Prosperity Prognosticator: Startup Success Prediction

Author: Chaitanya Potnuri

Date: 17 February 2026

Project Type: End-to-End Machine Learning Web Application

---

1. Project Overview

The **Prosperity Prognosticator** is an analytical tool developed to mitigate the high risk associated with early-stage venture capital. By leveraging a **Random Forest** architecture, the system evaluates the health of a startup based on its funding trajectory and operational milestones to predict a binary outcome: **Success (Acquired)** or **Risk (Closed)**.

---

2. Detailed Project Lifecycle

Phase I: Data Acquisition & Exploratory Data Analysis (EDA)

The dataset was sourced from **Crunchbase**, consisting of historical records of global startups.

- **Target Mapping:** The status variable was identified as the label. Categorical values were transformed: acquired $\rightarrow$ 1, closed $\rightarrow$ 0.
- **Correlation Heatmapping:** Using Seaborn, a correlation matrix was generated to identify significant predictors. Factors such as "Number of Relationships" and "Total Funding" showed the strongest positive correlation with acquisition.

Phase II: Feature Engineering & Preprocessing

To ensure model stability, the raw data underwent several transformation steps:

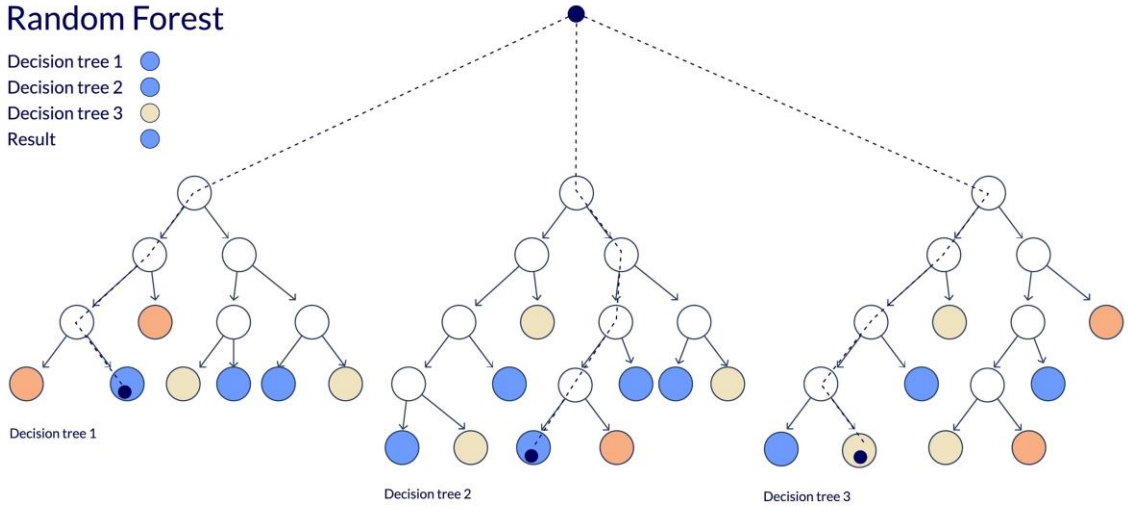
- **Outlier Mitigation:** The Interquartile Range (IQR) method was applied to normalize extreme values in funding amounts.
- **Imputation:** Missing values for milestone dates were imputed with 0.0. This treats the absence of a milestone as a valid feature (i.e., the milestone was never achieved) rather than missing data.
- **Feature Selection:** A final subset of **33 numeric features** was selected to represent each startup’s profile.

Phase III: Model Architecture & Training

The core intelligence was developed in a cloud-based **Kaggle** environment for optimized processing.

- **Algorithm: Random Forest Classifier**. This ensemble method was chosen for its ability to manage high-dimensional data and reduce variance.

- **Evaluation:** The dataset was split using an **80/20 train-test ratio**.
- **Performance:** The model achieved a **0.85 Accuracy Score** with balanced Precision and Recall for the "Acquired" class.
- **Serialization:** The final trained object was exported using the pickle library as random_forest_model.pkl.



Phase IV: Backend Development (Flask Framework)

A robust backend was engineered to bridge the gap between the static model and a live user interface.

- **Inference Logic:** The script app.py loads the model once upon server initialization.
- **Feature Padding:** A specialized function was implemented to handle user inputs. If a user provides partial data, the script "pads" the remaining features with 0.0 to maintain the **33-feature input vector** required by the Random Forest model.

Phase V: Web Interface & Deployment

The frontend was developed using **HTML5** to provide a seamless user experience.

- **Template Engine:** Flask's Jinja2 engine renders index.html for inputs and result.html for outputs.
- **Deployment:** The application is hosted locally using a Flask development server, allowing for real-time inference on localhost:5000.

---

3. Repository Directory Structure

A clean architecture is maintained to allow for professional version control (Git):

Plaintext

```
startup_pred/
├── app.py                  # Flask Application (Controller)
├── random_forest_model.pkl      # Serialized ML Model (Brain)
├── startup data.csv           # Training Dataset
├── Startup_Analysis.ipynb      # Research & Training Source
├── .gitignore                # Excludes .DS_Store and __pycache__
└── templates/                # Frontend Components
    ├── index.html            # User Input Form
    └── result.html           # Prediction Output
```

---

4. Installation & Execution Protocol

1. **Initialize Environment:**

DOS

```
pip install flask pandas numpy scikit-learn
```

2. **Execute Server:**

DOS

```
python app.py
```

3. **Access Application:**

Navigate to http://127.0.0.1:5000 in a standard web browser.

---

5. Future Scalability

- **NLP Integration:** Incorporating Natural Language Processing to analyze startup "Elevator Pitches."
- **Cloud Migration:** Transitioning from local hosting to an AWS EC2 instance or Heroku.
- **Real-time API:** Connecting the system to the LinkedIn API to verify founder experience in real-time.