

# Disease Prediction Model

May 4, 2025

## Abstract

This project introduces a disease prediction model developed using an ensemble learning approach. The model analyzes user-provided descriptions and symptoms to predict the three most probable diseases based on a predefined medical data set using probabilistic methods. The implementation prioritizes simplicity, interpretability, and rapid response time, making it ideal for integration into diagnostic tools and early warning systems. To enhance accessibility, the model supports multilingual input including English, Hindi, Gujarati, and Marathi and accepts both text and voice input. Designed for user convenience, it interprets natural language descriptions of the patient's condition as well as direct symptom entries. This model has promising applications in healthcare chatbots and preliminary diagnostic tools, particularly in resource-constrained or multilingual environments.

## 1 Team Members

This project was completed with the collaborative efforts of the following team members:

- **PRIT JAGANI** - U23AI079
- **HET TALAVIA** - U23AI077
- **VISHAL GAUTAM** - U23AI0127
- **HANMANT JAJULWAR** - U23AI098
- **ANURAG KUMAR PANKAJ** - U23AI126

Each member contributed significantly to the development of the disease prediction model. Their efforts include data processing, model training, UI design, and overall project management.

Here is the [GitHub Repository](#) that contains the source code of the project.

## 2 Introduction

The growth of artificial intelligence in the healthcare domain has enabled intelligent systems to assist with early diagnosis, treatment recommendations, and disease prevention. This report presents a multilingual, voice and text enabled disease prediction model powered by ensemble learning techniques. The model analyzes user input and predicts the three most likely diseases, aiming to aid in early diagnosis and accessible healthcare delivery.

### 2.1 Background and Motivation of the Problem

In many parts of the world, especially in rural regions, there is limited access to doctors and diagnostic services. Language barriers and the inability of patients to express symptoms in a way that clinical systems can interpret further worsen this gap. . To address these challenges, this project aims to build a user friendly, multilingual disease prediction system that accepts both text and voice inputs. By simplifying the process of symptom reporting and offering probable diagnoses, this system can act as a preliminary diagnostic tool and reduce the burden on healthcare professionals.

### 2.2 Literature Survey or Related Works

Several disease prediction models have been developed using machine learning techniques such as Naive Bayes, Decision Trees, Support Vector Machines, and Neural Networks. Notably:

- Kumar et al. (2020) implemented a symptom-based diagnosis system using Decision Trees and achieved promising results for common diseases.
- Patil and Pawar (2019) explored Naive Bayes and Random Forest algorithms for disease prediction with accuracy-focused evaluations.
- Google’s Med-PaLM and Babylon Health’s AI systems use large-scale datasets and advanced NLP for diagnostic chatbots, though they often require English input and internet access.

### 2.3 Contributions

This project contributes the following:

- A disease prediction model based on ensemble learning, which improves diagnostic accuracy by combining predictions from multiple classifiers.

- Multilingual support for English, Hindi, Gujarati, and Marathi.
- Acceptance of both text and voice input, enhancing accessibility for users.
- Capability to process natural language descriptions of symptoms in addition to direct symptom keywords.

### 3 Flowchart of the project

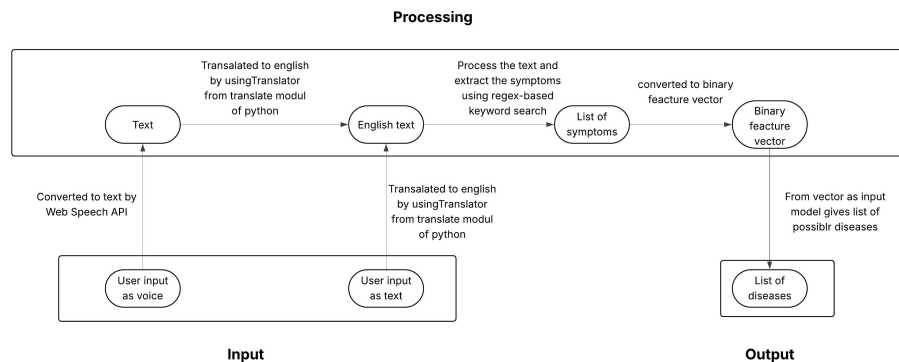


Figure 1: This is the flow of the data

The flowchart explain the data flow and transformation of data at each point as it passes through the various part of the pipeline of the project. Each step is explained below.

- **Voice to Text**

**Input:** User's spoken audio.

**Process:**

- The browser's Web Speech API (SpeechRecognition) captures audio and converts it to text.
- Language is dynamically detected.

**Output:** Raw text string.

- **Translate the Text to English**

**Input:** Raw text (potentially in any language).

**Process:**

- Language Detection: The input language is identified by the user (e.g., **hi** for Hindi).
- Translation: Converts non-English text to English via Translator.

**Output:** Standardized English text.

- **Extracting the Symptoms**

**Input:** English text.

**Process:**

- **Text Cleaning:**

- \* Lowercasing.
- \* Remove punctuation and numbers.
- \* Normalize spaces.

– **Keyword Matching:**

- \* Compares cleaned text against a predefined `feature_list`.
- \* Uses regex word boundaries (`\b`) for exact matches.

**Output:** Set of matched symptoms.

• **Conversion to Binary Feature Vector**

**Input:** Extracted symptoms.

**Process:**

- Initialize a zero-vector of size `len(feature_list)`.
- Set 1 for indices matching extracted symptoms.

• **Result in Form of the Name of the Diseases**

**Preprocessing:**

- Imputation (fills missing values, though unlikely here).
- Scaling (normalizes features via `scaler.pkl`).
- Dimensionality reduction (applies `pca.pkl`).

**Model Inference:**

- Ensemble model predicts disease probabilities.
- Outputs top 3 diseases.

**Output:** Ranked disease predictions with confidence scores.

## 4 Detailed Description of Methodology

- **Voice to Text**

**Input:** User's spoken audio.

**Process:**

- The browser leverages the Web Speech API (**SpeechRecognition**), which is natively supported in modern browsers like Google Chrome.
- **Language Selection:** The user selects a preferred language (e.g., Hindi, English-India) from a dropdown menu provided on the webpage interface.
- **Speech Capture:** Upon clicking the mic button, the browser begins listening for user voice input using the API.
- **Recognition Engine:** The captured speech is then processed by the browser's built-in recognition engine and automatically converted into text.
- **Language Handling:** The recognition object is assigned the language code (e.g., "hi-IN") using `recognition.lang`, which ensures accurate phonetic recognition and transcription.

**Output:** Raw text string representing the spoken input.

- **Translate the Text to English**

**Input:** Raw text (potentially in any of the supported Indian languages).

**Process:**

- **Language Detection:** The language is either pre-identified or selected manually by the user (e.g., `hi` for Hindi).
- **Translation Attempts:**
  1. **CSV-based Method:**
    - \* A custom CSV file was created containing common English terms for diseases and symptoms, along with their translations in Hindi, Gujarati, and Marathi.
    - \* *Implementation:* A Python script was used to automate translation using the Google Translate API. To prevent data loss due to request limits, the script saved the output every 50 words.
    - \* *Problem:* The IP address was blocked by Google after numerous requests due to API usage limits, making this method unsustainable.
  2. **Deep Translator Library:**
    - \* Attempted the use of the `deep-translator` Python library, which offers integration with Google Translate and other services.

- \* *Problem:* Translations were often inaccurate or inconsistent, resulting in a significant drop in disease prediction accuracy for non-English inputs.

### 3. Translate Library:

- \* Migrated to using the `translate` Python library and its `Translator` class.
- \* *Conclusion:* This approach provided more reliable and contextually accurate translations. It was therefore selected as the final translation method.

**Output:** Standardized English text suitable for further processing.

### • Extracting the Symptoms

**Input:** English text (either originally spoken in English or translated).

**Process:**

#### – Text Cleaning:

- \* The input text is first converted to lowercase to eliminate case sensitivity.
- \* All punctuation, numbers, and special characters are removed.
- \* White spaces are normalized to ensure uniform tokenization.

#### – Keyword Matching:

- \* The cleaned text is scanned for occurrences of symptoms using a predefined list called `feature_list`.
- \* Regex with word boundaries (`\b`) is applied to avoid partial or false-positive matches.

**Output:** A set of detected symptom keywords matched from the text.

### • Conversion to Binary Feature Vector

**Input:** Set of matched symptoms.

**Process:**

- A zero-initialized binary vector is created with a length equal to the number of elements in `feature_list`.
- Each index corresponding to a matched symptom is set to 1, thus creating a feature vector that represents the presence or absence of symptoms.

**Output:** Binary vector for use as input to the prediction model.

### • Result in Form of the Name of the Diseases

**Preprocessing:**

- Apply data imputation techniques, if necessary, to handle missing values in the vector.
- Normalize the input features using a pre-trained scaler object saved as `scaler.pkl`.
- Reduce the vector's dimensionality using Principal Component Analysis (PCA), implemented with a pre-trained model stored in `pca.pkl`.

### **Model Inference:**

- An ensemble classification model is used to predict disease probabilities.
- The top 3 diseases with the highest confidence scores are returned.
- Optionally, the predicted diseases are translated back to the user's original language for ease of understanding.

**Output:** List of the top 3 predicted diseases along with their confidence scores.

### **• Making of the Model:**

An ensemble model was created using four classifiers.

### **• Selection Process of Models:**

A Python script was developed to test all combinations from a set of eight candidate models:

- LogisticRegression
- RandomForestClassifier
- GradientBoostingClassifier
- SVC
- KNeighborsClassifier
- DecisionTreeClassifier
- GaussianNB
- AdaBoostClassifier

The script trained each combination on a sample of 10,000 rows from the dataset. Since the dataset is well-balanced (non-biased), accuracy was used as the evaluation metric. After comparing the accuracies of all combinations, the highest-performing combination was:

- K-Nearest Neighbors (KNN)
- Random Forest Classifier
- Gaussian Naive Bayes



- AdaBoost Classifier

with an accuracy of 0.9829.

These were combined using a soft voting classifier to form the final ensemble model.

- **How the Number of PCA Features Was Decided:**

Once the ensemble model was selected, another Python script was written to find the optimal number of principal components for PCA. This script:

- Trained the model on PCA-reduced data using component numbers ranging from 100 to 250.
- Evaluated accuracy for each configuration using 10,000 rows of training data.
- Recorded the accuracy score for each number of components.

The highest accuracy was achieved with 167 components, which was then chosen as the optimal number of features for PCA transformation.

## 5 Implementation Environment, software requirements, configuration

### 5.1 Software Requirements

- **Programming Language:** Python 3.10+
- **Framework:** Flask
- **Libraries Used:**
  - scikit-learn
  - pandas
  - numpy
  - pickle
  - translate
  - flask
- **Environment:** Python Virtual Environment (venv)
- **Deployment Platform:** Render (<https://render.com>)

### 5.2 Model Training and Export

The machine learning model is a `VotingClassifier` trained on a dataset with 377 binary-encoded symptoms. The training script preprocesses the data using an imputer and scaler, applies PCA for dimensionality reduction, and fits the ensemble model. The trained components are saved using `pickle`:

- `final_ensemble_model.pkl` – the trained classifier
- `imputer.pkl` – to handle missing values
- `scaler.pkl` – for normalization
- `pca.pkl` – for dimensionality reduction
- `features.json` – list of features used for training

### 5.3 Running the Web Application Locally

To run the Flask app locally:

```
python -m venv myenv
myenv\Scripts\activate # or source myenv/bin/activate on Linux/macOS
pip install -r requirements.txt
python app.py
```

This starts a local server at `http://localhost:10000`, where users can enter symptoms in multiple languages to get predicted diseases.

### 5.4 Deployment on Render

The application is deployed on **Render**, a cloud platform for hosting web applications. The following steps were used for deployment:

1. Created a GitHub repository with all required files.
2. Added `requirements.txt` and `render.yaml` for configuration.
3. Set the build command as `pip install -r requirements.txt`.
4. Set the start command as `gunicorn app:app`.
5. Configured static files and web service via the Render dashboard.

The app runs publicly from a Render URL, accessible to anyone with internet access.

### 5.5 Directory Structure

The following is the project directory structure:

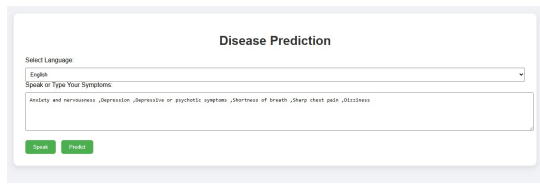
```
.
+-- app.py
+-- myenv/
+-- requirements.txt
+-- static/
|   +-- css/
|   |   +-- style.css
|   +-- features.json
|   +-- final_ensemble_model.pkl
|   +-- imputer.pkl
|   +-- js/
```

```
|   |   +-- speech.js
|   +-- pca.pkl
|   +-- scaler.pkl
+-- structure.txt
+-- symptoms.csv
+-- templates/
    +-- index.html
    +-- predicted.html
```

This structure separates the source code, static assets (CSS, JS, models), templates for rendering the frontend, and the virtual environment (‘myenv’) for dependency isolation.

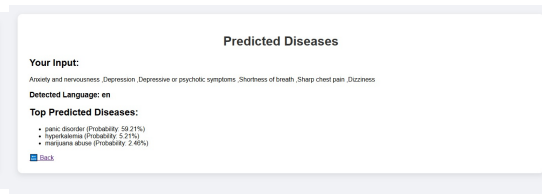


Here are the images of the of the input and output as exampls:



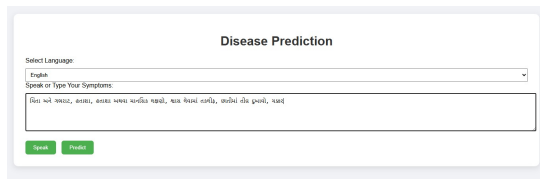
The screenshot shows the 'Disease Prediction' interface. The 'Select Language' dropdown is set to 'English'. The 'Speak or Type Your Symptoms' text area contains the text: 'anxiety and nervousness ,depression ,depressive or psychotic symptoms ,shortness of breath ,sharp chest pain ,dizziness'. At the bottom, there are 'Speak' and 'Predict' buttons.

Figure 3: English Input



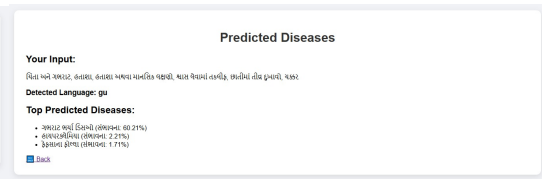
The screenshot shows the 'Predicted Diseases' interface. Under 'Your Input:', it lists 'Anxiety and nervousness ,Depression ,Depressive or psychotic symptoms ,Shortness of breath ,Sharp chest pain ,Dizziness'. The 'Detected Language: en' is shown. Under 'Top Predicted Diseases:', it lists: 'panic disorder (Probability: 59.21%)', 'hypertension (Probability: 5.27%)', and 'marijuana abuse (Probability: 2.46%)'. A 'Back' button is at the bottom.

Figure 4: English Output



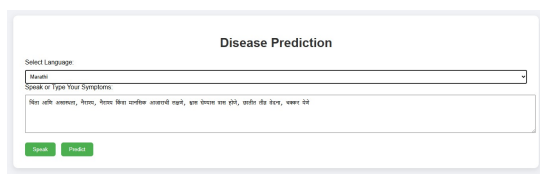
The screenshot shows the 'Disease Prediction' interface. The 'Select Language' dropdown is set to 'English'. The 'Speak or Type Your Symptoms' text area contains the text: 'દિલ નો તણાવ, ડેપ્રેશન, ડેપ્રેશન અથવા પશ્ચાત્ત્ય સંપત્તિ, શ્વાસ લેવામાં તકલીફ, તીવ્ર ધડકા, ડાઘાં'. At the bottom, there are 'Speak' and 'Predict' buttons.

Figure 5: Gujarati Input



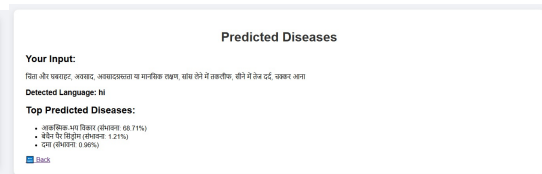
The screenshot shows the 'Predicted Diseases' interface. Under 'Your Input:', it lists 'દિલ નો તણાવ, ડેપ્રેશન, ડેપ્રેશન અથવા પશ્ચાત્ત્ય સંપત્તિ, શ્વાસ લેવામાં તકલીફ, તીવ્ર ધડકા, ડાઘાં'. The 'Detected Language: gu' is shown. Under 'Top Predicted Diseases:', it lists: 'પેનિક ડિઝોર્ડર (સંભાવના: 60.21%)', 'હાયપરટેન્શન (સંભાવના: 2.27%)', and 'કેન્સર (સંભાવના: 1.77%)'. A 'Back' button is at the bottom.

Figure 6: Gujarati Output



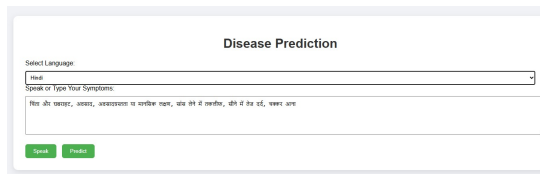
The screenshot shows the 'Disease Prediction' interface. The 'Select Language' dropdown is set to 'Hindi'. The 'Speak or Type Your Symptoms' text area contains the text: 'दिल में तनाव, अवसाद, अवसाद या मनोवैज्ञानिक लक्षण, सांस लेने में तकलीफ, तेरे में तेज दर्द, चक्कर आना'. At the bottom, there are 'Speak' and 'Predict' buttons.

Figure 7: Hindi Input



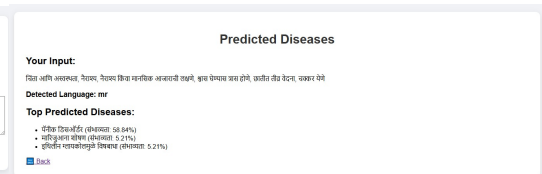
The screenshot shows the 'Predicted Diseases' interface. Under 'Your Input:', it lists 'दिल में तनाव, अवसाद, अवसाद या मनोवैज्ञानिक लक्षण, सांस लेने में तकलीफ, तेरे में तेज दर्द, चक्कर आना'. The 'Detected Language: hi' is shown. Under 'Top Predicted Diseases:', it lists: 'पैनिक डिऑर्डर (संभावना: 60.21%)', 'कोर डी डिऑर्डर (संभावना: 1.21%)', and 'डिऑर्डर (संभावना: 0.96%)'. A 'Back' button is at the bottom.

Figure 8: Hindi Output



The screenshot shows the 'Disease Prediction' interface. The 'Select Language' dropdown is set to 'Hindi'. The 'Speak or Type Your Symptoms' text area contains the text: 'दिल में तनाव, अवसाद, अवसाद या मनोवैज्ञानिक लक्षण, सांस लेने में तकलीफ, तेरे में तेज दर्द, चक्कर आना'. At the bottom, there are 'Speak' and 'Predict' buttons.

Figure 9: Marathi Input



The screenshot shows the 'Predicted Diseases' interface. Under 'Your Input:', it lists 'दिल में तनाव, अवसाद, अवसाद या मनोवैज्ञानिक लक्षण, सांस लेने में तकलीफ, तेरे में तेज दर्द, चक्कर आना'. The 'Detected Language: mr' is shown. Under 'Top Predicted Diseases:', it lists: 'पैनिक डिऑर्डर (संभावना: 58.84%)', 'कोर डी डिऑर्डर (संभावना: 5.21%)', and 'डिऑर्डर (संभावना: 5.21%)'. A 'Back' button is at the bottom.

Figure 10: Marathi Output

## 7 Conclusion

The development of a multilingual, voice-enabled disease prediction system represents a significant step toward enhancing accessibility and efficiency in preliminary healthcare diagnostics. This project integrates a range of technologies—voice recognition, language translation, natural language processing, and machine learning—to construct a user-centric system capable of interpreting symptom descriptions and predicting potential diseases with reasonable accuracy.

### Key Achievements

- **Voice and Text Input:** The system supports both voice-based and text-based inputs, enabling users with varying levels of digital literacy to interact seamlessly.
- **Multilingual Capability:** By utilizing translate library of python and speech recognition tools, the system accepts input in multiple Indian languages, ensuring inclusivity for non-English speakers.
- **Symptom Extraction and Processing:** User-provided symptoms are extracted using keyword matching techniques and mapped against a curated dataset for accurate prediction.
- **Disease Prediction Model:** The ensemble model, trained on over 130 disease classes and more than 40 symptoms, uses machine learning algorithms to deliver precise predictions based on user inputs.

### Impact and Relevance

This project addresses the challenges of language diversity and digital inaccessibility in India's healthcare context. By facilitating early disease prediction through an accessible interface, it can potentially reduce the burden on medical professionals and improve patient awareness. The incorporation of artificial intelligence and natural language processing aligns with the ongoing digital transformation in healthcare delivery.

### Limitations

While the system performs well under controlled conditions, there are some notable limitations:

- **Translation Accuracy:** Mismatches due to literal translations may affect symptom recognition accuracy.

- **Data Generalizability:** The prediction is limited to diseases present in the training dataset.
- **Context Understanding:** The system currently relies on keyword matching, lacking deeper contextual analysis of user inputs.

### Conclusion Statement

Overall, this project lays a strong foundation for voice-based, multilingual health support systems. It showcases how artificial intelligence can be employed to bridge healthcare accessibility gaps in linguistically diverse populations. With further development and dataset enhancement, such systems have the potential to be integrated into real-world telemedicine platforms, offering intelligent, inclusive, and scalable solutions for preliminary health assessment.



## 8 References

### References

- [1] Kumar, A., Sharma, P., & Gupta, R. (2020). Symptom-Based Disease Diagnosis Using Decision Trees.
- [2] Patil, S., & Pawar, K. (2019). Comparative Study of Naive Bayes and Random Forest for Disease Prediction.
- [3] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., *et al.* (2011). Scikit-learn: Machine Learning in Python.