AUTOMATED REVIEW RATING SYSTEM

PROJECT OVERVIEW

The Automated Review System processes and analyzes customer reviews to predict ratings or sentiment automatically. It cleans and preprocesses review data, splits it for training and testing to ensure reliable evaluation, and uses machine learning models to classify reviews. This system provides businesses with fast, data-driven insights into customer satisfaction, product quality, and areas for improvement, enabling better decision-making and enhanced customer experience.

ENVIRONMENTAL SETUP

The project is executed in Google Colab, which provides a cloud-based Python environment with free CPU and GPU resources. The setup includes:

- Python Version: Python 3.x (pre-installed in Colab)
- Data Handling & Preprocessing: pandas, numpy
- Text Processing & NLP: scikit-learn (for train-test split, feature extraction, and machine learning models)
- Visualization: matplotlib, seaborn (for plotting data distributions and results)

DATA COLLECTIONS

The dataset for this project was sourced from several publicly available Kaggle repositories, encompassing a diverse range of reviews for sentiment analysis:

- Amazon Reviews for Sentiment Analysis: A collection of Amazon product reviews in fastText format, facilitating sentiment classification tasks. kaggle kaggle
- ChatGPT Users' Reviews: User feedback on ChatGPT, including textual comments, ratings, and review dates, offering insights into user experiences. kaggle
- Sentiment Analysis Dataset: A dataset containing labeled sentiment data, suitable for training and evaluating sentiment analysis models. kaggle
- Reviews Dataset: A compilation of reviews from various industries, providing a broad spectrum of textual data for analysis. kaggle

The collected datasets, including Amazon product reviews, ChatGPT user reviews, and various sentiment-labeled review datasets, have been uploaded and made publicly available on GitHub for reproducibility and ease of access. github

DATA PREPROCESSING

In this stage, several cleaning and transformation steps were performed to ensure data quality and consistency.

Handling Missing Data

Missing values in the dataset were identified and appropriately handled to maintain data consistency and avoid bias during model training.

Removing Duplicates

Duplicate entries were removed to ensure that repeated reviews did not distort the analysis or affect the accuracy of the system.

Feature Engineering

New features such as review length and sentiment indicators were created to enhance the dataset's representation and improve model performance.

Removing Very Short Reviews

Reviews containing fewer than three words were removed, as they carry little semantic meaning and do not contribute effectively to the analysis.

Removing Special Characters and Emojis

Unnecessary special characters and emojis were eliminated from the text to standardize the review content and make it more suitable for text processing.

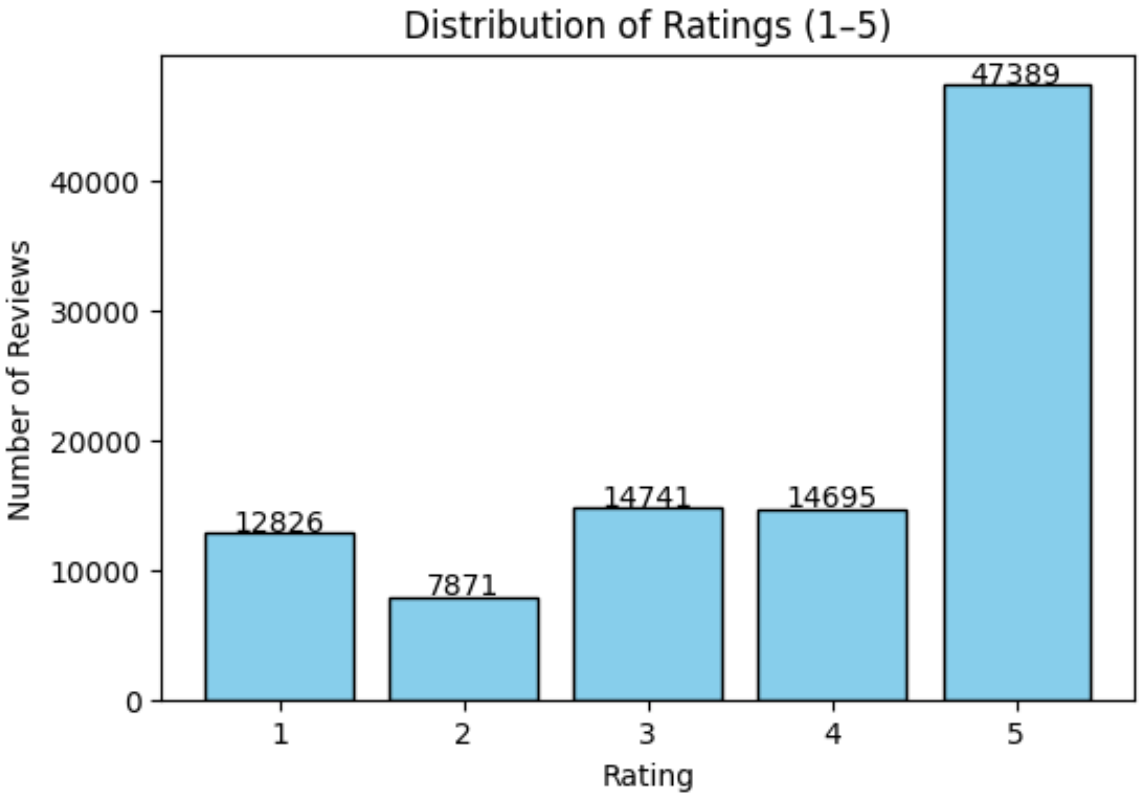
Removing URLs

URLs present in reviews were removed since they do not contribute to sentiment or review quality and could introduce noise into the dataset.

The cleaned and preprocessed dataset, ready for training and analysis, has been uploaded to GitHub for easy access. It includes all reviews with duplicates, very short entries, URLs, special characters, and emojis removed, along with added features like review length. The dataset can be accessed here. github

DATA VISUALIZATION

After preprocessing, a bar chart was generated to display the distribution of reviews across different rating categories. This visualization helps in understanding the balance of the dataset and ensures that preprocessing steps such as removing duplicates, short reviews, and special characters did not distort the class proportions.



CONVERTING TO LOWERCASE

All review text was converted to lowercase to ensure uniformity and reduce duplication caused by capitalization differences.

BALANCED DATASET

To ensure that the model is trained on an unbiased dataset, a balanced dataset was created by selecting 2000 reviews for each rating from 1 to 5. This resulted in a total of 10,000 reviews, ensuring equal representation across all rating categories.

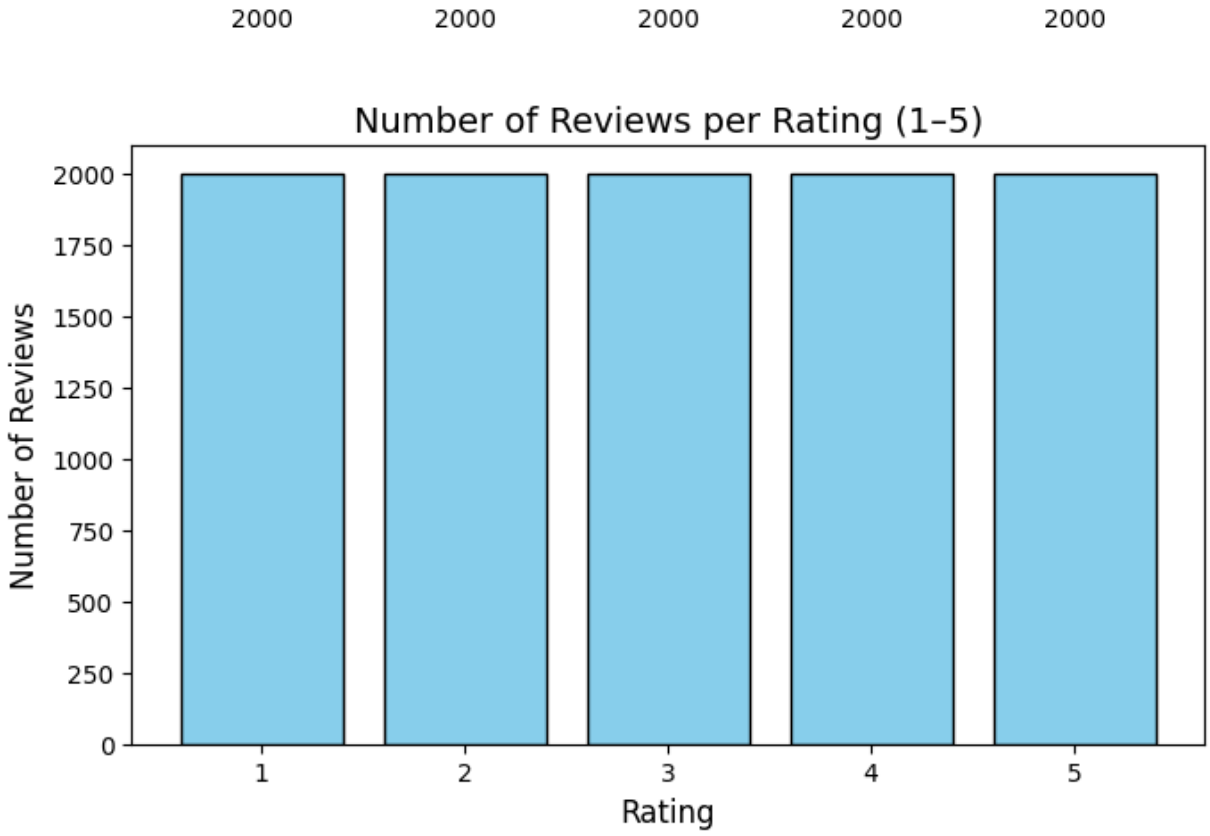
Code

```
# CREATE BALANCED DATASET (2000 each)
balanced_parts = []
for rating, remaining_df in zip(imbalanced_counts.keys(), remaining_parts):
    balanced_sample = remaining_df.sample(n=2000, random_state=42)
    balanced_parts.append(balanced_sample)

balanced_df = pd.concat(balanced_parts).sample(frac=1, random_state=42).reset_index(drop=True)
```

DATA VISUALIZATION

A bar chart was plotted to visualize the distribution of reviews in the balanced dataset. As expected, each rating category has an equal number of reviews, confirming that the dataset is balanced and ready for model training.



STOPWORD REMOVAL

After creating the balanced dataset, **stopword removal** was applied to clean the review text. Stopwords are common words (like "the", "is", "and") that do not carry significant meaning for sentiment or rating analysis. Negations such as “no”, “not”, and “never” were retained to preserve sentiment context. This step helps reduce noise in the dataset and improves the effectiveness of subsequent feature extraction and model training.

Code

```
# Load spaCy model (disable heavy components for speed)
nlp = spacy.load("en_core_web_sm", disable=["parser", "ner", "textcat"])

# Define custom stop words (keep negations)
stop_words = set(stopwords.words('english'))
negations = {"no", "not", "nor", "never", "n't"}
stop_words = stop_words - negations
```

Removed words

['a', 'about', 'above', 'after', 'again', 'against', 'all', 'am', 'an', 'and', 'any', 'are', 'as', 'at', 'be', 'because', 'been', 'before', 'being', 'below', 'between', 'both', 'but', 'by', 'can', 'd', 'did', 'do', 'does', 'doing', 'don', 'down', 'during', 'each', 'few', 'for', 'from', 'further', 'had', 'has', 'have', 'having', 'he', 'her', 'here', 'him', 'himself', 'his', 'how', 'i', 'if', 'in', 'into', 'is', 'it', 'its', 'itself', 'just', 'll', 'm', 'ma', 'me', 'more', 'most', 'my', 'myself', 'now', 'o', 'of', 'off', 'on', 'once', 'only', 'or', 'other', 'our', 'ourselves', 'out', 'over', 'own', 're', 's', 'same', 'she', 'should', 'so', 'some', 'such', 't', 'than', 'that', 'the', 'their', 'theirs', 'them', 'themselves', 'then', 'there', 'these', 'they', 'this', 'those', 'through', 'to', 'too', 'under', 'until', 'up', 've', 'very', 'was', 'we', 'were', 'what', 'when', 'where', 'which', 'while', 'who', 'whom', 'why', 'will', 'with', won', 'y', 'you', 'your', 'yours', 'yourself', 'yourselves']

Why is stemming preferred over lemmatization?

Stemming is often preferred over lemmatization when speed is more important than linguistic accuracy. Unlike lemmatization, which converts words to their correct dictionary forms using vocabulary and grammar rules, stemming simply chops off word endings to reduce words to a root form. This makes stemming faster and less resource-intensive, especially for large datasets, though it may produce non-standard or incomplete words. It is useful in applications like search engines or text classification where exact word forms are less critical.

Lemmatization

Lemmatization is a text preprocessing technique used to reduce words to their base or dictionary form, called a lemma. Unlike stemming, which may produce incomplete or non-dictionary words, lemmatization preserves the actual meaning of words. This is particularly useful in sentiment analysis and review processing, as it ensures that different forms of a word are treated consistently. For example, the words “running”, “ran”, and “runs” are all converted to “run”, and “better” is converted to “good”. By applying lemmatization, the model can better understand the semantic content of the reviews.

Code

```
def preprocess_doc(doc):
    tokens = [
        token.lemma_ for token in doc
        if token.is_alpha and token.text not in stop_words
    ]
    return " ".join(tokens)
```

SPLITTING THE BALANCED DATASET

After creating a balanced dataset with 2000 reviews for each rating, the dataset was split into training and testing sets to prepare for model development. 80% of the data was allocated for training the model, while 20% was reserved for testing its performance. This ensures that the model is trained on a representative sample while also allowing evaluation on unseen data to assess generalization.

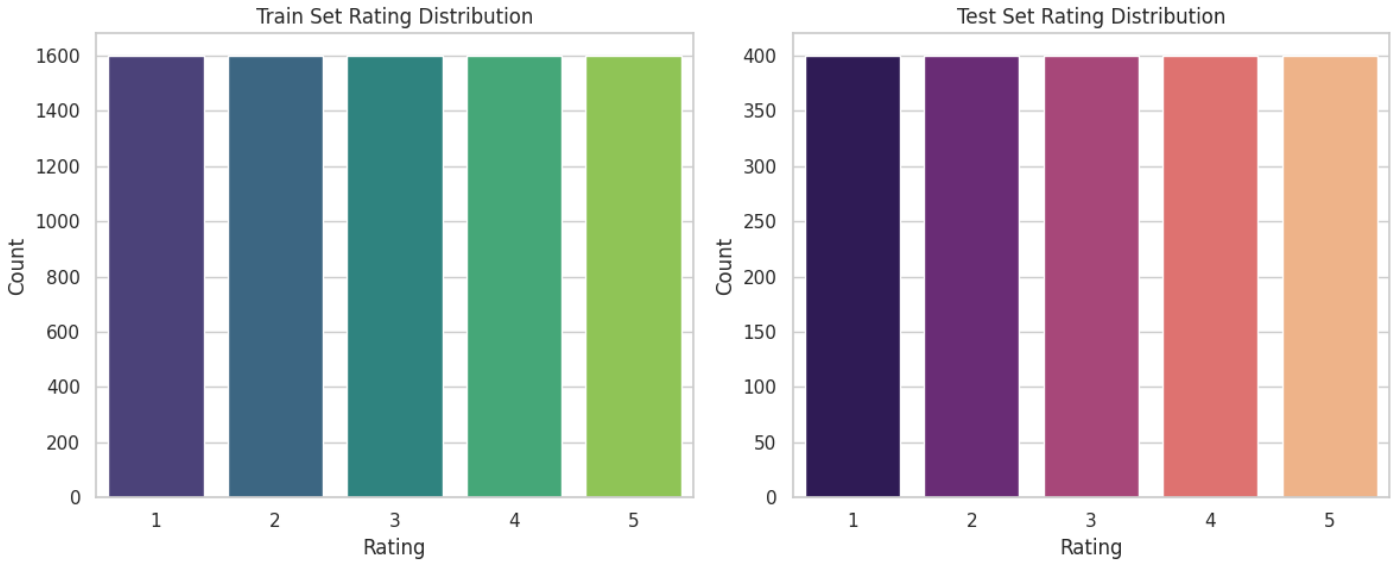
Code

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

The preprocessed train (80%) and test (20%) datasets are available on GitHub for access. github

DATA VISUALIZATION

To verify the distribution of reviews after splitting, a **bar chart** was plotted showing the count of each rating in the training and testing sets. The chart confirms that both sets maintain a proportional distribution of ratings, thanks to stratified splitting.



TF-IDF CALCULATION

Term Frequency–Inverse Document Frequency (TF-IDF) is a numerical statistic that reflects how important a word is to a document relative to a collection of documents (corpus). It helps highlight significant words while reducing the weight of common words. TF-IDF is calculated as:

$$\text{TF(t,d)} = \left(\frac{\text{Number of times term t appears in document d}}{\text{Total number of terms in document d}}\right)$$

$$IDF(t) = log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ containing\ term\ t}\right)$$

Code

```
tfidf = TfidfVectorizer(max_features=5000, stop_words='english')

# Fit only on training data
X_train_tfidf = tfidf.fit_transform(train_df['review'])

# Transform test data using the same vocabulary and IDF
X_test_tfidf = tfidf.transform(test_df['review'])
```