Project Report: Prosperity Prognosticator

Abstract

Prosperity Prognosticator is a small machine-learning web application that estimates a startup's probability of success using early-stage funding and milestone indicators. A Random Forest classifier is trained in the included notebook and saved as `startup_rf_model.pkl`. The Flask app (`app.py`) loads the model and provides a form-based UI to collect inputs and show a probability plus a simple success/failure decision.

1. Problem Statement

Given a set of startup attributes (funding timeline, funding amount, milestones, relationships), predict whether the startup is likely to succeed.

2. Project Goals

- Train a binary classifier for startup outcome.
- Export the trained model to a reusable file.
- Build a Flask UI to test predictions.

3. Project Structure (Important Files)

- `app.py`: Flask server + prediction logic.
- `startup_rf_model.pkl`: Trained Random Forest model (Joblib).
- `templates/index.html`: Input form.
- `templates/result.html`: Output page.
- `reqiurements.txt`: Dependencies (note: filename is intentionally spelled this way in the repo).
- `startup-prediction-eda-model.ipynb`: EDA + training notebook.

4. Data and Features

4.1 Training Data

The notebook loads a CSV named `startup data.csv`. If this CSV is not present in the repo, it is only required to retrain the model. The Flask app does not need the CSV at runtime.

4.2 Target

The notebook trains on `data['labels']` (binary classification). The web app assumes:

- `1` = Success
- `0` = Failure

4.3 Model Inputs (9 Features)

The model uses the following numeric features (same order used in the app):

Feature	Meaning
`age_first_funding_year`	Age at first funding
`age_last_funding_year`	Age at last funding
`age_first_milestone_year`	Age at first milestone
`age_last_milestone_year`	Age at last milestone
`relationships`	Number of relationships
`funding_rounds`	Number of funding rounds
`funding_total_usd`	Total funding (USD)
`milestones`	Number of milestones
`avg_participants`	Average participants

No scaler is used for this Random Forest model.

5. Model Summary

The notebook trains a `RandomForestClassifier`, evaluates it with a train/test split, and reports standard classification metrics (accuracy, classification report, confusion matrix). It also uses grid search to find a stronger configuration and exports the best model:

- Export step: `joblib.dump(best_model, 'startup_rf_model.pkl')`

6. Application (Flask) Summary

6.1 How the App Works

- On startup, `app.py` loads `startup_rf_model.pkl`.
- The home page (`/`) shows the input form.
- Submitting the form (`/predict`) builds a 9-value feature vector and calls `predict_proba()`.

6.2 Output Format and Threshold

The app shows a message like:

```
the startup has XX.XX% of success, likely to be success/failure
```

Decision threshold (currently hard-coded in `app.py`):

- `success` if `success_prob >= 0.70`
- otherwise `failure`

7. How to Run (Local)

Detailed steps are in `README.md`. Summary:

1. Create and activate a virtual environment
2. Install packages from `reqiurements.txt`
3. Run the server
4. Open the browser

Default URL:

- `http://127.0.0.1:5000/`

8. Quick Test (Sample Input)

Use this sample to verify the UI and model connection:

Field	Value
Age at First Funding Year	2
Age at Last Funding Year	5
Age at First Milestone Year	1
Age at Last Milestone Year	4
Number of Relationships	10
Number of Funding Rounds	3
Total Funding (in USD)	1500000
Number of Milestones	2
Average Participants	5

Expected behavior:

- You should see a success probability and a label. The exact percentage can vary slightly by environment, but the app should run without errors and produce a reasonable result.

9. Limitations

- Inputs are cast directly to `float` (limited validation).
- The 0.70 threshold is not calibrated to a specific real-world cost/benefit.
- Model quality depends on the dataset and label definition used during training.

10. Conclusion

This project demonstrates a complete mini pipeline: training a Random Forest model, saving it with Joblib, and serving predictions through a simple Flask web interface.