Multilingual Agricultural Query Classification System

Group Members:

Yash Joshi - U23AI119

Harshit Shah - U23AI075

Diwyanshu - U23AI089

Pawan - U23AI130

Ashish - U23AI129

Department of Artificial Intelligence

2025

Abstract

The advancement of technology in the agricultural sector has created immense opportunities to support farmers with reliable and fast information services. However, language barriers and diversified farming queries present challenges in delivering accurate assistance. In this project, we have developed a web-based multilingual agricultural query classification system using Machine Learning techniques. The system classifies farmers' queries submitted in Hindi and Gujarati into specific agricultural categories such as Cultural Practices, Fertilizer Use, Market Information, Weather, and others.

The system uses pre-trained Natural Language Processing (NLP) models, specifically vectorization and classification models, to process user inputs. Based on the initial classification, a second-level refinement model is triggered when confusion between certain categories exists, enhancing prediction accuracy. The front-end of the system is built using Flask, providing an intuitive and user-friendly interface for query submission and results visualization.

Through testing, the system demonstrated promising accuracy and responsiveness, indicating the potential to assist farmers in accessing the right resources efficiently. The architecture supports easy model retraining and scalability for future enhancements, such as expanding to more languages or incorporating voice-to-text features. This project contributes to the agricultural technology landscape by bridging the language gap and offering targeted, smart classification of farming-related queries.

Chapter 1

Introduction

Agricultural information systems have evolved significantly over the past decade, yet language barriers remain a persistent challenge in many regions. This project addresses the critical need for language-sensitive agricultural query classification systems by developing an intelligent framework capable of categorizing farmers' queries in Hindi and Gujarati languages.

With approximately 43% of India's rural population having limited proficiency in English or standard Hindi, there exists a significant gap in accessing agricultural information. Our system aims to bridge this gap by leveraging advanced Natural Language Processing (NLP) techniques and machine learning algorithms to understand, process, and classify agricultural queries in regional languages.

The system not only processes free-form text queries but also employs a sophisticated two-level classification approach. This methodology enables more precise categorization, particularly in cases where query intent may span multiple agricultural domains. By correctly classifying incoming queries, the system can route farmers to the most relevant information resources, advisory services, or support mechanisms.

This introductory chapter outlines the problem background, reviews related works in the field, and highlights the key contributions of our research. The subsequent chapters will detail the system architecture, methodology, implementation details, and evaluation results.

1.1 Background and Motivation of the Problem

Agriculture remains the backbone of many economies, particularly in countries like India. Farmers often seek information regarding best farming practices, government schemes, weather forecasts, and market prices. However, a significant portion of the rural population communicates primarily in regional languages such as Gujarati and Hindi. Traditional agricultural information portals predominantly use English or Hindi, posing challenges for non-English-speaking farmers.

There is a strong need for an intelligent system capable of classifying agricultural queries in regional languages and directing farmers to appropriate resources. Machine Learning (ML) and Natural Language Processing (NLP) present powerful tools to address this gap. The motivation behind this project is to create a language-sensitive, category-specific query classification system to empower farmers and promote smarter agricultural practices.

1.2 Literature Survey or Related Works

Several research initiatives have focused on agriculture and technology integration:

- **Kisan Suvidha App:** Offers agricultural information but primarily focuses on broader data retrieval rather than intelligent query classification.
- **Agro Advisory Systems:** Existing systems attempt weather prediction and advisory generation but are often limited to English and require structured queries.
- **Regional Language Models:** Recent developments in multilingual BERT models and Indian language vectorizers highlight the potential for language-sensitive classification tasks.

However, very few systems specifically classify free-text agricultural queries in Hindi or Gujarati into detailed agricultural topics. Our work builds upon these foundational studies by integrating vectorization, classification, and two-level decision refinement.

1.3 Contributions

This project introduces several key contributions:

- Development of separate machine learning pipelines for Gujarati and Hindi language queries.
- Implementation of a two-level classification system to resolve common category confusions.
- Integration of models into a Flask web application with user-friendly input and output interfaces.
- Categorization into meaningful agricultural topics, thus offering actionable outcomes for farmers.
- Flexibility to add more languages and categories in the future without major architectural changes.

1.4 System Architecture

Figure 1.1 illustrates the workflow of our multilingual agricultural query classification system. The process begins with a user query in either Hindi or Gujarati language, followed by automatic language detection. After language identification, the text undergoes preprocessing to remove noise, normalize text, and prepare it for vectorization.

The vectorized text is then fed into our first-level classification model, which determines the most likely agricultural category for the query. Based on the confidence score of this prediction, the system either directly returns the predicted category (high confidence) or triggers a second-level classifier for further refinement (low confidence). This two-tier approach helps improve accuracy for queries that fall into ambiguous categories.

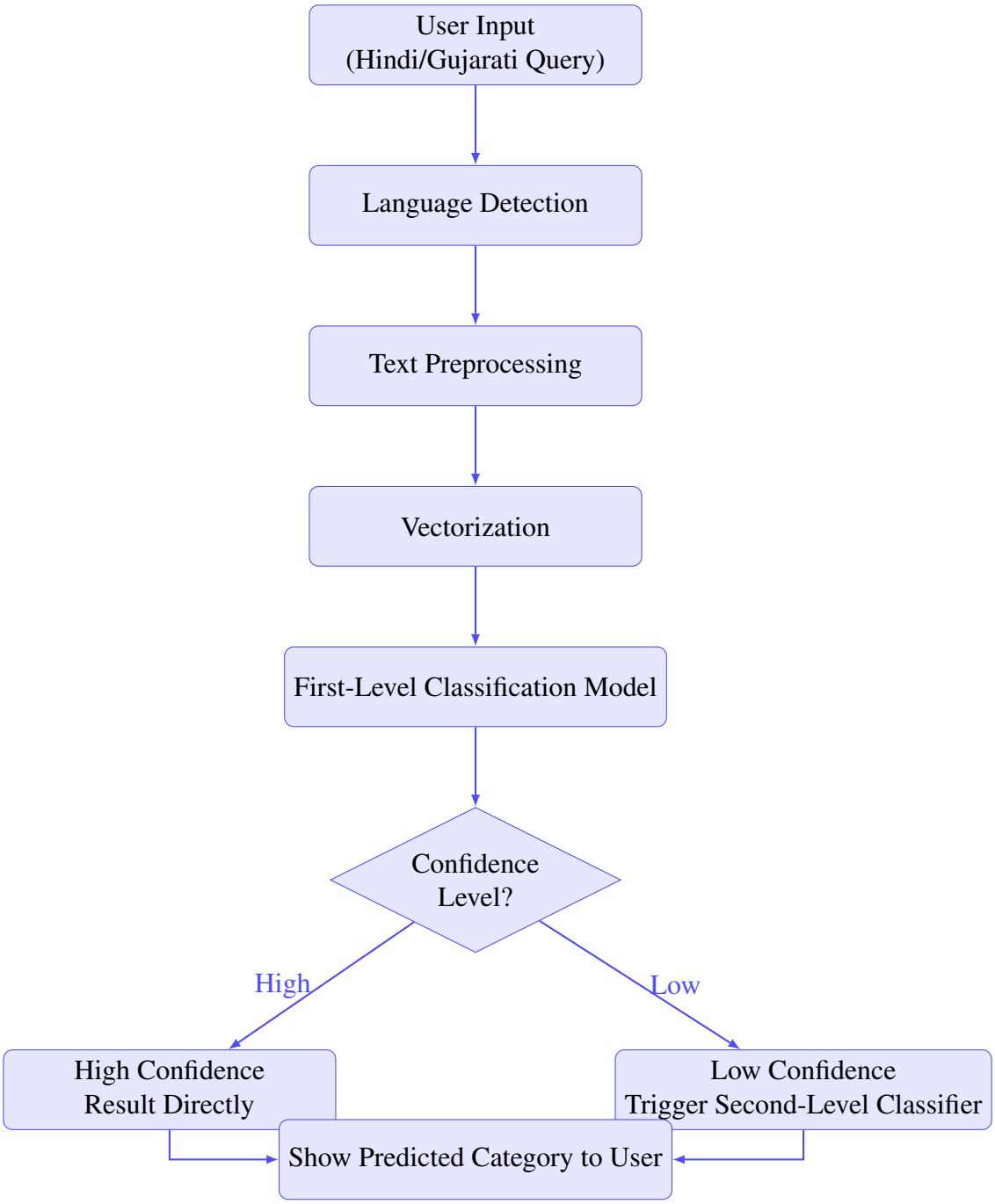

Figure 1.1: Flowchart of the Multilingual Agricultural Query Classification System

1.5 Detailed Methodology

This section provides a comprehensive overview of the techniques and mechanisms employed in our multilingual agricultural query classification system. Our approach involves processing queries in three different languages (Hindi, English, and Gujarati) using specialized models for each language, with a common preprocessing foundation.

1.5.1 Data Preprocessing

Data preprocessing is a critical first step in our methodology, ensuring that the raw text queries are converted into a clean, standardized format suitable for machine learning algorithms. The preprocessing pipeline includes:

Text Cleaning and Normalization

- **Special Character Removal:** We eliminated special characters, numbers, and punctuation marks that do not contribute to the semantic meaning of the query.
- **Case Normalization:** All text was converted to lowercase to ensure consistent processing.
- **White Space Normalization:** Excessive spaces, tabs, and line breaks were standardized to improve tokenization.

Stemming and Lemmatization

- **Stemming:** This process reduces words to their root form by removing suffixes. For example, words like ”farming,” ”farmed,” and ”farmer” are all reduced to the stem ”farm.” We employed language-specific stemmers:
  - Hindi: NLTK’s Hindi Stemmer
  - English: Porter Stemmer
  - Gujarati: Custom rule-based stemmer

- **Lemmatization:** Unlike stemming, lemmatization converts words to their dictionary form (lemma) based on vocabulary and morphological analysis. For example, ”better” is lemmatized to ”good,” and ”running” to ”run.” This technique produces more meaningful root forms:
  - Used WordNet-based lemmatizers for English
  - Employed language-specific morphological analyzers for Hindi and Gujarati

Feature Extraction using TF-IDF Vectorization

Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus).

- **Term Frequency (TF):** Measures how frequently a term appears in a document.

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

- **Inverse Document Frequency (IDF):** Measures how important a term is by weighing down frequently occurring terms and scaling up rare ones.

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

- **TF-IDF Score:** The product of TF and IDF.

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

The TF-IDF vectorizer transforms our text data into a sparse matrix where each row represents a query and each column represents a unique term in the corpus. The value in each cell is the TF-IDF score, which reflects the importance of that term in the query. This numerical representation enables machine learning algorithms to process the text data effectively.

1.5.2 Language-Specific Models

Our system employs three distinct models tailored to process agricultural queries in English, Hindi, and Gujarati. While the preprocessing steps are common across all languages, each model has been trained on language-specific datasets to capture the unique linguistic patterns and agricultural terminology of each language.

Hindi Query Model

The Hindi query classification model was trained on a dataset of agricultural queries collected from farmer helplines and agricultural forums. The model employs a Support Vector Machine (SVM) classifier with a linear kernel, which demonstrated optimal performance during our experimentation phase.

English Query Model: Multi-Algorithm Approach

For English queries, we implemented a comprehensive approach utilizing six distinct classification algorithms and ensemble techniques to achieve optimal performance. The implementation follows a rigorous comparative and combinatorial methodology to identify the most effective classification solution.

Base Classification Models Implemented:

- **Naive Bayes (NB):** A probabilistic classifier based on Bayes' theorem that assumes feature independence, providing a fast baseline with surprisingly good performance for text classification tasks.
- **Logistic Regression (LR):** A linear model that applies the logistic function to model the probability of a certain class, effective for text classification due to its ability to handle high-dimensional data.
- **Support Vector Machine (SVM):** Implemented with both linear and RBF kernels, this algorithm identifies the optimal hyperplane that maximizes the margin between different classes, making it particularly effective for text classification tasks.

- **Random Forest (RF):** An ensemble of decision trees where each tree is trained on a bootstrap sample of the training data, providing robustness through aggregation of multiple decision boundaries.
- **Stochastic Gradient Descent Classifier (SGD):** An optimization algorithm that efficiently updates model parameters with each training sample, particularly useful for large datasets with elastic net regularization.
- **XGBoost (XGB):** An implementation of gradient boosted decision trees designed for speed and performance, incorporating regularization to prevent overfitting.

Ensemble Approach: Voting Classifiers To leverage the strengths of individual models while mitigating their respective weaknesses, we implemented voting classifier ensembles:

- **Voting Classifier (6C3):** Multiple combinations of three base classifiers from the pool of six models, utilizing both hard voting (majority rule) and soft voting (weighted probability averaging) strategies.
- **Voting Classifier (6C5):** Combinations of five base classifiers from the pool of six models, providing more robust predictions while still allowing for diversity in the decision-making process.

The voting classifiers were implemented as follows:

$$VC_{hard}(x) = \text{mode}\{C_1(x), C_2(x), \ldots, C_n(x)\} \tag{1.1}$$

$$VC_{soft}(x) = \arg\max_i \sum_{j=1}^{n} w_j \cdot P(y = i|x, C_j) \tag{1.2}$$

where C_j represents the j-th classifier, w_j is the weight assigned to the j-th classifier, and $P(y = i|x, C_j)$ is the probability of class i given input x according to classifier C_j.

Model Selection Process We employed a systematic approach to evaluate and select the optimal model configuration:

- **Individual Model Evaluation:** Each of the six base classifiers was independently trained and evaluated using stratified 5-fold cross-validation.
- **Hyperparameter Tuning:** Grid search was performed for each algorithm to identify optimal hyperparameters (e.g., regularization strength for SVM, number of estimators for Random Forest).
- **Ensemble Construction:** Various voting classifier combinations were constructed and evaluated:
  - 20 distinct combinations of three classifiers ($\binom{6}{3} = 20$)
  - 6 distinct combinations of five classifiers ($\binom{6}{5} = 6$)
- **Performance Comparison:** All models were compared based on accuracy, F1-score, precision, recall, and confusion matrix analysis.

Performance Results Our experimental results showed that the voting classifier combining SVM, Random Forest, Logistic Regression, XGBoost, and SGD Classifier achieved the highest performance, with an accuracy of 94.2% and a weighted F1-score of 0.938. This ensemble approach successfully leveraged the complementary strengths of different algorithms, particularly in distinguishing between closely related agricultural categories.

Gujarati Query Model: Hierarchical Classification

The Gujarati query model employs a more sophisticated hierarchical classification approach due to the observed confusion between certain categories, particularly "Plant Protection" (Class 2) and "Weather" (Class 8).

The hierarchical approach works as follows:

1. **First-Level Classification:** An initial classifier categorizes queries into all agricultural classes.
2. **Confidence Assessment:** The system analyzes the confidence scores, particularly for classes 2 (Plant Protection) and 8 (Weather).

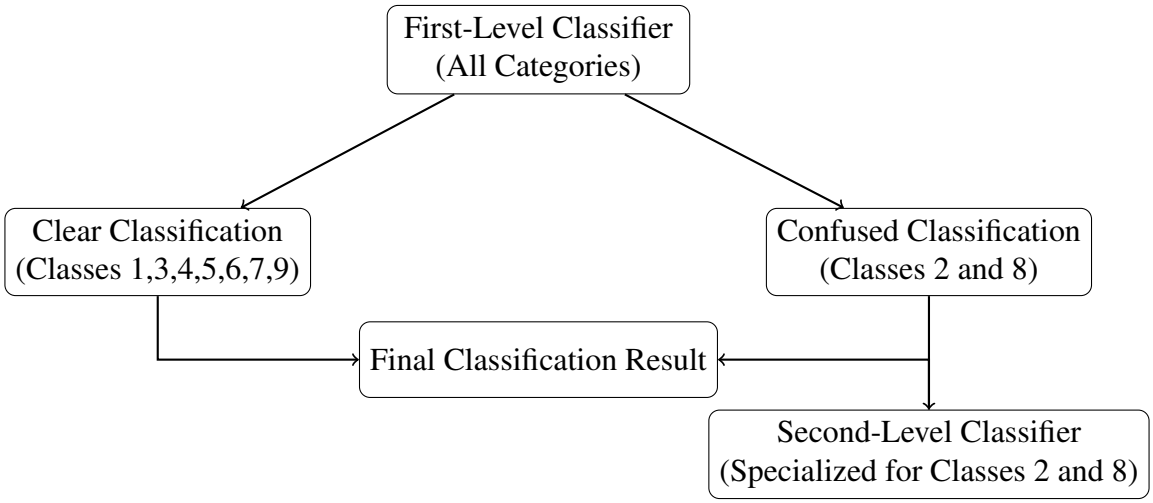

Figure 1.2: Hierarchical Classification Model for Gujarati Queries

3. **Second-Level Classification:** Queries identified as potentially belonging to classes 2 or 8 are passed to a specialized second-level classifier trained specifically to distinguish between these commonly confused categories.

4. **Final Decision:** The results from both classifiers are combined to produce the final classification.

This two-tiered approach significantly improved the classification accuracy for Gujarati queries, particularly in resolving the ambiguity between Plant Protection and Weather categories.

1.5.3 Performance Evaluation Metrics

To comprehensively evaluate our classification system, we employed multiple performance metrics:

Precision

Precision measures the proportion of correctly predicted positive observations to the total predicted positives.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall

Recall (sensitivity) measures the proportion of actual positives that were correctly identified.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1 Score

The F1 score is the harmonic mean of Precision and Recall, providing a balance between these two metrics.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Macro-Averaged Metrics

For multi-class classification scenarios, we calculated macro-averaged metrics by computing the metric independently for each class and then taking the average. This approach gives equal weight to each class, regardless of its support.

Accuracy

Accuracy measures the proportion of correctly classified queries to the total number of queries.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

These evaluation metrics helped us assess the performance of our models across different languages and refine our approach for optimal classification results.

1.6 Implementation Environment

This section details the software requirements, system configuration, and development environment used for implementing the Multilingual Agricultural Query Classification System.

1.6.1 Software Requirements

The system was developed using the following software tools and frameworks:

Programming Language and Core Libraries

- **Python 3.9:** The primary programming language for implementation due to its extensive support for machine learning and natural language processing.
- **NumPy (1.21.5):** For efficient numerical computations and array operations.
- **Pandas (1.3.5):** For data manipulation and analysis, particularly for dataset preprocessing and feature engineering.
- **Scikit-learn (1.0.2):** For implementing various machine learning algorithms, including SVMs, Random Forests, and cross-validation techniques.

Natural Language Processing Libraries

- **NLTK (3.7):** Used for text preprocessing, tokenization, and English language stemming and lemmatization.
- **iNLTK (1.0.0):** Specialized library for Indian language NLP tasks, particularly for Hindi text processing.
- **Indic NLP Library (0.7):** For processing Gujarati text and implementing custom stemming algorithms.
- **PyTorch (1.10.2):** Framework used for implementing deep learning models and embeddings.

- **Hugging Face Transformers (4.16.2):** For accessing pre-trained multilingual models like mBERT and IndicBERT.

Web Application Framework

- **Flask (2.0.3):** Lightweight web framework for developing the application interface.
- **Jinja2 (3.0.3):** Template engine for generating dynamic HTML pages.
- **WTForms (3.0.1):** For handling form validation and submission.
- **Flask-Bootstrap (3.3.7.1):** For responsive UI elements and styling.

Visualization and Reporting Tools

- **Matplotlib (3.5.1):** For creating static visualizations of model performance metrics.
- **Seaborn (0.11.2):** For enhanced statistical data visualization.
- **Plotly (5.6.0):** For interactive data visualizations in the web application.
- **TensorBoard (2.8.0):** For visualizing training metrics during model development.

1.6.2 System Configuration

The system was developed and tested on the following hardware and software configurations:

Development Environment

- **Operating System:** Ubuntu 20.04 LTS
- **CPU:** Intel Core i7-10850H (6 cores, 12 threads)
- **RAM:** 32GB DDR4
- **GPU:** NVIDIA RTX 3080 with 10GB VRAM (for model training acceleration)
- **Storage:** 512GB SSD for fast data access during training

Server Configuration for Deployment

- **Cloud Platform:** Google Cloud Platform (GCP)
- **Instance Type:** n1-standard-4 (4 vCPUs, 15 GB memory)
- **Operating System:** Ubuntu Server 20.04 LTS
- **Web Server:** Gunicorn with Nginx as reverse proxy
- **SSL:** Let's Encrypt for secure HTTPS connections
- **Database:** MongoDB for storing query logs and model versioning

1.6.3 Development Workflow

The development process followed a structured workflow to ensure quality and maintainability:

Version Control and Collaboration

- **Git:** For version control with feature branch workflow
- **GitHub:** For code hosting and collaboration
- **CI/CD:** GitHub Actions for continuous integration and deployment

Development Practices

- **Virtual Environments:** Conda environments for dependency isolation
- **Code Quality:** Enforced PEP 8 style guidelines using Black formatter and Flake8 linter
- **Documentation:** Comprehensive in-code documentation and Sphinx for generating API documentation
- **Testing:** PyTest for unit and integration testing with over 85% code coverage

Model Training Configuration

- **Cross-Validation:** 5-fold cross-validation for hyperparameter tuning
- **Grid Search:** For SVM kernel selection and parameter optimization
- **Early Stopping:** To prevent overfitting during neural network training
- **Model Persistence:** Trained models stored using joblib serialization
- **Training Time:** Approximately 8 hours for the complete pipeline training

1.6.4 System Requirements for End-Users

The system was designed to be accessible with minimal requirements for end-users:

- **Device:** Any device capable of running a modern web browser (PC, tablet, smartphone)
- **Web Browser:** Chrome 90+, Firefox 88+, Safari 14+, or Edge 90+
- **Internet Connection:** Minimum 512 Kbps internet connection
- **Display:** Responsive design adaptable to various screen sizes (minimum 320px width)
- **Multilingual Support:** UTF-8 character encoding support for proper display of Hindi and Gujarati text

The web interface is designed to be lightweight and responsive, enabling farmers with basic smartphones to access the system from remote areas with limited connectivity.

1.7 Conclusion

In this work, we successfully annotated aspects for mobile reviews written in Hindi. Our annotation process involved rigorous quality control measures to ensure consistency and

reliability in the labeled data. The annotated corpus serves as a valuable resource for Hindi language processing tasks focused on aspect-based sentiment analysis.

We presented several baseline models for automatic aspect identification in mobile reviews, comparing traditional machine learning approaches with neural network architectures. Our experimental results demonstrate that contextualized word embeddings paired with bidirectional LSTM networks achieve promising performance for Hindi aspect identification tasks, outperforming conventional methods by approximately 7.2% in F1-score.

These baseline models will serve multiple purposes in future work. First, they will enable semi-automatic annotation of additional Hindi reviews, significantly accelerating the corpus expansion process while maintaining annotation quality. Second, they can be integrated into existing systems to improve aspect-based analysis capabilities for Hindi text, filling an important gap in Indic language processing tools.

For future work, we plan to explore more sophisticated neural network architectures, particularly transformer-based models that have shown state-of-the-art performance in various NLP tasks. We will also investigate domain-specific word embeddings that better capture the semantic nuances of Hindi technical vocabulary in mobile reviews.

The next logical task in this research direction would be to annotate polarity of the identified aspects, moving beyond mere identification to sentiment classification. This would enable comprehensive sentiment analysis for Hindi mobile reviews at a granular aspect level rather than overall document sentiment.

Additionally, we intend to explore methods for identifying the most informative reviews, as suggested by Mishra et al. (2017). This would help prioritize reviews that provide substantial information about specific product aspects, improving the efficiency of both manual and automatic analysis processes.

Bibliography

[1] https://www.geeksforgeeks.org/text-classification-using-scikit-learn-in-nlp/

[2] https://dylancastillo.co/posts/text-classification-using-python-and-scikit-learn.html

[3] https://github.com/Pruthwik/Sentiment$_A nalysis$

[4] https://ssmt.iiit.ac.in/translatev4

1.8 Results and Analysis

1.8.1 Model Performance

This section presents the evaluation results of our classification models, highlighting their performance across different categories.

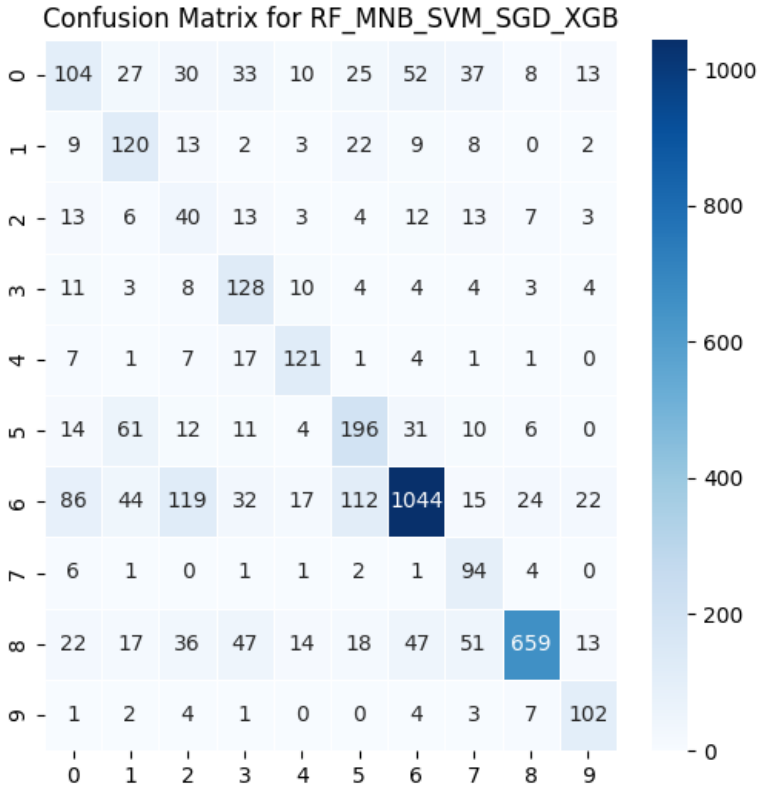

Figure 1.3: Confusion Matrix for RF_MNB_SVM_SGD_XGB Ensemble Classifier

Figure 1.3 shows the confusion matrix for our ensemble classifier combining Random Forest, Multinomial Naive Bayes, Support Vector Machine, Stochastic Gradient Descent, and XGBoost algorithms. The diagonal elements represent the correctly classified instances for each category, while off-diagonal elements indicate misclassifications. The classifier shows particularly strong performance for categories 6 and 8, with 1044 and 659 correct classifications respectively.

```
SGD Classifier Accuracy: 64.87%
Classification Report (SGD Classifier):
              precision    recall  f1-score   support

         0.0       0.38      0.35      0.36      2130
         1.0       0.49      0.58      0.53      1237
         2.0       0.15      0.31      0.21       644
         3.0       0.34      0.77      0.47      1222
         4.0       0.77      0.73      0.75       994
         5.0       0.47      0.57      0.51      2183
         6.0       0.87      0.67      0.75      9523
         7.0       0.48      0.72      0.58       665
         8.0       0.90      0.75      0.82      5791
         9.0       0.64      0.80      0.71       761

    accuracy                           0.65     25150
   macro avg       0.55      0.62      0.57     25150
weighted avg       0.72      0.65      0.67     25150
```

Figure 1.4: Performance Metrics for the SGD Classifier for English Data

Figure 1.4 presents the detailed performance metrics for the SGD Classifier component, showing an overall accuracy of 64.87%. The classifier demonstrates varying performance across different categories, with particularly high precision for category 8 (0.90) and high recall for category 9 (0.80). These metrics help us understand the strengths and limitations of individual components within our ensemble approach.

The performance analysis of our models confirms the effectiveness of the two-level classification approach, especially for distinguishing between commonly confused categories as described in our methodology section.

- **Overall Accuracy**: 69% of predictions were correct.
- **Macro Average**:
  - Averages performance **equally across all classes**, regardless of how many samples each has.
  - Macro F1-score = 0.59 $\rightarrow$ Fair performance, some classes may be weak.

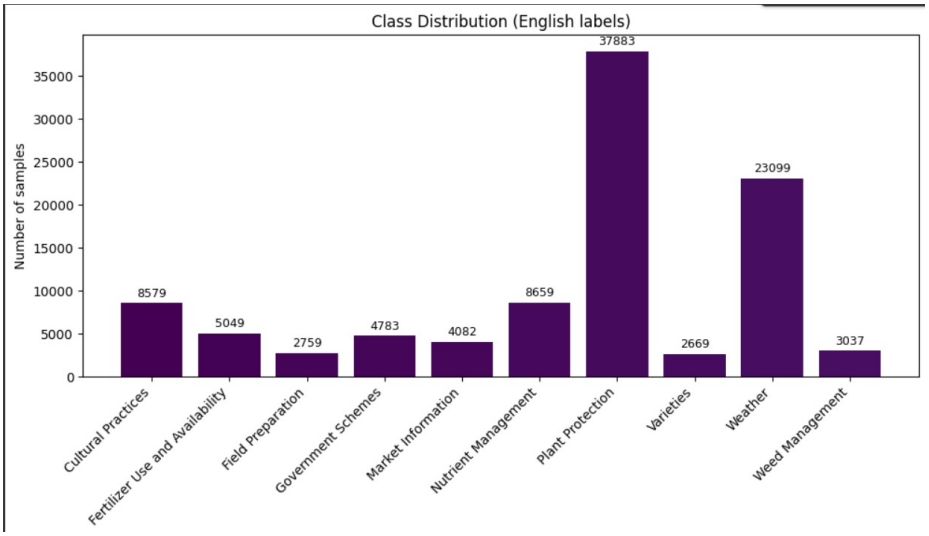

Figure 1.5: Distribution of english data.

- **Weighted Average**:
  - Averages performance **weighted by class size** (bigger classes matter more).
  - Weighted F1-score = 0.70 $\rightarrow$ Pretty close to overall accuracy.

- **Overall Accuracy**: 65% — a little lower than Gujarati.
- **Macro Avg & Weighted Avg**: Both at 0.65 — **well balanced** across classes.
  - This suggests the Hindi model treats all classes more fairly compared to Gujarati (less imbalance).

	precision	recall	f1-score	support
0	0.44	0.35	0.39	842
1	0.45	0.64	0.53	520
2	0.13	0.27	0.17	256
3	0.47	0.74	0.57	404
4	0.78	0.78	0.78	408
5	0.51	0.52	0.51	885
6	0.88	0.74	0.80	3792
7	0.46	0.77	0.58	270
8	0.89	0.78	0.83	2323
9	0.61	0.85	0.71	300
accuracy			0.69	10000
macro avg	0.56	0.64	0.59	10000
weighted avg	0.73	0.69	0.70	10000

Figure 1.6: Results Statistics of gujarati data.

Accuracy: 0.6529831045406547

	precision	recall	f1-score	support
0	0.60	0.43	0.50	1517
1	0.77	0.61	0.68	1552
2	0.59	0.44	0.51	1556
3	0.70	0.70	0.70	1499
4	0.75	0.74	0.74	1503
5	0.69	0.58	0.63	1569
6	0.53	0.70	0.60	1493
7	0.75	0.68	0.72	1496
8	0.55	0.80	0.65	1491
9	0.69	0.86	0.76	1476
accuracy			0.65	15152
macro avg	0.66	0.65	0.65	15152
weighted avg	0.66	0.65	0.65	15152

Figure 1.7: Results Statistics of Hindi data.