Bot Detection via Mouse Mapping

Evan Winslow
Stanford University
Stanford, CA
ewinslow@cs.stanford.edu

Abstract
Bot detection has become a key need on the internet for a variety of reasons. Take as one motivating example online registrations. If bots are not prevented from registering automatically, then spam invariably becomes a serious problem. Also consider polling data. If bots are allowed to participate in polls, then poll data can be easily and dramatically skewed by bot participation. The first step to preventing their participation is detecting their presence. In this paper, I present a method for detecting bots called mouse mapping, which relies on UI-based features to distinguish between human and bot activity.

Introduction

In this paper I present a technique called mouse mapping, which relies on user interface-based features of web activity to distinguish between bot and human traffic. Some examples of UI features that could be extracted are clicks, mouse movements, typing, and screen size. I derived the name mouse mapping from a related practice called click mapping, in which a website administrator records the timing and location of clicks on various pages. The difference between that and mouse mapping is that I have intended mouse mapping to be broader. Mouse mapping records all mousemove events as well as clicks.

Motivation

Many current defenses against spam rely on detecting the presence of spam itself. However, if the creators of the spam can be detected before the spam is ever created, then the would-be spam becomes a moot issue. This is part of the usefulness of CAPTCHA, for example. CAPTCHA aims to prevent spam from ever being created by presenting the machine with a problem it cannot solve, but humans can. However, the weakness of CAPTCHA is that it adds a slight barrier to the genuine user. I propose mouse mapping a complementary method to CAPTCHA for detecting unusual activity on the web. Mouse mapping has the same advantage of preventing spam before it happens, but the advantage that mouse mapping has over CAPTCHA is that the presence of a mouse mapping defense is invisible to the genuine user. That is, a mouse mapping defense can be in place on a website, but the everyday user would never know it because data collection happens behind the scenes, with no appreciable effect on web application responsiveness and therefore no effect on user experience.

Problem Description

The goal of mouse mapping is to determine whether a user is behaving "correctly." In this case a user can be one of two classes: either bot or human. If the user is behaving "correctly," then they would be allowed to continue using the website as normal, but in the event they are not behaving "correctly," some action may be taken. However, we run into the problem of defining what "correct" behavior is for users on a website. We already have an intuition that human-like should be more correct than bot-like, but then what is human-like UI activity? This is where machine learning techniques come in. Instead of setting arbitrary thresholds and requirements for

UI activity, we allow real users to determine for us what "correct" behavior looks like, and then build a classifier based on that input data.

Features

There are several aspects of GUI interaction from which features can be extracted to distinguish between bots and humans. These include but are not limited to: mouse clicks (position and frequency), mouse movements, time spent on a page, screen resolution, and keystroke activity. Since clicking is commonly tracked already, I decided to focus mainly on extracting mouse movement features. Initially I started with a proliferation of features: too many to keep track of and too many to be useful. Many of the features were obviously linearly dependent or deterministic, not varying significantly at all even among human users, so I dropped a significant number of features before settling on the final seven. The final set of features I used was:

1) The number of move segments generated on a page. A move segment is the line between two successive records of a mousemove event generated by the browser which occur less than 500 milliseconds apart. This seemed useful to try because computers, assuming they used the mouse at all, would most likely take the straightest and shortest path and therefore have the least number of segments per page view on average.
2) The number of distinct mouse motions. A motion is defined as any string of successive segments that had no more than 500 milliseconds between points. This essentially captures the number of times the mouse is "rested" while the user was viewing the page.
3) The average length of these mouse motions. This is a measurement of how far the user's mouse travels before resting.
4) The average time of these mouse motions. This is a measurement of how long the user's mousemoves consistently in between rests of 500 milliseconds or more.
5) The average speed of mouse movement. One of the primary reasons bots are used is because they are so much faster than humans at accomplishing repetitive tasks. Speed is therefore likely to be a distinguishing factor between humans and bots.
6) The variance in speed of mouse movement. Humans are not completely consistent like computers are. Therefore we would expect there to be some variance in human activity while there is little to none in computer activity.
7) The variance in acceleration of mouse movement. Supposing a bot incorporated variant speed into its activity, it may still not be able to mimic or predict the variant acceleration that humans are sure to express.

Experiment setup

To make the necessary measurements, I leveraged Courseware[1], a social network and course management website. Courseware provided an interesting variety of genuine-user data, since it has many different kinds of web environments such as a calendar, forum, FAQ, dashboard, wiki, and more. These web environments provided many varieties of mouse maps that I tested against. Courseware also has a constant stream of genuine users, so supply of genuine-user data was not a problem.

In order to gather the necessary features from Courseware users, I extended an open source project called Clickheat[2] which only records user clicks to record mouse positions and keystrokes as well. Every time a user's browser fires a mousedown, mousemove, keydown, or keyup event, javascript records a timestamp of the event, as well as the x, y coordinates, and other event

metadata where applicable (e.g., which button was being pressed at the time of the event). The user's browser, operating system, and screen resolution were also recorded, but only once per page load. In order to prevent damage to the user experience, the javascript performing the recording accumulates the data in browser memory instead of sending the data to the server as soon as possible. It does not send its current bundle of data to the server until it accumulates data for 1000 events, the user clicks one of their mouse buttons, or the user leaves the page. I also adapted the backend of Clickheat to store data in a database, rather than in files, so that the analysis of the data would be much easier. By the end of the project, Courseware users had generated over 14 million data points from which to extract and analyze features.

Visualization

In order to confirm my intuitions about the distinctness of human activity on a website, it was helpful to have a visual representation of the data early on. Figure 2 to the right depicts a map of users clicks for a particular web page on Courseware. It is immediately obvious that there are certain hotspots on the page which are clicked more often than other spots on the page. Links are the main attractions causing these clicks to be grouped as they are; however, clicks are not evenly distributed on a link, meaning that mimicking human activity is not as trivial as evenly distributing click activity across the visual space of a link or other clickable entity. I had aimed to do the same kind of visualization for mouse movements and keystroke events, but time constraints prevented me from accomplishing that goal.

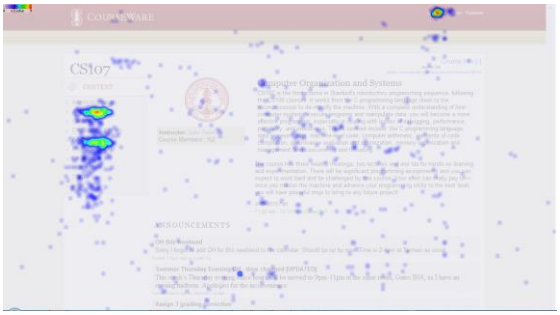

Figure 1: An example map of human click patterns on a web page

Model

I decided to use a Naïve Bayes model for the classifier. However, since I only gathered data for one of the two relevant classes, I had to use an unsupervised version of Naïve Bayes. The algorithm constructed based its classifications solely on the distribution of observed features generated by humans, and the goal became to detect anomalies in the distribution. The anomalies would be considered bot activity. Each feature in a test observation was compared to a discretized distribution of features in the training set. Based on which of the 100 buckets the measured feature fell into, the feature was assigned a probability. This was repeated for all features in the test observation to form a vector of probabilities corresponding to each of the features in the test observation. All of the values in this resulting vector were multiplied together to give an overall probability of the observation. If this overall probability was below a certain threshold, the observation would be marked as bot-generated activity; otherwise it would "pass" and be considered human activity.

Results of Classifier

In order to test my classifier, I was going to use a few black box security scanners that the CS Security Lab had access to. However, upon inspection it turned out that these scanners don't generate mouse events. This means that it is trivial to detect their activity. In another attempt to test the classifier I tried using a macro generator for the Chrome browser, but this suffered from the same kinds of problems as the scanners. It generated events that were impossible to create

through normal use by human users (e.g. click events had no x or y coordinates) and was therefore trivial to detect.

Conclusions

This research has been a promising first step in the direction of detecting bot and abnormal behaviors by using GUI features. My research demonstrated that current bots and scanners are not prepared to cope with this detection mechanism. On the one hand, this is good news because it can give those on the defense the upper hand. On the other hand, this makes the effectiveness of the classifier hard to assess. Because there is so much variance in human GUI activity, it is not clear how difficult it would be to create a bot that could beat the classifier. In sum, I have only scratched the surface of the problem, and it seems clear that more in depth research and rigorous analysis to determine the robustness of this approach will be needed before mouse mapping can become an effective tool in real-world defense.

Acknowledgements

Specials thanks go to Elie Burzstein for the initial inspiration, constant support, and helpful advice throughout this project.

[1] https://courseware.stanford.edu/
[2] http://www.labsmedia.com/clickheat/index.html