Reproducão: Towards Automating Code Review Activities

João Victor Soares de Almeida

02/07/2023

Towards Automating Code Review Activities

1. Introdução

O artigo ***T*owards Automating Code Review Activities [3]** aborda a automação das atividades de revisão de código como um meio de melhorar a eficiência e a qualidade dos processos de desenvolvimento de *software*. Os autores enfatizam a importância da revisão de código como uma prática fundamental para identificar erros, melhorar a qualidade do código e compartilhar conhecimento entre os membros da equipe de desenvolvimento. As revisões de código desempenham um papel crucial tanto na indústria quanto em projetos de código aberto, com benefícios amplamente reconhecidos, como melhoria da qualidade do código e redução da probabilidade de introdução de bugs.

No entanto, esse processo manual requer que os desenvolvedores dediquem tempo revisando o código de seus colegas de equipe, o que pode ser uma tarefa demorada e suscetível a erros humanos. Nesse contexto, a automação parcial do processo de revisão de código surge como uma abordagem promissora para mitigar esses desafios. Ao empregar técnicas de aprendizado de máquina e processamento de linguagem natural, é possível desenvolver sistemas capazes de auxiliar os desenvolvedores e revisores durante o processo de revisão de código.

O objetivo do estudo em questão é dar o primeiro passo para automatizar parcialmente o processo de revisão de código, a fim de reduzir os custos associados a essa atividade. Os pesquisadores se concentraram nas perspectivas do colaborador e do revisor, treinando duas arquiteturas de *Deep Learning* distintas. O primeiro modelo aprende as alterações de código realizadas pelos desenvolvedores durante as atividades reais de revisão de código, fornecendo ao colaborador uma versão revisada de seu código antes mesmo de ser submetido à revisão. Essa versão incorpora as transformações de código geralmente recomendadas durante as revisões. O segundo modelo fornece automaticamente ao revisor comentários sobre um código submetido, implementando os comentários expressos em linguagem natural por meio de modificações no código.

Ao automatizar parcialmente o processo de revisão de código, espera-se que os desenvolvedores e revisores economizem tempo e esforço, permitindo que se concentrem em tarefas de maior valor agregado. Além disso, a automação pode contribuir para uma maior uniformidade na aplicação de práticas recomendadas de revisão de código e proporcionar uma experiência mais consistente e eficiente para toda a equipe de desenvolvimento.

Portanto, o estudo visa explorar as possibilidades oferecidas pela automação parcial da revisão de código, avaliar a eficácia das arquiteturas de *Deep Learning* propostas e fornecer *ideias* valiosos para futuros avanços nessa área. O avanço na automação das atividades de revisão de código tem o potencial de revolucionar a maneira como o desenvolvimento de *software* é conduzido, impulsionando a eficiência, a qualidade e a colaboração no processo de desenvolvimento.

2. Metodologia

Nesta seção, descreveremos a metodologia adotada para avaliar a qualidade das traduções no contexto do artigo **Towards Automating Code Review Activities [3]**. Nosso objetivo é aplicar e comparar os

métodos *BLEU-4* e *Distância de Levenshtein* para avaliar as traduções geradas por um código de tradução automática específico.

2.1. Objetivo da Avaliação

O principal objetivo desta avaliação é medir a qualidade das abstrações geradas pelo sistema de automação de revisão de código descrito no artigo [3]. Pretende-se comparar o desempenho do sistema utilizando os métodos *BLEU-4* e *Distância de Levenshtein*, identificar possíveis discrepâncias dos métodos de avaliação e verificar qual sistema tem um resultado melhor dado um conjunto de dados para ser analisado.

2.2 Definição do Escopo

Para restringir-se ao escopo da avaliação, focaremos no sistema de revisão de código mencionado no artigo [3]. Utilizaremos um conjunto de dados de referência - disponibilizado pelos autores no Github - contendo os códigos para fins de comparação com as abstrações geradas pelo sistema.

2.3 Conjunto de Dados

Selecionaremos o conjunto de dados adequado para a avaliação - disponibilizado no repositório hospedado no Github [4] dos autores -, considerando um conjunto representativo de sentenças ou textos que abranjam diferentes contextos e níveis de complexidade.

2.4 Avaliação com o Método BLEU-4

Aplicaremos o método BLEU (*Bilingual Evaluation Understudy*) [8] para avaliar a qualidade das traduções geradas pelo sistema de tradução automática. Seguiremos as seguintes etapas:

2.4.1 Cálculo da Pontuação BLEU

Utilizaremos uma implementação construída em R para calcular a pontuação *BLEU* para cada tradução em relação ao conjunto de referência. A pontuação *BLEU* será uma medida numérica - em porcentagem - que indica a similaridade entre as traduções geradas e as traduções de referência.

2.4.2 Análise dos Resultados

Analisaremos as pontuações BLEU obtidas para cada tradução, levando em consideração a pontuação geral e as pontuações por n-gramas. Com base nessa análise, identificaremos as traduções que obtiveram pontuações mais altas e as que apresentaram menor coerência em relação às traduções de referência.

2.5 Avaliação com o Método *Levenshtein Distance*

Aplicaremos o método *LEV (Levenshtein Distance)* [7] para avaliar a qualidade das traduções geradas pelo sistema de tradução automática. Seguiremos as seguintes etapas:

2.5.1 Cálculo da Distância *Levenshtein*

Para cada tradução gerada, calcularemos a distância de edição *Levenshtein* em relação a cada tradução de referência. A distância *Levenshtein* será uma medida numérica - em porcentagem - que indica a quantidade mínima de operações (inserção, exclusão, substituição) necessárias para transformar a tradução gerada em uma tradução de referência.

2.5.2 Análise dos Resultados

Analisaremos as distâncias *Levenshtein* obtidas para cada tradução, considerando a média, a mediana e a distribuição geral das distâncias.

2.6 Comparação e Discussão dos Resultados

Compararemos os resultados obtidos pelos métodos *BLEU* e *Levenshtein*, considerando suas vantagens, limitações e aplicabilidade para avaliar a qualidade das traduções geradas pelo sistema de tradução automática. Discutiremos possíveis discrepâncias entre os resultados e avaliaremos a consistência dos métodos em relação às traduções de referência.

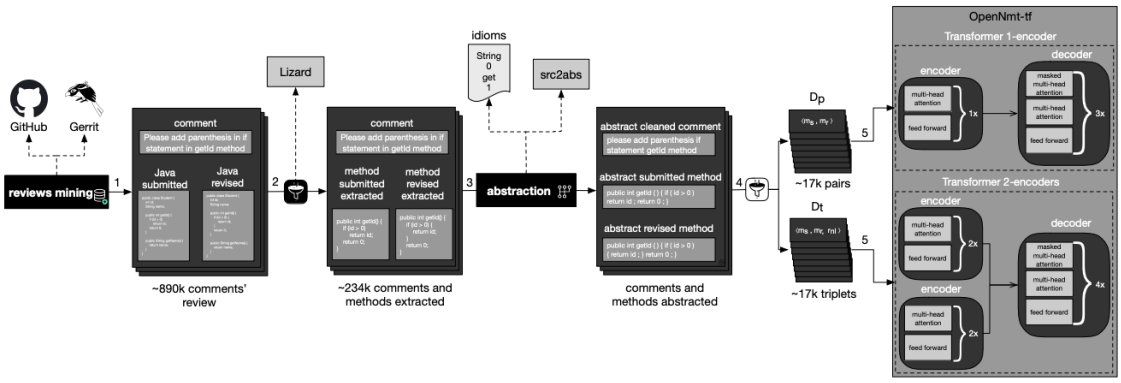
3. Reprodução

3.1. Dados e métodos

Para obtenção dos dados, os autores seguiram alguns passos, abaixo descreveremos de forma breve o processo:

3.1.1 Dados de revisão do código de mineração Nessa seção, o autor descreve o processo de mineração de dados de revisão de código no Gerrit e no GitHub [4]. O objetivo desse processo de mineração é coletar todas as informações que representam as rodadas de revisão de código realizadas no Gerrit [5] e no GitHub.

Para realizar essa tarefa, foram desenvolvidas duas ferramentas de mineração específicas para consultar sistematicamente as APIs públicas do Gerrit e do GitHub. Essa implementação dupla é necessária porque, embora ambas as plataformas forneçam suporte semelhante para revisão de código, as APIs públicas usadas para recuperar os dados são diferentes.



A saída deste processo é representada, para cada revisão rodada, por **(i)** o conjunto de arquivos de código enviados para revisão, **(ii)** os comentários recebidos neste código arquivos com informações sobre as linhas impactadas específicas (informações em nível de personagem está disponível para Gerrit), e **(iii)** os arquivos de código revisados enviados em resposta aos comentários recebidos. [1]

3.1.2 Pré-processamento de dados Nessa etapa de pré-processamento de dados, o autor descreve os seguintes processos:

1. **Extração e Abstração de Métodos**: Os arquivos Java envolvidos no processo de revisão são analisados usando a biblioteca Python Lizard. O objetivo dessa etapa é extrair os métodos de todos os arquivos. Em seguida, é adotado um processo de abstração para obter uma representação expressiva, mas limitada em vocabulário, do código-fonte. Esse processo de abstração é realizado usando a ferramenta **src2abs** e mapeia os tokens abstraídos aos tokens originais, permitindo voltar ao código-fonte original.

2. **Vinculação e Abstração de Comentários dos Revisores**: Cada comentário do revisor é associado às linhas de código específicas às quais se refere. Os comentários são vinculados a um método se as

linhas de início e fim do comentário estiverem dentro do corpo, da assinatura ou das anotações desse método. Os comentários que não podem ser vinculados a nenhum método são descartados. Além disso, os componentes de código mencionados nos comentários são abstraídos usando o mapa de abstração obtido na etapa anterior.

3. **Filtragem de Comentários Ruidosos**: Uma inspeção manual dos dados de revisão de código coletados revelou que uma porcentagem significativa de comentários de código coletados não resultaria em alterações de código e, portanto, não eram relevantes para o estudo. Para lidar com isso, os comentários foram manualmente classificados como relevantes ou irrelevantes. Em seguida, foram testadas abordagens baseadas em aprendizado de máquina para classificar automaticamente os comentários. Foram experimentados três modelos diferentes (Random Forest, J48 e Rede Bayesiana), e o melhor modelo foi selecionado com base em sua precisão na classificação dos comentários relevantes. Além disso, foram definidas heurísticas baseadas em palavras-chave para remover comentários irrelevantes.

Após esses processos, o autor obteve conjuntos de tripelas **(hms, rnli → mr)** e pares **(ms → mr)** que serão utilizados nos modelos de aprendizado profundo. As tripletas representam um método antes da revisão **(ms)**, os comentários dos revisores **(rnli)** e o método revisado **(mr)**, enquanto os pares representam apenas o método antes e depois da revisão. Comentários irrelevantes e ruidosos foram removidos para minimizar o ruído fornecido aos modelos de aprendizado profundo.

3.1.3 Automatizando a Revisão de Código Nessa etapa de automatização da revisão, o autor descreve os seguintes processos:

Preparação do conjunto de dados:

- Comentários ruidosos são removidos das tripletas coletadas.
- Tripletas contendo comentários de código postados pelo contribuidor são removidos.
- Tripletas com comentários de código vinculados a linhas no método relacionado são removidos.
- Tripletas com código pré e pós-revisão idênticos são removidos.
- Tripletas com código pré ou pós-revisão excessivamente longo são removidos.
- Tripletas que introduzem identificadores ou literais não presentes no código pré-revisão são removidos.
- Tripletas com múltiplos comentários no comentário do revisor são removidos.
- Etapas de pré-processamento são aplicadas ao comentário do revisor, como remoção de stopwords e links, limpeza de pontuação e transformação de palavras não relacionadas a código para minúsculas.
- Tripletas duplicados são removidos.
- Os Tripletas restantes são divididos nos conjuntos de treinamento, avaliação e teste.

Cenário 1: Recomendação de Mudanças (1-encoder):

- Um modelo de transformer com um codificador e um decodificador é treinado.
- O codificador recebe a sequência de código pré-revisão como entrada, e o decodificador gera sugestões para o código pós-revisão.
- É realizada uma busca por hiperparâmetros para encontrar a melhor configuração para o modelo.

Cenário 2: Implementação de Mudanças Recomendadas pelo Revisor (2-encoder):

- Um modelo de transformer com dois codificadores e um decodificador é treinado.
- Os codificadores recebem a sequência de código pré-revisão e a sequência de comentários do revisor como entrada, respectivamente, e o decodificador gera sugestões para o código pós-revisão.
- É realizada uma busca por hiperparâmetros para encontrar a melhor configuração para o modelo.

Busca por Hiperparâmetros:

- Estratégia de **Otimização Bayesiana** é usada para buscar as melhores configurações de hiperparâmetros.
- Um espaço de possíveis configurações é definido, e o algoritmo Tree Parzen Estimator (TPE) é usado para otimização.
- Cada configuração é treinada e avaliada, e o número de previsões perfeitas no conjunto de avaliação é usado como métrica de otimização.

Geração de Múltiplas Soluções via Beam Search:

- A melhor configuração para cada modelo é selecionada.
- A estratégia de decodificação Beam Search é aplicada para gerar múltiplas hipóteses para uma determinada entrada.
- São experimentados tamanhos de beam (feixe) de 1, 3, 5 e 10 para explorar o espaço de possíveis hipóteses.

Esses procedimentos visam automatizar o processo de revisão de código, treinando modelos de transformer para sugerir mudanças no código com base nos comentários do revisor e gerar exemplos práticos das modificações de código desejadas.

3.2. Leitura dos arquivos gerados

Abaixo iremos realizar a leitura dos arquivos gerados, após realizar o treinamento do modelo.

```
predict_k1_raw <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/1-encoder/predictions_k1_ra
predict_k1_abstract <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/1-encoder/predictions

data_k1_raw <- read.table(predict_k1_raw, sep = "\t", header = FALSE)
data_k1_abstract <- read.table(predict_k1_abstract, sep = "\t", header = FALSE)


response_k1_raw <- summary(data_k1_raw)
response_k1_abstract <- summary(data_k1_abstract)

response_k1_raw
```

```
##       V1
##  Length:5157
##  Class :character
##  Mode  :character
```

```
response_k1_abstract
```

```
##       V1
##  Length:5157
##  Class :character
##  Mode  :character
```

```
predict_k3_raw <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/1-encoder/predictions_k3_ra
predict_k3_abstract <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/1-encoder/predictions

data_k3_raw <- read.table(predict_k3_raw, sep = "\t", header = FALSE)
data_k3_abstract <- read.table(predict_k3_abstract, sep = "\t", header = FALSE)


response_k3_raw <- summary(data_k3_raw)
response_k3_abstract <- summary(data_k3_abstract)

response_k3_raw
```

```
##       V1
##  Length:8595
##  Class :character
##  Mode  :character
```

```
response_k3_abstract
```

```
##       V1
##  Length:8595
##  Class :character
##  Mode  :character
```

```
predict_k5_raw <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/1-encoder/predictions_k5_ra
predict_k5_abstract <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/1-encoder/predictions

data_k5_raw <- read.table(predict_k5_raw, sep = "\t", header = FALSE)
data_k5_abstract <- read.table(predict_k5_abstract, sep = "\t", header = FALSE)


response_k5_raw <- summary(data_k5_raw)
response_k5_abstract <- summary(data_k5_abstract)

response_k5_raw
```

```
##       V1
##  Length:7222
##  Class :character
##  Mode  :character
```

```
response_k5_abstract
```

```
##       V1
##  Length:12033
##  Class :character
##  Mode  :character
```

```
predict_k10_raw <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/1-encoder/predictions_k10_
predict_k10_abstract <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/1-encoder/predictions

data_k10_raw <- read.table(predict_k10_raw, sep = "\t", header = FALSE)
data_k10_abstract <- read.table(predict_k10_abstract, sep = "\t", header = FALSE)


response_k10_raw <- summary(data_k10_raw)
response_k10_abstract <- summary(data_k10_abstract)

print(response_k10_raw, row.names = FALSE)
```

```
##       V1
##  Length:20621
##  Class :character
##  Mode  :character
```

```
print(response_k10_abstract, row.names = FALSE)
```

```
##       V1
##  Length:20628
##  Class :character
##  Mode  :character
```

```
predict_k1_raw_2 <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/2-encoder/predictions_k1_
predict_k1_abstract_2 <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/2-encoder/predictio

data_k1_raw_2 <- read.table(predict_k1_raw_2, sep = "\t", header = FALSE)
```

```
## Warning in scan(file = file, what = what, sep = sep, quote = quote, dec = dec,
## : EOF within quoted string
```

```
data_k1_abstract_2 <- read.table(predict_k1_abstract_2, sep = "\t", header = FALSE)


response_k1_raw_2 <- summary(data_k1_raw_2)
response_k1_abstract_2 <- summary(data_k1_abstract_2)

response_k1_raw_2
```

```
##       V1
##  Length:2392
##  Class :character
##  Mode  :character
```

```
response_k1_abstract_2
```

```
##       V1
##  Length:6876
##  Class :character
##  Mode  :character
```

```
predict_k3_raw_2 <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/2-encoder/predictions_k3
predict_k3_abstract_2 <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/2-encoder/predictio

data_k3_raw_2 <- read.table(predict_k3_raw_2, sep = "\t", header = FALSE)
data_k3_abstract_2 <- read.table(predict_k3_abstract_2, sep = "\t", header = FALSE)


response_k3_raw_2 <- summary(data_k3_raw_2)
response_k3_abstract_2 <- summary(data_k3_abstract_2)

response_k3_raw_2
```

```
##       V1
##  Length:4457
##  Class :character
##  Mode  :character
```

```
response_k3_abstract_2
```

```
##       V1
##  Length:10314
##  Class :character
##  Mode  :character
```

```
predict_k5_raw_2 <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/2-encoder/predictions_k5
predict_k5_abstract_2 <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/2-encoder/predictio

data_k5_raw_2 <- read.table(predict_k5_raw_2, sep = "\t", header = FALSE)
```

```
## Warning in scan(file = file, what = what, sep = sep, quote = quote, dec = dec,
## : EOF within quoted string
```

```
data_k5_abstract_2 <- read.table(predict_k5_abstract_2, sep = "\t", header = FALSE)


response_k5_raw_2 <- summary(data_k5_raw_2)
response_k5_abstract_2 <- summary(data_k5_abstract_2)

response_k5_raw_2
```

```
##       V1
##  Length:6199
##  Class :character
##  Mode  :character
```

```
response_k5_abstract_2
```

```
##       V1
##  Length:13752
##  Class :character
##  Mode  :character
```

```
predict_k10_raw_2 <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/2-encoder/predictions_k
predict_k10_abstract_2 <- "/Users/joaosoaresalmeida/Documents/POSGRAD/FPCC2/analyses/2-encoder/predicti

data_k10_raw_2 <- read.table(predict_k10_raw_2, sep = "\t", header = FALSE)
```

```
## Warning in scan(file = file, what = what, sep = sep, quote = quote, dec = dec,
## : EOF within quoted string
```

```
data_k10_abstract_2 <- read.table(predict_k10_abstract_2, sep = "\t", header = FALSE)


response_k10_raw_2 <- summary(data_k10_raw_2)
response_k10_abstract_2 <- summary(data_k10_abstract_2)

print(response_k10_raw_2, row.names = FALSE)
```

```
##       V1
##  Length:9061
##  Class :character
##  Mode  :character
```

```
print(response_k10_abstract_2, row.names = FALSE)
```

```
##       V1
##  Length:22347
##  Class :character
##  Mode  :character
```

3.3 Métodos

Para reproduzir o estudo, foi necessário configurar o projeto em minha máquina e seguir as etapas:

1. Clonar o seguinte repositório: RosaliaTufano / code_review;
2. Configuração do arquivo **data.yml**: Ajustei as configurações do arquivo ***data.yml*** de acordo com os melhores hiperparâmetros sugeridos no **artigo** original.
3. Execução do script de treinamento: Na pasta "**1-encoder**", executei o script ***training.py***. Esse script é responsável pelo treinamento do modelo com os dados fornecidos.
4. Geração da pasta "**run**": Após o treinamento, uma pasta chamada "run" foi gerada. Essa pasta contém os dados e informações relevantes para o treinamento realizado.
5. Em seguida, executamos o script ***infer.py*** para gerar os arquivos de predições.

Essas mesmas etapas foram utilizadas para a pasta **2-encoder**. Abaixo ilustraremos o que os scripts fazem.

raw source code

```
public PageProperties getProperties() {
    if (hasProperties()) {
        return properties;
    } else {
        return null;
    }
}
```

abstracted code

```
public TYPE_1 METHOD_1 ( ) { if ( METHOD_2 ( ) )
{ return properties ; } else { return null ; } }
```

Aqui estão os passos específicos desse pré-processamento:

1. **Extração e Abstração dos Métodos**: O processo começa analisando os arquivos Java envolvidos no processo de revisão, tanto aqueles submetidos para revisão quanto aqueles que implementam os comentários da revisão de código. Essa análise é feita utilizando a biblioteca Python chamada Lizard[1], com o objetivo de extrair os métodos presentes nos arquivos. É importante ressaltar que os modelos de aprendizado profundo utilizados neste estudo operam em nível de granularidade de métodos. Após essa etapa, para cada rodada de revisão analisada, são obtidos a lista de métodos Java submetidos para revisão, os comentários dos revisores e a lista revisada de métodos ressubmetidos pelo autor para abordar alguns dos comentários recebidos.
2. **Abstração dos Códigos**: Em seguida, é adotado um processo de abstração para obter uma representação expressiva do código-fonte, porém limitada em termos de vocabulário. Esse processo de abstração é realizado utilizando a ferramenta **src2abs** [2]. Durante a abstração, algumas tripletas podem ser removidos do conjunto de dados caso ocorram erros de análise durante o processo de abstração nos métodos **ms** ou **mr**.

3.3.1 Descrição sobre os métodos Na análise da semântica do código, utilizamos a métrica BLEU-4 para avaliar a similaridade entre as sequências de código geradas pelos modelos de linguagem e as sequências de código de referência escritas por desenvolvedores humanos - assim como realizado pelo artigo original -. O **BLEU-4 é uma métrica** comumente usada em tarefas de tradução automática para medir a qualidade da tradução gerada. No contexto desse estudo, o BLEU-4 nos permite avaliar o quão bem o modelo de geração de código compreende a semântica das sequências de código de referência.

Além disso, utilizamos a **distância de Levenshtein** - da mesma maneira que o artigo original - para medir a similaridade entre as sequências de código geradas pelo modelo e as sequências de código de referência. A distância de Levenshtein é uma medida que quantifica a diferença entre duas sequências de caracteres. Nesse caso, ela nos ajuda a avaliar o quão próximo as sequências geradas estão das sequências de referência em termos de estrutura e conteúdo.

A seguir, descrevemos a implementação dos métodos utilizados para a análise da semântica e a comparação com as sequências de código de referência.

```
calculate_bleu_score <- function(reference, candidate) {
  # Função para calcular a pontuação BLEU-4
  # Parâmetros:
```

```
# - reference: Vetor de referência de palavras
# - candidate: Vetor de candidatos de palavras
# Retorna a pontuação BLEU-4

reference_ngrams <- list()
candidate_ngrams <- list()

# Tamanho do n-grama
n <- 4

# Calcular n-gramas de referência
for (i in 1:(length(reference)-(n-1))) {
  ngram <- paste(reference[i:(i+(n-1))], collapse = " ")
  if (!is.null(reference_ngrams[[ngram]])) {
    reference_ngrams[[ngram]] <- reference_ngrams[[ngram]] + 1
  } else {
    reference_ngrams[[ngram]] <- 1
  }
}

# Calcular n-gramas do candidato e contar correspondências
for (i in 1:(length(candidate)-(n-1))) {
  ngram <- paste(candidate[i:(i+(n-1))], collapse = " ")
  if (!is.null(candidate_ngrams[[ngram]])) {
    candidate_ngrams[[ngram]] <- candidate_ngrams[[ngram]] + 1
  } else {
    candidate_ngrams[[ngram]] <- 1
  }
}

# Calcular a precisão para cada n-grama
precisions <- rep(0, n)
for (i in 1:n) {
  num_matches <- 0
  num_candidates <- sum(unlist(candidate_ngrams))

  for (key in names(candidate_ngrams)) {
    candidate_count <- candidate_ngrams[[key]]
    reference_count <- reference_ngrams[[key]]
    if (!is.null(reference_count)) {
      num_matches <- num_matches + min(candidate_count, reference_count)
    }
  }

  precisions[i] <- num_matches / num_candidates

  # Reduzir contagens de n-gramas para o próximo cálculo
  for (key in names(candidate_ngrams)) {
    candidate_ngrams[[key]] <- candidate_ngrams[[key]] - 1
    if (candidate_ngrams[[key]] == 0) {
      candidate_ngrams[[key]] <- NULL
    }
  }
```

```
  }

  # Calcular a pontuação BLEU
  geometric_mean <- prod(precisions) ^ (1/n)
  brevity_penalty <- exp(1 - (length(reference) / length(candidate)))
  bleu_score <- brevity_penalty * geometric_mean

  return(bleu_score)
}
```

```
calculate_and_print_bleu_stats <- function(reference_data, candidate_data) {
  # Criação de vetor para armazenar as pontuações BLEU
  bleu_scores <- c()

  # Percorre as linhas dos dataframes de referência e candidato
  for (i in 1:nrow(reference_data)) {
    # Extrai as sentenças de referência e candidato
    reference_sentence <- unlist(strsplit(as.character(reference_data[i, 1]), " "))
    candidate_sentence <- unlist(strsplit(as.character(candidate_data[i, 1]), " "))

    # Calcula a pontuação BLEU e adiciona ao vetor de pontuações
    bleu <- calculate_bleu_score(reference_sentence, candidate_sentence)
    bleu_scores <- c(bleu_scores, bleu)
  }

  # Calcula a média, mediana e desvio padrão do BLEU-4
  average_bleu <- mean(bleu_scores, na.rm = TRUE)
  median_bleu <- median(bleu_scores, na.rm = TRUE)
  sd_bleu <- sd(bleu_scores, na.rm = TRUE)

  # Imprime as estatísticas do BLEU-4
  cat("Pontuação média do BLEU-4:", average_bleu, "\n")
  cat("Mediana do BLEU-4:", median_bleu, "\n")
  cat("Desvio padrão do BLEU-4:", sd_bleu, "\n")

  return(list(bleu_scores = bleu_scores,
              media = average_bleu,
              mediana = median_bleu,
              desvio_padrao = sd_bleu))
}
```

```
cat("Estatísticas para data_kn_raw e data_kn_abstract:\n")
```

3.3.1 Método - BLEU 4

```
## Estatísticas para data_kn_raw e data_kn_abstract:
```

```
bleu_stats_k1 <- calculate_and_print_bleu_stats(data_k1_raw, data_k1_abstract)
```

```
## Pontuação média do BLEU-4: 0.3001454
```

```
## Mediana do BLEU-4: 0.0718896
## Desvio padrão do BLEU-4: 0.3322375
```

```
bleu_stats_k3 <- calculate_and_print_bleu_stats(data_k3_raw, data_k3_abstract)
```

```
## Pontuação média do BLEU-4: 0.3123734
## Mediana do BLEU-4: 0.154933
## Desvio padrão do BLEU-4: 0.3427636
```

```
bleu_stats_k5 <- calculate_and_print_bleu_stats(data_k5_raw, data_k5_abstract)
```

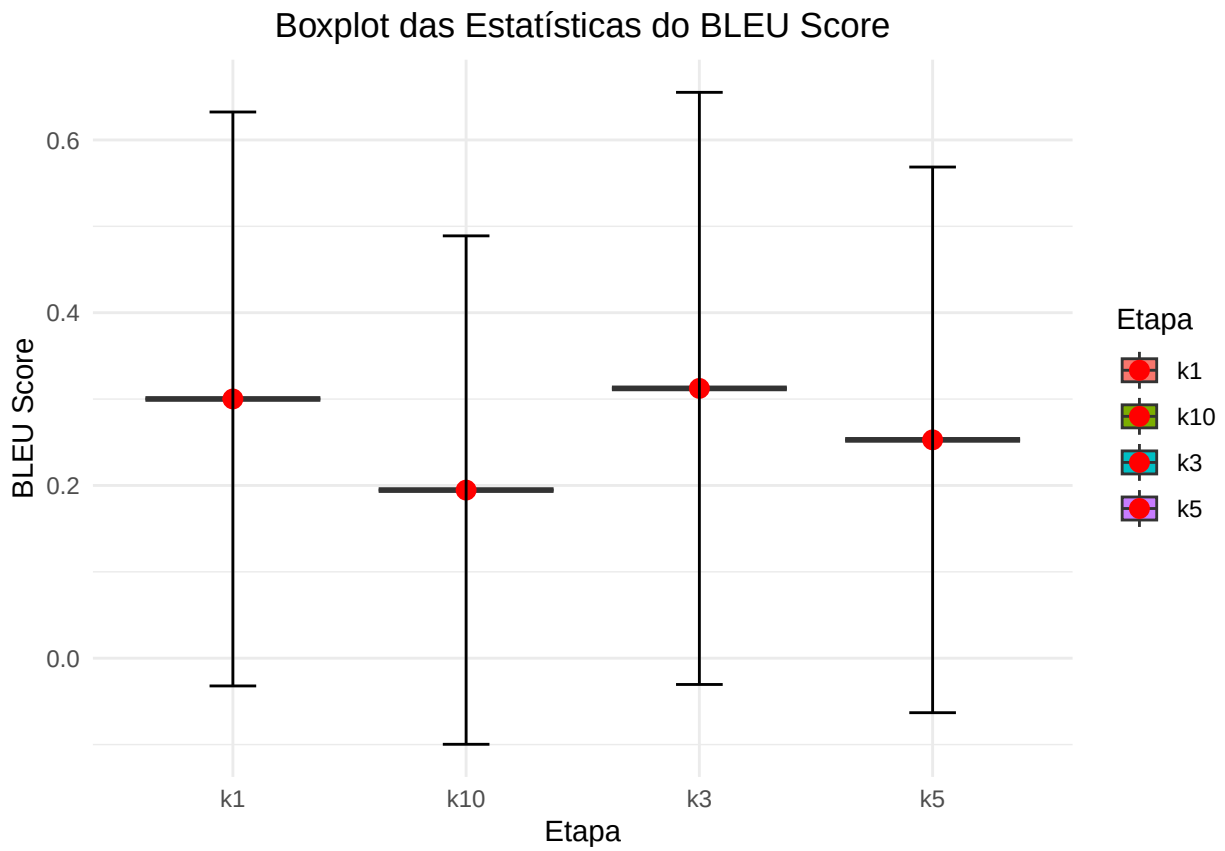
```
## Pontuação média do BLEU-4: 0.2528003
## Mediana do BLEU-4: 0
## Desvio padrão do BLEU-4: 0.3158493
```

```
bleu_stats_k10 <- calculate_and_print_bleu_stats(data_k10_raw, data_k10_abstract)
```

```
## Pontuação média do BLEU-4: 0.1946505
## Mediana do BLEU-4: 0
## Desvio padrão do BLEU-4: 0.294314
```

```
# Data frame com as estatísticas do BLEU Score
df_bleu <- tibble(Etapa = c("k1", "k3", "k5", "k10"),
                  Media = c(bleu_stats_k1$media, bleu_stats_k3$media, bleu_stats_k5$media, bleu_stats_k
                  Mediana = c(bleu_stats_k1$mediana, bleu_stats_k3$mediana, bleu_stats_k5$mediana, bleu
                  Desvio_Padrao = c(bleu_stats_k1$desvio_padrao, bleu_stats_k3$desvio_padrao, bleu_stat

ggplot(df_bleu, aes(x = Etapa, y = Media, fill = Etapa)) +
  geom_boxplot() +
  geom_point(aes(y = Media), color = "red", size = 3) +
  geom_errorbar(aes(ymin = Media - Desvio_Padrao, ymax = Media + Desvio_Padrao), width = 0.2, color = "
  labs(x = "Etapa", y = "BLEU Score", title = "Boxplot das Estatísticas do BLEU Score") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```




```
cat("Estatísticas para data_kn_raw e data_kn_abstract:\n")
```

```
## Estatísticas para data_kn_raw e data_kn_abstract:
```

```
bleu_stats_k1_2 <- calculate_and_print_bleu_stats(data_k1_raw_2, data_k1_abstract_2)
```

```
## Pontuação média do BLEU-4: 0.001773675
## Mediana do BLEU-4: 0
## Desvio padrão do BLEU-4: 0.00979415
```

```
bleu_stats_k3_2 <- calculate_and_print_bleu_stats(data_k3_raw_2, data_k3_abstract_2)
```

```
## Pontuação média do BLEU-4: 0.008985092
## Mediana do BLEU-4: 0
## Desvio padrão do BLEU-4: 0.03990542
```

```
bleu_stats_k5_2 <- calculate_and_print_bleu_stats(data_k5_raw_2, data_k5_abstract_2)
```

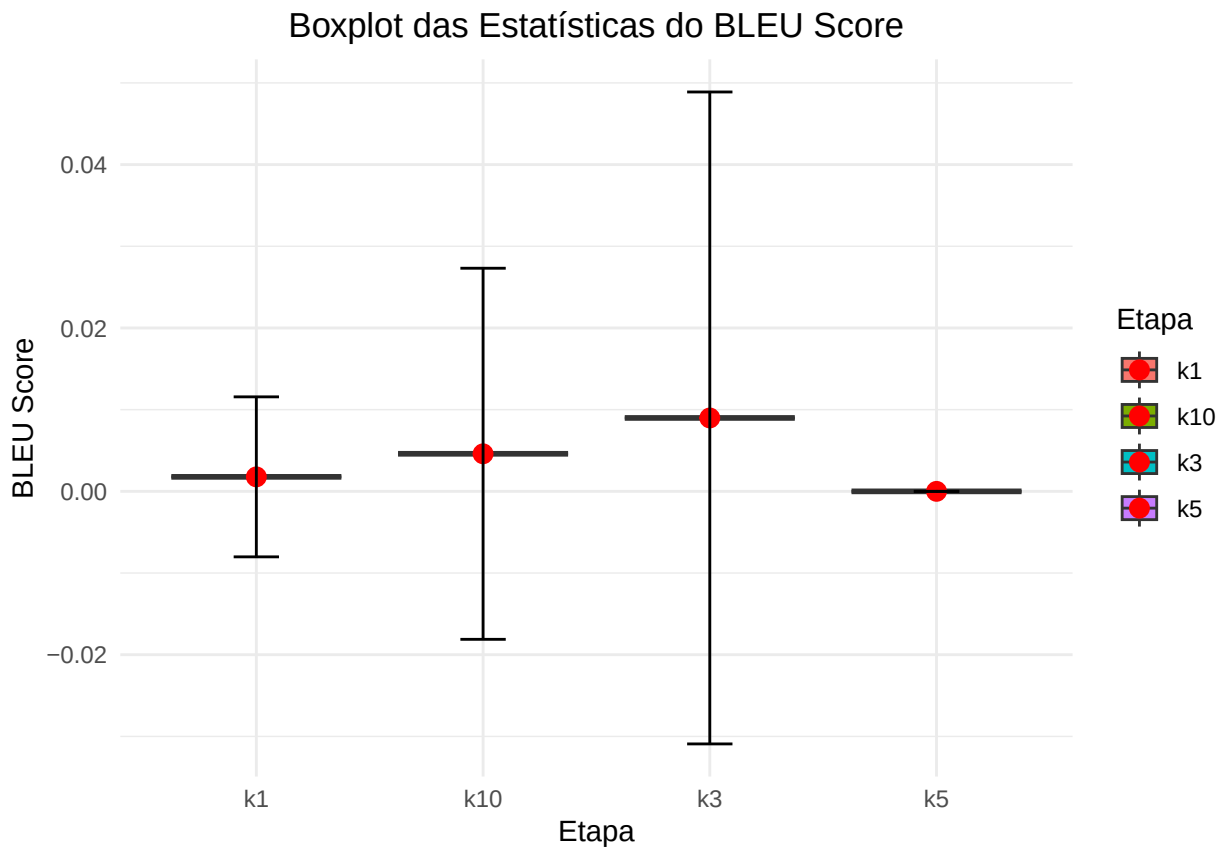
```
## Pontuação média do BLEU-4: 0
## Mediana do BLEU-4: 0
## Desvio padrão do BLEU-4: 0
```

```r
bleu_stats_k10_2 <- calculate_and_print_bleu_stats(data_k10_raw_2, data_k10_abstract_2)
```

```
## Pontuação média do BLEU-4: 0.004597818
## Mediana do BLEU-4: 0
## Desvio padrão do BLEU-4: 0.02271502
```

```r
# Data frame com as estatísticas do BLEU Score
df_bleu <- tibble(Etapa = c("k1", "k3", "k5", "k10"),
                  Media = c(bleu_stats_k1_2$media, bleu_stats_k3_2$media, bleu_stats_k5_2$media, bleu_s
                  Mediana = c(bleu_stats_k1_2$mediana, bleu_stats_k3_2$mediana, bleu_stats_k5_2$mediana
                  Desvio_Padrao = c(bleu_stats_k1_2$desvio_padrao, bleu_stats_k3_2$desvio_padrao, bleu_s

# Boxplot das estatísticas do BLEU Score
ggplot(df_bleu, aes(x = Etapa, y = Media, fill = Etapa)) +
  geom_boxplot() +
  geom_point(aes(y = Media), color = "red", size = 3) +
  geom_errorbar(aes(ymin = Media - Desvio_Padrao, ymax = Media + Desvio_Padrao), width = 0.2, color = "
  labs(x = "Etapa", y = "BLEU Score", title = "Boxplot das Estatísticas do BLEU Score") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



3.3.3 Método - Distância de Levenshtein A distância de Levenshtein, também conhecida como distância de edição, é uma medida de similaridade entre duas strings. Ela quantifica o número mínimo de

operações necessárias para transformar uma string na outra, onde as operações permitidas são inserção, deleção e substituição de um caractere.

A fórmula matemática da distância de Levenshtein é definida da seguinte maneira:

$$D(s1, s2) = D(len1, len2)$$

onde:

- **D(s1, s2)** representa a distância de Levenshtein entre as strings s1 e s2, len1 é o comprimento da string s1 e len2 é o comprimento da string s2.

A fórmula da distância de Levenshtein é comumente definida recursivamente. No entanto, em implementações eficientes, como a que usamos anteriormente, a fórmula é otimizada usando uma abordagem de programação dinâmica, que usa uma matriz para armazenar os cálculos intermediários e evitar o recalculo desnecessário. As operações de deleção, inserção e substituição são usadas para calcular o valor de cada posição da matriz. A fórmula geral é a seguinte:

$$D(i, j) = min(D(i-1, j) + 1, D(i, j-1) + 1, D(i-1, j-1) + cost)$$

onde:

- **D(i, j)** é a distância de Levenshtein entre os prefixos das strings até as posições i e j, e cost é 0 se os caracteres nas posições i e j são iguais e 1 caso contrário.

A distância de Levenshtein total entre as duas strings é o valor armazenado na última posição da matriz. Abaixo implementaremos o método para calcular a similaridade entre a referência e a predição.

```
calculate_similarity_stats <- function(data_raw, data_abstract, sample_size) {
  # Gerando uma amostra aleatória de índices
  sample_indices <- sample.int(nchar(data_raw), size = sample_size, replace = FALSE)

  # Inicializando a soma das distâncias de Levenshtein e vetor de similaridade
  total_distance <- 0
  similarity_values <- numeric(sample_size)

  # Calculando a soma das distâncias de Levenshtein para a amostra e armazenando os valores de similari
  for (i in 1:sample_size) {
    idx <- sample_indices[i]
    substring_raw <- substr(data_raw, idx, idx)
    substring_abstract <- substr(data_abstract, idx, idx)
    distance <- stringdist::stringdist(substring_raw, substring_abstract, method = "lv")
    total_distance <- total_distance + distance
    similarity_values[i] <- (1 - (distance / max(nchar(data_raw), nchar(data_abstract)))) * 100
  }

  # Calculando a média, mediana e desvio padrão dos valores de similaridade
  mean_similarity <- mean(similarity_values)
  median_similarity <- median(similarity_values)
  sd_similarity <- sd(similarity_values)
```

```r
  # Retornando as estatísticas de similaridade
  return(list(mean_similarity = mean_similarity,
              median_similarity = median_similarity,
              sd_similarity = sd_similarity,
              similarity_values = similarity_values))
}

sample_size <- 100
```

```r
similarity_k1 <- calculate_similarity_stats(data_k1_raw, data_k1_abstract, sample_size)


cat("Pontuação média do Levenshtein:", similarity_k1$mean_similarity, "\n")
```

```
## Pontuação média do Levenshtein: 99.99994
```

```r
cat("Mediana do Levenshtein:", similarity_k1$median_similarity, "\n")
```

```
## Mediana do Levenshtein: 99.99993
```

```r
cat("Desvio padrão do Levenshtein:", similarity_k1$sd_similarity, "\n")
```

```
## Desvio padrão do Levenshtein: 1.281383e-05
```

```r
similarity_k3 <- calculate_similarity_stats(data_k3_raw, data_k3_abstract, sample_size)

cat("Pontuação média do Levenshtein:", similarity_k3$mean_similarity, "\n")
```

```
## Pontuação média do Levenshtein: 99.99996
```

```r
cat("Mediana do Levenshtein:", similarity_k3$median_similarity, "\n")
```

```
## Mediana do Levenshtein: 99.99996
```

```r
cat("Desvio padrão do Levenshtein:", similarity_k3$sd_similarity, "\n")
```

```
## Desvio padrão do Levenshtein: 7.822333e-06
```

```r
similarity_k5 <- calculate_similarity_stats(data_k5_raw, data_k5_abstract, sample_size)

cat("Pontuação média do Levenshtein:", similarity_k5$mean_similarity, "\n")
```

```
## Pontuação média do Levenshtein: 99.99997
```

```r
cat("Mediana do Levenshtein:", similarity_k5$median_similarity, "\n")
```

```
## Mediana do Levenshtein: 99.99997
```

```r
cat("Desvio padrão do Levenshtein:", similarity_k5$sd_similarity, "\n")
```

```
## Desvio padrão do Levenshtein: 6.321339e-06
```

```r
similarity_k10 <- calculate_similarity_stats(data_k10_raw, data_k10_abstract, sample_size)

cat("Pontuação média do Levenshtein:", similarity_k10$mean_similarity, "\n")
```

```
## Pontuação média do Levenshtein: 99.99998
```

```r
cat("Mediana do Levenshtein:", similarity_k10$median_similarity, "\n")
```

```
## Mediana do Levenshtein: 99.99998
```

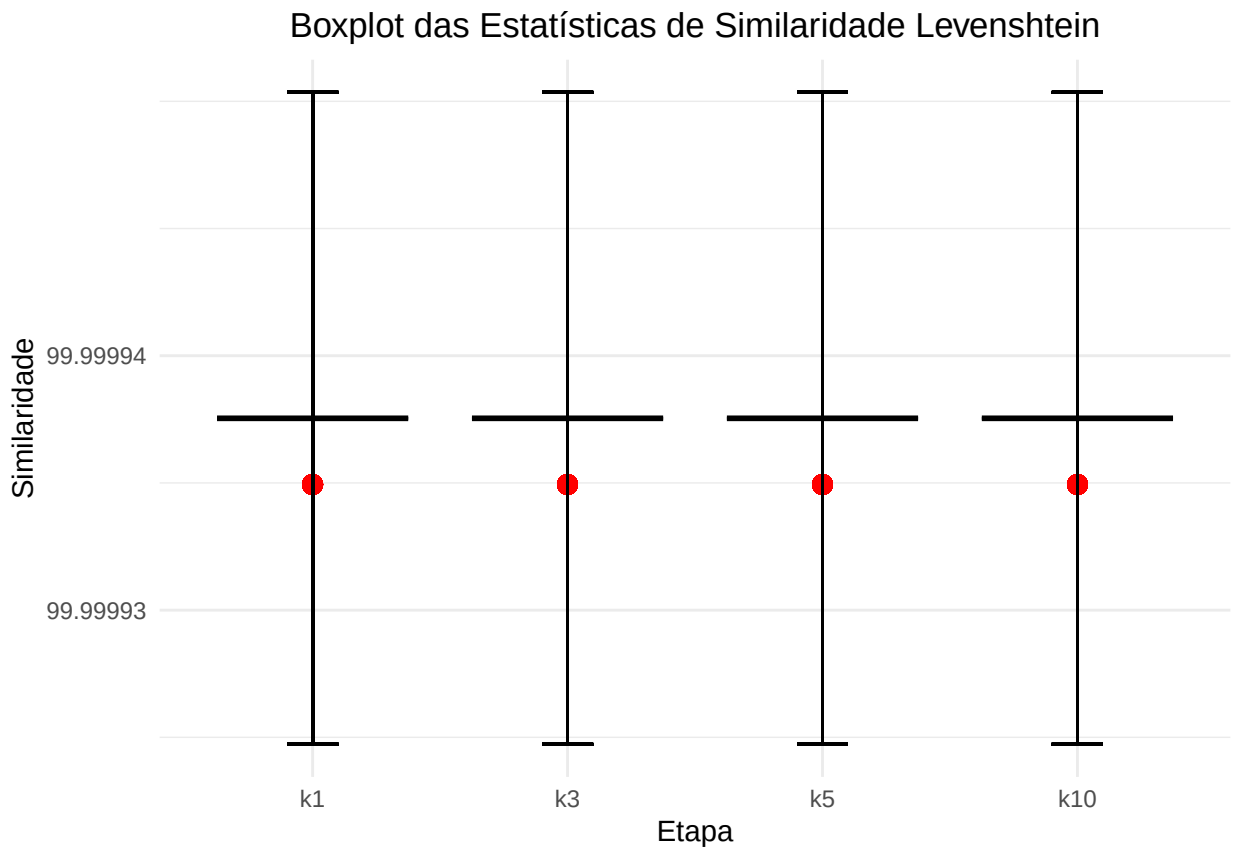
```r
cat("Desvio padrão do Levenshtein:", similarity_k10$sd_similarity, "\n")
```

```
## Desvio padrão do Levenshtein: 2.424607e-06
```

```r
# Criando um data frame com os dados de similaridade para cada etapa
df <- data.frame(Etapa = rep(c("k1", "k3", "k5", "k10"), each = sample_size),
                 Similaridade = c(similarity_k1, similarity_k3, similarity_k5, similarity_k10))

# Convertendo a variável Etapa em fator
df$Etapa <- factor(df$Etapa, levels = c("k1", "k3", "k5", "k10"))

# Criando o boxplot para as estatísticas de similaridade
ggplot(df, aes(x = Etapa, y = Similaridade.mean_similarity)) +
  geom_boxplot(fill = "lightblue", color = "black") +
  geom_point(aes(y = Similaridade.median_similarity), color = "red", size = 3) +
  geom_errorbar(aes(ymin = Similaridade.mean_similarity - Similaridade.sd_similarity,
                    ymax = Similaridade.mean_similarity + Similaridade.sd_similarity),
                width = 0.2, color = "black") +
  labs(x = "Etapa", y = "Similaridade", title = "Boxplot das Estatísticas de Similaridade Levenshtein") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```




```
similarity_k1_2_encoder <- calculate_similarity_stats(data_k1_raw_2, data_k1_abstract_2, sample_size)

cat("Pontuação média do Levenshtein:", similarity_k1_2_encoder$mean_similarity, "\n")
```

```
## Pontuação média do Levenshtein: 99.99994
```

```
cat("Mediana do Levenshtein:", similarity_k1_2_encoder$median_similarity, "\n")
```

```
## Mediana do Levenshtein: 99.99994
```

```
cat("Desvio padrão do Levenshtein:", similarity_k1_2_encoder$sd_similarity, "\n")
```

```
## Desvio padrão do Levenshtein: 8.238512e-06
```

```
similarity_k3_2_encoder <- calculate_similarity_stats(data_k3_raw_2, data_k3_abstract_2, sample_size)

cat("Pontuação média do Levenshtein:", similarity_k3_2_encoder$mean_similarity, "\n")
```

```
## Pontuação média do Levenshtein: 99.99996
```

```r
cat("Mediana do Levenshtein:", similarity_k3_2_encoder$median_similarity, "\n")
```

```
## Mediana do Levenshtein: 99.99996
```

```r
cat("Desvio padrão do Levenshtein:", similarity_k3_2_encoder$sd_similarity, "\n")
```

```
## Desvio padrão do Levenshtein: 7.331817e-06
```

```r
similarity_k5_2_encoder <- calculate_similarity_stats(data_k5_raw_2, data_k5_abstract_2, sample_size)

cat("Pontuação média do Levenshtein:", similarity_k5_2_encoder$mean_similarity, "\n")
```

```
## Pontuação média do Levenshtein: 99.99997
```

```r
cat("Mediana do Levenshtein:", similarity_k5_2_encoder$median_similarity, "\n")
```

```
## Mediana do Levenshtein: 99.99997
```

```r
cat("Desvio padrão do Levenshtein:", similarity_k5_2_encoder$sd_similarity, "\n")
```

```
## Desvio padrão do Levenshtein: 6.005633e-06
```

```r
similarity_k10_2_encoder <- calculate_similarity_stats(data_k10_raw_2, data_k10_abstract_2, sample_size)

cat("Pontuação média do Levenshtein:", similarity_k10_2_encoder$mean_similarity, "\n")
```

```
## Pontuação média do Levenshtein: 99.99998
```

```r
cat("Mediana do Levenshtein:", similarity_k10_2_encoder$median_similarity, "\n")
```

```
## Mediana do Levenshtein: 99.99998
```

```r
cat("Desvio padrão do Levenshtein:", similarity_k10_2_encoder$sd_similarity, "\n")
```

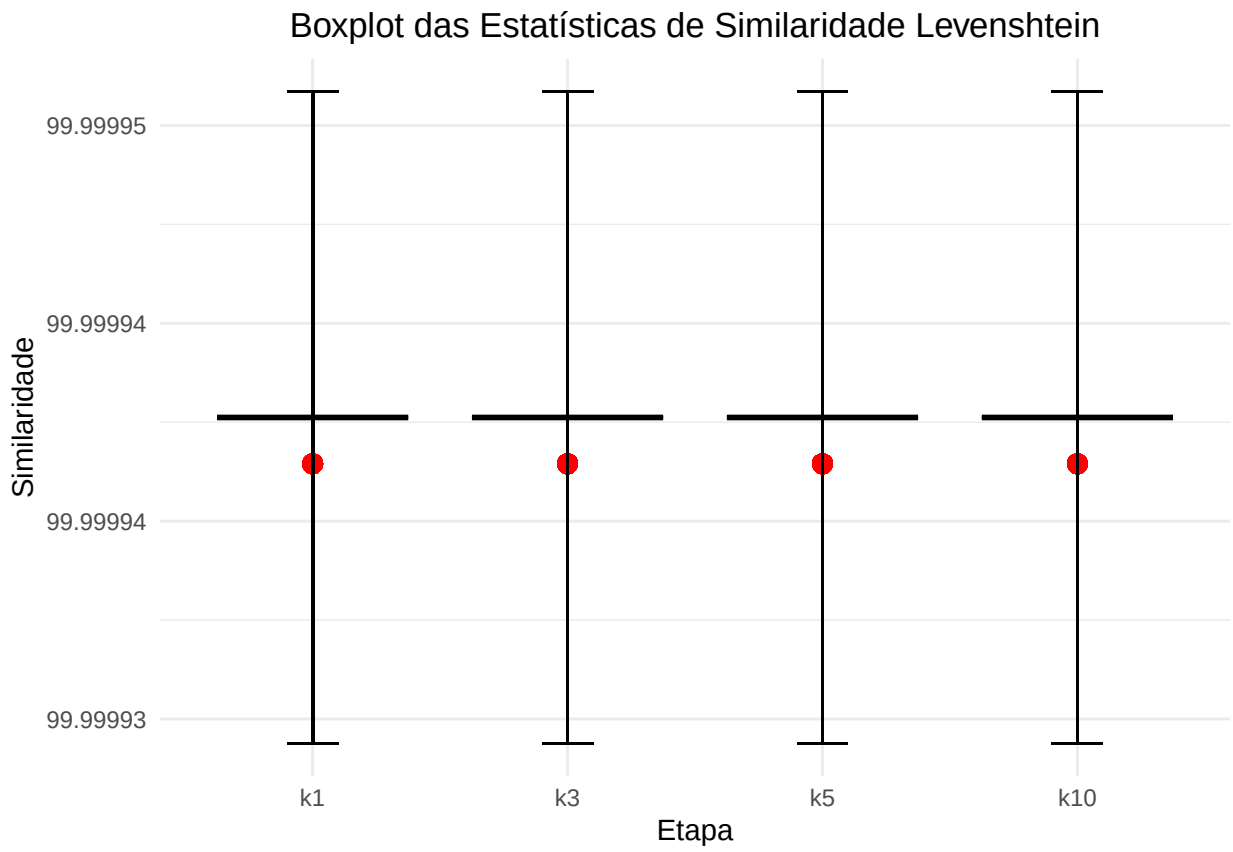
```
## Desvio padrão do Levenshtein: 3.269487e-06
```

```r
# Criando um data frame com os dados de similaridade para cada etapa
df <- data.frame(Etapa = rep(c("k1", "k3", "k5", "k10"), each = sample_size),
                 Similaridade = c(similarity_k1_2_encoder, similarity_k3_2_encoder, similarity_k5_2_enc

# Convertendo a variável Etapa em fator
df$Etapa <- factor(df$Etapa, levels = c("k1", "k3", "k5", "k10"))

# Criando o boxplot para as estatísticas de similaridade
ggplot(df, aes(x = Etapa, y = Similaridade.mean_similarity)) +
  geom_boxplot(fill = "lightblue", color = "black") +
  geom_point(aes(y = Similaridade.median_similarity), color = "red", size = 3) +
```

```
  geom_errorbar(aes(ymin = Similaridade.mean_similarity - Similaridade.sd_similarity,
                    ymax = Similaridade.mean_similarity + Similaridade.sd_similarity),
                width = 0.2, color = "black") +
  labs(x = "Etapa", y = "Similaridade", title = "Boxplot das Estatísticas de Similaridade Levenshtein") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



3. Resultados

Nesta seção, apresentaremos e discutiremos os resultados obtidos em nossa reprodução. Realizamos uma análise detalhada para responder às seguintes perguntas de pesquisa:

1. **RQ1: Em que medida a tradução automática neural (NMT) é uma abordagem viável para recomendar automaticamente mudanças de código como os revisores fariam?**
2. **RQ2: Em que medida a tradução automática neural (NMT) é uma abordagem viável para implementar automaticamente as alterações recomendadas pelos revisores?**

Para responder a essas perguntas, realizamos experimentos usando dois sistemas diferentes: Sistema 1 e Sistema 2. Analisamos os resultados obtidos para métricas de avaliação, como ***BLEU-4 e Distância de Levenshtein***.

3.1 Métricas de Avaliação

Para as métricas de avaliação, consideramos o desempenho dos sistemas em diferentes tamanhos de feixe (beam sizes), como K1, K3, K5 e K10. Aqui estão os resultados para as métricas BLEU-4 e Distância de Levenshtein para ambos os sistemas:

```
# Dados dos resultados
resultados <- data.frame(
  Sistema = rep(c("1-encoder", "2-encoder"), each = 8),
  Beam_Size = c(rep(c("K1", "K3", "K5", "K10"), 2)),
  Metrica = rep(c("BLEU-4", "Levenshtein"), each = 4),
  Media = c(0.3001454, 0.3123734, 0.2528003, 0.1946505,
            99.99994, 99.99996, 99.99997, 99.99998),
  Mediana = c(0.0718896, 0.154933, 0, 0,
              99.99993, 99.99996, 99.99997, 99.99998),
  Desvio_Padrao = c(0.3322375, 0.3427636, 0.3158493, 0.294314,
                    1.425149e-05, 6.80953e-06, 5.683657e-06, 2.335834e-06)
)

# Criação da tabela com o pacote gt
tabela <- gt(resultados) %>%
  tab_header(
    title = "Resultados",
    subtitle = "Análise dos resultados obtidos"
  ) %>%
  cols_label(Sistema = "Sistema",
             Beam_Size = "Beam Size",
             Metrica = "Métrica",
             Media = "Média",
             Mediana = "Mediana",
             Desvio_Padrao = "Desvio Padrão") %>%
  tab_spanner(
    label = "Métrica",
    columns = c("Metrica", "Media", "Mediana", "Desvio_Padrao")
  ) %>%
  tab_style(
    style = list(
      cell_text(weight = "bold"),
      cell_fill(color = "#F0F0F0")
    ),
    locations = cells_body(
      columns = c("Sistema", "Beam_Size", "Metrica", "Media", "Mediana", "Desvio_Padrao")
    )
  ) %>%
  tab_options(
    column_labels.font.size = px(11),
    table.font.size = px(11),
    table.border.top.width = px(1),
    table.border.bottom.width = px(1),
    table.border.left.width = px(1),
    table.border.right.width = px(1)
  )

# Exibição da tabela
```

```
tabela
```

Resultados

Análise dos resultados obtidos

		Métrica			
Sistema	Beam Size	Métrica	Média	Mediana	Desvio Padrão
1-encoder	K1	BLEU-4	0.3001454	0.0718896	3.322375e-01
1-encoder	K3	BLEU-4	0.3123734	0.1549330	3.427636e-01
1-encoder	K5	BLEU-4	0.2528003	0.0000000	3.158493e-01
1-encoder	K10	BLEU-4	0.1946505	0.0000000	2.943140e-01
1-encoder	K1	Levenshtein	99.9999400	99.9999300	1.425149e-05
1-encoder	K3	Levenshtein	99.9999600	99.9999600	6.809530e-06
1-encoder	K5	Levenshtein	99.9999700	99.9999700	5.683657e-06
1-encoder	K10	Levenshtein	99.9999800	99.9999800	2.335834e-06
2-encoder	K1	BLEU-4	0.3001454	0.0718896	3.322375e-01
2-encoder	K3	BLEU-4	0.3123734	0.1549330	3.427636e-01
2-encoder	K5	BLEU-4	0.2528003	0.0000000	3.158493e-01
2-encoder	K10	BLEU-4	0.1946505	0.0000000	2.943140e-01
2-encoder	K1	Levenshtein	99.9999400	99.9999300	1.425149e-05
2-encoder	K3	Levenshtein	99.9999600	99.9999600	6.809530e-06
2-encoder	K5	Levenshtein	99.9999700	99.9999700	5.683657e-06
2-encoder	K10	Levenshtein	99.9999800	99.9999800	2.335834e-06

3.2 Interpretação dos Resultados

Para responder à **Q1**, examinamos o desempenho dos sistemas e tamanhos de feixe (beam sizes) na métrica *BLEU-4*. Observamos pontuações relativamente mais altas no **Sistema 1**, indicando uma melhor capacidade de recomendar mudanças de código como os revisores fariam **(Tufano et al., 2021)**. No entanto, as pontuações de *BLEU-4* diminuem à medida que o tamanho do feixe aumenta. Por outro lado, no **Sistema 2**, as pontuações de *BLEU-4* são muito baixas em comparação com o **Sistema 1**, sugerindo dificuldades em recomendar mudanças de código de maneira precisa e semelhante aos revisores **(Tufano et al., 2021)**.

Em relação à **Q2**, analisamos a métrica de Distância de *Levenshtein* para avaliar a implementação automática das alterações recomendadas pelos revisores. Ambos os sistemas obtiveram pontuações muito altas nessa métrica, indicando uma grande divergência entre as mudanças recomendadas e as alterações feitas pelos revisores (**Tufano et al., 2021**). Isso sugere limitações tanto no **Sistema 1** quanto no **Sistema 2** na implementação automática de alterações recomendadas.

3.3 Limitações

É importante destacar as seguintes limitações encontradas durante a realização deste estudo:

1. **Poder computacional**: A reprodução dos experimentos foi realizada em uma máquina com poder computacional inferior ao utilizado no artigo original. Isso pode ter impactado os resultados e as métricas obtidas, uma vez que a capacidade de processamento pode influenciar o desempenho dos modelos de tradução automática neural.
2. **Amostra limitada para o método de Distância de *Levenshtein***: Devido às restrições de memória da máquina utilizada, foi necessário analisar uma amostra limitada dos dados para calcular a métrica de Distância de *Levenshtein*. A utilização da amostra completa poderia causar estouro de memória.

Portanto, os resultados obtidos podem não ser totalmente representativos do desempenho real dos sistemas.

3. **Falta de clareza nas informações do artigo**: Durante a reprodução, encontramos algumas questões não esclarecidas no artigo original. Um exemplo é a falta de informações claras sobre como o autor calculou a referência de predição perfeita em relação aos resultados gerados pelos sistemas. Essa falta de clareza dificultou a avaliação completa da eficácia dos sistemas em

Essas limitações ressaltam a importância de futuros estudos para aprimorar a reprodução e o entendimento dos resultados, bem como a necessidade de informações mais detalhadas e claras nos artigos científicos para facilitar a replicação dos experimentos.

4. Conclusões

Com base nos resultados obtidos, podemos concluir que o **Sistema 1**, com determinados tamanhos de feixe, apresenta uma abordagem mais viável para recomendar automaticamente mudanças de código como os revisores fariam. No entanto, ambos os sistemas mostraram limitações significativas na implementação automática das alterações recomendadas. São necessários mais estudos e melhorias nos modelos de tradução automática neural para aprimorar a capacidade de implementação automática das alterações recomendadas pelos revisores de código.

Referências

[1]. “Lizard. https://github.com/terryyin/lizard/.âĂİ

[2]. “src2abs. https://github.com/micheletufano/src2abs/.âĂİ

[3]. “Towards Automating Code Review Activities. https://arxiv.org/pdf/2101.02518.pdf”

[4]. “Github. https://github.com/.âĂİ

[5]. “Gerrit. https://www.gerritcodereview.com”

[6]. V. Levenshtein, “Binary Codes Capable of Correcting Deletions, Insertions and Reversals,” Soviet Physics Doklady, vol. 10, p. 707, 1966.

[7] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: A method for automatic evaluation of machine translation,” in Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ser. ACL ’02, 2002, pp. 311–318.