Automated Answer Script Evaluation

By

Jishnu Bandyopadhyay

Ramakrishna Mission Vivekananda Centenary College, Rahara

Bitan Majumder

Ramakrishna Mission Vivekananda Centenary College, Rahara

Under the guidance of

Dr. Asit Kumar Das

Indian Institute of Engineering Science and Technology, Shibpur

Problem Statement

The answer script evaluation is a crucial task. A manually done evaluation can be biased. It depends on various factors like the mental condition of the examiner or the relationship between the student and the examiner. It is also a hectic and time-consuming task. In this article, a natural language processing-based method is discussed that could do automatic answer script evaluation. Our experiment consists of vectorization of answer text, clustering the vectors, and measuring similarity (Cosine).

Automatic evaluation of answer scripts has been found very useful in our experiments, and often the assigned marks are the same as manually scored marks.

Keywords - Automatic evaluation, NLP, Similarity measure, Clustering, Marks scoring

Problem Description

Answer script evaluation is a very important task. It is not only useful to assess the academic prowess of a student but it also gives insights into various characteristics of the student including time management ability, creativity, communication skills, etc. One of the most important types of questions that the students answer is the broad-answer type questions. Unlike Multiple Choice Questions (M.C.Q.) and Short-answer type questions that require only correct option(s) and one or two particular words, these questions span at least 4 to 5 lines in answer. The students express their opinions in response to the question in a textual way. However, it is a tedious job and might also be erroneous based on the condition of the evaluator. All students possess different thoughts, vocabulary, and ways of writing. All the answers should be different from each other, assuming that there is no malpractice. And the evaluator has to extract the main idea behind the answer from those lines to check if that is correct or not. Different answers possess different fractions of correctness.

The automation of this task will be helpful in numerous ways. It will help the institutions or exam-conducting bodies to evaluate a huge number of answers effortlessly and efficiently. The M.C.Q. type questions have gained popularity due to their convenient evaluation process. Most of the competitive exams have changed their format of questions to M.C.Q. type only and prefer not to include broad-answer type questions due to its hectic evaluation process. With automation, this problem could be solved and thus, broad-answer type questions could also be included in the paper. The evaluation process will be less erroneous and more unbiased if it is automatic. Also, it will be less time-consuming and more efficient.

Requirements

The proposed system uses document vectors, which are trained on a carefully curated data corpus pertinent to the subject matter of the examination paper. This corpus may comprise textbooks or scholarly articles available in PDF format, ensuring relevance and coherence with the examination topics. Moreover, a diverse collection of questions and corresponding answers is indispensable for effective system operation. A comprehensive set of answers enhances system performance, enabling robust analysis across a broad spectrum of answer scripts.

In the development of the system, Python programming language was used. Fundamental Python packages such as NumPy, Scipy, and Matplotlib formed the core of the implementation. Additionally, advanced libraries including NLTK (Natural Language Toolkit), Gensim, and Scikit-Learn were employed to perform specialized tasks. Natural Language Toolkit (NLTK) is a Python library that offers different functionalities for processing texts in the English language. It is used to perform tasks like Stemming, Lemmatization, Stop-word removal, Tagging, Parsing, etc. which are considered fundamental natural language processing tasks. Gensim is another library based on Python and Cython that provides different unsupervised learning models and natural language processing functionalities. Gensim includes models for vectorization of documents like word2vec and doc2vec which are used in our experiment. Scikit-learn is the most popular machine-learning library in Python, across the globe. It offers both supervised and unsupervised learning models to use in the program.

During the training phase, some '.rtf' files were utilized necessitating the inclusion of the striprtf library. However, this dependency is redundant if standard text files are employed instead of '.rtf' files.

Aim of the Project

As previously outlined, the primary objective of this project is to develop a system capable of evaluating answer scripts encompassing more than a few lines, to predict a grade for each answer. The ultimate goal is to automate the entire process of answer script evaluation. However, the realization of this objective is impeded by certain limitations inherent to the system. We have divided this system into two major parts. The first part focuses on the conversion of answers written on hard copies (papers) into digital formats, where the implementation of Computer Vision techniques for handwriting recognition is necessary. Despite extensive exploration of available technologies, the current state-of-the-art in handwriting recognition is not sufficiently advanced to accurately decipher handwritten text. Thus, we decided to continue with the other part, which leverages Natural Language Processing techniques. In this article, we are going to discuss this portion elaborately.

This project aims to find a way to automate the evaluation process. To achieve this, the answers are processed through different natural language techniques. After that, they are being vectorized. And finally, they are put into clusters. The analysis of the answers is done based on this clustering.

Algorithm

Training Algorithm -

Step 1: Extract the text from the document.
Step 2: Tokenize the data.
Step 3: Preprocess the data -
   a. Remove whitespaces.
   b. Remove any digit in between square brackets.
   c. Remove Stop words.
Step 4: Tag the data.
Step 5: Train a model using the tagged data.
Step 6: Store the trained model.
Step 7: Stop.

Vectorization Algorithm -

Step 1: Extract the text from the original answer script and the sample answer script.
Step 2: Tokenize the data.
Step 3: Preprocess the data -
   a. Remove whitespaces.
   b. Remove any digit in between square brackets.
   c. Remove Stop words.
Step 4: Vectorize the text using the pre-trained model.
Step 5: Store the vectors.
Step 6: Stop.

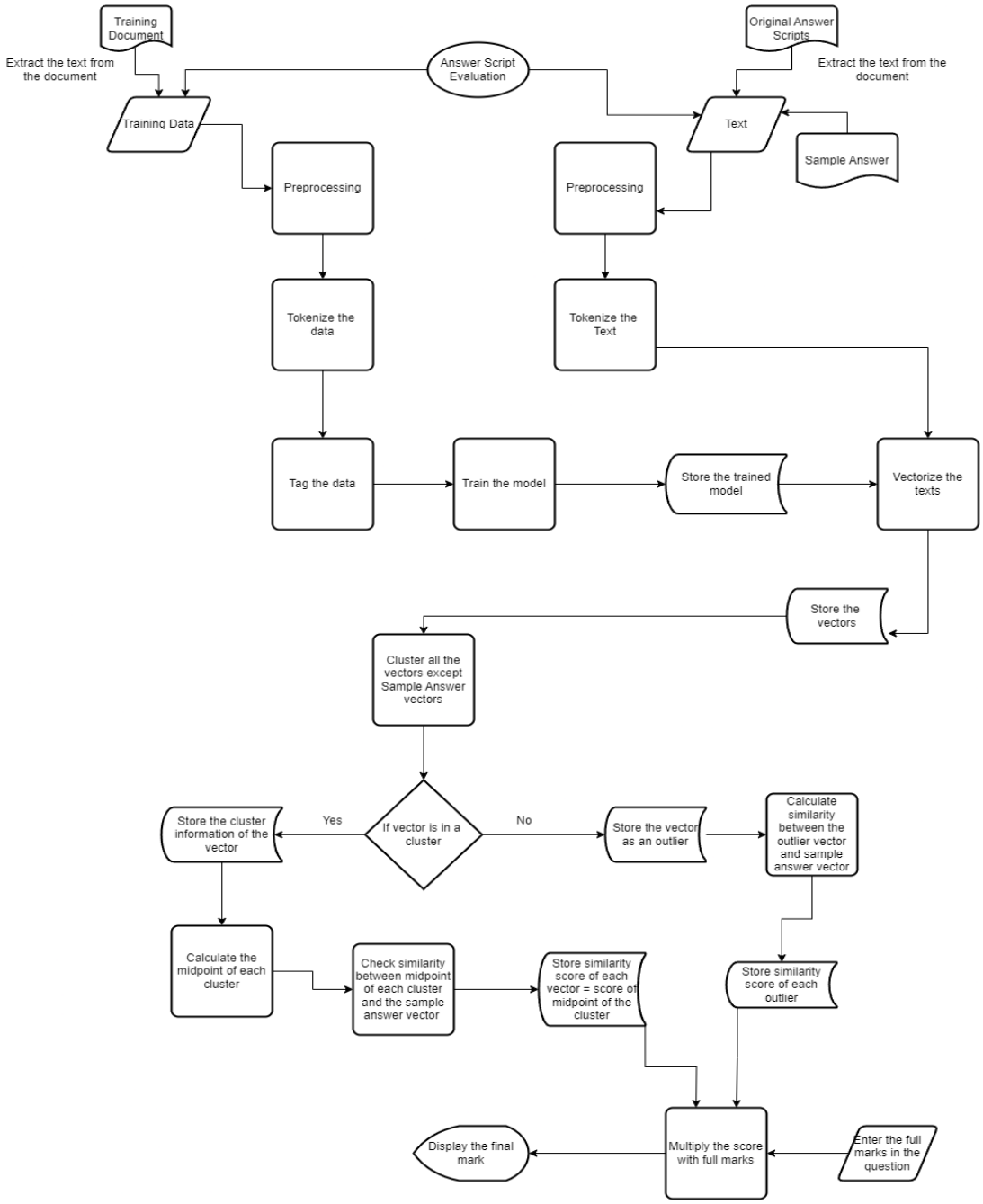
Clustering Algorithm -

Step 1: Cluster all the vectors except the Sample answer vector.
Step 2: If Vector is in a Cluster :
        Store the Cluster number and the vector;
        Go to Step 3;
    Else :
        Store the vector as an Outlier;
        Go to Step 7;
Step 3: For n in number(clusters):
        Get the midpoint of n;
Step 4: Calculate the similarity between the midpoint and the sample answer vector.
Step 5: For n in number(clusters) :
        For i in points(n) :
            Score of i = Score of midpoint(n);
Step 6: Store the similarity scores in Scores.
        Go to Step 9;
Step 7: For n in number(outliers) :
        Calculate the similarity score of n with the sample answer vector.
Step 8: Go to Step 6.
Step 9: Stop.

Final marking Algorithm -

Step 1: Enter full marks F for the question.
Step 2: For j in Scores :
Final_marks = j * F;
Step 3: Display the Final_marks.

Flowchart

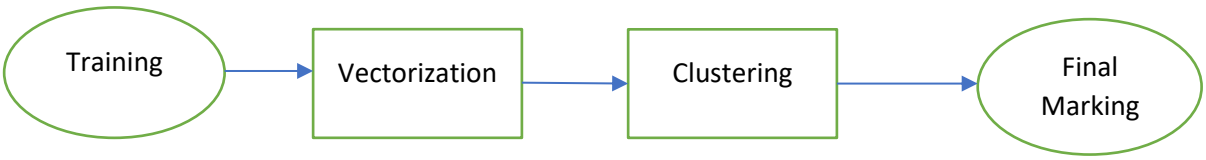


Discussion

The automation of Answer script evaluation is a critical task. It has quite a lot of challenges like understanding the literature of every answer, processing the idea behind the answer, and evaluating the correctness of that idea. In this article, we are going to discuss a method by which we can overcome these challenges and can automate the task.

This technique consists of 4 major sections. These are -

1. Training
2. Vectorization
3. Clustering
4. Final Marking



1. Training

In Machine Learning, Training is a pivotal phase. It requires additional care as it impacts on the outcome of the process. The processing of data, the selection of hyperparameters, and analysis of error - these are all integral parts of efficient training of a machine learning model. In this article, we have discussed an Unsupervised learning technique called Word-Embeddings. This technique requires training with tagged text data that will put every word in a multi-dimensional vector space. This technique helps to determine how close two words are, in terms of semantic meaning. Thus for training, we require a text corpus that will have relevance to the topic of the exam. This corpus could be supplied by any textbook or research paper on the topic.

For training purposes, the book has to be segmented into several text files. Now, a complete text consists of different unnecessary things that are used only for literature purposes and have little to do with the analysis of text. Adding these to the training data could lead to biased and incorrect conclusions. To avoid this, the text needs to be processed first. Several natural language techniques offer efficient preprocessing of data. As part of preprocessing the whitespaces in the text are removed. Then any digit within square brackets is removed. Any sequence of numbers of digits is also removed. And finally, stop words are removed.

Stop words are some extremely common words that have little value in expressing the significance of certain text. Some examples of Stop Words are - 'a', 'an', 'is', 'of', 'on' etc. Removing these words will not affect the training process. After the preprocessing is done, the tokenization is applied to the text. Tokenization is a technique where documents are chopped into pieces, which are called Tokens. In this process, the punctuations are also removed. After the tokenization is done, the text is ready to go for training.

To train the model, we need tagged text. Document tagging is a process where each document is represented with a numeric value, called Tag. Once the tagging is done, we could train the embedding model using the tagged data.

2. Vectorization

In the post-training phase, the trained model is implemented on actual data. The original answer scripts are passed in text format as input. Additionally, a sample answer to the questions is needed for evaluation in the final phase. This sample answer file is also included as input at this stage. After the files are inputted, they are to be passed through processing. We will perform the same preprocessing steps as we did in the training phase:

1. Remove one or more whitespaces.
2. Remove any digit within square brackets.
3. Remove any sequence of numbers or digits.
4. Remove the stop words.
5. Tokenize the text.

Upon completion of preprocessing, the text is prepared for vectorization. The cleaned text is passed through the pre-trained model, enabling the conversion of text data into vector representations. This vectorization process facilitates subsequent analysis and evaluation of the answer scripts, ultimately contributing to the automated answer script evaluation system's effectiveness and accuracy.

3. Clustering

In the preceding step, we obtained document vectors representing numerical representations of each document. Now to determine the proximity of these vectors, we will perform clustering on them. The idea behind this clustering is to divide all the answers into a few categories so that we can evaluate the categories to determine the quality of each answer. The clustering step is quite simple. By choosing a clustering algorithm, pass all the vectors, except the sample answer vector, through it. If a vector is in a cluster, store the vector and its corresponding cluster-ID in a separate space. If the vector is not in a cluster, it would be classified as an outlier and will be stored in a different place. We do not pass the sample answer vector in the clustering model, it will be used for the final evaluation.

After classifying all the vectors correctly between cluster member and outlier, we will proceed to the subsequent step. For the clustered vectors, we will compute the midpoints of all the clusters. Then the similarity between the midpoints and the sample answer vector is calculated. Finally, the same score for each cluster midpoint is assigned to every member of the corresponding cluster.

In the case of outliers, we directly calculate the similarity of each one with the sample answer vector. Here, we assume that the number of outliers is lesser so that the computation takes less time. However, if the number of outliers is substantial, it may necessitate longer computation times and could potentially lead to different conclusions. Possible implications include inefficient training or the need to fine-tune clustering hyperparameters.

By systematically applying clustering and similarity evaluation techniques, we aim to enhance the accuracy and efficiency of the automated answer script evaluation process.

4. Final Evaluation

In the final evaluation step, we take the highest mark of the question as input. The similarity score that we obtained in the previous section denotes the similarity between the sample answer and the corresponding original answer. That way, the similarity score serves as a metric indicating the correctness of the specific answer. We can assume it as a percentage of how much the answer is correct.

In the final evaluation, we leverage this similarity score by multiplying it with the full marks designated for the question. This calculation produces the final marks awarded to the answer, encapsulating both its accuracy and alignment with expected standards. By employing this approach, we aim to provide a comprehensive and equitable assessment of each answer's merit, facilitating meaningful feedback to students and enhancing the effectiveness of automated answer script evaluation.

Experimental Result

Various types of vector embeddings were used in this evaluation process to determine the best possible way for marking. The experiments are elaborated below.

In Natural Language Processing, words or collections of words are represented by vectors to make sense to the computer. There are various techniques for creating these embeddings. In this Word2Vec, Doc2Vec, Bert, etc.

BERT (Bidirectional Encoder Representations from Transformers) is a comparatively newer NLP model developed by Google researchers. It uses transformers which have two main mechanisms, encoding and decoding. It learns relationships between words from a huge corpus of data. In our project, we pre-trained base Bert model and SciBert(Bert model trained on a huge corpus of scientific writings). Bert model can also be used to generate sentence embeddings. The embeddings are then used to calculate similarity. This was done with both aforementioned models of Bert but the result was not satisfactory. The dataset contained a range of answers, some good and some bad, but the cosine similarity from Bert gave a similarity range between 80-90% (80% for base pre-trained Bert and minimum 83% for SciBert).

Word2Vec is one of the famous word embedding techniques in the field of Natural language Processing. It uses a neural network to generate a vector space of a certain dimension, where each word is represented by a vector in the space. In the vector space, words used in similar contexts are closer than other words. Also, the semantic meaning is preserved in the model. As a famous example, in a well-trained model, if one subtracts the vector ‘man’ from that of ‘king’ and adds the vector of ‘woman’, the resultant vector should represent ‘queen’. WMD or Word Mover’s Distance(NLP implementation of Earth Mover’s Distance) can be used to calculate the distance between two sentences efficiently but in the context of evaluating answers of multiple lines where even the number of lines, therefore number of words, can differ from person to person. Moreover,

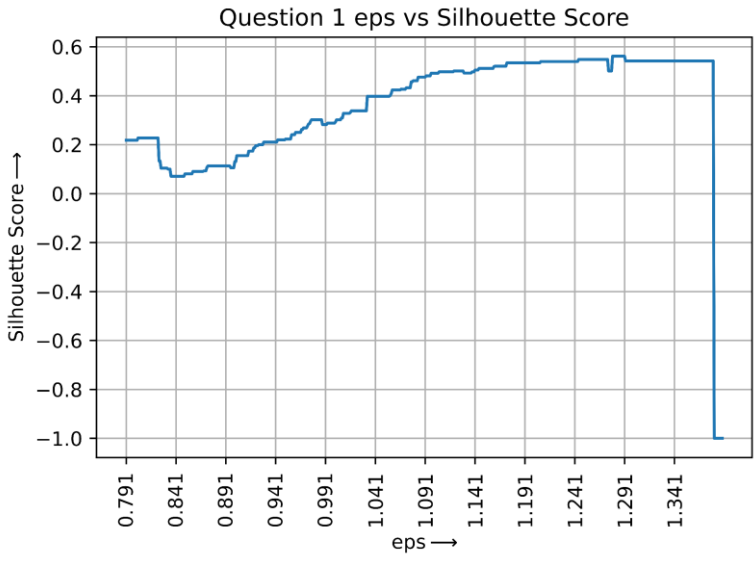
it is hard to map distance with similarity. For these reasons, Word2Vec was not used in this evaluation approach.

Instead, we use **Doc2Vec**, an implementation of the previously mentioned **Word2vec** model. In Doc2Vec, instead of word embeddings, document embedding is used. Doc2Vec captures what kind of words are normally used together and determines vectors for each document(here, answer script). As Doc2Vec is an implementation of Word2Vec, it also has two variances. One is a Paragraph Vector With A Distributed Bag Of Words (PVDBOW), parallel to a Continuous Bag Of Words(CBOW) in Word2Vec and a Distributed Memory Model Of Paragraph Vectors (PV-DM), similar to skip-gram in Word2Vec. Here, PV-DM is used because of the small size of the dataset. As Doc2Vec determines a vector for each answer document, we can use Cosine Similarity to determine the similarity between an answer script and the ideal answer script for a certain question. Cosine Similarity is the cosine value of the angle between two vectors. It is an implementation of the concept of vector dot product. The formula for Cosine Similarity is given by

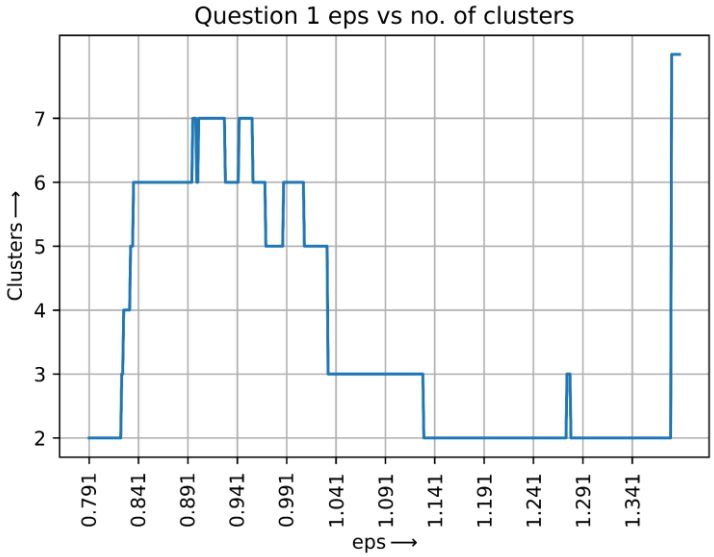
Cosine Similarity = (A . B)/(||A|| * ||B||)

where A and B are two vectors. The value of Cosine Similarity ranges between -1 and 1. Values near 1 mean they are the same in meaning, and 0 means they have little to no similarity. Values near -1 mean they have opposite meanings. For the Data Mining Dataset, a Doc2Vec model with a dimension of 100 is trained after preprocessing. After training the Doc2Vec model, the answer scripts are clustered.

There exist many different kinds of clustering algorithms, like K-means, K-medians, DBSCAN, etc. If we know the number of clusters, K-means is a good choice, but in this context, the number of clusters cannot be determined beforehand without trial and error. This can be countered with the DBSCAN clustering algorithm because it works in the general case. The DBSCAN algorithm has two hyperparameters, eps which is the radius of the neighborhood of a certain point, and MinPts which is the minimum number of necessary points in eps radius to make a cluster. MinPts is taken as 3 to maximize the number of possible clusters, the value of eps is calculated from plotted graphs as they vary from question to question. The graphs are depicted below.

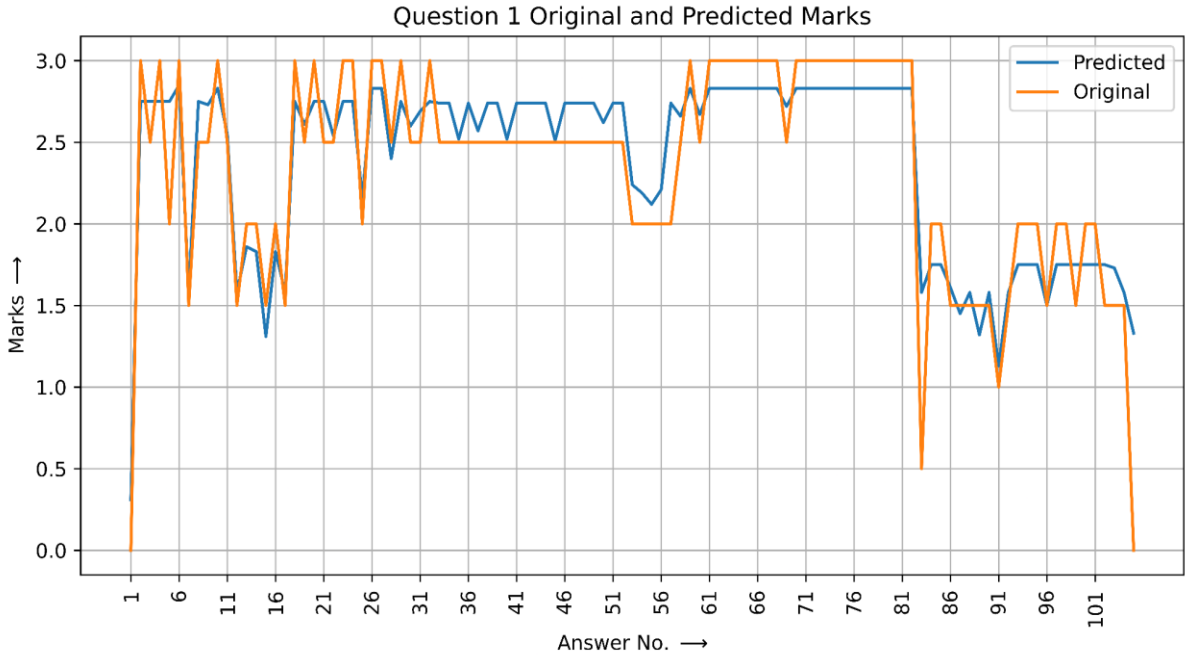


silhouette score is steadily increasing with increasing EPS value, but there is a sudden dip. This is because the Silhouette score cannot be calculated when the number of clusters is 0 or 1, so those cases are explicitly handled and the score is made -1.



From the graph, it is observed that the value of eps is optimal in the range 0.841-0.991.

After clustering with optimal cluster count and Silhouette Score, marks are assigned to each cluster. The scripts not belonging to any clusters (Outliers), are handled separately.



In this approach, 5 different questions are considered for evaluation. The above graph is the depiction of a comparison between original marking by teachers and marks prediction by our model. There were 105 different answers for question number 1, and they are plotted serially. The quality of answers ranged from bad to good, and predicted marks were also similar with some rare exceptions.

This scheme rarely outputs low marks. Generally marking is done with a gap of 0.5, for example, in a question of full marks 3 (just like the graph above), the possible marks are 0, 0.5, 1, 1.5, 2, 2.5, and 3. But in the case of marking by a computer program, the possibilities are virtually endless. That is why there are many fluctuations in the predicted graph compared to the original marks. But the overall flow of the graph is very similar. We calculated the mean squared error by subtracting the original marks from the predicted marks, squaring it, adding this value for each answer script, and dividing it by the number of scripts. The calculated Mean Squared Error for question 1 is 0.07454 which is a good score.

Limitations

The system demonstrates efficiency in evaluating broad-answer type questions. However, the system has certain limitations which may obstruct implementation in real-time scenarios.

As we have discussed earlier, the lack of an efficient Handwritten text recognizer (HTR) is a problem that is restricting this system's ability to achieve its goals. As of now, the system only performs the text analysis and grading tasks. With a powerful HTR, this system could automate the whole process. It could read the answer scripts from any scanner or similar device and then pass through the rest of the process, generating marks for every answer.

Due to the lack of efficient handwriting recognition technology, it is difficult to get a question-and-answer set that could be used in this system. Thus, the answers have to be typed in manually from the original copies to be used in the system. We believe that with the emergence of an efficient handwriting recognition system, this problem could also be countered. However, in any computerized exam where broad-answer types of questions are asked, this system would work perfectly fine.

Poor performance of Mathematical expressions. Any type of mathematical expression is removed from the text during the processing steps. This will lead to the inability of any computation-related task. Broad mathematical answers are not written serially and often contain a lot of white spaces. A system might face difficulties in handling those white spaces and understanding the sequence of steps in the sum and will end up evaluating wrongly. For these drawbacks, we have omitted the scope of evaluating any mathematical question by the system.

This system works using document vectors which take every document and plot it in a multi-dimensional space as a vector. Document vectors are based on word embeddings where every word carries a value. So, if anyone writes nothing but some specific words in an answer, it will still assign

some marks for that answer, even if that is not correct. Though the amount of marks will be very low, it could not be negligible.

These limitations underscore the need for research and development to enhance the system's capabilities and address its shortcomings. Despite these challenges, the system offers significant potential for automating the evaluation of broad-answer type questions in various educational settings.

Future Scope

The system discussed here has showcased great results. However, the method used here is quite elementary. It has a lot of future scope. Currently, the system does not work on mathematical expressions due to certain limitations, discussed in the previous section. However, this problem could be solved using more complex techniques or procedures. The removal of white spaces, and maintaining the correct sequence of steps in a mathematics answer will lead to more efficient handling of sums.

With the advancement in handwritten text recognition technology, the system will be able to automate the task entirely. An effective HTR will read the text from any image or PDF file, uploaded by using a scanner or similar device.

Additionally, there is scope for enhancing the accuracy of the system through the implementation of various preprocessing techniques and the utilization of more efficient word embeddings. By optimizing these components, the system could achieve greater accuracy in evaluating answer scripts and providing meaningful feedback to students.

In summary, the future scope of this research project encompasses the refinement of mathematical expression handling, leveraging advancements in HTR technology for automation, and improving overall accuracy through enhanced preprocessing techniques and word embeddings. These advancements have the potential to significantly enhance the effectiveness and usability of the automated answer script evaluation system in educational settings.

Bibliography

1. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pretraining of deep bidirectional transformers for language understanding, 2019.
2. Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: Pretrained language model for scientific text. In EMNLP, 2019.
3. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

4. Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents, 2014.
5. Md Rahman and Fazlul Siddiqui. Nlp-based automatic answer script evaluation. 4:35–42, 12 2018.