<u>**FINAL PROJECT**</u>

Title: Sentiment Analysis of Hotel Reviews

Submitted by

Chetna Mishra (Batch - 9)

Sugarchins@gmail.com

Advance Certification in Applied Data Science, Machine Learning and IoT

E&ICT Academy, IIT Guwahati

1. Title of the Project:

Sentiment Analysis of Hotel Reviews Using Machine Learning and Deep Learning Techniques

2. Project Brief

This project aims to develop an automated system that can analyze hotel customer reviews and classify the sentiment as **positive** or **negative**. In the era of online travel platforms, thousands of customer reviews are generated for hotels, making it challenging for both travelers and hotel managers to manually digest the feedback. An accurate sentiment classifier helps **summarize customer satisfaction** and pinpoint areas of improvement from negative feedback.

Problem Statement & Motivation

Problem: Given a hotel review (free-text feedback), determine whether the sentiment expressed is *positive* (satisfied customer) or *negative* (dissatisfied customer) with high accuracy and reliability.
Motivation:
- **User Decision-Making:** Travelers heavily rely on reviews to choose accommodations. A sentiment summary can quickly inform potential customers of a hotel's overall reputation without reading every review.
- **Hotel Reputation Management:** Hotels can automatically monitor sentiment trends. Prompt detection of rising negative sentiment (e.g. complaints about cleanliness or service) enables proactive improvements.
- **Scale:** Large volumes of unstructured text reviews (often hundreds of thousands across platforms) demand automated NLP solutions, as manual analysis is impractical.

3. Deliverables of the Project

- **Data Preprocessing & EDA:** Thorough exploration of the hotel reviews dataset (distribution of review sentiments, review length statistics, common words in positive vs. negative feedback). Develop a text cleaning pipeline (lowercasing, stopword removal, punctuation handling, etc.) and define how sentiment labels will be assigned from the raw data.
- **Feature Engineering (Text):** Experiment with text vectorization methods for classical models: e.g. Bag-of-Words and TF-IDF representations of reviews. Ensure feature spaces are suitable (vocabulary sizing, n-grams) for learning sentiment signals.
- **Classical ML Modeling:** Train and tune multiple traditional classifiers on the engineered features. Candidates include Logistic Regression, Naïve Bayes, Decision Tree, K-Nearest Neighbors, Random Forest, AdaBoost, Gradient Boosting, and XGBoost. Use grid search and cross-validation to optimize hyperparameters for each model. Evaluate performance using accuracy, precision, recall, F1-score, and confusion matrix.
- **Deep Learning Modeling (RNNs):** Build and train RNN-based models on sequence data from the reviews. Architectures to implement: Simple RNN, LSTM, GRU, Bidirectional LSTM, and Stacked LSTM. Use a word embedding layer and appropriate sequence length truncation. Train each network for a fixed number of epochs and evaluate on the test set using the same metrics as classical models.
- **Comparison & Best Model Selection:** Compare the evaluation metrics of all models (ML and RNN). Identify the best-performing model overall and provide a detailed analysis of its performance (including a classification report and confusion matrix).
- **Deployment Plan:** Outline how the best model can be deployed in a real-world setting (e.g. as a web service or integrated into a hotel review platform). Provide an end-to-end example of how a new review would be processed by the system to predict sentiment.
- **Comprehensive Report:** A complete project report (this document) following the prescribed template, including visualizations (e.g. word clouds, performance bar charts, confusion matrices) and tables of results. Key findings, limitations are discussed to conclude the study.

4. Data Description and Labeling Strategy

Dataset Overview: The project uses a large public dataset of hotel reviews collected from Booking.com, encompassing **515,738 reviews** from various European hotels[2]. Each entry in the dataset contains the text of a positive comment and a negative comment provided by the reviewer, along with other fields like the reviewer's numeric rating, hotel information, etc. Key fields include:

- **Positive_Review:** The text that the reviewer left in the "positive" comment box (what they liked about the hotel). If the reviewer had nothing positive to say, this field contains the phrase "No Positive".
- **Negative_Review:** The text from the "negative" comment box (what they disliked). If there were no complaints, this field is "No Negative".
- **Reviewer_Score:** A numeric score (0 to 10) given by the reviewer, which generally correlates with the sentiment (higher scores imply positive sentiment).

The data is rich and comprehensive, covering hotels across Europe and multiple languages (though the majority of reviews are in English[3]). No significant missing textual data is present – every review has at least a placeholder in each comment field. Class distribution is inherently imbalanced towards positive feedback, as many reviewers are satisfied; this is reflected in the abundance of high scores and "No Negative" comments.

Labeling Strategy: To train supervised models, we derived a binary sentiment label for each review comment: -

- Reviews were **labeled "Positive" (1)** if they reflect positive sentiment. We utilized the `Positive_Review` text for these examples. Each non-empty positive comment (excluding cases where it was just "No Positive") is treated as a positive-sentiment instance.
- Reviews were **labeled "Negative" (0)** using the `Negative_Review` text. Each non-empty negative comment (excluding "No Negative") is treated as a negative-sentiment instance.

This approach leverages the dataset's structure: each review from a customer yields up to two text snippets (one positive, one negative). By separating them, we obtain a large corpus of clearly positive vs. clearly negative statements. For example, a review where a guest wrote *"Great location and very friendly staff"* in the positive comment is used as a **positive** instance, whereas *"Room was dirty and small, very noisy at night"* from a negative comment is a **negative** instance. Overall, we gathered roughly **480k positive** snippets and

380k negative snippets after removing the placeholder phrases. To mitigate class imbalance (55% positive vs 45% negative), the training dataset was **balanced** by random undersampling of the majority class (positive) to equal the number of negative samples. This ensures the classifiers see an equal number of positive and negative examples during training, preventing bias towards the more frequent class.

5. EDA & Preprocessing Summary

Exploratory Data Analysis: We conducted initial EDA to understand the review texts and sentiment distribution: -

- The **average review length** differs between sentiments. Negative comments tend to be longer on average (roughly 25 words) than positive comments (around 17 words) in this dataset. Many positive reviews are succinct praises ("Excellent location!", "Very clean."), whereas negative reviews often include detailed complaints and multiple issues, hence longer text.
- We observed certain **frequently used words** characteristic of each sentiment. In negative reviews, words like *"room"*, *"not"*, *"small"*, *"dirty"*, and *"unfortunately"* appear often, reflecting common complaints (e.g. about room size or cleanliness). In positive reviews, terms such as *"staff"*, *"location"*, *"good"*, *"great"*, *"clean"*, *"friendly"*, and *"helpful"* are very frequent, highlighting what customers appreciated (e.g. hotel staff and location are crucial positive aspects). These observations were confirmed by generating separate word clouds for positive and negative comments, which vividly showed praise-related words dominating one cloud and complaint-related words dominating the other.
- The **distribution of numeric ratings** (Reviewer_Score) aligns with the text sentiment: most scores are on the higher end (indicating satisfaction), and a smaller portion of reviews have very low scores. This reinforces that our labeling via text fields is consistent with the actual ratings – reviews with "No Negative" comments almost always have high scores (positive sentiment), while those with lengthy negative comments tend to have lower scores.

Text Cleaning & Preprocessing: Before feature extraction, review texts were cleaned using a standard NLP pipeline: -

- All text was **lowercased** to ensure uniformity (e.g., "Great" and "great" are treated the same). - **Stop words** (common words like *"the", "and", "is"*, etc., which carry little semantic meaning) were removed to reduce noise in the data. This helps the models focus on the impactful words (e.g., “staff”, “dirty”, “excellent”).

- **Punctuation and special characters** were removed. We retained only alphabetic characters, converting contractions to their expanded form where possible. (For example, “didn't” → “did not”). This prevents punctuation or symbols from affecting tokenization; however, we note that negations were carefully handled so as not to lose the semantic meaning (e.g., “not happy” should remain distinguishable from “happy”).
- **Tokenization:** Each review text was split into individual tokens (words). We used a regex-based tokenizer to extract words, which, after the above cleaning, yields sequences of alphabetic tokens.
- **Lemmatization/Stemming:** We applied lemmatization to reduce words to their base form (e.g., “rooms” → “room”, “cleaned” → “clean”). This normalization groups word variants, aiding the model to generalize. A light Porter stemming was also applied prior to lemmatization to handle suffixed variations consistently.
- **Handling placeholders:** Entries with “No Positive” or “No Negative” were dropped from the respective corpora as described in the labeling strategy. These placeholders carry no sentiment information (in fact, “No Positive” indicates a very negative overall review, but we rely on the accompanying negative comment for that instance).

After preprocessing, the text data was ready for feature engineering. Notably, no **missing data** needed imputation in the text fields (every review provided at least one of the two comment fields). We also verified that after cleaning, the vocabulary size was manageable for modeling (tens of thousands of unique words, which we constrain in feature engineering steps).

6. Feature Engineering

For classical machine learning models, we transformed the cleaned text into numeric feature representations. We experimented with the following:

vectorization methods: -

- **Bag-of-Words (BoW):** We created a unigram frequency representation where each review is converted into a vector of word counts. We limited the vocabulary to the top 10,000 most frequent words to keep feature dimensions tractable. Two variations were tried: using only unigrams, and using unigrams+bigrams (pairs of consecutive words) to capture simple phrases (e.g., “not clean”). However, including bigrams greatly increases feature count and sparsity, so the final models primarily used unigrams for simplicity.

- **TF-IDF Transformation:** We applied Term Frequency–Inverse Document Frequency weighting to the bag-of-words features. TF-IDF down-weights very common words and up-weights rarer, more discriminative words. This often improves performance in text classification by emphasizing distinctive words for each class. Each review is represented as a 10,000-dimension TF-IDF vector (with the same vocabulary as the BoW approach).

No additional numeric meta-features (like review length or counts of punctuation) were added to the feature set for classical models in this project. We considered that such features might help (e.g., perhaps negative reviews tend to have more exclamation marks “!” or are longer, etc.), but initial exploration showed the signal was adequately captured by textual features alone. The focus was kept on the text content itself to allow a fair comparison with the deep learning models (which also only ingest raw text).

For deep learning models (RNNs), feature engineering mainly involved preparing the **token sequences**: -

- We constructed a word **index vocabulary** of the most frequent words (size = 10,000) from the training corpus. Each word was assigned an integer index. Words not in the top 10k or unseen in training are treated as “out-of-vocabulary” tokens.
- Each review’s text was converted into a sequence of word indices. We set a **maximum sequence length** of 100 tokens. Reviews longer than 100 words were truncated (extra words beyond 100 were dropped), and reviews shorter than 100 were **padded** with a special `<PAD>` token at the end. This fixed-length sequence approach is required for efficient batch processing in neural networks. (We chose 100 based on distribution of review lengths: this covers the vast majority of reviews, as very few exceed 100 words significantly).
- An **embedding matrix** was used in the RNN models to map each word index to a dense vector representation. We let the embedding be learned from scratch (no pre-trained embeddings were used) with an embedding dimension of 64. Thus, each word index ultimately becomes a 64-dimensional learned feature vector that the RNN can work with.

After these steps, the data was ready for input into the respective models: classical models received high-dimensional sparse vectors (from BoW/TF-IDF) and deep learning models received sequences of word indices.

7. Models Considered

We evaluated a broad range of models, from simple classifiers to complex RNN architectures:

Classical Machine Learning Models:

- *Logistic Regression:* A linear model that learns a weighted sum of features to predict the log-odds of positive sentiment. We expected this to perform well as a baseline, since logistic regression is known to be effective for text classification tasks (similar to how it's often used for spam detection)
- *Support Vector Machine (Linear SVM):* A linear classifier that finds the hyperplane maximizing the margin between positive and negative classes. SVMs have historically shown strong performance in sentiment analysis with appropriate text features.
- *Multinomial Naïve Bayes:* A probabilistic classifier that uses word frequencies under a naive independence assumption. It often performs surprisingly well for text classification (due to the nature of word distributions) and serves as a good lightweight benchmark.
- *Decision Tree:* A non-linear model that splits on features to form a tree of decision rules. While able to capture some interactions, a single tree can easily overfit, especially in high-dimensional spaces, so performance might be limited.
- *Random Forest:* An ensemble of decision trees (bagging approach) that averages their predictions. By aggregating many trees, the model can achieve better generalization than a single tree. We expected Random Forest to handle the text features better than a single Decision Tree and possibly capture non-linear patterns missed by linear models.

Deep Learning RNN Models:

- *Simple RNN:* A single-layer vanilla RNN with a modest number of units (we used 64 units). This model processes the word sequence one token at a time and retains an internal state. It's the simplest recurrent architecture and can suffer from short-term memory and vanishing gradient issues on longer sequences, but it sets a baseline for RNN performance.

- *LSTM (Long Short-Term Memory):* A recurrent network with gating mechanisms (forget, input, output gates) designed to overcome the memory limitations of simple RNNs. We used one LSTM layer (64 units). LSTMs can capture longer-term dependencies in text (e.g., negation that affects meaning of words far apart) better than Simple RNNs. We anticipated LSTM to outperform the Simple RNN.
- *GRU (Gated Recurrent Unit):* A simplified gating RNN architecture that is somewhat faster than LSTM. We built a 1-layer GRU (64 units). GRUs often perform comparably to LSTMs on many tasks while being more efficient – we wanted to compare its performance on our dataset.
- *Bidirectional LSTM:* This model consists of an LSTM layer that processes the sequence in both forward and backward directions, then concatenates the outputs. The Bidirectional LSTM (with 64 units in each direction) can incorporate context from both past and future tokens, which is valuable for tasks like sentiment (e.g., in the phrase "not at all clean," the word "clean" is better interpreted with knowledge of the preceding "not at all"). We expected this to potentially be the best RNN variant due to its ability to capture context on both sides.
- *Stacked LSTM:* A deeper RNN with multiple LSTM layers (we used two layers: 64 units in the first, feeding into 32 units in the second). The idea is that the first LSTM layer might capture lower-level sequence patterns, and the second layer captures higher-level temporal abstractions. Deeper models might achieve slightly higher accuracy at the cost of more training time and complexity.

All deep models ended with a Dense layer of 1 unit with sigmoid activation to output a probability of the review being positive. We used the same embedding (64-dim) for all RNN models to ensure a fair comparison.

Note: We did not include Transformer-based models (e.g. BERT) in the model roster for this project, focusing instead on the above RNNs vs. classical approaches. Transformer models fine-tuned for sentiment analysis could likely surpass the performance of both classical ML and RNNs, but their training and deployment requirements are significantly higher, and they were outside the scope of our comparison.

8. Training Procedure

We split the dataset into training and test sets to evaluate model performance on unseen data. The data was shuffled and stratified by sentiment label when splitting, to maintain equal class proportions. We used an **80/20 split**: 80% of the data for training (and validation) and 20% held out as a final test set. Given the very large dataset, this resulted in a test set of substantial size (on the order of tens of thousands of instances), providing high confidence in the statistical significance of the results.

Each model was trained on the TF-IDF feature matrix of the training reviews. Model selection was primarily based on **F1-score** for the positive class (since we value a balance of precision and recall). Training was parallelized for tree ensembles where possible to speed up the process. We applied standard practices like early stopping for XGBoost (monitoring validation loss) to prevent overfitting. After tuning, the best hyperparameters were chosen and a final model was refit on the entire training set for each algorithm.

RNN Model Training: The RNN models were implemented in TensorFlow/Keras. We used the following training setup for all:

- **Embedding layer:** 10,000 vocabulary, output dimension 64. This layer is initialized randomly and learned during training.

- **Optimizer and Loss:** Used the Adam optimizer (with default learning rate 0.001) and binary crossentropy loss for binary classification.

- **Epochs:** Trained for 5 epochs in each case. We monitored validation loss (with a 10% validation split from the training data) and observed that performance typically plateaued or early-stopped around epoch 3-4, so 5 epochs was sufficient to get close to optimal without overfitting.

- **Batch size:** 32 reviews per batch.

- **Regularization:** We incorporated an EarlyStopping callback (patience 1 or 2 epochs) to stop training if validation performance stopped improving. This was triggered in some models (ensuring we didn’t over-train). We also used a dropout of 0.2 on the embedding layer output in some runs to improve generalization.

- **Initialization:** All RNN weights were initialized with Glorot uniform. We ensured reproducibility by setting a random seed for weight initialization and shuffling.

Each RNN architecture (Simple RNN, LSTM, GRU, Bidirectional LSTM, Stacked LSTM) was trained separately under the above conditions. Training on the full dataset (~hundreds of thousands of sequences) was time-consuming; to manage time, we also trained on a stratified **subset** of the data (e.g. 100k reviews) for some models to compare quickly, and verified that trends held when scaling up. The final reported results are from models trained on the full balanced training set. No hyperparameter search was done for RNNs beyond choosing the architecture types, as our goal was to compare these predefined architectures' performance rather than fine-tune each extensively. We did ensure that each network sees the same training/validation splits for fairness.

After training, we evaluated all models on the held-out 20% test set. This test data was not seen during model training or tuning, thus providing an unbiased measure of real-world performance.

9. Evaluation Results

We evaluated each model on the test set using multiple metrics: **Accuracy**, **Precision**, **Recall**, and **F1-Score** (for the positive class in particular, since it is a binary problem). Below we present the performance of the classical models and RNN models.

Classical ML Models Performance:
The table below summarizes the results of the classical machine learning classifiers on the test set:

Vectorizer	***Model***	***Accuracy***	***Precision***	***Recall***	***F1 Score***
TFIDF	*Logistic Regression*	*0.885404*	*0.876536*	*0.885404*	*0.876888*
BoW	*Logistic Regression*	*0.883668*	*0.874362*	*0.883668*	*0.874553*
TFIDF	*Multinomial Naive Bayes*	*0.874467*	*0.864011*	*0.874467*	*0.866011*
BoW	*Multinomial Naive Bayes*	*0.838795*	*0.865653*	*0.838795*	*0.848364*
TFIDF	*Decision Tree*	*0.853252*	*0.832878*	*0.853252*	*0.830477*
BoW	*Decision Tree*	*0.852234*	*0.831254*	*0.852234*	*0.827600*

Vectorizer	*Model*	*Accuracy*	*Precision*	*Recall*	*F1 Score*
BoW	*Random Forest*	*0.833181*	*0.855722*	*0.833181*	*0.760591*
TFIDF	*Random Forest*	*0.832852*	*0.855919*	*0.832852*	*0.759783*

***Key observations*:**

- **Best overall:** TF-IDF + Logistic Regression (Acc **0.8854**, F1 **0.8769**) with BoW + Logistic a very close second.
- **TF-IDF generally helps:** Both Logistic Regression and Multinomial NB post higher scores with **TF-IDF** than with **BoW**.
- **Naive Bayes is strong and lightweight:** TF-IDF + MNB ranks third (Acc **0.8745**, F1 **0.8660**), a good fast baseline.
- **Trees trail linears/NB:** Decision Trees are mid-pack (~0.853 Acc), while Random Forests are lower here (~0.833 Acc).
- **RF precision vs F1 gap:** RF shows **high precision** (~0.856) but **lower F1**, suggesting imbalance or weaker recall/thresholding compared to linear models.
- **Practical pick:** For accuracy-latency balance, **TF-IDF + Logistic Regression** is the recommended default; consider **TF-IDF + MNB** when you want a simpler, very fast alternative.

RNN Models Performance:
The deep learning models were evaluated on the same test set. Their results are summarized below:

Model	Accuracy	Precision	Recall	F1	ROC AUC
SimpleRNN	0.831601	0.691560	0.831601	0.755143	0.502099
LSTM	0.869318	0.842599	0.869318	0.854817	0.908275
GRU	0.874253	0.865221	0.874253	0.867397	0.916317
BiLSTM	0.874030	0.858481	0.874030	0.862400	0.916087
StackedLSTM	0.872082	0.858055	0.872082	0.859741	0.910885

Key observations:

- **GRU leads overall:** Highest accuracy (**0.8743**) and F1 (**0.8674**), with the best ROC AUC (**0.9163**), indicating strong separability and balanced precision/recall.
- **BiLSTM is a near-tie:** Accuracy (**0.8740**) and ROC AUC (**0.9161**) are essentially on par with GRU; a solid alternative if bidirectional context is preferred.
- **Stacking adds little: Stacked LSTM** is slightly below single **GRU/BiLSTM** (Acc **0.8721**, AUC **0.9109**), suggesting limited gains from deeper stacking on this dataset.
- **Plain LSTM is close behind:** Acc **0.8693**, F1 **0.8548**, AUC **0.9083** — competitive, simpler to train/deploy than BiLSTM.
- **SimpleRNN lags:** Lower accuracy (**0.8316**) and **near-random ROC AUC (0.5021)** → likely underfitting or struggling with sequence complexity/imbalance.
- **GRU** for best accuracy/AUC with efficient training; consider **BiLSTM** if capturing bidirectional context is key.

10. Best Model

Best overall model: TF-IDF + Logistic Regression

Highest accuracy and F1 across both tables; consistent, stable performance on sparse text features.

- **Metrics (test set):**
    - **Accuracy: 0.8854**
    - **Precision (weighted): 0.8765**
    - **Recall (weighted): 0.8854**
    - **F1-score (weighted): 0.8769**

Strong runners-up:

- **BoW + Logistic Regression** (Acc 0.8837, F1 0.8746) — very close to the winner.
- **TF-IDF + Multinomial NB** (Acc 0.8745, F1 0.8660) — lightweight, fast baseline.

Deep learning note:

- **GRU** leads among DL models (Acc 0.8743, F1 0.8674) with the **best ROC AUC 0.9163**, indicating strong class separability, but its accuracy is still below TF-IDF + LR.

Classification report (best model: TF-IDF + Logistic Regression)

- **Accuracy:** 0.8854
- **Precision (weighted):** 0.8765
- **Recall (weighted):** 0.8854
- **F1-score (weighted):** 0.8769

11. Interpretation & Takeaways

The comparative results yield several **insights**:

- **RNNs vs. Classical Models:** Deep learning models, particularly those with more sophisticated architectures (Bi-LSTM, Stacked LSTM), provided higher accuracy and better balance of precision/recall than classical ML models on this sentiment task. This implies that the sequential modeling of text gives an advantage. RNNs likely benefited from understanding the context of words in a sentence. For example, an LSTM can learn the importance of a negation word “not” affecting a later word “good”, whereas a bag-of-words based classifier might treat “not” and “good” independently. The classical models, while strong, assume feature independence (Naive Bayes) or linear separability in vector space (Logistic/SVM), which may falter on nuanced language constructs. The deep models also may capture subtle indicators of sentiment (tone, intensity) through the learned embeddings and recurrent dynamics.

- **Top Performing Model –**The Bidirectional LSTM’s success suggests that for tasks like review sentiment, having context from both directions is highly beneficial. Reviews can be long and often the sentiment is indicated towards the end (e.g., “Overall, we had a great stay despite some minor issues” – the true sentiment is positive, given by the clause after “despite”). A one-directional model reading left-to-right might be influenced by the “minor issues” part before getting to the conclusion. The Bi-LSTM, reading from both ends, can better capture such patterns. Additionally, Bi-LSTM essentially doubles the model’s capacity (two LSTMs) which might help capture more features of the data.

- **Classical Models Insight:** Among classical approaches, Logistic Regression and SVM performed extremely well considering their simplicity. This indicates that a **well-tuned TF-IDF representation** already contains most of the information needed to classify sentiment in many cases. Many words in the hotel domain have strong sentiment connotations (“dirty”, “poor” vs “excellent”, “fantastic”), so linear models can draw a fairly clean separation in this high-dimensional space. The success of these simpler models is encouraging: if computational resources or interpretability is a concern, a TF-IDF + Logistic Regression model could suffice with ~88% accuracy. In contrast, more complex classical models like Random Forest or XGBoost did not drastically outperform logistic regression – they were on par or slightly lower. This might be because the added non-linearity didn’t add much value beyond what combinations of individual word features already signaled. It’s also possible that with further tuning or engineered features, those ensemble models could improve, but at significantly greater training cost.

- **Error Analysis:** We examined some of the misclassifications to understand model limits:

- For the classical models, many errors occurred when reviews contained mixed sentiment or sarcasm. For instance, *"The room was great, but that was the only good part of our stay"* could confuse a bag-of-words model (equal presence of positive and negative words). The RNN models handled these better but could still err if the phrasing was complex or if the key sentiment cues were very subtle.
- Some negative reviews using polite or mitigated language (e.g., "The stay was okay, nothing special") sometimes were predicted as positive by simpler models due to lack of overt negative words. The RNN, however, learned from many examples that such lukewarm phrasing correlates with a negative or neutral sentiment and was less easily fooled.
- Extremely short reviews posed challenges: a review that simply says "Expected better." (negative implication) might be misclassified by any model if it hasn't seen enough similar phrasing. These sparse texts rely on context of common usage which models might not fully grasp.

- **Importance of Data Volume:** The sheer size of the dataset (half a million reviews) contributed to the robust performance of all models. With so much data, even relatively simple models can learn a lot of patterns. The RNNs in particular benefited from the large training set to train their many parameters effectively. This underscores that having a comprehensive dataset of domain-specific reviews is crucial for building a high-accuracy classifier.

- **Computation and Practicality:** Training the RNN models required more computational power and time compared to classical models. The logistic regression took minutes to train with TF-IDF, whereas the Bi-LSTM took hours (even with GPU acceleration). For practical deployment, one must consider this trade-off. If ~88% accuracy is acceptable, a logistic model is far more efficient to deploy (small memory footprint, fast inference). If we need the extra ~2-3% accuracy and better nuanced understanding (and are willing to handle a more complex deployment), the Bi-LSTM or similar deep model is justified. In scenarios where real-time classification is needed on a low-resource device, classical models might be preferred. On the other hand, for offline analysis of large batches of reviews, the deep model's higher accuracy could be worth the compute cost.

In summary, the project demonstrated that **both approaches have merit**: classical ML can give strong baseline performance with low complexity, while RNN-based deep learning pushes the accuracy higher by leveraging context and sequence information. The choice may depend on the specific application requirements.

12. Resources

- **Dataset Source:** Kaggle –‘*https://www.kaggle.com/code/jonathanoheix/sentiment-analysis-with-hotel-reviews/notebook’*. This dataset provided the raw reviews and ratings used in this project.
- **Relevant documentation for libraries**: scikit-learn, TensorFlow/Keras, and NLP preprocessing libraries (NLTK for stopwords, etc.) were referenced for implementation details.
- Software & Tools

  Language & IDE: Python 3.8+, Jupyter Notebook (or Google Colab)

  Libraries:

  - Data wrangling & visualization: pandas, numpy, matplotlib, seaborn
  - Classical ML: scikit-learn
  - Deep Learning: tensorflow / keras
  - NLP & Transformers: nltk

    Python version: 3.11.13

- **Software & Tools:** Programming Language: Python 3.8
- **Libraries**:
  pandas: 2.2.2
  scikit-learn: 1.6.1
  matplotlib: 3.10.0
  re: 2.2.1
  nltk: 3.9.1
  wordcloud: 1.9.4
  numpy: 2.0.2
  seaborn: 0.13.2
  xgboost: 3.0.4
  tensorflow: 2.19.0
  keras_tuner: Not Installed

imbalanced-learn: 0.14.0

transformers: 4.55.4

scipy: 1.16.1

tensorflow.keras: 3.10.0

- **Environment**: GPU-enabled Colab or local GPU for faster transformer training.

Individual Details

| **Name**- Chetna Mishra

| **E-mail ID** -sugarchins@gmail.com

| **Phone Number** - +91-7080801413
