PWSkills Mini-Hackathon Submission: Intelligent Health Predictor

Author: Ubaid Khan
Date: September 2, 2025

1. Introduction

The timely and accurate diagnosis of diseases is one of the most critical challenges in modern healthcare. Early detection can significantly improve patient outcomes, reduce healthcare costs, and alleviate the burden on medical professionals. This project, developed for the PWSkills Mini-Hackathon, tackles this challenge by building an "Intelligent Health Predictor"—a machine learning application that predicts one of 41 diseases based on a patient's reported symptoms.

The primary goal was to create a robust, end-to-end solution encompassing everything from data analysis to a live, interactive web application. The final product is not just a predictive model, but a user-friendly tool that demonstrates the potential of AI to assist in early-stage disease identification.

2. Methodology

Our approach was systematic, following industry best practices for a machine learning project to ensure a high-quality and reproducible result.

2.1 Data Preprocessing

The initial step involved a thorough cleaning and preparation of the provided `Training.csv` dataset.

- **Data Loading & Cleaning**: The dataset was loaded using Pandas. An extraneous, unnamed column (`Unnamed: 133`) was identified and removed. Whitespace was also stripped from all column names to prevent potential errors.
- **Target Variable Encoding**: The text-based `prognosis` column was converted into numerical labels using scikit-learn's `LabelEncoder` . This is a necessary step for most machine learning algorithms. The fitted encoder was saved for later use to decode predictions back into human-readable disease names.

2.2 Model Selection and Justification

A two-stage modeling approach was adopted to ensure high performance.

- **Baseline Model**: A `RandomForestClassifier` was initially trained to establish a baseline performance metric. On this clean dataset, it surprisingly achieved 100% accuracy on the validation set, indicating the presence of strong, learnable patterns in the data.
- **Advanced Model**: To demonstrate proficiency with more advanced, industry-standard tools, a **LightGBM (Light Gradient Boosting Machine)** classifier was chosen. LightGBM is renowned for its high speed and efficiency on tabular data. The goal was to build a second, high-performance model and compare its efficacy.

2.3 Hyperparameter Tuning

To optimize the LightGBM model, we employed **Optuna**, an advanced hyperparameter optimization framework.

- **Why Optuna?** Instead of a brute-force `GridSearchCV`, Optuna uses a more intelligent search strategy (Bayesian Optimization) to find the best hyperparameters much more efficiently.
- **Process**: An optimization study was configured to run for 50 trials, searching for the best combination of parameters like `n_estimators`, `learning_rate`, and `max_depth`. The final LightGBM model was then trained using these optimal parameters.

2.4 Deployment Procedure

A key requirement of the hackathon was to create a live, usable application.

- **Framework**: **Flask** was chosen as the web framework due to its simplicity and flexibility.
- **Structure**: The application was structured with a "flat" hierarchy to be compatible with the deployment platform, separating the `app.py` server, models, templates, and static files.
- **Platform**: **Hugging Face Spaces** was selected for deployment due to its excellent free tier for public projects and seamless integration with Git.
- **Configuration**: The deployment required specific configuration in the `README.md` file, setting the `sdk` to `docker` and defining the application file and port. This step was crucial for successfully running a custom Flask app in the Hugging Face environment.

3. Results

3.1 Model Performance

The final, tuned LightGBM model demonstrated exceptional performance.

- **Validation Accuracy**: 100% on the held-out validation set.
- **Final Test Accuracy**: 97.62% on the unseen `Testing.csv` data.

The slight drop in accuracy from the validation to the test set is a positive sign, indicating that the model generalizes well to new data and is not overfit.

3.2 Insights from Exploratory Data Analysis (EDA)

The EDA phase yielded several crucial insights that guided the project:

- **Perfectly Balanced Dataset**: The training data was found to be perfectly balanced, with exactly 120 samples for each of the 41 diseases. This simplified the modeling process, as techniques for handling class imbalance were not required.
- **"Symptom Signatures"**: Through visualization (specifically heatmaps), it was discovered that many diseases have a unique combination of highly correlated symptoms. For example, Jaundice was almost always associated with `yellowish_skin`, `dark_urine`, and `vomiting`. These strong patterns are the primary reason for the model's high accuracy.
- **Frequency vs. Importance**: A key finding was that the most frequent symptoms (e.g., fatigue) were not the same as the most predictive symptoms. The baseline model's feature importance plot revealed that rarer but more specific symptoms were often more valuable for making accurate predictions.

4. Discussion

4.1 Challenges Faced

- **Data Consistency Bug**: A critical bug was discovered where the model, trained on 132 features, would crash during prediction because the application was loading 133 features from the CSV file. This was traced to an unnamed column created by trailing commas and was fixed by implementing a more robust data loading function that explicitly drops the target column and any unnamed columns. This fix was applied across all scripts to ensure system-wide consistency.
- **Python Module Import Errors**: When transitioning from notebooks to a structured application, we faced a `ModuleNotFoundError`. This was resolved by correctly structuring the project as a Python package with `__init__.py` files and by using the `python -m src.app` command to run the application from the root directory, which correctly handles the system path.
- **Deployment Configuration**: The most significant challenge was deploying the Flask application on Hugging Face Spaces. The initial deployment was stuck on a "Starting" screen. The issue was debugged and traced to an incorrect SDK configuration (`gradio` instead of `docker`). The problem was solved by updating the `README.md` with the

correct Docker SDK configuration and modifying the Flask `app.run()` command to be publicly visible on the correct port.

4.2 Limitations and Potential Improvements

- **Dataset Cleanliness**: The dataset is exceptionally clean and well-balanced. While this is ideal for a hackathon, real-world medical data is often messy, imbalanced, and contains missing values. A future version of this project could be tested on more realistic data.
- **Feature Engineering**: The project currently uses the raw symptom data. Future improvements could involve engineering new features, such as symptom counts or interaction terms, to potentially capture more complex patterns.

4.3 Real-World Implications

The "Intelligent Health Predictor" serves as a powerful proof-of-concept. While not a replacement for a doctor, such a tool could be used as a preliminary diagnostic aid, helping individuals understand potential health issues based on their symptoms and encouraging them to seek professional medical advice sooner.

5. Conclusion

This hackathon was a comprehensive journey through the entire machine learning lifecycle. We successfully transformed a raw dataset into a highly accurate, tuned predictive model and deployed it as a polished, interactive web application. The final model's 97.62% accuracy, combined with the professional user interface, demonstrates a complete and effective solution to the problem statement. This project stands as a strong example of how data science can be leveraged to create practical tools with the potential for real-world impact.