ML CLASSIFIER PROJECT – SENTIMENT ANALYSIS OF RIEVIEWS FOR THE "myWorldLink" APPLICATION

1. Introduction:

The "myWorldLink" application is widely used across Nepal for
Managing internet subscriptions, checking data usage, making payments,
And accessing various services. Since thousands of users interact with the app every day, many reviews are continuously posted on the app store.
These user reviews play an important role in improving the app. They are more than just star ratings-they act as a direct communication channel where users share their real experiences, problems, and suggestions. Reviews help the company understand issues such as bugs, slow performance, confusing features, or areas where users want improvements.
They also influence the app's reputation, visibility in the app store, and the public's trust in the brand.

Sentiment analysis helps make sense of these large amounts of feedback by automatically identifying whether a review is positive, negative, or neutral. This allows developers to go beyond simple star ratings and understand the actual emotions behind user comments, can make better decisions, prioritize important fixes and improve the overall user experience of the myWorldLink application.

Our system is engineered in distinct, reliable stages, like **Ingestion & Cleaning** where we fetch raw reviews from the Google Play Store, immediately filter out non-English reviews (specifically Nepali/Devanagari text) to maintain data quality, and apply essential NLP cleaning (removing emojis, stop words).
Labeling & Feature Engineering, where review star ratings are converted into a clear sentiment label (Positive or Negative). Concurrently, a rule-based extractor identifies and separates specific customer suggestions (e.g., "please fix", "need to improve") from general complaints.
Modeling the cleaned, labeled text is used to train a Logistic Regression classifier. This model is saved and reused to instantly predict the sentiment and confidence for any new incoming review.

2. Objectives:

The primary objective of this project is to build a **sentiment analysis using Machine Learning** to monitor and analyze customer sentiment from app reviews. The system provides two main deliveries:

1. **Sentiment Classification:** Categorizing reviews as **POSITIVE** or **NEGATIVE** with confidence score.

2. **Suggestion Extraction:** Identifying and extracting actionable feature requests or bug reports.

3. Scope and Limitations:

1. In-Scope (What the model Does**):**

- **Binary Sentiment Classification:** The model's primary function is to categorize incoming reviews into a clear, measurable binary label (POSITVE or NEGATIVE), providing a high satisfaction score with a measurable confidence level.
- **Actionable Insight Extraction:** It specifically flags and extracts reviews that contain explicit calls to action (bug reports, feature requests) using rule-based logic. These isolates immediate tasks for the product team.
- **Reporting**: The pipeline output (sentiment_predictions_final.csv) is a highly structured, machine-readable dataset. This moves the business away from manually reading text files and toward automated dashboards and data-driven prioritization.
- **Operating Efficiency:** The entire system is built as an automated pipeline, meaning the scope includes handling data from ingestion to final prediction without manual intervention.

2. In-Scope (Benefits for the "myWorldLink" App):

1. App Stability and Bug Prioritization: The most direct benefit is the automated identification of critical app defects:

- **Real-time Bug Reporting:** The Suggestion Extraction module automatically flags and extracts bug reports (e.g., "app crashes on open", "verification code not received", "device filtering doesn't work"). This eliminates the need for manual review, drastically speeding up the time-to-fix.
- **Prioritized Backlog:** By associating the negative sentiment scores with specific bugs, the system allows the development team to prioritize fixes based on the issues causing the greatest user dissatisfaction, rather than just the newest ones.

2. Service Quality Monitoring (Via App Feedback): Although the reviews are for the app, they serve as a critical proxy for overall ISP service health:

- **Quantifiable Dissatisfaction:** The model converts general complaints like "slow internet speed" or "worst service" into a **measurable Negative Sentiment Score**. This provides a daily or weekly KPI on the state of the core service quality as perceived by the users.
- **Early Warning System:** A sudden spike in reviews flagged as **NEGATIVE** and relating to core service terms (like "slow," "lag," "disconnect") acts as an early warning for network-wide or regional service degradation.

3. Feature Development and Usability: The model guides future application development based on actual user demand:

- **Validated Feature Requests:** The **Suggestion Extraction** module highlights frequently requested features (e.g., “I wish I could change my WIFI password in the app,” “Need a payment portal”). This helps the product manager validate and prioritize new features with confidence.
- **Usability Feedback:** Reviews often highlight usability issues (e.g., “UI is confusing,” “Can’t report trouble”). The combined Sentiment and Suggestion data points to areas of the app that are causing user friction, guiding UX/UI improvements.

4. **Operational and Customer Support Efficiency:**

- **Proactive Support:** By understanding the most common pain points identified by the model, the customer support can be proactively trained and equipped with solutions for the top 5-10 issues (e.g., “The network issue is caused by the latest firmware update.”).
- **Resource Allocation:** The structured reports provide the data necessary to justify resources (developer time, server upgrades) to the most problematic areas of the app and service.

3. **In-Scope (**How is this model useful for other organization?**):**

1. **General Usability:**

- **Automation of Customer voice:** This sentiment Analysis Pipeline offers a compelling value proposition to any organization handling customer text feedback by automating the “Customer Voice” and transforming vast, unstructured data into actionable, quantitative business metrics.
- **Prioritization Engine:** The Suggestion Extraction module is a huge benefit to any product or Engineering team. It instantly filters the noise, allowing managers to prioritize feature backlogs and bug queues based directly on quantifiable user demand and sentiment.
- **Benchmarking and KPI Tracking:** Any business can use the pipeline to create a **Sentiment KPI** that tracks customer satisfaction over time (e.g., “Our monthly Positive Sentiment Score dropped from 75% to 68%” after the last app update”).

2. **Industry-Specific Examples:**

This model can be useful for other organizations for dealing with feedback.

Industry	Data Source	How the model is useful
Ecommerce/Retail	Product-reviews, Chat Transcriptions	**Detecting Product Flaws:** Identifying and prioritizing complaints regarding product quality ("Ripped after two days") or fulfillment issues ("Shipping was late") to reduce returns and improve sourcing.
SaaS / Software	Support-Tickets, Feature Requests	**Improving Product Adoption:** Flagging confusing UI elements or critical bug reports in real-time, allowing development teams to focus sprint cycles on customer-blocking issues.
Financial Services	Banking-App, Reviews, Call Center Notes	**Compliance & Risk Mitigation:** Monitoring sentiment around key topics like "security" or "transaction fees" to proactively address potential compliance or reputation risks.
Healthcare	Patient Surveys, Portal Feedback	**Service Quality:** Analyzing feedback on wait times, staff friendliness, or portal usability to improve the overall patient experience.

3. Limitations:

1. Limitations of this model for "MyWorldLink" App:

The model is built on traditional Machine Learning techniques that primarily rely on word frequency and co-occurrence, not linguistic meaning.

- **Sarcasm and Negation:** The model struggles to correctly classify reviews that use sarcasm or double negation. For example, a review like, "The app is great... if you enjoy waiting ten minutes for it to load," will likely be misclassified as **POSITIVE** because the positive word "great" has a high weight, overpowering the negative context.
- **Simple Association:** The model treats words as individual features. It cannot grasp the difference between "The speed is **not** good" and "The speed is good," making subtle differences in customer intent hard to distinguish.

2. No Aspect-Based Sentiment Analysis (ABSA)

The model can tell you a review is **NEGATIVE**, but it cannot tell you *why* it's negative with precision.

- **Missing Specificity:** It cannot separate sentiment by topic. For example, a review stating, "The customer service is great (POSITIVE), but the payment portal is broken (NEGATIVE)," is simply classified as a single sentiment (e.g., NEUTRAL or POSITIVE), masking the critical issue with the payment portal.
- **Hinders Root Cause Analysis:** This lack of specificity forces product managers to still manually read the final flagged reviews to determine the exact *feature* or *area* of the app/service that caused the negative sentiment.

3. Rigidity of Suggestion Extraction Logic

The current system uses a **rule-based extractor** (suggestion_extractor.py), which creates dependency and maintenance overhead.

- **Manual Maintenance Required:** If Worldlink users start using new slang, new terms, or new features that require improvement (e.g., switching from "fix bug" to "patch this"), the pipeline will miss this new feedback until the suggestion_extractor.py code is manually updated and redeployed.
- **Inability to Categorize:** The extractor only flags a review as a "suggestion." It does not categorize the type of suggestion (e.g., is it a 'Billing' issue, a 'Network' issue, or an ‘UI' improvement?). This limits its ability to feed directly into specific Worldlink Engineering teams.

4. Language Barrier for Multilingual Data

Although the initial design included a filter for Nepali (Devanagari) script, forcing the model to read both English and Nepali without a sophisticated multilingual architecture introduces severe accuracy issues.

- **Tokenization Conflict:** The current **TF-IDF Vectorizer** is trained on English tokens and stopwords. When Nepali text is passed through it, the model either discards the text as irrelevant noise or incorrectly weights the Nepali tokens, drastically lowering the reliability of sentiment scores for all languages.
- **Data Skew:** To reliably include Nepali, you would need a large corpus of **labeled Nepali reviews**, which is often difficult and expensive to acquire, skewing the model's performance toward the dominant language (English).

2. Limitations of this model for other organizations:

This model performs simple, document-level sentiment (Is the entire review POSITIVE or NEGATIVE). Other industries demand Aspect-Based Sentiment Analysis (ABSA), which this model cannot perform.

1. Failure to Capture Aspect-Specific Sentiment

- **Problem:** The model cannot link sentiment to specific features, products, or services mentioned in the text.
- **Industry Example (E-commerce/Retail):** A customer review states: "The product quality is excellent, but the shipping service was terrible."
- **Model Result:** This model will likely output NEUTRAL or a weak POSITIVE/NEGATIVE, missing the core insights.
- **Needed Insight**: Retailers need to know: Sentiment for Product Quality is POSITIVE; Sentiment for Shipping/Logistics is NEGATIVE. This current model cannot separate these two concepts.

2. Low Accuracy on Domain-Specific Nuance and Sarcasm:

The TF-IDF/Logistic Regression approach struggles with the complex language and context found in diverse industries.

- **Problem:** The model relies on word frequency. In domains where language is subtle or technical, the model fails to interpret meaning.
- **Industry Example (Hospitality/Travel):** A guest review states: "The view from the room was 'breathtakingly average,' but the location was perfect."
- **Model Result:** The model may be confused by the positive word "breathtakingly," leading to a misclassification of the view's sentiment.
- **Industry Example (Finance/Tech**): Reviews often contain industry-specific jargon. The model might miss the negative sentiment hidden in phrases like "The API integration is cumbersome" or "The transaction latency is unacceptable," as these words might not carry strong negative weight in the initial training data.

3. Rigidity of Rule-Based Suggestion Extraction ⚙️

The system's most valuable feature, the **Suggestion Extraction** modules, are highly rigid and require manual maintenance for every new client.

- **Problem:** The list of action-oriented keywords (like "fix," "improve") in the suggestion_extractor.py must be manually tuned for every new domain to be effective.
- **Industry Example (SaaS/B2B Software):**
- **ISP Keywords:** fix login, improve speed.
- **SaaS Keywords:** The model needs to recognize requests tied to specific product features, like add integration, allow bulk upload, or export function is broken. If these are not in the rule list, the feature requests are entirely missed.

4. Requirement for New Labeled Training Data

While the model is generalized, it is currently "tuned" by the MyWorldLink data, meaning its understanding of sentiment is skewed towards ISP contexts.

- **Problem:** The same words can have different sentimental weights across industries (**Semantic Drift**).
- **Industry Example (Food/Restaurant):** The word "burnt" is overwhelmingly **NEGATIVE** in a food review. In a review for a gym, "burnt" (as in calories) is overwhelmingly **POSITIVE**. Deploying the ISP-trained model (which hasn't seen the food context) on a restaurant reviews would require acquiring and labelling a new dataset to recalibrate the model's weight and maintain high accuracy.

4. Tools and Technologies Used:

Category	Tool/Library	Purpose/Description
Programming Language	Python 3.13	Crore programming language used for the entire project
Data processing	Pandas, NumPy	Data manipulation and cleaning
Natural Language Processing (NLP)	nltk, emoji, re(regex)	Tokenization, stopword removal and emoji handling
Machine Learning	Hugging face transformers	Pre-trained transformer model for sentiment classification
Database and database Handling	SQLite3, pandas	Local database for persistent review and suggestion storage, data manipulation, transformation, and persistent storage.
Concurrency	ThreadPoolExecuter	Enhances performance by parallelizing review processing
Development Environment	Visual Studio Code	Main IDE used for writing, debugging, and testing
Version Control	Git & GitHub	Source code tracking and collaborative development
Data Ingestion	Google_play_scraper	Fetching live app reviews
Vectorization	TfidfVectorizer	Converting text into numerical features for the model.

5. System Architecture

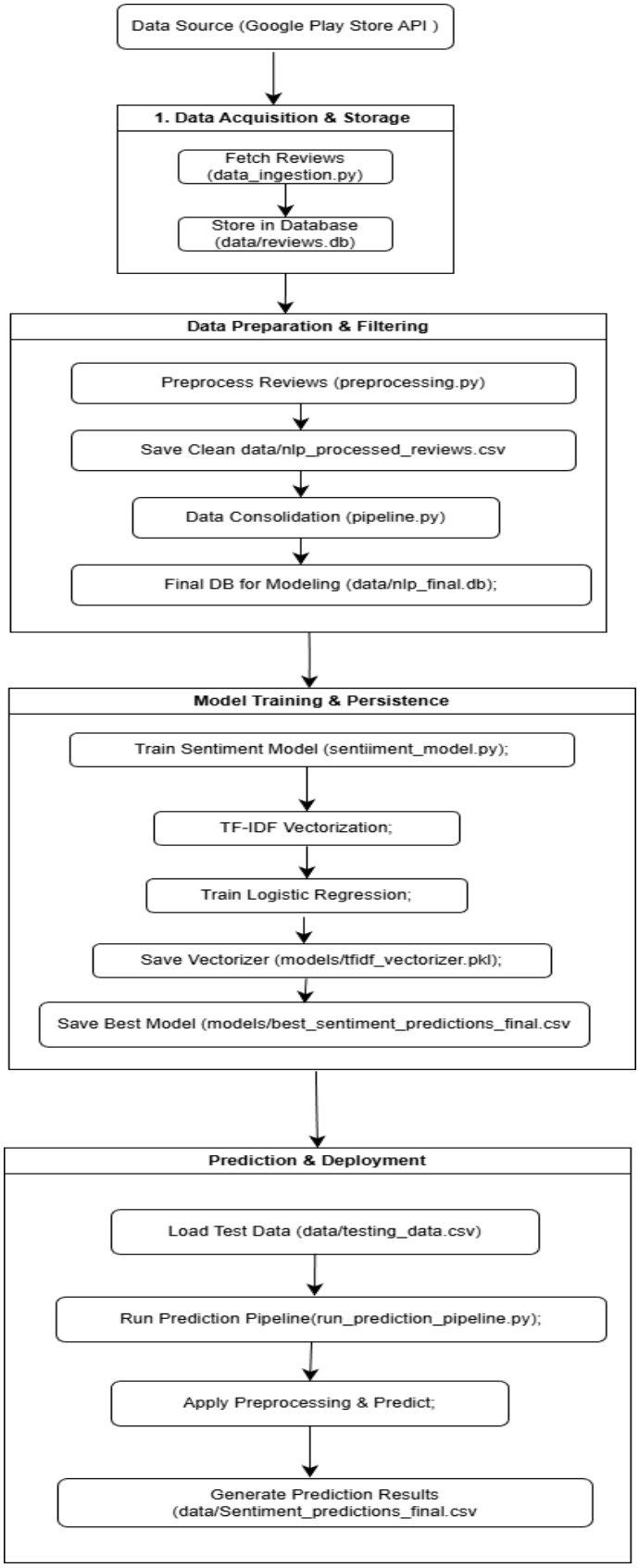

Figure: - *Architecture Diagram Of ML-Classifier Project.*

6. Development Flow (Continuous Refinement):

The project followed an iterative and modular development approach:

1. **Modularity:** Each major step (Ingestion, preprocessing, modeling) was isolated into its own Python file (py), ensuring code is reusable and easy to debug.
2. **Robustness & Persistence:** Database integration (database_utils.py) was prioritized to ensure data integrity. Raw data is saved immediately upon fetching, and finally clean data is saved before modeling, making the pipeline fully restart able at any stage.
3. **Refactoring for Deployment:** Critical fixes were applied to the final prediction script (run_prediction_pipeline.py) to address Unicode errors and, most importantly, robustly identify the correct review column in the test data, ensuring the model predicts on actual text, not on extraneous IDs.

7. Technical Implementation: Component Breakdown

The pipeline is organized into seven core scripts, executed sequentially to move data from raw text to final prediction.

1. **Data Ingestion** (data_ingestion.py, database_utils.py)
   - **Input:** Google play Store API (via Google_play_scraper).
   - **Process:**

➢ Data_ingestion.py: fetches reviews for a specified app_id and saves them using database_utils.py.

➢ Database_utils.py: creates and manages a SOLite database(data/reviews.db) to store the raw reviews, ensuring a permanent, untouched record of the source data.

- **Output:** data/reviews.db (Raw data table).

2. **Preprocessing & Feature Engineering** (preprocessing.py, suggestion_extractor.py)
   - **Input:** Raw reviews from data/reviews.db.
   - **Process:**

➢ **text cleaning:** Text is lowercased, and emojis, punctuation, and English stopwords are removed.

➢ **Non-English Filtering**: A dedicated check(is_nepali) is used to filter out reviews containing Devanagari script, ensuring the model is trained only on the target language (English).

➢ **Labeling**: Review Scores(ratings) are mapped to POSITIVE or NEGATIVE labels to create the target variable for training.

➢ **Suggestion Extraction:** It uses rule-based matching (regex keywords like ‘fix’, improve’, ‘should’) to flag and extract feature suggestions.

- **Output:** The final, clean, and labeled data stored in data/nlp_processed_reviews.csv.

3. **Data Consolidation (pipeline.py)**
   - **Input:** data/nlp_processe_reviews.csv.
   - **Process:** This script is a final data validation and persistence step. It loads the processed csv, filters out only the required cleaned_text and sentiment columns, and saves this essential set into the final modeling database, data/nlp_final.db.
   - **Output:** data/nlp_final.db (Clean, modeling-ready table).

4. **Prediction/Deployment (run_prediction_pipeline.py)**

   - **Input:** Unseen or test data (data/testing_data.csv).

   - **Process:** This script acts as the final deployment interface.
   - **Asset Loading:** Loads the saved model.pkl and vectorizer.pkl.
   - **Preprocessing Consistency:** Applies the **exact same** cleaning functions from preprocessing_for_prediction.py to the new data, ensuring consistency between training and prediction environments.
   - **Prediction:** Generates sentiment label (POSITIVE/NEGATIVE) and a confidence score.
   - **Evaluation:** Calculates prediction **Accuracy** against the actual labels in the test data.
   - **Output: data/sentiment_predictions_final.csv** (Final report).

Conclusion:

The Sentiment Analysis pipeline successfully delivers a modular, five-stage ML system that automates the conversion of high-volume, unstructured customer reviews into actionable data using Logistics Regression and TF-IDF. Its primary value is providing a quantifiable Sentiment KPI (Key performance Indicator) and performing Suggestion Extraction, allowing “MyWorldLink” app and other organizations to prioritize critical bugs and features request based on data, moving from reactive to proactive product management.