Spam Filter Empirical Study Project Proposal

Mihir Patel, Jacob Huber, and Ben Fintel

Introduction

When building a website that revolves mostly around user input to fill a database with, the main worry that comes to mind is how to control the spam information that is put onto the site. This is where the use of spam filters comes into play when they submit a form. When the user submits and it is caught by the spam filters in place on the backend, it allows the site to not post any malicious or misinformation directly on the site. This is related to our senior design project because the website we are developing almost solely relies on user generated information. Our site is a college rating site called *Campus Tier*. It is a place where users all across the U.S (and hopefully globally) are allowed to leave reviews for any aspect of the campus they attend. This can range anywhere from the dorms, to the dining halls, and even the professors and classes that are offered at their specific college. As you can imagine, all of this user generated data can get a bit hard to manually manage. A spam filter on our site will allow us to catch any profanity, ascii characters spam, and just general word spam before it can be seen by any other user in our site. There are many different kinds of spam filters available and one may find it difficult to select the right one. Doing the research to find the most time efficient and effective filters will allow us to use them effectively on our site, as well as providing the information to others who are struggling with the same issue.

Background

This project will utilize technology that is already available in our backend for our senior design project. Our backend is written in TypeScript with Node.js and it runs on an Azure serverless environment. Specifically, Azure Functions are used as REST APIs for our frontend. We will explore spam filter solutions that work with this backend. Mainly, we'll look at various NPM Node.js packages that are available. Readers who are interested in reading our project will need not need any kind of familiarity with Azure Functions, however, familiarity with Node.js is helpful. At a minimum, some knowledge about the domain of the problem, reducing spam on user-generated sites, is required.

Preliminary Results

Ben Fintel: My experience with any kind of live filtering on a website was an autocomplete function that was developed for one of my co-ops. This autocomplete was run any time a user inputted a character into the search bar that was on our site. With each keypress, the search bar would show a list of items that might be what the user is looking to search. This gives the user a quality of life experience as well as giving some users a direction to go with their search if they are not 100% confident in the terms they are searching for. This functionality was done in the back end; but instead of filtering out what the user was typing, I was using it to further

specify their search. It's not quite the same thing as a spam filter, but a lot of the techniques I used can be applied to implementing a spam filter.
Mihir Patel: I've had previous experience working with Node.js and TypeScript which will be useful for implementing different kinds of spam filters for this project. Having worked with the package manager NPM will allow us to quickly set up scripts to test various NPM packages. At my co-op with Siemens, I had some experience working natural language processing to classify words in certain categories. This knowledge will help us when exploring which spam filter methods we want to use for our empirical study.
Jacob Huber: I have worked with NLP with a few of my co-oping experiences as they were also focused on allowing influencers to use words that allow their writing to be catchy and engaging. This was all written in Java and I have experience in Typescript as well which practically all of these things use and it is what our website will be using as well.

We have also done some initial research on the kinds of spam filters that are available for Node.js and we found that there are a lot of options. The main goals of this project is to narrow down the choices.

Feasibility

Date To Complete	Action
Jan 28	Initial Proposal submitted
Feb 4	Finalized Proposal submitted
Feb 25	Milestone 1: Have all of our preliminary research complete, and have all the different kinds of spam filters we want to try to implement and test. We also want to have the skeleton set up for our app for when we want to start testing.
March 18	Milestone 2: We will perform our experiments with the chosen spam filters and document the results.
April 15	Milestone 3: Perform an analysis on our results and create graphical representations to summarize our findings. Have a presentation created to share our results with the class.

To split up the project work evenly among us, we will do our equal parts in researching the most promising spam filters we want to experiment with. Each of us will be responsible for implementing the three spam filters we individually researched (9 total). Once we have proceeded with the results and we are comfortable with it, we will each start creating graphs for the respective spam filters we worked on.

Merit

We would like a good spam filter and profanity filter for our website. There are so many options available to choose from it would be nice to know which options are good for each individual use-case. Do we want a filter that is very fast but can miss a lot of things or do we want something that won't miss anything but can take a lot of time to process? By testing out a lot of different packages and comparing what they accomplish, we can categorize many packages for others to use and figure out which will be best for them in their given situation. We will measure the efficiency by the percentage of spam caught and how fast the spam filter runs. The categories we want to use are: best percent spam caught for large text, best percent spam caught for small text, speed of spam caught for large text, speed of spam caught for small text. These results will take account of different test cases so we can cover a broad range of possible use cases. We also would like to note down the difficulty of implementing each spam filter so developers are aware of the effort required. Other readers will also be able to utilize our findings to select the best spam filter for their project.