Predicting Integer Ratings From Text Reviews

Dhairye Gala, Himanshu Nimonkar, and Shreyas Shah

***Abstract*—The mobile app market is intensely competitive, making user feedback critical for enhancing app quality and driving downloads. Analyzing user reviews offers valuable insights into app performance and areas for improvement. This project addresses the challenge of predicting app ratings based on user reviews by employing advanced Natural Language Processing (NLP) methods. By leveraging RNN and LSTM models for classification and regression, as well as fine-tuning an Electra Small transformer, this study aims to develop a robust system for understanding user sentiment and predicting ratings, empowering developers with actionable insights to optimize app development.**

***Index Terms*—App ratings, NLP, Neural Networks, App reviews, RNN**

I. Introduction

One of the biggest problems in the mobile app ecosystem is predicting integer app ratings (such as 1–5 stars) from textual user evaluations [1] . On sites like the Apple App Store, ratings play a crucial role in determining an app's visibility and legitimacy [2] . Higher ratings have the power to greatly improve user perception and encourage downloads. Contrarily, reviews include qualitative commentary on user experiences, covering both the app's advantages and disadvantages [3] . App evaluation and improvement procedures are hampered by the enormous number of reviews that are produced every day, which makes it hard for developers to manually examine and extract insightful information [4] .

The subjective character of textual assessments, which can differ greatly in tone, length, and content, exacerbates this issue even further [5] . One review may offer thorough, helpful criticism, while another may be brief and ambiguous. Notwithstanding this variation, the related rating provides a numerical representation of the user's overall emotion, which is reflected in each review. It is a difficult but necessary work to bridge the gap between textual reviews and their ratings since it enables developers to automatically identify an app's strengths, shortcomings, and areas that need improvement.

This project enables scalable feedback analysis by predicting app ratings from user evaluations using natural language processing (NLP) approaches. It facilitates the rapid identification of problems, prioritization of enhancements, and strategy refinement for developers. This improves competitiveness, customer pleasure, and app maintenance. Accurate predictions are achieved by models such as Electra Small [6] , LSTMs (Long Short-Term Memory) [7] , and RNNs (Recurrent Neural Networks).

II. Literature Survey

To categorize reviews and forecast ratings, Romadhony et al. [8] applied sentiment analysis to the FDReview dataset, which included more than 700,000 product reviews from Indonesia. Multinomial Naïve Bayes (MNB), Support Vector Machine (SVM), LSTM, and BiLSTM models were used in their investigation. MNB scored better than SVM in rating prediction, whereas BiLSTM produced the highest overall accuracy, reaching 57% as opposed to 55% with LSTM. In addition to providing insights from tests on balanced and unbalanced small-sized datasets, the study demonstrated the efficacy of BiLSTM for sentiment classification and rating prediction.

Liu et al. [9] presented a deep learning-based method for online product ranking that uses probabilities linguistic term sets (PLTS) to predict sentiment intensities. Their approach, which merged sophisticated models with natural language processing, produced accuracy rates of 40.36% with a textRNN model and 57.18% with a transformer-based model. They illustrated the potential of deep learning for detecting sentiment tendencies in review texts by adding a matching mechanism to increase sentiment classification accuracy. Their research demonstrated how PLTSs can improve online product evaluation decision-making by increasing the accuracy of competition predictions.

III. Methodology

A. Outline

To start, we preprocessed the dataset by eliminating noise and filling in any missing values. After that, we tried a number of regression and classification strategies, adjusting various hyperparameters to efficiently train the models. Then, using bidirectional LSTM, layer normalization, synonym replacement, and tuned hyperparameters, we created the best-performing regression model. To improve classification performance, we also used a final classification head to train the Electra Small model. Lastly, in an attempt to enhance the overall outcomes, we increased the model complexity for both the regression and classification tasks.

B. Data Collection & Cleaning

The data for this project was collected from iOS App Store applications by performing web scraping to extract user reviews and their associated integer ratings. The code for the web scraping step is not included in the notebook that is provided because it is outside the scope of this project, even though it was essential for collecting the dataset.

To guarantee consistency and eliminate noise, the text reviews are preprocessed as part of the data cleaning procedure. This include changing the text's capitalization, eliminating HTML tags, URLs, punctuation, emoticons, emojis, and mentions. The text is lemmatized to reduce words to their most basic forms and stopwords are eliminated. For easier reading, contractions are enlarged (for example, "I'm" to "I am"). A CSV file with the cleaned data is stored for later use.

We use a dataset of 90,000 text reviews and their corresponding ratings for training. This data is then split into 80% for training, and 10% each for validation and testing.

C. Model Architecture for Classification and Regression

Our proposed model uses an LSTM network along with batch normalization and dropout for strong performance. The architecture can process sequential input data, including tokenized text, to predict a continuous or discrete value. The main components and the justification for their inclusion are described below:

- **Embedding Layer:** Each word in the input vocabulary is mapped to a dense vector representation by the model's embedding layer, which comes first. By capturing the semantic links between words and providing a lower-dimensional input to the LSTM layer, this layer lowers processing overhead while preserving important patterns in the data.
- **LSTM Layer:** The LSTM layer processes the embedded input sequentially, capturing temporal dependencies and long-range contextual information. Unlike traditional RNNs, LSTM mitigates the vanishing gradient problem, making it well-suited for handling lengthy sequences of text reviews.
- **Batch Normalization:** To normalize the activations and enhance training stability, batch normalization is performed to the output of the LSTM's final hidden state. By avoiding overfitting, this normalization step improves the model's generalization, speeds up convergence, and lessens internal covariate shifts.
- **ReLU Activation:** A ReLU activation function is applied to introduce non-linearity, enabling the model to capture complex relationships between input features and the target variable.
- **Dropout Regularization:** During training, a dropout layer is added to randomly deactivate a portion of the neurons. By keeping the model from depending too much on any one neuron, this regularization strategy reduces overfitting and enhances the model's capacity for generalization.
- **Fully Connected (FC) Layer:** In the final FC layer, the processed feature vector is projected into a single output value, which in the case of classification represents a single integer or the predicted continuous target for regression.
- **Rationale:** Our model architecture is tailored to handle the sequential nature of text data and the regression/classification task requirements effectively. This combination of components ensures that the model can learn meaningful representations of text data while being resilient to overfitting and capable of generalizing well to unseen samples. The architecture strikes a balance between complexity and computational efficiency, making it suitable for large-scale datasets with high variability.

$$i_t = \sigma(W_i e_t + U_i h_{t-1} + b_i), \quad \text{(Input gate)} \tag{1}$$

$$f_t = \sigma(W_f e_t + U_f h_{t-1} + b_f), \quad \text{(Forget gate)} \tag{2}$$

$$o_t = \sigma(W_o e_t + U_o h_{t-1} + b_o), \quad \text{(Output gate)} \tag{3}$$

$$\tilde{c}_t = \tanh(W_c e_t + U_c h_{t-1} + b_c), \quad \text{(Candidate memory)} \tag{4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad \text{(Cell state)} \tag{5}$$

$$h_t = o_t \odot \tanh(c_t), \quad \text{(Hidden state)} \tag{6}$$

where:

i_t, f_t, o_t	are the input, forget, and output gates, respectively,
$c_t, \tilde{c}_t$	are the cell state and candidate memory,
h_t	is the hidden state,
W, U, b	are learnable weight matrices and biases,
σ	is the sigmoid activation function,
tanh	is the hyperbolic tangent activation function, and
$\odot$	denotes element-wise multiplication.

The equations (1), (2), (3), (4), (5) and (6) describe the LSTM layer's internal operations, including gate functions and state updates. These are included to explain how the model captures sequential dependencies in text data effectively.

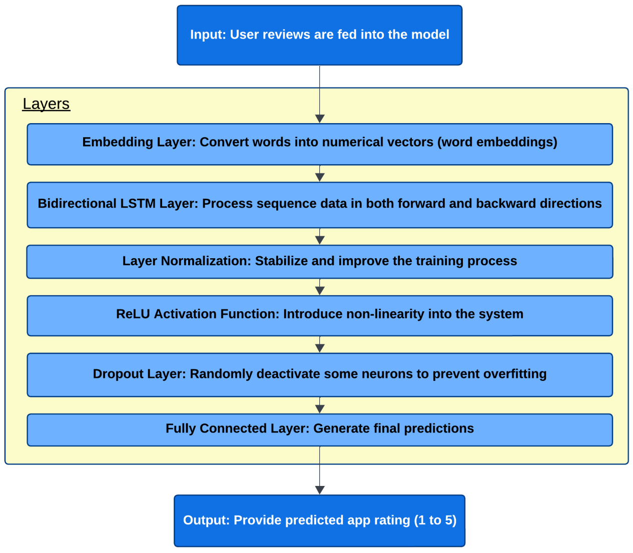
Input: User reviews are fed into the model
Layers
Embedding Layer: Convert words into numerical vectors (word embeddings)
Bidirectional LSTM Layer: Process sequence data in both forward and backward directions
Layer Normalization: Stabilize and improve the training process
ReLU Activation Function: Introduce non-linearity into the system
Dropout Layer: Randomly deactivate some neurons to prevent overfitting
Fully Connected Layer: Generate final predictions
Output: Provide predicted app rating (1 to 5)

Fig. 1. Model Architecture Diagram

D. Classification

To train the classification models, we conducted experiments by varying key hyperparameters to optimize performance. The hyperparameters and their values are as follows:

- **Weight Decay**: Values of 0.00001, 0.0001, and 0.001 were tested to control regularization strength and prevent overfitting.
- **Dropout Rates**: We evaluated dropout rates of 0.0, 0.1, 0.2, 0.25, and 0.35 to introduce regularization and improve generalization.

- **Hidden Layer Sizes**: The hidden layer sizes were varied among 32, 64, 128, and 256 units to analyze the impact of model capacity on performance.
- **Learning Rate Scheduling**: A StepLR scheduler was used with a step size of 5 epochs and a decay factor (γ) of 0.5, allowing the learning rate to decrease periodically during training.

The models were trained using the Adam optimizer and the loss function was **Cross Entropy loss**. Each hyperparameter combination was evaluated based on validation accuracy and loss.

Finally, the model complexity was increased to use a bidirectional LSTM-based architecture with layer normalization and two fully connected layers for feature refinement.

E. Regression

To train the regression models, we conducted experiments by varying key hyperparameters to optimize their performance. The following hyperparameters and values were explored:

- **Weight Decay**: Values of 0.00001, 0.0001, and 0.001 were tested to control regularization strength and reduce overfitting.
- **Dropout Rates**: We experimented with dropout rates of 0.1, 0.2, and 0.4 to introduce regularization and enhance generalization.
- **Hidden Layer Sizes**: The hidden layer sizes were varied among 32, 64, and 256 units to evaluate the impact of model capacity on performance.
- **Learning Rate Scheduling**: A StepLR scheduler was employed with a step size of 5 epochs and a decay factor (γ) of 0.5 to systematically reduce the learning rate during training.

The models were trained using the Adam optimizer and the loss function was **Mean Squared Error** (MSE) loss. Each hyperparameter combination was evaluated based on validation accuracy and loss.

Finally, the model complexity was increased to use a bidirectional LSTM-based architecture with layer normalization and two fully connected layers for feature refinement.

F. Optimized Model

The best regression model leverages a bidirectional Long Short-Term Memory (LSTM) network with layer normalization and dropout to predict app ratings from textual reviews. Layer normalization is used instead of batch normalization as it is better suited for sequential models, normalizing across features within a timestep rather than across the batch, which ensures consistent performance regardless of batch size. The bidirectional LSTM is employed instead of a standard LSTM to capture contextual information from both past and future tokens, improving the model's ability to understand the nuances of text reviews. Synonym replacement is used as a data augmentation technique to introduce lexical diversity, improving the model's robustness to variations in textual input. The architecture includes an embedding layer, a single bidirectional LSTM layer, layer normalization, ReLU activation, dropout, and a fully connected layer for final prediction. Key hyperparameters include an embedding size of 200, a hidden size of 256, a dropout rate of 0.5, and a learning rate of 0.001. Training is regularized with a weight decay of 0.00001 and a StepLR learning rate scheduler with a step size of 5 epochs and a decay factor of 0.1. This design effectively balances complexity and generalization for textual data regression tasks.

G. Electra Small Model

First we train **ElectraForSequenceClassification** for predicting app ratings (1–5), with all layers, including the pre-trained Electra encoder, set to trainable by default. It uses a pre-trained Electra encoder and a classification head. Hyperparameters include a learning rate of 0.00005, batch size of 32, and max sequence length of 128. AdamW optimizer and CrossEntropyLoss are used. The model trains on tokenized reviews, evaluates accuracy and loss, and visualizes performance with a confusion matrix.

Then we fine-tune **ElectraForSequenceClassification**, a transformer-based architecture, for predicting app ratings (1–5). The hidden layers of the pre-trained Electra encoder are frozen, training only the classification head to focus on task-specific fine-tuning. Hyperparameters, including learning rate (0.000001 to 0.0001), weight decay (0.000001 to 0.01), maximum sequence length (64 to 256), and batch size (16, 32, 64), are optimized using **Optuna**. The AdamW optimizer and CrossEntropyLoss are used during training. After hyperparameter tuning, the best configuration is used for full training, with performance evaluated on validation and test sets.

IV. Results

- ***For this paper, we will consider the Electra Model Fully Trained with Classification Head as the Baseline Model, and Optimized Model using Regression as the best model.***
- All models were trained for 20 epochs, as experiments indicated that accuracy plateaued beyond this point and, in most cases, began overfitting to the training data. For the optimized model, training was extended to 30 epochs.
- Adjustments to dropout rates, hidden sizes, and weight decay were incorporated only when they resulted in a substantial increase in accuracy, which was observed infrequently.
- For all models, the statistics reported in Table I are for the 20th epoch, and in cases where hyperparameter adjustments are made through an array of values, the best test accuracy has been reported.

The confusion matrix (Figure 2) of our best model clearly displays a diagonal in the heat map, indicating that the model accurately predicts the correct rating most of the time. Even when the model is wrong, the predicted rating is usually within one integer of the actual rating, suggesting that the model effectively captures the sentiment of the rating. Additionally, the graphs for validation loss (Figure 3), validation accuracy (Figure 4), training loss (Figure 3), and training accuracy (Figure 4) plateau after just 10 epochs, demonstrating that

TABLE I
RESULTS OF EXPERIMENTS

Model Name	Validation Accuracy	Validation Loss	Test Accuracy
Classification with Batch Normalization	47.81	2.31	49.01
Classification with Weight Decay Adjustments	49.75	1.20	50.18
Classification with Dropout Adjustments	48.06	2.11	48.92
Classification with Hidden Size Adjustments	50.19	1.22	51.31
Classification with StepLR	50.12	1.39	50.42
Regression with Batch Normalization	46.98	0.96	48.24
Regression with Weight Decay Adjustments	48.65	0.92	48.85
Regression with Dropout Adjustments	48.96	0.82	49.38
Regression with Hidden Size Adjustments	47.85	0.92	49.64
Regression with StepLR	46.57	1.03	46.34
Optimized Model using Regression	**53.43**	**0.66**	**52.9**
Electra Model Fully Trained with Classification Head	**51.12**	**1.99**	**50.83**
Electra Model with Hidden Layers Frozen and Fine-Tuned Using Optuna	38.81	1.39	39.27
Classification with Increased Model Complexity	50.41	1.16	50.40
Regression with Increased Model Complexity	43.03	0.89	43.64

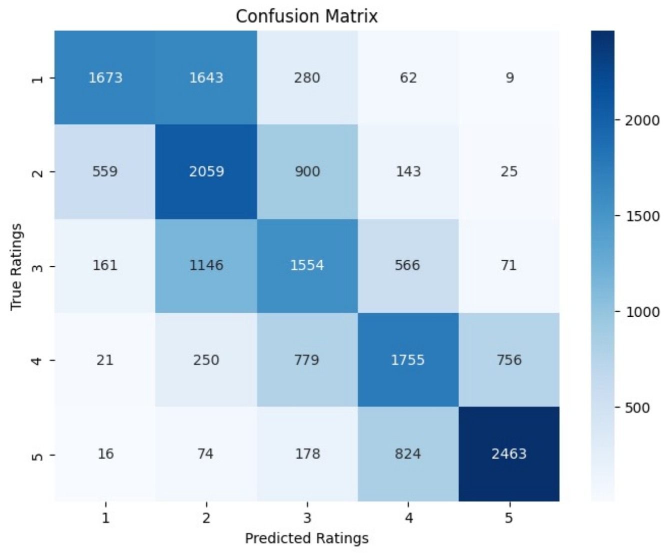

Fig. 2. Confusion Matrix for Optimized Regression Model

TABLE II
OPTIMIZED REGRESSION MODEL PARAMETERS

Layer	Parameters
Embedding	6519200
LSTM (combined)	2514944
FC (combined)	513
Total (including additional parameters)	9035681

peak model performance is achieved relatively quickly during training.

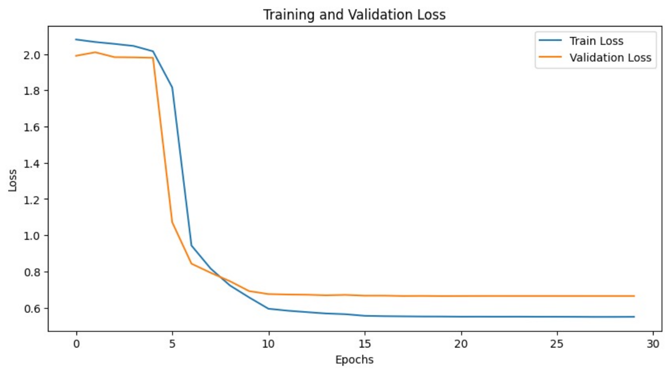

Fig. 3. Training and Validation Loss for Optimized Regression Model

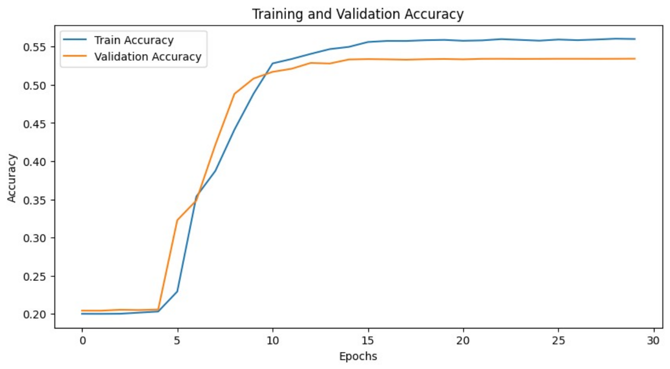

Fig. 4. Training and Validation Accuracy for Optimized Regression Model

TABLE III
OPTIMIZED REGRESSION MODEL SIZE & FLOPS PER FORWARD PASS

FLOPs	505.08 M
Size	34.47 MB

Table II shows the number of parameters and Table III shows the model size and FLOPs per forward pass for the optimized regression model. The model size and number of FLOPs indicates a lightweight model.

A. Comparison of Optimized Model with Baseline (Electra) and Literature Survey

For app rating prediction from textual reviews, the comparison demonstrates the superiority of the optimized bidirectional LSTM model above the baseline Electra and literature benchmarks. By using effective strategies such layer normalization, dropout, and synonym-based data augmentation, the optimized model outperformed Electra with a validation accuracy of 53.43% and a test accuracy of 52.9%, respectively, outperforming its 51.12% and 50.83% respective results. On the other hand, earlier research revealed that BiLSTM achieved 57% accuracy, proving its resilience in comparable tasks but with different datasets and configurations. The outcomes support the efficacy of strategic regularization and bidirectional architectures in sentiment and rating prediction tasks. Additionally, our model is much smaller and computationally efficient than Electra Small [10] .

V. Conclusion

This study demonstrates the potential of leveraging advanced Natural Language Processing techniques to predict integer app ratings from textual user reviews. By utilizing a combination of RNNs, LSTMs, and transformer-based architectures such as Electra Small, we developed models capable of effectively bridging the gap between qualitative feedback and quantitative metrics. Our optimized regression model shows robust performance, accurately predicting ratings in most cases, with incorrect predictions typically deviating by only one integer. This capability highlights the model's practical utility in capturing user sentiment and providing actionable insights for developers.

The efficiency of the proposed approach is further evident in its lightweight design (34.47 MB), requiring only 505.08 M FLOPs per forward pass, and in its training metrics, which plateau after just 10 epochs. These characteristics make the model suitable for scalable deployment, addressing the challenge of processing large volumes of user feedback in real time. The inclusion of techniques such as batch normalization, dropout, and layer normalization ensured both stability and generalization, further enhancing the model's applicability.

This research not only provides a framework for automating feedback analysis but also opens avenues for future work.

VI. Future Scope

This work establishes the groundwork for future developments in the automation of feedback analysis through the use of natural language processing techniques. In order to improve the model's generalizability across a range of scenarios, future research could concentrate on growing the dataset to include reviews from various languages and sources. Prediction accuracy may be improved by including multimodal inputs, such as text reviews combined with metadata like timestamps or app classifications. Furthermore, investigating transformer designs and self-supervised learning techniques could result in even greater model performance gains. These developments could increase the system's effect by expanding its application to other fields.

Credit authorship contribution statement

All authors have equal contributions

References

[1] P. Schneider, "App ecosystem out of balance: An empirical analysis of update interdependence between operating system and application software," 2020.

[2] N. Genc-Nayebi and A. Abran, "A systematic literature review: Opinion mining studies from mobile app store user reviews," *Journal of Systems and Software*, vol. 125, pp. 207–219, 2017.

[3] A. Ciurumelea, A. Schaufelbühl, S. Panichella, and H. C. Gall, "Analyzing reviews and code of mobile apps for better release planning," in *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)*, 2017, pp. 91–102.

[4] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, "AR-miner: mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 767–778.

[5] S. Hunston and G. Thompson, *Evaluation in text: Authorial stance and the construction of discourse: Authorial stance and the construction of discourse*. Oxford University Press, UK, 2000.

[6] K. Clark, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[7] S. Hochreiter, "Long Short-term Memory," *Neural Computation MIT-Press*, 1997.

[8] A. Romadhony, S. A. Faraby, R. Rismala, U. N. Wisesti, and A. Arifianto, "Sentiment Analysis on a Large Indonesian Product Review Dataset." *Journal of Information Systems Engineering &amp; Business Intelligence*, vol. 10, no. 1, 2024.

[9] Z. Liu, H. Liao, M. Li, Q. Yang, and F. Meng, "A deep learning-based sentiment analysis approach for online product ranking with probabilistic linguistic term sets," *IEEE Transactions on Engineering Management*, vol. 71, pp. 6677–6694, 2023.

[10] K. Clark, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.