Writing assistance report

Created	@November 5, 2024 6:06 AM
Class	JetBrains

This report compares the performance of four different spell-checking tools—PySpellChecker, TextBlob, a fine-tuned transformer model, and OpenAI's GPT-3.5—using accuracy, average Levenshtein distance, precision, recall, and F1 score. These metrics provide a well-rounded view of each tool's effectiveness on a sample of 70 test cases. The tools vary significantly in their approaches to spell correction, and each offers distinct strengths and weaknesses. The evaluation was ran multiple times and the scores were compared between runs.

To evaluate the effectiveness of various spell-checking tools, I designed a systematic approach that focused on dataset preparation, model selection, metric evaluation, and analysis.

1. **Dataset Preparation**:
    - **Selection of Datasets**: I utilized a combined dataset of misspelled words and their corresponding correct forms. This dataset included both a synthetic set generated through data augmentation and a publicly available misspelled dataset. By combining different sources, I ensured the dataset captured a diverse range of spelling errors, from minor typographical mistakes to complex misspellings.
    - **Data Sampling**: To make the evaluation manageable and consistent across different tools, I sampled 70 examples from the combined dataset for evaluation. This sample size was chosen to allow for comprehensive testing without introducing excessive processing time.
2. **Tool and Model Selection**:
    - **Choice of Spell Checkers**: I selected four distinct spell-checking tools: PySpellChecker, TextBlob, a fine-tuned transformer model, and OpenAI's GPT-3.5. Each tool represents a different approach to spell-checking, allowing us to compare dictionary-based, probabilistic, transformer-based, and large language model methods.
    - **Fine-Tuned Transformer Model**: For the transformer model, I chose a grammar-focused model available on Hugging Face (`pszemraj/grammar-`

`synthesis-small`). Although this model was not exclusively trained on spelling correction, it provided insights into how a language model fine-tuned for grammar correction might perform in spelling tasks.

3. **Implementation**:
    - **Script Design**: The code was structured to load and preprocess the dataset, run each spell-checking function on the dataset, and calculate the metrics for each tool. I implemented a function that generates a CSV file containing the evaluation metrics for each model, allowing me to track and compare their performance easily.
    - **Performance Considerations**: I also included handling for cases where corrections might be `None` or ambiguous, ensuring that evaluations did not introduce errors due to missing or invalid values. Additionally, the models with higher latency (fine-tuned transformer and GPT-3.5) were highlighted for further consideration, as runtime impacts the suitability of these models in real-time applications.

Tool-by-Tool Analysis

1. **PySpellChecker**
    - **Metrics Observed**: PySpellChecker achieved an accuracy of 77% and a low Levenshtein distance of 0.44, indicating it was able to make accurate corrections with minimal deviation. However, its precision (0.5), recall (0.11), and F1 score (0.19) were relatively low, indicating inconsistent performance in handling nuanced cases.
    - **Strengths**: PySpellChecker is effective at identifying and correcting common misspellings due to its dictionary-based approach. Its relatively high accuracy and low Levenshtein score suggest it is reliable for simple corrections.
    - **Weaknesses**: The low recall and F1 score imply that PySpellChecker missed many more complex errors or struggled in cases where context was required. This aligns with its limited ability to understand the surrounding text context.
    - **Ideal Use Case**: PySpellChecker is best suited for applications that require quick and basic spell-checking without the need for context, such as form validation or single-word corrections.

2. **TextBlob**

   - **Metrics Observed**: TextBlob displayed comparable performance to PySpellChecker, with an accuracy of 73% and an average Levenshtein distance of 0.5. Its precision (0.5), recall (0.13), and F1 score (0.21) indicate a slight improvement in handling complex cases, but it still struggled with contextual relevance.
   - **Strengths**: TextBlob leverages a probabilistic approach that allows it to handle a broader range of spelling variations, making it somewhat more versatile than PySpellChecker for nuanced text.
   - **Weaknesses**: Similar to PySpellChecker, TextBlob's lower recall and F1 scores suggest it struggles with contextual accuracy, especially in more specialized text. Additionally, its performance dips with domain-specific terminology.
   - **Ideal Use Case**: TextBlob is useful for slightly more complex text-processing tasks than PySpellChecker, making it suitable for entry-level content editing or applications where moderate complexity is expected. However, when comparing the different test runs, the differences between the scores for TextBlob and PySpellChecker were quite minimal.

3. **Fine-tuned Transformer Model** (`pszemraj/grammar-synthesis-small`)

   - **Metrics Observed**: The fine-tuned model achieved a precision, recall, and F1 score of 1.0 across all cases, which may imply overfitting or a lack of nuanced evaluation on diverse errors. However, its accuracy was zero, and the extremely high Levenshtein distance of 23.44 indicates significant deviation from expected corrections.
   - **Strengths**: The model has the potential for high precision and recall when tuned on spelling-specific data, suggesting it may be valuable if adjusted properly.
   - **Weaknesses**: Currently, the model's extreme Levenshtein distance and low accuracy show it fails to provide useful corrections for misspellings, possibly due to its training focus on grammar rather than spelling. The model struggled with uninitialized weights, which likely contributed to its low performance metrics. This highlights the necessity of task-specific fine-tuning, as generic models may underperform without proper adaptation. Furthermore, this model took by far the most amount

of time to run, implying computational complexity stemming from the transformer-based processing. This could be acceptable if the model is applied slightly differently (e.g., for grammar), or the weights are initialized well.

- **Ideal Use Case**: With further fine-tuning on spelling-focused datasets, this model could serve more advanced text-processing applications. Currently, its application is limited due to its lack of accuracy in spelling corrections.

4. **OpenAI GPT-3.5**

- **Metrics Observed**: GPT-3.5 achieved an accuracy of 77% and an average Levenshtein distance of 0.59, with precision, recall, and F1 scores similar to PySpellChecker. Its balanced metrics across most categories suggest that it is generally accurate in correcting misspellings.
- **Strengths**: GPT-3.5's ability to contextualize corrections allows it to handle more complex or nuanced errors. Its strong balance of accuracy and Levenshtein distance demonstrates a capability to perform reliably across a diverse set of errors.
- **Weaknesses**: The primary limitation of GPT-3.5 is its operational cost, making it less practical for large-scale, low-budget applications. Additionally, like simpler tools, its F1 score suggests it could improve in handling varied cases. Furthermore, like the fine-tuned transformer model, GPT-3.5 took considerably longer to process the dataset compared to the simpler spell checkers, which is a common cost associated with LLMs.
- **Ideal Use Case**: GPT-3.5 is best suited for high-stakes or high-accuracy applications, such as professional text editing where contextual understanding is essential.

Challenges Encountered

1. **Contextual Limitations of Simpler Models**:

- Both PySpellChecker and TextBlob had difficulty with contextual spell-checking. Their limited recall and F1 scores highlighted challenges in handling cases where the correct spelling depended on surrounding context. This restricted their usefulness in specialized or nuanced text.

2. **Transformer Model Overfitting**:
    - The fine-tuned model's perfect recall and F1 score, despite low accuracy and high Levenshtein distance, suggest overfitting or lack of suitability for spelling correction. This reveals the need for spelling-specific fine-tuning, as grammar-focused training hindered its ability to address spelling errors effectively. Furthermore, the transformer's slower processing speed affected its scalability and usability.
3. **Limitations of Large Language Models**:
    - While GPT-3.5 exhibited balanced metrics, its API cost and processing time limit its scalability. Additionally, although it achieved high accuracy, its moderate F1 score suggests that it occasionally failed in cases requiring specific contextual corrections, hinting at areas for further improvement in specialized applications.

Conclusions and Recommendations

- **Basic Spell-Checking with Low Complexity Needs**: For tasks with minimal contextual needs, PySpellChecker and TextBlob offer a quick, efficient solution with reliable accuracy. These tools are ideal for use cases with single-word or low-context spell-checking requirements.
- **Improvement Potential in Transformer Models**: The fine-tuned transformer model, while currently ineffective for spelling correction, could benefit from further training on spelling-specific data. Transformer models may offer an ideal balance of accuracy and flexibility if properly tailored to spelling nuances.
- **High-Quality Correction for Complex Text**: GPT-3.5, while effective in handling complex cases, presents a high-cost solution. For scenarios where quality is prioritized over budget, such as professional editing or specialized content creation, it remains the best option. Further fine-tuning or model adjustments could reduce its dependency on cost-intensive API calls while maintaining high-quality output.