Homework Assignment 3
Question Answering via Reading Comprehension

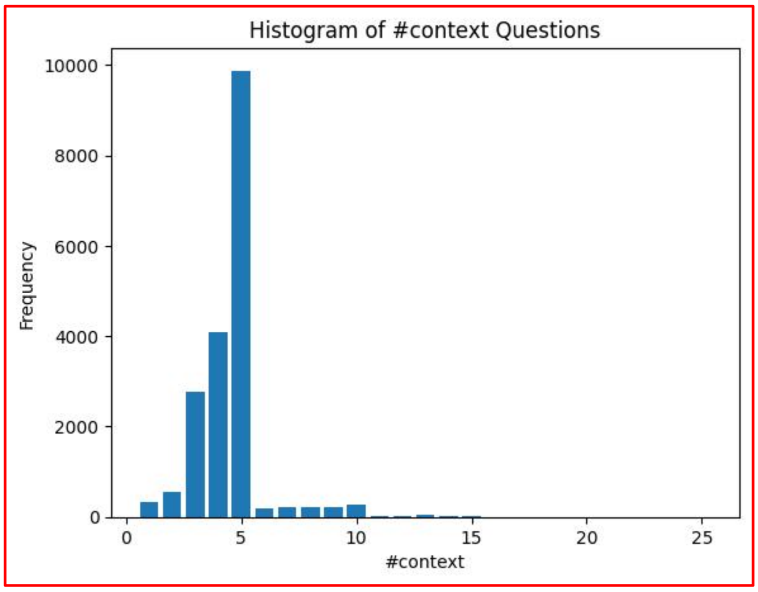
Task 1: Implement a question-answering model

Data Processing

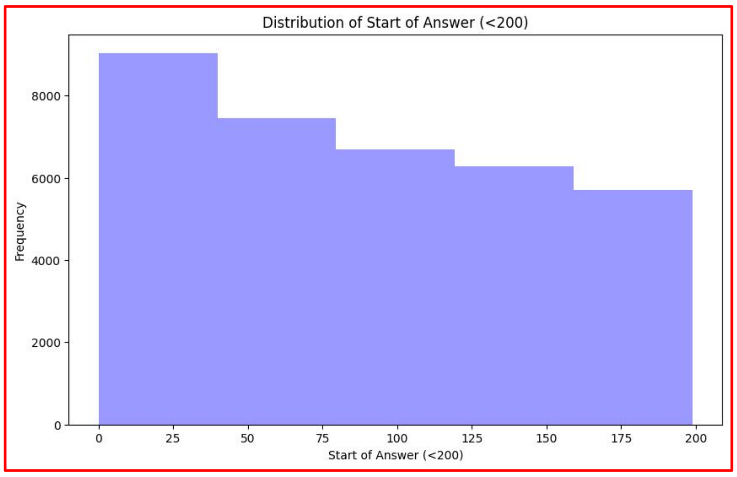
After loading and converting the JSON to a DataFrame, the next step involved verifying that all the questions were answerable using the "is_impossible" tag. Since all questions were answerable, the dataset was filtered to create a smaller subset.

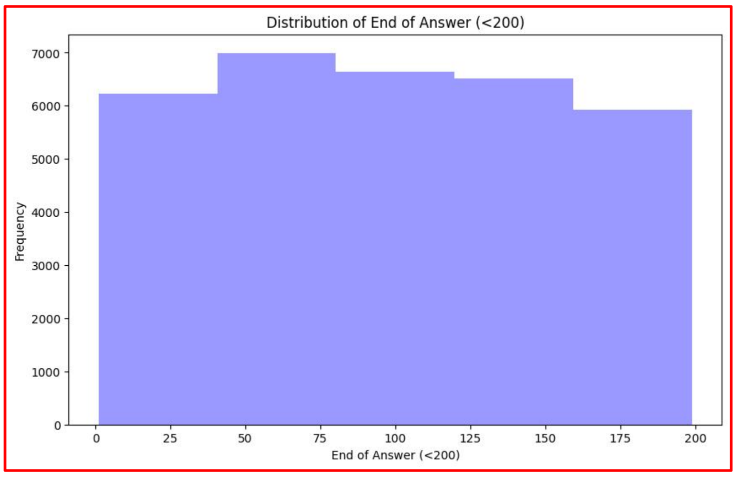
The steps taken were as follows:

1. Plotting a histogram to visualize the total number of questions posed for each content. This helped determine the cutoff to filter the data, with most content having a minimum of 5 questions.

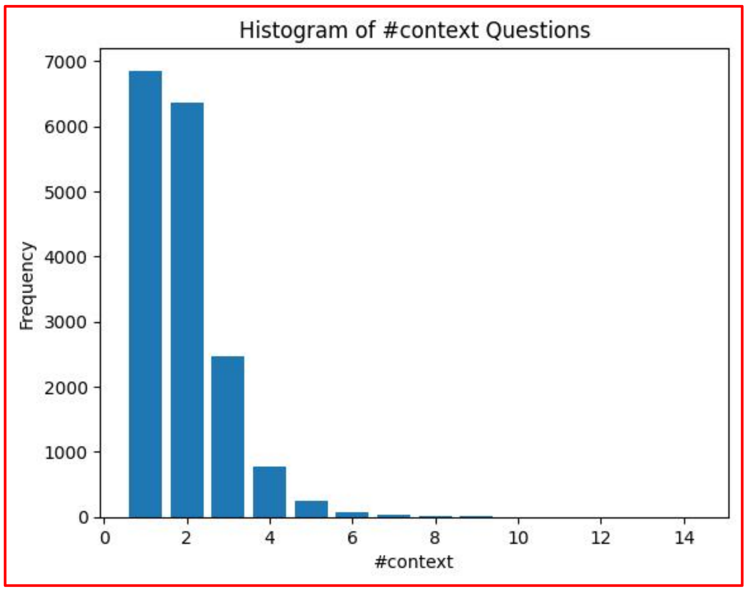


2. Since the plan was to build a deep learning model, it was necessary to determine the maximum sequence length for padding the sequences. This was done by plotting the answer start and end indices, with a maximum cap of 200 characters chosen based on the data.

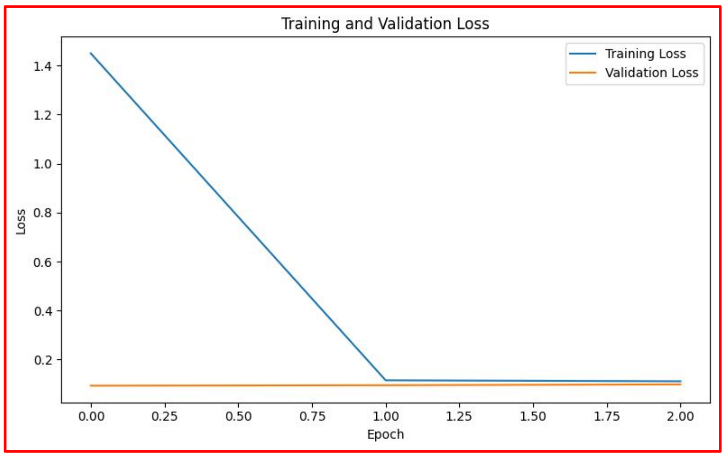


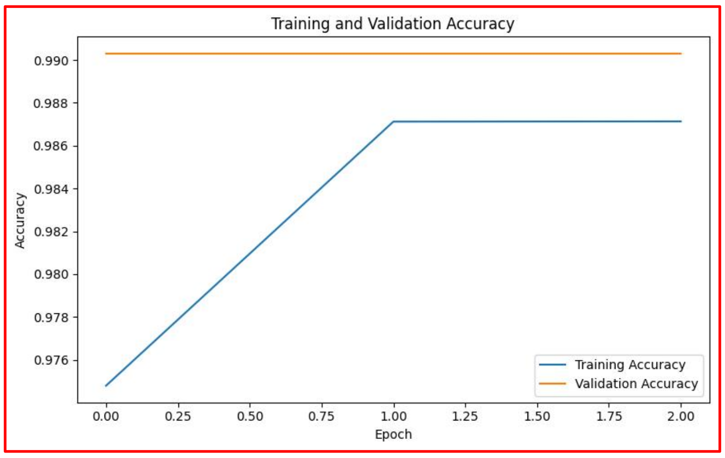


3. The dataset was further filtered to include only records where the answer start and end indices were less than 201 characters, and the content was retained only up to 200 characters. Another histogram was plotted to check the total number of questions posed for each content, and the data was filtered based on this criteria. The top 900 unique content were chosen, and 3 questions were selected from each content, with 2 questions used for training and 1 for testing due to resource limitations.

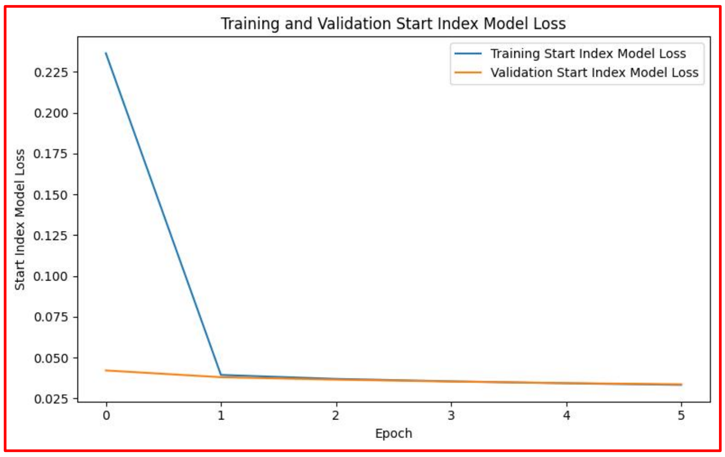


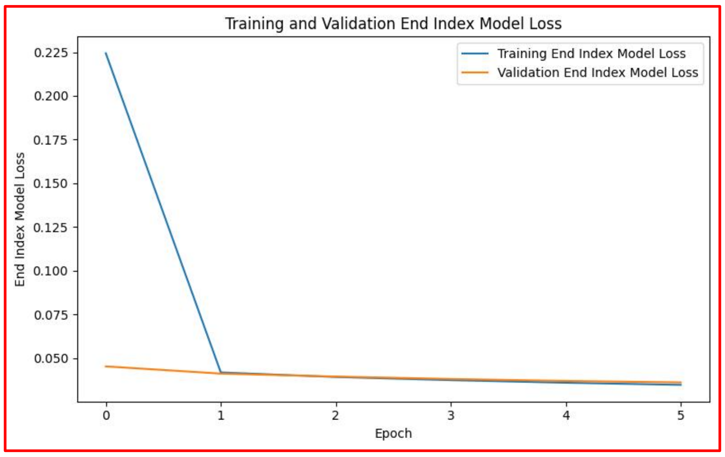
The BiLSTM model trained from scratch to predict the start and end tokens performed poorly, with an F1 metric almost at 0.



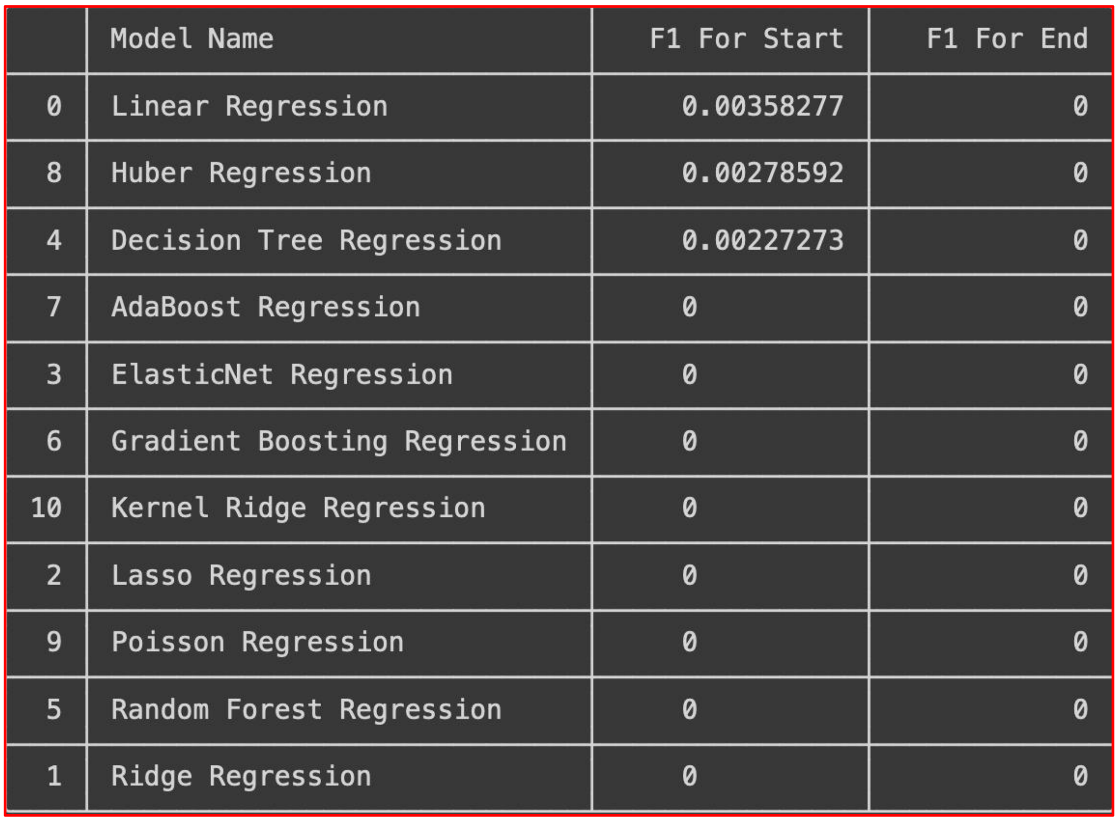


Subsequently, two BiLSTM models were trained separately to predict only the start and end indices, followed by post-processing to extract the text. However, these models also performed poorly on the F1 metric.



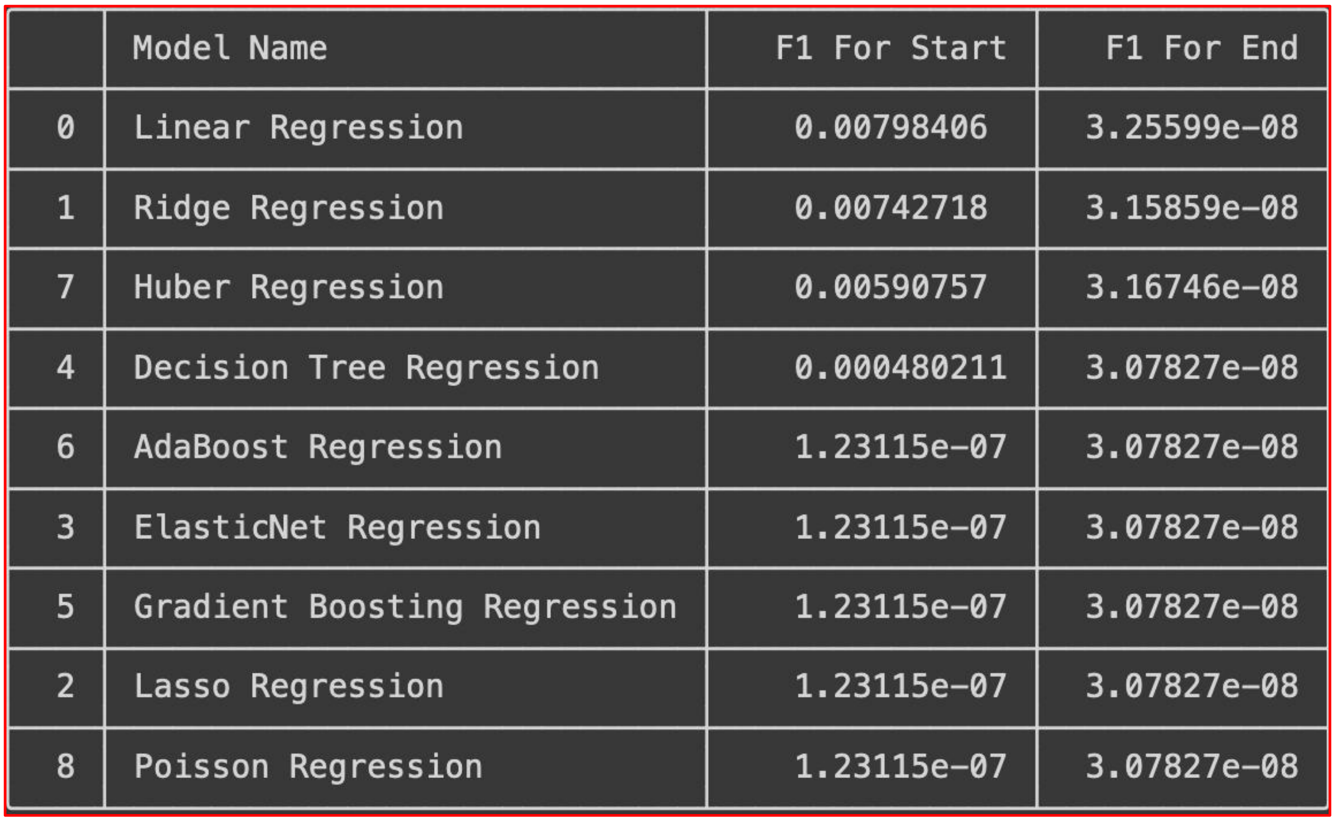


Traditional machine learning algorithms, including Linear Regression, Huber Regression, Decision Tree Regression, AdaBoost Regression, ElasticNet Regression, Gradient Boosting Regression, Kernel Ridge Regression, Lasso Regression, Poisson Regression, Random Forest Regression, and Ridge Regression, were implemented to predict the start and end tokens separately. Features were extracted using TF-IDF, and the results were printed using tabulate, but the F1 scores remained poor.



	Model Name	F1 For Start	F1 For End
0	Linear Regression	0.00358277	0
8	Huber Regression	0.00278592	0
4	Decision Tree Regression	0.00227273	0
7	AdaBoost Regression	0	0
3	ElasticNet Regression	0	0
6	Gradient Boosting Regression	0	0
10	Kernel Ridge Regression	0	0
2	Lasso Regression	0	0
9	Poisson Regression	0	0
5	Random Forest Regression	0	0
1	Ridge Regression	0	0

Finally, the entire dataset was filtered to include 6 questions for each content, with 1 question used for testing and the rest for training. Except for Random Forest Regression and Kernel Ridge Regression, all models were used to train and test. All results are shown below.



	Model Name	F1 For Start	F1 For End
0	Linear Regression	0.00798406	3.25599e-08
1	Ridge Regression	0.00742718	3.15859e-08
7	Huber Regression	0.00590757	3.16746e-08
4	Decision Tree Regression	0.000480211	3.07827e-08
6	AdaBoost Regression	1.23115e-07	3.07827e-08
3	ElasticNet Regression	1.23115e-07	3.07827e-08
5	Gradient Boosting Regression	1.23115e-07	3.07827e-08
2	Lasso Regression	1.23115e-07	3.07827e-08
8	Poisson Regression	1.23115e-07	3.07827e-08

Task 2: Unanswerable question detection

Hypothesis: I hypothesize that unanswerable questions can be detected effectively by leveraging contextual information and linguistic patterns in the question and reference text. By training a binary classification model to distinguish between answerable and unanswerable questions, we can improve the overall performance and reliability of question-answering systems.

Motivation: The motivation behind detecting unanswerable questions is to enhance the accuracy and robustness of question-answering systems. In real-world scenarios, not all questions have clear answers, and providing incorrect or misleading answers can negatively impact user experience and trust in the system. By identifying unanswerable questions, we can avoid providing erroneous responses and instead indicate when a question cannot be answered with the available information.

Model Design:

The proposed model design comprises several crucial components:

1. *Pre-Processing:* Data undergoes pre-processing to encode questions and reference text into numerical representations. This involves utilizing pre-trained language models (e.g., BERT, GPT), word2vec models, GloVe embeddings, or training a new word2vec model from scratch.

2. *Feature Extraction:* Contextual features are extracted from the encoded question-reference text pairs to capture relevant information essential for determining answerability.

3. *Binary Classification:* A binary classification model, such as logistic regression, support vector machine, or neural network classifier, is employed to predict whether a question is answerable or unanswerable based on the extracted features.

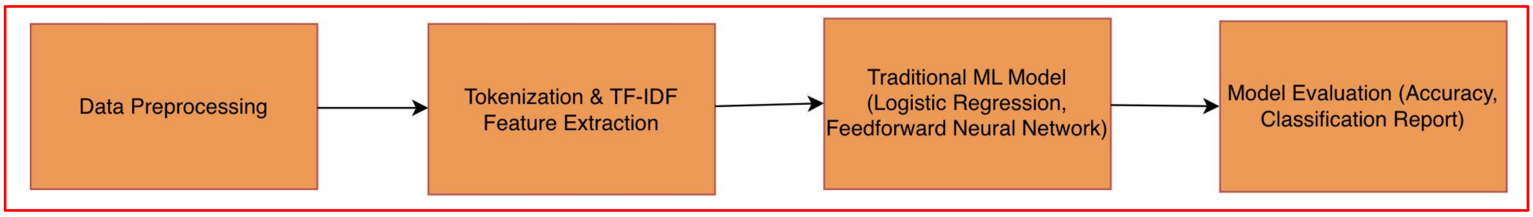
4. *Fine-tuning:* The pre-trained language model and classification layers are fine-tuned on a labeled dataset of answerable and unanswerable questions to optimize performance for the specific task.

5. *Evaluation:* The model's performance is evaluated using metrics such as accuracy, precision, recall, and F1-score to assess its ability to correctly classify answerable and unanswerable questions.

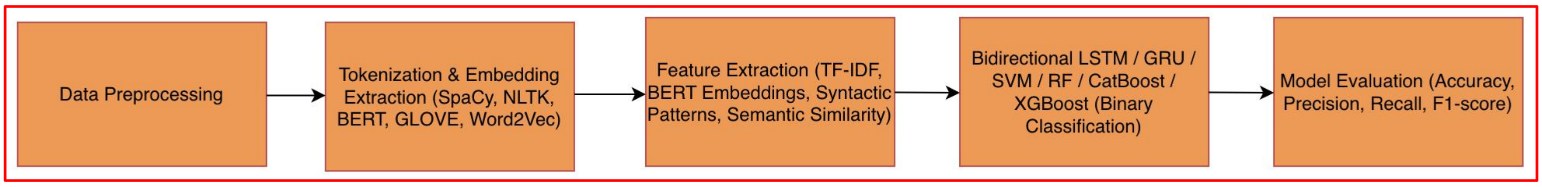
Key Parameter Settings:
- *Choice of pre-trained language model:* Options include BERT, GPT, RoBERTa, among others.
- *Choice of traditional machine learning models:* Consider logistic regression, support vector machine, random forest, CatBoost, XGBoost, etc.
- *Hyperparameters for fine-tuning:* Parameters to tune include learning rate, batch size, number of epochs, dropout rate, etc.
- *Size and architecture of the classification model for pre-trained language model:* Factors to consider include the number of hidden layers, activation functions, optimizer selection, etc.

Task 3: Unanswerable question detection

In ***Part 1*** of our approach to tackling the issue of "Unanswerable question detection," we adopted a traditional machine learning methodology. This involved initial data acquisition and preprocessing steps, where we imported essential libraries and read the dataset containing questions and contexts. We then proceeded with feature engineering, where both tokenized sequences and TF-IDF features were utilized to represent the textual data. These features were extracted using techniques provided by TensorFlow/Keras and scikit-learn. Subsequently, we trained two models for classification: a logistic regression model and a feedforward neural network (multi-layer perceptron). The logistic regression model leveraged the traditional machine learning approach to classify questions as answerable or unanswerable based on the engineered features. On the other hand, the neural network model, built using TensorFlow/Keras, utilized a deep learning architecture to perform the same classification task. Both models were evaluated using metrics such as accuracy and classification reports to assess their performance in detecting unanswerable questions.

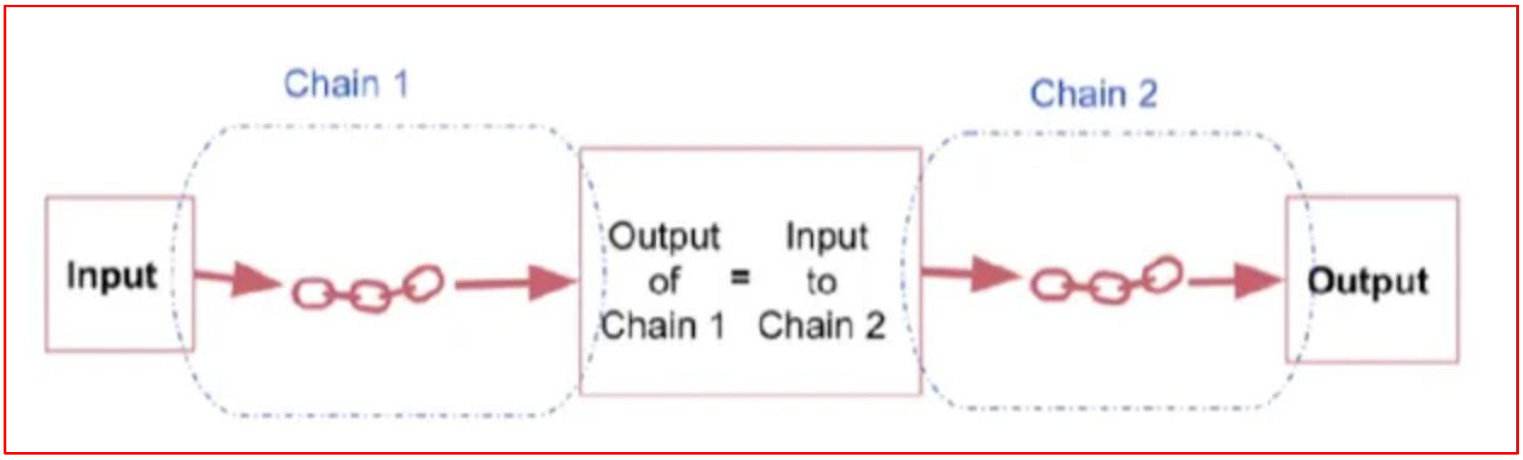

Data Preprocessing
Tokenization & TF-IDF Feature Extraction
Traditional ML Model (Logistic Regression, Feedforward Neural Network)
Model Evaluation (Accuracy, Classification Report)

In ***Part 2***, we adopted a more advanced approach leveraging advanced natural language processing (NLP) techniques and deep learning methodologies. This involved additional data processing steps where we incorporated libraries such as spaCy, NLTK, and Transformers. We also utilized the BERT tokenizer and model for tokenization and generating contextual embeddings. Feature extraction and representation were performed using a combination of techniques, including tokenization using spaCy and NLTK, TF-IDF feature extraction, and BERT embeddings. These extracted features, along with syntactic patterns and semantic similarity scores, were combined into a comprehensive feature matrix. Subsequently, a deep learning model based on a bidirectional LSTM architecture was constructed using TensorFlow/Keras. This model was trained using the combined feature matrix to classify questions as answerable or unanswerable. The performance of the trained model was evaluated using appropriate metrics to gauge its effectiveness in detecting unanswerable questions. Overall, the approach in Part 2 aimed to leverage both traditional and advanced NLP techniques to enhance the accuracy and robustness of unanswerable question detection.


Data Preprocessing
Tokenization & Embedding Extraction (SpaCy, NLTK, BERT, GLOVE, Word2Vec)
Feature Extraction (TF-IDF, BERT Embeddings, Syntactic Patterns, Semantic Similarity)
Bidirectional LSTM / GRU / SVM / RF / CatBoost / XGBoost (Binary Classification)
Model Evaluation (Accuracy, Precision, Recall, F1-score)

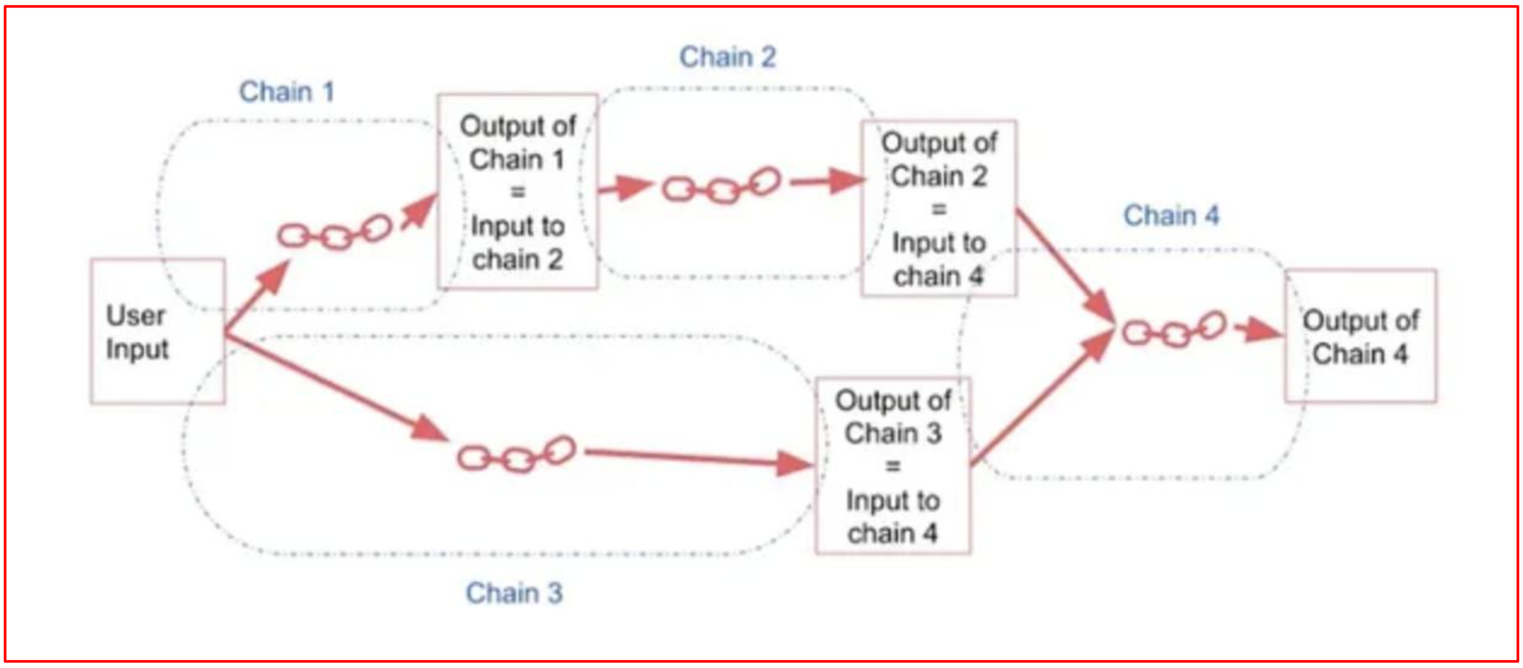
Reference URL: https://online.visual-paradigm.com/app/diagrams/#diagram:proj=0&type=DecisionTree&width=11&height=8.5&unit=inch

Task 4: LangChain practice



Reference URL: https://www.analyticsvidhya.com/blog/2023/10/a-comprehensive-guide-to-using-chains-in-langchain/

A sequential chain is a process that links multiple chains together, where the output of one chain feeds into the input of the next one in a continuous sequence. It functions by executing a series of chains consecutively. There are two types: the Simple Sequential Chain, which handles one input and output, and the Sequential Chain, which manages several inputs and outputs at once. This type of chain combines various chains by using the output of one as the input for the next. It's useful for maintaining a smooth flow of processes, especially when you need to use the result of one operation as the starting point for the next. In its most basic form, a sequential chain consists of steps where each step takes one input and generates one output. This ensures a seamless and uninterrupted flow of information, with each step smoothly passing its output to the next step. This straightforward approach is particularly effective when dealing with sub-chains designed for singular inputs and outputs, facilitating the continuous transfer of data between each step.


Chain 1
Chain 2
Chain 3
Chain 4
User Input
Output of Chain 1 = Input to chain 2
Output of Chain 2 = Input to chain 4
Output of Chain 3 = Input to chain 4
Output of Chain 4

Reference URL: https://www.analyticsvidhya.com/blog/2023/10/a-comprehensive-guide-to-using-chains-in-langchain/

Sequential chains vary in complexity and may not always involve single inputs and outputs. In more complex setups, these chains manage multiple inputs and produce multiple outputs. It's important to name input and output variables carefully in these complex chains. Additionally, a more general form of sequential chains permits multiple inputs and outputs, allowing any step in the chain to accept multiple inputs simultaneously. This flexibility enables the chain to handle a wider range of scenarios and data types, making it more versatile and useful in different situations. Overall, sequential chains play an important role in coordinating a series of connected operations, ensuring that data and processes flow smoothly in complex systems.