DSGA 1011: Assignment 4

Eshika Janbandhu
ej2485

November 18, 2025

Q0. 1.

https://github.com/jeshika30/NLP-hw4.git

Q2. 1.

I designed a transformation that simulates realistic user noise by combining **synonym replacement** (using WordNet) and **typing mistakes based on the QWERTY keyboard layout**. The transformation operates on each review independently and only modifies the text field; the sentiment label is left unchanged.

Step-by-step algorithm:

1. **Tokenization:** Each review's text is tokenized into word-level tokens using NLTK's `word_tokenize`.
2. **Iterate through tokens:** For each token, if it is not alphabetic, leave it unchanged. Otherwise, apply a transformation based on a random draw.
3. **Random decision per word:**
   - **Synonym replacement (10% chance):** Query WordNet for synonyms, select one if available, preserving capitalization.
   - **Typo introduction (next 10% chance):** Pick a random character in the word, replace it with a nearby key on a QWERTY keyboard.
   - **No change (remaining 80% chance):** Keep the original token.
4. **Detokenization:** Use TreebankWordDetokenizer to reconstruct the text and assign it back to `example["text"]`.
5. **Labels:** The label `example["label"]` remains unchanged.

Rationale:

- **Natural typing errors:** Simulates realistic user typos without producing gibberish.
- **Paraphrasing via synonyms:** Models natural vocabulary variation while preserving sentiment.
- **Maintains sentence coherence:** Only ∼20% of tokens are modified, preserving grammar and readability.
- **Realistic distribution shift:** Tests model robustness to real-world variations in user input.

Q3. 1

Report & Analysis

- **Accuracy:**
  - Original test set: 0.92452
  - Transformed test set: 0.87612
- **Analysis:**
  1. The model's performance on the transformed test set decreased compared to the original test set. This indicates that the introduced typos and synonym replacements successfully created a distribution shift that challenged the model.
  2. Data augmentation was not applied during training in this evaluation, so the model's accuracy on the original test data remains high (0.9006). The drop in accuracy on the transformed set (to 0.8416) reflects the model's sensitivity to minor input variations.
- **Intuitive explanation:** The model likely relies on exact word forms seen during training. Small changes, such as typos or synonyms, can reduce its confidence or trigger misclassification. Data augmentation during training could help the model become more robust to these perturbations by exposing it to more varied inputs.
- **Limitation:** A key limitation of this transformation is that it only simulates minor noise and simple synonym variations. More complex out-of-distribution shifts (e.g., sentence reordering, idiomatic expressions, or domain changes) are not captured, so the model may still fail on realistic yet semantically valid but structurally different inputs.

Part II. Q4

Before Pre-processing		
Statistics Name	Train	Dev
Number of examples	4225	466
Mean sentence length	10.96	10.91
Mean SQL query length	60.90	58.90
Vocabulary size (natural language)	868	444
Vocabulary size (SQL)	644	393

Table 1: Data statistics before any pre-processing.

After Pre-processing (T5 fine-tuned model)		
Statistics Name	Train	Dev
Mean tokenized sentence length	18.10	18.07
Mean tokenized SQL length	192.73	188.59
Vocabulary size (encoder tokens)	792	466
Vocabulary size (decoder tokens)	543	386

Table 2: Data statistics after pre-processing.

Q5

Design choice	Description
Data processing	We used the provided natural language and SQL query pairs without additional cleaning. The dataset was split into train, dev, and test as given. For each example, the input is the natural language query, and the target output is the corresponding SQL query. We did not perform any complex augmentation or filtering.
Tokenization	We used the default pretrained T5Tokenizer from Huggingface `google/t5-small` for both encoder and decoder inputs. The natural language queries were tokenized as encoder inputs, and the SQL queries were tokenized as decoder targets. We did not modify or customize the tokenizer. This ensured consistent tokenization aligned with the pretrained model vocabulary and subword segmentation scheme.
Architecture	We fine-tuned the full `t5-small` encoder-decoder model, which consists of 6 layers each in the encoder and decoder. No layers were frozen; all parameters were updated during training. We used the pretrained weights as initialization and trained the model end-to-end to adapt it to the Text-to-SQL task.
Hyperparameters	The key hyperparameters for training were: learning rate 3e-4, batch size 16, 10 epochs with early stopping on validation loss, AdamW optimizer with weight decay, maximum input length 128 tokens, and maximum output length 256 tokens. These settings were chosen based on common fine-tuning recommendations for T5 and hardware constraints.

Table 3: Details of the best-performing T5 model configurations (fine-tuned).

Q6.

System	Query EM	F1 score
Dev Results		
T5 fine-tuned		
Full model	61.59	63.20
Test Results		
T5 fine-tuning	60.94	63.39

Table 4: Development and test results.

Quantitative Results:

Qualitative Error Analysis:

Q7.

Provide a link to a google drive which contains a model checkpoint used to generate outputs you have submitted.

Error Type	**Example of Error**	**Error Description**
Syntax Errors	`SELECT DISTINCT flight_1.flight_id FROM flight flight_1 WHERE flight_1.departure_time 900`	The generated SQL query is incomplete or contains syntax errors (e.g., missing operators, misplaced parentheses).
Invalid Column Names	`SELECT DISTINCT fare_1.fare_id FROM fare fare_1, flight_fare flight_fare_1, flight flight_1 WHERE fare_1.fare_id = flight_fare_1.fare_id`	The SQL query references non-existent or incorrect columns, leading to database errors.
Incomplete SQL Queries	`SELECT DISTINCT flight_1.flight_id FROM flight flight_1, airport_service airport_service_1 WHERE flight_1.from_airport = 'NY'`	The model generates incomplete SQL queries that do not provide the full set of conditions or joins necessary for execution.
Logical Errors	`SELECT DISTINCT fare_1.fare_id FROM fare fare_1, flight_fare flight_fare_1, flight flight_1 WHERE fare_1.fare_id = flight_fare_1.fare_id AND flight_fare_1.flight_id = flight_1.arrival_airport`	The generated SQL query logically doesn't make sense, such as selecting the wrong table, columns, or incorrect comparisons.
Misplaced Conditions	`SELECT DISTINCT flight_1.flight_id FROM flight flight_1 WHERE flight_1.from_airport = 'NY' AND flight_1.arrival_time 1800`	The query has misplaced conditions, e.g., missing logical operators or incorrect grouping of conditions.
Ambiguous or Vague Queries	`SELECT DISTINCT flight_1.flight_id FROM flight flight_1 WHERE flight_1.departure_time BETWEEN 1200 AND 1800`	The query does not provide enough information to identify what specific data is being asked for or is too general.

Table 5: Qualitative Error Analysis