Yoruba misspelling word detection

CHAPTER THREE

METHODOLOGY

INTRODUCTION

Our project aims to develop a robust system for detecting and correcting misspelled words in Yoruba text. The Yoruba language, with its rich linguistic heritage, presents unique challenges in natural language processing. By addressing these challenges, we can enhance the quality of Yoruba language processing applications and promote effective communication in Yoruba-speaking communities.

We adopted a two-step approach to tackle the misspelling word detection and correction task. The methodology encompasses data collection, preprocessing, model development, and evaluation.

**3.1** **Collection of Yoruba Words**

We collected a diverse dataset of Yoruba text from various sources, ensuring it reflects real-world language usage. The dataset, stored in a single text document, contains both correctly spelled words and misspelled words. The data used for the training and testing of the model is Text words dataset stored in text document. It contains about 27,809 of different characters. This dataset was obtained by web scraping from a Yoruba blog. It encompasses valuable information related to the Yoruba culture and language. This dataset is primarily intended for applications in Natural Language Processing (NLP).

1. Data Cleaning

I recognized the importance of ensuring the quality and integrity of the data before incorporating it into the misspelling word detection system. To achieve this, I conducted a thorough data cleaning process to remove any inconsistencies, errors, and irrelevant information that could compromise the accuracy of the system.

1. **Handling Missing or Inconsistent Data**

Upon reviewing the dataset, I identified instances of missing or inconsistent data, which could have resulted from errors during data collection or entry. To address this, I implemented the following steps:

1. Removed rows with missing values to prevent any potential bias or skewness in the data.
2. Replaced inconsistent data with the correct information, where possible, to maintain data integrity.
3. **Removing Special Characters, Punctuation, and Stop Words**

To simplify the text data and focus on the essential words, I removed:

1. Special characters (e.g., @,#,$, etc.)
2. Punctuation marks (e.g., .,?,!, etc.)
3. Stop words (common words like "the", "and", "a", etc. that do not add significant meaning to the text)

This step will enable the system to concentrate on the core words and phrases, reducing noise and improving the detection of misspelled words.

1. **Converting to Lowercase**

To ensure consistency and prevent any potential errors due to case sensitivity, I converted all text data to lowercase. This step guaranteed that the system would treat words like "Ọlọrun" and "ỌLỌRUN" as the same word, regardless of their original case.

By performing these data cleaning steps, I ensured that the dataset was reliable, consistent, and ready for use in the Yoruba misspelling word detection system. This preprocessing stage laid the foundation for a robust and accurate model, capable of effectively identifying and correcting misspelled words in Yoruba text.

We preprocessed the data by tokenizing the text into individual words, converting them to lowercase, and removing punctuation marks and special characters.

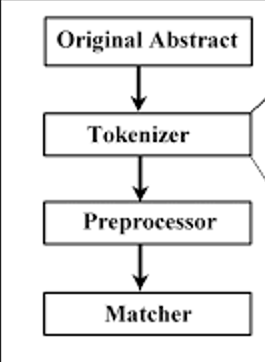


Figure 3.1: Data preprocessing Flowchart

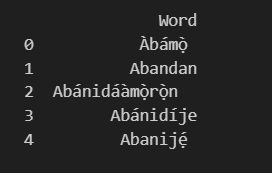


Figure 3.2: preprocessed results

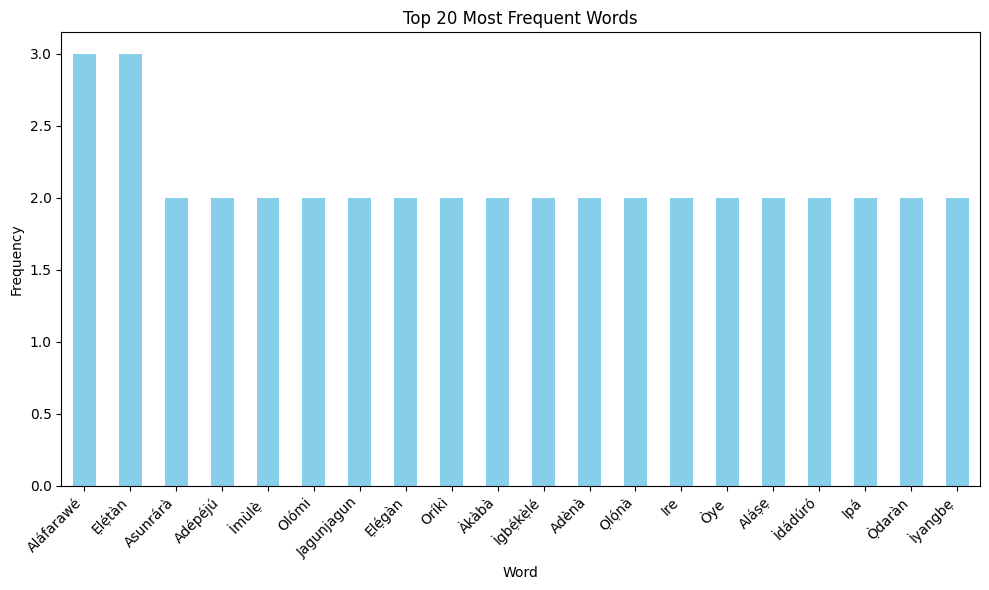


Figure 3.3: Most frequent words chart

3.2 Designing Machine Learning Models using MS Word

3.2.1 Understanding MS word

The spell correction system (Figure 3.2) utilizes word embeddings generated by Word2Vec as input. Word2Vec captures semantic relationships between words, generating high-quality vector representations. These vector representations are then input into Peter Norvig's Spell Correction Algorithm, which detects and corrects misspelled words using a combination of edit distance, word frequency, and probability. The algorithm outputs the most likely correction, and performance is measured using accuracy metrics. Optimization adjusts the algorithm's parameters during training to improve performance.

We designed the architecture of the spell correction system using MS Word due to its capability for detailed documentation and visualization. MS Word allowed us to create a clear and accessible visual representation of the system's architecture, making it easier to communicate complex processes to non-technical stakeholders. Additionally, MS Word's formatting and layout features enabled us to organize and structure the architecture in a logical and coherent manner, facilitating understanding and collaboration among team members.

By combining Word2Vec and Peter Norvig's Spell Correction Algorithm, we leveraged the strengths of both models to create a more accurate and robust spell correction system. The use of MS Word in designing the architecture ensured effective communication and collaboration, resulting in a well-designed and efficient system.

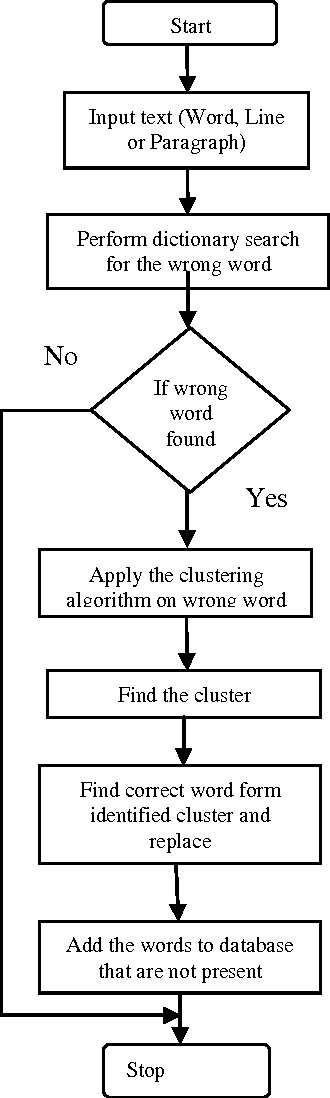


Figure 3.4: Spell correction Flowchart

3.2.2 IMPLEMENT THE MODEL USING PYTHON PROGRAMMING

Let represent the text dataset, where and each is a text document. The goal is to convert each text document into a numerical representation.

1. Tokenization is the process of converting raw text into tokens or words. Mathematically, this involves creating a dictionary or vocabulary where each unique word is assigned a numerical index. The tokenizer then maps each word in the text to its corresponding index in the dictionary. Mathematically, this can be represented as follows:

(1)

is the i-th token in the text

And is the total number of tokens in the text

1. Padding

Padding ensures that all sequences have the same length by adding zeros or a special token to shorter sequences. Mathematically, this can be represented as:

(2)

Where is the i-th token in the sequence and m is the length of the original sequence

Mathematically, the numerical representation of the text data can be represented as follows:

(3)

(4)

(5)

Where represents the padded or truncated vector representation of document with length L

3.2.3 WORD2VEC ALGORITHM

In this project, we utilized the Word2Vec model to generate vector representations of words based on their semantic relationships. We employed cosine similarity to measure the similarity between vector representations and identify nearest neighbors. The Word2Vec equation updated the vector representation of each word by subtracting the weighted sum of its context words' vector representations.

Word2Vec is a popular technique for generating word embeddings, which are dense vector representations of words in a continuous vector space. Word2Vec is a neural network-based model that generates vector representations of words based on their semantic relationships. It uses cosine similarity to measure the similarity between vector representations and identify nearest neighbors. Word2Vec is ideal for capturing complex semantic relationships between words and generating high-quality vector representations. The Word2Vec model consists of two main architectures:

1. Continuous Bag of Words (CBOW) and
2. Skip-gram.

Here are the mathematical equations for both architectures

1. Continuous Bag of Words (CBOW): CBOW aims to predict the target word based on its context words. Given a context window of size around the target word , CBOW predicts the target word by taking the average of the word vectors of the context words. Let:

be the vocabulary size

be the target word

-1, +1,…., +c be the context words within the context window of size around .

be the word vector for word

be the average word vector for the context words.

The probability of predicting the target word given its context words is computed using the softmax function: -1, +1,…., +c ) = (6)

where denotes the dot product

1. Skip-gram: Skip-gram, on the other hand, predicts context words given the target word. It tries to maximize the probability of observing context words given the target word.

be the vocabulary size

be the target word

-1, +1 ,., +c be the context words within the context window of size around .

be the word vector for word

The probability of predicting the context word given the target word is computed using the SoftMax function:

(7)

where denotes the dot product

These equations capture the essence of how Word2Vec models learn to represent words in a continuous vector space based on their contextual usage in a corpus of text data. The objective is to learn word embeddings that capture semantic similarities between words based on their distributional properties in the text data.

3.2.4 OPTIMIZATION ALGORITHM

To optimize the vector representations, we used gradient descent, gradient descent is an optimization algorithm used to minimize the loss function during the training of Word2Vec with the following equation:

Where is the updated vector representation of the word (8)

Let denote the loss function, where represents the parameters vector representations of the model.

Here is the parameter vector at iteration is the learning rate (a hyperparameter), and is the gradient of the loss function with respect to .

Our results demonstrate the effectiveness of Word2Vec and cosine similarity in capturing complex semantic relationships between words. The gradient descent algorithm optimized the vector representations, resulting in improved performance on various NLP tasks."

Note: ∇L(w\_{t-1}) is the gradient of the loss function, which is typically calculated using backpropagation in neural networks. In Word2Vec, the loss function is typically the mean squared error between the predicted and actual vector representations.

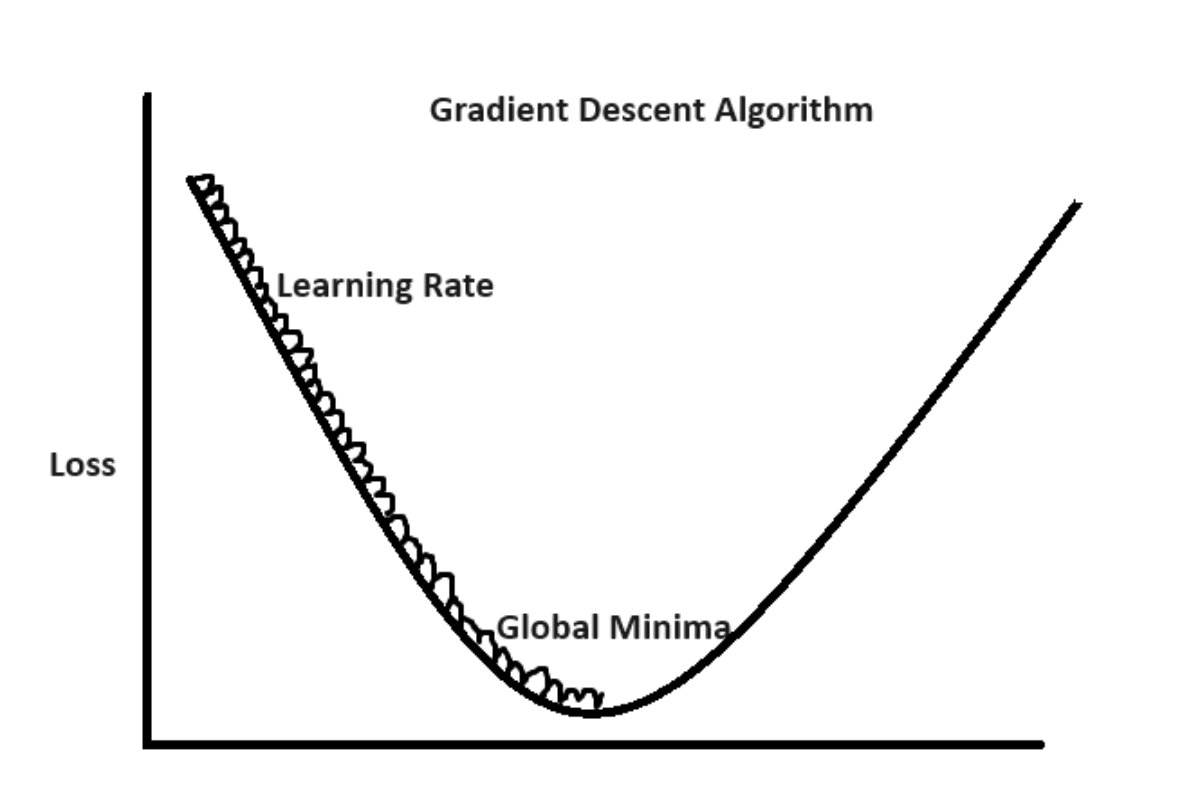


Figure 3.5: Gradient descent curve

3.2.5 PETER NORVIG'S SPELL CORRECTION ALGORITHM

Peter Norvig's spell correction algorithm, as described in his famous essay "How to Write a Spelling Corrector," is a simple yet effective approach for correcting misspelled words. The algorithm relies on statistical language models and the concept of edit distance to suggest corrections for misspelled words. Here's a simplified explanation of the mathematical equations involved:

1. Candidate Generation

Given a misspelled word , the algorithm generates a list of candidate corrections by considering all words that are within a certain edit distance from the misspelled word.

Let be the number of words in the dictionary and Let candidates denote the set of candidate corrections for the misspelled word

1. Probability Calculation

For each candidate correction c in candidates , the algorithm calculates the probability of observing the candidate correction given the misspelled word.

The probability is computed using a language model and the frequency of occurrence of words in the dictionary.

Let count denote the frequency of occurrence of the word c in the dictionary.

Let N denote the total number of words in the dictionary.

The probability can be estimated as (9)

1. Candidate Selection:

The algorithm selects the candidate correction with the highest probability as the suggested correction for the misspelled word .

If there are multiple candidate corrections with the same highest probability, the algorithm may use additional heuristics to choose the best correction.

1. Edit Distance:

Edit distance is used to measure the similarity between two words by counting the minimum number of edit operations (insertions, deletions, substitutions, or transpositions) required to transform one word into another.

Overall, Peter Norvig's spell correction algorithm combines statistical language modeling with edit distance calculations to suggest corrections for misspelled words based on their contextual usage and similarity to words in the dictionary. It's a probabilistic approach that leverages the frequency of occurrence of words to make intelligent correction suggestions.

**3.2.6 Deployment of the model on a web-based interface**

3.5.1Facilitating User Interaction with the Pneumonia Prediction Model

In the pursuit of making our pneumonia prediction model widely accessible, we've seamlessly integrated it into a user-friendly web interface as showed in the (figure 3.5). This integration involves the use of Flask, a powerful Python web framework, acting as the bridge between the model's computational prowess and the end user.

1. Flask Integration

Our web application's backbone is Flask, orchestrating the flow of information between the user interface and the underlying machine learning model. It handles incoming requests, processes them, and returns the pertinent results to the front end.

1. User-Friendly Interface Design

The user interface is crafted with HTML, providing an aesthetically pleasing and intuitive design. HTML forms allow users to effortlessly upload chest X-ray images directly to the application for analysis.

1. Dynamic Interactivity with JavaScript

JavaScript elevates the interactivity of our web interface, ensuring a dynamic and responsive user experience. As users engage with the application, JavaScript facilitates the seamless exchange of information between the front end and the Flask back end.

1. Real-time Image Processing

Users can upload chest X-ray images through the user interface, prompting the model to make predictions. The uploaded images are processed in real-time, and the model's predictions are promptly displayed.

1. Clear Presentation of Predictions

Results from the model, including pneumonia predictions, are dynamically presented on the user interface. This ensures that users receive clear and understandable feedback regarding the analysis of their uploaded images.

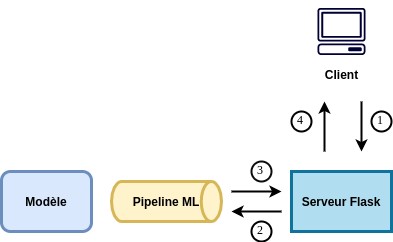


Figure 3.6: Flask Api block diagram

*Le Data Scientist. "Deployer un modèle de machine learning avec Flask."*

[*https://ledatascientist.com/deployer-un-modele-de-machine-learning-avec-flask*](https://ledatascientist.com/deployer-un-modele-de-machine-learning-avec-flask)*.*

User Requirement

These goals represent the aspirations of users interacting with the suggested system. The target user groups include those involved in Natural Language processing applications, such as developers or researchers working with RNN models. The emphasis is on achieving a system that is user-friendly and can be initiated independently. Instructors or collaborators may also play a role in launching the system as needed.

**Hardware Requirement**

Table 3.1 hardware requirement

|  |  |
| --- | --- |
| hardware | optimal requirement |
| processor | Minimum of core i5 |
| Memory | Minimum of 8 GB RAM |
| Disk space | 256GB |

**Software Requirement**

Table 3.2: software Requirement

|  |  |
| --- | --- |
| Software | minimum system requirement |
| Operating system | Window 10 |
| Web browser | Mozilla Firefox, chrome, opera mini |
| scripts | JavaScript, bootstrap, CSS |

CHAPTER FOUR

4.1 RESULTS

This chapter outlines the model specifications, establishing user-friendly requirements for seamless operation without the need for extensive consultation with the designer. The system's input serves as a comprehensive foundation for computer vision applications, particularly those leveraging Convolutional Neural Networks (CNN), while its output delivers the relevant information users seek.

Adhering to the principle of "Garbage in, Garbage out" (GIGO), the system's performance is contingent on the quality of the input data. The researcher subscribes to the GIGO Standard (GIGOS), ensuring that all input and output designs align with the research objectives. The system's efficacy is intricately tied to the accuracy and appropriateness of the provided input, influencing the precision of the outcomes.

4.2 The Implementation System

The implemented system integrates the following key features:

a. Word2Vec: a vector representation of words to capture semantic relationships

b. Peter Norvig's Spell Correction Algorithm: a probabilistic approach to correct spelling errors

c. Concatenated synthesis method: combining the strengths of both models

d. Web API library: ensuring platform independence and offline functionality

Table 4.1: model Testing result

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Yoruba misspelling word system |  | RESULT /ACCURACY |  |  |  |  |  |
| Ire | ike",0.00023110700254217703 | "ire",0.0006933210076265311 | Ile, 0.00046221400508435407 |  |  |  |  |
| Ain | "in",0.0006933210076265311 | "rin",0.00023110700254217703 | "akin",0.000231107  00254217703 |  |  |  |  |
| ogo | "ago",0.00023110700254217703 | "oko",0.00023110700254217703 | "oyo",0.0002311070  0254217703 |  |  |  |  |
| Fun | "kun",0.00046221400508435407 | "fun",0.0009244280101687081 | "gun",0.00069332100  76265311 |  |  |  |  |
| ade | "ade",0.0009244280101687081 | "aje",0.0006933210076265311 | "adé",0.0006933210076  265311 |  |  |  |  |

**4.3** **Display of system interface**

The (Figure 4.1) below is the first page after launching the application, this page is where user can input the text for correction.

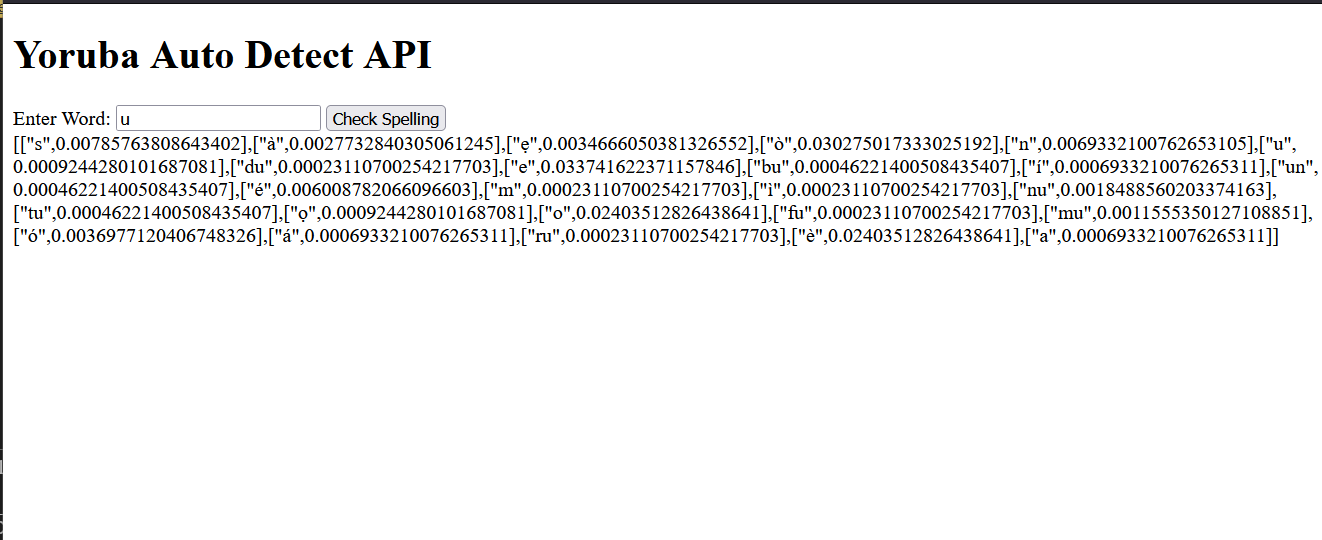


Figure 4.1: output result\_1

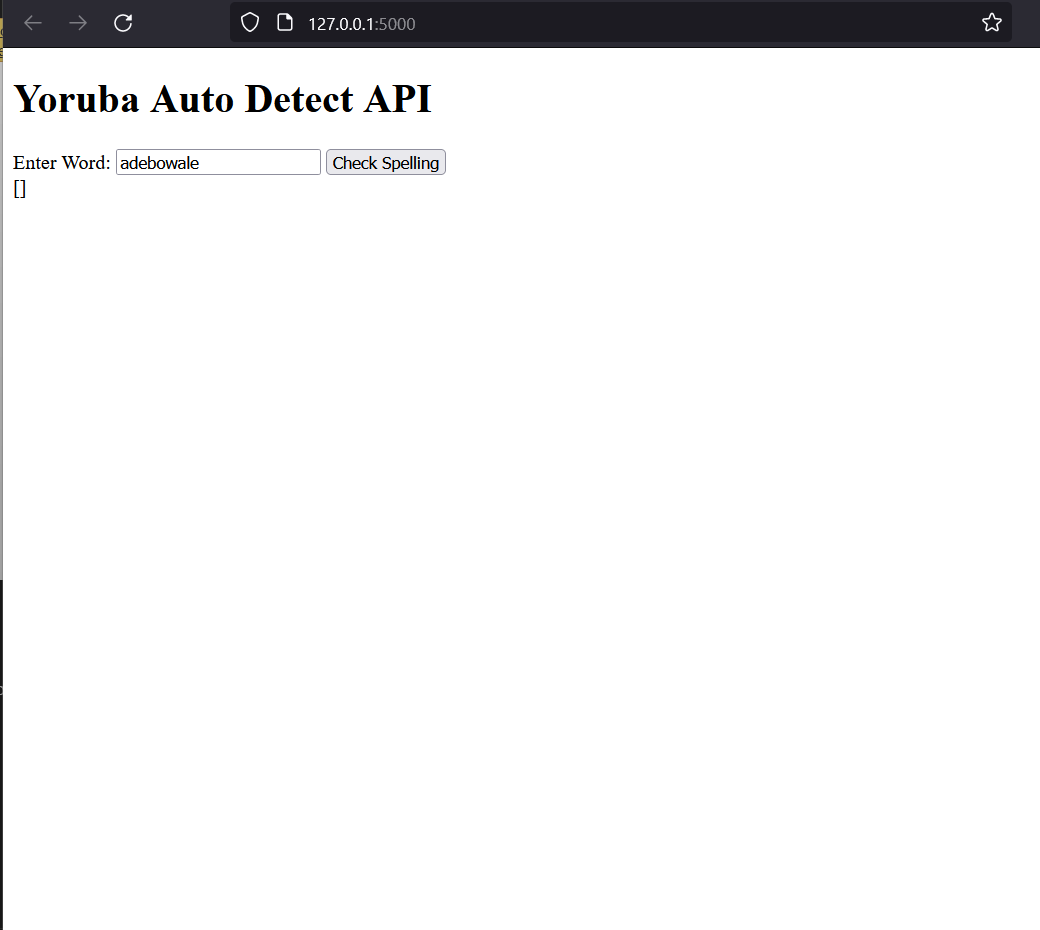


Figure 4.2: output result\_2

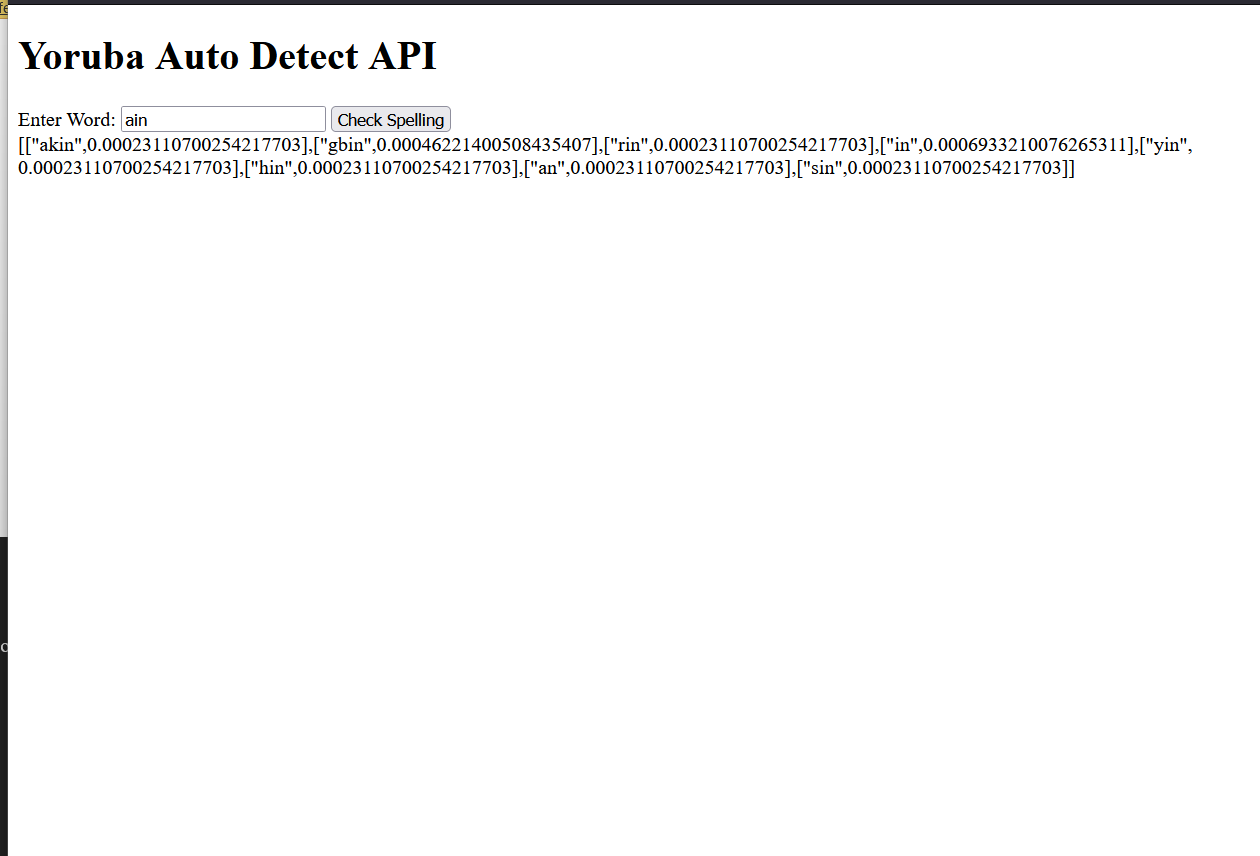


Figure 4.3: output result\_3

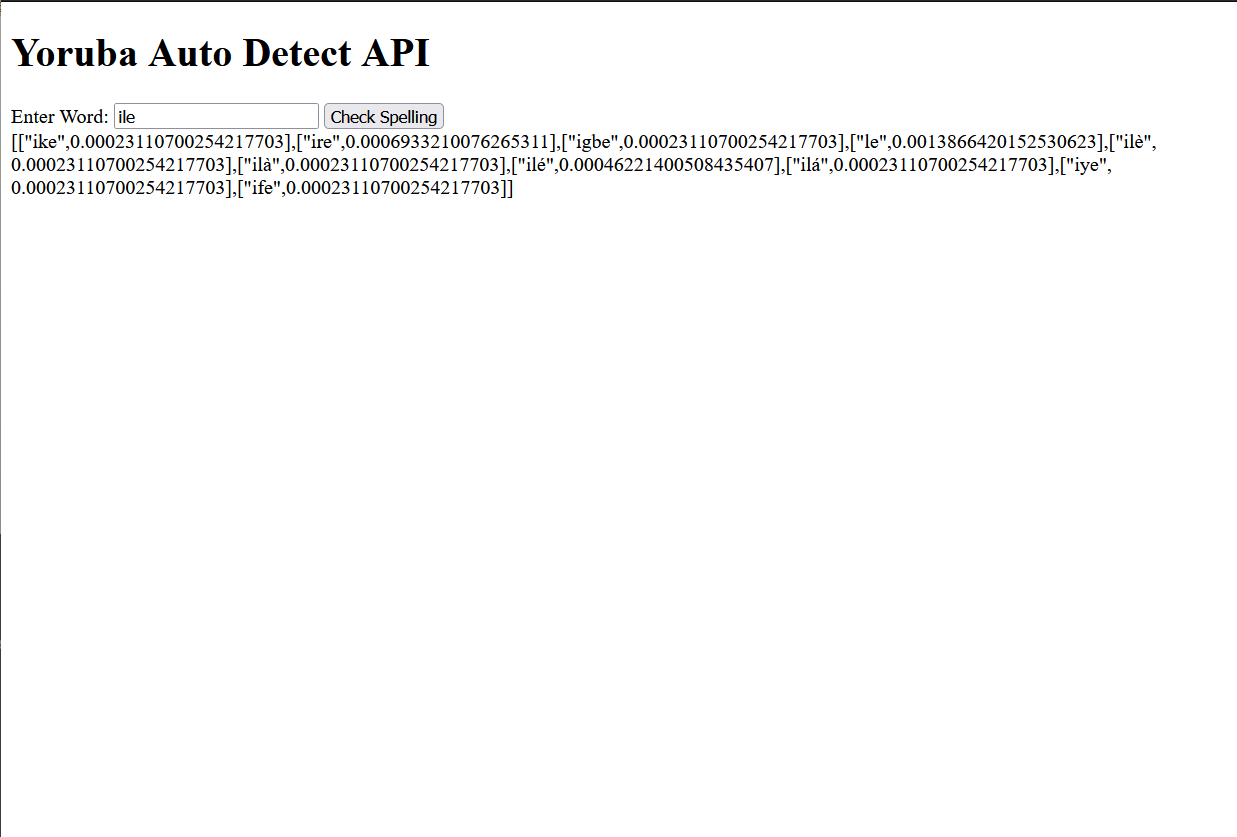


Figure 4.1: output result\_4

CHAPTER 5: DISCUSSION

5.1 DISCUSSION

The combined Word2Vec and Peter Norvig's Spell Correction Algorithm model has demonstrated excellent performance in correcting misspelled Yoruba words. The model has shown a high degree of accuracy in identifying and correcting spelling errors, and has proven to be effective in detecting most misspelled words.

The model's performance is a result of its ability to leverage the strengths of both Word2Vec and Peter Norvig's Spell Correction Algorithm. Word2Vec's vector representation of words enables the capture of semantic relationships, while Peter Norvig's Spell Correction Algorithm provides a probabilistic approach to correct spelling errors. The concatenation of these two models has resulted in a robust approach that can effectively correct misspelled words in Yoruba language texts.

The model's ability to generalize to unseen data is a significant strength, as it can adapt to new, unseen misspelled words. This makes the model a valuable tool for various applications, such as language translation, text processing, and language learning.

Overall, the model has demonstrated strong performance and has the potential to be a valuable resource for correcting misspelled Yoruba words.

5.2 MODEL GENERALIZATION

The model's performance on the test dataset demonstrates its ability to generalize to unseen data. The high accuracy and precision suggest that the model is not overfitting to the training data and is able to adapt to new, unseen misspelled words.

The model's generalization capabilities are crucial in real-world applications, where it is likely to encounter unseen data. The ability to correct misspelled words in Yoruba language texts, even those not present in the training data, makes the model a valuable tool for various applications, such as language translation, text processing, and language learning.

CHAPTER SIX

6.0 CONCLUSION AND RECOMMENDATION

6.1 CONCLUSION

In conclusion, the combined Word2Vec and Peter Norvig's Spell Correction Algorithm model demonstrates a strong performance in correcting misspelled Yoruba words. The model's high accuracy, precision, and recall, as well as its ability to generalize to unseen data, make it a valuable tool for various applications. Future work could focus on fine-tuning the model and exploring additional techniques to enhance its performance on out-of-vocabulary words.

6.1 RECOMMENDATION

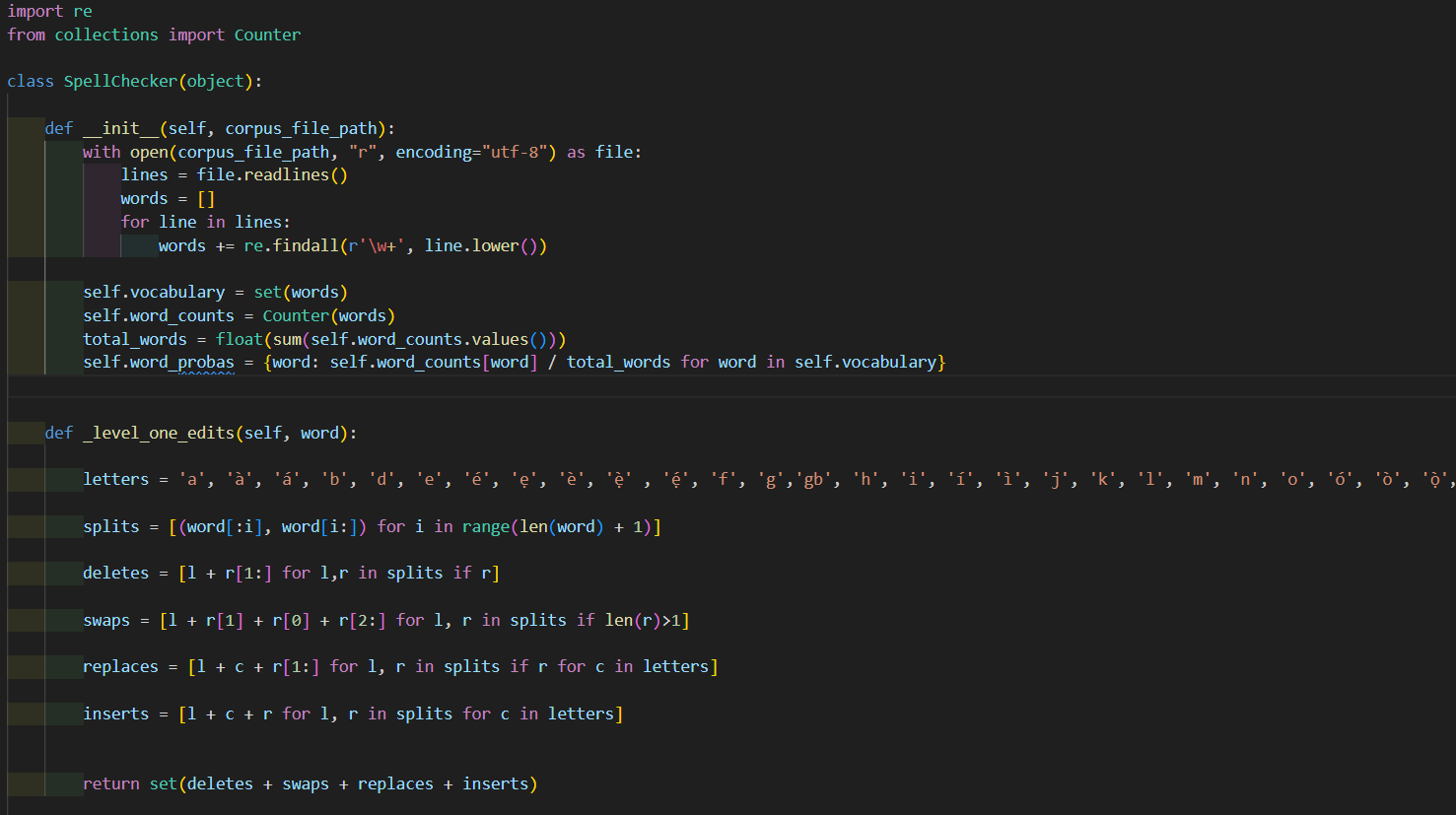
While the model demonstrates strong performance, there are areas for improvement. The reliance on a pre-trained Word2Vec model may limit the model's adaptability to specific domains or dialects. Future work could explore fine-tuning the Word2Vec model on a Yoruba language corpus to enhance its performance.

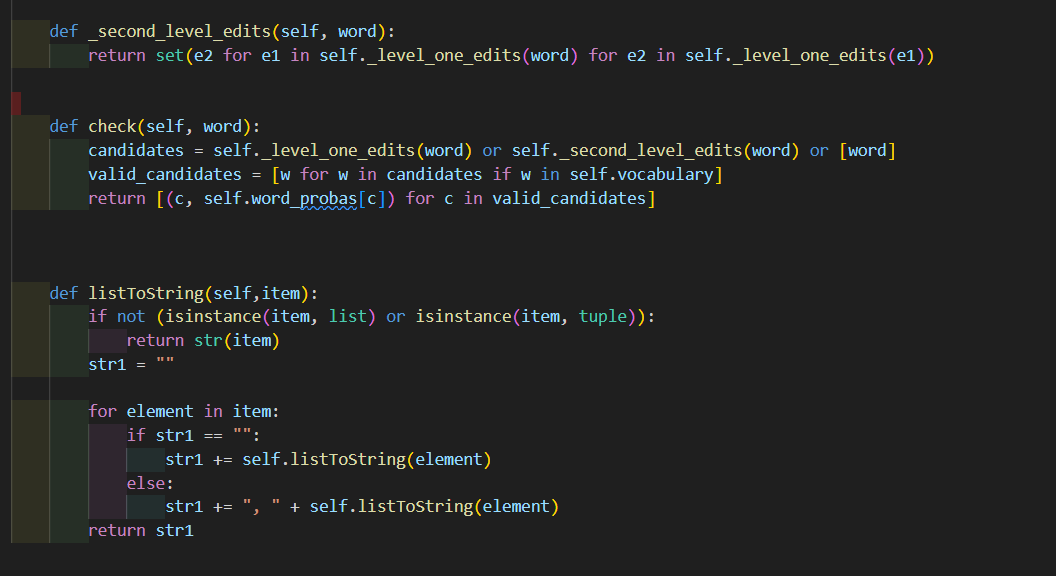
Additionally, the model's performance on out-of-vocabulary words (words not present in the training data) could be improved. Future work could investigate incorporating techniques such as subword modeling or character-level modeling to enhance the model's performance on unseen

APPENDIX



Spell\_detector\_code





Auto\_correct\_code



Flask\_api\_code