Predictive Text Model - Code Overview

Sarim Rizvi

Contents

Introduction

This project builds a predictive text model using n-gram language modeling techniques. The model uses Laplace smoothing and a backoff mechanism and supports up to 4-grams. A Shiny application is also included to demonstrate the model interactively.

The model was trained on a cleaned English corpus and evaluated using accuracy, perplexity, and runtime on randomly selected sentences from the `crude` dataset.

Data Preparation

The raw text data was preprocessed as follows:

- Converted to lowercase
- Removed punctuation and numbers
- Removed extra whitespace
- Tokenized into sentences and then into words
- Created n-grams (1- to 4-grams) using `tokenizers::tokenize_ngrams`

Example for generating bigrams:

```
bigram_df <- sampled_df %>%
  unnest_tokens(output = "bigram", input = text, token = "ngrams", n = 2) %>%
  count(bigram, sort = TRUE)
```

The final n-gram data frames were saved as .rds files for reuse.

Model

The core prediction function is `next_word_pred_opt_prob_all(input_text)`. It performs the following steps:

- Tokenizes and cleans the input
- Attempts to find a matching quadgram
- Falls back to trigram, bigram, or unigram as needed
- Applies Laplace smoothing

Probabilities are calculated using the Laplace formula:

```
P(wi | history) = (frequency + 1) / (sum(frequencies) + vocabulary_size)
```

Optimisation Used

- All n-gram data frames were converted to `data.table` objects for faster filtering and lookups.
- Keyed joins were created to quickly match rows based on input words(s).
- To reduce memory usage, rarely occurring n-grams (<2) were removed.

Evaluation

Model performance is measured using:

1. Accuracy: Whether the actual next word appears in the top-N predictions
2. Perplexity: How well the model predicts sequences overall
3. Runtime: Time taken to generate predictions

Evaluation functions include:

```
get_accuracy(text, model_func)
get_perplexity(text, model_func, vocab_size)
evaluate_model_with_timing(sentences, model_func, vocab_size)
```

To generate test data:

```
test_sentences <- get_test_sentences(n = 10) #10 sentences
```

Model Comparison Summary

The table below summarizes the average performance of the base and optimized models evaluated on a set of test sentences.

Table 1: Comparison of Base vs. Optimised Model

Model	Accuracy	Perplexity	Time
Base	0.1142	5.7954	0.7586
Optimised	0.1168	26.3711	0.0541

This comparison highlights the following:

- The optimized model significantly improves prediction time.
- It maintains similar accuracy to the base model.
- The optimized model has a higher perplexity, which is expected due to pruning and more aggressive fallback behavior.

Shiny App

The application (in `prediction_app.R`) provides a minimal interface to enter a phrase and receive next-word predictions.

Features:

- Uses only the optimized prediction model.
- Displays top-N predictions with probabilities.
- Combines predictions across levels to return at least 3 predictions.