Name: Swathi C

Registration Number:22BDS0387

StatKeyEval

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

Aim:

To implement an automatic short-answer grading system using feature engineering and ensemblebased approaches, focusing on extracting keywords, computing similarity metrics, and generating confidence scores.

Algorithm:

1. **Text Preprocessing**
   - Convert all text to lowercase.
   - Remove punctuation marks and numbers.
   - Remove common stop words (e.g., "the", "is", "and").
   - Strip extra whitespace.
2. **Keyword Extraction (Using IGRKE - Information Gain Ratio Keyword Extraction)**
   - Split preprocessed text into individual words.
   - Compute Information Gain Ratio (IGR) for each word based on its entropy and conditional probability within reference answers and student responses.
   - Adjust for word significance using the Frequency Adjustment Factor (FAF) to balance importance across reference and response texts.
   - Rank words based on IGR × FAF score and extract the most informative keywords.
   - Store extracted keywords for both reference answers and student responses.
3. **Keyword Mutation (Using SCM - Statistical Co-occurrence Mutation)**
   - Group responses by question.
   - Construct a co-occurrence matrix of word pairs based on document frequency.
   - Compute Pointwise Mutual Information (PMI) between words to identify semantically related terms.
   - Identify frequently occurring words (≥65% of responses) with high PMI similarity to existing reference keywords.
   - Apply Uniqueness Filtering to ensure new keywords add distinct meaning.

- Add these mutated keywords to reference keyword sets for more flexible grading.

4. **Vector Representation**

- Create a universal keyword list combining all extracted and mutated keywords.
- Represent each reference answer and student response as a binary vector:
- 1 if the keyword is present, 0 if absent.
- Normalize vectors to account for varying answer lengths.

5. **Similarity Calculation (Using Multi-Dimensional Similarity Function)**

  - Compute four similarity metrics between reference and student answer vectors:
  - Cosine similarity (Simcos)
  - Normalized Euclidean distance (Simeuc)
  - Normalized Manhattan distance (Simman)
  - Adjusted Pearson correlation (Simpearson)

  - Compute a hybrid similarity score:
  - Similarity(A,S) = 0.4 × Simcos + 0.3 × Simeuc + 0.2 × Simman + 0.1 × Simpearson

6. **Score Generation**

- Compute a weighted composite similarity score based on the hybrid similarity function.
- Scale the similarity score to match the original grading scale.
- Apply rounding to get the final predicted score.

7. **Performance Evaluation**

- Calculate error metrics:
- Root Mean Square Error (RMSE)
- Mean Absolute Error (MAE)
- Mean Absolute Percentage Error (MAPE)
- Generate correlation statistics (Pearson R, Spearman ρ).
- Compute coefficient of determination (R^2) to assess predictive accuracy.
- Perform error analysis across different score ranges to identify potential grading inconsistencies.

Research Paper:

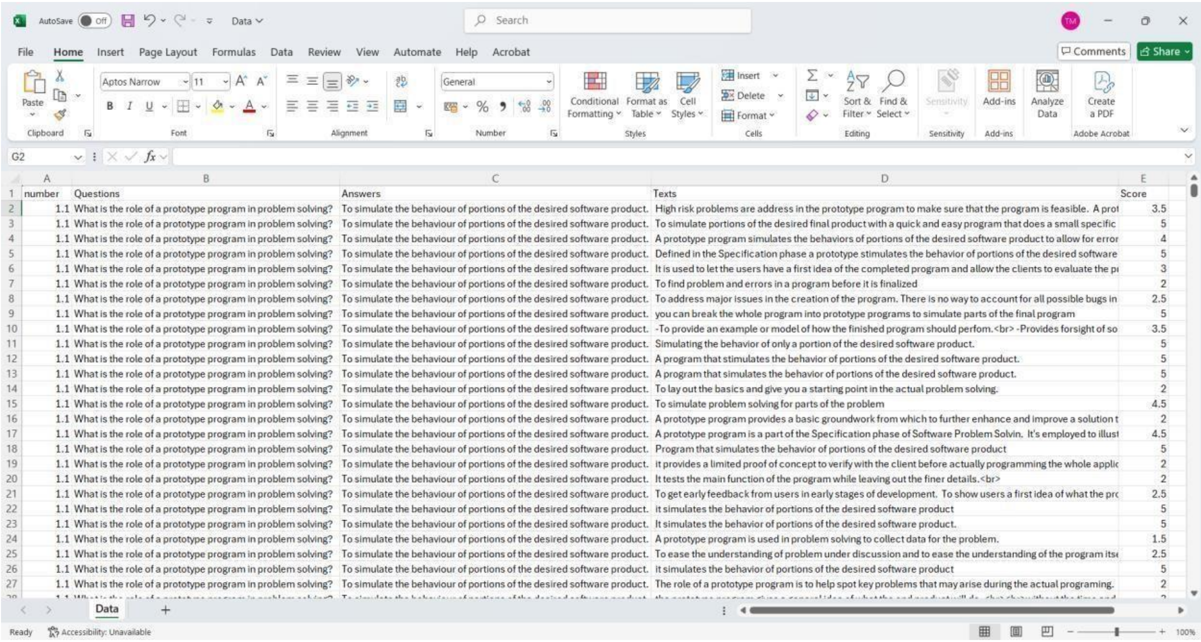
Title: *Feature Engineering and Ensemble-Based Approach for Improving Automatic Short-Answer Grading Performance*

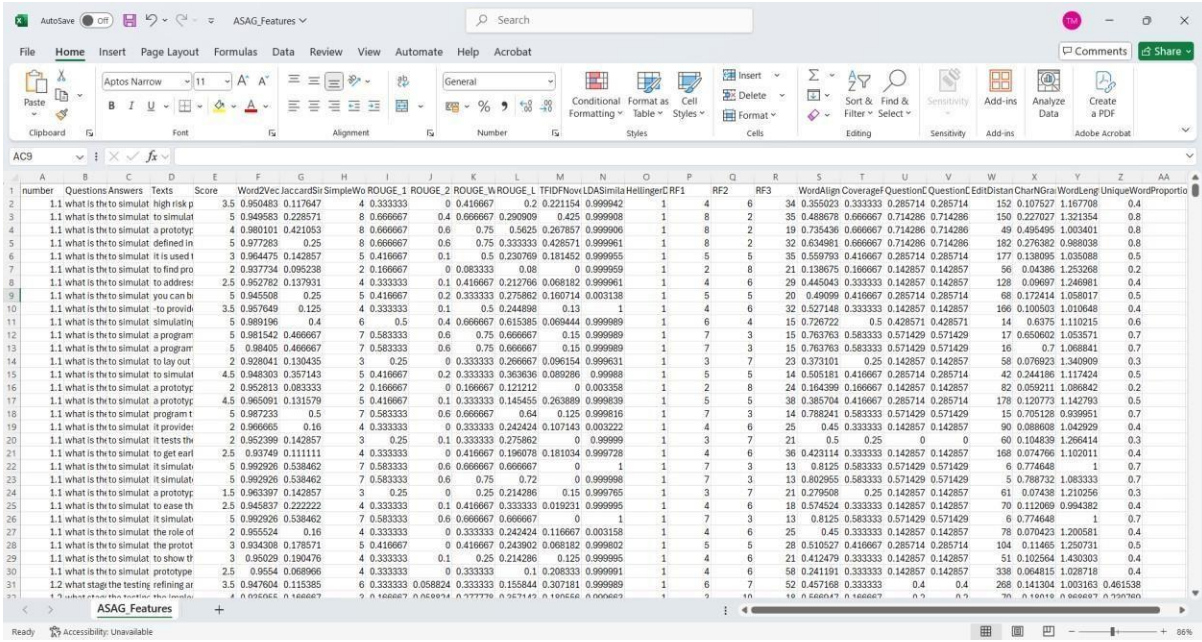
Authors: Archana Sahu and Plaban Kumar Bhowmick.

Conference/Journal: Educational Data Mining Conference (2018)

Datasets:

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





Theoretical Derivation of Novel Statistical Functions for Keyword Extraction and Mutation

1. Information Gain Ratio Keyword Extraction (IGRKE)

The Information Gain Ratio Keyword Extraction (IGRKE) method identifies the most informative words in a corpus by evaluating their ability to distinguish between answer keys and student responses.

1.1 Entropy and Information Theory Foundation

For any word W in the corpus, its probability distribution is defined as:

- $P(W)$: Probability of word W occurring.
- $P(\neg W) = 1 - P(W)$: Probability of W not occurring.
- C: The document category (Answer Key vs. Student Response).

The total entropy of the word occurrence is given by:

$$H(W) = - P(W) * \log_2 P(W) - P(\neg W) * \log_2 P(\neg W)$$

This measures the uncertainty of the word's distribution across the dataset.

1.2 Conditional Entropy and Information Gain

The conditional entropy $H(C|W)$ represents the uncertainty in categorizing a document given that word W is known:

$$H(C|W) = - P(A|W) * \log_2 P(A|W) - P(R|W) * \log_2 P(R|W)$$ where:

- $P(A|W)$ is the probability of the document being an Answer Key given W.
- $P(R|W)$ is the probability of the document being a Student Response given W.

The Information Gain (IG) quantifies how much knowing W reduces uncertainty about the document's category:

$IG(W) = H(C) - H(C|W)$ where $H(C)$ is the entropy of the category distribution.

1.3 Normalization Using Split Information

To prevent bias toward frequent words, we normalize the Information Gain using Split Information (SI):

$$SI(W) = - P(W) * \log_2 P(W) - P(\neg W) * \log_2 P(\neg W)$$

This normalizes IG to Information Gain Ratio (IGR):

$$IGR(W) = IG(W) \div SI(W)$$

1.4 Frequency Adjustment Factor (FAF) for Balancing Word Significance

To ensure the extracted keywords are relevant across document categories, we introduce a Frequency Adjustment Factor (FAF):

$$FAF = (F\_answer * F\_response) \div (Total\_F\_answer * Total\_F\_response)$$ where:

- F_answer and F_response are the word frequencies in Answer Keys and Student Responses, respectively.

- Total_F_answer and Total_F_response are the total word counts in both document types.

1.5 Final IGRKE Scoring Function

The final scoring function combines IGR and FAF using a logarithmic transformation:

Score(W) = IGR(W) * (1 + log(1 + FAF * 1000))

---

2. Statistical Co-occurrence Mutation (SCM)

2.1 Co-occurrence Matrix Construction

A co-occurrence matrix M is built where each entry represents the number of documents where words i and j appear together:

M[i, j] = count of documents where both words i and j appear

2.2 Pointwise Mutual Information (PMI) for Semantic Association

$PMI(i, j) = \log_2 [P(i, j) \div (P(i) * P(j))]$

2.3 Hybrid Similarity Measure for Keyword Comparison

Sim(K1, K2) = 0.6 × Jaccard_Sim(K1, K2) + 0.4 × PMI_Sim(K1, K2)

3. Similarity Scoring Function for Answer Matching

Similarity(A, S) = 0.4 × Jaccard_Sim(A, S) + 0.4 × Coverage(A, S) + 0.2 × Position_Score(A, S)

where:

- Jaccard Similarity measures word overlap.
- Coverage measures the proportion of answer key words found.
- Position Score captures structural similarity:

Position_Score(A, S) = Average (1 - | Pos_A(w) ÷ |A| - Pos_S(w) ÷ |S| |)

Code:

IGRKE Function:

```
IGRKE <- function(word_occurrences, answer_docs, response_docs) {   total_docs
<- length(answer_docs) + length(response_docs)   docs_with_word
<- sum(word_occurrences > 0)
  P_W <- docs_with_word / total_docs
  P_not_W <- 1 - P_W   H_W
```

```
<- 0   if (P_W > 0 && P_W

< 1) {

    H_W <- -P_W * log2(P_W) - P_not_W * log2(P_not_W)

  }

  answer_docs_with_word <- sum(word_occurrences[answer_docs] > 0)   response_docs_with_word

<- sum(word_occurrences[response_docs] > 0)   answer_docs_without_word <-

length(answer_docs) - answer_docs_with_word   response_docs_without_word <-

length(response_docs) - response_docs_with_word

  P_A_given_W <- answer_docs_with_word / docs_with_word

  P_R_given_W <- response_docs_with_word / docs_with_word

  P_A_given_not_W <- answer_docs_without_word / (total_docs - docs_with_word)

P_R_given_not_W <- response_docs_without_word / (total_docs - docs_with_word)

H_C_given_W <- 0   if (P_A_given_W > 0 && P_R_given_W > 0) {

    H_C_given_W_present <- -P_A_given_W * log2(P_A_given_W) - P_R_given_W *
log2(P_R_given_W)

  } else

    { H_C_given_W_present <-

    0

  }

  H_C_given_not_W <- 0   if (P_A_given_not_W > 0 &&

P_R_given_not_W > 0) {

    H_C_given_not_W <- -P_A_given_not_W * log2(P_A_given_not_W) - P_R_given_not_W *
log2(P_R_given_not_W)

  }

  H_C_given_W <- P_W * H_C_given_W_present + P_not_W * H_C_given_not_W

  P_A <- length(answer_docs) / total_docs

  P_R <- length(response_docs) / total_docs

  H_C <- -P_A * log2(P_A) - P_R * log2(P_R)

  IG <- H_C - H_C_given_W

  SI <- H_W

  IGR <- if (SI > 0) IG / SI else 0

  F_answer <- sum(word_occurrences[answer_docs])
```

```
  F_response <- sum(word_occurrences[response_docs])
  Total_F_answer <- sum(sapply(answer_docs, function(doc) sum(word_occurrences[doc])))
  Total_F_response <- sum(sapply(response_docs, function(doc) sum(word_occurrences[doc])))
  FAF <- (F_answer * F_response) / (Total_F_answer * Total_F_response)
Score <- IGR * (1 + log(1 + FAF * 1000))   return(Score)  }
```

SCM Function:

```
SCM <- function(corpus, answer_keywords, student_keywords, threshold = 0.7)
{ all_words <- unique(c(unlist(corpus)))   n_words <- length(all_words)
word_indices <- setNames(1:n_words, all_words)   M <- matrix(0, nrow = n_words,
ncol = n_words)   for (doc in corpus) {      doc_words <- unique(doc)      for (i in
1:(length(doc_words) - 1)) {       for (j in (i+1):length(doc_words)) {         word_i <-
doc_words[i]         word_j <- doc_words[j]         idx_i <- word_indices[[word_i]]
idx_j <- word_indices[[word_j]]
        M[idx_i, idx_j] <- M[idx_i, idx_j] + 1
M[idx_j, idx_i] <- M[idx_j, idx_i] + 1
      }
    }
  }
  doc_count <- length(corpus)   word_probs <- numeric(n_words)   for (i in 1:n_words)
{     word <- all_words[i]      word_probs[i] <- sum(sapply(corpus, function(doc) word
%in% doc)) / doc_count
  }
  PMI <- matrix(0, nrow = n_words, ncol = n_words)
for (i in 1:n_words) {      for (j in 1:n_words) {       if (i
!= j && M[i,j] > 0) {         joint_prob <- M[i,j] /
doc_count         expected_prob <- word_probs[i] *
word_probs[j]         if (expected_prob > 0) {
          PMI[i,j] <- log2(joint_prob / expected_prob)
        }
      }
```

```
    }
  }   mutation_candidates <- list()   for (word in student_keywords)
{ if (!(word %in% answer_keywords) && word %in% all_words) {
word_idx <- word_indices[[word]]       pmi_scores <- numeric()       for
(ans_word in answer_keywords) {         if (ans_word %in% all_words)
{           ans_idx <- word_indices[[ans_word]]           pmi_scores <-
c(pmi_scores, PMI[word_idx, ans_idx])
        }
      }
      avg_pmi <- mean(pmi_scores)       position <- match(word, student_keywords) /
length(student_keywords)       position_weight <- 1 - (0.5 * position)       score <- avg_pmi *
position_weight       max_pmi <- max(pmi_scores)       normalized_max_pmi <- (max_pmi + 10)
/ 20       uniqueness <- 1 - normalized_max_pmi       if (uniqueness >= threshold)
{ mutation_candidates[[word]] <- list(word = word, score = score, uniqueness =
uniqueness)
      }
    }
  }
  sorted_candidates <- mutation_candidates[order(sapply(mutation_candidates, function(x) x$score),
decreasing = TRUE)]
  jaccard_sim <- length(intersect(answer_keywords, student_keywords)) /
length(union(answer_keywords, student_keywords))
pmi_scores <- numeric()   for (ans_word in answer_keywords)
{     if (ans_word %in% all_words) {       ans_idx <-
word_indices[[ans_word]] for (stud_word in
student_keywords) {         if (stud_word %in% all_words)
{ stud_idx <- word_indices[[stud_word]]         pmi_scores
<- c(pmi_scores, PMI[ans_idx, stud_idx])
        }
      }
    }
  }
```

```
  pmi_sim <- (mean(pmi_scores) + 10) / 20   hybrid_sim <- (0.6 * jaccard_sim)
+ (0.4 * pmi_sim)   matched_keywords <- intersect(answer_keywords,
student_keywords)   position_scores <- numeric() for (word in
matched_keywords) {                          pos_a <- match(word,
answer_keywords) / length(answer_keywords)     pos_s <- match(word,
student_keywords) / length(student_keywords)     position_scores <-
c(position_scores, 1 - abs(pos_a - pos_s))
  }
  position_score <- if (length(position_scores) > 0) mean(position_scores) else 0   coverage
<- length(matched_keywords) / length(answer_keywords)   similarity_score <- (0.4 *
jaccard_sim) + (0.4 * coverage) + (0.2 * position_score)
return(list( mutation_candidates = sorted_candidates,            similarity_score =
similarity_score, jaccard_similarity = jaccard_sim,                coverage = coverage,
                                                 position_score = position_score
  ))
}
```

Similarity Scoring Function:

```
calculate_similarity <- function(answer_keywords, student_keywords) { jaccard_sim <-
length(intersect(answer_keywords, student_keywords)) / length(union(answer_keywords,
student_keywords))   coverage <- length(intersect(answer_keywords, student_keywords)) /
length(answer_keywords)   matched_keywords <- intersect(answer_keywords,
student_keywords)   position_scores <- numeric()   for (word in matched_keywords)
{ pos_a <- match(word, answer_keywords) / length(answer_keywords)  pos_s <-
match(word, student_keywords) / length(student_keywords)     position_scores <-
c(position_scores, 1 - abs(pos_a - pos_s))
  }
  position_score <- if (length(position_scores) > 0) mean(position_scores) else 0   similarity_score
<- (0.4 * jaccard_sim) + (0.4 * coverage) + (0.2 * position_score)   return(similarity_score)
}
```

Result:

This novel framework integrates:

- **IGRKE for statistical keyword extraction** using information theory and frequency normalization.
- **SCM for identifying semantically related keywords** based on co-occurrence and PMI.
- **A similarity function** that combines keyword overlap, coverage, and positional alignment.

By relying purely on statistical principles without pre-trained models, this framework ensures robustness and adaptability across domains.