Sentiment Analysis for Product Reviews

1. Main Objective

The primary goal of this analysis is to develop and compare deep learning models that classify customer product reviews into three sentiment classes: positive, neutral, or negative. By accurately detecting customer sentiment, businesses can rapidly address issues, enhance overall customer satisfaction, and make data-driven decisions on product improvements, marketing strategies, and support services.

2. Dataset Summary

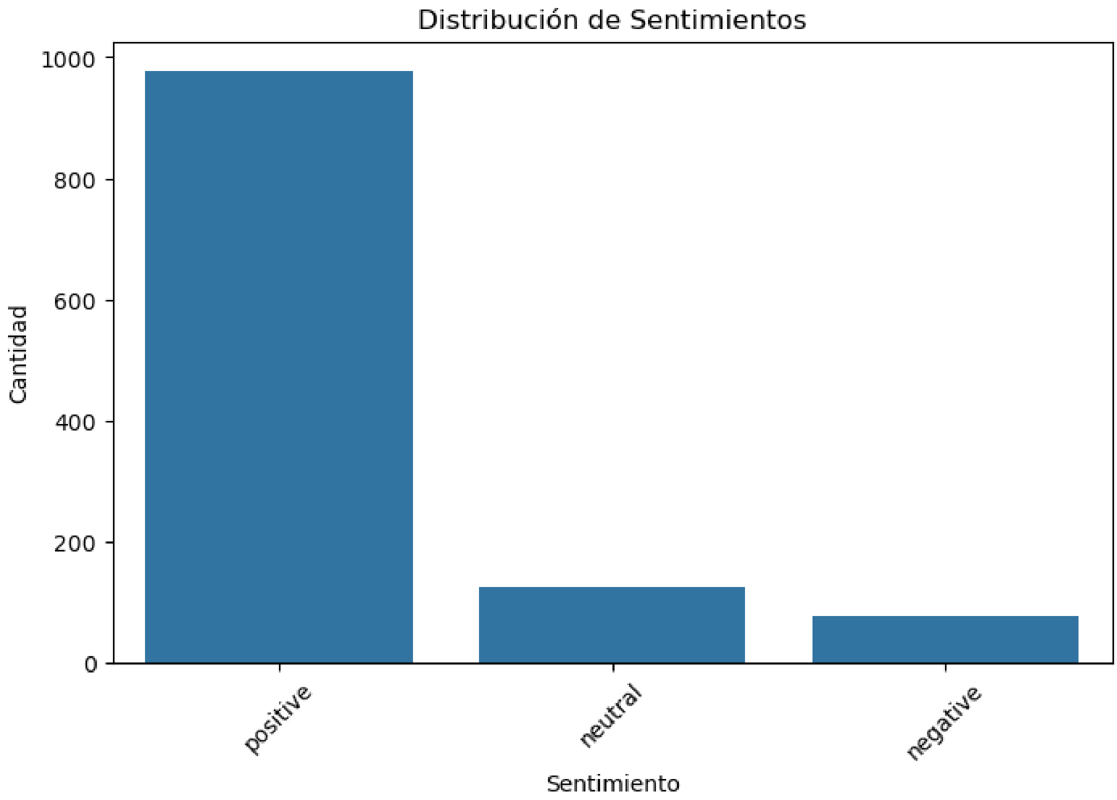
- **Data Source:** Amazon Product Reviews dataset, originally comprising 50,000 reviews.
- **Features:**
    - **Review text**: Main textual feedback provided by customers.
    - **Rating**: Numerical score provided by the customer.
    - **Sentiment label**: Derived from the rating (converted to `0`, `1`, or `2` for negative, neutral, or positive).
    - **Other metadata (optional)**: Product category, review timestamp, and user information.

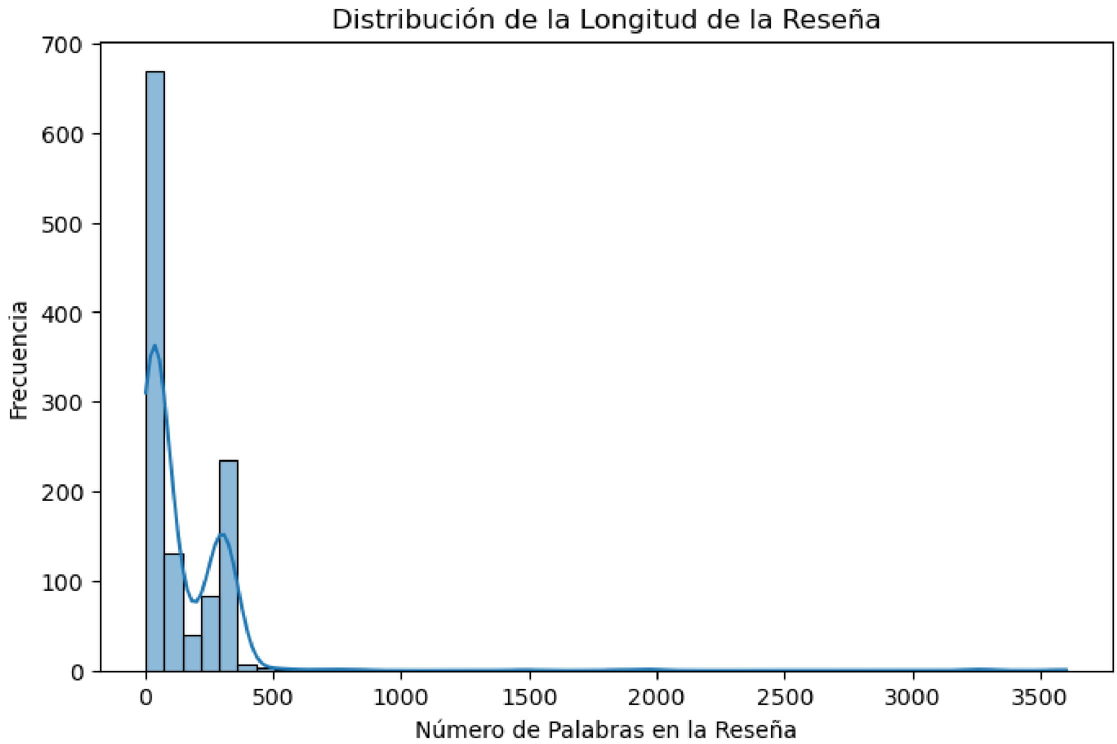
Data Utilization:

- The **review text** is the central feature for building the sentiment classification models.
- **Ratings** were used to generate labels (`0`, `1`, or `2`) to supervise training.
- **Additional metadata** (if leveraged in further analysis) could reveal insights on specific product categories or seasonal trends in sentiment.

Before modeling, significant preprocessing steps were performed:

1. Text normalization (e.g., lowercasing, removing punctuation).
2. Removal of stop words and irrelevant characters.
3. Tokenization to split reviews into meaningful units.
4. Vectorization and embedding for deep learning input.



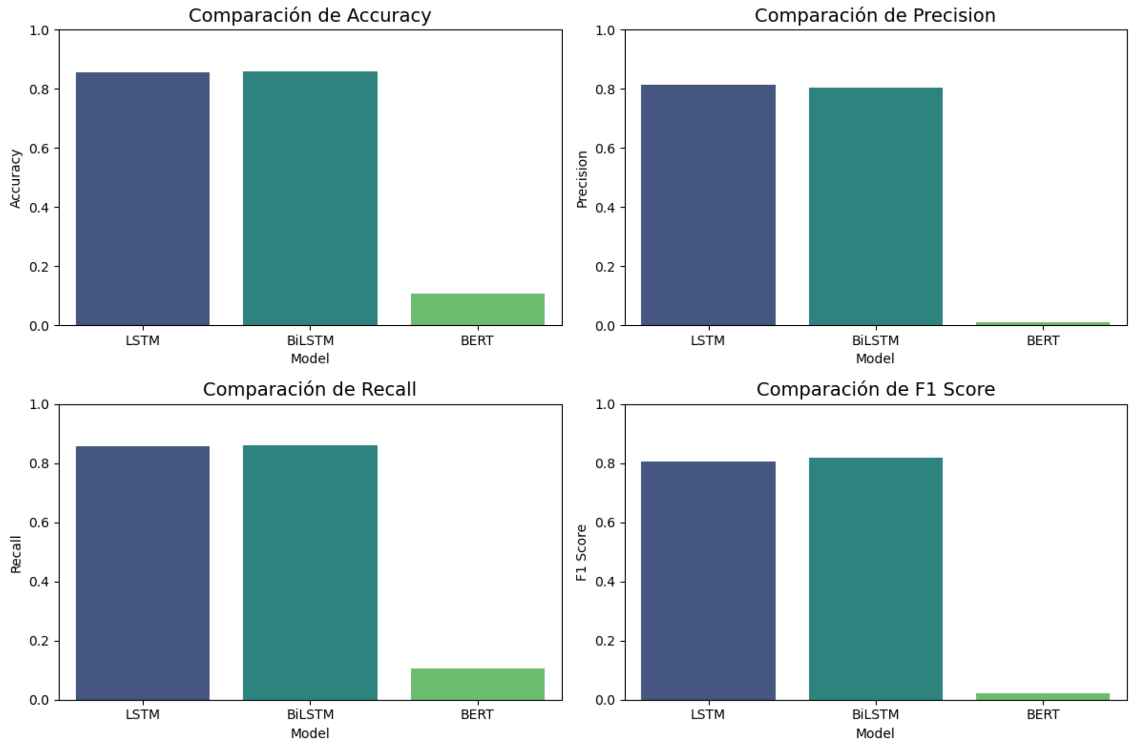


3. Deep Learning Models and Their Performance
Three deep learning approaches were tested to determine the most effective model for sentiment classification:

1. **LSTM (Long Short-Term Memory) Model**

- **Summary**: Uses an LSTM layer to capture sequential dependencies in text.
- **Performance** (on test set):
  - Accuracy: ~85.59%
  - Weighted F1 Score: ~0.8052
  - Challenges handling very small classes (e.g., class `0` with only 15 samples).

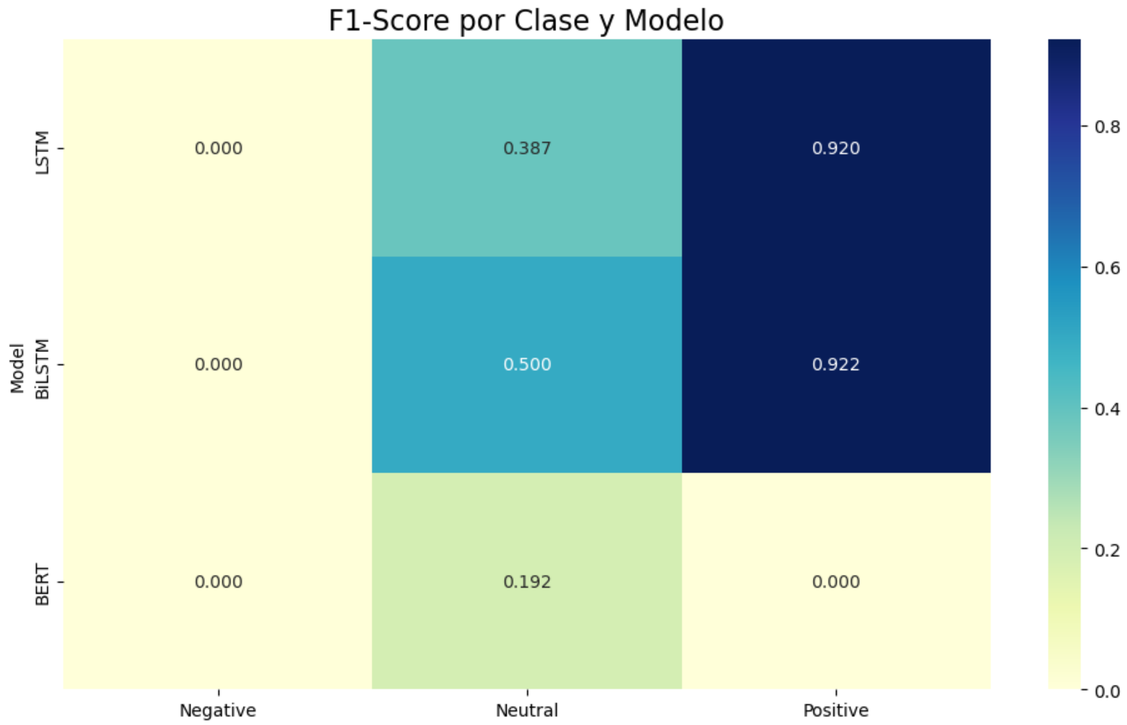
2. **Bidirectional LSTM Model**
   - **Summary**: Processes text in both forward and backward directions, improving context capture.
   - **Performance**:
     - Accuracy: ~86.02%
     - Weighted F1 Score: ~0.8184
     - Shows better recall for the largest class (`2`), but still struggles with minority classes (`0` and `1`).
3. **BERT (Transformer-Based) Model**
   - **Summary**: Fine-tuned from the pre-trained `bert-base-uncased` model, expected to capture more nuanced language.
   - **Performance**:
     - Accuracy: ~10.59%
     - Weighted F1 Score: ~0.0203
     - The poor performance suggests a mismatch or potential issue in data encoding, hyperparameter configuration, or training steps. It may also be influenced by the extreme class imbalance.



Model Best Suited to the Objective

- Despite BERT’s typical superiority in many NLP tasks, the **Bidirectional LSTM** currently outperforms the other models given the reported metrics. The

BiLSTM shows the highest F1 score (~0.8184) and the best balance across classes under the current training setup.
- The BERT model's unexpectedly low performance indicates that additional debugging or adjustments are necessary (e.g., verifying tokenization, class weights, or re-checking the training pipeline).



4. Key Findings and Observations

- **Class Imbalance**: The dataset appears highly skewed toward class `2` (positive reviews). Classes `0` (negative) and `1` (neutral) have significantly fewer samples, which negatively affects precision and recall metrics.
- **Minority Class Performance**: LSTM and BiLSTM models exhibit near-zero f1-scores for class `0`, indicating difficulty in properly learning from scarce negative samples.
- **Model Behavior**:
  - Bidirectional LSTM handles context slightly better than the basic LSTM, improving performance in the largest class while moderately increasing recall for the neutral class.
  - BERT, which typically excels in complex language understanding, underperformed here, suggesting a possible flaw in how the data was encoded, how the model was trained, or how labels are distributed.

5. Potential Flaws and Plan of Action

1. **Extreme Class Imbalance**
   - **Flaw**: Classes `0` and `1` have too few samples relative to class `2`.
   - **Action**:

        - Collect more data or apply oversampling/undersampling/augmentation techniques to balance classes.
        - Assign class weights in the loss function to penalize misclassification of minority classes.
2. **Data Encoding and Preprocessing for BERT**
    - **Flaw**: Very low accuracy for BERT suggests issues in the preprocessing or tokenization steps.
    - **Action**:
        - Revisit tokenization and ensure input shapes are correct.
        - Verify if the labels (0, 1, 2) align correctly with the model's output layer.
        - Explore learning rate and hyperparameter tuning specifically for BERT.
3. **Reviewing Hyperparameters**
    - **Flaw**: Default hyperparameters may not be optimal for a three-class problem with imbalanced data.
    - **Action**:
        - Conduct a systematic hyperparameter search (e.g., learning rate, batch size, epochs) for each model.
        - Investigate advanced architectures (e.g., ensembles) to improve minority class detection.
4. **Domain-Specific Adjustments**
    - **Flaw**: Generic preprocessing may overlook domain-specific terms or synonyms.
    - **Action**:
        - Incorporate a domain-specific vocabulary or additional textual features (e.g., product categories).
        - Fine-tune embeddings or BERT on a more domain-relevant corpus if available.

6. Next Steps

- **Data Augmentation**: Increase representation of minority sentiment classes (negative and neutral) to improve training.
- **Hyperparameter Tuning**: Use grid search or Bayesian optimization to find optimal configurations for learning rate, batch size, and sequence lengths.
- **Refine the BERT Pipeline**:
    - Confirm the correctness of input IDs, attention masks, and label mapping.
    - Verify the custom loss function (`categorical_crossentropy`) aligns with the label encoding.
    - Experiment with training BERT for more epochs, using a smaller batch size or a lower learning rate.
- **Ensemble Approaches**: Combine predictions from BiLSTM and a corrected BERT model to further improve robustness.
- **Periodic Retraining**: Integrate newly acquired reviews to keep the model updated, especially if product lines or customer behaviors change over time.

7. Questions to Refine the Analysis
To further enhance the report and your model performance, it would be helpful to clarify:

1. **Class Distribution in the Full Dataset**: How many samples belong to each of the three classes (negative, neutral, positive) before the train-test split?
2. **Preprocessing for BERT**: Could there be any mismatch between label encoding (0, 1, 2) and the BERT model's expectation of `[0, 1, 2]`?
3. **Loss Function**: Are you certain `categorical_crossentropy` is appropriate, or should we use `sparse_categorical_crossentropy` depending on how labels are encoded?
4. **Validation Data**: Are you using enough and balanced validation samples for monitoring the training process of BERT?
5. **Hyperparameter Settings**: Which specific optimizer parameters (learning rate, epsilon) and number of epochs did you use for BERT, and have you tried different settings?