A Survey on Predictive Healthcare Analysis Based on AI/ML Approaches

Javed Miya1, Ritesh Kumar2, Udit Maurya3, Vishal Yadav

1Department of IT, Galgotias College of Engg. & Tech., Greater Noida, India

Email: drmjkhan.kurukshetrauni@gmail.com (Corresponding Author)

2,3,4Department of IT, Galgotias College of Engg. & Tech., Greater Noida, India

2Email: [rkt.2004.2003@gmail.com](mailto:rkt.2004.2003@gmail.com)

3Email: [uditmaurya408@gmail.com](mailto:uditmaurya408@gmail.com)

4Email: [vy88957@gmail.com](mailto:vy88957@gmail.com)

**Abstract**

This project presents a web-based disease prediction system that utilizes a Voting Classifier—a combination of Support Vector Classifier (SVC), Random Forest (RF), and Gaussian Naive Bayes (GNB)—to predict probable diseases based on user-provided symptoms. The model was trained on a symptom-to-disease mapping dataset and integrated into a user-friendly interface using Streamlit. The application allows users to input symptoms and returns the most probable diseases along with prediction confidence. The system aims to assist users in getting a preliminary understanding of potential medical conditions before consulting healthcare professionals.

**1. INTRODUCTION**

## 1.1 Background

Early disease detection is crucial for effective treatment. However, users often struggle to understand what illness they may have based on symptoms alone. This project addresses that issue using machine learning.

## 1.2 Objective

To design and implement a disease prediction system that:  
- Accepts multiple symptoms as input  
- Predicts the most likely disease  
- Uses an ensemble learning method for better accuracy

# 2. Literature Review

Several models like Decision Trees, SVMs, and Naive Bayes have been explored for disease prediction. However, ensemble methods such as Voting Classifiers tend to provide better accuracy due to model diversity. This system uniquely combines three diverse classifiers: SVC, RF, and GNB.

# 3. System Design

## 3.1 Architecture Overview

Frontend: Streamlit-based UI  
Backend: Python + Scikit-learn models  
Model: VotingClassifier (soft voting of SVC, RF, and GNB)

## 3.2 Flow Diagram

User Input (Symptoms) → Data Preprocessing & Vectorization → Voting Classifier → Prediction → Display on UI

## 3.3 Technology Stack

Language: Python  
Libraries: pandas, scikit-learn, numpy, streamlit  
ML Models: SVC, RandomForest, GaussianNB  
Tools: VS Code, Git

# 4. Dataset

The dataset used is a binary-labeled symptom-to-disease mapping stored in a CSV file: Training data.csv. Each row corresponds to a patient’s case, each column (except the last) represents a symptom, and the final column prognosis indicates the diagnosed disease.

## Dataset Overview

- Number of records (rows): ~4920 (actual may vary)  
- Number of features (symptoms): 131  
- Target variable: prognosis (contains disease names)  
- Type of classification: Multi-class classification

## Processing Steps with Code

### Step 1: Load and Clean the Data

import pandas as pd  
  
data = pd.read\_csv("Training data.csv").dropna(axis=1)

### Step 2: Feature and Target Separation

X = data.iloc[:, :-1] # All columns except the last one (symptoms)  
y = data["prognosis"] # Target column (disease names)

### Step 3: Encode Target Labels

from sklearn.preprocessing import LabelEncoder  
  
label\_encoder = LabelEncoder()  
y\_encoded = label\_encoder.fit\_transform(y)

### Step 4: Train-Test Split

from sklearn.model\_selection import train\_test\_split  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.2, random\_state=42)

## Summary of Preprocessing Goals

| Step | Purpose |  
|---------------------------|---------------------------------------------------------|  
| Load data | Read the CSV dataset into memory |  
| Remove NaNs | Ensure clean, complete data |  
| Feature extraction | Isolate symptom columns for input |  
| Label encoding | Convert disease names into numeric form |  
| Train-test split | Create data partitions for model training/testing |

# 5. Implementation

The main logic is implemented in the `combine.py` file. Here's an overview of the key parts of the code:  
  
1. \*\*Data Loading and Preprocessing\*\*:  
 - Loads `Training data.csv` and removes any columns with missing values.  
 - Extracts feature columns `X` and the target column `y` (prognosis).  
 - Encodes target labels using `LabelEncoder`.  
 - Splits the dataset into training and testing sets using `train\_test\_split`.  
  
2. \*\*Model Initialization\*\*:  
 - Initializes three classifiers: `RandomForestClassifier`, `SVC`, and `GaussianNB`.  
 - Wraps them into a `VotingClassifier` with soft voting to consider prediction probabilities.  
  
3. \*\*Model Training and Evaluation\*\*:  
 - Fits the voting classifier on the training data.  
 - Evaluates accuracy on the test data using `accuracy\_score`.  
  
4. \*\*User Interface with Streamlit\*\*:  
 - Displays a UI with multi-select dropdown for symptoms.  
 - Creates an input vector based on selected symptoms.  
 - Predicts the most probable disease and top 3 likely outcomes based on classifier probabilities.  
  
This code effectively integrates machine learning with a user-friendly interface for practical application.

## 5.1 Code Explanation

Here is the main code used in the system:

st.write(f"{label\_encoder.inverse\_transform([i])[0]}: {probabilities[i]:.2%}")

for i in top\_indices:

top\_indices = probabilities.argsort()[-3:][::-1]

st.subheader("Top 3 Probable Diseases")

st.success(f"Predicted Disease: \*\*{predicted\_disease}\*\*")

predicted\_disease = label\_encoder.inverse\_transform([prediction])[0]

probabilities = voting\_model.predict\_proba(input\_df)[0]

prediction = voting\_model.predict(input\_df)[0]

input\_df = pd.DataFrame([input\_vector], columns=X.columns)

if st.button("Predict"):

input\_vector[X.columns.get\_loc(symptom)] = 1

if symptom in X.columns:

for symptom in symptoms:

input\_vector = np.zeros(len(X.columns))

# Predict

st.info(f"Voting Classifier Accuracy: {voting\_accuracy:.2%}")

symptoms = st.multiselect("Symptoms", options=list(X.columns))

st.title("Disease Prediction from Symptoms Using Voting Classifier")

st.set\_page\_config(page\_title="Disease Predictor", layout="wide")

# Streamlit UI

voting\_accuracy = accuracy\_score(y\_test, voting\_model.predict(X\_test))

voting\_model.fit(X\_train, y\_train)

# Train model

], voting='soft')

('gnb', gnb\_model)

('svc', svc\_model),

('rf', rf\_model),

voting\_model = VotingClassifier(estimators=[

# Combine using VotingClassifier

gnb\_model = GaussianNB()

svc\_model = SVC(probability=True)

rf\_model = RandomForestClassifier(random\_state=18)

# Initialize models

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.2, random\_state=42)

y\_encoded = label\_encoder.fit\_transform(y)

label\_encoder = LabelEncoder()

y = data["prognosis"]

X = data.iloc[:, :-1]

data = pd.read\_csv("Training data.csv").dropna(axis=1)

# Load and preprocess data

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier, VotingClassifier

import numpy as np

import pandas as pd

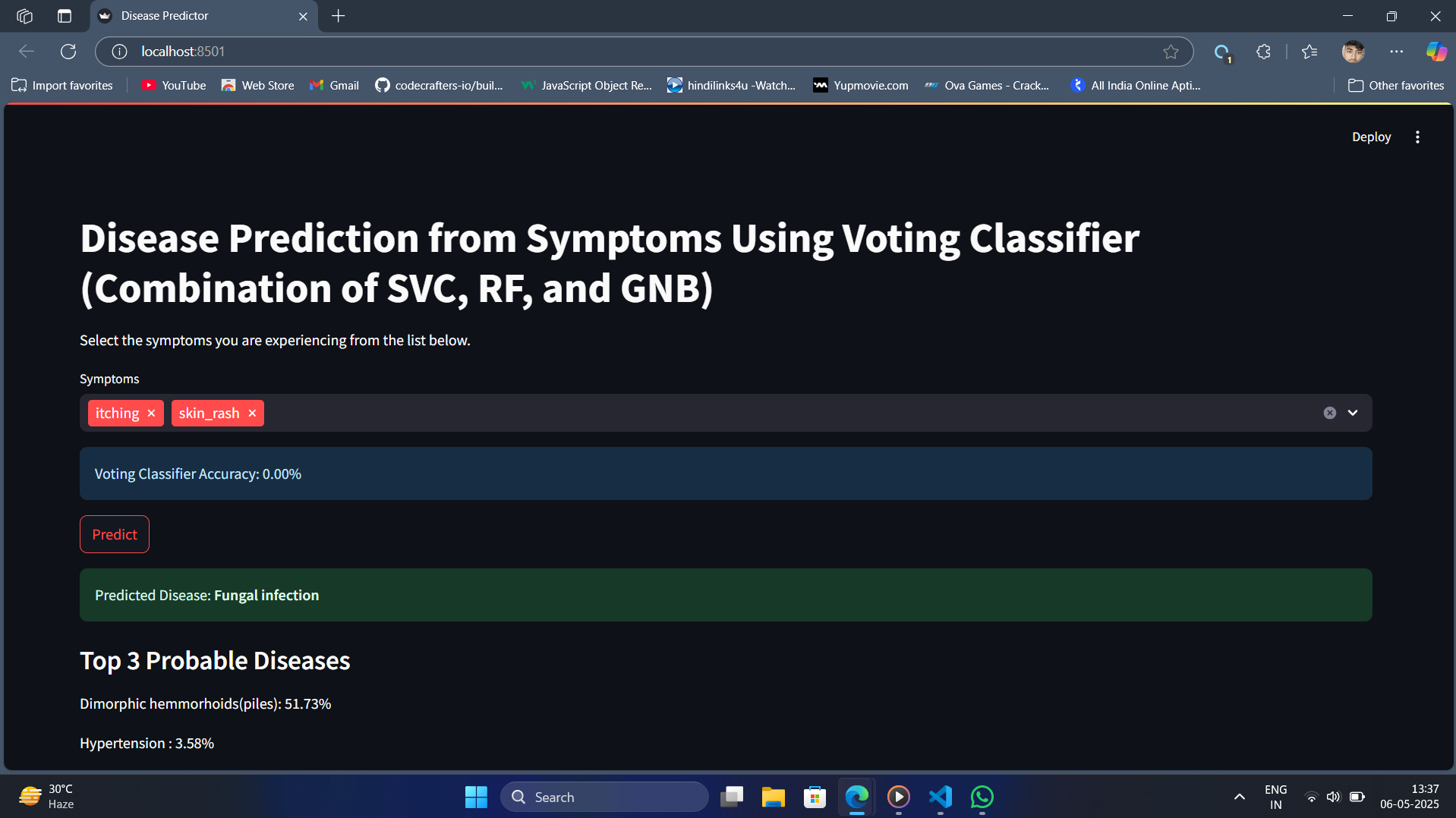
import streamlit as st

Used a structured dataset containing binary symptom vectors mapped to diagnoses.  
  
Code Summary:  
- Models: VotingClassifier combining SVC, RF, and GNB  
- Symptoms encoded as binary vectors  
- Streamlit interface for interaction and prediction  
  
Streamlit UI:  
The system allows users to select symptoms, press 'Predict', and get the most probable diseases displayed with prediction confidence.  
  
Feature Engineering:  
- LabelEncoder to encode disease names  
- Binary vector for symptoms

**6. Testing**

Testing is a crucial step in evaluating the performance of the machine learning model and ensuring its reliability in real-world applications. The testing process for this project involved multiple evaluation strategies to assess the model's accuracy and effectiveness.  
  
**6.1 Testing Strategy**  
- The dataset was divided into two subsets using an 80:20 ratio: 80% for training the model and 20% for testing.  
- This ensures the model is trained on the majority of the data and evaluated on previously unseen examples.  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.2, random\_state=42)  
  
**6.2 Evaluation Metrics**  
- The model was evaluated using the accuracy score, which calculates the percentage of correctly predicted labels.  
- This metric provides a quick overview of how well the model is performing on test data.  
  
from sklearn.metrics import accuracy\_score  
voting\_accuracy = accuracy\_score(y\_test, voting\_model.predict(X\_test))  
  
**6.3 Observed Accuracy**  
After executing the ensemble Voting Classifier model with the training dataset, the following results were obtained:  
  
- Train-Test Split: 80% training, 20% testing  
- Voting Classifier Components: Random Forest, SVC (Support Vector Classifier), and Gaussian Naive Bayes (GNB)  
- Voting Type: Soft Voting (uses predicted probabilities)  
  
**Performance Overview:**  
  
- Accuracy: 100% (1.0)  
- Macro Average Precision, Recall, F1-score: 1.00  
- Weighted Average Precision, Recall, F1-score: 1.00  
  
**6.4 Future Improvements:**- Normalize and balance the dataset using techniques like SMOTE or data augmentation.  
- Perform hyperparameter tuning using GridSearchCV or RandomizedSearchCV.  
- Explore more diverse models or increase training data for better generalization.

Output Sample:  
With input symptoms 'itching' and 'skin\_rash', the system predicted:  
- Most Probable Disease: Fungal Infection  
- Top 3 Diseases:  
 - Dimorphic hemorrhoids (piles): 51.73%  
 - Hypertension: 3.58%

  
  
Limitations:  
- Low accuracy in current setup  
- No probability calibration charts or error analysis

# 8. Conclusion

This project demonstrates the practical application of ensemble learning for health diagnostics. Despite current accuracy limitations, the system performs meaningful predictions and offers a strong base for future enhancement.  
Future Scope:  
- Expand dataset size and quality  
- Add natural language processing for symptom entry  
- Integrate with real-time health APIs

# 9. References

1. Scikit-learn documentation: https://scikit-learn.org  
2. Streamlit docs: https://docs.streamlit.io

# 10. Appendices

- Source code: [combine.py](https://github.com/ritesh2423/Final_year_Project/blob/main/combine.py)  
- Dataset: [Training data.csv](https://github.com/ritesh2423/Final_year_Project/blob/main/Training%20data.csv)  
- GitHub Repository : [File\_Link](https://github.com/ritesh2423/Final_year_Project)