Bot or not: modeling HTTP traffic to classify humans and bots

Trey Moen[1]*

Abstract
In today's Internet-connected world, there is an immense amount of internet traffic. It may seem like most traffic is composed of normal people on social media and e-commerce platforms, there is a large amount of web traffic coming from robots — programs written in languages like Python and Node.JS that scrape valuable data like user info and product availability. The ethics of such practices aside, these programs put additional load onto company infrastructure, forcing businesses to scale up their hosting capabilities, costing them additional funds to simply ensure their customers can access their resources and websites quickly. Machine learning can help identify threat actors *before* accessing certain resources, allowing businesses to classify bot traffic and prevent access. In this paper, I explore a method on how to identify bot traffic based on overall session and IP behavior, and implement a basic interface to block bots in a Flask web server.

Keywords
Bot detection — Python — Machine Learning

[1]*Department of Computer Science, George Fox University, Newberg, Oregon*
***Corresponding author:** tmoen18@georgefox.edu

Contents

1. Introduction

I knew I wanted to attempt a bot detector when we were assigned this project. I used to write web scrapers almost daily, looking at content from various web pages and trying to evade the types of protection mechanisms I attempted at creating in this paper. My goal was to use this prior knowledge and the help of other researchers to help build a feature set that I knew would help categorize humans versus bots.

2. Background

This project primarily revolved around creating a loosely-coupled adapter for a Flask web application. This would enable a new user to simply add a few lines of code to their Flask app and get protection from bots. The final model is saved locally so that it is easy to load into a new UWSGI thread (the backbone of Flask's web server).

3. Framework

I went with a simple framework to test out my HTTP bot protection

system. I created a Python library that exposes a few functions to the end user that hook into existing Flask routes and add a boolean as to whether the request ought to be blocked or not. The end goal was to make it as easy to integrate into a Flask application as adding the following to a Flask project:

```
# Add request collector
@app.before_request(request_handler)
# Add response collector
@app.after_request(response_handler)

# Add to protect endpoint
@protected_endpoint
```

```
@app.route('/')
def home(blocked: bool):
  if blocked:
    # Handle bot
  else:
    # Handle human
```

This interface makes it incredibly simple for a developer (or client) to integrate a bot prevention mechanism.

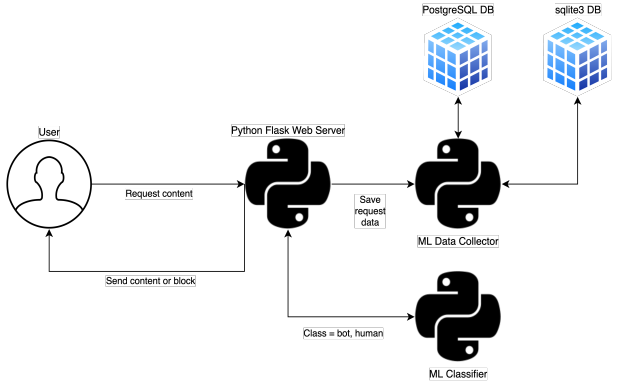
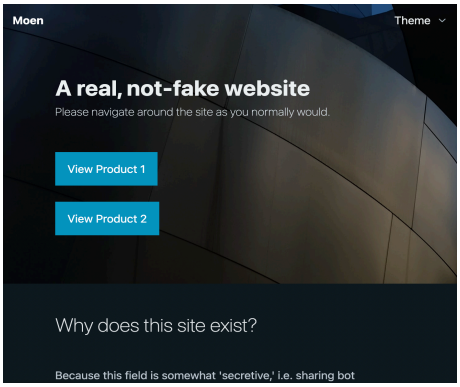


Diagram of framework

Above is a diagram of the entire application. It consists of a simple Flask web server, a Collector component, and a Classifier component. For each request the application receives, it logs some statistics about the request. Next, the Classifier takes what we currently know about a given HTTP session (identified by a Session ID cookie) and makes a prediction about whether the session is a bot or a human. Finally, when we are about to send a response back, the Collector analyzesthe response and records information and records information about the response. The "blocked" attribute is passed to the Flask route handler so the programmer can use it when handling the response, i.e. if the Classifier wants to block this request, then we can provide old, stale data or even a block HTML page.



Sample app homepage

4. Evaluation

4.1 Feature selection

Selecting the right features to delineate humans and bots is a difficult process. The feature set I gathered was heavily influenced by several research papers [1] [2] [3]:

- Number of requests total
- Number of bytes requested
- Number of GET requests
- Number of HEAD requests
- Number of POST requests
- Number of HTTP 3xx Codes (cached)
- Number of HTTP 4xx Codes (client error)
- Maximum number of requests for one page
- Average number of requests per page
- Standard deviation of page depth
- Maximum consecutive requests for one page
- Percent consecutive requests for one page out of all
- Overall session time
- Browse speed (num requests / session time)
- Average time between requests
- Standard deviation of time between requests
- Percent of requests with the Referer header
- Percent of requests without the Referer header

There were a number of features I added from prior experience and through attempting to implement the pipeline that provide signals as to whether the client is a human or bot:

- IP
- Session ID (cookie)
- Number of sessions on IP
- Number of unique User Agents
- Number of unique Referer headers
- Number of unique HTTP header hashes

A large number of sessions from one IP likely tells us that there is a cluster of bots running on a given IP. However, it's not an entirely important metric because of cases like large organizations that many clients will connect through, so this feature should be utilized with caution. However, for a given session, if the number of unique user agents or HTTP Referer headers change, then one can presume the session is utilizing header randomization attacks. Another metric I used is the HTTP header "hash." This technique is another way to identify if a client is randomizing the HTTP headers it is sending to the server, and an anomalous value here is also a prime signal as to whether the client is a bot.

There were a number of features I *wanted* to add into the model that give greater clarity into the types of entities that attempt to connect to the server, but these would require additional layers of complexity than what this final project allowed for. These include:

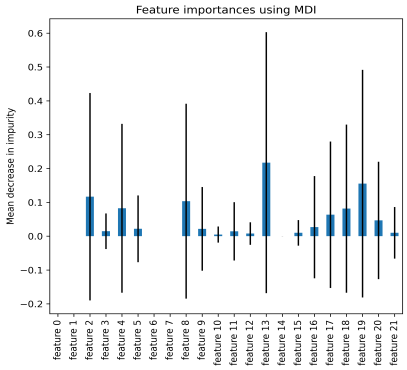
- JA3 hash [4]
- "Picasso" fingerprint [5]

The JA3 hash is a fingerprint of the TLS ClientHello packet. It records a number of attributes like the SSL/TLS version and supported cipher suites. This info is computed and MD5 hashed to create a "fingerprint" of the client making the request. JA3 hashes can be tied to specific browsers like Tor, Google Chrome, and Firefox, as well as programming language request libraries like Python Requests and Node.js fetch. Finding a number of varying JA3 hashes from a specific session or IP will flag an IP or session as a bot (or, finding a JA3 hash associated with a specific blocklist of known scraping programs).

The "Picasso" fingerprint is a method of client-side fingerprinting that relies on the HTML5 Canvas. It relies on the stable yet random noise a browser implementation gives off in order to classify it amongst many different devices and browsers. This is one way to identify if one session (or many sessions from an IP) is being used across many different browsers (e.g. a session initiated in Chrome is later copied to headless Chrome). This would help identify bots manipulating and transitioning between different browsers.

These two additional metrics could severely narrow down the type of clients that are obviously bots.

4.2 Feature significance



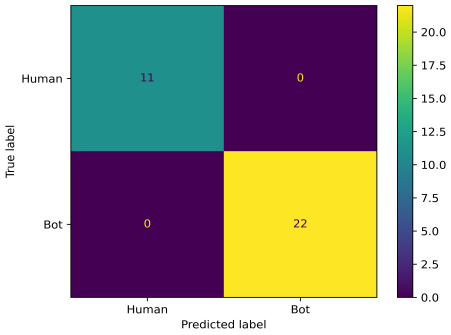
Feature significance

According to my small dataset, Feature 13 (maximum number of consecutive requests to one page) held the greatest importance out of all features recorded. From my past experience, this makes a lot of sense — many web scrapers will repeatedly scrape the same web page over and over, trying to get the latest information from a given resource.

4.3 Classifier

Because this problem lends itself to classification by segmenting chunks of data based on the feature's value, a Decision Tree Classifier makes the most sense to start with. I tried a Random Forest Classifier, utilizing a cluster of Decision Trees, trained on separate parts of the data set, and voting on the final classification. As you will see later, this approach did very well, and I did not need to try out other classifiers.

4.4 Results



Final result: 100% accuracy

After running the training and testing data through the classifier, it classified from 96.9% to 100% depending on the seed — a pretty good outcome!

4.5 Shortcomings and considerations

As I noted on my collector webpage, I could not find a large corpus of HTTP request logs that met my needs, so I needed to collect and generate my own human and bot traffic. I created a sample implementation of my collector onto a Flask web server and had my classmates interact with the website as they normally would browse the Internet, and collected their behavioral patterns that helped influence my model's outcome. In this "production" server, I could easily classify all traffic as "human" because I knew only my classmates accessed this page. As well, I wrote a "quick-and-dirty" Python `requests` web scraper employing various techniques like random delays, header randomization, and varying navigation patterns, and recorded the request history locally. I could label all this local traffic as "bot." Through this, I had my corpus of model training data.

Because of this, however, my model was very skewed toward my classmates' browsing behavior as well as my one robot web scraper and its randomization techniques. To make this model especially robust, I need hundreds of thousands or millions of requests to summarize data on, in addition to more intelligent strategies on session isolation (e.g. make a session expire after several hours).

5. Conclusion and reflections

I enjoyed writing this program and technology stack. It has been a dream of mine to create this program for some time, and what better reason to start it than use it as assigned coursework! I learned an immense amount about building Flask "plugins," collecting machine learning training data "at scale" (e.g. creating a solution that could scale), and doing research on techniques that others have tried and attempting to synthesize many techniques into one solution. I wish I had more time to continue developing this stack as well as find another large corpus of HTTP requests that would fit my needs; I hope to continue this past the end of this course. Thank you to all who helped generate human behavior on my website, I couldn't have formulated these results without you!

References

[1] C. Iliou, T. Kostoulas, T. Tsikrika, V. Katos, S. Vrochidis, and Y. Kompatsiaris, "Towards a framework for detecting advanced web bots," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 2019, pp. 1–10.

[2] G. Suchacka, "Analysis of aggregated bot and human traffic on e-commerce site," in *2014 Federated Conference on Computer Science and Information Systems*. IEEE, 2014, pp. 1123–1130.

[3] G. Suchacka and I. Motyka, "Efficiency analysis of resource request patterns in classification of web robots and humans." in *ECMS*, 2018, pp. 475–481.

[4] J. Althouse, J. Atkinson, and J. Atkins, "Ja3 - a method for profiling ssl/tls clients," https://github.com/salesforce/ja3, 2017.

[5] E. Bursztein, A. Malyshey, T. Pietraszek, and K. Thomas, "Picasso: Lightweight device class fingerprinting for web clients," in *Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2016.