.1: Example of an Amul advertisement[5] containing an intra-word code-mixed pun from 2014.

Total	**Intra-Sentential Code-Mixed**	**Intra-Word Code-Mixed**
110	51	59

Table 3.3: Distribution of the different types of puns in our test dataset.

The annotators were asked to identify the location of the pun in each of the advertisements and write down the target of the pun. Any disagreements between annotators were resolved by mutual discussion.

In a few cases where puns were identified to have multiple targets, we kept all such possibilities in our dataset. A few puns were identified to be non-recoverable because of the lack of contextual knowledge, while a few puns had multiple pun locations. We removed both these types from our dataset, which left us with 110 puns.

Finally, we divided these 110 annotated puns into the two categories as defined in Section 3.1 thereby getting 51 advertisements categorized as intra-sentential code-mixed puns, and the rest as intra-word code-mixed puns (as shown in Table 3.3). We use the former as our test data.

[5]http://www.amul.com/m/amul-hits?s=2014&l=2 (Amul owns all copyrights to this image).

3.6 Our Model

For preprocessing the text we give as input to our system, we first tokenize the advertisement text using NLTK's [16] tokenizer and remove all punctuations. We then give the resultant tokens as input to our model, which is a four step process as described below an dillutrated in Figure 3.2.

Step 1: Language Identification

At this step, we aim to identify the language of each of the tokens in the input text by classifying them into one of the 5 categories: English (E), Hindi (H), Named Entity (NE), Out of Vocabulary (OOV), or Ambiguous (A) for words that could belong to both English and Hindi (Summarised in Table 3.4).

Category	Label
English	E
Hindi	H
Named Entity	NE
Out of Vocabulary	OOV
Ambiguous	A

Table 3.4: Labels for various categories in the language identification step.

We use a dictionary-based lookup method to classify a word in English or Hindi. Since the input is in Roman script, to recognize Hindi words, we use a list of 30k transliterated Hindi words in Roman to their Devanagari counterparts [27]. For the English language, we collected news data from the archives of a leading Indian Newspaper, The Hindu[6]. Data from 2012-2018 under the tags National, International, Sports, Cinema, Television was collected, amounting to 12,600 articles with 200k sentences and around 38k unique words. We use this data to build an English dictionary. Also, we used NLTK's [16] Named Entity Recognition module on the same data to get a dictionary of Named Entities.

We first try to classify all tokens as English, Hindi and NE using these dictionaries. Then, words which are found in both English and Hindi are marked as Ambiguous. The words which do not fall into any of these are classified as OOV.

Step 2: Identification of Candidate Pun Locations

We now identify all possible punning locations in the text. For this, we consider words on the boundaries of language change as candidates for pun locations. Then, all NEs and OOV words are added to

[6]http://www.thehindu.com/archive/

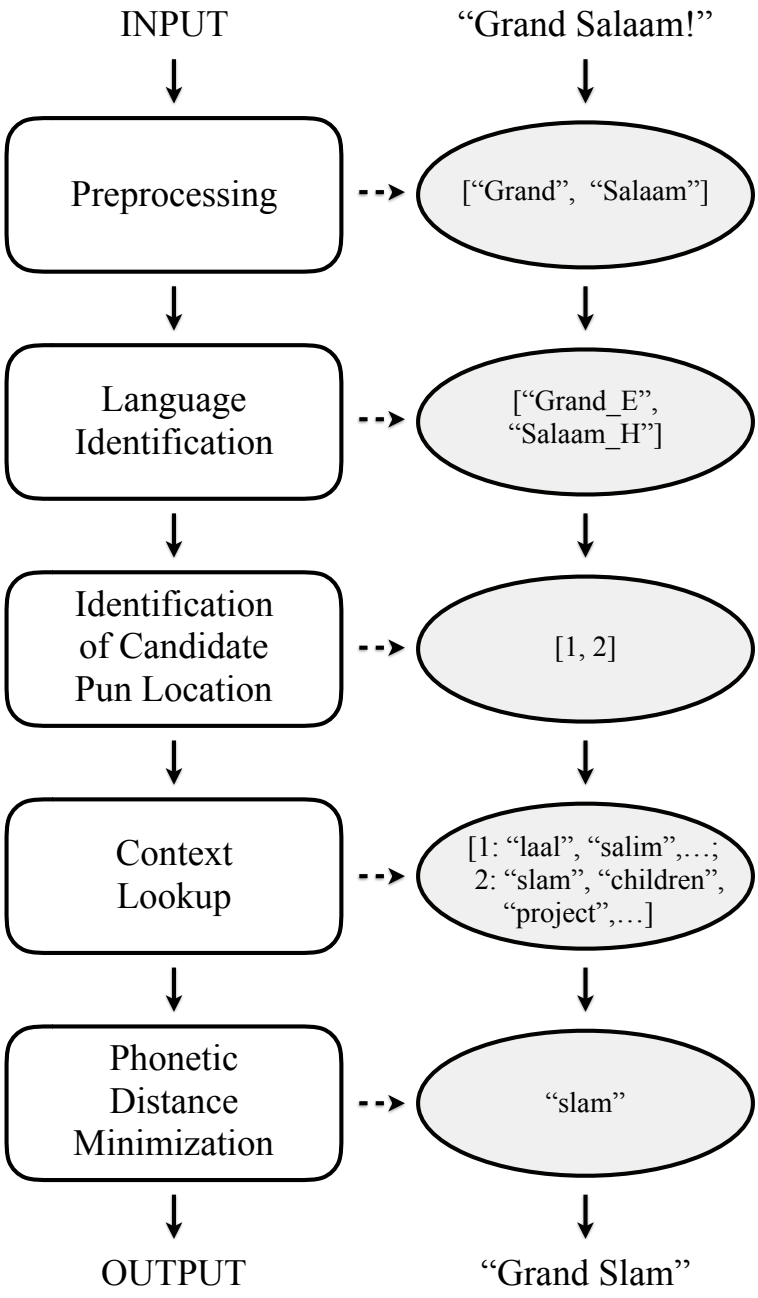

Figure 3.2: Flowchart depicting the model for target recovery of code-mixed puns.

the list of pun candidates as well. Third, if any Ambiguous words exist in the text, we consider it once as English and once as Hindi for the next steps.

Step 3: Context Lookup

In this step, we contextually lookup all the candidate locations using left context and right context to get a list of all words that may occur at that position. We use bi-gram language models we built using Knesser-Ney smoothing[7] [35]. We used the data mentioned in the previous step to build the language model for English, and 100k sentences from Hindi monolingual data from [36] to build the language models for English and Hindi respectively. As it is highly likely that the left and the right context at a pun location belong to different languages, we look at each of those separately instead of taking an intersection of the left and the right context.

Step 4: Phonetic Distance Minimization

Lastly, at each pun location, we calculate the similarity of the word at that location with all the words that can occur at that location depending on the context and pick the most similar words as the possible targets.

To compare words belonging to two different languages on a phonetic basis, we convert both of them to WX notation [10], which denotes a standard way to represent Indian languages in the Roman script. We transliterate our identified Hindi words from Devanagari to WX notation[8]. To convert English words to the same notation, we use the CMU phonetic dictionary [9], which uses a 39 phoneme set to represent North American pronunciations of English words. We build a mapping between this phoneme set and WX notation. Whenever there is no exact parallel between CMU pronouncing dictionary's notation and WX, we used the word's Indian English pronunciation to find the closest match.

Once we convert all to WX notation, we use a modified version of Levenshtein Distance [47] to find most similar words. In this normalized version of Levenshtein distance, we account for a few features like aspirations (for example /p/,/ph/) which are phonemic in Hindi but non-phonemic in English, vowel elongations, rhyme and same beginning or ending sounds.

In case of an OOV word, since it cannot be converted to WX notation due to non-availability of any phonetic transcription, we simply find the words with the least orthographic distance when written in Roman script, using a similar measure as used for phonetic distance with a few more normalizations (for example, considering 'w' and 'v' as similar).

[7] we used the implementation from https://github.com/smilli/kneser-ney to build our language model
[8] We used the open source code available at https://github.com/ltrc/indic-wx-converter for this
[9] http://www.speech.cs.cmu.edu/cgi-bin/cmudict

3.7 Results and Discussion

We test the model explained in the previous section on our test dataset described in Section 3.2 and note that this method is correctly able to recover targets for 34 out of these 51 puns, or around 67% of the puns, which are very encouraging results for this complex task. Examples where the system performed successfully are given in Table 3.5.

Pun$_5$	:	*Aa* bail *mujhe maar.*
Translation	:	Bail, come hit me.
Location	:	Bail
Target	:	*Aa bail mujhe maar*
Translation (target)	:	Come bull, hit me.
Pun$_6$	:	*Doodh* Morning!
Translation	:	Milk Morning
Location	:	*Doodh*, Morning
Target	:	Good Morning!

Table 3.5: Examples of puns successfully recovered by our system

Error Analysis

We perform a thorough error analysis below on the cases our method fails for. These cases are described below.

1. This method does not work when one word in the pun translates to multiple words in the target language. For example, pun$_7$ given in Table 3.6, where a single word *Vir*, which is the name of a person is a pun on “We’re". Our methodology is as of yet unable to recover such puns.

Pun$_7$	:	*Vir* proud of you.
Translation	:	Vir (a name) proud of you
Location	:	*Vir*
Target	:	We’re proud of you.

Table 3.6: Example for the error case were a single pun word maps to more than one word in the target.

2. Our method fails when puns are based on pronunciation of abbreviations because we are not able to transliterate their pronunciation. For example Pun$_8$ given in Table 3.7.

Pun$_8$	:	Greece, EU *ro mat.*
Translation	:	Greece, EU don't cry.
Location	:	EU
Target	:	Greece, *yun ro mat.*
Translation (target)	:	Greece, don't cry like this.

Table 3.7: Example for the error case where the pun is based on the pronunciation of an abbreviation.

3. In certain cases, the puns are recoverable only if Indian English pronunciations are considered, while the CMU pronunciation dictionary we use gives us the North American pronunciations.

Pun$_9$	:	*Main hun* con*? Main hun don.*
Translation	:	I am a con? I am a don.
Location	:	con
Target	:	Main hun kaun? Mein hun don.
Translation(target)	:	Who am I? I am a don.

Table 3.8: Example for the error case where the pun is based on the pronunciation of an abbreviation.

For example, in Table 3.8, the north american pronunciation of the word "con" is /kan/, where as in Indian English it is pronounced as /kon/. The target word *kaun* is also pronounced as /kon/. Hence our system is not able to disambiguate

3. Our model is also unable to recover the target, when the bigram doesn't exist in the language model because it may be a new coinage or a very unusual phrase. In the example given in Table 3.9, the target has "slow food", but "slow" is not an adjective that is normally associated with the noun "food", and so the probability that it will occur in our language model is very low. Hence, our system fails to recover such targets.

Pun$_{10}$	:	Fast food ho ya *sulu* food.
Translation	:	Whether it is fast food or 'sulu' food.
Location	:	*'Sulu'*
Target	:	fast food ho ya slow food.
Translation (target)	:	Whether it is fast food or slow food.

Table 3.9: Example for the error case where the target is rare or a new coinage.

3.8 Future Work

In the future, we want to perform a more comprehensive evaluation of this approach on a larger, more diverse set of puns. We want to improve and extend our approach to be able to recover intra-word code-mixed puns along with the intra-sentential ones that it handles right now. After that, the system should be extended to be able to recover all kinds of puns in code-mixed language, regardless of whether the pun itself is monolingual or code-mixed.

3.9 Conclusions

To conclude, in this chapter, we present a first-ever work on target recovery of code-mixed puns. We study various puns where the wordplay is a result of code-switching, and classify them into 2 categories - puns with intra-sentential code mixing and those with intra-word code mixing. We then propose a methodology to recover the targets for puns belonging to the former category, using only monolingual language data. We test our proposed approach on a small manually annotated dataset, and we see that our system was able to successfully recover 67% of the puns from the set.

Although, we developed and tested our methodology on Hindi-English Code-Mixed puns, our proposed model's pipeline is language independent. This model can easily be adapted to any other language pair by changing the language models and the phonetic models.

Chapter 4

Humour Generation

When it comes to computational language generation systems, humour is a relatively unexplored domain, especially more so for Hindi (or rather, for most languages other than English). In this chapter, we explore and build prototypes for two tasks in the domain of automatic generation of humour. First, we look at *Dur-se-Dekha* jokes, a restricted domain of humorous three liner poetry in Hindi. We analyze their structure to understand how humour is encoded in them and formalize it. We then develop a system which is successfully able to generate a basic form of these jokes. Second, we consider the task of designing short humorous texts in the Hindi language and propose a method to induce humour by performing single word substitutions in a given non-humorous input text on the basis of phonetic similarity and sentiment opposition. We perform human evaluation on the resulting outputs from both the tasks to show that our methods are successfully able to come up with funny instances.

4.1 Automatic Generation of *Dur-se-Dekha* jokes

4.1.1 Introduction

Verbally expressed humour is a common aspect of the everyday use of natural language. Joke, a subclass of verbally expressed humour, is commonly considered the prototypical form of verbally expressed humour, produced orally in conversations or published in collections [20]. Any short text deliberately designed to elicit humorous response could be referred to as a joke [65]. Most researcheres have defined jokes in terms of their structure or constituent parts, and acknowledge that a joke consists of two parts - a setup and a punchline [2, 3, 73, 76, 85]. The setup builds a narrative and induces some form of expectation, while the punchline is the final portion of the text, which violates that expectation or causes a conflict, thereby generating humour due to the production of incongruity and its consequent resolution.

In this section, we attempt to build a system which generates humorous texts in the restricted domain of *Dur-se-Dekha* jokes, a form of humorous three liner poetry[1] in Hindi. This kind of poetry was popular

[1] A three line poem is called a tercet.

in the Indian culture at one point but is almost lost now, with only a few repeated examples available online.

The rest of this section is organised as follows: Section 4.1.2 discusses in detail the structure of *Dur-se-Dekha* jokes as they exist, the way humour is encoded in these jokes and their classfication based on their structure and humour encoding. Section 4.1.3 describes our model to automaticaly generate such jokes. We ask humans to evaluate our jokes and discuss its results in Section 4.1.4. We end with ideas for future extensions of this protoype in Section 4.1.5 and conclude in Section 4.1.6 with a summary of the work.

4.1.2 *Dur-se-Dekha*

Dur-se-Dekha is a focused form of poetic three liner jokes in the Hindi language. These jokes have a typical structure, with the humour lying in the incongruity of the punchline against the expectation of the listener. A typical *Dur-se-Dekha* joke consists of three parts. The structure of a typical *Dur se Dekha* joke can be surmised as in Table 4.1 (Translation of the same can be seen in Table 4.2)

Part_1	:	*Dur se Dekha to* $\underline{NP_1/VP_1}$ *tha,*
Part_2	:	*Dur se dekha to* $\underline{NP_1/VP_1}$ *tha,*
Part_3	:	*Paas jaakar dekha to* $\underline{NP_2/VP_2}$ *tha.*

Table 4.1: The basic structure of *Dur-se-Dekha* jokes

Part_1	:	From afar I saw $\underline{\text{NP}_1/\text{VP}_1}$,
Part_2	:	From afar I saw $\underline{\text{NP}_1/\text{VP}_1}$,
Part_3	:	On going closer it turned out to be $\underline{\text{NP}_2/\text{VP}_2}$

Table 4.2: The basic structure of *Dur-se-Dekha* jokes translated to English.

A *Dur se Dekha* joke employs stylistic features like alliteration, repetition and rhyme, which have previously been associated with humour appreciation [18, 46]. The first part (Part_1) is the setup of the joke, the second (Part_2) is a repetition of the first. This repetition reinforces the idea contained in the first part. The beginning of the third part (Part_3) raises the listener's expectation, and it is what comes at the end that creates humour in this joke. That is the element of surprise, the punch line. Alliteration exists in the first and the third word of the template. Also, the setup and the punchline normally rhyme and contradict in meaning and sentiment, which are two more indicators of humour present in these jokes. This is also precisely why these jokes lose their humour when translated into another language.

On further analysis, we classify these jokes into two main types on the basis of how humour is encoded in them. We discuss both these types below with examples.

Type$_1$: Incongruity between the content of the setup and the punchline

This is the class of the *Dur-se-Dekha* jokes where humour arises out of the incongruity between the subject of the setup and the punchline. These jokes can further have two different syntactic structures:

a) **Jokes with Noun Phrase (NP) punchlines.** The formal structure of these types of jokes is illustrated in Table 4.3, while Table 4.4 gives examples for the same.

Part$_1$	:	*Dur se dekha to $\underline{NP_1}$ tha,*
Part$_2$	:	*Dur se dekha to $\underline{NP_1}$ tha,*
Part$_3$	:	*Paas jaakar dekha to $\underline{NP_2}$ tha.*

Table 4.3: The structure of Type$_{1(a)}$ *Dur-se-Dekha* jokes (with NP punchlines).

Hindi	**English tranlation**
Dur se dekha to Dharmendra tha, *dur se dekha to Dharmendra tha,* *paas jaakar dekha to bandar tha.*	From afar I saw Dharmendra, from afar I saw Dharmendra, on going closer it turned out to be a monkey.
Dur se dekha to Gabbar Singh ka adda tha, *dur se dekha to Gabbar Singh ka adda tha* *paas jaakar dekha to aapka chadda tha.*[2]	From afar I saw Gabbar Singh's haunt, from afar I saw Gabbar Singh's haunt, on going closer it turned out to be your underpants.

Table 4.4: Examples of Type$_{1(a)}$ *Dur-se-Dekha* jokes (those with NP punchlines).

b) **Jokes with Verb Phrase (VP) punchlines.** The formal structure of these types of jokes is illustrated in Table 4.5, while Table 4.1.2 gives examples for the same.

[2]http://www.shayri.com/forums/showthread.php?t=27592

$Part_1$	:	*Dur se dekha to* $\underline{VP_1}$ *tha,*
$Part_2$	:	*Dur se dekha to* $\underline{VP_1}$ *tha,*
$Part_3$	:	*Paas jaakar dekha to* $\underline{VP_2}$ *tha.*

Table 4.5: The structure $Type_{1(b)}$ *Dur-se-Dekha* jokes (with VP punchlines)

Table 4.6: Examples of $Type_{1(b)}$ *Dur-se-Dekha* jokes (those with VP punchlines).

Hindi	**English translation**
Dur se dekha to ande ubal rahe the, *dur se dekha to ande ubal rahe the,* *paas jaakar dekha to ganje uchal rahe the.*[3]	From afar I saw eggs boiling, from afar I saw eggs boiling, on going closer it turned out to be bald men jumping.
Dur se dekha to hasina baal bana rahi thi, *dur se dekha to hasina baal bana rahi thi,* *paas jaakar dekha to gaay puch hila rahi thi.*[4]	From afar I saw a beautiful girl grooming her hair, from afar I saw a beautiful girl grooming her hair, on going closer it turned out to be cow swinging its tail

$Type_2$: Incongruity between the expectation of the hearer and the punchline

This second type of *Dur-se-Dekha* jokes can have the sysntactic structure as either Type 1(a), or Type 1(b) (illustrated in Table 4.7).

Difference between the two categories lies in how humour is encoded in the joke. $Type_1$ jokes are humorous because of how the punch line contrasts with the subject of the setup in terms of the sentiment. The incongruity between the subject (what something seems to be from far away) versus the punchline (what it actually turns out to be), induces surprise and amusement in the listener. The use of celebrity names further makes a joke funnier. On the other hand, for this type ($Type_2$) of jokes, humour arises out of the incongruity between the expectation of the hearer on hearing the first two parts of the jokes, and the unconventionality of the punchline. The listener expects a conventional punchline, something

[3] http://www.shayri.com/forums/showthread.php?t=27592
[4] http://desipoetry.com/door-se-dekha-series/

$Part_1$	:	*Dur se dekha to* $\underline{NP_1/VP_1}$ *tha,*
$Part_2$	:	*Dur se dekha to* $\underline{NP_1/VP_1}$ *tha,*
$Part_3$	:	*Paas jaakar dekha to* $\underline{NP_2/VP_2}$ *tha.*

Table 4.7: Structure of $Type_2$ *Dur-se-Dekha* jokes

conflicting the subject, what he gets instead is an affirmation of the subject, or maybe a consequence of the subject in the real world (See Table 4.8 for examples). This is not unlike the way humour is encoded in some shaggy dog jokes, “a nonsensical joke that employs in the punchline a psychological non sequitur ... to trick the listener who expects conventional wit or humour" [17].

Hindi	**English translation**
Dur se dekha to kuch nahi dikha, *dur se dekha to kuch nahi dikha,* *paas jaakar dekha to kuch tha hi nahi*[5].	From afar I could see nothing,, from afar I could see nothing, on going closer it actually turned out to be nothing.
Dur se dekha to baarish ho rahi thi, *dur se dekha to baarish ho rahi thi,* *paas gaya to bheeg gaya*[6].	From afar I saw that it was raining, from afar I saw that it was raining, on going closer I got drenched.

Table 4.8: Examples of $Type_2$ *Dur-se-Dekha* jokes

4.1.3 Our Model

We build a system that generates the $Type_1$ jokes mentioned in the previous section. We deal with only $Type_1$ jokes here because the incongruity present in them can be modelled linguistically. Our protoype system generates only the basic form of Type 1(a) jokes, where the noun phrase consists of a single noun. We approach this as a template filling task and propose a joke generation process that has the following four steps:

$Step_1$: Template selection
$Step_2$: Setup Formation
$Step_3$: Punchline Formation
$Step_4$: Compilation

[5]http://desipoetry.com/door-se-dekha-series/
[6]http://www.shayri.com/forums/showthread.php?t=27592

These steps are explained below.

Step 1: Template Selection

We create a collection of three templates for a *Dur-se-Dekha* joke. All three templates have the same basic structure as described in Table 4.3, but vary in terms of auxiliary verbs, postpositions etc. These varied templates are used to naturalize the final jokes, and as a small measure against joke fatigue. In this step of the algorithm, we randomly pick one of the three templates to fill in to create a joke.

Step 2: Setup Formation

We consider two opposing semantic categories - `HUMAN` and `NON-HUMAN`, and manually compile a lexicon of words from both these categories. For `HUMAN`, we use a list of popular celebrities names, both male and female, from different domains - actors, politicians, players as well as popular fictional characters. For the second category (`NON-HUMAN`), we pick some generic adjectives, and words (mostly nouns) from the Hindi language that would have a negative sentiment when associated with a human being, for example, names of animals, or vegetables. For the form of $\text{Type}_{1(a)}$ jokes we are working on, one word is picked randomly from this lexicon. This forms the setup, the subject for our joke.

Step 3: Punchline Formation

We select a single word from the lexicon as our punchline. This selection is done following the three constraints explained below:

1. `CATEGORY` constraint: The semantic category (`HUMAN` or `NON-HUMAN`) of the punchline has to be oposite to that of the setup.

2. `GENDER` Constraint: As even all non-human things in Hindi have a gender associated with them, we ensure that the punchline has the same gender as the setup.

3. `FORM` constraint: A typical characteristic of this type of jokes is that the subject of the setup and the punchline are lexically similar, giving a poetic effect to the joke. So, we first look for a word which rhymes with the setup to act as the punchline. If a rhyming word is not found in the lexicon, we use Levenshtein distance[7] [47] or orthographic distance to find a phonetically similar word which is then used as the punchline.

[7]Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.

Hindi	English translation
Dur se dekha to mowgli tha, *dur se dekha to mowgli tha,* *paas jaakr dekha to chimpanzi tha.*	From afar I saw mowgli, from afar I saw mowgli, on going near I saw it was a chimpanzee.
Dur se dekha to mussadi tha, *dur se dekha to mussadi tha,* *paas jaakr dekha to fissadi nikla.*	From afar I saw Mr. Mussadi, from afar I saw Mr.Musadi, on going near he turned out to be a loser.

Table 4.9: Examples of *Dur-se-Dekha* generated by our system.

Step 4: Compilation

We fill in the selected setup and the punchline into the template chosen in Step 1. Next, we inflect the template according to the genders of the setup and the punchline, to ensure subject verb agreement. This results in our final joke.

Given in Table 4.9 are examples of jokes generated by our system using the above mentioned algorithm.

4.1.4 Evaluation and Results

We evaluate our system in two parts. We first draw a comparison between human created jokes available and the ones generated by our system to see how they differ in naturalness. Second, we vary our constraints to see how that leads to changes in humour perception of the resultant jokes.

Since, in our sample set, we had only 5 instances of the subtype of *Dur se Dekha* jokes that we are working on, we took all of those 5 and an equivalent number of jokes generated by our system and put them in a survey in a random order. We then asked 15 participants to rate how natural each joke felt to them on a scale of 1 to 5 (with 5 being completely natural and 1 being completely unnatural).

The analyses of the responses showed us that the average score of the jokes generated by our system was comparable to the average of our exemplar human made jokes. With jokes from our system having an average score of 3.16/5, our system only marginally underperforms the human jokes with an average score of 3.33/5 (Table 4.10). Another interesting observation is that of all the jokes present in the survey, the one with the highest score was one of the computer generated jokes.

In the second part of the evaluation, we look at how the three proposed constraints affect the resultant joke. For this, we created a set of 80 jokes - 20 from the system with all three constraints, and 20 each obtained removing one constraint at a time. We then conducted a survey on this entire set. Each person

Human Created Jokes	Computer Generated Jokes
3.16/5	3.33/5

Table 4.10: Mean naturalness values of human created jokes vs system generated jokes.

Experimental Conditions	**Funniness** (mean $\pm$ std.dev.)
FORM + GENDER + CATEGORY	2.40 ± 0.53
GENDER + CATEGORY	2.11 ± 0.54
FORM + GENDER	1.97 ± 0.49
FORM + CATEGORY	1.68 ± 0.24

Table 4.11: Mean Funniness values, and standard deviations of system generated jokes with varying constraints.

who took the survey got a different randomly generated subset of jokes to rate, with each joke being evaluated by 5 people. In this survey, the evaluators were asked to judge how funny they found each joke to be on a scale of 1 to 5 (with 1 being not funny at all, to 5 being hilarious). The results of this evaluation have been summarized in Table 4.11.

From Table 4.11, we see that people found our jokes only mildly funny. We believe that the simplicity, the lack of a deeper semantic meaning is the reason for this. Varying the constraints, we see that the jokes work best when they adhere to all three constraints. We infer from the results that GENDER constraint contributes the most to humour in our jokes, while FORM constraint contributes the least.

We were unable to perform an inter-rater agreement analysis because each joke was rated by a different set of people. Instead, we chose to include the standard deviations for each set in our analysis.

4.1.5 Future Extensions

Our joke generator prototype is giving encouraging results for the basic form of the jokes but it still has a long way to go to improve domain coverage, and only then can it be considered for practical use.

The lexicon needs to be expanded to add adjectives, adverbs and verbs so that noun phrases and verb phrases can be formed and used as the subject and the punchline. The representation of words in the lexicon can be improved by adding associated features of nouns, in terms of their physical representation in the world, which would help add semantic significance to the results. This will require much more

sophisticated algorithms. This task is especially challenging due to a lack of availability of language resources and tools for Hindi. We will need to develop phrase generators for Hindi for the task. Also, as the punchline should have the sentiment opposite to the subject line, more thought needs to put into what that means for complete phrases.

The lexicon can be updated regularly. In fact, we can make our system such that it automatically picks up trending celebrity names, adjectives and verbs from social media websites and use them as subjects for the joke. This will be instrumental in avoiding joke fatigue and would help our system keep up with the fast changing “in-trend" names.

Also, a much more extensive evaluation should be done for the system when it is capable of generating more complex jokes. Naturalness of the jokes, as well as their funniness needs to be evaluated on a larger scale. Using crowdsourcing for such an evaluation would be a good choice to learn more about the bigger picture.

4.1.6 Summary

Our *Dur se Dekha* joke generator is a first step towards the exploration of humour in Hindi language generation. In this section, we take a focused form of humorous tercets in Hindi - *Dur se Dekha*, and perform an analysis of its structure and humour encoding. We then create a lexicon, and come up with an algorithm to form the various elements of the joke following specific constraints. We see that the jokes generated by our system give encouraging results in terms of naturalness and funniness to serve as a baseline for future work.

4.2 Word Substitutions to Induce Humour in Short Phrases

4.2.1 Introduction

In this section, we consider the task of automatically generating short humorous phrases in Hindi. In particular, we look at the task where humour is induced in a given input phrase by substituting a single word in it using certain constraints. From past attempts made in English [92], we adopt phonetic similarity and tabooness as the basis for substitution. We then propose a more generic sentiment based approach for the word replacement. On performing human evaluation on the resultant phrases, we compare the performance of both these models and note a few interesting observations.

4.2.2 Experiment

We first adopt the approach described in Valitutti et al, 2013 [92] to Hindi and consider this a baseline for our work. We then propose a sentiment based approach for the same problem. We use Hindi movie names as inputs for these systems.

Baseline Implementation

Valitutti et al, 2013 [92] used three constraints (`FORM`, `TABOO`, `CONTEXT`) to perform lexical substitution, which we adopted for Hindi as follows:

- `FORM` constraint: The substitute word should be phonetically similar (having Levenshtein Distance[8] as 1) [47], or rhyming with the word being substituted (ending with same syllable).
- `TABOO` constraint: The substituted word should be a connotational taboo word, where the taboo is inherent in the utterance of the word itself, not necessarily in the topic (for example, profanity words). We collect a of 658 such words for Hindi.
- `CONTEXT` constraint: The substitute and its context should form a coherent compound. We ensure this by checking the presence of this resultant bi-gram in our language model. The language model is a bi-gram language model built using Knesser-Ney smoothing [35], using 100k sentences from Hindi monolingual data from [36].

Table 4.12 shows examples of resultant texts when applying this method on Hindi movie names.

Original Text	**Generated Text**
Saavan Ka Mahina (The Month of Monsoon)	*Chumban Ka Mahina* (The Month of Kiss)
Pyaar Ka Rishta (Relationship of Love)	*Pyaar Ka Kutta* (Dog of Love)

Table 4.12: Example texts from the Baseline model.

Sentiment Based Approach

We propose using strong sentiments instead of taboo words to induce humour. We first do a Part-of-speech tagging (using POS tagger provided by [62]) on the input text and consider content words (nouns, verbs, adjectives, adverbs) as possible candidates for substitution.

[8]Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.

We then find possible substitutions for each candidate word by relaxing the FORM constraint a little to include Levenshtein distance of 2 or less, and looking for words with strong negative sentiment polarity from the Hindi SentiWordnet [75] and the Hindi subjective lexicon [6]. These words are then inflected with the same morphological information as the original word. The CONTEXT constraint is then imposed as in the baseline implementation. Examples of resultant texts using this method are illustrated in Table 4.13.

The idea behind using sentiment as a feature instead of tabooness is twofold. First, the resultant jokes would be more appropriate for a social context, with less chances of offending someone, or being socially unacceptable. Second, sentiment is a more commonly studied phenomenon in NLP with already developed corpora, which makes it easier to adopt this approach to a new language.

Original Text	**Generated Text**
Shadi Ka Maamla (Marriage Affair)	*Bandi Ka Maamla* (Girlfriend Affair)
Biwi Aur Makaan (Wife And House)	*Biwi Aur Thakaan* (Wife And Tiredness)

Table 4.13: Example texts from the our sentiment based model.

4.2.3 Evaluation and Discussion

We take 40 movie titles as input texts and run both the algorithms on these inputs. From all the possible results, we randomly select 40 jokes generated from our model and 20 jokes generated by the implementation of [92] to act as the representative set for human evaluation (Baseline implementation resulted in much fewer jokes, hence we picked only 20 from there). We asked 20 human evaluators (10 male and 10 female, all in 20-23 years age group) to rate 15 jokes each on a 4 point scale (4: It is funny,

Rating	**Baseline(%)**	**Our model(%)**
Funny	33	36
Might be funny	29	22.5
Not sure	13	12.5
Not funny	25	29

Table 4.14: Breakdown of ratings given to jokes from both models

3: Might be funny but not to me or not now, 2: Not sure, 1: Not funny), with each joke getting rated 5 times.

The distribution of the evaluation scores for outputs from each of the models are given in Table 4.14. The mean funniness rating for the baseline model is 2.7, and 2.66 for our model. Thus, both models show comparable performance.

One interesting observation from the evaluations is the difference across gender in appreciation of jokes from the two models. For the baseline model, while 38.4% of the jokes were rated “funny" by males, only 27% jokes were rated “funny" by femals (A gap of 11.4%). But for our model, 34.6% jokes were rated “funny" by males and 37% jokes were rated “funny" by femals (a gap of only 2%). Thus, our model seems to be much more gender neutral.

4.2.4 Summary

We propose a more generic sentiment based architecture for short humorous phrase generation in Hindi. Our model performs comparable to taboo based model proposed by [92] in terms of the percentage of funny jokes and mean funniness ratings, but appears to be more gender neutral and generated texts which are considerably less offensive.

We show that inculcating results from behavioral studies, such as differential humour appreciation based on demographics and humour category, could be an important aspect when developing computational humour systems. We believe that exploring such differences is a worthwhile venture in this domain, Also, improving the density of the “good" jokes that a system outputs should be worked on, by filtering out the bad outputs using humour recognition techniques like ambiguity and semantic relatedness [63].

4.3 Conclusion

In this section we discussed two prototype computational humour generation systems we developed for Hindi. First, we developed a system which generates a simple form of *Dur-se-Dekha* jokes, given a small handmade lexicon. On human evaluation we prove that by enforcing simple constraints (`FORM`, `GENDER` and `CATEGORY`) on the template, the system can generate atleast mildly funny jokes, which can be improved by adding more semantic and syntactic copmlexity. Second, we develop a system to induce humour in a non-humorous Hindi phrase by performing lexical substitution based on sentiment opposition and phonetic similarity. Again, on human evaluation we see that we are able to generate funny instances successfully.

A major challenge when developing a computational humour generation system is its evaluation. Currently, human evauation is the only way to judge such a sysytem. But to come up with an appropriate methodology is challenging. Since humour can be a subjective thing, and its perception heavily influenced by context, or the mood of the hearer, we recognise that the method we used - asking our evaluators to rate jokes on a web platform one after the other may not be the best way of getting accurate perception. It can be argued that it too serious a way to judge something like humour. But, for lack of a better method, this is the approach we take to show that our prototypes work.

Chapter 5

Conclusion

In this thesis, we present, to the best of our knowledge, a first ever work on computational humour for the Hindi language. We explored both subfields in computational humour - Humour Understanding and Humour Generation, and built prototypes for specific applications.

For the problem of computational understanding of humour in Hindi, we considered code-mixed puns, studied the structure and variations in such puns. We classify them in intra-sential code-mixed and intra-word code-mixed puns and built a system which understands Hindi-English intra-sentential code-mixed puns. Our developed system is able to recover 67% of the puns from our test data. We also identify several limitations of our system, such as the inability to deal with unusual and creative language use. This work is also a first work looking into code-mixed humour.

For the problem of computational generation of humour in Hindi, we considered two tasks. First, we built a prototype system which automatically generates a simple form of *Dur-se-Dekha* jokes using a small lexicon and handcrafted rules. Second, we came up with an algorithm which would induce humour in a short non-humorous text by performing single-word substitutions on the basis of phonetic similarity and sentiment opposition. We performed human evaluation on the results of both, and observed that we were able to successfully generate funny instances in both these cases. We recognise the limitation that our humour generation systems lack a comprehensive evaluation, and observe the need for more research and thought into judging how well a creative language generation system performs in any domain.

There are a lot of avenues yet to be explored before we have a computational humour model which can be integrated into real world AI systems. While we have only worked with structured form of humour, a lot of real world funny utterances are unstructured and this needs to be dealt with. The side effects of bilingualism and multilinguism (code-mixing and code-switching) which are becoming increasingly popular phenomenon with rapid globalisation all over the world add so much variety in humour usage that these cannot be ignored when building a computational humour system. There is a need for a more robust theory, novel and creative methods to evaluate the developed systems, and to consider human behaviour and user personality when designing such systems.

“Endeavouring to develop computational models of humour is a worthwhile enterprise both for artificial intelligence and for those interested in humour, but we are starting from a very meagre foundation, and the challenges are significant."

— Graeme Ritchie

Related Publications

1. **Automatic Generation of Jokes in Hindi,** Srishti Aggarwal and Radhika Mamidi, *Student Research Workshop. Association of Computational Linguistics (ACL), 2017*

2. **Sentiment Based Word Substitutions to Generate Humorous Phrases in Hindi (Extended Abstract),** Srishti Aggarwal, Kritik Mathur and Radhika Mamidi, *Widening Natural Language Processing (WiNLP) Workshop. North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), 2018*

3. **Automatic Target Recovery of Hindi-English Code-Mixed Puns,** Srishti Aggarwal, Kritik Mathur and Radhika Mamidi, *Humanizing Artificial Intelligence (HAI) Workshop. International Joint Conference on Artificial Intelligence (IJCAI), 2018*

Bibliography

[1] D. Aarons. *Puns and Tacit Linguistic Knowledge*, chapter chapter7. Routledge, 2017.

[2] S. Attardo. *Humorous texts: A semantic and pragmatic analysis*, volume 6. Walter de Gruyter, 2001.

[3] S. Attardo and J.-C. Chabanne. Jokes as a text type. *Humor: International Journal of Humor Research*, 1992.

[4] S. Attardo and V. Raskin. Script theory revis (it) ed: Joke similarity and joke representation model. *Humor-International Journal of Humor Research*, 4(3-4):293–348, 1991.

[5] P. Auer. *Bilingual conversation*. John Benjamins Publishing, 1984.

[6] A. Bakliwal, P. Arora, and V. Varma. Hindi subjective lexicon: A lexical resource for hindi adjective polarity classification. In *In Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), Chair*, 2012.

[7] J. A. Banas, N. Dunbar, D. Rodriguez, and S.-J. Liu. A review of humor in educational settings: Four decades of research. *Communication Education*, 60(1):115–144, 2011.

[8] F. Barbieri and H. Saggion. Automatic detection of irony and humour in twitter. In *ICCC*, pages 155–162, 2014.

[9] U. Barman, A. Das, J. Wagner, and J. Foster. Code-mixing: A challenge for language identification in the language of social media. In *In Proceedings of the First Workshop on Computational Approaches to Code-Switching*, 2014.

[10] A. Bharati, V. Chaitanya, R. Sangal, and K. Ramakrishnamacharyulu. *Natural language processing: a Paninian perspective*. Prentice-Hall of India New Delhi, 1995.

[11] K. Binsted. Machine humour: An implemented model of puns. 1996.

[12] K. Binsted et al. Using humour to make natural language interfaces more friendly. In *Proceedings of the AI, ALife and Entertainment Workshop, Intern. Joint Conf. on Artificial Intelligence*, 1995.

[13] K. Binsted, A. Nijholt, O. Stock, C. Strapparava, G. Ritchie, R. Manurung, H. Pain, A. Waller, and D. O'Mara. Computational humor. *IEEE Intelligent Systems*, 21(2):59–69, 2006.

[14] K. Binsted and G. Ritchie. An implemented model of punning riddles. Technical report, University of Edinburgh, Department of Artificial Intelligence, 1994.

[15] K. Binsted and G. Ritchie. Computational rules for generating punning riddles. *HUMOR-International Journal of Humor Research*, 10(1):25–76, 1997.

[16] S. Bird, E. Klein, and E. Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, 2009.

[17] J. H. Brunvand. A classification for shaggy dog stories. *The Journal of American Folklore*, 76(299):42–68, 1963.

[18] C. Bucaria. Lexical and syntactic ambiguity as a source of humor: The case of newspaper headlines. *Humor*, 17(3):279–310, 2004.

[19] S. Doogan, A. Ghosh, H. Chen, and T. Veale. Idiom savant at semeval-2017 task 7: Detection and interpretation of english puns. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 103–108, 2017.

[20] M. Dynel. Beyond a joke: Types of conversational humour. *Language and Linguistics Compass*, 3(5):1284–1299, 2009.

[21] P.-J. Farrugia. Tts pre-processing issues for mixed language support. In *Proceedings of CSAW'04*, page 36, 2004.

[22] S. Freud and P. Gay. *Jokes and their relation to the unconscious*. WW Norton & Company, 1960.

[23] J. Gafaranga and M.-C. Torras. Interactional otherness: Towards a redefinition of codeswitching. *International Journal of Bilingualism*, 6(1):1–22, 2002.

[24] T. Gottron and N. Lipka. A comparison of language identification approaches on short, query-style texts. In *European Conference on Information Retrieval*, pages 611–614. Springer, 2010.

[25] M. K. Gray, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, and S. Pinker. Quantitative analysis of culture using millions of digitized books. *science*, 1199644(176):331, 2011.

[26] H. P. Grice. Logic and conversation. *1975*, pages 41–58, 1975.

[27] K. Gupta, M. Choudhury, and K. Bali. Mining hindi-english transliteration pairs from online hindi lyrics. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, 2012.

[28] C. Hempelmann. *Paronomasic puns: Target recoverability towards automatic generation*. PhD thesis, Indiana University-Purdue University Indianapolis, 2003.

[29] N. Hossain, J. Krumm, L. Vanderwende, E. Horvitz, and H. Kautz. Filling the blanks (hint: plural noun) for mad libs humor. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 638–647, 2017.

[30] L.-F. Hurtado, E. Segarra, F. Pla, P. Carrasco, and J.-A. González. Elirf-upv at semeval-2017 task 7: Pun detection and interpretation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 440–443, 2017.

[31] V. Indurthi and S. R. Oota. Fermi at semeval-2017 task 7: Detection and interpretation of homographic puns in english language. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 457–460, 2017.

[32] A. Jaech, R. Koncel-Kedziorski, and M. Ostendorf. Phonological pun-derstanding. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 654–663, 2016.

[33] A. Kilgarriff. Wordnet: An electronic lexical database, 2000.

[34] B. King and S. Abney. Labeling the languages of words in mixed-language documents using weakly supervised methods. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1110–1119, 2013.

[35] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184 vol.1, May 1995.

[36] A. Kunchukuttan, P. Mehta, and P. Bhattacharyya. The iit bombay english-hindi parallel corpus. *arXiv preprint arXiv:1710.02855*, 2017.

[37] M. Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation*, pages 24–26. ACM, 1986.

[38] G. Lessard and M. Levison. Computational modelling of linguistic humour: Tom swifties. In *Selected Papers from the 1992 Association for Literary and Linguistic Computing (ALLC) and the Association for Computers and the Humanities (ACH) Joint Annual Conference*, pages 175–178. Oxford University Press, 1992.

[39] D. Loehr. An integration of a pun generator with a natural language robot. In *Proc. International Workshop on Computational Humor, 1996*, pages 161–172. University of Twente, 1996.

[40] R. Mamidi. Context and Humor: Understanding Amul advertisements of India. *ArXiv e-prints*, Apr. 2018.

[41] R. A. Martin. Approaches to the sense of humor: A historical review. *The sense of humor: Explorations of a personality characteristic*, 15, 1998.

[42] R. A. Martin. The psychology of humor: An integrative approach. 2007.

[43] C. J. McDonough. Mnemonic string generator: Software to aid memory of random passwords. Technical report, CERIAS Technical report, West Lafayette, IN, 2001.

[44] J. Mckay. Generation of idiom-based witticisms to aid second language learning. In *The April Fools' Day Workshop on Computational Humor, 2002*, pages 77–87. University of Twente, 2002.

[45] R. McKeon et al. *The basic works of Aristotle*. Modern Library, 2009.

[46] R. Milalcea and C. Strapparava. Learning to laugh (automatically): Computational models for humor recognition. *Computational Intelligence*, 22(2), 2006.

[47] F. P. Miller, A. F. Vandome, and J. McBrewster. *Levenshtein Distance: Information Theory, Computer Science, String (Computer Science), String Metric, Damerau?Levenshtein Distance, Spell Checker, Hamming Distance*. Alpha Press, 2009.

[48] T. Miller and I. Gurevych. Automatic disambiguation of english puns. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 719–729, 2015.

[49] T. Miller, C. Hempelmann, and I. Gurevych. Semeval-2017 task 7: Detection and interpretation of english puns. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 58–68, 2017.

[50] D. H. Monro. Theories of humor. *Writing and reading across the curriculum*, pages 349–355, 1988.

[51] J. Morreall. Philosophy of humor. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2016 edition, 2016.

[52] P. C. Muysken. Code-switching and grammatical theory. 1995.

[53] D. Nguyen and A. S. Doğruöz. Word level language identification in online multilingual communication. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 857–862, 2013.

[54] A. Nijholt. Embodied agents: A new impetus to humor research. In *The April Fools Day Workshop on Computational Humour*, volume 20, pages 101–111. In: Proc. Twente Workshop on Language Technology, 2002.

[55] D. Oele and K. Evang. Buzzsaw at semeval-2017 task 7: Global vs. local context for interpreting and locating homographic english puns with sense embeddings. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 444–448, 2017.

[56] T. Pedersen. Duluth at semeval-2017 task 7: Puns upon a midnight dreary, lexical semantics for the weak and weary. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 416–420, 2017.

[57] S. Petrović and D. Matthews. Unsupervised joke generation from big data. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 228–232, 2013.

[58] Plato. *The Collected Dialogues, including the Letters*. Pantheon Books, 1961.

[59] V. Raskin. *Semantic Mechanisms of Humor*, volume 24. Springer Science & Business Media, 1984.

[60] V. Raskin. Computer implementation of the general theory of verbal humor. In *Automatic Interpretation and Generation of Verbal Humor. International Workshop on Computational Humor, IWCH'96. Twente Workshop on Language Technology, TWLT*, volume 12, pages 9–19, 1996.

[61] V. Raskin and S. Attardo. Non-literalness and non-bona-fîde in language: An approach to formal and computational treatments of humor. *Pragmatics & Cognition*, 2(1):31–69, 1994.

[62] S. Reddy and S. Sharoff. Cross language pos taggers (and other tools) for indian languages: An experiment with kannada using telugu resources. In *Proceedings of the Fifth International Workshop On Cross Lingual Information Access*, pages 11–19, Chiang Mai, Thailand, November 2011. Asian Federation of Natural Language Processing.

[63] A. Reyes, P. Rosso, and D. Buscaldi. From humor recognition to irony detection: The figurative language of social media. *Data & Knowledge Engineering*, 74:1–12, 2012.

[64] G. Ritchie. Developing the incongruity-resolution theory. *Institute for Communicating and Collaborative Systems*, 1999.

[65] G. Ritchie. Describing verbally expressed humour. *Institute for Communicating and Collaborative Systems*, 2000.

[66] G. Ritchie. Current directions in computational humour. *Artificial Intelligence Review*, 16(2):119–135, 2001.

[67] G. Ritchie. The jape riddle generator: technical specification. *Institute for Communicating and Collaborative Systems*, 2003.

[68] G. Ritchie. *The Linguistic Analysis of Jokes*. Routledge Studies in Linguistics. Taylor & Francis, 2004.

[69] G. Ritchie. Can computers create humor? *AI Magazine*, 30(3):71, 2009.

[70] M. Rosner and P.-J. Farrugia. A tagging algorithm for mixed language identification in a noisy domain. In *Eighth Annual Conference of the International Speech Communication Association*, 2007.

[71] W. Ruch. The perception of humor. In *Emotions, qualia, and consciousness*, pages 410–425. World Scientific, 2001.

[72] W. Ruch, S. Attardo, and V. Raskin. Toward an empirical verification of the general theory of verbal humor. *Humor-International Journal of Humor Research*, 6(2):123–136, 1993.

[73] A. Salvatore. Linguistic theories of humor. *Berlin/New York*, 1994.

[74] T. Scheel. Definitions, theories, and measurement of humor. In *Humor at Work in Teams, Leadership, Negotiations, Learning and Health*, pages 9–29. Springer, 2017.

[75] R. Sharma and P. Bhattacharyya. A sentiment analyzer for hindi using hindi senti lexicon. In *Proceedings of the 11th International Conference on Natural Language Processing*, pages 150–155, 2014.

[76] J. Sherzer. Puns and jokes. *Handbook of discourse analysis*, 3:213–221, 1985.

[77] J. Sherzer. Verbal play. *International encyclopedia of linguistics*, 4:220–222, 1992.

[78] J. Simpson, E. S. Weiner, et al. Oxford english dictionary online. *Oxford: Clarendon Press. Retrieved March*, 6:2008, 1989.

[79] W. Sobkowiak. *Metaphonology of English paronomasic puns*. Bamberger Beitrage Zur Englischen Sprachwissenschaft, University of Bamberg Studies in English Linguistics. Peter Lang Publishing, Incorporated, 1991.

[80] O. Stock and C. Strapparava. Getting serious about the development of computational humor. In *IJCAI*, volume 3, pages 59–64, 2003.

[81] O. Stock and C. Strapparava. Hahacronym: Humorous agents for humorous acronyms. *Humor*, 16(3):297–314, 2003.

[82] O. Stock and C. Strapparava. Hahacronym: A computational humor system. In *Proceedings of the ACL 2005 on Interactive Poster and Demonstration Sessions*, ACLdemo '05, pages 113–116, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[83] O. Stock and C. Strapparava. Laughing with hahacronym, a computational humor system. In *21st conference of American Association for Artificial Intelligence (AAAI-06)*, pages 1675–1678. AAAI, 2006.

[84] J. Suls, A. Chapman, and H. Foot. Cognitive and disparagement theories of humor: A theoretical and empirical synthesis. *Chapman and Foot*, pages 41–5, 1977.

[85] J. M. Suls. A two-stage model for the appreciation of jokes and cartoons: An information-processing analysis. *The psychology of humor: Theoretical perspectives and empirical issues*, 1:81–100, 1972.

[86] K. Tanaka. The pun in advertising: A pragmatic approach. *Lingua*, 87(1):91 – 102, 1992.

[87] J. M. Taylor. *Computational recognition of humor in a focused domain*. PhD thesis, University of Cincinnati, 2004.

[88] J. M. Taylor and L. J. Mazlack. Computationally recognizing wordplay in jokes. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 26, 2004.

[89] A. Vadehra. Uwav at semeval-2017 task 7: Automated feature-based system for locating puns. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 449–452, 2017.

[90] A. Valitutti. How many jokes are really funny? towards a new approach to the evaluation. In *Human-Machine Interaction in Translation: Proceedings of the 8th International NLPCS Workshop*, volume 41, page 189. Samfundslitteratur, 2011.

[91] A. Valitutti, A. Doucet, J. M. Toivanen, and H. Toivonen. Computational generation and dissection of lexical replacement humor. *Natural Language Engineering*, 22(5):727–749, 2016.

[92] A. Valitutti, A. Doucet, H. Toivonen, J. M. Toivanen, et al. Let everything turn well in your wife. In *The 51st Annual Meeting of the Association for Computational Linguistics (ACL) Volume 2: Short Papers*, 2013.

[93] O. Vechtomova. Uwaterloo at semeval-2017 task 7: Locating the pun using syntactic characteristics and corpus-based metrics. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 421–425, 2017.

[94] C. Venour. The computational generation of a class of puns. *Master's thesis, Queen's University*, 1999.

[95] Y. Xiu, M. Lan, and Y. Wu. Ecnu at semeval-2017 task 7: Using supervised and unsupervised methods to detect and locate english puns. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 453–456, 2017.

[96] D. Yang, A. Lavie, C. Dyer, and E. Hovy. Humor recognition and humor anchor extraction. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2367–2376, 2015.

[97] T. Yokogawa. Japanese pun analyzer using articulation similarities. In *Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on*, volume 2, pages 1114–1119. IEEE, 2002.

[98] R. Zhang and N. Liu. Recognizing humor on twitter. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 889–898. ACM, 2014.

[99] A. Zwicky and E. Zwicky. Imperfect puns, markedness, and phonological similarity: With fronds like these, who needs anemones. *Folia Linguistica*, 20(3-4):493–503, 1986.