Heterographic Pun Identification

Group 1

Melissa Kronenberger, Vinicius Lima, Aidan Sommers, Adrienne Zhang

CNIT 51900

PURDUE UNIVERSITY | Polytechnic Institute

First Ideas

Scope: Heterographic Puns

1. ~~Homographs~~
2. Pun word = word we **can** *change*
3. Often:

    Difficult-to-Notice meaning: Written

    Easy-to-Notice meaning: Aural

“I only lift weights on Saturday and Sunday because Monday to Friday are **weak** days”

“I only lift weights on Saturday and Sunday because Monday to Friday are [redacted] days”

>Prediction Magic<

“I only lift weights on Saturday and Sunday because Monday to Friday are **week** days”

First Ideas

Getting Started

1. Mask word or words in sentence
2. Language model predicts missing words
   - Predict **multiple options**, ranked
3. Prune word list until only "Sounds Similar" words remain
   - Eliminate homographs
   - Address plurals and contractions
4. Evaluate similarity of remaining words to original word
5. Do this for all words
6. Select least similar actual/predicted pair????
   - This is hopefully the pun
7. Weaknesses:
   - May struggle if hard-to-notice interpretation isn't the written one
   - Language model may be trained on puns

Identification of pun source and target word/phrase

Using typical pun structure

1. Split the pun into two parts
2. Find the source word/phrase based on connecting a term from one part of the context to a supporting term in the other part of the context

   We can also base this identification on grammatical inconsistencies due to the nature of heterographic puns
3. Find similar sounding words to the source (potential target words/phrases)

   Alternatively, we can mask the source and attempt to fill with a more typical word and/or correct the grammatical error. We then accept the fill as the target if it sounds similar to the source.
4. Filter and prune through similar sounding words based on contextual usage and grammatical correctness (prune using "LanguageTool")

Reasoning if the resulting source and target make a pun

Prompting an LLM based on typical pun structure

You will be given a context of one or more sentences and will decide if it contains a pun.

You will be given two versions of the context. One version contains a source word (or phrase) and the other version contains a target word (or phrase).

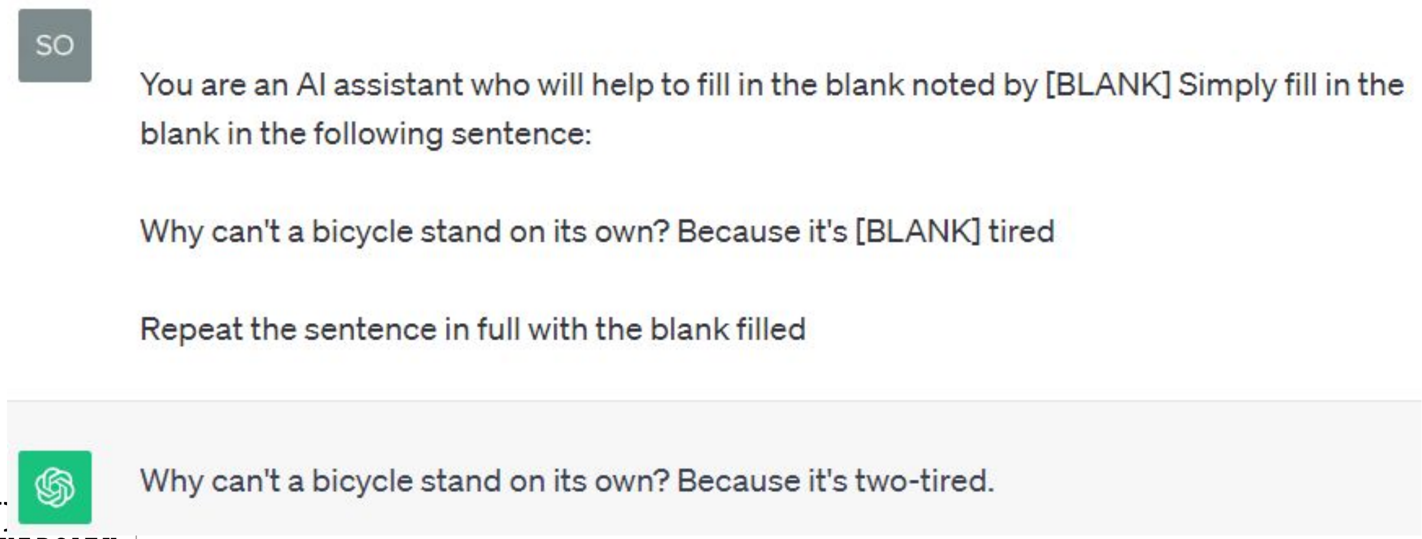
If the meaning of source is compatible with one part of the text while the meaning of target is compatible with another (different) part of the text, say this is a pun. If this is not the case, say this is not a pun.

For example, if the target can stand on its own within a part of the context while the source cannot (such as the source needing a supporting term from a different part of the context), it would be a pun.

Prioritizing step by step reasoning over memorization

Splitting the steps into many prompts and avoiding mention of humor

- We want to achieve a structure for dissecting and reasoning about puns, not simply memorizing puns.
- LLM memorization of puns actually caused performance issues in our structure:



Determining Similarity

First Attempts:

- "Minimum distance"

```
similarity = len(set(target_pronunciation) & set(pronunciation[0])) / len(set(target_pronunciation)
| set(pronunciation[0]))
```

['AH0', 'L', 'OW2', 'P'] -> CantALOUPE
['IH0', 'L', 'OW1', 'P'] - can't ELOPE

- "Minimum distance" + new rules

```
pronunciations_two_letters = [item[:2] for item in pronunciations]
similarity = len(set(sub_phonetics) & set(transformed_list)) / max(len(set(sub_phonetics)),
len(set(transformed_list)))
```

They worked well, however they resulted in too many possibilities.

Determining Similarity

Options for Non-Minimum Edit Distance

- Phonetic encoding/matching algorithms
    - SoundeX
    - Metaphone
    - Caverphone
    - NYSIIS
    - Match Rating Codex
- Edit distance measures
    - Levenshtein
    - **Weighted-Levenshtein**
    - Damerau-Levenshtein
    - Jaro Distance
    - Hamming
- Edit distance implementations
    - Simple Recursive
    - Hirschberg's algorithm
    - **Wagner–Fischer algorithm**
    - Levenshtein automatons

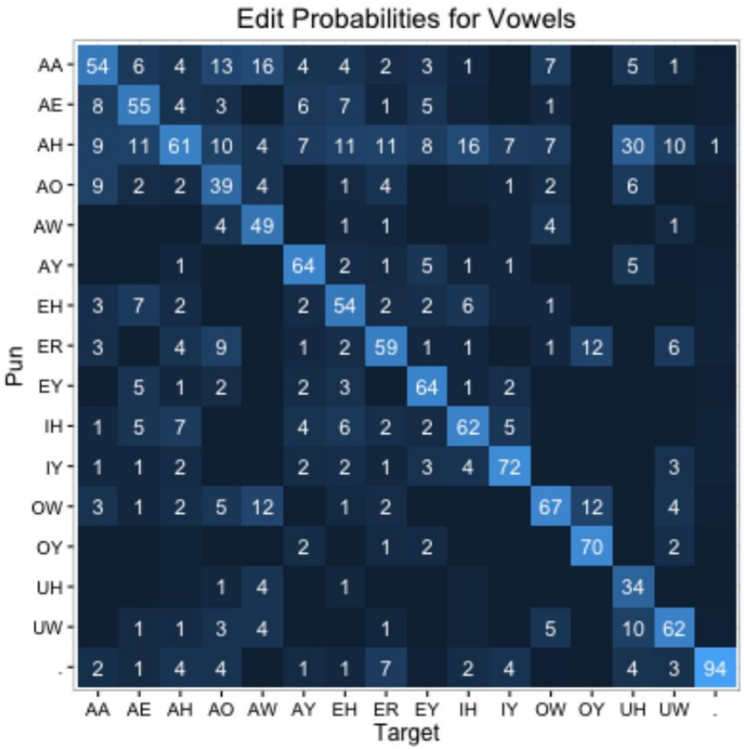
Determining Similarity

Grocery List

- Convert words to phonemes
  - CMU Pronouncing Dictionary
- Calculate edit distance
  - strsimpy - library for string similarity/distance measures
  - Weighted levenshtein distance
- Weighting system
  - Phonological Pun-derstanding matrices

Jaech, A., Koncel-Kedziorski, R., & Ostendorf, M. (2016). Phonological Pun-derstanding. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 654–663. https://doi.org/10.18653/v1/N16-1079

Determining Similarity



Manipulate Pun-derstanding Matrices

- #s are percentages
- Unmarked squares are < 0%
- Add up vertically to 100%
- Indicate % likelihood phoneme in original word became phoneme in target word

- Normalize: Dividing by chance of self-replacement
- Subtract from 1
- Result is now *cost* to replace

Determining Similarity

Putting it Together

- Existing Levenshtein implementations…
  - Manipulate singular characters
  - Arphabet -> single-character phoneme format (IPA-based)
- Levenshtein calculates *distance*; We want *similarity*
  - Normalize: Divide by max phoneme length of longest string
  - Subtracted from one
- No matrix exists for vowel-to-consonant
  - Fallback: Compute cost using insertion + removal*
- Test it out:
  - ‘Hello’ vs ‘Hell no!’ = 65.3% similarity
  - ‘Cantaloupe’ vs ‘Can’t Elope’ = 90.7% similarity

* room for improvement

Optimization

Tradeoff between number of words and similarity score

- We were not only interested in exact similar words (TWO and TOO), but also in breaking down the pun word(s) into two possible similar words.

  Example: cantaloupe -> can't elope / a waifer -> away for
- This approach crashed the code, due to overwhelm amount of similarity calculations.
- One solution was to only check similarity on words that have approximate size.

```
if abs(len(sub_phonetics) - len(pronunciation[0])) <= 1:
```

Limitations

Detecting Similarity

1. Reliant on CMU Pronouncing Dictionary

   “What’s a ghost’s favorite pie flavor? Booberry!”

   Options: CMU’s LOGIOS or trained machine algorithm could generate phonemes.
2. Normalization disproportionately affects small words

   far : for = ~66% related (1/3rd of the string is different)

   bailed : boiled ~83% related (1/6th of the string is different)

   Options: Ignore vowels
3. Speed

   Large number of comparisons: Slow

   Fixes: Exterior optimizations
4. Only checks one-word or two-words similar sounds

Limitations

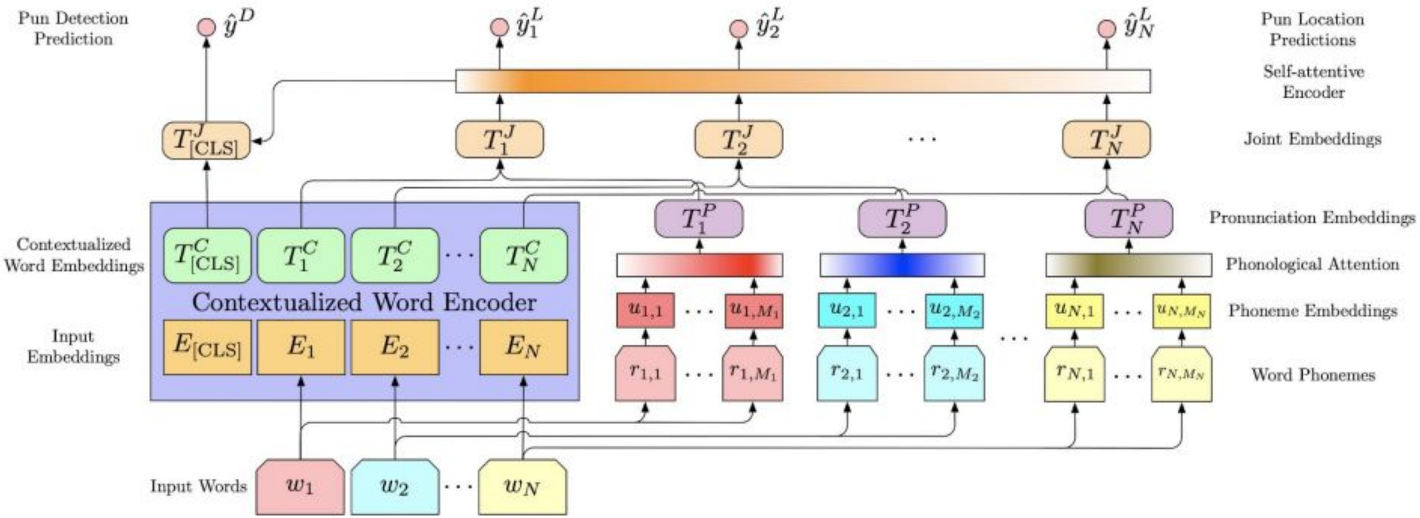
The Trouble with Vowel and Consonants

1. Most Systems handle Vowels separately from Consonants, or Discard them
    - “Bell**s** coasting”-> “Bell**i**cose thing“
    - Levenshtein: All phonemes equally interchangeable
    - Matrices: Consonant/Vowel not interchangeable
    - Importance: Consonants > Vowels?
    - Options: Only process consonants? Dim weighting effect? Process vowels and consonants as separate strings and add?
2. ‘R’ is a Vowel and a Consonant
    - “for” vs “fur” = ‘F AO R’ vs ‘F ER’
    22% Similarity - ‘AO’ gets paired with dissimilar ‘ER’, and a whole consonant ‘R’ is “deleted”!
    - Options: Special conversion case? Process as ‘ER’ as ‘EH R’?
3. ‘DH’ was substituted more often than it remained the same
    - The Pun-derstanding paper was based on real world data
    - Our normalization method accidentally caused negative edit weights

Exploring Further Optimizations

Pronunciation-attentive Contextualized Pun Recognition

Overall Idea: Pun detection and location utilizing contextualized word embeddings and pronunciation embeddings



Zhou, Y., Jiang, J.-Y., Zhao, J., Chang, K.-W., & Wang, W. (2020). "The boating store had its best sail ever": Pronunciation-attentive contextualized pun recognition. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/2020.acl-main.75

Exploring Further Optimizations

Progress

Currently training pun classification models

- Trained on SemEval2017 Data
- Utilizing BERT to extract contextualized word embeddings
  - Using bert-base-cased
  - Focusing on semantics of the entire input
- Applying attention mechanism to derive pronunciation embeddings and identify important phonemes
  - Estimates importance vector for each word
- Concatenating (overall) word and joint pronunciation embeddings
  - i.e., combining overall contextualized embedding and aggregated self-attentive embeddings
- Training a Linear Classifier for Binary Classification (currently optimizing hyperparameters)
  - Utilizing softmax function on fully connected neural network layer, deriving logits from the vector modelling overall semantics
- Optimization with cross-entropy loss

Exploring Further Optimizations

Hoping to see improved results

Claim: static word embeddings/external knowledge bases are not effective at categorizing heterographic puns

Comparison with Benchmarks:

- Rule-based machine learning classifiers
  - Duluth, JU_CSE_NLP, PunFields, UWAV, Fermi, UWaterloo
- Recurrent NN (Sense)
- Captures Linguistic Features (CRF)
- RNN and CRF (Joint)
- No Phoneme Consideration (CPR)

Model	Heterographic Puns					
	Pun Detection			Pun Location		
	P	R	F_1	P	R	F_1
Duluth	73.99	86.62	68.71	-	-	-
JU_CSE_NLP	73.67	94.02	71.74	37.92	37.92	37.92
PunFields	75.80	59.40	57.47	35.01	35.01	35.01
UWAV	65.23	41.78	42.53	42.80	42.80	42.80
Fermi	-	-	-	-	-	-
UWaterloo	-	-	-	79.73	79.54	79.64
Sense	-	-	-	-	-	-
CRF	89.56	70.94	79.17	88.46	62.76	73.42
Joint	86.67	93.08	89.76	81.41	77.50	79.40
CPR	93.35	95.04	94.19	92.31	88.24	90.23
PCPR	**94.84**	**95.59**	**95.22**	**94.23**	**90.41**	**92.28**

Demo

[redacted]

Questions?

polytechnic.purdue.edu

/ TechPurdue