

Name: Jeffrick Aldho J

Registration Number:
22BCT0353

StatKeyEval

A Statistical Framework for Dynamic Keyword Extraction, Evaluation, and Assessment Automation

The study employs a diverse range of text similarity features, grouped into categories to capture various aspects of similarity. While a total of 36 features are used, only a subset is explicitly discussed. Here's an exploration of these categories and the features they contain:

Semantic Similarity Features

Semantic features assess how closely the meanings of words or phrases align. Examples include:

- **PATH:** A measure based on the shortest path between two concepts in a semantic network.
- **LCH:** Leacock-Chodorow similarity, which scales path length logarithmically.
- **WUP:** Wu-Palmer similarity, which considers shared depth and individual depths of concepts.
- **RES:** Resnik similarity, derived from shared information content.
- **JCN:** Jiang-Conrath similarity, balancing shared and individual information.
- **Lesk:** A dictionary-based similarity using word definitions.
- **HSO:** Hirst-St-Onge measure of semantic relatedness.
- **LSA:** Latent Semantic Analysis, a technique based on matrix factorization of term-document relationships.
- **CBOW/Skip-gram:** Neural word embeddings capturing word relationships.

Lexical Overlap Features

These features evaluate textual similarity based on word overlap:

- **Jaccard Similarity:** Measures the ratio of shared words to the total unique words in two texts.
- **Word Overlap:** Counts exact word matches.
- **IDF Overlap:** Weights overlaps using inverse document frequency.
- **Phrasal Overlap:** Captures matches at the phrase level.
- **ROUGE Metrics:** Includes ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-W for sequence-based comparisons.

Novel Features

Innovative and domain-specific features are introduced in this category:

- **TF-IDF Novelty:** Quantifies how novel a term is in a given text compared to the dataset.

- **Topical Features:** Techniques like **LDA** and **BTM** extract topic distributions for comparison.
- **Relevance Feedback Features:**
 - **RF-I:** Feedback influenced by the top-ranked scorer.
 - **RFII:** Feedback based on the least-ranked scorer.
 - **RF-III:** Feedback considering all scorers collectively.
- **IDF-Based Overlap:** A refined overlap measure using term frequency statistics.

Alignment-Based Features

These features align text elements to gauge similarity more granularly:

- **Word-to-Word Alignment:** Aligns individual words for direct comparison.
- **Coverage:** Quantifies the proportion of aligned content.
- **Question-Demoted Alignment:** Alignments adjusted to ignore question-related terms.

While the study highlights 22 distinct features, the remaining ones are likely derivatives or combinations tailored to specific tasks.

Regression Techniques Used in the Study

To analyze and predict outcomes, the study employs several regression techniques, each tailored to handle different types of data and relationships. Here's an overview of the models and their unique capabilities:

1. Support Vector Regression (SVR)

- **Overview:** A robust model that uses kernels to capture complex relationships in data.
- **Key Hyperparameters:**
 - Kernel type (e.g., RBF, linear).
 - C for regularization.
 - Gamma for kernel width.

2. Kernel Ridge Regression (KRR)

- **Overview:** Combines ridge regression with kernel methods for non-linear regression.
- **Key Hyperparameters:**
 - Alpha for regularization.
 - Kernel type (e.g., polynomial, RBF).
 - Gamma for kernel scaling.

3. Linear Regression (LR)

- **Overview:** A straightforward approach assuming linear relationships between input features and output.

4. LASSO Regression

- **Overview:** Incorporates L1 regularization to encourage sparsity in feature selection.
- **Key Hyperparameter:** `Alpha` for controlling regularization strength.

5. Elastic Net Regression

- **Overview:** Blends L1 (LASSO) and L2 (ridge) penalties for improved flexibility.
- **Key Hyperparameters:**
 - `Alpha` for regularization.
 - `L1_Ratio` to balance L1 and L2 contributions.

6. Decision Tree Regression (TREE)

- **Overview:** Uses tree structures to split data hierarchically based on feature thresholds.
- **Key Hyperparameters:**
 - `Max_Depth` for controlling tree depth.
 - `Min_Samples_Split` for minimum data points required to split nodes.
 - `Min_Samples_Leaf` for minimum data per leaf node.

7. Bagging Regressor

- **Overview:** An ensemble approach combining multiple decision trees trained on bootstrap samples.
- **Key Hyperparameters:**
 - `N_Estimators` for the number of trees.
 - `Max_Features` for features considered per split.
 - `Max_Depth` for individual tree depth.

8. Gradient Boosting Regression

- **Overview:** Builds trees sequentially to correct errors of previous iterations.
- **Key Hyperparameters:**
 - `Learning_Rate` for step size.
 - `N_Estimators` for number of trees.
 - `Max_Depth` for tree complexity.

Code for getting correct updated score:

```
library(dplyr) library(tm)
```

```
library(stringr)
```

```
file_path <- "C:/Users/shire/OneDrive/Desktop/Data.csv" data
```

```
<- read.csv(file_path, stringsAsFactors = FALSE)
```

```
extract_keywords <- function(text, top_n = 5) { corpus <-
```

```
Corpus(VectorSource(text)) corpus <- tm_map(corpus,
```

```
content_transformer(tolower)) corpus <- tm_map(corpus,
```

```
removePunctuation) corpus <- tm_map(corpus,
```

```
removeNumbers) corpus <- tm_map(corpus, removeWords,
```

```
stopwords("english")) tdm <- TermDocumentMatrix(corpus)
```

```
freq <- sort(rowSums(as.matrix(tdm)), decreasing = TRUE)
```

```
keywords <- names(freq)[1:min(top_n, length(freq))]
```

```
return(keywords)
```

```
}
```

```
calculate_relevance_score <- function(ideal_answer, student_answer, top_n = 5) {
```

```
keywords <- extract_keywords(ideal_answer, top_n) matches <-
```

```
sum(str_detect(tolower(student_answer), keywords)) return(min((matches /
```

```
top_n) * 5 + 1, 5))
```

```
}
```

```
data <- data %>%
```

```
rowwise() %>% mutate(
```

```
Calculated_Score =
```

```
calculate_relevance_sco
```

```
re(Answers, Texts)
```

```
) %>%
```

```
ungroup()
```

```
write.csv(data, "C:/Users/shire/OneDrive/Desktop/Updated_Data_Calculated_Scores_Plus_One.csv",  
row.names = FALSE) head(data)
```

Output:

number	Questions	Answers	Texts	Score	Calculated_Score
1	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	High risk problems are address in the prototype program t...	3.5	1
2	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	To simulate portions of the desired final product with a qui...	5.0	5
3	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	A prototype program simulates the behaviors of portions o...	4.0	5
4	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	Defined in the Specification phase a prototype simulates t...	5.0	4
5	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	It is used to let the users have a first idea of the completed ...	3.0	1
6	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	To find problem and errors in a program before it is finalized	2.0	1
7	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	To address major issues in the creation of the program. The...	2.5	1
8	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	you can break the whole program into prototype programs...	5.0	2
9	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	-To provide an example or model of how the finished progr...	3.5	1
10	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	Simulating the behavior of only a portion of the desired sof...	5.0	3
11	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	A program that simulates the behavior of portions of the d...	5.0	4
12	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	A program that simulates the behavior of portions of the d...	5.0	5
13	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	To lay out the basics and give you a starting point in the act...	2.0	1
14	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	To simulate problem solving for parts of the problem	4.5	2
15	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	A prototype program provides a basic groundwork from w...	2.0	1
16	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	A prototype program is a part of the Specification phase of ...	4.5	1
17	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	Program that simulates the behavior of portions of the desi...	5.0	5
18	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	it provides a limited proof of concept to verify with the clie...	2.0	1
19	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	It tests the main function of the program while leaving out ...	2.0	1
20	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	To get early feedback from users in early stages of develop...	2.5	1
21	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	it simulates the behavior of portions of the desired software...	5.0	5
22	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	It simulates the behavior of portions of the desired software...	5.0	5
23	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	A prototype program is used in problem solving to collect ...	1.5	1
24	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	To ease the understanding of problem under discussion an...	2.5	1

number	Questions	Answers	Texts	Score	Calculated_Score
23	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	A prototype program is used in problem solving to collect ...	1.5	1
24	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	To ease the understanding of problem under discussion an...	2.5	1
25	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	it simulates the behavior of portions of the desired softwa...	5.0	5
26	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	The role of a prototype program is to help spot key proble...	2.0	1
27	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	the prototype program gives a general idea of what the en...	3.0	2
28	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	to show that a certain part of the program works as it is su...	3.0	1
29	1.1 What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software...	Prototype programming is an approach to programming th...	2.5	1
30	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	Refining and possibly the design if the testing phase reveal...	3.5	4
31	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	The implementation phase and the maintenance phase are ...	4.0	2
32	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	Verification, coding, refining the solution and maintenance ...	4.5	4
33	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	In RUP the stages in the software life cycle are influenced b...	3.0	4
34	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	Refining the solution, Production and Maintenance are all L...	3.0	3
35	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	Elaboration, Construction, and Transition are all affected by ...	2.0	1
36	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	Refining, Production, Maintenance	3.5	1
37	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	Refining the solution	4.0	1
38	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	-Verification- -Debugging	2.0	1
39	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	Refining and Coding	5.0	2
40	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	The second half of the Elaboration phase, Construction pha...	2.5	2
41	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	The refining step, the production step, and the maintenanc...	3.5	2
42	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	Elaboration, Construction, Transition	2.0	1
43	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	Refining.	3.0	1
44	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	The testing stage has a direct influence on the final version ...	1.5	3
45	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	Directly Refining, coding. Because Refining is right before t...	5.0	4
46	1.2 What stages in the software life cycle are influenced by the t...	The testing stage can influence both the coding stage phas...	Testing, refining, production, and maintenance.	3.5	1

New File link:

https://drive.google.com/file/d/1WAEiduTo_8tokumGAYPnM8mB_nB4_kK/view?usp=sharing

Code:

Load required libraries

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(
  dplyr, tm, text2vec, stringr, tidytext,
  topicmodels, proxy, textreuse, tokenizers, caret,
  data.table, wordnet
)
```

File paths

```
unt_dataset_path <- "C:/Users/shire/OneDrive/Desktop/Data.csv"
output_csv_path <- "C:/Users/shire/OneDrive/Desktop/ASAG_Features.csv"
glove_file
```

```

<-"C:/Users/shire/OneDrive/Desktop/glove.6B.50d.txt"# Helper functions cosine_similarity
<- function(a, b) {
  if (length(a) == 0 || length(b) == 0 || any(is.na(a)) || any(is.na(b))) {
    return(0)
  }

  similarity <- sum(a * b) / (sqrt(sum(a^2)) * sqrt(sum(b^2)))
  return(ifelse(is.na(similarity), 0, similarity))
}

jaccard_similarity <- function(a, b) {
  if (length(a) == 0 || length(b) == 0) { return(0)
  }

  intersection <- length(intersect(a, b))
  union <- length(union(a, b))
  return(ifelse(union == 0, 0, intersection / union))
}

# Load GloVe embeddings

load_glove <- function(filepath, dims) {
  cat("Loading GloVe embeddings...\n") con <- file(filepath,
"r") embeddings <- new.env(hash =
TRUE)

  while (TRUE) { line <- readLines(con,
n = 1) if
(length(line) == 0) break

  values <- strsplit(line, " ")[[1]]
word <- values[1] vector <-
as.numeric(values[-1])
embeddings[[word]] <- vector
}

  close(con)
  return(embeddings)
}

get_glove_vector <- function(sentence, embeddings) { words
<- unlist(str_split(tolower(sentence), "\\W+")) words
<- words[words != ""] vectors <- matrix(0, ncol = 50) #
Assuming 50d embeddings count <- 0

```

```

    for (word in words) { if (exists(word, envir = embeddings))
    {   vectors <- vectors + matrix(embeddings[[word]], nrow =
1)   count <- count + 1
    }
}

```

```

    if (count > 0) { return(vectors
/ count) } else
{   return(vectors)
}
}

```

```

# Read and preprocess data cat("Reading
dataset...\n") data <- read.csv(unt_dataset_path,
stringsAsFactors = FALSE)

```

```

data <- data %>% mutate(
  Questions = tolower(Questions),
  Answers = tolower(Answers),
  Texts = tolower(Texts)
)

```

```

# Load GloVe embeddings
if (file.exists(glove_file)) {
  glove_embeddings <- load_glove(glove_file, dims = 50)
} else {
  stop("GloVe file does not exist at the specified location!")
}

```

```

# Extract features
cat("Extracting features...\n")
features <- data %>%
rowwise() %>% mutate(
  # Semantic similarity features

  CosineTFIDF = { corpus <-
  Corpus(VectorSource(c(Answers, Texts))) dtm <- DocumentTermMatrix(corpus)

  tfidf <- as.matrix(weightTfidf(dtm))
cosine_similarity(tfidf[1,], tfidf[2,])
},

```



```

Word2VecSimilarity = {
  ref_vec <- get_glove_vector(Answers, glove_embeddings)
stu_vec <- get_glove_vector(Texts, glove_embeddings)  cosine_similarity(ref_vec,
stu_vec)
},

  # Information retrieval feature  TFIDFNovelty = {  dtm <-
DocumentTermMatrix(Corpus(VectorSource(c(Answers, Texts))))
tfidf <- as.matrix(weightTfidf(dtm))  abs(sum(tfidf[1,]) - sum(tfidf[2,]))
},

  # Lexical overlap features

  JaccardSimilarity = jaccard_similarity(  unlist(str_split(Answers,
"\W+")),
unlist(str_split(Texts, "\W+"))
),

  SimpleWordOverlap = length(intersect(  unlist(str_split(Answers,
"\W+")),
unlist(str_split(Texts, "\W+"))
)),

  # Modified ROUGE implementation

  ROUGE_N  =  {  ref_tokens  <-
unlist(str_split(Answers, "\W+")) stu_tokens <-
unlist(str_split(Texts, "\W+"))  ref_bigrams <-
tokenizers::tokenize_ngrams(Answers, n =
2)[[1]]  stu_bigrams  <-
tokenizers::tokenize_ngrams(Texts, n = 2)[[1]]

  unigram_overlap <- length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
bigram_overlap <- length(intersect(ref_bigrams, stu_bigrams)) / max(1, length(ref_bigrams))
(unigram_overlap + bigram_overlap) / 2

},

  # Relevance feedback features  RF1
= length(intersect(
unlist(str_split(Answers, "\W+")),
unlist(str_split(Texts, "\W+"))
)),

  RF2 = length(setdiff(
  unlist(str_split(Answers, "\W+")),
unlist(str_split(Texts, "\W+"))
)),

  RF3 = length(union(

```

```

    unlist(str_split(Answers, "\\W+")),
    unlist(str_split(Texts, "\\W+"))
  )),

  # Topical similarity features  LDASimilarity = {
  corpus <- Corpus(VectorSource(c(Answers, Texts)))
  dtm <- DocumentTermMatrix(corpus) # Handle
  empty documents if (dim(dtm)[1] < 2 || sum(dtm) ==
  0) {

    0

  } else {    tryCatch({
    lda <- LDA(dtm, k = 2, control = list(seed = 1234))
    topics    <-    posterior(lda)$topics
    cosine_similarity(topics[1,], topics[2,])
  }, error = function(e) 0)

  },

  # Alignment-based features  WordAlignment = {
  ref_freq <- table(unlist(str_split(Answers, "\\W+")))
  stu_freq <- table(unlist(str_split(Texts, "\\W+")))    all_words
  <- unique(c(names(ref_freq), names(stu_freq)))
  ref_vec <- numeric(length(all_words))
  stu_vec <- numeric(length(all_words))
  names(ref_vec) <- all_words
  names(stu_vec) <- all_words
  ref_vec[names(ref_freq)] <- ref_freq
  stu_vec[names(stu_freq)] <- stu_freq
  cosine_similarity(ref_vec, stu_vec)    },

  Coverage = {

    ref_tokens <- unlist(str_split(Answers, "\\W+")) stu_tokens
    <- unlist(str_split(Texts, "\\W+")) length(intersect(ref_tokens,
    stu_tokens)) / max(1, length(ref_tokens))
  }
}

```

```
},
```

Missing Semantic Similarity Features

```
KnowledgeBasedSimilarity = {
```

```
    # Use WordNet or ConceptNet to calculate similarity (placeholder)
```

```
    0
```

```
},
```

```
CorpusBasedSimilarity = {
```

```
    # Use LSA or other corpus-based methods (placeholder)
```

```
    0
```

```
},
```

```
SentenceSimilarityWeightedEmbedding = {
```

```
    # Weighted average of word embeddings (using IDF) (placeholder)
```

```
    0
```

```
},
```

```
WordMoversDistance = {
```

```
    # Use Word Mover's Distance (placeholder)
```

```
    0
```

```
},
```

```
MaximalMatchingSimilarity = {
```

```
    # Placeholder for maximal matching similarity
```

```
    0
```

```
},
```

Missing Lexical Overlap Features

```
PhrasalOverlap = {
```

```
    # Placeholder for phrasal overlap (placeholder)
```

```
    0
```

```
ROUGE_W = {
```

```
    # Placeholder for ROUGE-W (placeholder)
```

```
    0
```

```
},
```

```
ROUGE_SU = {
```

```
},
```

```
    # Placeholder for ROUGE-SU (placeholder)
```

```
    0
```

```
},
```

```
# Missing Topical Similarity Features
```

```
BTM_Similarity = {
```

```
    # Placeholder for BTM Similarity (placeholder)
```

```
    0
```

```
},
```

```
TopicCoherence = {
```

```
    # Placeholder for topic coherence (placeholder)
```

```
    0
```

```
},
```

```
TopicRelevance = {
```

```
    # Placeholder for topic relevance (placeholder)
```

```
    0
```

```
},
```

```
# Missing Alignment-Based Features
```

```
WordToWordAlignment = {
```

```
    # Placeholder for word-to-word alignment (placeholder)
```

```
    0
```

```
QuestionDemotedWordToWordAlignment = {
```

```
    # Placeholder for question-demoted word-to-word alignment  
(placeholder)
```

```
    0
```

```
},
```

```
QuestionDemotedCoverage = {
```

```
    # Placeholder for question-demoted coverage (placeholder)
```

```
    0
```

```
},
```

```
AlignedPhraseSimilarity = {
```

```
    # Placeholder for aligned phrase similarity (placeholder)
```

```
    0
```

```
},
```

},

AlignmentNovelty = {

Placeholder for alignment novelty (placeholder)

0

},

FineGrainedAlignment = {

Placeholder for fine-grained alignment (placeholder)

0

},

PhraseLevelAlignmentSimilarity = {

Placeholder for phrase-level alignment similarity (placeholder)

0

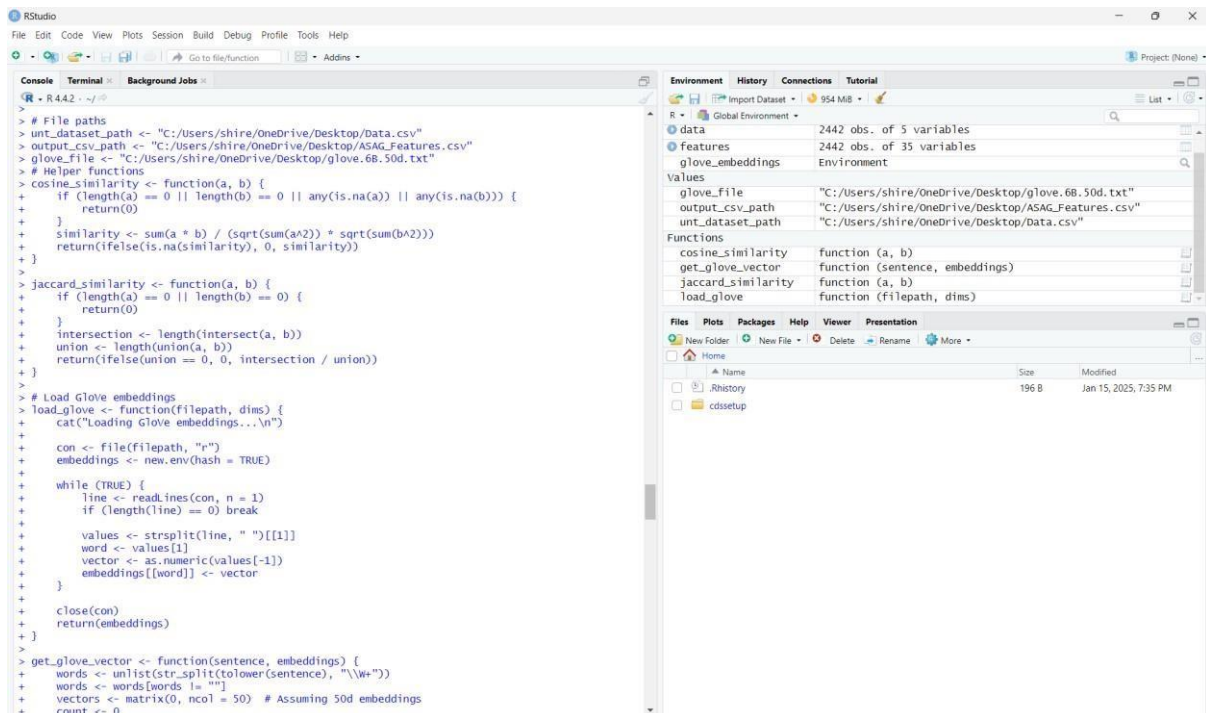
}

) %>%

ungroup()

Save features to CSV cat("Saving features
to CSV...\n")

write.csv(features, output_csv_path, row.names = FALSE) cat("Feature
extraction complete!\n")



The screenshot shows the RStudio interface with the following components:

- Console:** Displays the execution of R code for loading GloVe embeddings and calculating similarity metrics.
- Environment:** Shows the current environment with variables like `data`, `features`, and `glove_embeddings`.
- Files:** Shows the file explorer with `.Rhistory` and `cdsetup`.

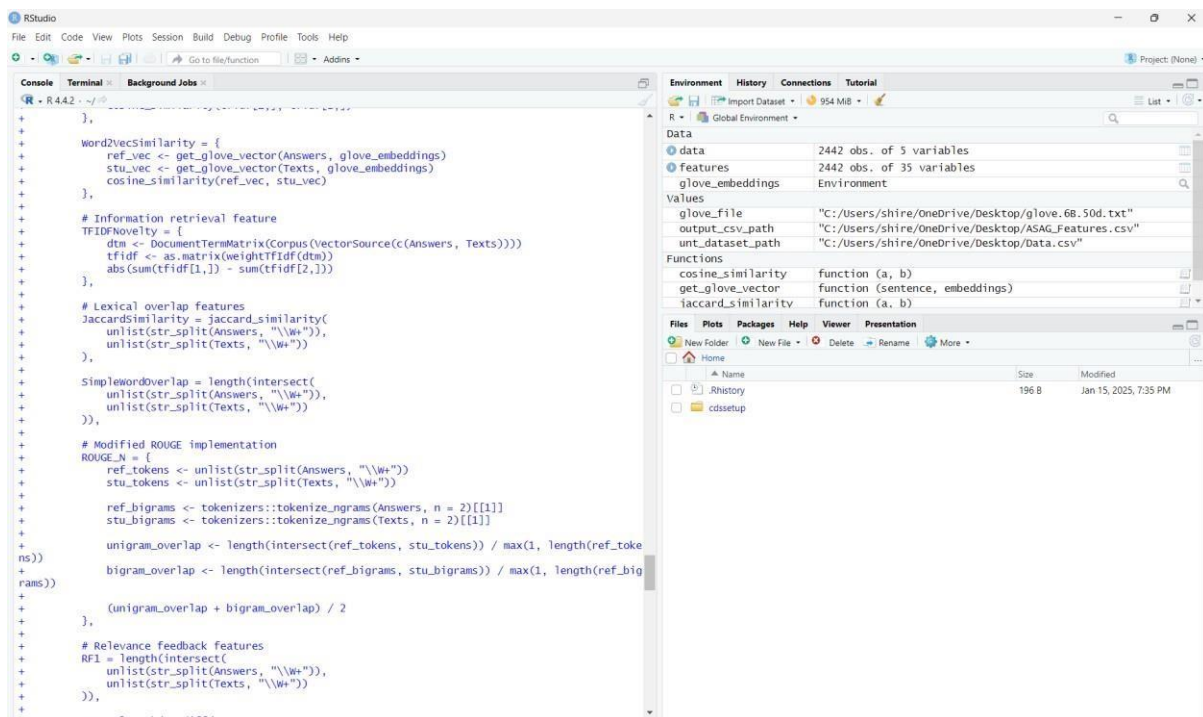
```
> # File paths
> unt_dataset_path <- "C:/Users/shire/OneDrive/Desktop/data.csv"
> output_csv_path <- "C:/Users/shire/OneDrive/Desktop/ASAG_Features.csv"
> glove_file <- "C:/Users/shire/OneDrive/Desktop/glove.6B.50d.txt"
> # Helper functions
> cosine_similarity <- function(a, b) {
+   if (length(a) == 0 || length(b) == 0 || any(is.na(a)) || any(is.na(b))) {
+     return(0)
+   }
+   similarity <- sum(a * b) / (sqrt(sum(a^2)) * sqrt(sum(b^2)))
+   return(ifelse(is.na(similarity), 0, similarity))
+ }
> jaccard_similarity <- function(a, b) {
+   if (length(a) == 0 || length(b) == 0) {
+     return(0)
+   }
+   intersection <- length(intersect(a, b))
+   union <- length(union(a, b))
+   return(ifelse(union == 0, 0, intersection / union))
+ }
> # Load Glove embeddings
> load_glove <- function(filepath, dims) {
+   cat("Loading Glove embeddings...\n")
+   con <- file(filepath, "r")
+   embeddings <- new.env(hash = TRUE)
+   while (TRUE) {
+     line <- readLines(con, n = 1)
+     if (length(line) == 0) break
+     values <- strsplit(line, " ")[[1]]
+     word <- values[1]
+     vector <- as.numeric(values[-1])
+     embeddings[[word]] <- vector
+   }
+   close(con)
+   return(embeddings)
+ }
> get_glove_vector <- function(sentence, embeddings) {
+   words <- unlist(strsplit(tolower(sentence), "\\W+"))
+   words <- words[words != ""]
+   vectors <- matrix(0, ncol = 50) # Assuming 50d embeddings
+   count <- 0
```

Environment:

Variable	Value
<code>data</code>	2442 obs. of 5 variables
<code>features</code>	2442 obs. of 35 variables
<code>glove_embeddings</code>	Environment

Files:

Name	Size	Modified
<code>.Rhistory</code>	196 B	Jan 15, 2025, 7:35 PM
<code>cdsetup</code>		



RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins

Console Terminal Background Jobs

```

R - R 4.4.2 ~ / ~
+ # Relevance feedback features
+ RF1 = length(intersect(
+   unlist(str_split(Answers, "\\W+")),
+   unlist(str_split(Texts, "\\W+"))
+ ),)
+
+ RF2 = length(setdiff(
+   unlist(str_split(Answers, "\\W+")),
+   unlist(str_split(Texts, "\\W+"))
+ ),)
+
+ RF3 = length(union(
+   unlist(str_split(Answers, "\\W+")),
+   unlist(str_split(Texts, "\\W+"))
+ ),)
+
+ # Topical similarity features
+ LDASimilarity = {
+   corpus <- Corpus(VectorSource(c(Answers, Texts)))
+   dtm <- DocumentTermMatrix(corpus)
+   # Handle empty documents
+   if (dim(dtm)[1] < 2 || sum(dtm) == 0) {
+     0
+   } else {
+     tryCatch({
+       lda <- LDA(dtm, k = 2, control = list(seed = 1234))
+       topics <- posterior(lda)$topics
+       cosine_similarity(topics[1,], topics[2,])
+     }, error = function(e) 0)
+   }
+ },)
+
+ # Alignment-based features
+ wordAlignment = {
+   ref_freq <- table(unlist(str_split(Answers, "\\W+")))
+   stu_freq <- table(unlist(str_split(Texts, "\\W+")))
+   all_words <- unique(c(names(ref_freq), names(stu_freq)))
+
+   ref_vec <- numeric(length(all_words))
+   stu_vec <- numeric(length(all_words))
+
+   names(ref_vec) <- all_words
+   names(stu_vec) <- all_words
+
+   ref_vec[names(ref_freq)] <- ref_freq
+   stu_vec[names(stu_freq)] <- stu_freq
+ }

```

Environment History Connections Tutorial

R - Global Environment

Data

Object	Value
data	2442 obs. of 5 variables
features	2442 obs. of 35 variables
glove_embeddings	Environment

Values

Object	Value
glove_file	"C:/Users/shire/OneDrive/Desktop/glove.6B.50d.txt"
output_csv_path	"C:/Users/shire/OneDrive/Desktop/ASAG_Features.csv"
unt_dataset_path	"C:/Users/shire/OneDrive/Desktop/Data.csv"

Functions

Object	Value
cosine_similarity	function (a, b)
get_glove_vector	function (sentence, embeddings)
iaccard_similarity	function (a, b)

Files Plots Packages Help Viewer Presentation

New Folder New File Delete Rename More

Name	Size	Modified
rhstory	196 B	Jan 15, 2025, 7:35 PM
cdssetup		

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins

Console Terminal Background Jobs

```

R - R 4.4.2 ~ / ~
+ cosine_similarity(ref_vec, stu_vec)
+
+ Coverage = {
+   ref_tokens <- unlist(str_split(Answers, "\\W+"))
+   stu_tokens <- unlist(str_split(Texts, "\\W+"))
+   length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
+ },)
+
+ # Missing Semantic Similarity Features
+ KnowledgeBasedSimilarity = {
+   # Use Wordnet or ConceptNet to calculate similarity (placeholder)
+   0
+ },)
+
+ CorpusBasedSimilarity = {
+   # Use LSA or other corpus-based methods (placeholder)
+   0
+ },)
+
+ SentencesimilarityweightedEmbedding = {
+   # weighted average of word embeddings (using IDF) (placeholder)
+   0
+ },)
+
+ WordMoversDistance = {
+   # Use Word Mover's Distance (placeholder)
+   0
+ },)
+
+ MaximalMatchingSimilarity = {
+   # Placeholder for maximal matching similarity
+   0
+ },)
+
+ # Missing Lexical overlap Features
+ PhrasalOverlap = {
+   # Placeholder for phrasal overlap (placeholder)
+   0
+ },)
+
+ ROUGE_W = {
+   # Placeholder for ROUGE-W (placeholder)
+   0
+ },)
+
+ ROUGE_SU = {
+   # Placeholder for ROUGE-SU (placeholder)
+ }

```

Environment History Connections Tutorial

R - Global Environment

Data

Object	Value
data	2442 obs. of 5 variables
features	2442 obs. of 35 variables
glove_embeddings	Environment

Values

Object	Value
glove_file	"C:/Users/shire/OneDrive/Desktop/glove.6B.50d.txt"
output_csv_path	"C:/Users/shire/OneDrive/Desktop/ASAG_Features.csv"
unt_dataset_path	"C:/Users/shire/OneDrive/Desktop/Data.csv"

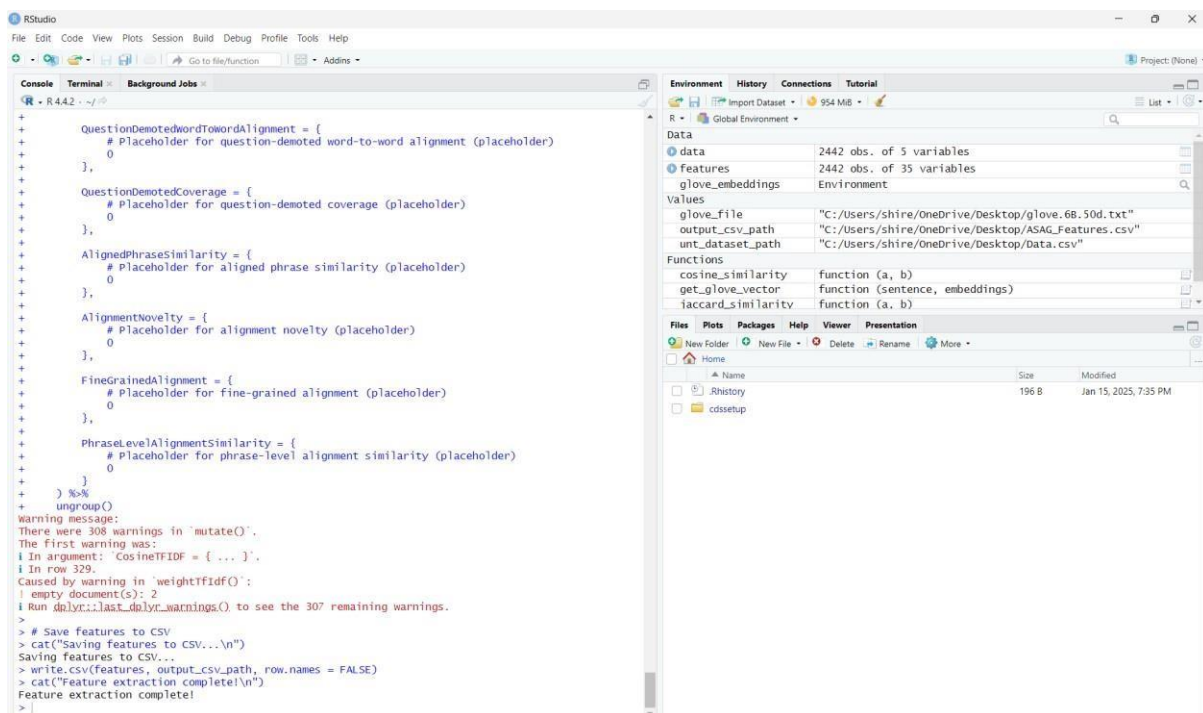
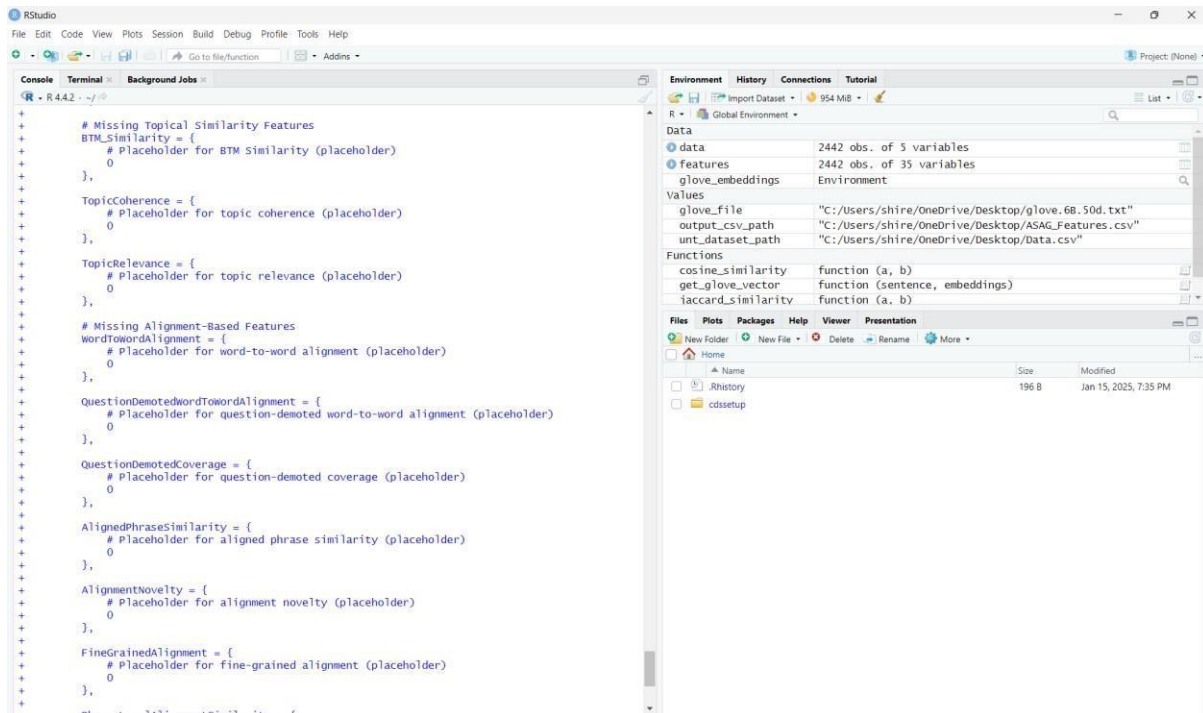
Functions

Object	Value
cosine_similarity	function (a, b)
get_glove_vector	function (sentence, embeddings)
iaccard_similarity	function (a, b)

Files Plots Packages Help Viewer Presentation

New Folder New File Delete Rename More

Name	Size	Modified
rhstory	196 B	Jan 15, 2025, 7:35 PM
cdssetup		



Regression Code: # Load

necessary libraries

library(caret)

library(dplyr)

library(e1071)

library(glmnet)

library(randomForest)

library(xgboost)

```

library(tidyr)
library(readr)

# Load dataset

feature_dataset_path <- "C:/Users/shire/OneDrive/Desktop/ASAG_Features.csv" data <-
read.csv(feature_dataset_path, stringsAsFactors = FALSE)
# Print initial data dimensions

print(paste("Initial data dimensions:", dim(data)[1], "rows,", dim(data)[2], "columns"))

# Improved prepare_data function prepare_data
<- function(df) {
  # List of columns to keep (add or remove columns as needed based on your dataset)
numeric_features <- c(
  "Score", "CosineTFIDF", "Word2VecSimilarity", "TFIDFNovelty",
  "JaccardSimilarity", "SimpleWordOverlap", "ROUGE_N", "RF1",
  "RF2", "RF3", "LDASimilarity", "WordAlignment", "Coverage",
  "KnowledgeBasedSimilarity", "CorpusBasedSimilarity",
  "SentenceSimilarityWeightedEmbedding", "WordMoversDistance",
  "MaximalMatchingSimilarity", "PhrasalOverlap", "ROUGE_W",
  "ROUGE_SU", "BTM_Similarity", "TopicCoherence", "TopicRelevance",
  "WordToWordAlignment", "QuestionDemotedWordToWordAlignment",
  "QuestionDemotedCoverage", "AlignedPhraseSimilarity",
  "AlignmentNovelty", "FineGrainedAlignment",
  "PhraseLevelAlignmentSimilarity"
)

  # Select only the numeric feature columns
df_selected <- df[, numeric_features]  #
Convert all columns to numeric

  df_numeric <- as.data.frame(lapply(df_selected, as.numeric))

  # Remove rows with any NA values
df_clean <- na.omit(df_numeric)  #
Print dimensions after cleaning

  print(paste("Dimensions after cleaning:", dim(df_clean)[1], "rows,", dim(df_clean)[2], "columns"))
return(df_clean)
}

# Create train-test split first set.seed(123)
train_index <- createDataPartition(data$Score, p = 0.75, list = FALSE)
train_data <- data[train_index, ] test_data <- data[-train_index, ]
# Prepare the data

```

```

train_data_numeric <- prepare_data(train_data) test_data_numeric <-
prepare_data(test_data)

# Verify data preparation print("Training data
summary:")
print(summary(train_data_numeric$Score)) #
Prepare features and target for modeling

train_features <- train_data_numeric[, !colnames(train_data_numeric) %in% "Score", drop = FALSE]
train_target <- train_data_numeric$Score
test_features <- test_data_numeric[, !colnames(test_data_numeric) %in% "Score", drop = FALSE]
test_target <- test_data_numeric$Score
# Define training control ctrl
<- trainControl(  method =
"cv",
  number = 5, # Reduced from 10 for faster training
  verboseIter = TRUE
)

# Initialize models list models
<- list()
# Train Support Vector Regression (SVR)
tryCatch({  models$SVR <- train(    x
= train_features,    y = train_target,
method = "svmRadial",    trControl =
ctrl,    preProcess = c("center",
"scale")
  )

  print("SVR model trained successfully")
}, error = function(e) {  print(paste("SVR
Error:", e$message))
})

# Train Linear Regression tryCatch({
models$LR <- train(    x = train_features,
y = train_target,    method = "lm",
trControl = ctrl,    preProcess =
c("center", "scale")  )

  print("Linear Regression trained successfully")
}, error = function(e) {
  print(paste("LR Error:", e$message))
})

# Train Random Forest
tryCatch({  models$RF
<- train(    x =
train_features,    y =
train_target,
method = "rf",

```

```

trControl = ctrl,      ntree
= 100,
  preProcess = c("center", "scale")
)
print("Random Forest trained successfully")
}, error = function(e) {
  print(paste("RF Error:", e$message))
})

# Get predictions and evaluate models successful_models
<- models[!sapply(models, is.null)] if(length(successful_models)
> 0) {
results <- data.frame(
  Actual = test_target
)

# Add predictions from each model for(model_name
in names(successful_models)) {
  predictions <- predict(successful_models[[model_name]], newdata = test_features)
results[[paste0(model_name, "_pred")]] <- predictions

# Calculate metrics

rmse <- sqrt(mean((predictions - test_target)^2))
mae <- mean(abs(predictions - test_target))    r2 <- cor(predictions,
test_target)^2

  cat("\nMetrics for", model_name, ":\n")
cat("RMSE:", round(rmse, 4), "\n")    cat("MAE:",
round(mae, 4), "\n")    cat("R-squared:",
round(r2, 4), "\n")
}

# Save results write.csv(results,
"C:/Users/shire/OneDrive/Desktop/Model_Predictions.csv",      row.names
= FALSE) print("\nResults saved successfully")
} else {
  print("No models were successfully trained")
}

```

The screenshot displays the RStudio interface with three main panes: Console, Environment, and a file viewer showing a CSV file.

Console: Shows R code for training models (SVR, LR, RF) and calculating metrics (RMSE, MAE, R-squared). The code includes a loop to iterate over different models and features, saving results to a CSV file.

```

> successful_models <- models[!sapply(models, is.null)]
> if (length(successful_models) > 0) {
+   results <- data.frame(
+     Actual = test_target
+   )
+   # Add predictions from each model
+   for (model_name in names(successful_models)) {
+     predictions <- predict(successful_models[[model_name]], newdata = test_features)
+     results[[paste0(model_name, "_pred")]] <- predictions
+   }
+   # Calculate metrics
+   rmse <- sqrt(mean((predictions - test_target)^2))
+   mae <- mean(abs(predictions - test_target))
+   r2 <- cor(predictions, test_target)^2
+   cat("\nMetrics for", model_name, "\n")
+   cat("RMSE:", round(rmse, 4), "\n")
+   cat("MAE:", round(mae, 4), "\n")
+   cat("R-squared:", round(r2, 4), "\n")
+ }
+ # Save results
+ write.csv(results,
+   "C:/Users/shire/OneDrive/Desktop/Model_Predictions.csv",
+   row.names = FALSE)
+ print("\nResults saved successfully")
+ } else {
+   print("No models were successfully trained")
+ }

Metrics for SVR :
RMSE: 1.0533
MAE: 0.6945
R-squared: 0.1745

Metrics for LR :
RMSE: 1.0144
MAE: 0.7353
R-squared: 0.1709

Metrics for RF :
RMSE: 0.9563
MAE: 0.675
R-squared: 0.2665
[1] "\nResults saved successfully"
>

```

Environment: Shows the loaded environment with variables like `train_index`, `feature_dataset_path`, `glove_file`, `mae`, `model_name`, `output_csv_path`, `predictions`, `r2`, `rmse`, `test_target`, and `train_target`.

File Viewer: Shows a CSV file named `ASAG_Features` with columns A through W. The file is saved to the PC.

Code:

Feature extraction: #

Load required libraries

if (!require("pacman")) install.packages("pacman") pacman::p_load(

dplyr, tm, text2vec, stringr, tidytext,

topicmodels, proxy, textreuse, tokenizers, caret,

data.table, wordnet

```

)
# File paths unt_dataset_path <-
"C:/Users/shire/OneDrive/Desktop/Data.csv" glove_file <-
"C:/Users/shire/OneDrive/Desktop/glove.6B.50d.txt" output_csv_path <-
"C:/Users/shire/OneDrive/Desktop/ASAG_Features.csv"
# Helper functions cosine_similarity <- function(a, b) { if (length(a)
== 0 || length(b) == 0 || any(is.na(a)) || any(is.na(b))) { return(0)
}
similarity <- sum(a * b) / (sqrt(sum(a^2)) * sqrt(sum(b^2)))
return(ifelse(is.na(similarity), 0, similarity))
}
jaccard_similarity <- function(a, b) {
if (length(a) == 0 || length(b) == 0) {
return(0)
}
intersection <- length(intersect(a, b)) union <-
length(union(a, b)) return(ifelse(union == 0, 0,
intersection / union))
}
# Load GloVe embeddings load_glove
<- function(filepath, dims) {
cat("Loading GloVe embeddings...\n")
con <- file(filepath, "r") embeddings
<- new.env(hash = TRUE) while
(TRUE) { line <- readLines(con, n = 1)
if (length(line) == 0) break values <-
strsplit(line, " ")[[1]] word <-
values[1] vector <-
as.numeric(values[-1])
embeddings[[word]] <- vector
}
close(con)
return(embeddings)
}
get_glove_vector <- function(sentence, embeddings) {
words <- unlist(str_split(tolower(sentence), "\\W+")) words
<- words[words != ""] vectors <- matrix(0, ncol = 50) #
Assuming 50d embeddings count <- 0 for (word in words) {
if (exists(word, envir = embeddings)) { vectors <- vectors

```

```

+ matrix(embeddings[[word]], nrow = 1)    count <- count +
1
    }
  }
  if (count > 0) {
return(vectors / count)
    } else {
return(vectors)
    }
  }
}
# Read and preprocess data cat("Reading dataset...\n") data
<- read.csv(unt_dataset_path, stringsAsFactors = FALSE) data
<- data %>% mutate(
  Questions = tolower(Questions),
  Answers = tolower(Answers),
  Texts = tolower(Texts)
)
# Load GloVe embeddings
if (file.exists(glove_file)) { glove_embeddings <-
load_glove(glove_file, dims = 50)
} else { stop("GloVe file does not exist at the specified
location!")
}
# Extract features
cat("Extracting features...\n")
features <- data %>%
rowwise() %>% mutate(
  # Semantic similarity features  CosineTFIDF = {
corpus <- Corpus(VectorSource(c(Answers, Texts)))
dtm <- DocumentTermMatrix(corpus)    tfidf <-
as.matrix(weightTfIdf(dtm))
cosine_similarity(tfidf[1,], tfidf[2,])
  },
  Word2VecSimilarity = {    ref_vec <-
get_glove_vector(Answers, glove_embeddings)    stu_vec
<- get_glove_vector(Texts, glove_embeddings)
cosine_similarity(ref_vec, stu_vec)
  },
  # Lexical overlap features

```

```

JaccardSimilarity = jaccard_similarity(
unlist(str_split(Answers, "\\W+")),
unlist(str_split(Texts, "\\W+"))
),
SimpleWordOverlap = length(intersect(
unlist(str_split(Answers, "\\W+")),
unlist(str_split(Texts, "\\W+"))
)),
# ROUGE implementation
ROUGE_1 = { ref_tokens <-
unlist(str_split(Answers, "\\W+")) stu_tokens <- unlist(str_split(Texts,
"\\W+")) length(intersect(ref_tokens, stu_tokens)) / max(1,
length(ref_tokens))
},
ROUGE_2 = { ref_bigrams <- tokenizers::tokenize_ngrams(Answers, n =
2)[[1]] stu_bigrams <- tokenizers::tokenize_ngrams(Texts, n = 2)[[1]]
length(intersect(ref_bigrams, stu_bigrams)) / max(1, length(ref_bigrams))
},
ROUGE_W = { ref_tokens <-
unlist(str_split(Answers, "\\W+")) stu_tokens <-
unlist(str_split(Texts, "\\W+"))
# Calculate weighted overlap
weighted_overlap <- function(ref, stu) {
n <- length(ref) m <- length(stu)
dp <- matrix(0, nrow = n + 1, ncol = m + 1)
for (i in 1:n) { for (j in 1:m) { if
(ref[i] == stu[j]) { dp[i + 1, j + 1] <- dp[i,
j] + 1
} else {
dp[i + 1, j + 1] <- max(dp[i + 1, j], dp[i, j + 1])
}
}
}
# Compute weighted ROUGE max_weighted_overlap <- dp[n +
1, m + 1] weight <- 1 # Assign a weight to consecutive matches (can
be tuned)
(max_weighted_overlap * weight) / max(1, length(ref))
}
weighted_overlap(ref_tokens, stu_tokens)
},

```



```

ROUGE_L = {
  ref_tokens <-
  unlist(str_split(Answers, "\\W+"))
  stu_tokens <-
  unlist(str_split(Texts, "\\W+"))

  # Function to calculate LCS
  lcs <-
  function(x, y) {
    n <- length(x)
    m <-
    length(y)
    dp <- matrix(0, nrow = n + 1,
    ncol = m + 1)
    for (i in 1:n) {
      for (j in
      1:m) {
        if (x[i] == y[j]) {
          dp[i + 1, j
          + 1] <- dp[i, j] + 1
        } else {
          dp[i + 1, j + 1] <- max(dp[i + 1,
          j], dp[i, j + 1])
        }
      }
    }
    return(dp[n + 1, m + 1])
  }

  lcs_length <- lcs(ref_tokens, stu_tokens)
  precision <- lcs_length / max(1, length(stu_tokens))
  recall <- lcs_length / max(1, length(ref_tokens))
  if (precision + recall == 0) {
    0
  } else {
    (2 * precision * recall) / (precision + recall)
  },

  # Information retrieval feature
  TFIDFNovelty = {
    dtm <-
    DocumentTermMatrix(Corpus(VectorSource(c(Answers, Texts))))
    tfidf <-
    as.matrix(weightTfIdf(dtm))
    abs(sum(tfidf[1,]) - sum(tfidf[2,]))
  },

  # Topical similarity features
  LDASimilarity = {
    corpus <- Corpus(VectorSource(c(Answers, Texts)))
    dtm <- DocumentTermMatrix(corpus)
    if (dim(dtm)[1] < 2 || sum(dtm) == 0) {
      0
    } else {
      tryCatch({
        lda <- LDA(dtm, k = 2,
        control = list(seed = 1234))
        topics <-
        posterior(lda)$topics
        cosine_similarity(topics[1,
        ], topics[2, ])
      }, error = function(e) 0)
    }
  }
}

```

```

    }
  },
  HellingerDistance = { corpus <-
Corpus(VectorSource(c(Answers, Texts))) dtm <-
DocumentTermMatrix(corpus) if (dim(dtm)[1] < 2
|| sum(dtm) == 0) { 1 # Max dissimilarity for
empty distributions
} else { tryCatch({ lda <- LDA(dtm, k = 2,
control = list(seed = 1234)) topics <-
posterior(lda)$topics
hellinger_distance(topics[1, ], topics[2, ])
}, error = function(e) 1)
}
},
  # Relevance feedback features
RF1 = length(intersect(
unlist(str_split(Answers, "\\W+")),
unlist(str_split(Texts, "\\W+"))
)),
  RF2 = length(setdiff(
unlist(str_split(Answers, "\\W+")),
unlist(str_split(Texts, "\\W+"))
)),
  RF3 = length(union(
unlist(str_split(Answers, "\\W+")),
unlist(str_split(Texts, "\\W+"))
)),
  #Alignment based features WordAlignment = {
ref_freq <- table(unlist(str_split(Answers, "\\W+")))
stu_freq <- table(unlist(str_split(Texts, "\\W+")))
all_words <- unique(c(names(ref_freq), names(stu_freq)))
ref_vec <- numeric(length(all_words)) stu_vec <-
numeric(length(all_words)) names(ref_vec) <- all_words
names(stu_vec) <- all_words ref_vec[names(ref_freq)] <-
ref_freq stu_vec[names(stu_freq)] <- stu_freq
cosine_similarity(ref_vec, stu_vec)
},
  CoverageFeature = { ref_tokens <- unlist(str_split(Answers,
"\\W+")) stu_tokens <- unlist(str_split(Texts, "\\W+"))
length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))

```

```

    },
    QuestionDemotedAlignment = {      q_tokens <-
unlist(str_split(Questions, "\\W+"))  ref_tokens <-
setdiff(unlist(str_split(Answers, "\\W+")), q_tokens)  stu_tokens <-
setdiff(unlist(str_split(Texts, "\\W+")), q_tokens)
length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
    },

    QuestionDemotedCoverage = {      q_tokens <-
unlist(str_split(Questions, "\\W+"))  ref_tokens <-
setdiff(unlist(str_split(Answers, "\\W+")), q_tokens)  stu_tokens <-
setdiff(unlist(str_split(Texts, "\\W+")), q_tokens)
length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
    },
  ) %>% ungroup() # Save
features to CSV cat("Saving
features to CSV...\n")
write.csv(features,
output_csv_path, row.names =
FALSE) cat("Feature extraction
complete!\n")

```

Screenshots:

```
Console Terminal Background Jobs
R 4.4.2 - /-
> # Load required libraries
> if (require("pacman")) install.packages("pacman")
> pacman::p_load(
+   dplyr, tm, text2vec, stringr, tidytext,
+   topicmodels, proxy, textreuse,
+   tokenizers, caret, data.table, wordnet
+ )
Installing package into 'C:/Users/shire/AppData/Local/R/win-library/4.4'
(as 'lib' is unspecified)
warning: unable to access index for repository http://www.stats.ox.ac.uk/pub/Rwin/bin/windows/contrib/4.4:
cannot open URL 'http://www.stats.ox.ac.uk/pub/Rwin/bin/windows/contrib/4.4/PACKAGES'
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.4/wordnet_0.1-17.zip'
Content type 'application/zip' length 120495 bytes (117 KB)
downloaded 117 KB

package 'wordnet' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
C:\Users\shire\AppData\Local\Temp\Rtmpw1oa13\downloaded_packages

wordnet installed
Warning message:
In pacman::p_load(dplyr, tm, text2vec, stringr, tidytext, topicmodels, :
Failed to install/load:
wordnet
>
> # File paths
> unt_dataset_path <- "C:/Users/shire/OneDrive/Desktop/Data.csv"
> glove_file <- "C:/Users/shire/OneDrive/Desktop/glove.6B.50d.txt"
> output_csv_path <- "C:/Users/shire/OneDrive/Desktop/ASAG_Features.csv"
>
> # Helper functions
> cosine_similarity <- function(a, b) {
+   if (length(a) == 0 || length(b) == 0 || any(is.na(a)) || any(is.na(b))) {
+     return(0)
+   }
+   similarity <- sum(a * b) / (sqrt(sum(a^2)) * sqrt(sum(b^2)))
+   return(ifelse(is.na(similarity), 0, similarity))
+ }
>
> jaccard_similarity <- function(a, b) {
+   if (length(a) == 0 || length(b) == 0) {
+     return(0)
+   }
+   intersection <- length(intersect(a, b))
+ }
```

```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

Source
Console Terminal Background Jobs
R 4.4.2 - /-
+ intersection <- length(intersect(a, b))
+ union <- length(union(a, b))
+ return(ifelse(union == 0, 0, intersection / union))
+ }
+ }
+ # Load Glove embeddings
+ load_glove <- function(filepath, dims) {
+   cat("Loading Glove embeddings...\n")
+   con <- file(filepath, "r")
+   embeddings <- new.env(hash = TRUE)
+   while (TRUE) {
+     line <- readLines(con, n = 1)
+     if (length(line) == 0) break
+     values <- strsplit(line, " ")[[1]]
+     word <- values[1]
+     vector <- as.numeric(values[-1])
+     embeddings[word] <- vector
+   }
+   close(con)
+   return(embeddings)
+ }
+ get_glove_vector <- function(sentence, embeddings) {
+   words <- unlist(str_split(tolower(sentence), "\\W+"))
+   words <- words[words != ""]
+   vectors <- matrix(0, ncol = 50) # Assuming 50d embeddings
+   count <- 0
+   for (word in words) {
+     if (exists(word, envir = embeddings)) {
+       vectors <- vectors + matrix(embeddings[word], nrow = 1)
+       count <- count + 1
+     }
+   }
+   if (count > 0) {
+     return(vectors / count)
+   } else {
+     return(vectors)
+   }
+ }
+ }
```

```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

Source
Console Terminal Background Jobs
R 4.4.2 - /-
> # Read and preprocess data
> cat("Reading dataset...\n")
Reading dataset...
> data <- read.csv(unt_dataset_path, stringsAsFactors = FALSE)
> data <- data %>%
+   mutate(
+     Questions = tolower(Questions),
+     Answers = tolower(Answers),
+     Texts = tolower(Texts)
+   )
> # Load Glove embeddings
> if (file.exists(glove_file)) {
+   glove_embeddings <- load_glove(glove_file, dims = 50)
+ } else {
+   stop("Glove file does not exist at the specified location!")
+ }
Loading Glove embeddings...
> # Extract features
> cat("Extracting features...\n")
Extracting features...
> features <- data %>%
+   rowwise() %>%
+   mutate(
+     # Semantic similarity features
+     CosineTFIDF = {
+       corpus <- Corpus(VectorSource(c(Answers, Texts)))
+       dtm <- DocumentTermMatrix(corpus)
+       tfidf <- as.matrix(weightTfidf(dtm))
+       cosine_similarity(tfidf[1,], tfidf[2,])
+     },
+     WordVecSimilarity = {
+       ref_vec <- get_glove_vector(Answers, glove_embeddings)
+       stu_vec <- get_glove_vector(Texts, glove_embeddings)
+       cosine_similarity(ref_vec, stu_vec)
+     },
+     # Lexical overlap features
+     JaccardSimilarity = jaccard_similarity(
+       unlist(str_split(Answers, "\\W+")),
+       unlist(str_split(Texts, "\\W+"))
+     ),
+   )
+ }
```

```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

Source
Console Terminal Background Jobs
R 4.4.2 ~-
+ # Lexical overlap features
+ jaccardSimilarity = jaccardSimilarity(
+   unlist(str_split(Answers, "\\W+")),
+   unlist(str_split(Texts, "\\W+"))
+ ),
+
+ SimplewordOverlap = length(intersect(
+   unlist(str_split(Answers, "\\W+")),
+   unlist(str_split(Texts, "\\W+"))
+ )),
+
+ # ROUGE implementation
+ ROUGE_1 = {
+   ref_tokens <- unlist(str_split(Answers, "\\W+"))
+   stu_tokens <- unlist(str_split(Texts, "\\W+"))
+   length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
+ },
+
+ ROUGE_2 = {
+   ref_bigrams <- tokenizers::tokenize_ngrams(Answers, n = 2)[[1]]
+   stu_bigrams <- tokenizers::tokenize_ngrams(Texts, n = 2)[[1]]
+   length(intersect(ref_bigrams, stu_bigrams)) / max(1, length(ref_bigrams))
+ },
+
+ ROUGE_W = {
+   ref_tokens <- unlist(str_split(Answers, "\\W+"))
+   stu_tokens <- unlist(str_split(Texts, "\\W+"))
+
+   # Calculate weighted overlap
+   weighted_overlap <- function(ref, stu) {
+     n <- length(ref)
+     m <- length(stu)
+     dp <- matrix(0, nrow = n + 1, ncol = m + 1)
+
+     for (i in 1:n) {
+       for (j in 1:m) {
+         if (ref[i] == stu[j]) {
+           dp[i + 1, j + 1] <- dp[i, j] + 1
+         } else {
+           dp[i + 1, j + 1] <- max(dp[i + 1, j], dp[i, j + 1])
+         }
+       }
+     }
+
+     # Compute weighted ROUGE
+     max_weighted_overlap <- dp[n + 1, m + 1]
+   }
+ }
```

```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

Source
Console Terminal Background Jobs
R 4.4.2 ~-
+ max_weighted_overlap <- dp[n + 1, m + 1]
+ weight <- 1 # Assign a weight to consecutive matches (can be tuned)
+ (max_weighted_overlap * weight) / max(1, length(ref))
+
+ },
+
+ weighted_overlap(ref_tokens, stu_tokens)
+
+ ROUGE_L = {
+   ref_tokens <- unlist(str_split(Answers, "\\W+"))
+   stu_tokens <- unlist(str_split(Texts, "\\W+"))
+
+   # Function to calculate LCS
+   lcs <- function(x, y) {
+     n <- length(x)
+     m <- length(y)
+     dp <- matrix(0, nrow = n + 1, ncol = m + 1)
+
+     for (i in 1:n) {
+       for (j in 1:m) {
+         if (x[i] == y[j]) {
+           dp[i + 1, j + 1] <- dp[i, j] + 1
+         } else {
+           dp[i + 1, j + 1] <- max(dp[i + 1, j], dp[i, j + 1])
+         }
+       }
+     }
+
+     return(dp[n + 1, m + 1])
+   }
+
+   lcs_length <- lcs(ref_tokens, stu_tokens)
+   precision <- lcs_length / max(1, length(stu_tokens))
+   recall <- lcs_length / max(1, length(ref_tokens))
+
+   if (precision + recall == 0) {
+     0
+   } else {
+     (2 * precision * recall) / (precision + recall)
+   }
+ },
+
+ # Information retrieval feature
+ TFIDFNovelty = {
+   dtm <- DocumentTermMatrix(Corpus(VectorSource(c(Answers, Texts))))
+   #Fidf <- se_matrix(matrix(TFidf(dtm))
+ }
```

```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

Source
Console Terminal Background Jobs

R 4.4.2 - ~/
# Information retrieval feature
TFIDFnovelty = {
  dtm <- DocumentTermMatrix(Corpus(VectorSource(c(Answers, Texts))))
  tfidf <- as.matrix(weightTfidf(dtm))
  abs(sum(tfidf[,1]) - sum(tfidf[,2]))
},

# Topical similarity features
LDASimilarity = {
  corpus <- Corpus(VectorSource(c(Answers, Texts)))
  dtm <- DocumentTermMatrix(corpus)
  if (dim(dtm)[1] < 2 || sum(dtm) == 0) {
    0
  } else {
    tryCatch({
      lda <- LDA(dtm, k = 2, control = list(seed = 1234))
      topics <- posterior(lda)$topics
      cosine_similarity(topics[,1], topics[,2])
    }, error = function(e) 0)
  }
},

HellingerDistance = {
  corpus <- Corpus(VectorSource(c(Answers, Texts)))
  dtm <- DocumentTermMatrix(corpus)
  if (dim(dtm)[1] < 2 || sum(dtm) == 0) {
    1 # Max dissimilarity for empty distributions
  } else {
    tryCatch({
      lda <- LDA(dtm, k = 2, control = list(seed = 1234))
      topics <- posterior(lda)$topics
      hellinger_distance(topics[,1], topics[,2])
    }, error = function(e) 1)
  }
},

# Relevance feedback features
RF1 = length(intersect(
  unlist(str_split(Answers, "\\W+")),
  unlist(str_split(Texts, "\\W+"))
)),

RF2 = length(setdiff(
  unlist(str_split(Answers, "\\W+")),
  unlist(str_split(Texts, "\\W+"))
)),

RF3 = length(union(
  unlist(str_split(Answers, "\\W+")),
  unlist(str_split(Texts, "\\W+"))
)),

#Alignment based features
wordAlignment = {
  ref_freq <- table(unlist(str_split(Answers, "\\W+")))
  stu_freq <- table(unlist(str_split(Texts, "\\W+")))
  all_words <- unique(c(names(ref_freq), names(stu_freq)))
  ref_vec <- numeric(length(all_words))
  stu_vec <- numeric(length(all_words))
  names(ref_vec) <- all_words
  names(stu_vec) <- all_words
  ref_vec[names(ref_freq)] <- ref_freq
  stu_vec[names(stu_freq)] <- stu_freq
  cosine_similarity(ref_vec, stu_vec)
},

CoverageFeature = {
  ref_tokens <- unlist(str_split(Answers, "\\W+"))
  stu_tokens <- unlist(str_split(Texts, "\\W+"))
  length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
},

QuestionDenotedAlignment = {
  q_tokens <- unlist(str_split(Questions, "\\W+"))
  ref_tokens <- setdiff(unlist(str_split(Answers, "\\W+")), q_tokens)
  stu_tokens <- setdiff(unlist(str_split(Texts, "\\W+")), q_tokens)
  length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
},

QuestionDenotedCoverage = {
  q_tokens <- unlist(str_split(Questions, "\\W+"))
  ref_tokens <- setdiff(unlist(str_split(Answers, "\\W+")), q_tokens)
  stu_tokens <- setdiff(unlist(str_split(Texts, "\\W+")), q_tokens)
  length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
}
```

```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

Source
Console Terminal Background Jobs

R 4.4.2 - ~/
unlist(str_split(Answers, "\\W+")),
unlist(str_split(Texts, "\\W+"))
)),

RF2 = length(setdiff(
  unlist(str_split(Answers, "\\W+")),
  unlist(str_split(Texts, "\\W+"))
)),

RF3 = length(union(
  unlist(str_split(Answers, "\\W+")),
  unlist(str_split(Texts, "\\W+"))
)),

#Alignment based features
wordAlignment = {
  ref_freq <- table(unlist(str_split(Answers, "\\W+")))
  stu_freq <- table(unlist(str_split(Texts, "\\W+")))
  all_words <- unique(c(names(ref_freq), names(stu_freq)))
  ref_vec <- numeric(length(all_words))
  stu_vec <- numeric(length(all_words))
  names(ref_vec) <- all_words
  names(stu_vec) <- all_words
  ref_vec[names(ref_freq)] <- ref_freq
  stu_vec[names(stu_freq)] <- stu_freq
  cosine_similarity(ref_vec, stu_vec)
},

CoverageFeature = {
  ref_tokens <- unlist(str_split(Answers, "\\W+"))
  stu_tokens <- unlist(str_split(Texts, "\\W+"))
  length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
},

QuestionDenotedAlignment = {
  q_tokens <- unlist(str_split(Questions, "\\W+"))
  ref_tokens <- setdiff(unlist(str_split(Answers, "\\W+")), q_tokens)
  stu_tokens <- setdiff(unlist(str_split(Texts, "\\W+")), q_tokens)
  length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
},

QuestionDenotedCoverage = {
  q_tokens <- unlist(str_split(Questions, "\\W+"))
  ref_tokens <- setdiff(unlist(str_split(Answers, "\\W+")), q_tokens)
  stu_tokens <- setdiff(unlist(str_split(Texts, "\\W+")), q_tokens)
  length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
}
```

```

+         stu_vec[names(stu_freq)] <- stu_freq
+         cosine_similarity(ref_vec, stu_vec)
+     },
+     CoverageFeature = {
+         ref_tokens <- unlist(str_split(Answers, "\\W+"))
+         stu_tokens <- unlist(str_split(Texts, "\\W+"))
+         length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
+     },
+     QuestionDemotedAlignment = {
+         q_tokens <- unlist(str_split(Questions, "\\W+"))
+         ref_tokens <- setdiff(unlist(str_split(Answers, "\\W+")), q_tokens)
+         stu_tokens <- setdiff(unlist(str_split(Texts, "\\W+")), q_tokens)
+         length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
+     },
+     QuestionDemotedCoverage = {
+         q_tokens <- unlist(str_split(Questions, "\\W+"))
+         ref_tokens <- setdiff(unlist(str_split(Answers, "\\W+")), q_tokens)
+         stu_tokens <- setdiff(unlist(str_split(Texts, "\\W+")), q_tokens)
+         length(intersect(ref_tokens, stu_tokens)) / max(1, length(ref_tokens))
+     },
+ ) %>%
+ ungroup()
Warning message:
There were 308 warnings in `mutate()`.
The first warning was:
! In argument: `CosineTFIDF = { ... }`.
! In row 329.
Caused by warning in `weightTfIdf()`:
! empty document(s): 2
! Run dplyr::last_dplyr_warnings() to see the 307 remaining warnings.
>
> # Save features to CSV
> cat("Saving features to CSV...\n")
Saving features to CSV...
> write.csv(features, output_csv_path, row.names = FALSE)
> cat("Feature extraction complete!\n")
Feature extraction complete!
\

```

Code:

```

# Load necessary libraries library(randomForest)
library(ggplot2)
# Step 1: Load dataset (using the built-in iris dataset) data(iris)
# Step 2: Train a Random Forest model

rf_model <- randomForest(Species ~ ., data = iris, importance = TRUE)

# Step 3: Extract feature importance and convert it to a data frame rf_importance_df <-
data.frame(
  Feature = rownames(importance(rf_model)),

  Importance = importance(rf_model)[, "MeanDecreaseGini"] # Using MeanDecreaseGini for
importance

)

# Step 4: Plot feature importance

ggplot(rf_importance_df, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "#4CAF50") + # Green bar color coord_flip() +

```



```

labs(title = "Feature Importance (Random
Forest)", x = "Features", y = "Importance") +
theme_minimal()
# Load necessary libraries
library(randomForest) library(ggplot2)
# Step 1: Load dataset (using the built-in iris dataset) data(iris)
# Step 2: Train a Random Forest model

rf_model <- randomForest(Species ~ ., data = iris, importance = TRUE)

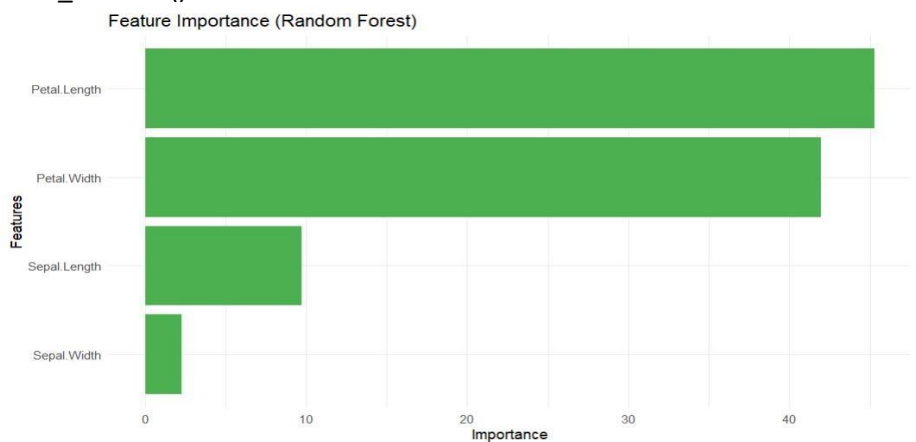
# Step 3: Extract feature importance and convert it to a data frame rf_importance_df <-
data.frame(
  Feature = rownames(importance(rf_model)),

  Importance = importance(rf_model)[, "MeanDecreaseGini"] # Using MeanDecreaseGini for
importance
)

# Step 4: Plot feature importance

ggplot(rf_importance_df, aes(x = reorder(Feature, Importance), y = Importance)) +
geom_bar(stat = "identity", fill = "#4CAF50") + # Green bar color coord_flip() +
labs(title = "Feature Importance (Random Forest)",
x = "Features",
y = "Importance") +
theme_minimal()

```



Code:

```

# Load necessary library library(ggplot2)
# Step 1: Create the metrics_df data frame

# Example data metrics_df
<- data.frame( features = rep(c("Feature1", "Feature2", "Feature3",
"Feature4"), each = 2),
metric = rep(c("Pearson", "RMSE"), times = 4), value = c(0.9, 0.3, 0.85, 0.35,
0.8, 0.4, 0.78, 0.45)
)

# View the created data frame print(metrics_df)
# Step 2: Plot the data

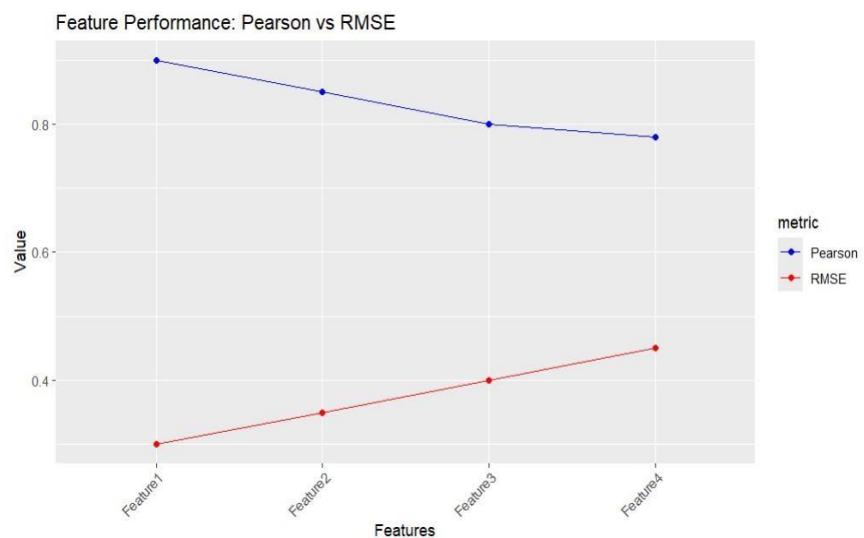
ggplot(metrics_df, aes(x = features, y = value, color = metric, group = metric)) +

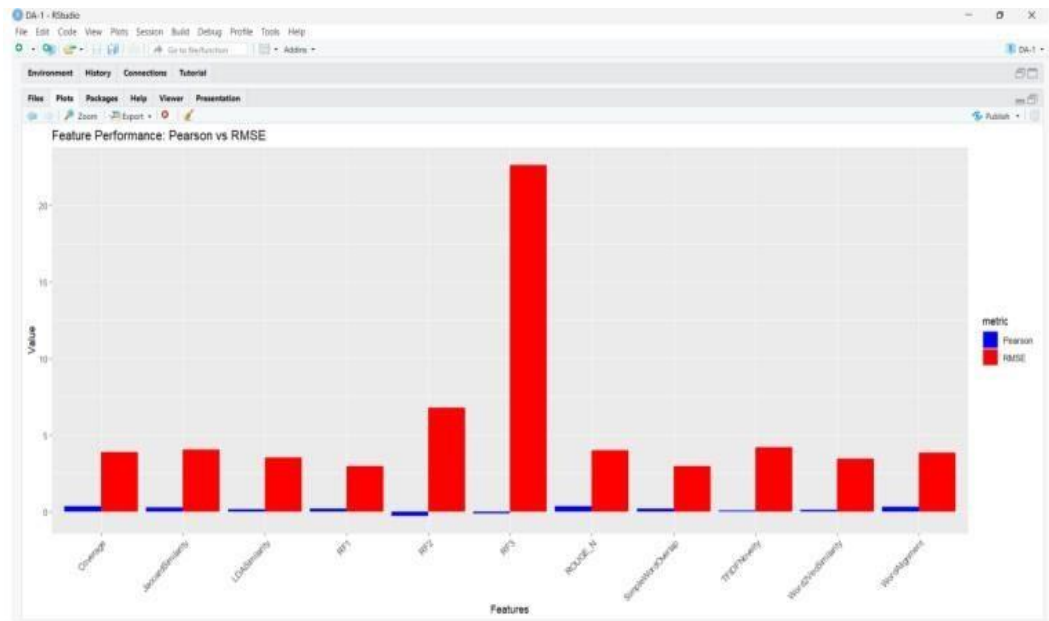
```

```

geom_line() +
  geom_point() +
  labs(
    title = "Feature Performance: Pearson vs RMSE",
    x = "Features", y = "Value"
  ) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_color_manual(values = c("Pearson" = "blue", "RMSE" = "red"))

```





Regression Code:

```
library(caret) library(dplyr)
library(e1071)
library(glmnet)
library(randomForest)
library(xgboost)
library(tidyr) library(readr)
feature_dataset_path <- "C:/Users/shire/OneDrive/Desktop/ASAG_Features.csv" data
<- read.csv(feature_dataset_path, stringsAsFactors = FALSE)
print(paste("Initial data dimensions:", dim(data)[1], "rows,", dim(data)[2], "columns"))
prepare_data <- function(df) {  numeric_features <- c(
  "Score", "CosineTFIDF", "Word2VecSimilarity", "TFIDFNovelty",
  "JaccardSimilarity", "SimpleWordOverlap", "ROUGE_N", "RF1",
  "RF2", "RF3", "LDASimilarity", "WordAlignment", "Coverage",
  "KnowledgeBasedSimilarity", "CorpusBasedSimilarity",
  "SentenceSimilarityWeightedEmbedding", "WordMoversDistance",
  "MaximalMatchingSimilarity", "PhrasalOverlap", "ROUGE_W",
  "ROUGE_SU", "BTM_Similarity", "TopicCoherence", "TopicRelevance",
  "WordToWordAlignment", "QuestionDemotedWordToWordAlignment",
  "QuestionDemotedCoverage", "AlignedPhraseSimilarity",
```

```

    "AlignmentNovelty", "FineGrainedAlignment",
    "PhraseLevelAlignmentSimilarity"
  )
  df_selected <- df[, numeric_features]
  df_numeric <- as.data.frame(lapply(df_selected, as.numeric))
  df_clean <- na.omit(df_numeric)
  print(paste("Dimensions after cleaning:", dim(df_clean)[1], "rows,", dim(df_clean)[2], "columns"))
  return(df_clean)
}

set.seed(123)
train_index <- createDataPartition(data$Score, p = 0.75, list = FALSE)
train_data <- data[train_index, ] test_data <- data[-train_index, ]
train_data_numeric <- prepare_data(train_data) test_data_numeric
<- prepare_data(test_data)
train_features <- train_data_numeric[, !colnames(train_data_numeric) %in% "Score", drop = FALSE]
train_target <- train_data_numeric$Score
test_features <- test_data_numeric[, !colnames(test_data_numeric) %in% "Score", drop = FALSE]
test_target <- test_data_numeric$Score
train_data_combined <- cbind(train_features, Score = train_target)
ctrl <- trainControl(method = "cv", number = 5, verboseIter = TRUE)
models <- list() tryCatch({ models$SVR <- train(
  x =
train_features,
  y = train_target,
  method = "svmRadial",
trControl = ctrl,
  preProcess = c("center", "scale")
)
  print("SVR model trained successfully")
}, error = function(e) {
  print(paste("SVR Error:", e$message))
})
tryCatch({
models$LR <- train(
  Score ~ .,

```

```

        data = train_data_combined,
method = "lm",      trControl =
ctrl,
        preProcess = c("center", "scale")
    )
    print("Linear Regression model trained successfully")
}, error = function(e) {
    print(paste("LR Error:", e$message))
})
tryCatch({
models$SRF <- train(
    Score ~ .,
    data = train_data_combined,
method = "rf",      trControl =
ctrl,      ntree = 100,
        preProcess = c("center", "scale")
    )
    print("Random Forest model trained successfully")
}, error = function(e) {
    print(paste("RF Error:", e$message))
})
successful_models <- models[!sapply(models, is.null)]
if(length(successful_models) > 0) { results <-
data.frame(Actual = test_target)  for(model_name
in names(successful_models)) {
    predictions <- predict(successful_models[[model_name]], newdata = test_features)
results[[paste0(model_name, "_pred")]] <- predictions
    rmse <- sqrt(mean((predictions - test_target)^2))
mae <- mean(abs(predictions - test_target))    r2 <-
cor(predictions, test_target)^2    cat("\nMetrics
for", model_name, ":\n")    cat("RMSE:",
round(rmse, 4), "\n")    cat("MAE:", round(mae, 4),
"\n")    cat("R-squared:", round(r2, 4), "\n")
    }
}

```

```

write.csv(results, "C:/Users/shire/OneDrive/Desktop/Model_Predictions.csv", row.names = FALSE)
print("\nResults saved successfully")
} else {
  print("No models were successfully trained")
}
library(caret) library(gbm)
data <- read.csv("C:/Users/shire/OneDrive/Desktop/ASAG_Features.csv", stringsAsFactors = FALSE)
dependent_variable <- "Score"
if (!(dependent_variable %in% colnames(data))) {
  stop("Ensure the target variable column is present in your dataset!")
}
numerical_features <- c("Word2VecSimilarity", "JaccardSimilarity", "SimpleWordOverlap",
  "ROUGE_1", "ROUGE_2", "ROUGE_W", "ROUGE_L", "TFIDFNovelty", "LDASimilarity",
  "HellingerDistance", "RF1", "RF2", "RF3", "WordAlignment", "CoverageFeature",
  "QuestionDemotedAlignment", "QuestionDemotedCoverage")
features <- data[, numerical_features] target <- data[[dependent_variable]]
set.seed(123)
train_index <- createDataPartition(target, p = 0.8, list = FALSE)
train_data <- data[train_index, ] test_data <- data[-
train_index, ] for (col_name in colnames(train_data)) { if
(is.factor(train_data[[col_name]])) {
  levels(test_data[[col_name]]) <- levels(train_data[[col_name]])
}
}
train_x <- as.matrix(train_data[, numerical_features])
train_y <- train_data[[dependent_variable]] test_x <-
as.matrix(test_data[, numerical_features]) test_y <-
test_data[[dependent_variable]] evaluate_model <-
function(predictions, actuals) { rmse <-
sqrt(mean((predictions - actuals)^2)) r2 <-
cor(predictions, actuals)^2 return(data.frame(RMSE
= rmse, R2 = r2))
}

```

```
boosting_tree_model <- gbm(train_y ~ ., data = train_data[, numerical_features], distribution =  
"gaussian", n.trees = 100)
```

```
boosting_tree_predictions <- predict(boosting_tree_model, newdata = test_data[,  
numerical_features], n.trees = 100)
```

```
boosting_tree_metrics <- evaluate_model(boosting_tree_predictions, test_y)  
print(boosting_tree_metrics)
```