Algorithm Overview

The goal of this algorithm is to predict star ratings based on the provided Amazon movie review text data. This solution combines **TF-IDF vectorization** for feature extraction, **incremental learning with SGDClassifier** for efficient training, and **chunk-based data processing** to handle large-scale datasets and conserve memory. This approach is optimized for efficient performance in memory-limited environments, such as Google Colab.

Solution Steps

Data Preparation:

Load test.csv and extract the Id column to identify the records in train.csv that correspond to the test dataset.

Process train.csv in chunks to locate and collect data matching the Ids from test.csv. By reading in chunks, we avoid loading the entire dataset at once, thus conserving memory.

Combine the Summary and Text fields for each review into a single text field, Combined_Text, which is used for subsequent feature extraction.

TF-IDF Feature Extraction:

Initialize a TfidfVectorizer with a maximum of 10,000 features, setting ngram_range=(1,2) to capture both unigrams and bigrams. This setup achieves a balance between capturing sufficient text context and controlling memory usage.

Fit the TfidfVectorizer on the Combined_Text field of each review, extracting features representing the term frequency-inverse document frequency (TF-IDF) of words in the documents.

Model Training using Incremental Learning:

Use SGDClassifier with log_loss as the loss function for incremental learning. Incremental learning allows the classifier to learn from data in chunks, enabling training on large datasets under memory constraints.

For each chunk from train.csv, sample 30% of the data to reduce data volume and avoid memory overload.

For each sampled data chunk:

Vectorize the Combined_Text field using the previously fitted TfidfVectorizer.

Perform incremental training of the classifier (SGDClassifier) using partial_fit.

Regularly call garbage collection to free up memory.

Prediction:

After training, predict the scores for the test data.

To conserve memory, the test data is also processed in chunks, and predictions for each chunk are stored in a list.

Finally, the predictions are saved in the submission file submission.csv in the required format.

Special Optimizations

Chunk-Based Processing:

Reading the dataset in chunks allows processing large files without loading everything into memory, making the solution scalable and memory-efficient.

Incremental Training with Partial Fit:

Using partial_fit in SGDClassifier allows us to handle large datasets by iterating through data. This is especially useful when single-batch training is impossible due to memory limitations.

Sampling:

To further control memory usage, each data chunk from train.csv is sampled, using only 30% of the data per chunk. This helps reduce computational load and improves training speed, though it may slightly impact accuracy.

Assumptions and Observations

Assumption: Only the Summary and Text columns contain the information needed to predict the star rating. Other metadata columns are omitted to simplify and improve efficiency.

Observation: Using n-grams, especially bigrams, helps capture more context within the text, which can improve model performance without significantly increasing memory usage.

Assumption: Since the TfidfVectorizer vectorizes the text before incremental training, we assume the model can generalize well across chunks without retraining the vectorizer on each chunk. This approach reduces computational complexity.

Conclusion

This approach achieves a balance between memory efficiency and predictive accuracy by combining chunk-based processing, TF-IDF vectorization, and incremental learning. The final model is efficient enough to be trained in limited-memory environments, making it suitable for large-scale text data classification tasks.