1. C
    a. **CountVectorizer:** CountVectorizer is a technique that converts a collection of text documents into a matrix of token counts. Each document is represented as a vector where each element corresponds to a word in the vocabulary and contains the count of that word's occurrences in the document.
    b. **TF-IDF (Term Frequency-Inverse Document Frequency):** TF-IDF is a numerical representation that reflects the importance of words in a document relative to the entire dataset. It combines term frequency (how frequently a word appears in a document) with inverse document frequency (how unique the word is across all documents).
    c. **GloVe Embeddings (Global Vectors for Word Representation):** GloVe is a word embedding method that maps each word to a fixed-length vector based on word co-occurrence statistics from large text corpora. This results in vectors that capture semantic relationships between words.
    d. **BERT Embeddings (Bidirectional Encoder Representations from Transformers):** BERT is a transformer-based model that generates embeddings for each token by considering the context in which the token appears. BERT captures complex relationships between words in a sentence
2. A
    a. **Model Type**: MLPClassifier (Multi-layer Perceptron) from scikit-learn.
    b. **Hidden Layers**: The model has two hidden layers, each with 128 neurons. This architecture allows the model to capture complex patterns in the data by learning non-linear relationships between input features and target labels.
    c. **Activation Function**: The default activation function for `MLPClassifier` is the ReLU (Rectified Linear Unit), which is used to introduce non-linearity in the model, allowing it to learn complex functions.
    d. **Learning Rate (learning_rate_init)**: The model explores different values of the learning rate, ranging from 0.0001 to 0.1. The learning rate controls how much the model adjusts weights with respect to the gradient during each training step. A grid of candidate learning rates ([0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1]) is tested to find the optimal rate for model convergence and performance.

e. **Maximum Iterations** (`max_iter`): The model is set to train for a maximum of 100 iterations per learning rate and fold in cross-validation. This parameter limits the number of training epochs and prevents excessive computation time.

f. **Random State** (`random_state=42`): A fixed random state ensures reproducibility, meaning that the same splits and model initialization will yield identical results if the code is re-run.

g. **Evaluation Metric**: The accuracy score is used as the metric to evaluate training and validation performance for each learning rate. This provides a straightforward measure of how well the model predicts the correct class.

3. A In this code, I used BERT embeddings as features, which involves the following pre-processing steps:

   a. **Text Preprocessing**: The raw text from each document is passed through the BERT tokenizer, which converts text into a format that BERT can process. This includes converting words to lowercase, tokenizing each word, and padding or truncating sentences to a fixed length (if necessary).

   b. **BERT Embedding Extraction**: For each document, the get_bert_embedding function generates embeddings using a pre-trained BERT model. Specifically, it takes the mean of the last hidden states from BERT, resulting in a fixed-length vector for each document. This vector captures semantic relationships and contextual meaning, which are useful for classification.

   c. **Feature Matrix for Training and Testing**: The BERT embeddings for all training documents are stacked to create X_best, the feature matrix for training. Similarly, embeddings for all testing documents are generated and stacked to form X_bert_test, the feature matrix for testing.

   B **Parameter Settings**:

   d. **Hidden Layers**: The MLP is configured with two hidden layers, each containing 128 neurons. This architecture balances model complexity and computational efficiency.

   e. **Learning Rate (`learning_rate_init=0.001`)**: After experimenting with several candidate learning rates in previous cross-validation tests, a learning rate of `0.001` was found to work well, providing stable convergence without overfitting.

f. **Solver (`solver='adam'`)**: The Adam optimizer is chosen for its adaptive learning rate capabilities and showed high efficiency.
g. **Max Iterations (`max_iter=500`)**: This parameter limits the number of training epochs to prevent excessive computation time while ensuring the model can converge.
h. **Random State (`random_state=42`)**: A fixed random state is used for reproducibility, ensuring consistent results across runs.
i. **Model Choice**:To find the best model, tests were carried out using various learning rates and optimisers.

C

j. The performance of the MLP model on the training data is evaluated using 5-fold cross-validation.
k. **Cross-Validation**: The training dataset (X_best) and labels (y) are split into 5 folds, with each fold taking turns as the validation set while the other four are used for training. This process provides a more reliable estimate of model performance by reducing variance associated with a single train-test split.
l. **Training and Validation Accuracy**: The cross-validation results provide both training accuracy and validation accuracy scores for each fold. The average training and validation accuracies across all folds are then computed to give an overall assessment of the model's performance.