Software Requirements Specification (SRS)

Twitter Fake Account Detector

Version 1.2

Prepared by:

Udit Jhanjhariya
Registration No: 23FE10CSE00640
Manipal University Jaipur
B.Tech CSE — 2027

August 8, 2025

Contents

1 Introduction

1.1 Purpose

The purpose of this Software Requirements Specification document is to provide a detailed, comprehensive description of the Twitter Fake Account Detector system. This system leverages machine learning techniques to classify Twitter user profiles as real or fake based on multiple profile features and user behavior metrics. The document is intended for the development team, project managers, testers, as well as stakeholders involved in the project lifecycle.

1.2 Scope

The Twitter Fake Account Detector is a standalone web application that provides real-time analysis of Twitter profiles to detect potentially fake accounts. Users input various profile statistics manually, which the system then processes via a backend machine learning model to classify the account's authenticity. The application focuses on usability, accuracy, and scalability but does not interface directly with Twitter's API.

1.3 Definitions, Acronyms, and Abbreviations

ML	Machine Learning
RF	Random Forest (Classifier)
API	Application Programming Interface
SRS	Software Requirements Specification
CSRF	Cross-Site Request Forgery
JSON	JavaScript Object Notation
REST	Representational State Transfer

1.4 References

- scikit-learn documentation - `https://scikit-learn.org/stable/`
- Flask web framework - `https://flask.palletsprojects.com/`
- Bootstrap CSS framework - `https://getbootstrap.com/`
- Joblib for model serialization - `https://joblib.readthedocs.io/en/latest/`

1.5 Overview

This document covers the overall description of the system, detailed functional and non-functional requirements, external interfaces, and appendices discussing data features and performance metrics.

2 Overall Description

2.1 Product Perspective

The system is designed as a web-based standalone application with a clear separation between the frontend, backend, and the machine learning model. It uses Flask as the backend server, providing RESTful endpoints which interact with the frontend built using HTML, CSS (Bootstrap), and JavaScript (jQuery). The machine learning component is encapsulated in the backend, using a Random Forest classifier trained on a labeled dataset of real and fake Twitter profiles.

2.2 Product Functions

- User input interface to accept various profile features such as tweet count, followers count, following count, favorites, listed count, account age, bio length, language, and gender.
- Preprocessing inputs, including normalization, encoding categorical features such as language and gender.
- Feeding processed features into the pretrained Random Forest model for classification.
- Returning prediction results with classification label and confidence scores.
- Displaying user-friendly visual output on the frontend.

2.3 User Characteristics

Intended users include:
- Researchers and analysts studying social media fraud.
- Educators and students learning about machine learning applications.
- Security specialists monitoring social media authenticity.
- General users curious about the authenticity of Twitter profiles.

Users should have basic familiarity with web browsing and interpreting simple analytical results.

2.4 Constraints

- The system currently requires manual entry of profile features; no direct API integration for live data retrieval.
- Performance depends on the accuracy and quality of input data.
- The backend requires Python 3.8+ and installed dependencies.
- Model updates require retraining offline and redeployment.

2.5 Assumptions and Dependencies

- Users provide honest and accurate profile feature inputs.
- The deployed environment supports Flask and Waitress servers.
- Dependencies like scikit-learn, pandas, gender-guesser are installed correctly.
- Network connectivity for web frontend-backend communication.

3 External Interface Requirements

3.1 User Interfaces

- The user interface is a responsive web form with labeled fields for all required Twitter profile metrics.
- Includes dropdowns for gender and language selection with default values.

- Provides validation feedback for input errors with tooltip hints.
- The prediction result includes clear labels with confidence percentages and color-coded indicators (green for real, red for fake).

3.2 Hardware Interfaces

The application runs on standard server hardware, requiring:
- Minimum CPU: Dual-core 2.0 GHz
- RAM: 2 GB
- Disk Space: 1 GB for model and datasets

3.3 Software Interfaces

- Backend: Flask web framework, Python 3.8+, Waitress WSGI server
- Machine Learning: scikit-learn Random Forest model serialized with Joblib
- Frontend libraries: Bootstrap 4+, jQuery
- Gender Detection: `gender-guesser` library for estimating gender from first names

3.4 Communication Interfaces

- HTTP/HTTPS protocols for frontend-backend communication
- RESTful POST endpoint `/predict` receives JSON payload or form-data with profile features
- Responses are JSON objects containing classification and confidence data

4 System Features

4.1 Feature 1: Profile Data Input

Description: The system provides a form for users to submit key Twitter profile metrics.
Functional Requirements:
- The form must accept integer inputs for:
  - Tweet count
  - Followers count
  - Following count
  - Favorites count
  - Listed count
  - Account age in days
  - Bio length
- Dropdowns for:
  - Gender: Female (-2), Mostly Female (-1), Unknown (0), Mostly Male (1), Male (2)
  - Language (mapped to language codes)
- Validate inputs are within reasonable min/max ranges:

  - e.g., Tweets ¿= 0, Followers ¿= 0, Age ¿ 0 etc.

- Provide user feedback on invalid inputs.

4.2 Feature 2: Data Preprocessing

Description: Extract and encode features suitable for model input.
Functional Details:

- Extract user's first name from full name and use `gender-guesser` if provided.
- Encode gender as numeric code.
- Map language input to numeric code using internally maintained dictionaries.
- Normalize feature scales if required.

4.3 Feature 3: Predictive Model Inference

Description: Use the pretrained Random Forest model to classify the Twitter profile.
Requirements:

- Load model at startup from `twitter_fake_account_detector.joblib`.
- Accept feature vector and return predicted label (Fake or Real).
- Return prediction confidence probabilities for both classes.
- Handle prediction errors gracefully.

4.4 Feature 4: Result Presentation

Description: Display prediction results with clear interpretation.
Requirements:

- Show returned label in textual form: "Real Account" or "Fake Account".
- Display confidence percentage for predicted class.
- Show probabilities side-by-side for Fake and Real classes.
- Use visual cues: color coding, icons for enhanced UX.

4.5 Feature 5: Error Handling

- Provide friendly error messages for missing inputs or system issues.
- Log errors internally for diagnostics.

5 Non-functional Requirements

5.1 Performance Requirements

- Response time for prediction shall be less than 2 seconds for 95% of requests.
- Support at least 10 concurrent users without degradation.
- Model loading at server start shall complete within 5 seconds.

5.2 Security Requirements

- Sanitize all user inputs to prevent injection attacks.
- CSRF tokens or SameSite cookies to prevent CSRF.
- Avoid logging personal data.

- Secure backend access via server-side validation.

5.3 Reliability and Availability

- 99% uptime excluding scheduled maintenance.
- Failover for server errors with appropriate user notifications.

5.4 Scalability

- Design for horizontal scaling with stateless backend.

5.5 Maintainability

- Codebase must follow PEP8 Python standards.
- Maintainable modularized code for backend and frontend.
- Well-documented API endpoints and model components.

5.6 Portability

- Support deployment on Windows, Linux, and macOS servers.
- Use portable dependencies with clear requirements documentation.

6 System Architecture

6.1 Component Descriptions

- **Frontend:** Responsive UI built on Bootstrap and jQuery, responsible for collecting user inputs and presenting results.
- **Backend Server:** Flask application hosting REST endpoints, performing pre-processing, model inference, and result formatting.
- **Machine Learning Module:** Random Forest model loaded from serialized joblib file; encapsulated in a class with training, prediction, and serialization methods.
- **Data Storage:** Persistent model file, static assets; no user data storage.

7 Data Design and Management

7.1 Training Dataset

- Two labeled datasets: `realusers.csv` and `fakeusers.csv`.
- Each contains profile features such as statuses_count, followers_count, lang, description, and preprocessed derived features.
- Dataset characteristics:
  - Real users: approximately 1,481 samples.
  - Fake users: approximately 1,337 samples.
  - Balanced to avoid bias.

7.2 Feature Engineering

- Gender extracted from user names using `gender-guesser` with category mapping.
- Language codes mapped from distinct `lang` string values.
- Ratios like statuses per day, friends/followers ratio computed as additional features.

7.3 Data Validation

- Handle missing datasets columns by dropping columns with excessive nulls.
- Validate input feature ranges during prediction.

8 Model Training and Evaluation

8.1 Model Selection

- Chosen model: Random Forest Classifier.
- Justification: Robust to noisy data, handles feature importance, non-linear separations.

8.2 Training Procedure

- Split data into 80% training, 20% testing subsets randomly.
- Train Random Forest with 100 estimators, random seed fixed for reproducibility.
- Apply cross-validation for hyperparameter tuning (optional).

8.3 Performance Metrics

- Accuracy, precision, recall, F1-score calculated.
- ROC curve and AUC plotted for performance visualization.

8.4 Model Outputs

- Model saved using `joblib` to `twitter_fake_account_detector.joblib`.
- Performance reports saved to `model_results.txt`.

9 API Specification

9.1 /predict Endpoint

- **Method:** POST
- **Input:** Form data or JSON object with:

```
{
    "gender": Integer [-2, 2],
    "tweets": Integer,
    "followers": Integer,
    "following": Integer,
    "favorites": Integer,
    "listed": Integer,
    "language": Integer (encoded)
}
```

- **Output:** JSON with predicted label and confidence:

```
{
    "prediction": "Real Account" / "Fake Account",
    "confidence": "95.13%",
    "fake_probability": "4.87%",
    "real_probability": "95.13%"
}
```

- **Error Responses:** HTTP 400 with JSON error message.

9.2 Usage Examples

Listing 1: Example curl request

```
curl -X POST http://127.0.0.1:8080/predict \
  -d "gender=1" \
  -d "tweets=2345" \
  -d "followers=432" \
  -d "following=500" \
  -d "favorites=100" \
  -d "listed=12" \
  -d "language=0"
```

10 User Interface Design

10.1 Design Goals

- Intuitive data entry with clear labeling.
- Responsive design for desktop and mobile.
- Dark theme interface for user comfort.
- Real-time feedback and error validation.
- Clean, distinguishable result display.

10.2 Accessibility

- Support keyboard navigation.
- Provide sufficient color contrast.
- Use ARIA labels for screen reader compatibility.

11 Testing and Validation

11.1 Test Plan

- **Unit Testing:** Individual modules including data preprocessing, model loading, and prediction.

- **Integration Testing:** Backend API endpoint testing with mocked input data.
- **System Testing:** End-to-end tests including frontend form submission and response rendering.
- **Performance Testing:** Load testing via concurrent simulated requests.
- **User Acceptance Testing (UAT):** Collect user feedback on UI and prediction usefulness.

11.2 Automated Testing

- Utilize `pytest` for backend unit and integration tests.
- Continuous Integration pipeline integration suggested.

11.3 Validation Criteria

- Model accuracy ≥ 0.90 on test sets.
- Response times under 2 seconds.
- Zero critical bugs on UAT.

12 Installation and Deployment

12.1 Prerequisites

- Python 3.8 or above installed.
- Required Python libraries listed in `requirements.txt` installed via `pip`.
- Network access for web server.

12.2 Setup Instructions

1. Clone the repository:

   ```
   git clone https://github.com/yourusername/Twitter-Fake-Profile-Detection
   cd Twitter-Fake-Profile-Detection
   ```

2. Create and activate virtual environment:

   ```
   python -m venv venv
   source venv/bin/activate  # Windows: venv\Scripts\activate
   ```

3. Install dependencies:

   ```
   pip install -r requirements.txt
   ```

4. Run the application:

   ```
   cd model
   python app.py
   ```

12.3 Deployment Considerations

- Use Waitress or Gunicorn for production WSGI hosting.
- Configure firewall and HTTPS in production.
- Schedule periodic model re-training as new data becomes available.

13 Maintenance and Support

13.1 Maintenance Tasks

- Updating dependencies and patching security vulnerabilities.
- Retraining the model periodically or when better datasets become available.
- Improving UI/UX based on user feedback.
- Monitoring server performance and uptime.

13.2 Support Contacts

- Maintainer: Udit Jhanjhariya
- Email: udit.jhanjhariya@example.com
- GitHub Issues: `https://github.com/yourusername/Twitter-Fake-Profile-Detection/issues`

14 Appendices

14.1 Appendix A: Feature Descriptions

Feature	**Description**
statuses_count	Number of tweets issued by the profile
followers_count	Number of users following the profile
friends_count	Number of accounts the profile is following
favourites_count	Number of tweets the profile has favorited
listed_count	Number of public lists that include the profile
age_in_days	Number of days since account creation
length_of_bio	Number of characters in the user's bio
Sex Code	Numeric gender code from name-based prediction
lang_code	Numeric encoding of user's interface language

14.2 Appendix B: Data Sample and Statistics

The training dataset consists of a combined set of labeled real and fake Twitter user profiles. Example statistics include:

- Real accounts exhibit higher followers, tweets, and bio completeness.
- Fake accounts tend to have fewer followers but higher friend counts relative to followers.
- Fake accounts are generally newer (low age_in_days).

14.3 Appendix C: Model Performance Metrics

- Training accuracy: approximately 91%
- Test accuracy: approximately 90%
- Precision, Recall, F1-score and confusion matrices indicating good class separation.

Glossary

False Positive: A real Twitter profile incorrectly classified as fake.
False Negative: A fake Twitter profile incorrectly classified as real.
True Positive: A fake profile correctly classified as fake.
True Negative: A real profile correctly classified as real.

End of Software Requirements Specification Document