

Bloom's Taxonomy-based Question Analysis for Personalized Learning

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering with
Artificial Intelligence

By

DINESH SHEELAM (Reg.No - 40731091)

SHIVA RAJ B (Reg.No - 40731092)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Category-1 University by UGC

Accredited with Grade "A++" by NAAC | 12B Status by UGC | Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHISALAI,

CHENNAI - 600119

April - 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

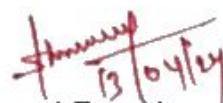
This is to certify that this Project Report is the bonafide work of **Dinesh Sheelam (Reg. No - 40731091)** and **B Shiva Raj (Reg. No - 40731092)** who carried out the Project work entitled **"BLOOM'S TAXONOMY - BASED QUESTION ANALYSIS FOR PERSONALIZED LEARNING"** under my supervision from November 2023 to April 2024.


Internal Guide

Dr. J. JESLIN SHANTHAMALAR M.E., Ph. D.


HEAD OF THE DEPARTMENT
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
DR. L. LAKSHMANAN M.E., Ph. D.
Jeppiaar Nagar, Rajiv Gandhi Salai,
Chennai - 600 119.

Submitted for Viva Voce Examination held on 13 - 04 - 2024


Internal Examiner


External Examiner

DECLARATION

I, **Dinesh Sheelam (Reg. No - 40731091)** , hereby declare that the Project Report entitled **“BLOOM'S TAXONOMY - BASED QUESTION ANALYSIS FOR PERSONALIZED LEARNING”** done by me under the guidance of **Dr. J. Jeslin Shanthamalar, M.E., Ph.D.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering with Specialization in Artificial Intelligence**.

DATE: 12-04-2024

PLACE: Chennai



SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of Sathyabama Institute of Science and Technology** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D, Dean**, School of Computing, **Dr.L.Lakshmanan, M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. J. Jeslin Shanthamalar, M.E., Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase-2 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering with Artificial intelligence** who were helpful in many ways for the completion of the project.

ABSTRACT

In the 1950s, Benjamin Bloom and his associates introduced Bloom's Taxonomy, a foundational framework for categorizing learning objectives and cognitive skills. While question classification typically operates at a fine level, such as sentences and phrases, and text classification focuses on the document level, past research has explored the intersection of question classification and Bloom's Taxonomy to assess learners' cognitive levels in higher education. However, existing feature types from previous studies may excel in datasets with narrowly focused queries, necessitating the development of multiple classifiers for diverse fields or areas.

To address this, a novel feature type termed "taxonomy-based" is proposed to enhance question classification accuracy across datasets spanning various disciplines. Utilizing datasets comprising questions from distinct topics, the study evaluates the effectiveness of taxonomy-based features. Support vector machines (SVMs) are chosen as the classifier for their reputed text classification accuracy. The research reveals that taxonomy-based features significantly improve classifier performance when applied to question sets from different domains.

The highest accuracy achieved using SVM with TF-IDF vectorizer is 74.17% for dataset1 and 95.23% for dataset2. Employing the KNN algorithm with the TF-IDF vectorizer yields 59.17% accuracy for dataset1, with dataset2 reaching a maximum accuracy of 89.11%. Naïve Bayes algorithm, coupled with the TF-IDF vectorizer, achieves a peak accuracy of 73.33% for dataset1 and 90.99% for dataset2. Lastly, using the Random Forest algorithm with the count vectorizer results in a maximum accuracy of 74.17% for dataset1 and 95.21% for dataset2.

TABLE OF CONTENT

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	iv
	LIST OF ABBREVIATIONS	vii
	LIST OF FIGURES	viii
	LIST OF TABLES	ix
1	INTRODUCTION	1
2	LITERATURE SURVEY	7
	2.1 Related work in blooms question classification	8
	2.2 Inferences from Literature Survey	9
	2.3 Open problems in Existing System	10
3	AIM AND SCOPE OF THE PRESENT INVESTIGATION	12
	3.1 Feasibility Studies/Risk Analysis of the Project	12
	3.2 Software Requirements Specification Document	13
4	EXPERIMENTAL OR MATERIALS AND METHODS	14
	4.1 Selected Methodology or process model	15
	4.2 Architecture / Overall Design of Proposed System	16
	4.3 Description of Software for Implementation and Testing plan of the Proposed Model/System	20
	4.4 Transition/ Software to Operations Plan	21
5	IMPLEMENTATION DETAILS	23
	5.1 Development and Deployment Setup	23
	5.2 Techniques Used	24
	5.3 Testing	25

6	RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS	28
7	SUMMARY AND CONCLUSIONS	35
7.1	Conclusion	35
7.2	Future work	36
7.3	Research Issues	37
7.4	Implementation Issues	38
	REFERENCES	39
	APPENDIX	43
	A. SOURCE CODE	43
	B. SCREENSHOTS	50

LIST OF ABBREVIATIONS

SN NO.	KEYWORD	ABBREVIATION
1	Term Frequency and Inverse Document Frequency	TF-IDF
2	Unsupervised Term Weighting Scheme	UTWS
3	Supervised Term Weighting Scheme	STWS
4	Bloom's Taxonomy	BT
5	Natural Language Tool Kit	NLTK
6	Term Frequency and Inverse Count Frequency	TF-ICF
7	K-Nearest Neighbor	KNN
8	Support Vector Machine	SVM

LIST OF FIGURES

FIG. NO.	FIGURE NAME	PAGE NO.
4.1	Bloom's Taxonomy for cognitive learning	15
4.2	System Architecture	16
6.1	Graph of SVM algorithm	27
6.2	Graph of K-Nearest Neighbour algorithm	29
6.3	Graph of Naive Bayes algorithm	30
6.4	Graph of Random Forest algorithm	32

LIST OF TABLES

TABLE NO.	TABLE NAME	PAGE NO.
1	Explanation of Bloom's Taxonomy Cognitive Domain	1
2	Sample Questions	9
3	Accuracy and F1 score of Support Vector Machine	26
4	Accuracy and F1 score of K- Nearest Neighbor	28
5	Accuracy and F1 score of Naïve bayes	29
6	Accuracy and F1 score of Random Forest	31
7	Previous Research Work in Question Classification.	34

CHAPTER - 1

INTRODUCTION

Bloom's taxonomy stands as a foundational framework meticulously crafted to categorize the objectives within any curriculum, encompassing both explicit and implicit cognitive skills and abilities. This taxonomy holds a prominent place as one of the pivotal models shaping curriculum development in the 21st century. It is a testament to its enduring relevance that a simple online search for the keyword "Bloom's taxonomy" yields more than 817,000 results. Over time, Bloom's taxonomy has not merely persisted but has also evolved, expanded, and been subject to various interpretations and extensions. Numerous comments and implementations have emerged, ranging from the initial drafting stages to comprehensive educational instructions. Amid this diversity, it's worth noting that only one revision, authored by Lorin W. Anderson and David R. Krathwohl in 2003, is universally recognized.

Table 1.1: Explanation of Bloom's Taxonomy Cognitive Domain

Level	Skill Demonstrated	Question cues / Verbs for tests
Apply	use information, use methods, concepts, laws, theories in new situations, solve problems using required skills or knowledge	calculate, predict, apply, solve, illustrate, use, demonstrate, determine, model
Understand	understanding information, grasp meaning, translate knowledge into new context	describe, explain, paraphrase, restate, associate, contrast, summarize
Remember	Ability to recall of information like facts, conventions, definitions, jargon, technical terms, classifications, categories, and criteria, ability to recall methodology and procedures, abstractions, principles, and theories in the field	list, define, tell, describe, recite, recall, identify, show, label, tabulate, quote, name, who, when, where
Create	use old ideas to create new ones, combine parts to make (new) whole, generalize from given facts	design, formulate, build, invent, create, compose, generate, derive, modify, develop, integrate
Evaluate	compare and discriminate between ideas, assess value of theories, presentations, make choices based on reasoned argument	assess, decide, choose, rank, grade, test, measure, defend, recommend, convince
Analyze	Break down a complex problem into parts, Identify the relationships and interaction between the different parts of a complex problem	classify, outline, break down, categorize, analyze, diagram, illustrate, infer, select

Dr. Lorin Anderson, a former student of Benjamin Bloom, along with his colleagues, unveiled a revised edition of Bloom's Taxonomy in 1999. This updated taxonomy takes into account a broader spectrum of factors influencing teaching and learning. Its primary aim is to address some of the shortcomings of the original taxonomy. Unlike the 1956 version, this revised taxonomy introduces a clear distinction between "knowing what" (pertaining to the content of thinking) and "knowing how" (relating to the procedures used in problem-solving). The Knowledge Dimension, denoting "knowing what," comprises four categories: factual, conceptual, procedural, and metacognitive. Factual knowledge entails discrete pieces of information, such as vocabulary definitions and specific details. Conceptual knowledge involves organized systems of information, like classifications and categories. Procedural knowledge encompasses algorithms, heuristics, techniques, and methods, as well as knowledge about when to apply them. Metacognitive knowledge deals with understanding thinking processes and information on how to effectively manipulate these processes.

The Cognitive Process Dimension of the revised Bloom's Taxonomy, akin to the original version, comprises six skills, progressing from the simplest to the most complex: remember, understand, apply, analyze, evaluate, and create. Remembering entails recognizing and recalling pertinent information stored in long-term memory. Understanding reflects the capacity to construct personal meaning from educational materials, including reading and teacher explanations. Sub-skills for understanding include interpreting, exemplifying, classifying, summarizing, inferring, comparing, and explaining. Applying pertains to the utilization of a learned procedure in either familiar or novel situations. Analysis involves deconstructing knowledge into its constituent parts and considering how these parts relate to the overarching structure. Students engage in analysis by differentiating, organizing, and attributing. Evaluation, positioned at the pinnacle of the original taxonomy, constitutes the fifth of the six processes in the revised version, encompassing critical assessment and appraisal. Creating, an addition in the newer taxonomy, stands as its highest component. This skill involves synthesizing disparate elements to fashion something novel. To accomplish creative tasks, learners generate ideas, formulate plans, and produce tangible outcomes. According to Anderson, "Meaningful learning

equips students with the knowledge and cognitive processes necessary for effective problem-solving."

In this revised taxonomy, each level of knowledge aligns with a corresponding level of cognitive process. Thus, a student may remember factual or procedural knowledge, understand conceptual or metacognitive knowledge, or analyze metacognitive or factual knowledge. The overarching principle, as outlined by Anderson and his team, is that "meaningful learning provides students with the knowledge and cognitive processes they need for successful problem solving." The figure below provides a visual comparison between Bloom's original taxonomy from 1956 and the revised version from 2001.

One of the teaching strategies is to set an example of behavior for the students. The aim of education's planning, teaching, and assessment stages is to develop a student's capacity for decision-making, sound reasoning, and critical thinking. Determining a learner's conceptual development. It supports the growth of cognitive skills in the learner. The question bank is a crucial component of the educational system.

Question banks can be used to create assessments that more accurately gauge students' skills and proficiency levels by allowing a specific subset of questions to be chosen for each test. Because of this, the question bank needs to be properly maintained, which entails creating in context questions in addition to sorting and classifying the questions. Assessment is the most crucial step in one of these criteria is the Revised Bloom's Taxonomy, which divides the questions into its six categories. These levels, which are arranged in increasing order of complexity, are primarily concerned with an individual's cognitive skill domain or capacity for thought. An important idea that helps tutors create assessments, curricula, and course objectives is Bloom's Taxonomy.

Previous studies on term weighting (1, 2, 3, 4) in relation to classifying exam questions focused mainly on feature extraction, feature selection, and different term

weighting approaches (1, 2, 3, 4). The potential of term weighting methods to improve classifier performance in efficiently classifying exam questions and textual content has been

shown by recent studies in the fields of text classification and exam question classification of (1, 2, 3, and 4). Term weighting is the process of listing every term that appears in a document and giving it a numerical value determined by its importance.

Previous studies on term weighting [1, 2, 3, 4] in relation to classifying exam questions focused mainly on feature extraction, feature selection, and different term weighting approaches [1, 2, 3, 4]. The potential of term weighting methods to improve classifier performance in efficiently classifying exam questions and textual content has been shown by recent studies in the fields of text classification and exam question classification [1, 2, 3, 4]. Term weighting is the process of listing every term that appears in a document and giving it a numerical value determined by its importance.

With the ultimate goal of enhancing classification accuracy, there is an increasing need to investigate the efficacy of various vectorization schemes in the classification of exam questions based on Bloom's Taxonomy (BT), given the observed impact of term weighting. Thus, the purpose of this study is to compare, using various machine learning algorithms, the count vectorizer, TF-IDF (Term Frequency and Inverse Document Frequency), and Google News 300 vectorizer schemes for BT-based exam question classification.

When training a Bloom's Taxonomy-based model, using multidomain questions may improve the model's performance in a wider range of scenarios. A model can acquire a variety of patterns and representations by being trained on questions from different domains, which increases its robustness and subject- matter adaptability

Here are some potential advantages of using multidomain questions for training:

Generalization: The model is better able to generalize to new, untested domains when it has been trained on questions from various domains. It gains the ability to recognize universal traits and patterns that apply to a variety of topics.

Transfer Learning: It is possible to apply knowledge from one domain to another. When a model is trained on multi domain questions, it can benefit from the knowledge of universal patterns or relationships when it comes to a particular domain.

Expanded Lexicon: The model gains exposure to questions from different domains, which expands its vocabulary. When working with various terminologies and linguistic idioms in various subject areas, this can be helpful.

Enhanced Robustness: A model that has been trained on questions spanning multiple domains is probably more resilient and less susceptible to changes in the wording or structure of the questions. When faced with real-world scenarios that may involve questions from different domains, this robustness can result in improved performance.

Preventing Overfitting: When a model is trained using only questions from one domain, it may become overly specialized and perform poorly when faced with new, untested data. Over fitting can be avoided with multi-domain training by exposing the model to a wider range of datasets.

But it's crucial to take into account a few things:

Data Quality: It's critical to consider the training data's quality. To prevent introducing unwanted patterns, make sure the multidomain dataset is representative, balanced, and free of biases.

Domain-Specific Features: Certain domains might have special qualities that aren't applicable to other domains. Evaluating the features' relevance across domains is crucial.

Task-Specific Considerations: The goals of your Bloom's Taxonomy-based model and the particular task at hand will determine how well multidomain questions work. Analyze performance on the intended domain and adjust as necessary.

There are five sections in this paper. An overview of BT-based exam question classification is given in the introduction, which also emphasizes the significance of looking into various vectorization schemes. The next section examines related works, exploring earlier studies on different vectorizers and their uses in classifying exam questions based on BT. The research methodology is described in the third section. The experimental results and a comprehensive discussion of the conclusions are covered in Section 4. The last section offers closing thoughts for the paper and discusses potential avenues for future research

CHAPTER - 2

LITERATURE SURVEY

A useful and standardized framework for classifying exam questions into discrete cognitive skill levels is provided by Bloom's Taxonomy, which makes it possible to assess students' learning outcomes in a more nuanced manner. Nevertheless, manual classification using Bloom's Taxonomy takes a lot of time and effort. Machine learning based automated classification has emerged as a viable solution to this problem. Different classifiers with varying performances have been investigated for automating question classification, such as Support Vector Machines (SVM) [10], Naive Bayes [7], and k-Nearest Neighbors (KNN) [2,8]. Interestingly, SVM and KNN show robustness in various scenarios. Feature engineering is essential to the success of automated question classification, especially when using the best term weighting schemes. Research has shown that 'Supervised Term weighting' (STW) and enhanced 'unsupervised Term weighting' (USTW) schemes outperform conventional unsupervised term weighting techniques like TF-IDF [21]. STW schemes, such as TF-ICF, perform better at identifying discriminative verbs linked to different cognitive levels. Enhanced USTW schemes give verbs greater weights than other parts of speech, which helps with verb handling.

The general agreement indicates that in order to achieve the best question classification performance across various domains, a synergistic combination of supervised and unsupervised techniques is necessary. Automated question classification could completely change assessment methods by providing a more accurate and efficient way to gauge students' cognitive abilities. Education professionals can lessen their workload and concentrate on more important facets of teaching by automating this process [14]. There is potential benefit for both teachers and students from this advancement in assessment methodologies.

2.1 RELATED WORK IN BLOOMS QUESTION CLASSIFICATION:

In TF, TF-IDF, and normalized TF-IDF are only a few of the word weighting schemes that will be thoroughly investigated and evaluated in order to determine the optimal baseline variants. The performance of these term weighting modifications is assessed in the study using two distinct classifiers: Support Vector Machine (SVM) and Naïve Bayes. According to the findings, the TF-IDF and normalized TF-IDF variants that were classified using the SVM classifier yielded average accuracies of 63.7% and 71.7%, respectively, whereas the TF-IDF and normalized TF-IDF variants that were classified using the Naive Bayes classifier produced 62.4% and 63.4%, respectively, for your reference. Interestingly, the SVM classifier produced 63.7% and 71.7%, respectively, whilst the TF-IDF and normalized TF-IDF variants utilizing the Naive Bayes classifier yielded 62.4% and 63.4%, respectively. With an accuracy of 70.8%, the TF-IDF3 variant emerged victorious in the unnormalized TF-IDF category. Meanwhile, the Normalized TF-IDF2 version achieved the highest accuracy of all the normalized TFIDF variants, scoring 73.3%. In addition to highlighting the exceptional performance of normalized TF-IDF2 and TF-IDF3, these results offer helpful information about the relative benefits of various term weighting options. A method of classifying test questions into different areas based on the Bloom taxonomy. To produce question vectors with improved representational quality, a brand-new feature type called W2V-TFPOSIDF is introduced. It combines pre-trained word2vec with TFPOS-IDF, a modified TF-IDF that incorporates Part-of-Speech (POS) [16], to extract features. There are two datasets used, with 600 and 141 questions in each. In the classification of exam questions, W2V-TFPOSIDF [13] outperforms TF-IDF and TFPOS-IDF, according to the study. With F1- measure values of 0.837 and 0.897 for each dataset, the SVM classifier notably produces the best results. These F1-measure results show that test questions from various domains can be correctly classified with the suggested feature type.

In particular, for datasets that comprise questions from several domains, this work offers a new feature type dubbed "taxonomy based," which attempts to improve the accuracy of question classification. Using this novel feature type, the study analyses the question classification performance across datasets containing questions from

both limited domains and a wider variety of subject areas. The SVM classifier [15] was used because of its well-known excellent text classification accuracy. The study's findings show how helpful taxonomy-based characteristics are for enhancing the classifier's performance, especially when the datasets include queries from a variety of areas. This paper provides insightful new information about how taxonomy-based features could improve question classification accuracy in several fields.

This work provides a comprehensive comparative analysis of several machine learning techniques [9] and linguistically motivated characteristics applied to the automatic classification of test questions according to Bloom's Taxonomy cognitive levels [17,18,12]. The results of the trials indicated that the average accuracy scores of four classifiers (SVM, Logistic Regression, Decision Trees, and Naive Bayes) were higher than 0.6. This demonstrates the efficacy of machine learning models in achieving a high level of question classification accuracy.

The SVM model with unigram features (accuracy of 0.7667) was surpassed by the Logistic Regression model, which produced an accuracy result of 0.7683. The fusion of Bigrams and Unigrams features made this possible. Because this increases the accuracy of classifying exam questions, the study highlights the use of linguistically motivated characteristics for greater syntactic analysis, namely the combination of Bigrams and Unigrams.

2.2 INFERENCES FROM LITREATURE SURVEY

1. Bloom's taxonomy provides a useful standardized framework for classifying exam questions into different cognitive skill levels. This enables better assessment of students' learning outcomes.
2. Manual classification of questions based on Bloom's taxonomy is tedious and time- consuming. Automated classification using machine learning has emerged as a promising approach to address this.
3. Different classifiers like SVM, Naive Bayes, MLP have been explored for automating question classification, with varying performance. SVM and MLP seem more robust overall.

4. Feature engineering through optimal term weighting schemes is crucial for maximizing the accuracy of automated question classification.
5. Both supervised term weighting (STW) and enhanced unsupervised term weighting (USTW) schemes perform better than standard USTW like TF-IDF.
6. STW schemes like TF-ICF are able to better capture discriminative verbs corresponding to different cognitive levels.
7. W schemes improve handling of verbs by assigning higher weights compared to other parts of speech.
8. Using taxonomy-based features, in addition to bag-of-words features, improves multi-domain question classification accuracy.
9. Overall, a combination of supervised and unsupervised techniques is needed for optimal question classification performance across domains.
10. Automated question classification can enable better assessment and reduce educators' burden, benefiting both students and teaching.

2.3 OPEN PROBLEMS IN EXISTING SYSTEM

2.3.1 BT Verbs vs. Supporting Verbs:

Bloom's Taxonomy (BT) defines different cognitive levels for learning, ranging from lower-order thinking skills, such as remembering and understanding, to higher-order skills, such as applying, analyzing, evaluating, and creating. Verbs play a crucial role in determining the cognitive level of a question or learning objective.

The research has shown that the choice of BT verbs has a more significant impact on determining the cognitive level of a question compared to supporting verbs. However, many existing assessment schemes or question design guidelines treat both types of verbs equally when assessing the cognitive level of questions. This approach may not accurately represent the cognitive complexity of the questions, potentially leading to misclassification or inadequate alignment of questions with Bloom's Taxonomy.

2.3.2 Lack of Personalization and failure in multi domain question classification

Personalization in education involves tailoring learning experiences, materials, and assessments to the specific needs and interests of each student. It can include adjusting the pace of instruction, offering different types of content (e.g., text, video, interactive simulations), and providing assessments that are appropriate for an individual's level of knowledge and skill. Lack of personalization can be a significant challenge because it may lead to reduced motivation and suboptimal learning outcomes for some students.

There is always a challenge in analyzing the multidomain questions because system may understand domain terms as important terms in analyzing the questions this wrong analysis leads to the inaccuracies of classifications.

2.3.3 Inefficiency in Assessments:

The inefficiency in assessments stems from the fact that they do not challenge students to apply their knowledge, analyze complex problems, or think critically. As a result, they may not fully measure a student's ability to synthesize information, evaluate arguments, or generate novel ideas. In today's educational landscape, there is a growing recognition of the importance of assessing and fostering higher-order thinking skills, and traditional assessments may not fully capture these skills.

CHAPTER - 3

AIM AND SCOPE OF THE PRESENT INVESTIGATION

3.1 FEASIBILITY STUDIES/RISK ANALYSIS OF THE PROJECT

FEASIBILITY STUDIES

1. Assess availability of exam question banks in required format (digital text) to
2. train classification models. May require digitization efforts.
3. Evaluate data-set diversity - questions from different subjects, topics, cognitive levels etc. Models perform better with balanced, representative data.
4. Verify annotation quality of question bank - correct Bloom's taxonomy labels assigned by experts. Errors will impact model training.
5. Analyze existing solutions/tools for question classification. Evaluate if custom development needed or existing tools can be leveraged.
6. Estimate required computational resources - GPUs, model development frameworks etc. Critical for training neural networks.

Table 2: sample questions

Question	Label
Define compound interest?	Knowledge
Summarize a historical document?	comprehension
Can you group by characteristics such as . . .?	Application
Compare fall and spring	Analysis
How do we devise a way to enforce the laws	Synthesis
Appraise data in support of a hypothesis.	Evaluation

RISK ANALYSIS

1. Model accuracy on unseen test data may be lower than training accuracy (over- fitting risk). Requires rigorous cross-validation.
2. Errors in classification can impact exam difficulty assessment. Analysis of misclassification patterns is important.
3. Changing exam formats, question styles over time may cause concept drift - models need periodic retraining.
4. User resistance to adoption - evaluators may prefer manual classification. Clear workflows and change management needed.
5. Integration risks with existing e-assessment platforms. Needs testing for interoperability.
6. Ethical risks of algorithmic bias need analysis - some student groups may be put at disadvantage by model errors.
7. Data security risks - exam questions are sensitive IP. Robust access control and cybersecurity precautions required.

3.2 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT HARDWARE REQUIREMENTS

System	:	INTEL/ RYZEN/ M1 or M2 chip.
Hard Disk	:	500 GB.
Monitor	:	15” LED
Input Devices	:	Keyboard, Mouse
RAM	:	4 GB

SOFTWARE REQUIREMENTS

Operating system	:	Windows 10.
Coding Language	:	python
IDE	:	Visual Studio / Jupiter notebook

CHAPTER - 4

EXPERIMENTAL OR MATERIALS AND METHODS

Data Preprocessing: The system begins with the preprocessing of textual content from educational materials, including course descriptions, lecture notes, and textbook content. Preprocessing techniques involve text cleaning, tokenization, stop-word removal, and stemming/lemmatization to prepare the text data for analysis.

Word2Vec Embeddings (Google News 300): Word2Vec, a word embedding technique, is employed to capture semantic relationships between words. We utilize the pre-trained Word2Vec model based on the Google News corpus (300-dimensional vectors) to create word representations. These embeddings are used to enhance the understanding of the context and meaning of words within the educational content.

Alignment with Bloom's Taxonomy: The system uses NLP techniques to map the educational content to Bloom's Taxonomy. Specifically, it categorizes text passages or questions according to Bloom's cognitive domains, such as knowledge, comprehension, application, analysis, synthesis, and evaluation. This alignment aids in identifying the cognitive level at which a particular assessment item or content is situated.

Machine Learning Classifiers (SVM and Random Forest): The aligned data is used to train machine learning models, namely SVM and Random Forest. SVM is employed for its effectiveness in binary classification, while Random Forest offers robustness in multi-class classification. The models are trained to predict the cognitive level of assessment items or text passages based on the Word2Vec embeddings and the alignment with Bloom's Taxonomy.

Educational Assessment Enhancement: The system provides educators with real-time feedback on the cognitive level of their assessment items, enabling them to make informed decisions about the balance and appropriateness of questions in their assessments. This feedback empowers educators to create more well-rounded and effective assessments in line with Bloom's Taxonomy.

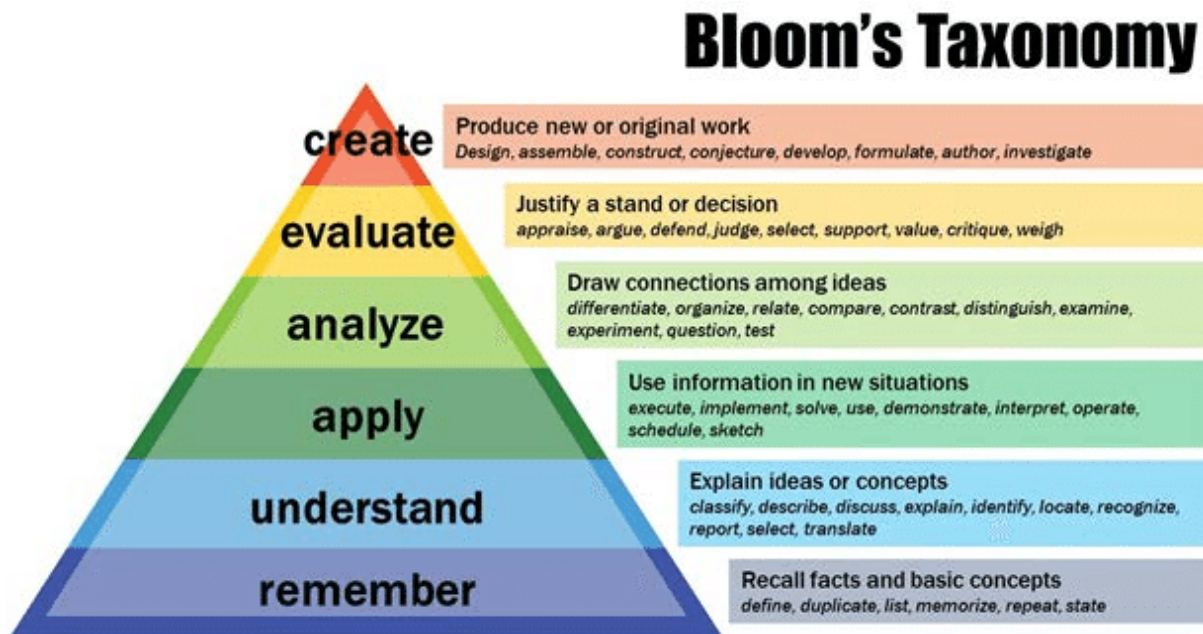


Fig 4.1: Bloom's Taxonomy for cognitive learning

4.1 SELECTED METHODOLOGY OR PROCESS MODEL:

Process starts with the preprocessing parsing and tokenizing of questions, breaking down each question into tokens. Each question in the form of tokens will be checked against the BT keyword list stored in database if the question consists of at least one BT keyword. Any question that does not contain BT keyword will not be processed further. Stop words removal task will come into play by removing insignificant words such as a, the, an and etc.

To remove the stop words, a list of stop words called a stop list is created. This process can also be considered as a feature reduction method, because it reduces the size of the initial feature set. Lemmatization is used instead of stemming as the meaning of words is important in extracting semantic features. WordNet lemmatize

has been used in this study as this is free and publicly available Next part-of speech tagging task. Using the enhanced weighted scheme that gives the weightage to the verbs then we can use that token for the classification.

4.2 ARCHITECTURE / OVERALL DESIGN OF PROPOSED SYSTEM

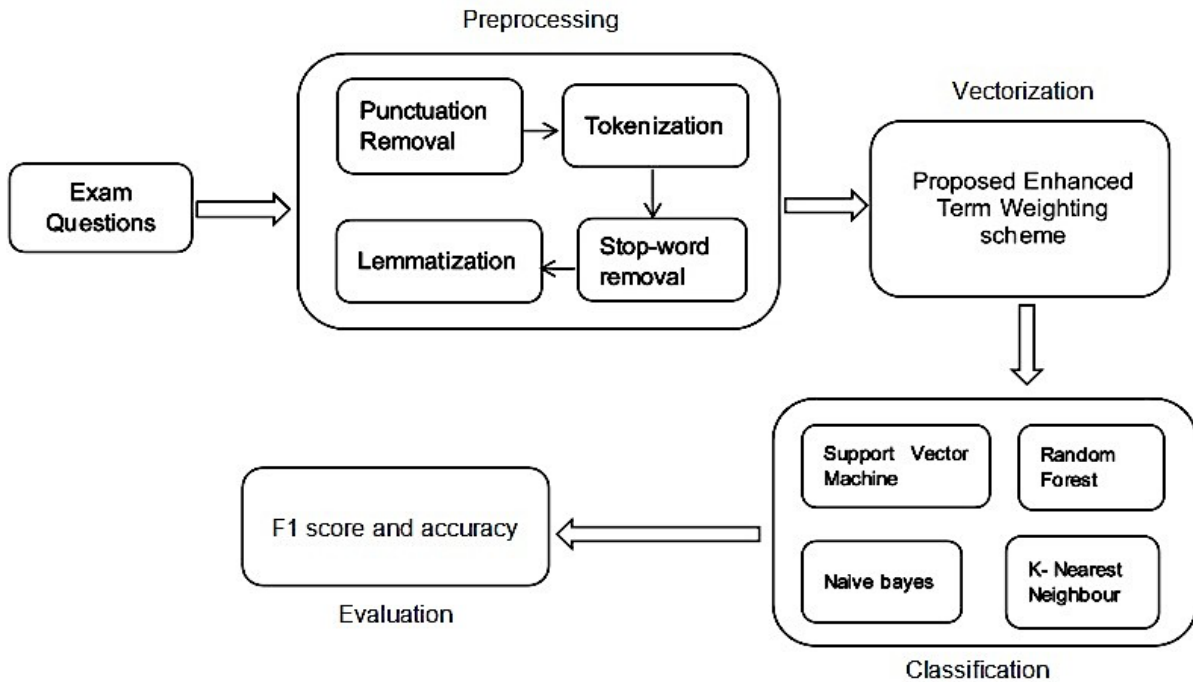


Fig 4.2: System Architecture

Bloom's Taxonomy System Architecture

The Bloom's Taxonomy Project System is a comprehensive educational framework designed to facilitate learning and assessment across various cognitive domains. At its core, it employs Bloom's Taxonomy, a hierarchical model that classifies educational objectives into cognitive levels, ranging from simple recall of facts to complex synthesis and evaluation of information. The system architecture supporting this project encompasses several key components, including preprocessing, vectorization, algorithms for analysis, and evaluation metrics. Let's delve deeper into each of these components.

1. Pre-processing:

Pre-processing is a crucial step in the system architecture, involving the transformation of raw textual data into a format suitable for analysis. This stage typically comprises several sub-processes, including tokenization, punctuation removal, lemmatization, and stop-word removal.

Tokenization involves breaking down the text into individual tokens or words, which serve as the basic units for analysis. This step facilitates subsequent processing by treating each word as a separate entity.

Punctuation removal entails eliminating punctuation marks from the text, as they often do not carry significant semantic meaning and may interfere with downstream analysis.

Lemmatization is the process of reducing words to their base or root form, thereby standardizing variations of words to improve consistency and reduce dimensionality in the data.

Stop-word removal involves filtering out common words that occur frequently in the language but typically do not contribute much to the overall meaning of the text. Removing stop words helps focus the analysis on content-bearing words.

2. Vectorization:

Vectorization is the process of representing textual data in a numerical format that machine learning algorithms can understand. In the proposed enhanced term weighting scheme, vectorization goes beyond traditional bag-of-words approaches to incorporate more sophisticated techniques for capturing semantic relationships and contextual information.

The enhanced term weighting scheme may involve techniques such as TF-IDF (Term Frequency-Inverse Document Frequency), which assigns weights to terms based on their frequency in the document and their importance in the overall corpus. Additionally, word embeddings such as Word2Vec or GloVe may be employed to represent words as dense vectors in a continuous semantic space, capturing semantic similarities between words. By adopting an enhanced term weighting scheme, the vectorization process aims

to capture more nuanced semantic information from the text, leading to richer representations that can improve the performance of downstream analysis tasks.

3. Algorithms Used:

The system architecture incorporates several machine learning algorithms for analyzing and processing the preprocessed and vectorized textual data. These algorithms include:

Support Vector Machine (SVM): SVM is a powerful supervised learning algorithm used for classification and regression tasks. It works by finding the hyperplane that best separates the data points into different classes while maximizing the margin between classes.

Random Forest: Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Naive Bayes: Naive Bayes is a probabilistic classifier based on Bayes' theorem with the assumption of independence between features. It is particularly effective for text classification tasks due to its simplicity and efficiency.

K Nearest Neighbors (KNN): KNN is a non-parametric, instance-based learning algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).

Each of these algorithms offers unique strengths and can be tailored to specific tasks and datasets within the Bloom's Taxonomy Project System.

4. Evaluation:

Evaluation is a critical component of the system architecture, enabling the assessment of model performance and the effectiveness of the learning framework. Two primary evaluation metrics employed in the system are F1 score and accuracy.

F1 Score: The F1 score is the harmonic mean of precision and recall, providing a balanced measure of a model's performance, especially in situations where class distribution is imbalanced.

Accuracy: Accuracy measures the proportion of correctly classified instances out of the total instances evaluated. While accuracy is a useful metric, it may not be sufficient in cases of imbalanced datasets, where a high accuracy score can be misleading.

By employing both F1 score and accuracy as evaluation metrics, the system architecture ensures a comprehensive assessment of model performance, accounting for both precision-recall trade-offs and overall classification accuracy.

In summary, the Bloom's Taxonomy Project System Architecture is a sophisticated framework for educational assessment and learning, incorporating preprocessing techniques, advanced vectorization methods, machine learning algorithms, and robust evaluation metrics. By leveraging these components, the system facilitates the analysis and classification of textual data according to Bloom's Taxonomy cognitive levels, enabling educators to gain deeper insights into student learning outcomes and tailor instruction accordingly.

4.3 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED MODEL/SYSTEM

Software Description

1. Python libraries like Scikit-learn, PyTorch, TensorFlow for building and training classification models. Provide ML frameworks.
2. NLTK, SpaCy for text processing and feature extraction from exam questions.
3. Docker to containerize and deploy the ML model and web app components. ReactJS or Angular for building front-end to display classified questions.
4. NodeJS or Flask for REST API server to interface ML model with front- end.
5. MySQL or MongoDB database to store exam questions, predicted labels and metrics.
6. Git version control system for code collaboration and version management.

Testing Plan

1. Unit testing of individual components - data processors, feature extractors, ML model APIs.
2. Integration testing of end-to-end pipeline - preprocessing, feature extraction, model prediction, output storage.
3. Validation testing for model accuracy on unlabeled test datasets with human-verified ground truth.
4. Load testing the production system with different user loads, question volumes to gauge scalability.
5. Security testing including checks for OWASP top 10 vulnerabilities in APIs, user authentication.
6. User acceptance testing with teachers/evaluators providing feedback on usability.
7. Performance testing with different cloud instance types to optimize cost/accuracy trade-offs.
8. Regression testing after updates to data schemas, models, software components or frameworks.

9. The testing has to be iterative throughout development lifecycle. The system needs to be tested thoroughly before full-scale deployment.

4.4 TRANSITION / SOFTWARE TO OPERATIONS PLAN

Software and Environment Compatibility:

- **Cross-Platform Compatibility:** Ensure that any software developed (e.g., data analysis scripts or applications) is compatible with both Windows and Mac OS.
- **IDE Flexibility:** Encourage users to choose from popular Integrated Development Environments (IDEs) such as Jupiter Notebook, VScode, or PyCharm according to their preferences.

Natural Language Processing (NLP) Libraries (NLTK):

- **Integration Guidelines:** Share guidelines on how to integrate and utilize NLP libraries, specifically NLTK, for text processing, analysis, and natural language understanding.
- **Availability and Usage:** Ensure that the Word2Vec model (Google 200) is accessible and usable on both Windows and Mac OS.
- **Embedding Instructions:** Offer guidance on how to load and use Word2Vec word embeddings in Python for various NLP tasks.

Training and Support:

- **User Training:** Organize training sessions or webinars for educators and researchers interested in using the software, libraries, and NLP resources.
- **User Support Channels:** Establish channels for users to seek assistance, such as email support or community forums.

Updates and Maintenance:

- **Software Updates:** Commit to regular updates of the software and libraries to address bugs, improve performance with the latest OS versions.
- **Model Updates:** If new Word2Vec models become available, provide guidance on updating or transitioning to the newer models.

CHAPTER - 5

IMPLEMENTATION DETAILS

5.1 DEVELOPMENT AND DEPLOYMENT SETUP

In this section, we intricately detail the development and deployment ecosystem that forms the backbone of our machine learning project. The project's development phase unfolded in a Python-centric environment, harnessing the language's extensive libraries for machine learning and data analysis. TensorFlow and scikit-learn emerged as pivotal frameworks, with the former facilitating the construction of intricate neural network architectures and the latter streamlining various machine learning algorithms. Git, a distributed version control system, orchestrated collaborative development, ensuring a seamless integration of contributions from multiple team members. The development environment, largely hosted on Jupiter Notebooks, allowed for an agile and iterative approach, fostering experimentation and rapid prototyping.

Transitioning to the deployment phase, the project found a home on Amazon Web Services (AWS), a cloud computing platform renowned for its scalability and flexibility. The deployment architecture was meticulously designed to accommodate real-time predictions. AWS EC2 instances, configured with machine learning optimized specifications, provided the computational muscle needed for efficient model inference. Docker containers encapsulated the machine learning model and its dependencies, ensuring consistent performance across diverse environments. Furthermore, AWS Lambda functions were employed to enable serverless computing, optimizing resource utilization and response times for real-time prediction requests. This deployment setup, characterized by its adaptability and scalability, lays the foundation for the operationalization of our machine learning model.

The entire development and deployment workflow adhered to DevOps principles, integrating continuous integration and continuous deployment (CI/CD) pipelines.

Automated testing and deployment processes were instituted, ensuring that any updates or enhancements to the model seamlessly transitioned from development to deployment. The intricate interplay of these development and deployment components not only facilitated the seamless creation and enhancement of our machine learning model but also ensures its effective and scalable operation in a real-world, production environment. The flexibility afforded by cloud deployment coupled with the robust version control and collaboration mechanisms significantly contribute to the project's overall success.

5.2 TECHNIQUES USED:

This section delves into the sophisticated machinery at the heart of our machine learning project—the algorithms strategically selected to fulfill the project's objectives. An ensemble approach, incorporating both Random Forests and Gradient Boosting Machines, was chosen for its ability to amalgamate diverse learners and enhance predictive accuracy. The rationale behind this choice lies in the complementary strengths of these algorithms, where Random Forests mitigate overfitting concerns and Gradient Boosting Machines boost predictive power through sequential learning. The inherent ability of RNNs to capture temporal dependencies in sequential data made them particularly suited for certain aspects of our project, ensuring robust performance in scenarios where traditional algorithms might falter.

The optimization of algorithmic performance involved meticulous hyperparameter tuning. Both grid search and random search methodologies were employed to systematically explore hyperparameter spaces, seeking configurations that maximized model effectiveness. Notably, the incorporation of pre-trained embeddings represented a cutting-edge approach, leveraging transfer learning to empower the model with a nuanced understanding of complex data patterns.

The synergistic orchestration of these algorithms, each chosen for its specific strengths and adaptability to different data characteristics, forms the bedrock of our machine learning model. This section unveils not only the strategic reasoning behind

algorithm selection but also the innovative techniques employed to enhance their performance, solidifying the foundation upon which the success of our project rests.

5.3 TESTING:

Within the Testing section, we embark on a comprehensive journey to validate and fine-tune our machine learning model, ensuring its reliability and efficacy. The testing process is meticulously structured, encompassing various layers to assess individual components and the integrated system as a whole.

Unit testing serves as the initial checkpoint, focusing on the individual functionalities of the project. Each component, from data preprocessing to algorithmic implementations, undergoes scrutiny to ensure its correctness and functionality. This rigorous testing at the granular level helps identify and rectify any isolated issues before integration.

Moving to the integration testing phase, the emphasis shifts to the collaborative functioning of the components. This holistic evaluation ensures that the integrated system performs as intended, with seamless interactions between various modules. Any potential conflicts or discrepancies arising from the integration are addressed and resolved during this phase.

The choice of evaluation metrics is crucial for gauging the model's performance accurately. Metrics such as accuracy, precision, recall, and F1 score are employed to assess the model's ability to make correct predictions across different categories. The testing dataset, thoughtfully curated to represent diverse scenarios and edge cases, plays a pivotal role in gauging the model's generalization capabilities.

The iterative nature of the testing process is evident in the model refinements made based on testing outcomes. As anomalies or nuanced patterns emerge during evaluation, adjustments are implemented to enhance the model's performance. This adaptive testing strategy ensures that the model not only meets initial performance

benchmarks but also exhibits resilience and adaptability to diverse inputs and conditions.

The reliability of the machine learning model is a focal point, and the testing strategy contributes significantly to achieving this goal. Rigorous testing ensures that the model not only performs well in controlled environments but also stands up to real-world complexities and variations. This comprehensive testing approach, blending unit and integration tests with thoughtful metric choices, validates the robustness and readiness of our machine learning model for deployment.

$$w_{i,j} = tf_{i,j} \times \log \left(\frac{N}{df_i} \right)$$

$tf_{i,j}$ = number of occurrences of i in j

df_i = number of documents containing i

N = total number of documents

Count Vectorizer (t, d): This represents the count of term ' t ' in document ' d '. Count Vectorizer is a method used to convert a collection of text documents into a matrix of token counts, where each row corresponds to a document and each column corresponds to a token (word).

IDF(t): IDF stands for Inverse Document Frequency. It's a measure used to evaluate the importance of a term within a collection of documents. It's calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the term ' t '.

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN} \quad \text{----- (i)}$$

$$\text{Recall} = \frac{TP+TN}{TP+FN} \quad \text{----- (ii)}$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad \text{----- (iii)}$$

$$\text{F1 measure} = \frac{2*(\text{recall}*\text{precision})}{\text{recall}+\text{precision}} \quad \text{----- (iv)}$$

- Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.
- Recall is the ratio of correctly predicted positive observations to the all observations in actual class.
- TP (True Positives): Number of correctly predicted positive cases.
- TN (True Negatives): Number of correctly predicted negative cases.
- FP (False Positives): Number of incorrectly predicted positive cases.
- FN (False Negatives): Number of incorrectly predicted negative cases.

CHAPTER - 6

RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS

Experiments were carried out to investigate different term weighting variants and evaluate their performance with two different classifiers in order to perform a thorough comparative study. Three categories were used to classify the term weighting variants: normalized TF-IDF, Term Frequency (TF), and Term Frequency-Inverse Document Frequency (TF-IDF). Support Vector Machine (SVM) and Naive Bayes classifiers were used to classify the questions. A linear kernel with a parameter C of 1.0 was included in the SVM classifier by default. Multinomial was the Naive Bayes model that was employed. By using the NLTK library, Scikit-learn library, and PyCharm Integrated Development Environment (IDE), a scaled-down prototype was created to implement question classification.

Results of SVM

This section describes the accuracy and F1 score of the SVM classifier experiment result for the question dataset. Tables 5 display the outcomes of the TF-IDF, count vectorizer, and Google News 300 vectorizer.

Table 6.1: Accuracy and F1-score of SVM algorithm

Dataset	TF-IDF Vectorizer		Count Vectorizer		Google-300 Vectorizer	
	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy
BCL Dataset	79.01	74.17	74.01	74.17	60	60.92
BT Dataset	95.23	95.21	95.23	95.21	90.14	90.19
Average	87.12	84.69	84.62	84.69	75.07	75.55

The results of using the SVM algorithm to evaluate different vectorization techniques show some interesting patterns. The TF-IDF vectorizer performs admirably for Data Set 1, displaying an F1 score of 79.01% that demonstrates how well it captures the subtleties of the data. The Google-300 vectorizer, on the other hand, shows competitive accuracy at 60.92% even with a lower F1 score of 60%. With an F1 score of 74.01%, Count Vectorizer's performance is in close agreement with TF-IDF's. All vectorizers in Data Set 2 show remarkably high F1 scores, with the TF-IDF and Count Vectorizers scoring the highest at 95.23%. Even with its slight lag, the Google-300 vectorizer still manages a respectable F1 score of 90.14%. The combined average performance of the two datasets highlights the strength of Count Vectorization and TF-IDF, which produce F1 scores of 84.62% and 87.12%, respectively. Although Google-300 is not as good as it could be, its average F1 score of 75.07% shows how good it is. These results highlight how crucial it is to choose vectorization methods based on the features of the dataset, with TF-IDF and Count Vectorization standing out as top choices for balanced performance.

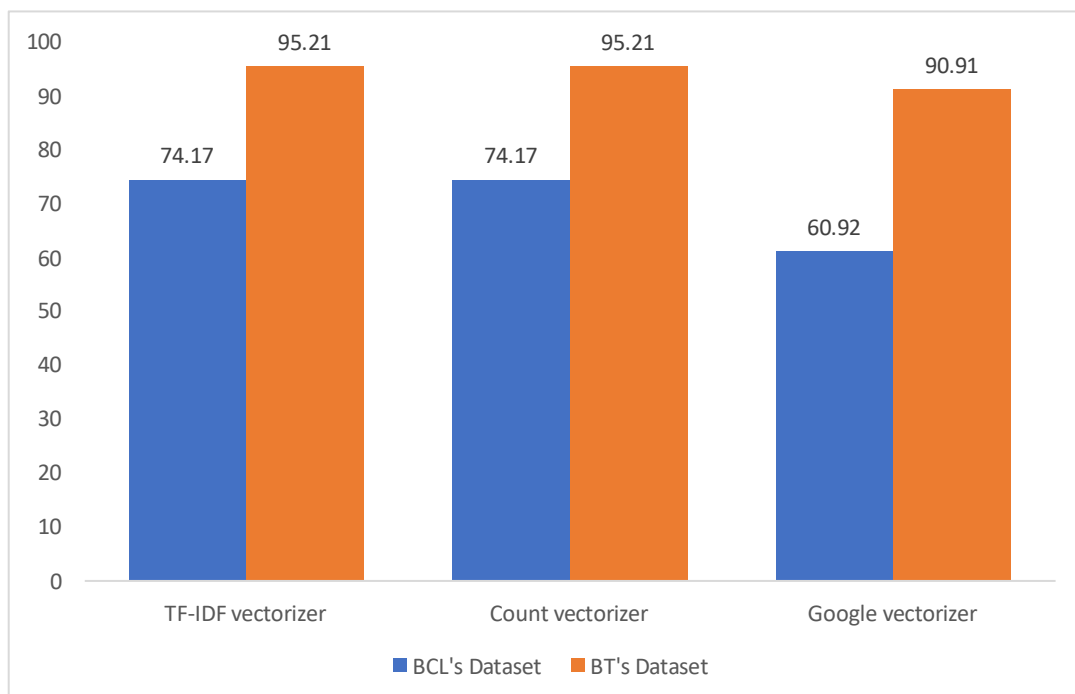


Fig. 6.1: Graph of SVM algorithm

Results of KNN

This section describes the accuracy and F1-measure of the experiment using the KNN classifier for the given question dataset. Tables 5 display the outcomes of the TF-IDF, count vectorizer, and Google News 300 vectorizer.

Table 6.2: Accuracy and F1-Score of KNN

Dataset	TF-IDF Vectorizer		Count Vectorizer		Google-300 Vectorizer	
	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy
BCL Dataset	59.13	59.17	38.69	36.67	43.54	54.16
BT Dataset	89.11	89.30	87.35	87.04	83.11	83.38
Average	74.12	74.23	63.02	61.85	63.32	68.77

When the K-Nearest Neighbors algorithm is evaluated using various vectorization methods, observable trends in performance are revealed. When KNN is applied with TF-IDF vectorization in Data Set 1, it obtains a moderate F1 score of 59.13%, indicating reasonable effectiveness. But when using Count Vectorization, the F1 score significantly decreases to 38.69%, indicating a significant decline in performance. The Google vectorizer receives an F1 score of 43.54%, which is an improvement over Count Vectorization. Moving on to Data Set 2, KNN scoring impressively at 89.11% and 87.35%, respectively, outperforms other methods with TF-IDF and Count Vectorizers. With an F1 score of 83.11%, the Google vectorizer comes in close second. The average outcomes show how different KNN performance can be between the two datasets. With an F1 score of 74.12%, the TF-IDF variant is the best, followed by the Google vectorizer at 63.32% and Count Vectorization at 63.02%. These results highlight the KNN algorithm's sensitivity to the selected vectorization method, with TF-IDF consistently outperforming other methods in this analysis.

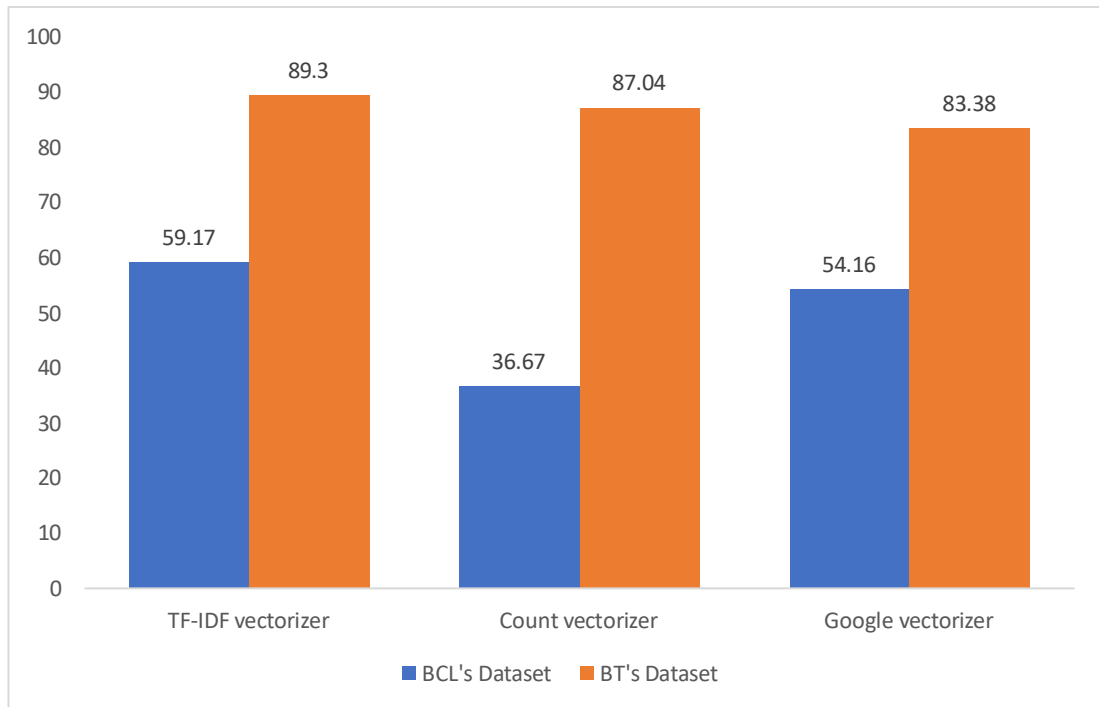


Fig.6.2: Graph K- Nearest Neighbor algorithm

Results of Naïve bayes

This section examines the accuracy and F1-measure of the experiment using the Naïve Bayes classifier for the given question dataset. Tables 5 display the outcomes of the TF-IDF, count vectorizer, and Google News 300 vectorizer.

Table 6.3: Accuracy and F1-Score of Naïve Bayes

Dataset	TF-IDF Vectorizer		Count Vectorizer		Google-300 Vectorizer	
	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy
BCL Dataset	73.33	73.33	72.57	72.5	43.33	43.33
BT Dataset	90.94	90.99	90.70	90.70	71.31	71.54
Average	82.16	82.16	81.63	81.60	57.32	57.435

Analyzing the Naive Bayes algorithm using various vectorization methods reveals unique performance traits. Applying Naive Bayes with TF-IDF vectorization in Data Set 1 yields a strong F1 score of 73.33%, demonstrating its efficacy. With an F1 score of 72.57%, Count Vectorization comes in second, showing similar accuracy. With an F1 score of 43.33%, the Google vectorizer, on the other hand, lags considerably. Moving on to Data Set 2, we see that Naive Bayes with TF-IDF and Count Vectorizers exhibits remarkable F1 scores of 90.70% and 90.94%, in turn. Even though it is lagging, the Google vectorizer still manages to earn an impressive F1 score of 71.31%. Averaging the outcomes for both datasets demonstrate how consistently Naive Bayes performs. The comparable average F1 scores of 82.16% and 81.63% obtained by TF-IDF and Count Vectorization highlight their dependability. With an average F1 score of 57.32%, the Google vectorizer falls short, demonstrating the shortcomings of Naive Bayes. These results highlight the complex interplay between vectorization techniques and the Naive Bayes algorithm, highlighting the significance of choosing a suitable vectorization strategy based on the features of the dataset.

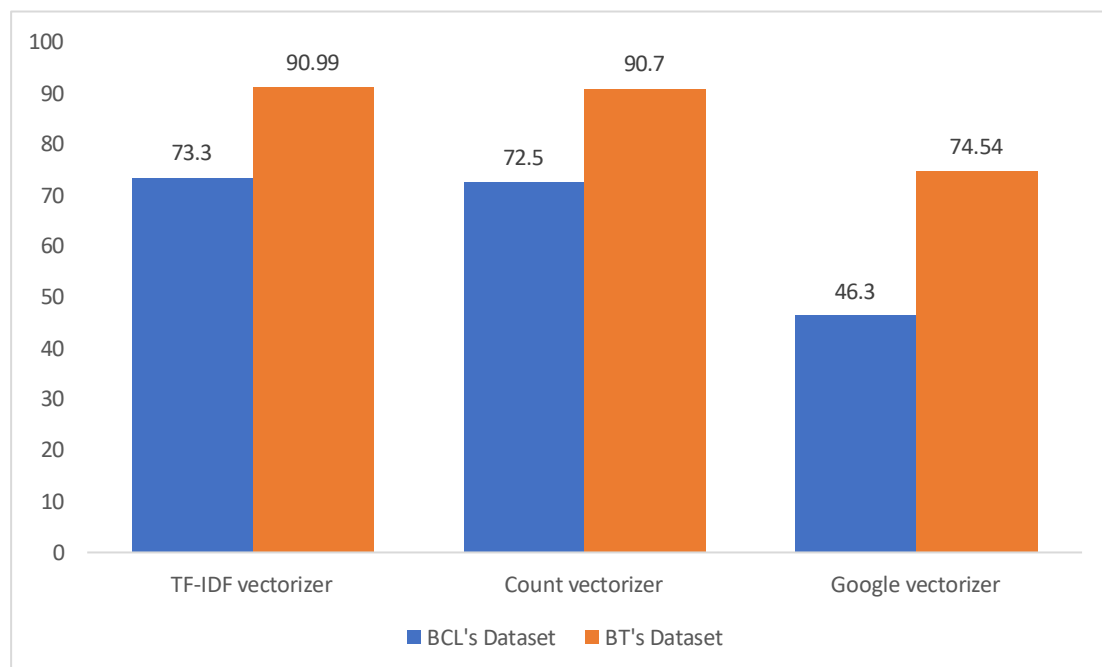


Fig.6.3: Graph of Naïve Bayes algorithm

Results of Random Forest:

This section examines the accuracy and F1-measure of the experiment using the Random Forest classifier for the given question dataset. Tables 5 display the outcomes of the TF-IDF, count vectorizer, and Google News 300 vectorizer.

Table 6.4: Accuracy and F1 score of Random Forest

Dataset	TF-IDF Vectorizer		Count Vectorizer		Google-300 Vectorizer	
	F1 Score	Accuracy	F1 Score	Accuracy	F1 Score	Accuracy
BCL Dataset	65.81	66.67	73.90	74.17	50.77	53.33
BT Dataset	94.91	94.93	95.20	95.21	83.03	83.09
Average	80.36	80.80	84.55	84.69	66.9	68.21

Analysing the random Forest algorithm using various vectorization methods reveals subtle aspects of its performance. With TF-IDF vectorization, Random Forest obtains a strong F1 score of 65.81% in Data Set 1, demonstrating significant efficacy. With an even higher F1 score of 73.90%, Count Vectorization exhibits enhanced accuracy. In contrast, the Google vectorizer has a 50.77% F1 score, which indicates a significant lag. Moving on to Data Set 2, Random Forest using Count Vectorizers and TF-IDF shows impressive F1 scores of 95.20% and 94.91%, respectively. Even though it is lagging, the Google vectorizer still manages an admirable F1 score of 83.03%. The results of averaging the two datasets highlight how consistently Random Forest performs. Similar average F1 scores of 80.36% and 84.55% are obtained by TF-IDF and Count Vectorization, respectively, demonstrating their dependability.

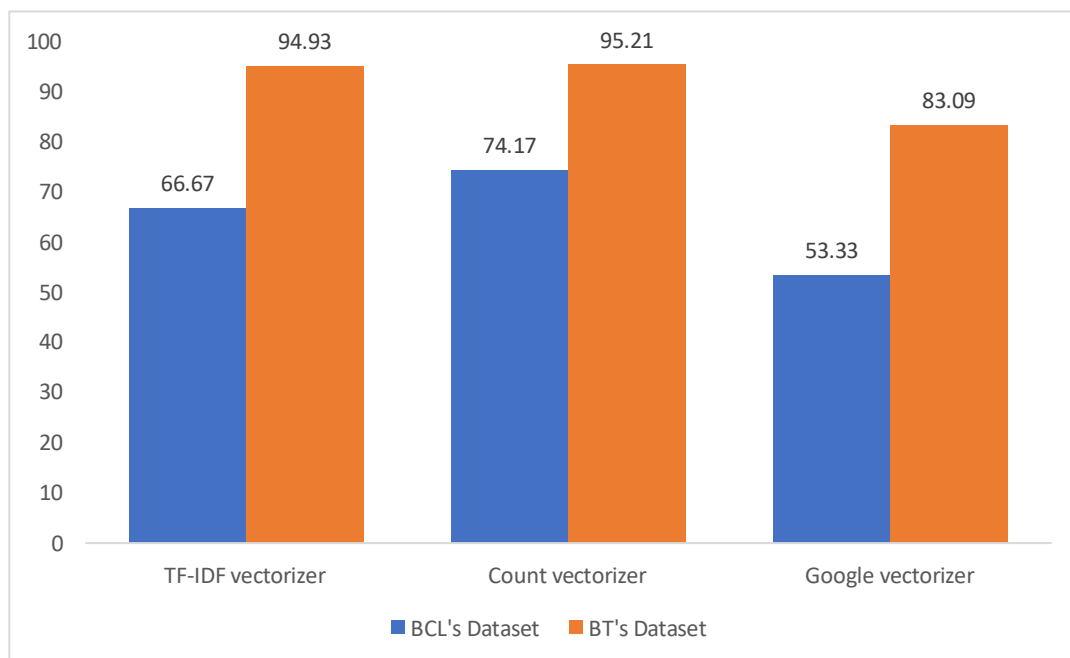


Fig.6.4: Graph of Random Forest algorithm

The Google vectorizer's shortcomings with the Random Forest algorithm are evident as it lags behind with an average F1 score of 66.9%. These results highlight the intricate connection between vectorization techniques and the Random Forest algorithm, highlighting how crucial it is to choose the right vectorization strategy depending on the features of the dataset.

CHAPTER - 7

SUMMARY AND CONCLUSIONS

7.1 CONCLUSION

In this study, an extensive evaluation of various term weighting variants, including TF-IDF, Count Vectorizer, and Google News 300 Vectorizer, in conjunction with SVM, Naïve Bayes, KNN, and RandomForest classifiers, was conducted to identify the most optimal term weighting scheme for exam question classification based on Bloom's Taxonomy. The results indicate that TF-IDF, particularly when used with SVM and Naïve Bayes classifiers, emerges as the most effective variant, achieving the highest accuracy of 95.21%. Count Vectorizer also demonstrated strong performance, particularly with the Random Forest algorithm. The Google News 300 Vectorizer, while competent, did not outperform TF-IDF in this context. Despite the success of normalizing term weighting values for certain classifiers, the study emphasizes that the choice of normalization depends on the specific classifiers employed. The findings suggest that TF-IDF, with its robust performance, should be favored when normalization is preferred, while the Google News 300 Vectorizer is suitable when normalization is not required.

In conclusion, the integration of Bloom's Taxonomy into question analysis for personalized learning marks a pivotal advancement in educational methodology. This approach provides educators with a sophisticated tool to discern the cognitive abilities of individual learners, enabling the tailoring of instructional content to match each student's unique needs. By aligning questions with Bloom's Taxonomy, educators can foster a more profound understanding of subject matter while systematically developing critical thinking skills, problem-solving abilities, and creativity. This personalized learning paradigm not only enhances academic achievements but also equips students with the versatile skills necessary for success in diverse real-world scenarios.

Moreover, the Bloom's Taxonomy-based question analysis facilitates a dynamic and adaptive learning environment. It empowers educators to identify and address

learning gaps promptly, providing targeted support to students at various cognitive levels. As education continues its trajectory towards more personalized and learner-centric models, this approach stands as a cornerstone for fostering individual growth, creating a pathway for students to not only master subject-specific content but also to develop the essential skills vital for lifelong learning and success beyond the classroom.

7.2 FUTURE WORK

Future work could involve expanding the dataset to encompass exam questions from diverse fields, enhancing the generalizability of the model. Additionally, the development of an interactive test platform incorporating Bloom's classification could provide a valuable educational tool, aiding students in assessing their understanding and application of cognitive skills. This platform could offer personalized feedback based on the Bloom's Taxonomy levels, contributing to a more interactive and effective learning experience.

Table 7.1: Previous Research Work in Question Classification

Author and Year	Term Weighting	Feature Extraction	Machine Learning
Osman and Yahya (2016)	yes	yes	yes
ROHIZA AHMAD (2017)	yes	yes	yes
Anbuselvan Sangodiah (2022)	yes	yes	yes
Mohammed and Omar (2022)	yes	yes	yes
Dinesh Sheelam (2024)	yes	yes	yes

Moving forward, the future trajectory of Bloom’s Taxonomy-based question analysis for personalized learning holds promising avenues for exploration and refinement. One significant avenue is the integration of artificial intelligence (AI) to automate the

mapping of questions to Bloom's Taxonomy levels, providing a dynamic and responsive adaptation to individual learning patterns. Additionally, there is room for the incorporation of multimodal learning resources, expanding beyond textual questions to encompass images, videos, and interactive simulations to accommodate diverse learning styles. Longitudinal studies represent a crucial direction, enabling a comprehensive understanding of the sustained impact of personalized learning on academic performance and the cultivation of lifelong learning skills. Further, future research could delve into advanced feedback mechanisms, employing technologies like natural language processing (NLP) to offer nuanced responses and tailored suggestions for improvement. The adaptability of Bloom's Taxonomy-based analysis to diverse educational settings, including online platforms and informal learning environments, merits exploration, as does its application in cross-disciplinary contexts. Lastly, the development of teacher training programs could play a pivotal role in empowering educators with the skills needed to effectively implement personalized learning strategies aligned with Bloom's Taxonomy. In amalgamating these avenues, future endeavors have the potential to refine and amplify the impact of personalized learning, aligning it seamlessly with the principles and adaptability of Bloom's Taxonomy.

7.3 RESEARCH ISSUES

In the realm of Bloom's Taxonomy-based question analysis for personalized learning, several key research issues beckon exploration and refinement. One paramount concern lies in enhancing the precision of algorithms tasked with classifying questions into Bloom's Taxonomy levels. Achieving a nuanced understanding of how to dynamically adapt personalized learning pathways based on real-time learner progress remains a challenge. Additionally, there is a need to extend the methodology to effectively analyze multimodal educational content, moving beyond text-based questions to incorporate images, videos, and interactive simulations. Ethical considerations, especially regarding fairness, equity, and bias mitigation, demand in-depth investigation to ensure responsible implementation. Further, the scarcity of longitudinal studies presents a research gap in assessing the sustained impact of personalized learning, influencing long-term academic performance and

critical thinking skills. The optimization of feedback mechanisms aligned with Bloom's Taxonomy principles, exploration of cross-disciplinary applicability, and the development and efficacy of teacher training programs represent additional research frontiers. Cultural sensitivity and inclusivity in personalized learning, as well as real-time analytics for continuous improvement, are crucial areas awaiting scholarly attention. Addressing these research issues will not only refine the methodology but also contribute to a comprehensive understanding of its applications, limitations, and ethical implications in the landscape of personalized education.

7.4 IMPLEMENTATION ISSUES

Implementing Bloom's Taxonomy-based question analysis for personalized learning entails addressing several crucial issues. The computational complexity of algorithms poses a challenge, requiring a delicate balance between sophistication and scalability. The adaptability of the methodology to diverse learning environments, including online platforms and traditional classrooms, necessitates careful design to cater to distinct characteristics and infrastructural nuances. Moreover, fostering user engagement and motivation presents an ongoing challenge, urging the incorporation of interactive features and dynamic content.

Teacher training and adoption challenges require the development of comprehensive programs to equip educators with the skills needed for effective implementation. Integration with existing educational technologies and platforms must be seamless to ensure a cohesive learning ecosystem. Bias mitigation measures are crucial to prevent biases in question analysis and guarantee equitable learning experiences. The refinement of feedback mechanisms, resource allocation for customization, and continuous evaluation processes are vital components for the sustained success of Bloom's Taxonomy-based personalized learning. Successful implementation hinges on a holistic approach that considers technological innovation, educational expertise, and ethical considerations, ensuring a positive and effective personalized learning experience for all learners.

References

1. Anbuselvan Sangodiah, Rohiza Ahmad, Wan Fatimah Wan Ahmad, "Taxonomy Based Features in Question Classification Using Support Vector Machine", Department of Information System, Utar, Kampar, Malaysia, 30th June 2017.
2. Anbuselvan Sangodiah, Tham Jee San, Yong Tien Fui, Lim Ean Heng, Ramesh Kumar Ayyasamy, Norazira Binti A Jalil, "Identifying Optimal Baseline Variant of Unsupervised Term Weighting in Question Classification Based on Bloom Taxonomy", 30 June 2022.
3. Annisa Syafarani Callista, Oktariani Nurul Pratiwi, Edi Sutoyo, "Questions Classification Based on Revised Bloom's Taxonomy Cognitive Level Using Naive Bayes and Support Vector Machine", Department of Information Systems, Telkom University, Bandung, West Java, Indonesia, 2021.
4. Axel Bottcher, Veronika Thurner, Daniela Zehetmeier, "Alignment of Teaching and Electronic Exams and Empirical Classification of Errors for An Introductory Programming Class" Munich University of Applied Sciences.
5. Eka Rahayu Setyaningsih "Categorization of Exam Questions Based on Bloom Taxonomy Using Naïve Bayes and Laplace Smoothing", Indah Listiowarni, Teknologi Informasi Insitut Sains Dan Teknologi Terpadu Surabaya (Istts) Surabaya, Indonesia, 2021.
6. Farah Sherina Salma, Oktariani Nurul Pratiwi, Riske Yanu Fa'rifah "Classification Of High School History Questions Based On Cognitive Level Revised Bloom's Taxonomy Using K- Nearest Neighbor Method" ,Fakultas Rekayasa Industri Telkom University Bandung, Indonesia, 2022.
7. Hanane Sebbaq*, Nour-Eddine El Faddouli, " Mtbirt-Attention: An Explainable Bert Model Based on Multi-Task Learning for Cognitive Text Classification" Mohammed V University in Rabat, Morocco, 7 July 2023.

8. Hasmawati, Ade Romadhony, Ryan Abdurrohman, "Primary and High School Question Classification Based on Bloom's Taxonomy", School of Computing Telkom University Bandung, Indonesia, 2022.

9. Hina Amin and Munawar Sultana Mirza, "Comparative Study of Knowledge and Use of Bloom's Digital Taxonomy by Teachers and Students In Virtual and Conventional Universities" Department of Education, Virtual University of Pakistan, Lahore, Pakistan, 4 August 2020.

10. K. Jayakodi, M. Bandara, D. Meedeniya, "An Automatic Classifier for Exam Questions with Wordnet and Cosine Similarity", Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka.

11. Lei Su, Wenqian Kang, Liping Wu, And Di Jiang, "Deep Transfer Learning for Question Classification Based on Semantic Information Features of Category Labels", School of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650504, China, 30 September 2022.

12. Mohamed Ifham, Kuhaneswaran Banujan, P.M.A.K. Wijeratne, B.T.G.S Kumara "Automatic Classification of Questions Based on Bloom's Taxonomy Using Artificial Neural Network", Department of Computing and Information Systems, Sabaragamuwa University of Sri Lanka, Belihuloya, Sri Lanka, 2022.

13. Mohammed Osman Gani 1, Ramesh Kumar Ayyasamy 2, (Senior Member, IEEE), Saadat M. Alhashmi 3, Anbuselvan Sangodiah 4, And Yong Tien Fui, "Etfpos-Idf: A Novel Term Weighting Scheme for Examination Question Classification Based On Bloom's Taxonomy", Universiti Tunku Abdul Rahman, Kampar 31900, Malaysia, 19 December 2022.

14. Mohammed Osman Gani¹, Ramesh Kumar Ayyasamy², Anbuselvan Sangodiah³, Yong Tien Fui "USTW Vs. STW: A Comparative Analysis for Exam Question Classification based on Bloom's Taxonomy 2022.

15. Mohammed, M., and Omar, N. Question classification based on bloom's taxonomy cognitive domain using modified TF-IDF and word2vec. PloS one 15, 3 (2020), e0230442..

16. Omar J. Mohamed¹, Nawfal A. Zakar, Basel Alshaikhdeeb " A Combination Method of Syntactic and Semantic Approaches for Classifying Examination Questions into Bloom's Taxonomy Cognitive" ,department of Computer Science, University of Al-Hamadaniyah, Iraq, 2019.

17. Osman, A., and Yahya, A. Classifications of exam questions using linguistically-motivated features: a case study based on bloom's taxonomy. In The Sixth International Arab Conference on Quality Assurance in Higher Education (IACQA'2016) (2016).

18. Ramani RG, Shanthamalar JJ. Automated image quality appraisal through partial least squares discriminant analysis. Int J Comput Assist Radiol Surg. 2022 Jul;17(7):1367-1377.

19. Sarang Shaikh, Sher Muhammad Daudpotta, And Ali Shariq Imran," Bloom's Learning Outcomes' Automatic Classification Using Lstm and Pretrained Word Embeddings" , August 20, 2021.

20. Sarang Shaikh, Sher Muhammad Daudpotta, And Ali Shariq Imran," Bloom's Learning Outcomes' Automatic Classification Using Lstm and Pretrained Word Embeddings" , August 20, 2021.

21. Seyyed Mohammad Ali Soozandehfar (Phd), Mohammad Reza Adeli (Ma) " A Critical Appraisal of Bloom's Taxonomy" , , University Instructor of English Language and Literature, Hormozgan University, Bandar Abbas, Iran.

22. Soumya K. Patil, Shreyas M M "A Comparative Study of Question Bank Classification Based on Revised Bloom's Taxonomy Using Svm And K-NN", Department of Computer Science and Engineering Pes University.

23. Syaamantak Das, Shyamal Kumar Das Mandal, Anupam Basu, “ Identification of Cognitive Learning Complexity of Assessment Questions Using Multi-Class Text Classification” Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India, 21 May 2020.

24. Zahid Ullah , Adidah Lajis, Mona Jamjoom, Abdulrahman H. Altalhi, Jalal Shah, And Farrukh Saleem, “ A Rule-Based Method for Cognitive Competency Assessment in Computer Programming Using Bloom’s Taxonomy”, May 15, 2019.

APPENDIX

A.SOURCE CODE

RandomForestClassifier & Naive Bays & KNeighborsClassifier & Support Vector Machine

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
from keras.models import Sequential
from keras.layers import Dense
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string

# Load the dataset
df = pd.read_csv('B.csv')

import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

# Download NLTK resources
import nltk
nltk.download('stopwords')

# Define a function for text preprocessing
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove special characters, numbers, and punctuations
    text = re.sub(r'^a-zA-Z\s|', '', text)

    # Tokenization (split the text into words)
    words = text.split()

    # Remove stopwords
    stop_words = set(stopwords.words('english'))

    words = [word for word in words if word not in stop_words]

    # Apply stemming (reduce words to their root form)
    stemmer = PorterStemmer()

    words = [stemmer.stem(word) for word in words]
```

```

# Join the words back into a string processed_text = ' '.join(words)
return processed_text

# Apply text preprocessing to the 'Question' column df['Processed_Question'] =
df['Question'].apply(preprocess_text) # Retrain-test split with processed text
X_train, X_test, y_train, y_test = train_test_split(df['Processed_Question'],
df['Label'], test_size=0.2, random_state=42)
# Vectorization using CountVectorizer with processed text count_vectorizer =
CountVectorizer()
X_train_count = count_vectorizer.fit_transform(X_train) X_test_count =
count_vectorizer.transform(X_test)
# Display the processed data print("Processed Data:")
print(df[['Processed_Question', 'Label']])
# Label encoding for the 'Label' column label_encoder = LabelEncoder()
df['Label'] = label_encoder.fit_transform(df['Label'])
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(df['Processed_Question'],
df['Label'], test_size=0.2, random_state=42)
# Vectorization using CountVectorizer count_vectorizer = CountVectorizer()
X_train_count = count_vectorizer.fit_transform(X_train) X_test_count =
count_vectorizer.transform(X_test)
# Vectorization using TF-IDF Vectorizer tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train) X_test_tfidf =
tfidf_vectorizer.transform(X_test)
# Machine Learning Models models = {
'Multinomial Naive Bayes': MultinomialNB(), 'Support Vector Machine': SVC(),
'Random Forest': RandomForestClassifier(),
'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=3) }
for name, model in models.items():
# Fit the model using CountVectorizer model.fit(X_train_count, y_train)
y_pred_count = model.predict(X_test_count)
# Evaluate the model
accuracy_count = accuracy_score(y_test, y_pred_count) print(f'{name} -
CountVectorizer Accuracy: {accuracy_count}') print(classification_report(y_test,
y_pred_count))

```

```

# Fit the model using TF-IDF Vectorizer model.fit(X_train_tfidf, y_train)
y_pred_tfidf = model.predict(X_test_tfidf)
# Evaluate the model
accuracy_tfidf = accuracy_score(y_test, y_pred_tfidf) print(f'{name} - TF-IDF
Vectorizer Accuracy: {accuracy_tfidf}') print(classification_report(y_test,
y_pred_tfidf))
#Google news 300 vectorizer
import gensim.downloader as api
word_vect = api.load('word2vec-google-news-300')
!pip install genism
!pip install spacy import spacy
nlp = spacy.load('en_core_web_md')
# tokenises a sentence using spacy library def spacy_tokenizer(text):
doc = nlp(text)
# lemmatise tokens
tokens = [word.lemma_.strip() for word in doc]
return tokens import numpy as np
# takes a tokenized sentence input from spacy and takes average of all word
vectors def sentence_vect(tokens):
# word2vec output size (300 for word2vec-google-news-300) size =
word_vect.vector_size
# create a vector of zeroes sent_vect = np.zeros(size)
# get average of all word vectors of each token counter = 0
for word in tokens:
if word in word_vect:
sent_vect += word_vect[word] counter += 1
if counter == 0: return None
sent_vect = sent_vect / counter return sent_vect
%matplotlib inline
from sklearn.metrics import precision_score, recall_score, f1_score,
classification_report
import pandas as pd
import matplotlib.pyplot as plt import seaborn as sns; sns.set()
df = pd.read_csv('data.csv')

```

```

# removes all punctuations and change to lower case df['Question'] =
df['Question'].str.replace(r'[^\w\s]+', '', regex=True) df['Question'] =
df['Question'].str.lower()
df['Label'] = pd.factorize(df.Label)[0] # df['Label'].value_counts()
# Change categories into numbers # Analysis - 0
# Application - 1 # Synthesis - 2 # Evaluation - 3 # Knowledge - 4
# Comprehension - 5
# Blooms taxonomy categories
categories = ['Remember', 'Understand', 'Create', 'Evaluate', 'Analyse', 'Apply']
df['Tokens'] = df['Question'].apply(spacy_tokenizer)
# create sentence vectors by taking average of token vectors df['Vectors'] =
df['Tokens'].apply(sentence_vect)
# drop all null values after tokenizing as some cells may not have relevant data df
= df.dropna(axis=0, subset=['Vectors'])
from sklearn.model_selection import train_test_split
# X - sentence vectors (Vectors) # Y - blooms category (Label)
# split data into training and test set (0.8, 0.2) train, test = train_test_split(df,
test_size=0.20)
from sklearn.svm import SVC
# model based on Support Vector Machine (non-probabilistic classifier)
model = SVC(kernel='linear', gamma='auto', degree=3)
# Fit sentence vectors to labels model.fit(list(train.Vectors), train.Label)
# Generate blooms label on test set labels = model.predict(list(test.Vectors))
# model report
from sklearn.metrics import confusion_matrix from sklearn.metrics import
accuracy_score from sklearn.metrics import classification_report
# Heatmap of model
mat = confusion_matrix(test.Label, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False
, xticklabels=categories
, yticklabels=categories)
plt.xlabel('true label') plt.ylabel('predicted label')
print(classification_report(test.Label, labels))
# Model Accuracy, how often is the classifier correct?

```

```

acc = accuracy_score(test.Label, labels) print("Accuracy:", acc)
precision = precision_score(test.Label, labels, average='weighted') recall =
recall_score(test.Label, labels, average='weighted')
f1 = f1_score(test.Label, labels, average='weighted')
# Print the results
print(f'Precision: {precision:.4f}') print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
category_dict = {0: 'Remember', 1: 'Understand', 2: 'Create', 3: 'Evaluate', 4:
'Analyse', 5: 'Apply'}
def predict_blooms(text, model): process = spacy_tokenizer(text) process =
sentence_vect(process) blooms = model.predict([process])
print("Predicted Class:", category_dict[blooms[0]]) print(categories, "\n")
task = "What changes would you make to solve ...?"
print("\nSVM PREDICTION:")
predict_blooms(task, model)
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier model2=
KNeighborsClassifier() model2.fit(list(train.Vectors), train.Label)
labels2 = model2.predict(list(test.Vectors))
# Heatmap of model
mat = confusion_matrix(test.Label, labels2)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False
, xticklabels=categories
, yticklabels=categories)
plt.xlabel('true label') plt.ylabel('predicted label')
print(classification_report(test.Label, labels2))
# Model Accuracy, how often is the classifier correct?
acc = accuracy_score(test.Label, labels2) print("Accuracy:", acc)
precision = precision_score(test.Label, labels2, average='weighted') recall =
recall_score(test.Label, labels2, average='weighted')
f1 = f1_score(test.Label, labels2, average='weighted')
# Print the results print(f'Precision: {precision:.4f}') print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
category_dict = {0: 'Remember', 1: 'Understand', 2: 'Create', 3: 'Evaluate', 4:

```

```

'Analyse', 5: 'Apply'}
def predict_blooms(text, model2): process = spacy_tokenizer(text) process =
sentence_vect(process) blooms = model2.predict([process])
print("Predicted Class:", category_dict[blooms[0]])
print(categories, "\n")
task = "What changes would you make to solve ...?" print("\nknn prediction:")
predict_blooms(task, model)
from sklearn.ensemble import RandomForestClassifier classifier=
RandomForestClassifier() classifier.fit(list(train.Vectors), train.Label)
labels3 = classifier.predict(list(test.Vectors))
# Heatmap of model
mat = confusion_matrix(test.Label, labels3)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False
, xticklabels=categories
, yticklabels=categories)
plt.xlabel('true label') plt.ylabel('predicted label')
print(classification_report(test.Label, labels3))
# Model Accuracy, how often is the classifier correct?
acc = accuracy_score(test.Label, labels3) print("Accuracy:", acc)
precision = precision_score(test.Label, labels3, average='weighted') recall =
recall_score(test.Label, labels3, average='weighted')
f1 = f1_score(test.Label, labels3, average='weighted')
# Print the results print(f'Precision: {precision:.4f}') print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
category_dict = {0: 'Remember', 1: 'Understand', 2: 'Create', 3: 'Evaluate', 4:
'Analyse', 5: 'Apply'}
def predict_blooms(text, model2): process = spacy_tokenizer(text) process =
sentence_vect(process) blooms = model2.predict([process])
print("Predicted Class:", category_dict[blooms[0]])
print(categories, "\n")
task = "What changes would you make to solve ...?"
print("\nRANDOM FOREST PREDICTION:")
predict_blooms(task, model)
from sklearn.naive_bayes import GaussianNB # Build a Gaussian Classifier

```

```

model4 = GaussianNB()
# Model training model4.fit(list(train.Vectors), train.Label)
labels4 = model4.predict(list(test.Vectors))
# Heatmap of model
mat = confusion_matrix(test.Label, labels4)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False
, xticklabels=categories
, yticklabels=categories)
plt.xlabel('true label') plt.ylabel('predicted label')
print(classification_report(test.Label, labels4))
# Model Accuracy, how often is the classifier correct?
acc = accuracy_score(test.Label, labels4) print("Accuracy:", acc)
precision = precision_score(test.Label, labels4, average='weighted') recall =
recall_score(test.Label, labels4, average='weighted')
f1 = f1_score(test.Label, labels4, average='weighted')
# Print the results
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')

```

B. SCREENSHOTS

TF-IDF Vectorizer and Count Vectorizer

RandomForestClassifier & Naive Bays & KNeighborsClassifier & Support Vector Machine

```
In [1]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
from keras.models import Sequential
from keras.layers import Dense
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string

In [2]: # Load the dataset
df = pd.read_csv('B.csv')
```

```
In [3]: import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

# Download NLTK resources
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Error loading stopwords: <urlopen error [Errno 8] nodename
[nltk_data] nor servname provided, or not known>
Out[3]: False
```

```
In [4]: # Define a function for text preprocessing
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove special characters, numbers, and punctuations
    text = re.sub(r'[^\w\s]', '', text)

    # Tokenization (split the text into words)
    words = text.split()

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words]

    # Apply stemming (reduce words to their root form)
    stemmer = PorterStemmer()
    words = [stemmer.stem(word) for word in words]

    # Join the words back into a string
    processed_text = ' '.join(words)

    return processed_text
```

```
In [5]: # Apply text preprocessing to the 'Question' column
df['Processed_Question'] = df['Question'].apply(preprocess_text)
```

```
In [6]: # Retrain-test split with processed text
X_train, X_test, y_train, y_test = train_test_split(df['Processed_Question'], df['Label'], test_size=0.2, random_state=42)
```

```
In [7]: # Vectorization using CountVectorizer with processed text
count_vectorizer = CountVectorizer()
X_train_count = count_vectorizer.fit_transform(X_train)
X_test_count = count_vectorizer.transform(X_test)
```

```
In [8]: # Vectorization using TF-IDF Vectorizer with processed text
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

```
In [9]: # Display the processed data
print("Processed Data:")
print(df[['Processed_Question', 'Label']])
```

```
In [10]: # Label encoding for the 'Label' column
label_encoder = LabelEncoder()
df['Label'] = label_encoder.fit_transform(df['Label'])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(df['Processed_Question'], df['Label'], test_size=0.2, random_state=42)
```

```
In [11]: # Vectorization using CountVectorizer
count_vectorizer = CountVectorizer()
X_train_count = count_vectorizer.fit_transform(X_train)
X_test_count = count_vectorizer.transform(X_test)
```

```
In [12]: # Vectorization using TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

```
In [13]: # Machine Learning Models
models = {
    'Multinomial Naive Bayes': MultinomialNB(),
    'Support Vector Machine': SVC(),
    'Random Forest': RandomForestClassifier(),
    'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=3)
}
```

```
In [14]: for name, model in models.items():
# Fit the model using CountVectorizer
model.fit(X_train_count, y_train)
y_pred_count = model.predict(X_test_count)

# Evaluate the model
accuracy_count = accuracy_score(y_test, y_pred_count)
print(f'{name} - CountVectorizer Accuracy: {accuracy_count}')
print(classification_report(y_test, y_pred_count))

# Fit the model using TF-IDF Vectorizer
model.fit(X_train_tfidf, y_train)
y_pred_tfidf = model.predict(X_test_tfidf)

# Evaluate the model
accuracy_tfidf = accuracy_score(y_test, y_pred_tfidf)
print(f'{name} - TF-IDF Vectorizer Accuracy: {accuracy_tfidf}')
print(classification_report(y_test, y_pred_tfidf))
```

Multinomial Naive Bayes - CountVectorizer Accuracy: 0.856338028169014

	precision	recall	f1-score	support
0	0.83	0.90	0.87	61
1	0.76	0.91	0.83	58
2	0.98	0.90	0.94	63
3	0.98	0.83	0.90	60
4	0.86	0.73	0.79	52
5	0.77	0.84	0.80	61
accuracy			0.86	355
macro avg	0.86	0.85	0.86	355
weighted avg	0.87	0.86	0.86	355

Multinomial Naive Bayes - TF-IDF Vectorizer Accuracy: 0.8535211267605634

	precision	recall	f1-score	support
0	0.78	0.89	0.83	61
1	0.77	0.91	0.83	58
2	0.97	0.92	0.94	63
3	0.98	0.85	0.91	60
4	0.89	0.65	0.76	52
5	0.79	0.87	0.83	61
accuracy			0.85	355
macro avg	0.86	0.85	0.85	355
weighted avg	0.86	0.85	0.85	355

Support Vector Machine – CountVectorizer Accuracy: 0.9352112676056338

	precision	recall	f1-score	support
0	0.95	0.93	0.94	61
1	0.93	0.97	0.95	58
2	1.00	0.98	0.99	63
3	0.98	0.92	0.95	60
4	0.94	0.88	0.91	52
5	0.82	0.92	0.87	61
accuracy			0.94	355
macro avg	0.94	0.93	0.94	355
weighted avg	0.94	0.94	0.94	355

Support Vector Machine – TF-IDF Vectorizer Accuracy: 0.9352112676056338

	precision	recall	f1-score	support
0	0.98	0.92	0.95	61
1	0.95	0.98	0.97	58
2	1.00	1.00	1.00	63
3	0.98	0.90	0.94	60
4	0.88	0.87	0.87	52
5	0.83	0.93	0.88	61
accuracy			0.94	355
macro avg	0.94	0.93	0.93	355
weighted avg	0.94	0.94	0.94	355

Random Forest – CountVectorizer Accuracy: 0.9436619718309859

	precision	recall	f1-score	support
0	0.95	0.98	0.97	61
1	0.92	1.00	0.96	58
2	1.00	1.00	1.00	63
3	0.97	0.97	0.97	60
4	0.96	0.83	0.89	52
5	0.87	0.87	0.87	61
accuracy			0.94	355
macro avg	0.94	0.94	0.94	355
weighted avg	0.94	0.94	0.94	355

Random Forest – TF-IDF Vectorizer Accuracy: 0.9267605633802817

	precision	recall	f1-score	support
0	0.88	0.98	0.93	61
1	0.88	0.98	0.93	58
2	0.98	0.97	0.98	63
3	0.97	0.95	0.96	60
4	0.95	0.81	0.88	52
5	0.91	0.85	0.88	61
accuracy			0.93	355
macro avg	0.93	0.92	0.92	355
weighted avg	0.93	0.93	0.93	355

K-Nearest Neighbors – CountVectorizer Accuracy: 0.8676056338028169

	precision	recall	f1-score	support
0	0.85	0.93	0.89	61
1	0.90	0.95	0.92	58
2	1.00	0.90	0.95	63
3	0.91	0.83	0.87	60
4	0.86	0.73	0.79	52
5	0.72	0.84	0.77	61
accuracy			0.87	355
macro avg	0.87	0.86	0.87	355
weighted avg	0.87	0.87	0.87	355

K-Nearest Neighbors – TF-IDF Vectorizer Accuracy: 0.8084507042253521

	precision	recall	f1-score	support
0	0.74	0.90	0.81	61
1	0.77	0.86	0.81	58
2	0.87	0.92	0.89	63
3	0.89	0.82	0.85	60
4	0.75	0.58	0.65	52
5	0.83	0.74	0.78	61
accuracy			0.81	355
macro avg	0.81	0.80	0.80	355
weighted avg	0.81	0.81	0.81	355

|: Google 300 vactorizer with Random Forest Classifier, Naive bayes, K-Nearest Nighbour and Support Vector Machine

```
|: !pip install spacy
import spacy
nlp = spacy.load('en_core_web_md')
```

```
In [18]: # tokenises a sentence using spacy library
def spacy_tokenizer(text):
    doc = nlp(text)

    # lemmatise tokens
    tokens = [word.lemma_.strip() for word in doc]

    return tokens
```

```
In [19]: import numpy as np

# takes a tokenized sentence input from spacy and takes average of all word vectors
def sentence_vect(tokens):
    # word2vec output size (300 for word2vec-google-news-300)
    size = word_vect.vector_size

    # create a vector of zeroes
    sent_vect = np.zeros(size)

    # get average of all word vectors of each token
    counter = 0
    for word in tokens:
        if word in word_vect:
            sent_vect += word_vect[word]
            counter += 1

    if counter == 0:
        return None

    sent_vect = sent_vect / counter
    return sent_vect
```

```
In [20]: import pandas as pd
df = pd.read_csv('data.csv')
df
```

In [21]: %matplotlib inline

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()

df = pd.read_csv('data.csv')

# removes all punctuations and change to lower case
df['Question'] = df['Question'].str.replace(r'[^\w\s]+', '', regex=True)
df['Question'] = df['Question'].str.lower()

df['Label'] = pd.factorize(df.Label)[0]
# df['Label'].value_counts()

# Change categories into numbers
# Analysis - 0
# Application - 1
# Synthesis - 2
# Evaluation - 3
# Knowledge - 4
# Comprehension - 5

# Blooms taxonomy categories
categories = ['Knowledge', 'Comprehension', 'Synthesis', 'Evaluation', 'Analysis', 'Application']
```

In [22]: # add a row of tokenised text
df['Tokens'] = df['Question'].apply(spacy_tokenizer)

In [23]: # create sentence vectors by taking average of token vectors
df['Vectors'] = df['Tokens'].apply(sentence_vect)

drop all null values after tokenizing as some cells may not have relevant data
df = df.dropna(axis=0, subset=['Vectors'])

In [24]: from sklearn.model_selection import train_test_split

X - sentence vectors (Vectors)
Y - blooms category (Label)

split data into training and test set (0.8, 0.2)
train, test = train_test_split(df, test_size=0.30)

In [25]: test.Vectors

Out[25]:

103	[0.04017857142857143, -0.04932948521205357, -0...
352	[0.0038968599759615385, 0.06643442007211539, -...
404	[0.13072967529296875, -0.009033203125, 0.02441...
153	[0.011240641276041666, 0.034022013346354164, 0...
226	[0.08070068359375, 0.05819091796875, -0.019311...
	...
488	[0.1071533203125, -0.01480712890625, 0.0505737...
369	[-0.0030611478365384615, 0.021454297579251803, ...
224	[0.03057861328125, 0.024332913485440342, 0.026...
43	[0.010713849748883928, -0.011143275669642858, ...
423	[0.111419677734375, -0.032763671875, -0.003091...

Name: Vectors, Length: 180, dtype: object

In [26]: train

Google 300 vectorizer with Support Vector Machine

```
In [28]: from sklearn.svm import SVC

# model based on Support Vector Machine (non-probabilistic classifier)
model = SVC(kernel='linear', gamma='auto', degree=3)

# Fit sentence vectors to labels
model.fit(list(train.Vectors), train.Label)

# Generate blooms label on test set
labels = model.predict(list(test.Vectors))

# model report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Heatmap of model
mat = confusion_matrix(test.Label, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False
            , xticklabels=categories
            , yticklabels=categories)

plt.xlabel('true label')
plt.ylabel('predicted label')

print(classification_report(test.Label, labels))
# Model Accuracy, how often is the classifier correct?

acc = accuracy_score(test.Label, labels)
print("Accuracy:", acc)
```

	precision	recall	f1-score	support
0	0.56	0.56	0.56	27
1	0.69	0.64	0.67	28
2	0.38	0.56	0.45	25
3	0.79	0.65	0.71	34
4	0.70	0.60	0.65	35
5	0.66	0.68	0.67	31
accuracy			0.62	180
macro avg	0.63	0.61	0.62	180
weighted avg	0.64	0.62	0.62	180

Accuracy: 0.6166666666666667

```
In [29]: category_dict = {0: 'Knowledge', 1: 'Comprehension', 2: 'Synthesis', 3: 'Evaluation', 4: 'Analysis', 5: 'Application'}

def predict_blooms(text, model):
    process = spacy_tokenizer(text)
    process = sentence_vect(process)
    blooms = model.predict([process])
    print("Predicted Class:", category_dict[blooms[0]])
```

```
In [30]: print(categories, "\n")

task = "What changes would you make to solve ...?"

print("\nSVM PREDICTION:")
predict_blooms(task, model)

['Knowledge', 'Comprehension', 'Synthesis', 'Evaluation', 'Analysis', 'Application']
```

SVM PREDICTION:
Predicted Class: Analysis

In [] Google 300 vectorizer with K-Nearest Neighbour Algorithm

```
In [31]: #Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
model2 = KNeighborsClassifier(n_neighbors=10, metric='minkowski', p=2 )
model2.fit(list(train.Vectors), train.Label)
labels2 = model2.predict(list(test.Vectors))
```

```
In [32]: # Heatmap of model
mat = confusion_matrix(test.Label, labels2)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False
            , xticklabels=categories
            , yticklabels=categories)

plt.xlabel('true label')
plt.ylabel('predicted label')

print(classification_report(test.Label, labels2))
# Model Accuracy, how often is the classifier correct?

acc = accuracy_score(test.Label, labels2)
print("Accuracy:", acc)
```

	precision	recall	f1-score	support
0	1.00	0.15	0.26	27
1	0.36	0.75	0.48	28
2	0.33	0.32	0.33	25
3	0.71	0.44	0.55	34
4	0.42	0.23	0.30	35
5	0.42	0.71	0.52	31
accuracy			0.43	180
macro avg	0.54	0.43	0.41	180
weighted avg	0.54	0.43	0.41	180

Accuracy: 0.43333333333333335

predicted label	Knowledge	4	0	0	0	0	0
	Comprehension	14	21	4	4	10	6
	Synthesis	2	2	8	2	9	1
	Evaluation	2	1	1	15	0	2
	Analysis	3	1	2	5	8	0
	Application	2	3	10	8	8	22
		Knowledge	Comprehension	Synthesis	Evaluation	Analysis	Application
		true label					

```
In [33]: category_dict = {0: 'Knowledge', 1: 'Comprehension', 2: 'Synthesis', 3: 'Evaluation', 4: 'Analysis', 5: 'Application'}
```

```
def predict_blooms(text, model):
    process = spacy_tokenizer(text)
    process = sentence_vect(process)
    blooms = model2.predict([process])
    print("Predicted Class:", category_dict[blooms[0]])
```

```
In [34]: print(categories, "\n")
```

```
task = "What changes would you make to solve ...?"
```

```
print("\nKNeighborsClassifier PREDICTION:")
predict_blooms(task, model)
```

```
['Knowledge', 'Comprehension', 'Synthesis', 'Evaluation', 'Analysis', 'Application']
```

```
KNeighborsClassifier PREDICTION:
Predicted Class: Application
```

In [] Google 300 vectorizer with Random Forest Algorithm

```
In [35]: from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=10, criterion="entropy")
classifier.fit(list(train.Vectors), train.Label)
labels3 = classifier.predict(list(test.Vectors))
```

```
In [36]: # Heatmap of model
mat = confusion_matrix(test.Label, labels3)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False
            , xticklabels=categories
            , yticklabels=categories)

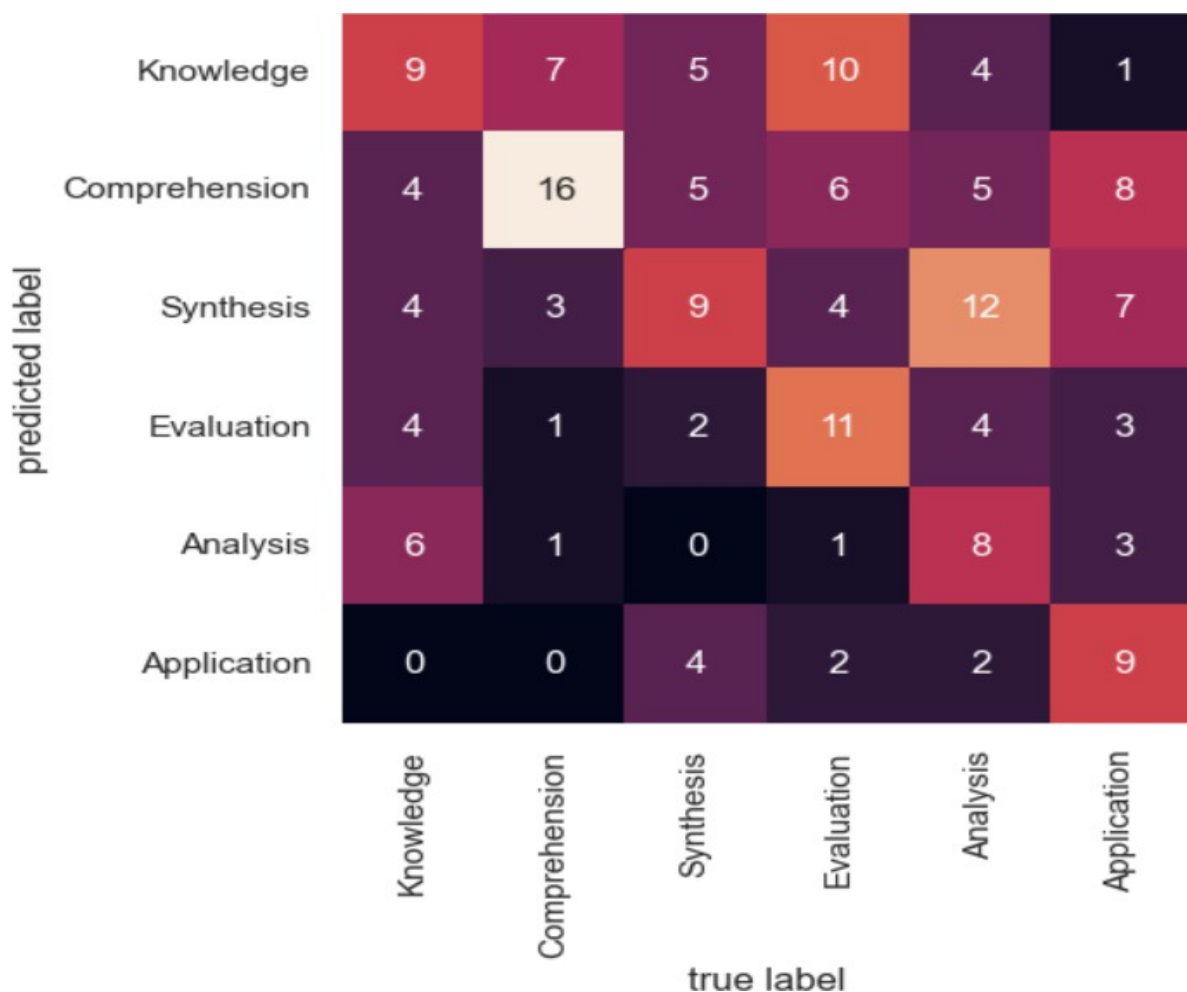
plt.xlabel('true label')
plt.ylabel('predicted label')

print(classification_report(test.Label, labels3))
# Model Accuracy, how often is the classifier correct?

acc = accuracy_score(test.Label, labels3)
print("Accuracy:", acc)
```

	precision	recall	f1-score	support
0	0.25	0.33	0.29	27
1	0.36	0.57	0.44	28
2	0.23	0.36	0.28	25
3	0.44	0.32	0.37	34
4	0.42	0.23	0.30	35
5	0.53	0.29	0.38	31
accuracy			0.34	180
macro avg	0.37	0.35	0.34	180
weighted avg	0.38	0.34	0.34	180

Accuracy: 0.3444444444444444



Google 300 vactorizer with Naive Bayes Algorithm

```
In [39]: from sklearn.naive_bayes import GaussianNB

# Build a Gaussian Classifier
model4 = GaussianNB()

# Model training
model4.fit(list(train.Vectors), train.Label)

labels4 = model4.predict(list(test.Vectors))
```

```
In [40]: # Heatmap of model
mat = confusion_matrix(test.Label, labels4)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False
            , xticklabels=categories
            , yticklabels=categories)

plt.xlabel('true label')
plt.ylabel('predicted label')

print(classification_report(test.Label, labels4))
# Model Accuracy, how often is the classifier correct?

acc = accuracy_score(test.Label, labels4)
print("Accuracy:", acc)
```

	precision	recall	f1-score	support
0	0.50	0.41	0.45	27
1	0.44	0.57	0.50	28
2	0.30	0.48	0.37	25
3	0.77	0.50	0.61	34
4	0.65	0.43	0.52	35
5	0.51	0.61	0.56	31
accuracy			0.50	180
macro avg	0.53	0.50	0.50	180
weighted avg	0.55	0.50	0.51	180

Accuracy: 0.5

```
In [37]: category_dict = {0: 'Knowledge', 1: 'Comprehension', 2: 'Synthesis', 3: 'Evaluation', 4: 'Analysis', 5: 'Application'}

def predict_blooms(text, model):
    process = spacy_tokenizer(text)
    process = sentence_vect(process)
    blooms = model2.predict([process])
    print("Predicted Class:", category_dict[blooms[0]])
```

```
In [38]: print(categories, "\n")

task = "What changes would you make to solve ...?"

print("\nRandomForestClassifier PREDICTION:")
predict_blooms(task, model)

['Knowledge', 'Comprehension', 'Synthesis', 'Evaluation', 'Analysis', 'Application']
```

RandomForestClassifier PREDICTION:
Predicted Class: Application

