ChatSphere
An Advanced Chatbot Framework
Final Report

Aarav Dawer Bhojwani Agam Harpreet Singh Nirmal Kumar Godara
Ishan Shah Mahi Chouhan

Indian Institute of Technology, Jodhpur
{b23cm1002, b23cm1004, b23cm1027, b23cm1050, b23ee1038}@iitj.ac.in

Abstract

This report documents the comprehensive development and deployment of ChatSphere—a modular chatbot framework designed from scratch using Python. The project features advanced text preprocessing, a suite of machine learning classifiers with detailed mathematical formulations, and an end-to-end containerized deployment pipeline. The machine learning models are trained using Colab Enterprise on Vertex AI in Google Cloud, deployed with Flask on Google Cloud and Express/Node.js on Vercel. Detailed experimental results for each classifier are provided, highlighting performance improvements and challenges encountered.

Contents

1 Introduction

Our earlier mid-term report focused on baseline performance with a DailyDialog dataset and a Naive Bayes classifier. While working on the general-purpose DailyDialog-based chatbot, we gained hands-on experience in implementing key components of a typical chatbot pipeline, including data preprocessing, feature extraction, intent classification, and response generation. This initial groundwork helped us understand the architectural and algorithmic foundations of conversational systems. Building upon this experience, we recognized the potential of tailoring such a system to address institution-specific queries, leading to the development of ChatSphere for our college environment.

ChatSphere is an end-to-end chatbot framework developed to efficiently handle natural language institution-specific queries and provide accurate intent-based responses. In the final iteration, we have:

- Replaced the DailyDialog dataset with a custom-built dataset tailored for our college environment, enabling ChatSphere to better understand and respond to institution-specific queries.
- Expanded our classifier suite to include Naive Bayes, Decision Tree, and Random Forest classifiers, and evaluated their accuracy on the tailored dataset to identify the most effective model.
- Developed a robust containerized deployment pipeline using Docker, with the machine learning backend served by Flask on Google Cloud, and the frontend developed using Express/Node.js and deployed on Vercel, ensuring scalability, portability, and ease of deployment.
- Leveraged Colab Enterprise and Vertex AI on Google Cloud for model training, allowing us to take advantage of GPU acceleration and seamless integration with our cloud-based workflow.

2 Resources and Project Links

Update these links with your project-specific URLs:

- **Project Link:** https://prml-project-tan.vercel.app
- **GitHub Repository:** https://github.com/Agam77055/PRML_Project
- **Colab File (Vertex AI: Colab Enterprise):** https://colab.research.google.com/drive/1BIHniBGs5HLxwrkyzVveOI5J0HNIwZl0scrollTo=fdefec39
- **Video:**

3 Dataset

The effectiveness of any intent-based chatbot system is deeply rooted in the quality and organization of its dataset. For ChatSphere, we curated a custom dataset specifically tailored to address the common queries encountered in our institutional environment. Each data point in the dataset is structured as an "intent," comprising a *tag* that labels the intent class, a list of representative *patterns* that simulate possible user inputs, and a corresponding set of *responses* from which the chatbot can reply.

```
{
  "tag": "events",
  "patterns": ["What events are organized in the college?", "Are
      there any upcoming events?", "Tell me about the college
      events."],
  "responses": ["IIT Jodhpur hosts a variety of events like Ignus
      (cultural fest), Prometeo (technical fest), and Varchas
      (sports fest), among others. Check the institute's website or
      social media for updates."],
  "context_set": ""
},
{
  "tag": "admission",
  "patterns": ["What is the admission process?", "How can I take
      admission in the college?", "Tell me about the admission
      criteria."],
  "responses": ["Admission to IIT Jodhpur is through national-level
      exams like JEE Advanced for BTech, GATE for MTech, JAM for
      MSc, and CAT for MBA. Visit www.iitj.ac.in for complete
      details."],
  "context_set": ""
},
{
  "tag": "library",
  "patterns": ["Does the college have a library?", "Where is the
      library located?", "What are the library timings?"],
  "responses": ["Yes, IIT Jodhpur has a central library located in
      the Academic Block. It is open from 9 AM to 9 PM on working
      days and 10 AM to 6 PM on weekends, but the students can use
      the facility for studying 24 x 7."],
  "context_set": ""
},
```

Figure 1: A sample overview of the Dataset

This structured format not only streamlines mapping diverse user expressions to well-defined intents by incorporating multiple representative patterns per intent and leveraging techniques such as synonym replacement for lexical augmentation, but it also facilitates seamless extension and customization of the chatbot's knowledge base. The clear separation of intent tags, sample patterns, and response templates promotes a modular design—enabling classifiers to learn robust associations between varied user inputs and their semantic meanings through advanced preprocessing steps like tokenization, lemmatization, and TF-IDF vectorization. This normalization of textual data ensures that even when user inputs deviate from the exact examples provided, the underlying semantic similarities are effectively captured. Furthermore, the inclusion of a `context_set` field—although currently inactive—lays the foundation for future development of context-aware, multi-turn dialogues, thereby enhancing the system's scalability and adaptability to more dynamic conversational scenarios.

4 System Architecture and Components

The ChatSphere framework is divided into several key components as outlined below.

4.1 Text Preprocessing and Feature Extraction

Our system employs a comprehensive text preprocessing pipeline to convert raw input into standardized features for intent classification. The process involves advanced tokenization and feature extraction techniques, as outlined below.

4.1.1 Text Preprocessing

The preprocessing module leverages a custom `Tokenizer` that implements multiple steps to normalize and clean input text:

- **Multi-Word Expression (MWE) Recognition:** The tokenizer uses WordNet to extract multi-word expressions (e.g., converting "new york" into "new_york") so that such phrases are treated as single tokens.
- **Contraction Handling:** A predefined contractions dictionary is employed to expand contractions, ensuring that words like "can't" are transformed into "cannot". This enhances the consistency of token representations.
- **Hyphen and Punctuation Management:** Hyphenated words are split into separate tokens (e.g., "high-quality" becomes "high quality"), while extraneous punctuation is removed—except when preserving numbers with attached units (e.g., "10kg" is processed as "10_kg").
- **Case Normalization and Token Splitting:** All input text is converted to lowercase, and tokenization is then performed based on whitespace to produce a uniform sequence of tokens.

4.1.2 Feature Extraction

After preprocessing, the cleaned text is transformed into numerical feature vectors using a custom `tf_idf_Vectorizer`. This vectorizer carries out the following steps:

- **Vocabulary Construction:** The vectorizer scans through the tokenized corpus to build a vocabulary, assigning unique indices to terms based on their frequency and relevance.
- **TF-IDF Computation:** For each token in a document, the Term Frequency (TF) is computed relative to the total number of tokens in the document, and the Inverse Document Frequency (IDF) is computed with smoothing as:

$$\text{TF-IDF}(t,d) = \left(\frac{\text{Count}(t,d)}{\sum_{t'} \text{Count}(t',d)}\right) \times \left[\log\left(\frac{N+1}{\text{DF}(t)+1}\right)+1\right],$$

  where N is the total number of documents and $\text{DF}(t)$ is the number of documents containing the term t.

This combined preprocessing and feature extraction pipeline ensures that textual variability—such as differences in phrasing, inflection, or punctuation—does not hinder the model's ability to capture the semantic meaning of user queries. As a result, even when inputs deviate from predefined patterns, the underlying classifiers can effectively map them to the correct intents.

4.2 Intent Classification and Entity Extraction

Multiple classifiers have been implemented to boost intent recognition accuracy. Below we describe each classifier in detail along with their experimental performance.

4.2.1 Multinomial Naive Bayes Classifier

Model Description: Multinomial Naive Bayes uses Bayes' theorem under the assumption of conditional independence among features. The posterior probability of class c given an input x is given by:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}, \tag{1}$$

and using the bag-of-words model with Laplace smoothing:

$$P(t|c) = \frac{\text{Count}(t,c)+1}{\sum_{t'} \text{Count}(t',c)+|V|}, \tag{2}$$

where $|V|$ is the vocabulary size.

Experimental Results: On our custom-built college dataset, the Multinomial Naive Bayes classifier achieved a baseline accuracy of 90%.

- **Accuracy:** 90.48%.
- **Precision:** 0.9115
- **Recall:** 0.9075
- **F1 Score:** 0.9012

4.2.2 Decision Tree Classifier

Model Description: Decision Trees segment data through recursive partitioning based on impurity measures. Two commonly used impurity measures are:

- **Gini Index:**

$$Gini = 1 - \sum_{i=1}^{C} p_i^2, \tag{3}$$

  where p_i is the proportion of samples of class i at the node.

- **Entropy:**

$$Entropy = -\sum_{i=1}^{C} p_i \log_2(p_i). \tag{4}$$

Experimental Results: On our custom-built college dataset, the Decision Tree classifier achieved a baseline accuracy of 89%.

- **Accuracy:** 89.08%.
- **Precision:** 0.9192
- **Recall:** 0.8964
- **F1 Score:** 0.8944

4.2.3 Random Forest Classifier

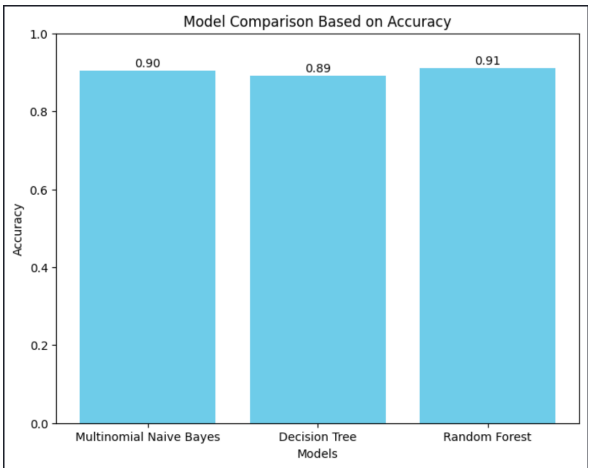
Model Description: Random Forest aggregates multiple Decision Trees to create an ensemble prediction, reducing variance and improving robustness. The final prediction is typically the mode (for classification) of individual tree predictions:

$$\hat{y} = \text{mode}\left(\{y^{(1)}, y^{(2)}, \ldots, y^{(T)}\}\right), \tag{5}$$

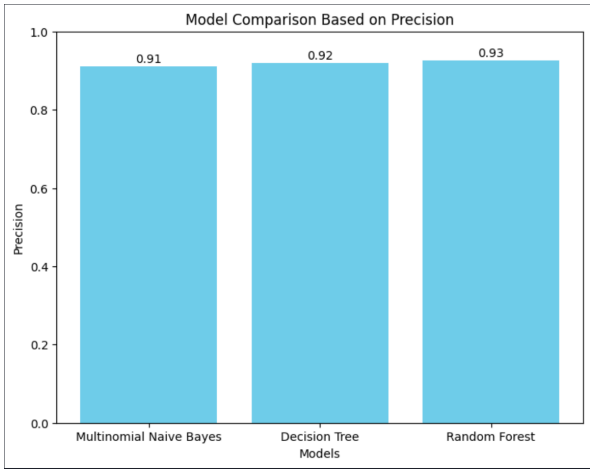
where T represents the number of trees in the forest.

Experimental Results: On our dataset, the Random Forest classifier achieved the highest accuracy among our models.

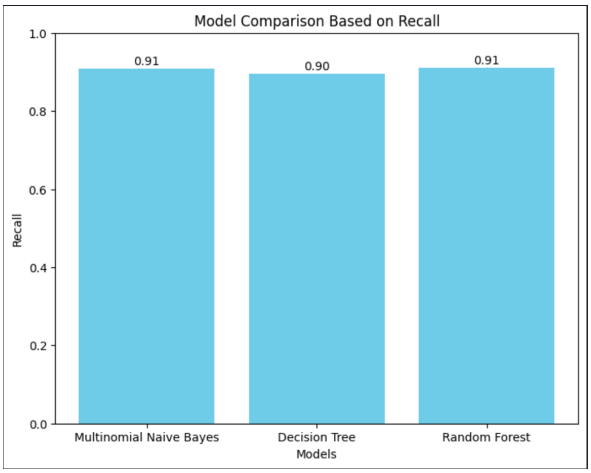
- **Accuracy:** 91.11%.
- **Precision:** 0.9258
- **Recall:** 0.9115
- **F1 Score:** 0.9116



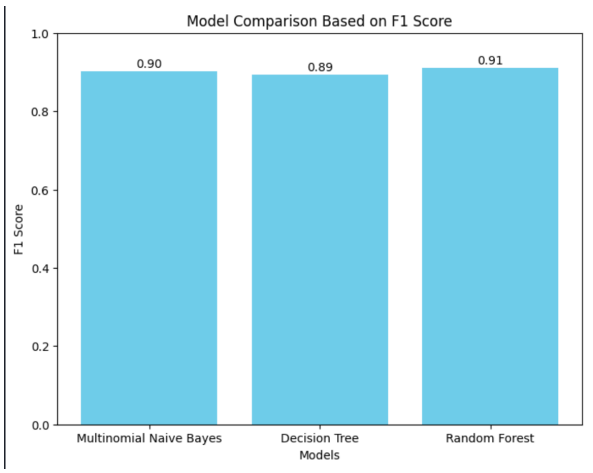
(a) Accuracy Comparison



(b) Precision Comparison



(c) Recall Comparison



(d) F1 Score comparison

Figure 2: Comparison between the Classification models based on various Metrics

5 Deployment Architecture (Using Google Cloud)

Our deployment strategy involves containerization, cloud deployment, and API key integration to ensure secure and scalable operations.

5.1 Containerization and Dockerization

- **Creating Docker Images:** We containerized only the machine learning model (With the help of Flask) using Docker, enabling consistent deployment across environments.
- **Dockerfile Configuration:** A custom Dockerfile bundles all required dependencies and model artifacts needed for inference.

5.2 Deployment on Google Cloud

- **ML Model and Backend:** The Docker images are deployed on Google Cloud where an API key is automatically generated. This key is injected into the backend code for secure communication.
- **API Key Integration:** The backend uses the provided API key to authenticate requests, and the same key is passed to the frontend as required.
- **GitHub Integration:** The backend codebase is deployed via a GitHub repository, enabling continuous integration and deployment workflows.

5.3 Frontend Deployment on Vercel

- **Frontend Setup:** The frontend, built with Express, Node.js, and vanilla JavaScript, is configured with the backend API key.
- **Deployment Process:** Once containerized, the frontend is deployed on Vercel, ensuring rapid hosting with support for real-time communication via WebSockets.

5.4 Training Infrastructure

- **Vertex AI on Google Cloud:** Model training was executed using the College_Chatbot.ipynb notebook on Vertex AI (Colab Enterprise) with an L4 GPU, providing the necessary computing power for training our custom-built college chatbot dataset.

6 Conclusion and Future Work

6.1 Conclusion

ChatSphere has evolved into a robust, modular chatbot framework capable of effectively handling real-world queries in a college environment. Our use of advanced NLP techniques, detailed mathematical modeling of classifiers, and an integrated containerized deployment pipeline underscores the system's technical depth and practical viability.

6.2 Future Work

Future enhancements will focus on:

- **Model Optimization and Advanced NLP Techniques**
  - **Hyperparameter Fine-Tuning:** Further optimize hyperparameters for traditional models such as SVM and Random Forest.
  - **Enhanced NLP Approaches:** Incorporate advanced techniques including contextual embeddings, transformer architectures, Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and hybrid models that combine classical machine learning with neural approaches to boost classification performance and enable dynamic response generation.
- **Data Expansion and Continuous Training**
  - **Dataset Enhancement:** Expand the custom-built dataset with continuous feedback to ensure the data remains representative and up-to-date.
  - **Continuous Pipelines:** Implement continuous training and deployment pipelines that allow the model to evolve with incoming data and feedback.
- **Advanced Response Generation and Data Integration**
  - **Dynamic Response Generation:** Investigate the use of advanced NLP models to facilitate more dynamic and context-aware responses.
  - **Chat History Integration:** Connect a robust database system (MySQL, MongoDB, etc.) to store chat histories, enabling detailed analysis and continuous improvement of chatbot responses.
  - **Monitoring and Analytics:** Enhance monitoring tools and analytical capabilities to gather user feedback and systematically refine system responses over time.
- **Scalability and Service Integration**
  - **Cloud Auto-Scaling:** Optimize auto-scaling configurations on Google Cloud to efficiently handle increased user load.
  - **Platform Integration:** Integrate the chatbot with mobile platforms and additional college services to extend reach and functionality.

Contributions

1. **Aarav Dawer Bhojwani:** Implemented Multinomial Naive Bayes Classifier and created the dataset (intents.json).
2. **Agam Harpreet Singh & Ishan Shah:** Developed Random Forest Classifier and managed Development (Backend + Frontend) and Deployment (Docker Containerization + Google Cloud API integration)
3. **Nirmal Kumar Godara:** Implemented the Decision Tree Classifier and created the dataset (intents.json)
4. **Mahi Chouhan:** Designed and implemented the tokenizer and TF-IDF vectorizer of PreProcesing and created the dataset (intents.json).

References

[1] "College Chatbot Using ML and NLP", GitHub repository, available at `https://github.com/roshancharlie/College-Chatbot-Using-ML-and-NLP/blob/main/College%20Chatbot.ipynb`.

[2] Adamopoulou, E., & Moussiades, L. (2020). Chatbot Systems: A Comprehensive Review. *Machine Learning with Applications*, 2.

[3] "ChatBot-ML-LSTM", GitHub repository, available at `https://github.com/TusharPaul01/ChatBot-ML-LSTM-`.

[4] *Multinomial Naive Bayes Classification Explained*, YouTube video, available at `https://www.youtube.com/watch?v=Vfo5le26IhY`.

[5] *Decision Trees in Machine Learning: A Complete Guide*, YouTube video, available at `https://www.youtube.com/watch?v=7VeUPuFGJHk`.

[6] *Random Forest Classifier from Scratch - Machine Learning Tutorial*, YouTube video, available at `https://www.youtube.com/watch?v=JP8pI0jVgFQ`.