Bot Detector - Documentation

1. Introduction

The **Bot Detector** is a machine learning-powered system designed to identify bot accounts on **X (formerly Twitter)**. By analyzing user profiles, tweets, and engagement metrics, the system classifies accounts as either **human or bot**. The system provides a **confidence score** and structured outputs to enhance accuracy and transparency.

Developed by **Team Cyfer-Trace, SSPU**.

---

2. System Architecture

2.1 Overview

The system is built using a **backend API**, **Streamlit frontend**, and a **machine learning-based classification model**. The primary components include:

- **Data Collection Module** - Fetches user profile and tweet data using the X API.
- **Preprocessing Module** - Cleans and structures the data.
- **Feature Extraction Module** - Extracts key metrics for classification.
- **Machine Learning Model** - Analyzes extracted features and classifies accounts.
- **API & Frontend** - Provides user-friendly interaction through CLI, web, or API calls.
- **Error Handling Module** - Manages API rate limits, authentication errors, and data retrieval failures.

2.2 Technology Stack

- **Backend:** FastAPI
- **Frontend:** Streamlit
- **ML Frameworks:** Scikit-learn, Transformers, FastText
- **Storage:** JSON-based config and model files
- **API Access:** Tweepy for X API integration

---

3. Data Collection & Processing

3.1 Data Sources

- **X API**: Fetches profile details, tweet history, and engagement metrics.
- **Twitter-Bot Detection Dataset**: Used for model training.

3.2 Feature Engineering

Key features used for bot classification include:

- **Numerical Features**: Follower count, retweet count, mentions, likes.
- **Behavioral Patterns**: Posting frequency, response time, URL sharing.
- **Text-Based Features**: Hashtag usage, sentiment analysis, linguistic patterns.
- **Account Metadata**: Profile age, username patterns, bio analysis.
- **Network Analysis**: Connection with other known bot accounts, retweet networks.

3.3 Data Preprocessing

- **Removing Noisy Data**: Eliminating unnecessary symbols, numbers, and unrelated words.
- **Normalization**: Converting text to lowercase and handling abbreviations.
- **Tokenization**: Splitting text into words and phrases for analysis.

---

4. Model & Classification

4.1 Models Used

- **DistilBERT** - Extracts textual patterns from tweets.
- **FastText** - Analyzes word embeddings for bot-like language.
- **LightGBM** - Classifies accounts based on extracted numerical and textual features.
- **Random Forest** - Secondary model to verify results and reduce biases.
- **Neural Networks** - Experimental usage for deep feature extraction.

4.2 Model Pipeline

1. **Preprocessing** - Text tokenization, normalization.
2. **Feature Extraction** - TF-IDF for text, statistical features for activity.
3. **Classification** - LightGBM assigns bot probability.
4. **Ensemble Learning** - Combining multiple models to increase accuracy.

4.3 Confidence Scoring

The system outputs a **bot probability score** between **0 and 1**, where values closer to **1** indicate higher bot likelihood. Multiple models provide a weighted confidence score for better precision.

---

5. System Usage

5.1 Running the Backend API

```
uvicorn app:app --host 0.0.0.0 --port 8000
```

5.2 Testing via API

```
import requests
response = requests.post(
    "http://localhost:8000/detect-bot/",
    json={"profile_link": "https://x.com/example_user"}
)
print(response.json())
```

5.3 Running the Streamlit Frontend

```
streamlit run frontend.py
```

5.4 Running the CLI Frontend

```
python cli_frontend.py
```

---

6. API Errors & Solutions

6.1 Common API Errors

- **401 Unauthorized**: Invalid or missing API key.
- **429 Too Many Requests**: Rate limit exceeded.
- **500 Internal Server Error**: Temporary API failure or issue in request structure.

6.2 Solutions

- **401 Unauthorized**: Ensure your API key is correct and configured with the necessary permissions.
- **429 Too Many Requests**: The system includes automatic retry mechanisms and rate limit handling to avoid disruptions. Implementing request throttling can prevent excessive calls.
- **500 Internal Server Error**: Verify that the request format matches the API's expected input and retry later.

---

7. Limitations & Future Improvements

7.1 Current Limitations

- API Rate Limits restrict real-time analysis.
- Model biases may exist due to dataset imbalances.
- Difficulty in detecting **adaptive bots** that evolve their behavior over time.

7.2 Future Enhancements

- Expanding to **Instagram bot detection**.

- Fine-tuning models using **active learning techniques**.
- Implementing **real-time bot behavior monitoring**.
- Enhancing **graph-based bot detection** to analyze interactions and network structure.
- Improving **deep learning integration** for better language understanding.
- Developing **a browser extension** for real-time bot detection.
- Incorporating **multi-platform bot tracking** to detect bot activity across different social media networks.

---

8. Contact & Support

For questions or collaboration opportunities, contact **codeitishant@gmail.com**.

---

9. License

This project is licensed under the **MIT License**.