Exo-Classifier: An AI-Powered Exoplanet Detection Application

Version: 1.0

Date: October 5, 2025

1. Project Overview

Exo-Classifier is a full-stack web application designed to identify exoplanet candidates from astronomical data using a machine learning model. The project leverages real data from the NASA Exoplanet Archive and employs an XGBoost (Extreme Gradient Boosting) classifier to distinguish between confirmed exoplanets and false positives.

The application provides an interactive front end where users can visualize sample light curves, submit their own candidate data, and receive an instant classification from the AI model. The goal is to create an accessible and educational tool that demonstrates the power of machine learning in modern astronomical research.

2. Core Features

- **Interactive AI Demo:** Visualize simulated light curves for a confirmed exoplanet and a false positive, complete with a mock AI classification.
- **Manual Exoplanet Classification:** A form allows users to input the 15 key parameters of a potential exoplanet candidate. The data is sent to the backend, processed by the trained XGBoost model, and the result is displayed back to the user.
- **Sample Data Buttons:** Two buttons automatically populate the form with data for a "Guaranteed Exoplanet" and a "Non-Exoplanet" (False Positive) to clearly demonstrate the model's decision-making capabilities.
- **Data Analysis Dashboard:** A section with mock data visualizations for dataset exploration, feature importance, and model performance metrics.

3. Technology Stack

The project is built using a modern stack for web development and data science:

- **Front End:**
  - **HTML5:** For the structure of the web page.
  - **Tailwind CSS:** For a utility-first, responsive, and modern design.
  - **JavaScript (ES6):** For user interaction, DOM manipulation, and API calls.
  - **Chart.js:** For rendering the light curve and data analysis charts.
- **Back End:**
  - **Python:** The core programming language.

    - **Flask:** A lightweight web framework to serve the application and handle API requests.
    - **Pandas:** For data manipulation and creating the DataFrame required by the model.
- **Machine Learning:**
    - **XGBoost:** The gradient boosting library used to train the classification model.
    - **Scikit-learn:** For data splitting and model evaluation (accuracy).
    - **Joblib:** For saving and loading the trained model object.

4. The Machine Learning Model

The heart of Exo-Classifier is the custom-trained XGBoost model that performs the classification.

4.1. Model Choice: XGBoost

We chose the XGBoost (Extreme Gradient Boosting) algorithm because it is highly effective for handling the complex, non-linear, and often imbalanced datasets found in NASA's exoplanet observations. Exoplanet detection is a binary classification problem, and XGBoost excels by combining multiple decision trees to learn intricate feature relationships. It efficiently handles missing or noisy data, which is common in astronomical datasets.

Furthermore, XGBoost is known for its speed, scalability, and ability to deliver high accuracy. It also provides interpretability by showing feature importance, helping us understand which attributes contribute most to a prediction.

4.2. Dataset Used

The model was trained on the official **Kepler Objects of Interest (KOI) Cumulative Table** from the NASA Exoplanet Archive.

- **Source:** NASA Exoplanet Archive
- **File Used for Training:** cumulative_2025.10.04_18.21.08.csv

4.3. Features Used in Training

Crucially, the final model was trained on **only the 15 features** that are collected by the web application's form. This ensures perfect compatibility between the front end and the model. The features are:

Form Input Name	**Dataset Column Name**	**Description**
koi_period	koi_period	Orbital Period [days]
koi_duration	koi_duration	Transit Duration [hours]
koi_prad	koi_prad	Planetary Radius [Earth

		radii]
koi_ror	koi_ror	Planet-to-Star Radius Ratio
koi_slogg	koi_slogg	Stellar Surface Gravity [log10(cm/s^2)]
koi_srad	koi_srad	Stellar Radius [Solar radii]
koi_impact	koi_impact	Impact Parameter
koi_insol	koi_insol	Insolation Flux [Earth flux]
koi_teq	koi_teq	Equilibrium Temperature [K]
koi_mass	koi_smass	Stellar Mass [Solar masses]
koi_snr	koi_model_snr	Transit Signal-to-Noise Ratio (SNR)
koi_density	koi_srho	Stellar Density [g/cm^3]
koi_time0bk	koi_time0bk	Transit Epoch [BKJD]
koi_dor	koi_dor	Planet-Star Distance / Stellar Radius
koi_incl	koi_incl	Orbital Inclination [degrees]

4.4. Training Process and Performance

The model was created using the train_compatible_model.py script, which performed the following steps:

1. Loaded the local cumulative.csv dataset.
2. Selected only the 15 features listed above and the koi_disposition target column.
3. Dropped rows with missing values in these critical columns to ensure data quality.
4. Encoded the target label (CONFIRMED -> 1, FALSE POSITIVE -> 0).
5. Split the data into an 80% training set and a 20% testing set.
6. Trained an XGBClassifier model on the training data.
7. Saved the final, compatible model as exo_classifier_compatible.joblib.

The final model achieved an accuracy of **89.78%** on the unseen test data.

5. How the Output Works

The model's output is its expert opinion based on the 15 features you provide.

- When you provide Correct Exoplanet Data:
  The model analyzes the relationships between the 15 features. If the data's "signature" matches the patterns of thousands of confirmed exoplanets it learned from, it will output:
  - **Prediction:** CONFIRMED EXOPLANET
  - **Confidence Score:** A high percentage (e.g., >90%).
- When you provide Non-Exoplanet Data:
  If you provide data that contains "red flags" (e.g., an object too large to be a planet, a transit that is too deep, or a very weak signal), the model will recognize that the pattern does not match a real exoplanet. It will output:
  - **Prediction:** FALSE POSITIVE
  - Confidence Score: A high percentage (e.g., >90%).
    A high confidence on a "False Positive" is a sign of an intelligent model, as it is confidently rejecting a bad signal.

6. Project Setup and Installation

To run this project on another device, follow these steps:

1. **Prerequisites:** Ensure you have Python 3.8+ and pip installed.
2. **Copy the Project:** Transfer the entire project folder to the new device. The folder must contain app.py, train_compatible_model.py, the /models folder with exo_classifier_compatible.joblib, and the /static and /templates folders.
3. Generate requirements.txt (On your original machine):
   Open a terminal in your project folder and run:
   ```
   pip freeze > requirements.txt
   ```
   This will create a list of all necessary libraries. Make sure this file is copied to the new device with the rest of the project.
4. Install Dependencies (On the new device):
   Open a terminal, navigate into the project folder, and run:
   ```
   pip install -r requirements.txt
   ```
5. Run the Application:
   Once the installation is complete, run the Flask app:
   ```
   python app.py
   ```