**3. Baseline Comparison**

**3.1 Loading Data and Initial Exploration**

The first step in our baseline approach is to load and explore the dataset. Using the `load\_data()` function, we retrieve the Stanford Natural Language Inference (SNLI) dataset, which consists of sentence pairs labeled as `entailment`, `neutral`, or `contradiction`. The `visualise\_data()` function provides a detailed overview of the dataset, including the number of pairs, the average length of premises and hypotheses, and the distribution of labels. This exploration helps us understand the dataset's structure and characteristics, which is crucial for selecting appropriate models and preprocessing techniques.

**3.2 Preparing Data for Classical Models**

To train classical machine learning models, we need to prepare the data in a suitable format. We combine each `premise` and `hypothesis` into a single text string using the `combine\_texts()` function. This transformation is applied to the entire dataset, effectively creating a new column with the combined text. We then split the data into training and validation sets using the `train\_test\_split()` function, ensuring that we have separate sets for model training and evaluation. This step is critical for preventing overfitting and assessing the generalization performance of our models.

**3.3 Creating Bag-of-Words Representation**

For classical models, we convert the combined text data into a bag-of-words representation using the `CountVectorizer()`. This method transforms each text into a vector of word counts, where each unique word in the dataset is represented as a feature. The bag-of-words model is a simple yet effective way to represent textual data for machine learning algorithms. It captures the frequency of words but ignores the order and context, making it suitable for models like Naive Bayes and Logistic Regression.

**3.4 Training and Evaluating Classical Models**

We train and evaluate two classical machine learning models: Naive Bayes and Logistic Regression. These models are chosen for their simplicity and efficiency in handling text classification tasks.

Naive Bayes and Logistic Regression are both classical models used for text classification. Naive Bayes is based on Bayes' theorem with a strong assumption of independence between features, making it simple and fast, particularly suited for tasks like spam filtering and sentiment analysis, though its independence assumption is often unrealistic for textual data. Logistic Regression, on the other hand, is a linear classification model that uses a logistic function to estimate the probability of class membership. It is effective for binary and multi-class classification, offers good interpretability, and handles high-dimensional data well, but it may be less effective for non-linear data and requires careful feature selection.

Using the bag-of-words vectors from the training set, we fit each model and then evaluate their performance on the validation set. Predictions are made using the validation data, and accuracy scores are calculated to measure how well each model performs. This step provides a baseline performance metric for classical models, against which we can compare more sophisticated approaches.

**3.5 Training and Evaluating the GloVe Model**

Next, we implement a GloVe (Global Vectors for Word Representation) model to learn word embeddings from the dataset. We create a co-occurrence matrix from the bag-of-words representation and train the GloVe model to generate dense vector representations for each word. These embeddings capture semantic relationships between words, providing a richer representation compared to the bag-of-words model. We then use these embeddings to transform our data and train a Logistic Regression classifier on the resulting vectors. The classifier is evaluated on the validation set, and its accuracy is calculated to assess the effectiveness of the GloVe embeddings.

**3.6 Preparing Data for the LSTM Model**

For the LSTM (Long Short-Term Memory) model, we need to tokenize the text data into sequences of tokens. Using the `DistilBertTokenizer`, we convert the texts into tokenized sequences suitable for input into the LSTM model. We prepare DataLoader objects for both the training and validation sets to facilitate efficient batch processing during model training and evaluation. This preprocessing step is essential for handling sequential data and leveraging the capabilities of LSTM networks in capturing temporal dependencies.

**3.7 Training and Evaluating the LSTM Model**

We initialize the LSTM model with parameters such as input dimension, hidden dimension, output dimension, and the number of layers. The model is trained for a specified number of epochs, during which it learns to classify the sentence pairs based on their tokenized representations. For each batch, the inputs and labels are processed by the model, and the loss is computed and minimized using backpropagation. After training, we evaluate the LSTM model on the validation set by making predictions and calculating the accuracy. This step demonstrates the potential of deep learning models in handling complex text classification tasks.

**3.8 Comparing Performances**

Finally, we compare the performance of all the trained models: Naive Bayes, Logistic Regression, GloVe + Logistic Regression, and LSTM. The accuracy scores of these models are printed and compared to determine which model performs best on the given task of determining the relationship between sentence pairs. This comprehensive comparison provides insights into the effectiveness of different modeling approaches and helps identify the most suitable model for the task at hand.

The performance comparison of the models reveals interesting insights into their effectiveness for text classification. Logistic Regression achieves the highest accuracy at 54.28%, indicating its robustness in capturing relevant features using the bag-of-words representation. Naive Bayes follows with an accuracy of 51.73%, performing slightly worse due to its strong independence assumption, which is less suited for natural language data where word dependencies matter. Surprisingly, the combination of GloVe embeddings with Logistic Regression results in a lower accuracy of 44.68%, suggesting that the semantic relationships captured by GloVe embeddings may not be as beneficial in this context as the simpler bag-of-words approach. The LSTM model, despite its potential for handling complex text classification tasks and capturing temporal dependencies, achieves the lowest accuracy of 33.82% and takes significantly longer to train, indicating that it might not be well-suited for this particular dataset and baseline comparison.