Pun and Multiple Meaning Detection

Chaim Schendowich
300307659
chysch@gmail.com

Anna Sokolov
306616178
nannasokolov@gmail.com

GitHub repository:
github.com/chysch/Phonetic-Pun-ishment

Abstract

Creative speakers tend occasionally to instill in their words puns and multiple meanings. When the multiple meaning arises from the sound of the words, namely is caused by saying sounds that can be interpreted as two or more different combinations of words which all make sense in the given context, it is called a homophonic pun. The intention of this project is detection of such puns in given sentences if they exist and indication if not.

1 Introduction

Joke detection is one of the harder problems in Natural Language Understanding. This is because a joke sometimes relies on ambiguity, exaggeration of reality or not having a totally reasonable meaning. These concepts are harder for a machine to recognize than for the human mind, since there is a large amount of data to be considered in order to be able to distinguish between what is a reasonable description of reality and what is not.

Puns in particular have various characteristics that complicate their detection. A pun can be caused by implications derived from alternate definitions of a word - a type of pun referred to as homographic. Understanding and detecting that type of pun require knowledge of word meanings and ways to undertand situations and contexts. A pun can also be caused by mutiplicity derived from the sound of the words said. That type of pun we call homophonic. Apart from the afore-mentioned capabilities, this type of pun requires also phonetic analysis abilities.

Since jokes and puns are frequently used in casual discourse and pose a big challenge to those trying to understand the intention of writers or speakers, the subject has been subject to a large amount of study. This area is especially interesting because lack of humor is one of the most notorious traits of robots and an obstacle researchers interested in humanifying robots will have to overcome.

This project attempts to tackle the problem of intelligible homophonic multiplicity detection. We aim to detect sentences that contain homophonic puns based on their phonetic structure and discover alternative sentences that can be derived from them.

2 Method

In order to find homophonic multiplicity in a sentence we decided to break the project into four stages described below. The stages are in a logical order and are modular and the basic logic is relevant for many languages. There are various features, however, that work only in English, so some of the modules will need to be changed to allow other languages.

We padded the logical core with a variety of hyperparameters controlled by a plain-text rule file which can be easily created and modified by the user. These will also be described below. The general idea of most of them is weakening the accuracy of the phonetic match to allow more positive results.

2.1 Training

The first stage is preparing a phonetic dictionary. We called this stage Training although it doesn't contain training in the traditional sense (Usually "Training" relates to creating data via statistical analysis). Instead, we take ready clean-text phonetic dictionaries in the CMU dictionary format and prepare our own set of two phonetic dictionaries: A Word-Phoneme dictionary (called Fore) and a Phoneme-Word dictionary (called Back).

This stage is very simple and technical since it only requires reading a set of files and outputting the relevant information into two files. There are however various improvements which can be achieved by applying small modifications to the information as shall be explained in the hyperparameter section.

A example input could have the line "ACTIVISTS(4) AE K T IH V IH S". In the output, Fore will contain "ACTIVISTS AE K T IH V IH S" while Back will contain "AE K T IH V IH S ACTIVISTS".

2.2 Analyzing

The second stage is translating the plain-text test sentences into phonemes. In this stage we simply translate the sentences word for word into phonemes. Since there are words with more than one way of translating, the output in this stage is a list of matches for each sentence.

Since puns don't always cover all the words of the sentence (they usually don't), we added functionality to control the amount translated for shorter running time in the next stages.

An example test sentence could be "HE IS A CREATURE OF FEW WORDS". The output for this sentence will be "HE IS A CREATURE OF FEW WORDS" followed by the number of matches followed by various matches like "HH IY IH Z AH K R IY CH ER AH V F Y UW W ER D Z".

2.3 Synthesizing

All the preparation done, the next stage is creating a list of new sentences that can be composed out of words that phonetically match the sets of phonemes created in the previous stage.

To achieve this purpose we created a phoneme search tree with words for leaves that the path to each leaf is the phonemes that compose the word. With this tree the new

sentences can easily be created with a simple deterministic recursive function that scans the original phoneme list and traverses the tree to retrieve words. For efficiency reasons we favored using a stack to which we add each relevant sentence beginning iteratively.

Now the example test sentence "HE IS A CREATURE OF FEW WORDS" will be followed by the number of matches followed by various matches like "HE IS ACHE REACH OR OF FEW WORDS".

2.4 Cleaning

This stage is responsible for cleaning the output of the previous stage, since we now have a long list of new sentences that match a given sentence. They may match this sentence phonetically, but their validity both syntactically and semantically should be tested and invalid sentences should be removed from the list.

The output here is similar to the previous stage except the list of matches should be meaningfully shorter.

In order to test whether a sentence is valid semantically and syntactically we use some open-source tools.

2.4.1 Parsing Tree

In order to test whether a sentence is valid grammatically, we use one of two tools. If 'Online Mode' is on, the 'Logon' tool (THE LOGON PROJECT) is used to parse a sentence to its derivation tree. This tool helps us indicate whether a sentence is syntactically valid. If it isn't, it retrieves zero parsing trees for the sentence. If 'Online Mode' is off, we use the NLTK library (NLTK) for python, that performs POS tagging of the sentence, and with its help we know whether the sentence is valid syntactically.

2.4.2 Avoid Uncommonly Used Words

This filters all produced sentences that use valid English words that are unlikely to be used in an everyday sentence, like acronyms, prefixes and the like.

2.4.3 Incorrectness Intended?

Sometimes a sentence may be invalid semantically but this incorrectness is a part of the intended pun. We wish to determine whether this is the case and avoid removing these kind of sentences. This is why we use the Datamuse tool (DATAMUSE API) that helps us determine whether the problematic word of the incorrect sentence is related to any other word in the original sentence.

2.5 Hyper-Parameters

The rule file required by the user can be empty, but usually will contain values for the hyper-parameters controlling the project. Below the hyper-parameters are listed by name with a short description and motivation.

2.5.1 OnlineMode

When 'OnlineMode' is on, the algorithms will use external sources that require active internet connection. This improves the results, but can also extend run-time significantly. Those online resources include the 'Logon' tool mentioned in section 2.4.1, and the 'Datamuse' tool mentioned in section 2.4.3. If the 'OnlineMode' os off, the use of the 'Logon' tool is replaced with the use of NLTK library for Python, and the 'Pun intended' feature from section 2.4.3 is not used.

2.5.2 Replace

In many cases, a word has a number of pronounciations and in some cases two sets of words are similar enough to allow a pun even though their phonetic build is not exactly the same. For example, "A PUN CAN DIE" can be said "A PUNK AND I". The words "PUN CAN" are "P AH N K AH N" phonetically, while the words "PUNK AND" are "P AH NG K AH N D" phonetically. Since the difference between "NG" and "N" are not so great in this case, we found it necessary to make a Replace option available. This option allows substitution of phoneme sets while creating the dictionaries, with the choice given if all matching sets should be replaced or only in the beginning or end of words.

2.5.3 Sticky

Sometimes a situation arises that a phoneme which should be said twice is sounded once or vice versa. The sentence "NO IDEA" as an answer to the question "WHAT DO YOU CALL A BLIND DEER?" (in British accent) can sound like "NO EYED DEER", even though "IDEA" has the phoneme "D" only once in contrast to "EYED DEER" which has it twice. To allow for such cases, we added the Sticky option which allows either repetition of a phoneme or removal of such repetitions (or both).

2.5.4 StickyAddSkip

A problem we encountered while using the Sticky parameter was single-phoneme words. When using the Add option which repeats phonemes, it caused an infinite loop of creation of single-phoneme words like "E". One way to deal with this problem was a hard-coded list of phonemes to skip and therefore not repeat, a solution implemented in the StickyAddSkip parameter. This is relevant in particular for phonemes that one can forsee as impossible to repeat like the phoneme "AW". For example: The combination "HOW OWL" translated as "HH AW AW L" will never be confused with "HOWL" translated as "HH AW L". This is because "AW" is a diphtong, combined of "AH" and "UW" so repetition sounds like "AH UW AH UW".

2.5.5 Threshold

An integer represents the maximal distance allowed between the length of a given sentence (i.e the number of words in the sentence) and the length of a sentence matched to it phonetically. The bigger this threshold, the stricter is the policy, and matches that differ from the original sentence are dissmissed.

2.5.6 WordBound

Like Threshold, this parameter controls the maximum number of words in a pun. An integer is provided which serves as an upper bound to the number of words translated phonetically leaving the rest of the sentence in all matches created identical to the origin.

For the test example "HE IS A CREATURE OF FEW WORDS", activating WordBound with value 3 might produce results like "HE IS A [K R IY CH ER AH V F Y UW] WORDS" in the Analyze stage.

2.5.7 WordNetFilter

While working with the CMU dictionary we discovered that since it is meant for Speech To Text, it contains many words which are not commonly used in spoken English. This parameter uses NLTK to filter out such words. It is obvious, however, that WordNetFilter is only relevant for English.

3 Challenges

The project was subject to many trials and tribulations. Following is a description of a number of central ones.

3.1 Weird Words

The CMU phonetic dictionary was originally made for Speech To Text. This means that it contains many words that are not usually part of the spoken tongue and some sounds

which are not even really words because they tend to be sounded while speaking. Even when these are filtered out, the dictionary is composed of many words that are not commonly used. This poses a problem because much too many phonetic matches can be found with such a rich dictionary while most of them probably won't be viable as possibilities for puns.

3.2 Stuck Infinitely

As mentioned above, while working on the Sticky parameter we had an infinite loop problem. Although the StickyAddSkip solution is sensible for many situations, it did not act as our complete savior in this case. The problem was solved finally when we discovered the actual cause of the problem was single-phoneme words which were repeated indefinitely. To bypass that problem we patched the code with an automatic skip if the phoneme being repeated has a word as its child in the phoneme search-tree.

3.3 Efficient Cleaning

3.3.1 Using Online Tools

The Cleaning stage of our model is responsible for removing the matches that are invalid either semantically or syntactically. This task required some research for external tools to validate the sentences, as well as many attempts to integrate those tools and methods into our model, in order to test their contribution.

The challenges in using an online tool are various. In addition to the large impact on runtime, we are also vulnerable to server errors and to be blocked by servers due to large amount of requests.

3.3.2 Semantic Validation of a Sentence

Semantic validation of a sentence is a challenging task since it requires understanding the meaning of a sentence. To tackle this task we tried a variety of external online tools that can help determine whether a sentence has a valid meaning.

One powerful method that stood out was the Google Search method. A query to Google Search could easily determine whether the sentence is likely to be used (by the amount of results retrieved for it) - and therefore is valid. The obstacle we encountered with this method was the limit on the amount of queries one can execute to Google Search engine, without being blocked by the server.

Other search engines have the same issue, although this may be solved by purchasing a proportionate package for the Search API usage.

3.4 Gold Mining

Creation of a gold file in this context is also no small deal. Throughout the project we discovered that even when we had a particular pun intended, the phonetic structure of a sentence could lead to many possibilities we hadn't considered, many of which given the correct context would work as puns. We created basic gold files according to the possibilities which seemed right to us on occasion, but a complete study cannot be done without a large group of clever and creative people looking over the data and determining which possibilities really cannot be puns.

3.5 Context-Related Puns

Another challenge was the treatment of sentences that could be considered as puns in a specific context only. As our model addressed each sentence as a 'stand alone', we could not take the context of the sentence in consideration, and this added to the challenge of determining whether a given match would work as a pun.

4 Results

Our results are based on several tests we ran and evaluated that contain sentences that homophonically may be interpreted as sentences with a different meaning. We used short sentences, as well as long sentences, and sentences with a variety of different types of puns.

In order to evaluate the results, we used a gold file constructed by a human. The comparison between our parsed file and the gold file consisted of counting the false results (both false negatives and false positives), as well as comparing the length of the matches list retrieved by the test and by the gold file, for each sentence.

The tests conducted a relatively low rate of false negatives, which means that our model is capable of producing homophonic puns. However, there was a large amount of false positives, due to the lack of efficient and reliable way to filter sentences that do not make sense semantically.

4.1 Baseline Comparison

The baseline of our model consisted simply of an empty rule file – a rule file without any hyper-parameters. In essence this means that we ran the test file with all hyper-parameters in their default values.

The result of the baseline compared to our best set of results had a significantly larger amount of false positives, and no difference for the false negative amount.

Other than counting the false positives and false negatives, we evaluate our results by comparing the length of the list of matched sentences, for a given sentence, and consider this ratio as the score for this sentence. Therefore, we aim for this ratio to be as close as can be to the value 1.

The baseline score was 98, while the score for our best set of results was 12.

In addition to the calculations mentioned above, we also calculate an 'Accuracy Score' which is calculated in the following way:

$$100(1-\frac{\frac{num\ of\ false\ negatives}{num\ of\ results\ in\ gold\ file}+\frac{num\ of\ false\ positives}{num\ of\ results\ in\ test\ file}}{2})$$

The accuracy score for the baseline was 42, while the accuracy score for our best set of results was 45.

One of the conclusions from these results was that the most effective parameter is WordBound because it keeps the number of matches under control.

5 Conclusions

We set ourselves a very ambitious task. Even though the synthesis of homophonic matches in itself is no great challenge, creating the matches so that they cover even just the most common types of homophonic puns requires a great deal of additional work, and filtering the list of matches efficiently and effectively is no joke.

We found the work intriguing and fascinating. We discovered many types of puns we hadn't previously considered and various problems that must be solved for the job to be done well. Through trial and error with the phoneme search tree we had hands-on experience with the efficiency advantages of depth-style searching with a stack as opposed to breadth-style searching with a queue, and the major efficiency deficits using recursion instead. While working on the filtering stage we learned about the problems using third party tools and online one in particular. When trying to evaluate our work we came in contact with the intricateness of distinguishing between puns and nonsensical homophonic matches. Viewing our output files showed us that using a dictionary for phonetic matching can be a serious overkill when the expected output is a pun that can be used in casual discussion.

6 Future Work

6.1 Lexical Inventiveness

Many puns rely on inventing a non-existing word that is composed of two parts relevant to the pun. As this project relies on the words used existing in the dictionary it cannot detect such puns even though they are fully homophonic. A future project could try to study such puns and find ways of detecting them.

6.2 Validate Output Sentences Semantically

This is a part of the Cleaning stage, that removes produced sentences that are invalid semantically or syntactically. As the syntax validation part is being covered with the help of parsing tree analysis, validating semantics of a sentence is a much more complicated task. A need for an efficient tool to detect whether a sentence makes sense has gradually intensified in the fields of NLP in recent years. The use of statistical techniques is a popular approach and may be used in future work in order to improve the cleaning stage of our project.

6.3 Fix the Sticky Problem

The solution presented for the infinite loop caused by single-phoneme words in the Sticky parameter is only a patch because it skips some phonemes which could actually be repeated or omitted. A future project could try to find a proper solution to the problem.

6.4 Contextual Approach

The test files we worked with are composed of out-of-context sentences. An important feature of many puns is their context and setting. A future project could study the effects of context and how to take advantage of them. In fact, context is not the only consideration that is required: Age and gender can be very influential on usage and understanding of puns and also other subjective characteristics. These should be taken into account as well in the research.

6.5 Integrated Puns

In this project we only dealt with ready stand-alone sentences as test data. An important improvment for future work is pun searching in paragraphs or even online pun detection during discourse with a user. It is important to note that this improvment is related very closely to the Contextual Approach described previously.

7 Related Works

Although much work has been done on classification and detection of humor in general and puns in particular, we did not find written material on detection of homophonic puns.

Yang et al. (2015) searched for humor anchors in sentences and detected humor through them. They did not search for puns specifically rather semantically humorous constructs. Even though their emphasis was not puns it is important to note that the idea of working through anchors is definately relevant in this context - we used a similar idea in the Threshold hyper-parameter.

Various groups worked on pun detection and understanding. Miller and Gurevych (2015) used Word Sense Disambiguation (WSD) to find the meanings of puns in corpora of homographic puns with only two meanings and exactly one pun word per sentence. Later Miller et al. (2017) also continued the research comparing various types of WSD to see which method has better success not only disambiguating the puns but also detecting their existance and locating them. Unfortunately, the restrictions they put on their research meant that they were not dealing with homophonic puns.

The closest study we found was done by Jaech et al. (2016). They cited various people who classified types of puns and analyzed puns theoretically. Their research was into understanding homophonic puns in sentences where the locations of the puns were already given. Since we are more interested in detecting if a homophonic pun exists and where, that approach was also not enough.

References

DATA-MUSE API. A word-finding query engine for developers.

The LOGON project. a Norwegian National initiative towards re-usable language technology for Norwegian English Machine Translation.

NLTK. Natural Language Toolkit for Python.

Aaron Jaech, Rik Koncel-Kedziorski, and Mari Ostendorf. 2016. *Phonological Pun-derstanding*. Association for Computational Linguistics.

Tristan Miller and Iryna Gurevych. 2015. *Automatic disambiguation of English puns*. Association for Computational Linguistics.

Tristan Miller, Christian F. Hempelmann, and Iryna Gurevych. 2017. *SemEval-2017 Task 7: Detection and Interpretation of English Puns*. Association for Computational Linguistics.

Diyi Yang, Alon Lavie, Chris Dyer, and Eduard Hovy. 2015. *Humor Recognition and Humor Anchor Extraction*. Association for Computational Linguistics.