Homophone Checker using Bert

Table of contents

Introduction

Unlike typical spelling or grammatical errors, using the wrong homophone can result in sentences that sound perfectly correct but actually use the wrong word(s) entirely. A homophone is defined as "one of two or more words pronounced alike but different in meaning or derivation or spelling (such as the words to, too, and two)". While most grammar and spelling checks can pick up on these mistakes, others, such as Apple's phone keyboards, fail to check and correct such mistakes. In this project, we hope to create a model capable of taking an input text and correctly identifying and correcting homophone mistakes to output a grammatically correct sentence.

A unique difficulty with homophones is that they can vary based on a dialect. For example, shed and shared are homophones in the Australian dialect, but in American English, these

words sound entirely different. Some homophones are homophones independent of dialect while others depend on the dialect. For this project, we have assembled 442 sets of homophones such as [to, too, and two] containing a total of 941 homophones. We do not claim that our list contains all possible homophones, if such a definitive list does exist. However, we believe our list encompasses the most common American and English dialects' homophones and that our results would hold for a larger list of homophones as the model would work similarly.

Test Data Creation

Text Retrieval

First, we had to find a sufficiently large, grammatically correct corpus. To do this we used Project Gutenberg to read in 10 books. We are operating on the assumption that published text, particularly the most popular ones on the project will be almost entirely grammatically correct, with some potential exceptions such as dialogue. However, with a large enough dataset, these occasion mistakes should not impact the overall accuracy reads on our model. We selected the top 10 books by downloads on Project Gutenberg:

- Frankenstein
- Moby Dick; Or, The Whale
- A Room with a View
- Middlemarch
- Pride and Prejudice
- The Complete Works of William Shakespeare
- Little Women
- The Enchanted April
- The Blue Castle

These were then read in using the requests library. Capitalization was then removed for consistent formatting. Then, we cleaned any text formatting to create a giant text that was then put through NLTK's sent_tokenize function to create a list of sentences from the 10 books. In total, we found 68,573 total sentences.

Homophones List

We then assembled our list of homophones using a variety of online homophone lists and combinations we could think of.

Error Insertion

As above, we are operating on the assumption that all of the sentences in our dataset begin as grammatically correct, containing no homophone mistakes. Therefore, we need to artificially create homophone mistakes in our to test our models effectiveness on a large scale.

We flattened our list of homophones sets to create a list containing all 942 possible homophones. We then iterated through each sentence. If a sentence contained no possible homophone mistakes, there is nothing further to do. If it contains one homophone, then the homophone is replaced with a mistaken homophone with probability p=0.7. This p value was selected to give a sufficiently large collection of mistakes to analyze the models performance. If a sentence contains multiple homophones, including the same homophone multiple times (most commonly occurs for "to"), each homophone is weighted in accordance with:

- `count` = total appearances of the homophone in the current dataset
- `max_count` = maximum count of homophones in the sentence

$$w_i = 1 - \frac{\text{count}}{\text{max\_count} + \epsilon}, \epsilon = 1e - 10$$

These weights seek to give words that we have less data on a higher probability of being selected, testing the model on more homophone mistakes. Without it, common homophones such as "to", "in", and "there" would dominate the test data; here, while they remain the most common, it is much more evenly distributed. The homophone is then selected, it and its index saved in order to ensure that the model is recognizing the correct homophone mistake in a sentence if it contains the same word multiple times. The functions final output is a dataframe with the following columns:

- sentence (object): New sentence for testing, potentially containing errors.
- has_homophone (bool): Boolean variable stating whether a sentence contains a homophone.
- is_error (bool): Boolean variable stating whether an error as been added to the sentence.
- error_idx (float64): Location of the error, if applicable.
- error (object): The incorrect homophone, if applicable.
- correct_word (object): The correct homophone, if applicable.
- correct_sentence (object): The final, correct sentence, will be the same as the original sentence if `is_error=False`.

The final dataframe has a shape of (68573, 7). 56,484 (82.37%) of these sentences contain at least one homophone, demonstrating the importance of checking for these mistakes. There are 39,446 (57.53%) sentences containing homophone errors. The most commonly replaced homophones were "to", "in", "you", "for", and "but". Below is the distribution of total sentences in which homophones were replaced.

The final output is saved as a csv file in the data folder as gutenberg-homophone-errors.csv to avoid having to rerun the model every time.

```
import plotly.express as px
import pandas as pd
from plotly.offline import init_notebook_mode

init_notebook_mode()

error_df = pd.read_csv("./data/gutenberg-homophone-errors.csv")
homophone_counts = error_df["correct_word"].value_counts(dropna=True).reset_index()

# Sort by count
homophone_counts = homophone_counts.sort_values(by="correct_word",ascending=False)
fig = px.histogram(
    homophone_counts,
    x="index",
    y="correct_word",
    title="<b>Distribution of Replaced Homophones</b>",
    width=800,
    height=500,
    category_orders={"correct_word": homophone_counts["index"]},
    hover_name="index",  # Name shown in the tooltip
    hover_data={"correct_word": True},
)
fig.update_traces(
    hovertemplate='<b>Homophone:</b> %{x}<br><b>Count:</b> %{y}',
)
fig.update_layout(xaxis_title="<b>Homophone</b>", yaxis_title="<b>Count</b>", hovermode="x
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json, text/html
```

Homophone Checker Function

The homophone checker function is used to test the model; the final model for the interactive webpage functions the exact same, just only outputting the final correct sentence. It first imports the same homophone list used to create the data. Next, it imports Hugging Face's

fillmask task on the BERT base model. BERT was selected due to its speed and accuracy compared to other models. Other models tested include the XLM Model (xlm-mlm-en-2048), Roberta Base, and Albert Base (albert-base-v2).

Once the model as been imported, the input text is lowered and separated to create a list of all words in the sentence. It then similarly scans the sentence to check if it contains any homophones. If it does not contain any homophones, the original sentence is returned.

If the sentence contains one or more homophones, it iterates through each homophone, replacing the homophone with the models mask token; in the case of BERT, it uses '[MASK]'. The masked string is then run through the model, returning the top 50 most likely tokens to appear where the mask token has been placed. In most cases, the original word will be one of the top tokens. However, in cases such as "I saw a deer in the woods", deer could be one of numerous possible words, requiring the model to return at least 10 potential tokens. Using the list of all possible homophone replacements, the most likely homophone is chosen as the correct homophone choice. If it matches the original word, then that homophone is presumed to be correct. We experimented with adding a probability threshold, necessitating that a replacement token be sufficiently more likely to occur than the original token to be ruled an error, however, found no evidence that this increases the accuracy of our model. Each homophone is then tested, using the most recently correct sentence each time moving from the beginning of the sentence to end. For example, if it was testing the sentence "I eight way two much food" it would mask "eight", testing on "I [MASK] way two much food". Then, it would correct "eight" to "ate". Next, it would test "two", but now using the corrected sentence to give "I ate way [MASK] food" to increase the likelihood of finding the proper replacement as the model moves unilaterally through the sentence. In future iterations, it could be improved to move through the sentence bidirectionally, to determine the most likely final sentence, but we avoided this to improve output speeds. Finally, function returns an identically shaped dataframe as the test data creation for efficient results:

- sentence (object): New sentence for testing, potentially containing errors.
- has_homophone (bool): Boolean variable stating whether a sentence contains a homophone.
- is_error (bool): Boolean variable stating whether an error as been added to the sentence.
- error_idx (float64): Location of the error, if applicable.
- error (object): The incorrect homophone, if applicable.
- correct_word (object): The correct homophone, if applicable.
- correct_sentence (object): The final, correct sentence, will be the same as the original sentence if `is_error=False`.

On average, the model can test a sentence in 1.51 seconds. This speed is impacted most by the number of homophones in the input.

The final output is saved as a csv file in the data folder as gutenberg-berg-uncased.csv to avoid having to rerun the model every time.

Results

Interactive Webpage

Conclusions

Sources