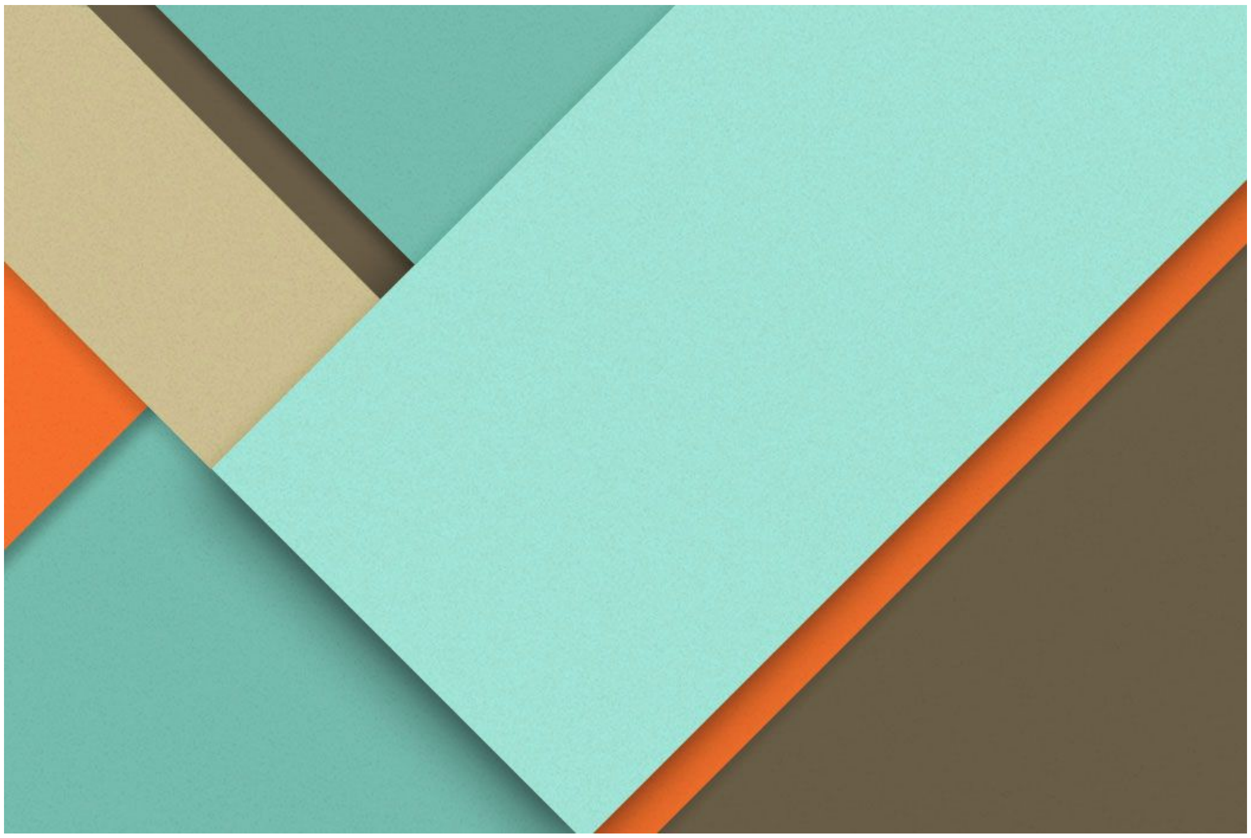


Automatic Target Recovery for Hindi-English Code Mixed Puns

Anshul Padhi - 2018114013

Akshat Gahoi - 2018114012

Overview

Puns

A pun is a play on words that produces a humorous effect by using a word that suggests two or more meanings, or by exploiting similar sounding words that have different meanings.

Humorous effects created by puns depend upon the ambiguities the words entail. These ambiguities arise mostly in homophones and homonyms. For instance, in the sentence, “A happy life depends on a liver,” the word *liver* can refer to the bodily organ, or simply a person who lives. Similarly, in the saying “Atheism is a non-prophet institution,” the word “prophet” is used instead of “profit” to produce a humorous effect.

Code Mixing

Code Mixing is the mixing of two or more languages or language varieties in speech. In this linguistic clauses from a language are inserted into sentences from another language. For example, in the sentence “mujhe mangoes chahiye” mangoes do not belong to Hindi. The word is a pat of the English vocabulary.

Code Mixed Puns

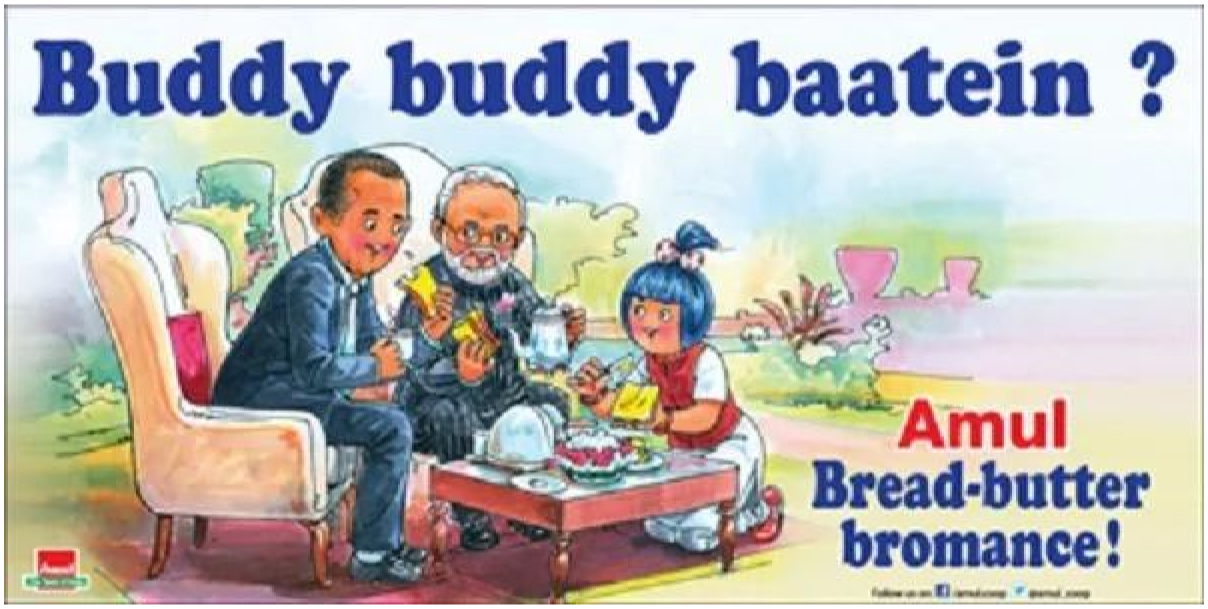
This is a pun where a word from another language is used to create the ambiguity in meaning in a sentence. For example, “Face bhookh with Amul, Mark”. In this ‘face bhookh’ could be replaced with ‘facebook’. Here ‘bhookh’ is a Hindi word and therefore this is an example of a code mixed pun.

Goal

Identify the pun word in a pun and find a suitable replacement to retrieve the secondary/hidden meaning of the pun.

Specifications

We will be given a string which will be the pun and we need to identify the possible word which is causing the ambiguity. This will most probably be the mixed word. Once we find the word, we have to find a word for replacing. This word has to follow the guidelines of a pun. It must make sense semantically, its replacement must form a syntactically acceptable utterance and it must be phonetically similar to the candidate word. Once we find this word we replace it with the original word to get the hidden meaning. For example,



Here 'buddy' could be replaced with 'badi' to form "badi badi baatein". Here buddy is the code mixed word and it is replaced by the hindi word 'badi'.

Approach

Our program is divided into 4 steps-

1.Language Identification

2.Phonetic Minimization

3.Language Modelling

Language Identification

Here we are given a pun in the form of a string and we have to separate the words into 2 lists, English and Hindi by passing it into the language identification part word by word.

To identify English words we use a python module called Enchant which uses its own searching algorithm to look through the dictionary. If the word is not found in the English word dictionary it is considered to be non English.

To identify Hindi words we have created a dictionary of Hindi words using a text file of about 30000 sentences. We first transliterate each word using a python module called indic. We then pass this transliterated word through the created dictionary. If it is found it is added to the Hindi list.

If a word is neither English or Hindi it is added to both lists.

Once the lists are added we have to decide whether an English item is present in a Hindi utterance or the other way around. To know this we check the count of the two lists. If count of English is more, we assume that a Hindi item has been inserted into an English sentence. We do the opposite if else.

So now all the words of the language with lower count become candidate words for the pun.

Phonetic Minimization

Once we have a list of the candidate words for the puns we go through them one by one and try to find a word in the other language phonetically similar to the candidate word as in puns the two words representing different meanings are generally similar phonetically.

To find such a word we use a two layered approach. First we find all the similar words using soundex. Then we pass these words through the levenshtein distance test to further filter out the words. The words we get finally are the possible replacements

Soundex

In this we group similar sounding letters like s, z together and give them the same score. We come up with a score for the whole letter based on the individual letters. We select only those words with the same score.

Levenshtein Distance

Levenshtein distance is similar to edit distance. Here we create a dp table which stores the minimum distance between str1[i] and str2[j] for all i, j. We find the levenshtein distance for the latter i, j using the given entries. The final entry will be the edit distance between str1[n1] and str2[n2]. We only consider a word as similar when it has an edit distance less than 5.

The list of words which passes through the two tests are the set of possible replacements. We pass these through the language model to check whether the replacement forms an acceptable occurrence.

Language Modelling

Once we have the set of possible replacement words we need to check whether the replacement forms an acceptable utterance. To do this we need a language model which can give us a score of how likely a sentence is to occur in a particular language. There are two approaches to it.

Rule Based Language Modelling

Here we have a set of syntactic rules (Context free Grammar) that a sentence has to follow to be accepted as a correct utterance. A sentence is supposed to be parsed and is given its appropriate POS tags. It's then checked for all the rules and only if it follows all the rules is it accepted as a part of the language.

We came up with a set of rules for English and Hindi respectively

English Rules:

SENTENCE

S → NP VP(Declarative)

S → VP(Imperative)

S → Aux NP VP(Yes/No question)

S → Wh-NP VP(Wh type question)

NP → NP and NP

VP → VP and VP

S → S and S

NOUN PHRASE

NP →(Det)(Card)(Ord)(Quant)(AP) Nominal

VERB PHRASE

Possible frames for verb decomposition
Depending on verb

1.VP -> VP NP NP
2.VP -> VP NP
3.VP -> VP PPfrom PPto
4.VP -> VP NP PPwith
5.VP -> VP S

Hindi Rules:

```
S -> S CC S

S -> (NP) (NEG) (VP)

NP -> PSP (JJ) NN

NP -> QF (JJ) NN

VP -> VP VAUX

JJP -> JJ (NEG)

VP -> RBP NP VP

VP -> PSP NP (NEG) VP
```

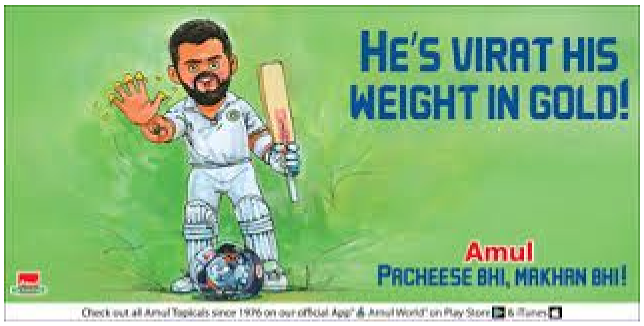
Unfortunately implementing these rules were too tough so we went with a statistical approach.

Statistical Language Modelling

We create a language model based by creating a table which stores the count of the co-occurrences of two words. We tried implementing a linear combination of a unigram, bigram and trigram model but due to insufficiency in data our trigram model was giving us very poor results so we stuck to the bigram model.

In the bigram model we first create a table storing the count of co-occurrences of two words. We convert this count to a probability score by simply dividing by the total

When we pass a string with the appropriate replacement we check the score of the replacing word with its neighbors. If it's above a certain cutoff we consider it acceptable. The new string formed is considered to be a possible interpretation of the pun. For example



In this if we input "He is Virat his weight in gold" we get-

English Words : ['He', 'is', 'his', 'weight', 'in', 'gold']

Hindi Words : ['Virat']

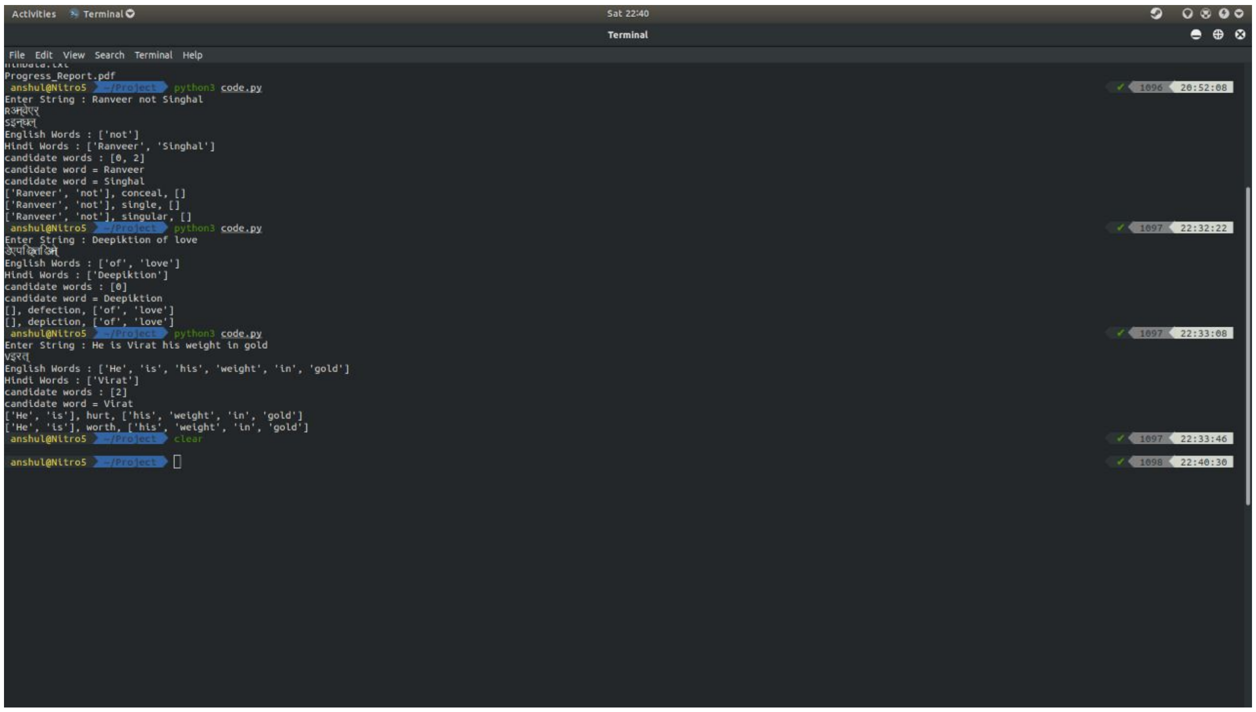
candidate word = Virat

['He', 'is'], hurt, ['his', 'weight', 'in', 'gold']

<u>['He', 'is'], worth, ['his', 'weight', 'in', 'gold']</u>

The underlined line is the possible interpretation of the pun. The first interpretation is an example of why the bigram model is not very accurate.

Results



```
Progress_Report.pdf
 anshul@Nitro5  ~/Project  python3 code.py                                    ✓  1096  20:52:08
Enter String : Ranveer not Singhal
Rअन्वेएर्
Sइन्घल्
English Words : ['not']
Hindi Words : ['Ranveer', 'Singhal']
candidate words : [0, 2]
candidate word = Ranveer
candidate word = Singhal
['Ranveer', 'not'], conceal, []
['Ranveer', 'not'], single, []
['Ranveer', 'not'], singular, []
 anshul@Nitro5  ~/Project  python3 code.py                                    ✓  1097  22:32:22
Enter String : Deepiktion of love
डेएपिक्तिओन्
English Words : ['of', 'love']
Hindi Words : ['Deepiktion']
candidate words : [0]
candidate word = Deepiktion
[], defection, ['of', 'love']
[], depiction, ['of', 'love']
 anshul@Nitro5  ~/Project  python3 code.py                                    ✓  1097  22:33:08
Enter String : He is Virat his weight in gold
Vइरत्
English Words : ['He', 'is', 'his', 'weight', 'in', 'gold']
Hindi Words : ['Virat']
candidate words : [2]
candidate word = Virat
['He', 'is'], hurt, ['his', 'weight', 'in', 'gold']
['He', 'is'], worth, ['his', 'weight', 'in', 'gold']
 anshul@Nitro5  ~/Project  clear                                              ✓  1097  22:33:46

 anshul@Nitro5  ~/Project                                                     ✓  1098  22:40:30
```

Further Improvements

1.We would like to build better transliteration pairs as the current module does not give reliable results

2. We would like to improve upon Levenshtein Distance to make it more adaptable to the length of the word.

3.We would like to incorporate the grammar rules for better accuracy

4.We would like to include a semantic analysis for better results.