Homophone Checker using Bert

Austin Barish, Marion Bauman, Hongxin Wu

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.

In addition to the development of this homophone correction model, we have taken a step further by creating an interactive web application. This web app, designed with user-friendliness in mind, allows users to input text and receive corrected versions in real-time. It's not just about correcting homophones; the web app also integrates spelling corrections, offering users a comprehensive text correction tool. This deployment as an interactive webpage demonstrates the practical application of our NLP model, making it accessible and useful for everyday text correction needs.

Through this project, we have expanded beyond traditional text correction, introducing a tool that is both theoretically robust and practically valuable, enhancing written communication across different dialects and everyday scenarios.

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

When measuring the performance of our model, we emphasized a few key metrics including: accuracy, speed, and precision. We also considered the impact of the number of homophones in a sentence on the model's performance. Additionally, we considered the edit distance between the original sentence and the corrected sentence, counting the number of words that were modified to correct the homophone mistake.

Our team tested multiple different models to determine which one would be the most effective for creating a homophone correction tool with low latency, high accuracy, and high precision. The following table shows the results of our testing:

Model	Accuracy	Precision (Error)	Precision (No Error)	Test Size
bert-base-uncased	0.9282	0.9957	0.8585	10,000
XLM-MLM-EN-2048	0.7280	0.6000	0.7454	250
roberta-base	0.8060	0.9697	0.7811	500
albert-base-v2	0.9100	0.9826	0.8883	500

While testing four models options, we found a clear winner both in terms of performance and speed. The XLM-MLM-EN-2048 model performed the worst, with only 72.8% accuracy. Furthermore XLM had a very low precision for making changes when an error was present. This means that the model was frequently modifying sentences that did not contain any errors, which we knew would be a bad result for end users. XLM also had a high latency, taking hours to run just 250 inputs, making it impractical for our model. The roberta-base model performed better than XLM but still had a low accuracy of 80.6%. The precision for roberta-base was much better than XLM, but still not very good. Roberta had a fairly low latency, but it was

still slower than bert-base-uncased. The albert-base-v2 model had a very good accuracy of 91%, with accompanying precision scores of 98.3% and 88.8%, and the albert model was also fairly fast, making it the second best model that we assessed.

Based on our testing, we chose bert-base-uncased as our final model. In order to assess our model's performance, we ran it on 10,000 sentences that we created using the Gutenberg corpus, where we artificially inserted homophone errors. Our model achieved a 92.8% accuracy on our test data. The model had a precision of 99.6% for identifying errors, indicating that the model is very good at modifying only sentences that contain errors. We wanted our model to avoid changing text as much as possible, only making edits where there was clearly an error, so this aligns well with our goal. The precision for sentences that did not contain errors was 85.9%, which indicates that our model misses some errors but generally performs well. The averge F1 score for our model for detecting errors is 0.93, showing our model's high performance. Below, we show the confusion matrix for our model's performance on determining whether a sentence contains a homophone error:

We further analyzed our model's performance by determining the most common error locations. We wanted to ensure that our model performed equally well, no matter where the error was located in the sentence. We found that there were no significant differences in performance based on the location of the error, which supports the idea that our model is robust and can handle different types of homophone errors. The following visual shows the confusiong matrix for our model's performance on detecting errors in different locations:

We also explored the most common homophones that our model failed to correct. Our model struggled with "it's" versus "its", failing 17 times to replace "its" with "it's". This could be connected to the tokenization of "it's" as two tokens or the commonality of this misuse in BERT's training data. Another common failure for our model is between "scene" and "seen", with the model correcting misuses of "seen" only 63% of the time.

Overall, we found that if our model detected an error, the correction would be correct. The main limitation of our model is in error detection, as our model sometimes fails to detect errors. However, our model also has a very low false positive rate, meaning that it rarely makes changes to sentences that do not contain errors. This is a good result for our model, as we want to avoid changing text as much as possible. We also found that our model performs well regardless of the location of the error in the sentence.

Interactive Webpage

In an effort to extend the reach and impact of our homophone checker model, we have developed an interactive web application. This application serves as a direct, real-time interface for the model, embodying simplicity and effectiveness. It's designed to engage users and make the sophisticated technology behind our model accessible and user-friendly.

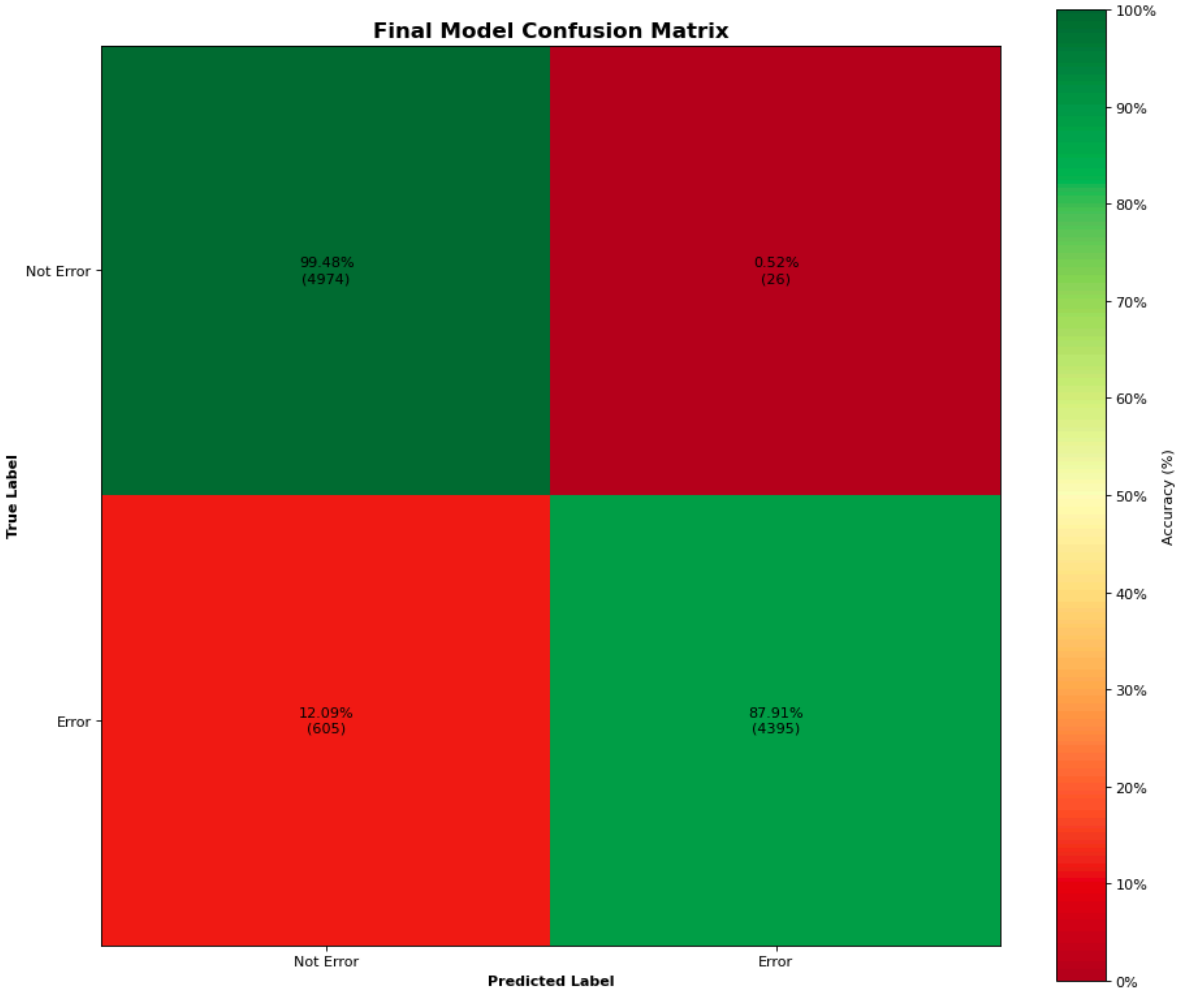

Figure 1: image.png

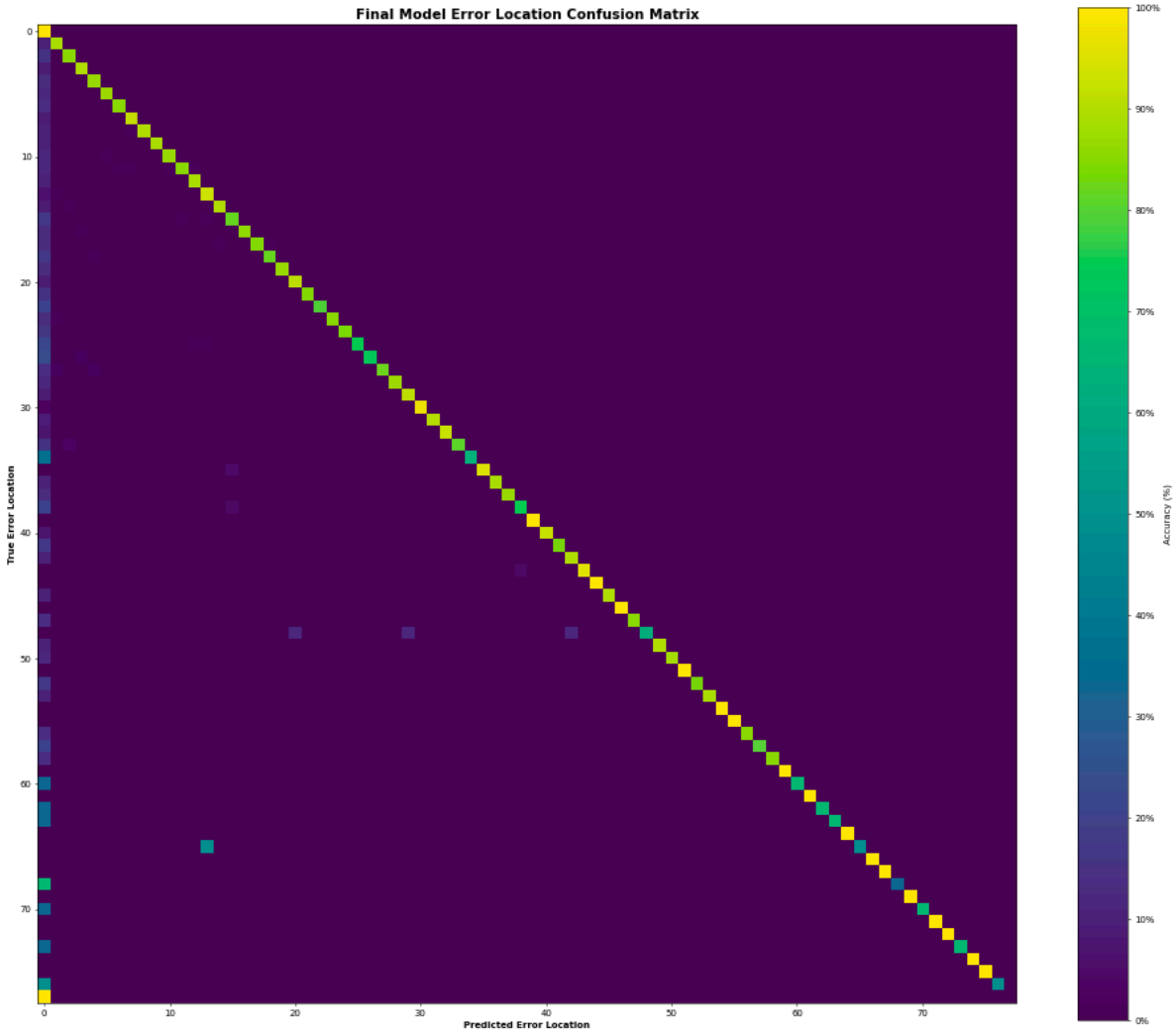

Figure 2: image-2.png

Web Application Design and Features

Our web app features a design that emphasizes ease of use and minimalism. The interface is intuitive, focusing on functionality, which ensuring that users of all levels can navigate and utilize it without complexity. Key features of the webpage include:

- **Text Input Area** A designated area where users can type or paste text. This is the primary interaction point where users input the sentences they want to check for homophone errors.
- **Correction Display** Upon submitting their text, users could immediately see the corrected version. The web app provides two types of corrections: one showing homophone corrections and another combining both homophone and spelling corrections. This dual-output approach allows users to compare and understand the enhancements made to their text.
- **Styling and Responsiveness** The webpage is styled using a `style.css` file, ensuring the interface is not only visually appealing with a light blue, 'cute' theme but also responsive to different device screens. This makes the web app accessible from various devices, enhancing user experience.

Behind the Scenes: app.py and homophone_utils.py

The backend of the web application, developed in `app.py` handles user requests and processes text through our homophone checker model. This Python script uses Flask as the web framework, enabling efficient handling of web requests and dynamic content generation.

`homophone_utils.py` plays a crucial role in the backend. It contains the logic for our homophone correction model, which is then used by `app.py` to process user-input text and generate corrected sentences.

Deployment and Usage

Our interactive web application is deployed on **Heroku**, ensuring easy access for anyone looking to enhance their text's accuracy. Users can simply visit the webpage, input their text, and swiftly receive corrections, making it an invaluable tool for everyday writing tasks.

This web app not only shows how well our model works but also makes advanced language processing techniques easy and practical for everyday use. It's a great example of turning complex language models into user-friendly tools that help improve how we write and communicate, making these advanced technologies accessible and useful for everyone.

Conclusions

Our project successfully addresses the challenge of correctly using homophones—words that sound the same but have different meanings or spellings. We developed a specialized model to fix these errors in writing. For this, we utilized a large collection of sentences (68,573) from Project Gutenberg books. Our tests demonstrated that the model was highly effective in finding and fixing homophones, achieving a 92.8% accuracy rate in homophone detection and correction, and an impressive 99.6% precision in correcting detected homophone errors across 10,000 sentences.

In the modeling phase, we utilized advanced Natural Language Processing (NLP) techniques, taking advantage of the speed and accuracy of models like BERT (bert-base-uncased). We integrated its predictions with our homophone correction logic. This strategy ensured that the model accurately comprehended the sentences and made precise corrections. During the testing phase, we further enhanced the model's accuracy by introducing artificial homophone errors into our dataset, aiming to replicate real-world scenarios. By randomly inserting homophone mistakes into sentences that were initially free of such errors and grammatically correct, we generated a diverse range of test cases. This method allowed us to thoroughly test and refine the model, ensuring its ability to handle a wide variety of textual inputs.

A key achievement of our project is the creation of an interactive web application, showcasing the practical application of our model. This user-friendly platform demonstrates the effectiveness of our model and ensures easy public access. Users can input text and receive immediate corrections, addressing both homophone and spelling errors. The simplicity of the web app, combined with its dual correction display feature, makes it an invaluable tool for enhancing everyday written communication.

We also put this web app on Heroku, an online platform. This means anyone with internet can use it to improve their writing. By doing this, we've made a tool that's not just for research but for everyday use, helping people communicate better.

In short, our project goes beyond just theory. We've made a practical tool that helps with real-world writing, showing how technology can make a difference in everyday life.

Sources