NAME: Jeffrick Aldho J

REG NO: 22BCT0353

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

Aim:

To implement an API for the available research paper models to give the answers for the short answers.
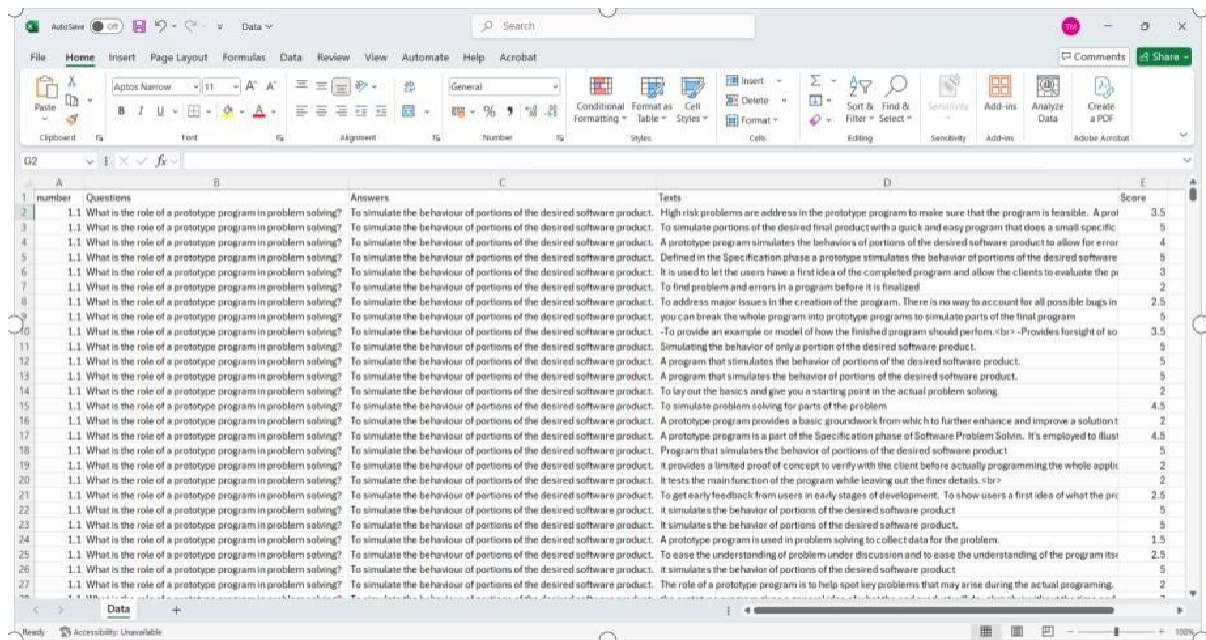
Research Paper:

Title: *Feature Engineering and Ensemble-Based Approach for Improving Automatic Short-Answer Grading Performance*
Authors: Archana Sahu and Plaban Kumar Bhowmick.
Conference/Journal: Educational Data Mining Conference (2018)

Datasets:

1. UNT Dataset
2. SciEntsBank Dataset
3. Beetle Dataset

Statistical Function for Keyword Extraction

$$\text{Relevance}(W) = \log\left(1 + \text{FreqRatio}(W)\right)^{\alpha} \times [1 + \text{InverseDistance}(W)]^{\beta} \times \text{Specificity}(W)^{\gamma}$$

Where:

- Relevance(W) is the final importance score for word W in the lexical extraction process
- FreqRatio(W) is the ratio of word W's frequency in relevant contexts to its frequency in irrelevant contexts plus a smoothing constant δ: frequency in relevant / (frequency in irrelevant + δ)
- InverseDistance(W) is the reciprocal of the average distance to other key terms plus 1: 1/(average distance to other key terms + 1)
- Specificity(W) is a measure of word uniqueness calculated as log(total corpus words / document frequency of W)
- α, β, and γ are tunable exponential parameters that control the influence of each component (typical values: $\alpha=0.5$, $\beta=0.7$, $\gamma=0.4$)

Statistical Function for Keyword Mutation

$$\text{ExpandMetric}(K_1, K_2) = [\text{SymbioticOverlap}(K_1, K_2) \times \text{LogisticDecay}(|K_1|, |K_2|)] \times [1 + \log(1 + \text{SemanticDensity}(K_1 \cap K_2))]$$

Where: • ExpandMetric(K_1, K_2) is the final expansion benefit score between keyword sets K_1 and K_2

- SymbioticOverlap(K_1, K_2) is the quadratic overlap measure calculated as $|K_1 \cap K_2|^2 / (|K_1| \times |K_2|)$
- LogisticDecay($|K_1|$, $|K_2|$) is a balanced size similarity function calculated as $2/(1 + \exp(|\text{abs}(|K_1| - |K_2|)|/\lambda))$, where λ is a scaling parameter

- SemanticDensity($K_1 \cap K_2$) is the sum of co-occurrence frequencies for all word pairs in the intersection
- $|K_1|$ and $|K_2|$ are the cardinalities (sizes) of the keyword sets
- log is the natural logarithm function
- exp is the exponential function

Statistical Functions for Score calculation

$$\text{AdjustedMetric}(R) = M \times \left[1 - e^{-R/\tau}\right] \times \left[1 + \beta \times \sin\left(\frac{\pi \times R}{2M}\right)\right]$$

Where:

- AdjustedMetric(R) is the final calibrated value after quantization to the nearest 0.5

- R is the original raw measurement

- M is the maximum threshold value

- τ is a scaling coefficient governing the saturation rate

- β is a modulation parameter controlling oscillation intensity

- e is the base of the natural logarithm

- sin is the sine function

<u>**CODE FOR API:**</u> try to build

upon with this code:

```
if (!require("shiny")) install.packages("shiny")

if (!require("tm")) install.packages("tm") if

(!require("dplyr")) install.packages("dplyr") if

(!require("readr")) install.packages("readr") if

(!require("DT")) install.packages("DT")


library(shiny) library(tm)

library(dplyr)

library(readr)  library(DT)
```

```r
# Basic functions extract_keywords <- function(text) {   text
<- tolower(text)   words <- unlist(strsplit(gsub("[[:punct:]]",
" ", text), "\\s+"))   words <- words[words != ""]
  stopwords <- c("the", "and", "a", "an", "in", "on", "at", "to", "for", "of", "with",
"is", "are", "was", "were", "be", "been", "being", "have", "has", "had",
           "do", "does", "did", "can", "could", "will", "would", "should", "may",
           "might", "must", "shall", "this", "that", "these", "those", "it", "its",
           "they", "them", "their", "we", "us", "our", "i", "me", "my", "you", "your")
words <- words[!words %in% stopwords]   words <- words[nchar(words) > 1]
words <- words[!grepl("^\\d+$", words)]   return(unique(words))
}


calculate_score <- function(answer_keywords, student_keywords)
{ jaccard_sim <- length(intersect(answer_keywords, student_keywords)) /
length(union(answer_keywords, student_keywords))   raw_score <-
jaccard_sim * 5   wpcs_score <- min(raw_score * 1.05, 5)   wpcs_score
<- round(wpcs_score * 2) / 2
return(list(    raw_score =
raw_score,    wpcs_score =
wpcs_score,    similarity =
jaccard_sim
  ))
}


SCM <- function(corpus, answer_keywords, student_keywords, threshold = 0.3) {   if
(length(corpus) == 0 || length(answer_keywords) == 0 || length(student_keywords) == 0)
{ return(list(mutation_candidates = list(), similarity_score = 0))
  }
  corpus <- lapply(corpus, function(x) if(length(x) == 0) c("") else x)
candidates <- setdiff(student_keywords, answer_keywords)    if
```

```r
(length(candidates) == 0) {     return(list(mutation_candidates =
list(), similarity_score = 0))
  }
  candidate_freq <- sapply(candidates, function(word)
{ sum(sapply(corpus, function(doc) word %in% doc))
  })
  candidate_rel_freq <- candidate_freq / length(corpus)
mutation_candidates <- list()    for (i in
1:length(candidates)) {     word <- candidates[i]
freq <- candidate_rel_freq[i]     if (freq >= threshold)
{     mutation_candidates[[word]] <- list(         word
= word,        score = freq,        uniqueness = 1 - freq
    )
  }
 }
  if (length(mutation_candidates) > 0)
{ sorted_candidates <-
mutation_candidates[order( sapply(mutation_candidat
es, function(x) x$score), decreasing = TRUE
  )]
  } else
{ sorted_candidates <-
list()
 }
 jaccard_sim <- length(intersect(answer_keywords, student_keywords)) /
length(union(answer_keywords, student_keywords))
  return(list(     mutation_candidates =
sorted_candidates,     similarity_score =
jaccard_sim
 ))
}
```

```r
update_keywords <- function(question_data) {   answer_keywords <-
unlist(strsplit(question_data$Answer_Keywords[1], ", "))   all_text_keywords
<- lapply(question_data$Text_Keywords, function(x) {     if (is.na(x) || x == "")
return(character(0))     unlist(strsplit(x, ", "))
 })
 all_student_keywords <- unique(unlist(all_text_keywords))    threshold <- 0.65   scm_result
<- SCM(all_text_keywords, answer_keywords, all_student_keywords, threshold)
mutation_candidates <- scm_result$mutation_candidates   new_keywords <-
names(mutation_candidates)
 return(list(    new_keywords = paste(new_keywords,
collapse = ", "),    similarity_score =
scm_result$similarity_score
 ))
}


process_batch <- function(data) {   if ("Score" %in% colnames(data) &&
!"WPCS_Score" %in% colnames(data)) {     data <- data %>%
                                  mutate( Score =
    as.numeric(Score),
    WPCS_Score = pmin(Score * 1.05, 5),
    WPCS_Score = round(WPCS_Score * 2) / 2
   )
 }
 if (all(c("Questions", "Answer_Keywords", "Text_Keywords") %in% colnames(data)))
{ result <- data %>%   group_by(Questions) %>%      group_modify(~{
mutation_result <- update_keywords(.x)
    .x$New_Keywords <- mutation_result$new_keywords
    .x$Similarity_Score <- mutation_result$similarity_score
    .x$Combined_Keywords <- ifelse(.x$New_Keywords != "",
paste(.x$Answer_Keywords, .x$New_Keywords, sep = ", "),
```

```r
                    .x$Answer_Keywords)

return(.x)      }) %>%     ungroup()

return(result)

  }

return(data)

}


# Ultra simple UI ui <-
fluidPage( titlePanel("Keyword Analysis"),


  # Single Analysis Tab
h3("Single Analysis"),


  textInput("question", "Question"),  textAreaInput("model_answer", "Model
Answer"),  textInput("manual_keywords", "Model Answer Keywords (comma-
separated)"),  textAreaInput("student_answer", "Student Answer"),
actionButton("analyze_btn", "Analyze"),


hr(),


  h4("Results:"),
verbatimTextOutput("score_output"),
verbatimTextOutput("keywords_output"),  hr(),


  # Batch Processing Tab
h3("Batch Processing"),


  fileInput("file_upload", "Upload CSV File"),
checkboxInput("header", "File has header", TRUE),
```

```r
actionButton("process_btn", "Process"),

downloadButton("download_results", "Download"),


hr(),


 DTOutput("results_table"),

verbatimTextOutput("batch_stats")
)
# Server logic server <- function(input,
output, session) {   results <-
reactiveValues(    model_keywords =
NULL,    student_keywords = NULL,
score = NULL,   batch_data = NULL,
processed_data = NULL
 )


 observeEvent(input$analyze_btn,
{ req(input$model_answer,
input$student_answer)


  if (input$manual_keywords != "") {      model_kw <-
unlist(strsplit(input$manual_keywords, ", "))      model_kw <-
trimws(model_kw)
  } else {
   model_kw <- extract_keywords(input$model_answer)
  }


  student_kw <- extract_keywords(input$student_answer)
score_result <- calculate_score(model_kw, student_kw)
```

```r
    results$model_keywords <- model_kw
results$student_keywords <- student_kw    results$score
<- score_result


  output$score_output <- renderPrint({       cat("WPCS Score: ", results$score$wpcs_score,
"/5.0\n", sep = "")     cat("Similarity: ", round(results$score$similarity * 100, 2), "%\n",
sep = "")       cat("Matching: ", length(intersect(results$model_keywords,
results$student_keywords)),
      " out of ", length(union(results$model_keywords, results$student_keywords)), "\n", sep="")
  })


  output$keywords_output <- renderPrint({       cat("Model
Answer Keywords:\n")
cat(paste(results$model_keywords, collapse = ", "), "\n\n")
cat("Student Answer Keywords:\n")
cat(paste(results$student_keywords, collapse = ", "))
  })
 })


 observeEvent(input$file_upload,
{ req(input$file_upload)


  tryCatch({       batch_data <- read_csv(input$file_upload$datapath, col_names =
input$header)      results$batch_data <- batch_data


   output$results_table <-
renderDT({ datatable(results$batch_data, options =
list(pageLength = 5))
   })
  }, error = function(e) {       showNotification("Error
reading file", type = "error")
```

```r
    })
  })

  observeEvent(input$process_btn,
{ req(results$batch_data)

    processed_data <- process_batch(results$batch_data)
results$processed_data <- processed_data

    output$results_table <-
renderDT({ datatable(results$processed_data, options =
list(pageLength = 5))
    })

    output$batch_stats <- renderPrint({      if
("WPCS_Score" %in% colnames(processed_data))
{ cat("Records: ", nrow(processed_data), "\n")


      if ("Score" %in% colnames(processed_data)) {        cat("Average Score: ",
round(mean(processed_data$Score, na.rm = TRUE), 2), "\n")        cat("Average WPCS: ",
round(mean(processed_data$WPCS_Score, na.rm = TRUE), 2), "\n")

      }
      if ("Similarity_Score" %in% colnames(processed_data)) {
        cat("Average Similarity: ", round(mean(processed_data$Similarity_Score, na.rm = TRUE) *
100, 2), "%\n")

      }
    } else
{ cat("Data
loaded")
}
  })
  })
```

```
 output$download_results <-

downloadHandler( filename = function() {

    "results.csv"

  },

  content = function(file)

{ write_csv(results$processed_data, file)

  }

 )

}
```
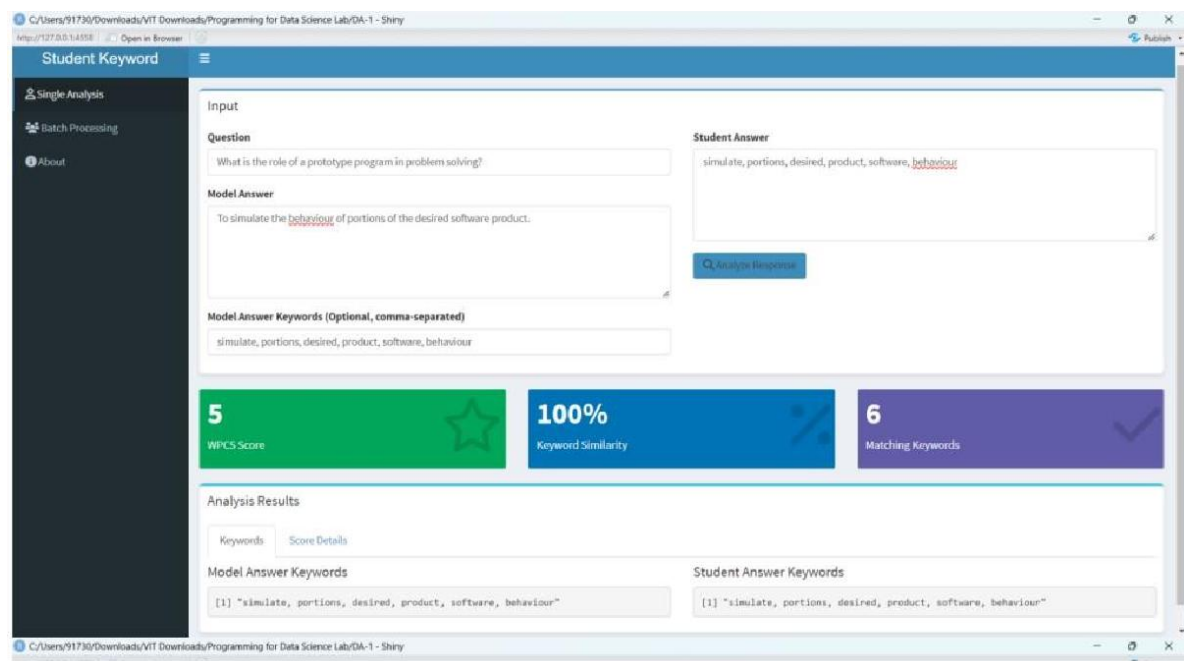
shinyApp(ui = ui, server = server)**RESULT:**

Student Keyword ≡

- 👤 Single Analysis
- 👥 Batch Processing
- ℹ️ About

File Upload

Upload a CSV file with questions, model answers, and student responses for batch processing.

Choose CSV File

| Browse... | Data.csv |

Upload complete

☑ File has header

[⚙ Process File] [⬇ Download Results]

Processing Results

Show 5 ▾ entries   Search: [   ]

number		Questions	Answers	Texts	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 product.	High risk problems are address in the prototype program to make sure that the program is feasible. A prototype may also be used to show a company that the software can be possibly programmed. 	3.5
2	1.1	What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software product.	To simulate portions of the desired final product with a quick and easy program that does a small specific job. It is a way to help see what the problem is and how you may solve it in the final project.	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 product.	A prototype program simulates the behaviors of portions of the desired software product to allow for error checking.	4
4	1.1	What is the role of a prototype program in problem solving?	To simulate the behaviour of portions of the desired software product.	Defined in the Specification phase a prototype stimulates the behavior of portions of the desired software product. Meaning, the role of a prototype is a temporary solution until the program itself is refined to be used extensively in problem solving.	5

Processing Results

Show 5 ▾ entries   Search: [   ]

number		Questions	Answers	Texts	Score	WPCS_Score
1	1.1	What is the role of a prototype program in problem solving?	To simulate the behaviour of the desired software product.	High risk problems are address in the prototype program to make sure that the program is feasible. A prototype may also be used to show a company that the software can be possibly programmed. 	3.5	3.5
2	1.1	What is the role of a prototype program in problem solving?	To simulate the behaviour of the desired software product.	To simulate portions of the desired final product with a quick and easy program that does a small specific job. It is a way to help see what the problem is and how you may solve it in the final project.	5	5
3	1.1	What is the role of a prototype program in problem solving?	To simulate the behaviour of the desired software product.	A prototype program simulates the behaviors of portions of the desired software product to allow for error checking.	4	4
4	1.1	What is the role of a prototype program in problem solving?	To simulate the behaviour of the desired software product.	Defined in the Specification phase a prototype stimulates the behavior of portions of the desired software product. Meaning, the role of a prototype is a temporary solution until the program itself is refined to be used extensively in problem solving.	5	5
5	1.1	What is the role of a prototype program in problem solving?	To simulate the behaviour of the desired software product.	It is used to let the users have a first idea of the completed program and allow the clients to evaluate the program. This can generate much feedback including software specifications and project estimations of the total project.	3	3

Showing 1 to 5 of 2,442 entries   Previous 1 2 3 4 5 ... 489 Next

Statistics

```
Processing Complete

Total Records:  2442
Average Original Score:  4.18
Average WPCS Score:  4.18
```