Use Cases of Large Language Models

S. Mason Garrison

Use Cases of Large Language Models

Roadmap

1. Text Understanding and Analysis
2. Text Generation and Transformation
3. Question Answering and Information Retrieval

1. Text Understanding and Analysis

Text Understanding and Analysis

- Extract insights from text data
- Applications:
  - Text classification
  - Named entity recognition
  - Text similarity and clustering
  - Anomaly detection

Text Classification

- Categorize text into predefined classes
- Applications:
  - Sentiment analysis
  - Topic classification
  - Intent detection in chatbots
- Fine-tuned LLMs like BERT achieve state-of-the-art results
  - `Tensorflow has really solid tutorials`

Named Entity Recognition

- Identify and classify named entities in within unstructured text.
- Use cases:
  - Information extraction
  - Building knowledge graphs
  - Enhancing search capabilities
- Models like `SpaCy` and `BERT` can be fine-tuned for NER

Text Similarity and Clustering

- Measure semantic similarity between texts
- Use cases:
  - Document de-duplication
  - Content recommendation systems
- Sentence transformers like `SBERT` are effective for this

Anomaly Detection in Text Data

- Identify unusual patterns or outliers in text
- Use cases:
  - Fraud detection in financial documents
  - Identifying errors in medical records
- Autoencoder architectures with LLMs can be used
- **Check out this** `Paper that talks more about this`

R Example: Text Classification (Sentiment Analysis)

```r
library(tidytext)
library(dplyr)
library(ggplot2)

# Sample text data
texts <- c(
  "I love this product! It's amazing.",
  "This is terrible. I hate it.",
  "It's okay, nothing special.",
  "Wow, absolutely fantastic experience!",
  "Disappointed with the quality."
)

# Create a tibble
df <- tibble(text = texts)
```

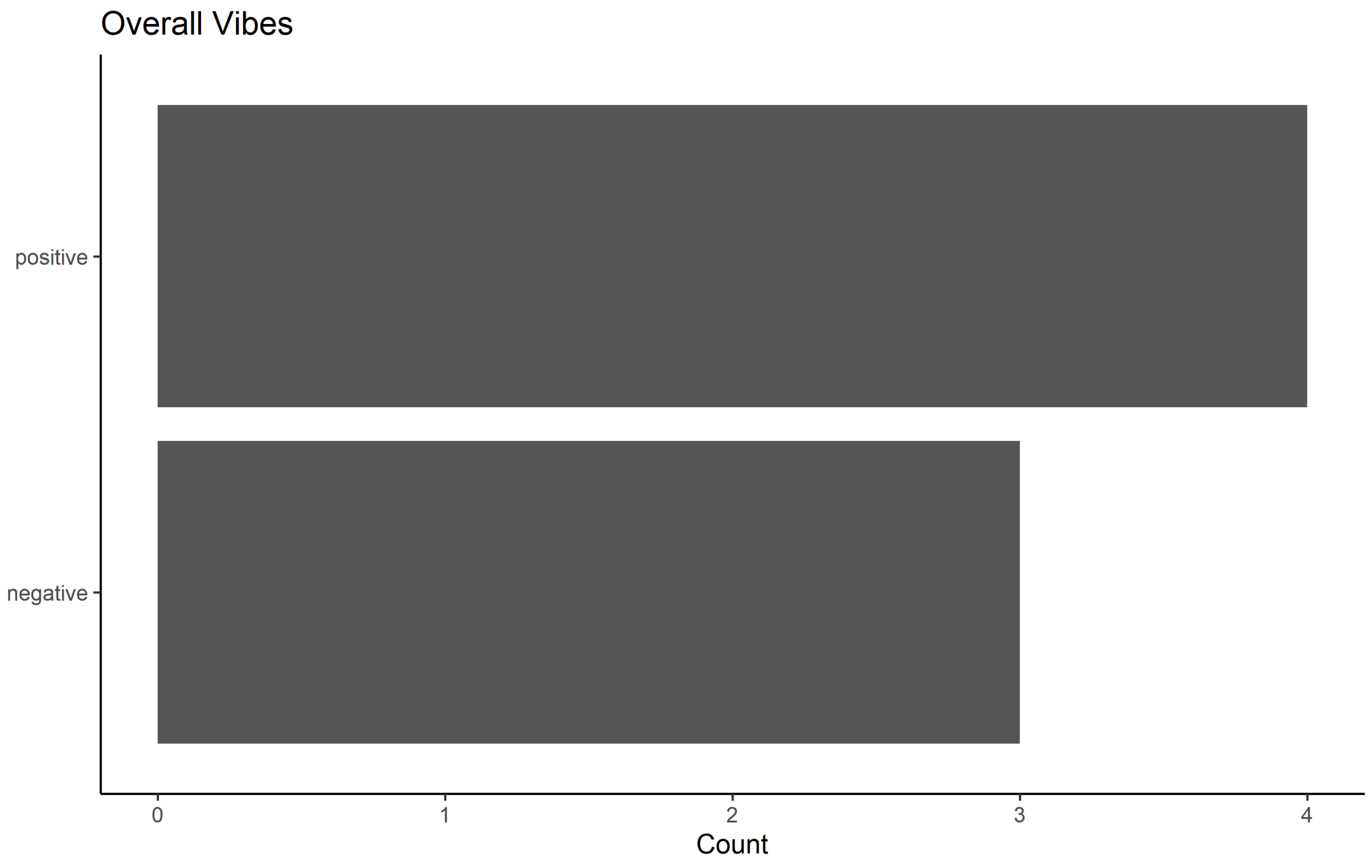
Tokenize and Perform Sentiment Analysis

```r
sentiment_scores <- df %>%
  unnest_tokens(word, text) %>%
  inner_join(get_sentiments("bing")) %>%
  count(sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment_score = positive - negative)

sentiment_scores
```

```
## # A tibble: 1 × 3
##   negative positive sentiment_score
##      <dbl>    <dbl>           <dbl>
## 1        3        4               1
```

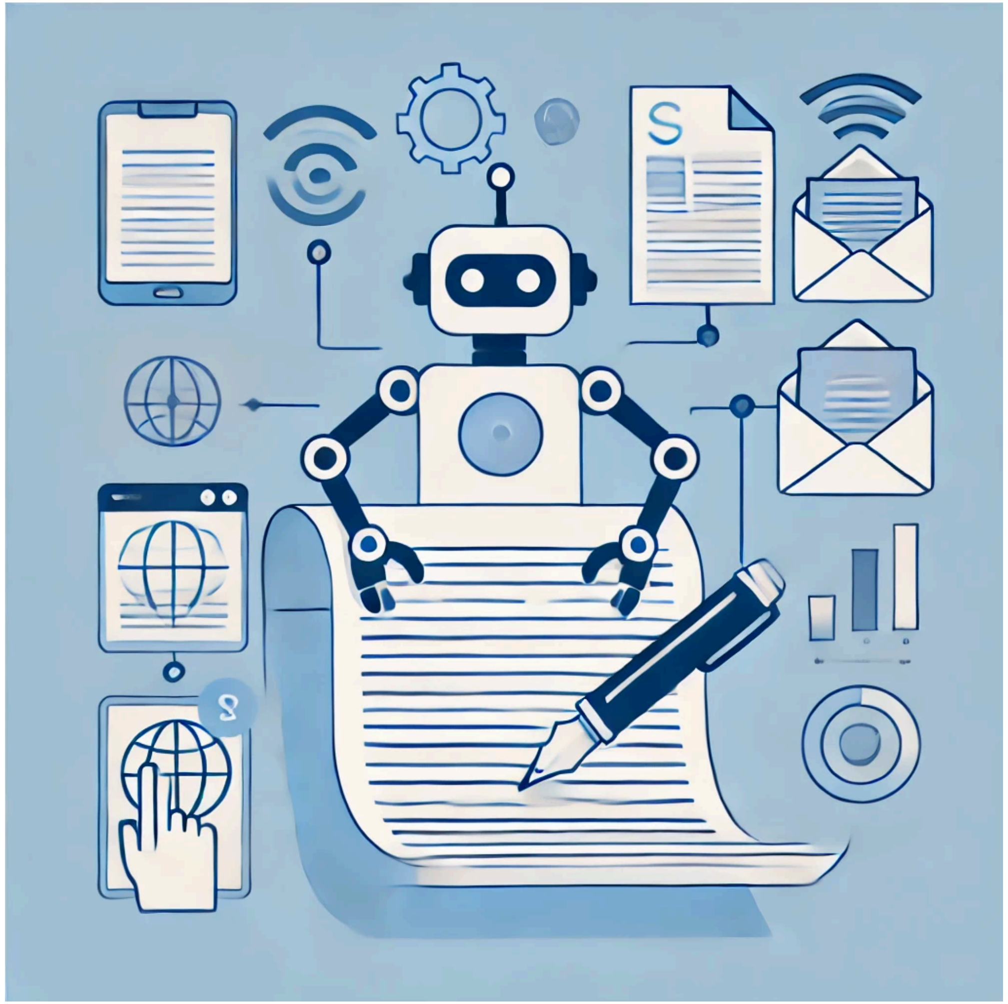
Visualize Sentiment Analysis Results



2. Text Generation and Transformation

Text Generation

- LLMs can generate human-like text on any topic
- Applications:
  - Automated content creation
  - Chatbots and conversational AI
  - Code generation
- Examples:
  - GPT-3 for natural language generation
  - GitHub Copilot for code completion



Summarization

- Condense long documents into brief summaries
- Use cases:
  - Summarizing research papers
  - Creating article abstracts
  - Distilling key points from large datasets
- Models like BART and T5 specialize in summarization tasks

Machine Translation

- Translate text between languages
- Applications:
  - Localization of content
  - Cross-lingual information retrieval
- Models like mT5 specialize in multilingual tasks

Data Augmentation

- Generate synthetic data to augment training sets
- Applications:
  - Addressing class imbalance
  - Creating larger datasets for model training
- GPT models can generate realistic synthetic data

R Example: Text Generation (Simple Markov Chain)

```r
library(tidytext)
library(dplyr)
library(stringr)

# Sample text
text <- "The quick brown fox jumps over the lazy dog. The dog barks at the fox. The fox runs away quickly."

# Tokenize and create word pairs
word_pairs <- tibble(text = text) %>%
  unnest_tokens(word, text) %>%
  mutate(next_word = lead(word)) %>%
  na.omit()

word_pairs
```

```
## # A tibble: 19 × 2
##   word  next_word
##   <chr> <chr>
## 1 the   quick
## 2 quick brown
## 3 brown fox
## 4 fox   jumps
## 5 jumps over
## 6 over  the
## # ℹ 13 more rows
```

Create Markov Chain

```
# Create a simple Markov chain
markov_chain <- word_pairs %>%
  group_by(word) %>%
  summarise(next_words = list(next_word))

markov_chain
```

```
## # A tibble: 12 × 2
##   word  next_words
##   <chr> <list>
## 1 at    <chr [1]>
## 2 away  <chr [1]>
## 3 barks <chr [1]>
## 4 brown <chr [1]>
## 5 dog   <chr [2]>
## 6 fox   <chr [3]>
## # ℹ 6 more rows
```

Text Generation Function

```
generate_text <- function(start_word, length = 10) {
  result <- start_word
  current_word <- start_word

  for (i in 1:length) {
    next_word_options <- markov_chain %>%
      filter(word == current_word) %>%
      pull(next_words) %>%
      unlist()

    if (length(next_word_options) == 0) break

    next_word <- sample(next_word_options, 1)
    result <- c(result, next_word)
    current_word <- next_word
  }

  str_c(result, collapse = " ")
}
```

Generate Text

```r
# Generate a sentence
cat(generate_text("the", 8))
```

```
## the fox the fox runs away quickly
```

3. Question Answering and Information Retrieval

Text-to-SQL

- Generate SQL queries from natural language
- Use cases:
  - Database querying for non-technical users
  - Automating data analysis workflows
- GPT-3 and other LLMs can be fine-tuned for this task

Question Answering

- Extract relevant answers from a given context
- Applications:
  - Automated customer support
  - Information retrieval systems
- Models like RoBERTa excel at question answering tasks

Question Answering Systems

```
library(tidytext)
library(dplyr)

context <- "The Eiffel Tower is a wrought-iron lattice tower on the Champ de Mars in Paris, France.
It is named after the engineer Gustave Eiffel, whose company designed and built the Tower."

question <- "Who designed the Eiffel Tower?"

# Tokenize context and question
context_tokens <- tibble(text = context) %>%
  unnest_tokens(word, text)

question_tokens <- tibble(text = question) %>%
  unnest_tokens(word, text)

# Simple word overlap for demonstration
matching_words <- intersect(context_tokens$word, question_tokens$word)

# Find sentence with most matching words
sentences <- tibble(text = context) %>%
  unnest_tokens(sentence, text, token = "sentences")

best_sentence <- sentences %>%
  mutate(matches = sapply(sentence, function(s) {
    sum(matching_words %in% unlist(strsplit(s, " ")))
  })) %>%
  arrange(desc(matches)) %>%
  slice(1)

print(
```

Question Answering with R: A Breakdown

Step 1: Setup

```
library(tidytext)
library(dplyr)

context <- "The Eiffel Tower is a wrought-iron lattice tower on the Champ de Mars

question <- "Who designed the Eiffel Tower?"
```

- Load necessary libraries
- Define the context (text containing the answer)
- Define the question we want to answer

Step 2: Tokenization

```
context_tokens <- tibble(text = context) %>%
  unnest_tokens(word, text)

(question_tokens <- tibble(text = question) %>%
  unnest_tokens(word, text))
```

```
## # A tibble: 5 × 1
##   word
##   <chr>
## 1 who
## 2 designed
## 3 the
## 4 eiffel
## 5 tower
```

- Break down context and question into individual words (tokens)
- `unnest_tokens()` from `tidytext` package does the heavy lifting
- Result: Two data frames with one word per row

Step 3: Word Matching

```r
matching_words <- intersect(
  context_tokens$word,
  question_tokens$word
)
```

- Find words that appear in both context and question
- `intersect()` function identifies common words
- This helps focus on potentially relevant parts of the context

Step 4: Sentence Splitting

```r
sentences <- tibble(text = context) %>%
  unnest_tokens(sentence,
                text,
                token = "sentences")
```

- Split the context into individual sentences
- Again using `unnest_tokens()`, but with `token = "sentences"`
- Prepares for sentence-level analysis

Step 5: Answer Extraction (Part 1)

```
best_sentence <- sentences %>%
  mutate(matches = sapply(sentence, function(s) {
    sum(matching_words %in% unlist(strsplit(s, " ")))
  }))
```

- Count matching words in each sentence
- `sapply()` applies the counting function to each sentence
- Creates a new column 'matches' with the count

Step 5: Answer Extraction (Part 2)

```
best_sentence <- best_sentence %>%
  arrange(desc(matches)) %>%
  slice(1)
```

- Rank sentences by number of matching words
- `arrange(desc(matches))` sorts in descending order
- `slice(1)` selects the top-ranked sentence

Step 6: Result Output

- Print the sentence most likely to contain the answer
- This sentence has the most word overlap with the question
- A simple but effective approach for basic question answering

```
print(best_sentence$sentence)
```

```
## [1] "the eiffel tower is a wrought
```

Debrief

- This method demonstrates a basic approach to question answering in R
- It relies on word overlap between question and context
- More advanced methods would involve semantic understanding and machine learning models

> How might you improve this basic question answering system?

Conclusion

- LLMs have diverse applications across data science tasks
- They excel at understanding and generating human language
- Continued research is improving their capabilities and efficiency
- R provides tools for implementing some of these techniques

Wrapping Up...