March 2019

Building a Task Blacklist for Online Social Systems

Quyen Dinh Thuc Hoang
Worcester Polytechnic Institute

Trang Dieu Ha
Worcester Polytechnic Institute

Building a Task Blacklist for Online Social Systems

Authors: Trang Ha and Quyen Hoang

Advisor: Professor Kyumin Lee

A Major Qualifying Project Proposal

March 21, 2019

Abstract

Recently, the use of crowdsourcing platforms (e.g Amazon Mechanical Turk) has boomed because of their flexible and cost-effective nature, which benefits both the employers and workers. However, hiding inside this mutually-beneficial model are malicious tasks, which target manipulating search results, leaving fake reviews, etc. Crowdsourced manipulation reduces the quality and trustworthiness of online social media, threatening the social values and security of the cyberspace as a whole. To help mitigate this problem, we came up with a classification model which filters out malicious campaigns from a large number of campaigns crawled from several popular crowdsourcing platforms. We then present this blacklist on a website, where users can query for malicious campaigns based on the targeted site, task description or requesters' information. Parties adversely affected by malicious campaigns, such as targeted websites owners, legitimate workers, owners of the crowdsourcing platforms themselves, can use this website as a tool to identify, moderate and eliminate potential malicious campaigns from the web.

Acknowledgements

We would like to extend our thanks to all of those who made this project possible. First and foremost, we wish to thank Professor Kyumin Lee, who provided us with this wonderful opportunity and continued support throughout the course of this project, for pointing us to the best resources we needed to accomplish our research and development. We would like to thank people at ARC, especially James Kingsley for setting up the server machine and supporting us throughout. Additionally, we wish to thank Thanh Tran, PhD student at WPI for helping us in troubleshooting and other advice. Without any of these people, this project would have encountered more challenges.

Table of Contents

List of Figures

List of Tables

1. Introduction

In recent years, we have observed the exponential growth of crowdsourcing use in many companies. Crowdsourcing, as defined by CrowdSourcing Week, is "the practice of engaging a 'crowd' or group for a common goal — often innovation, problem solving, or efficiency." (1) According to Spigit's 2018 State of Crowdsourced Innovation Report, over a third of all surveyed companies have been employing different kinds of crowdsourcing software as part of their program for more than two years (2). With the rise of online crowdsourcing platforms such as Amazon's Mechanical Turk (MTurk) and Figure Eight, the practice of appealing to the mass for completion of micro-tasks has become easier than ever. People who are sourcing labor to finish their tasks can submit task descriptions to these websites, and offer to pay an amount of "reward" for each job. These people are known as "requesters". On the other hand, people who do the tasks and receive rewards are called "workers". Tasks can be as simple as reading a shopping receipt, or more complicated such as completing a 45-minute survey for a research.

Crowdsourcing platforms provide requesters with a low-cost but constantly available workforce, mitigating the need to hire fixed-time employees to perform simple tasks. Furthermore, as simple as these tasks may be, it is not always feasible to make use of automation instead of human labor. For example, to perform a classification task, initial manual labeling is needed for training datasets before machine learning models can be applied. Therefore, many companies often find themselves in need for cheap human labor to complete these micro-tasks. This helps explain the reason crowdsourcing has been gaining much favor among companies. On the other side, the ease with which workers can complete micro-tasks and earn money in a short

amount of time is the substantial reason for their interest in completing tasks on crowdsourcing platforms. While this seems to be an ideal model for both requesters and workers, crowdsourcing platforms bring in the risk of being exploited by requesters who request malicious campaigns that target at specific sites by manipulating search engines, write fake reviews or create manipulated accounts, as found by Lee et al. in his paper *Detecting Malicious Campaigns in Crowdsourcing Platforms (2016)*(3) Additionally, in one of his researches, Wang et al. has found that as much as 90% of campaigns posted on two Chinese crowdsourcing platforms are malicious (4). As Lee et al. has analyzed in his paper, malicious campaigns have several notable characteristics: shorter estimated completion time, along with higher rewards rate, which make them more appealing to workers when selecting tasks to complete (3). Also in this paper, he has proposed several models to detect and predict malicious campaigns on crowdsourcing websites, and has achieved a highest accuracy of 99.2%.

Our goal in this project is to build upon these findings from Lee et al.'s research to come up with other models to ultimately classify malicious campaigns within over 450,000 campaigns crawled from several crowdsourcing sites. We build the models using a provided 23,220 rows labeled dataset, then run the selected model on the large unlabeled dataset to make predictions on malicious campaigns. We subsequently present this blacklist on a website, where users can query for malicious campaigns based on the targeted site, task description or requesters' information. The result of this project, the blacklist website, will be a useful tool for website owners, especially in the social media area, in detecting malicious attempts targeted at their sites, as well as for workers to filter out malicious tasks and find legitimate tasks to complete.

2. Background

This section will explain common terminologies used in this paper, and give an overview about the crowdsourcing platform systems and its components, as well as who will benefit from this crowdsourcing model. We will then go on to state our purpose, the need for a solution to filter out malicious campaigns on crowdsourcing platforms, and what studies have been done recently that tackle similar problems. Finally, we will describe the dataset that we would be working with throughout this project, from generating classification models to performing classification on and retrieving the final blacklist.

2.1 Terminology

2.1.1 Crowdsourcing

"Crowdsourcing" is a term first defined by Merriam-Webster in 2006 (5), even though it has been in practice for a long period of time by a large number of companies. It is generally thought of as the practice of obtaining services or ideas from the work of the community - online community in many cases, rather than hiring the traditional long-time and committed employees. Many companies and organizations have found great success in taking advantage of crowdsourcing. Most notable examples among them include Waze, a large worldwide app that allows users to share real-time road conditions, traffic jams and, in turn, gives navigational suggestions for users based on the collected data (7)(8). Another company that succeeded in employing the crowdsourcing model is Lego, the toy corporation who creates contests for users'

designs of new product ideas, and rewards the creator of the most voted design with 1% of their net revenue. This campaign has not only yielded Lego great financial success, but expanded Lego's popularity globally, engaging youths and adults alike as well (8)(9).

Crowdsourcing has gained its popularity among companies in recent years due to its cost-effective nature, quick completion time and the ability to appeal to a large diversity of people in different locations. Crowdsourcing is especially useful for employers who have a large number of microtasks to be done: tasks that are simple and relatively quick to do, but need human inputs and thus cannot be automated. Examples of this include answering to surveys, or reading blurry receipts which are not yet auto-detectable by machines. On the other side, crowdsourcing also benefits workers who seek to earn extra cash while doing short and simple tasks, which require little to no skill or experience. This mutually beneficial model has explained the rising popularity of crowdsourcing, and with the rise of online crowdsourcing platforms such as Amazon's Mechanical Turk (MTurk) and Figure Eight, online crowdsourcing has become more ubiquitous than it has ever been.

2.1.2 Components of crowdsourcing platform

On online crowdsourcing platforms, *requesters* are "employers", companies, merchants, etc. who create *campaigns*, which consist of several smaller tasks. These tasks are typically small in scale (microtasks) and may or may not require pre-qualifications to participate. Workers, or "employees" are people who, if qualified, work towards completing tasks in one or more campaigns. Each task will have an amount of *rewards*, and if workers' performance is deemed sufficient by the requester, they can earn rewards for each task they completed.

Many requesters have abused this simple model of online crowdsourcing to create malicious campaigns. A malicious campaign, as defined by Lee et al., may include manipulating search engines, writing fake comments and reviews on online stores, and create dummy accounts on social media for further cyber-attacks (3). In other words, malicious campaigns slip subjective, manipulated opinions, and misleading information into the web. Inherently, crowdsourcing manipulation reduces the quality and trustworthiness of online social media as well as search engines, causing political biases, and ultimately threatening the social values and security of the cyberspace.

2.2 Related works

A number of researchers have noticed the rising problem of crowdsourced manipulation in the past few years and have generated valuable findings related to the statistics of malicious campaigns on the web. In 2012, Wang et al. took the first step in tackling the problem of "Sybils" (fake accounts) in social networking websites such as Facebook by developing a scalable crowdsourced Sybil detection system (4). On another note, in his 2015 paper *"Uncovering crowdsourced manipulation of online reviews"*, Fayazi et al. has shown that, popular search engines like Google, Bing, as well as social media sites like Facebook, Pinterest, or online merchants such as Amazon, all have fallen victim to crowdsourced manipulation coming from malicious campaigns on crowdsourcing platforms (6). A more relevant research, in his 2016 paper *"Detecting Malicious Campaigns in Crowdsourcing Platforms"*, Lee et al. has proposed a novel approach to define, classify and detect malicious campaigns that exist on several popular crowdsourcing platforms (3). In this paper, he first identified the characteristics

of malicious campaigns as opposed to legitimate campaigns, and then made use of these characteristics (or features) to build a model that classify malicious and legitimate campaigns with an accuracy of up to 99.2%. This research will, as approved by the authors, serve as the foundation, a starting point and a baseline for our project.

Recently, in 2018, Su et al. has looked into the problem of crowdturfing "Add to Favorites" that occurs within the realm of online shopping. According to the paper, adding to favorites is a "popular function in online shopping sites which helps users make a record of potentially interesting items for future purchases" (17). Malicious requesters exploit this by putting up tasks for workers to add their items to "favorite". In this way, their products will have higher positions in the search ranking lists which will make them more noticeable to shoppers, increasing their number of sales. The authors proposed a factor graph based model to identify this kind of malicious crowdsourcing behaviors, which proved to efficiently help detect manipulated "Add to Favorites" tasks.

Several recent papers which target the crowdturfing phenomenon can also be mentioned. For example, in *"Detecting Crowdturfing in Social Media"* (18), Wu et al. defined the notion of "crowdturfing", which is a combination of "crowdsourcing" and "astroturfing", and "aims to gain or destroy reputation of people, products and other entities through spreading biased opinions and framed information" (Wu, 2017).

3. Problem statement

3.1 Motivation

There are several characteristics of malicious campaigns that make them appealing to workers. As investigated by Lee et al., malicious campaigns have higher hourly rewards than legitimate ones. This means that workers can, therefore, make more money in the same amount of time, which naturally attracts more workers. Furthermore, malicious tasks usually have lower expected completion time, meaning they are usually both shorter and simpler to finish compared to legitimate ones. Workers can, therefore, earn money in a short amount of time and with an ease of mind doing malicious tasks. In an experiment that we conducted on Amazon's MTurk in an attempt to understand workers' experience on a crowdsourcing platform, we became aware of the large gap in the level of commitment required between legitimate and malicious campaigns. While some campaigns only ask for simple tasks such as going to some external websites, leaving a short review and do not require pre-qualifications to do, a few others require lengthy trainings for pre-qualifications (Figure 1), and the tasks are also much more complicated and time consuming (filling out a quality research survey, listing out all items in a blurry receipt). The rewards are relatively equal for both cases, which inevitably causes a tendency in workers to lean towards the shorter and simpler tasks.

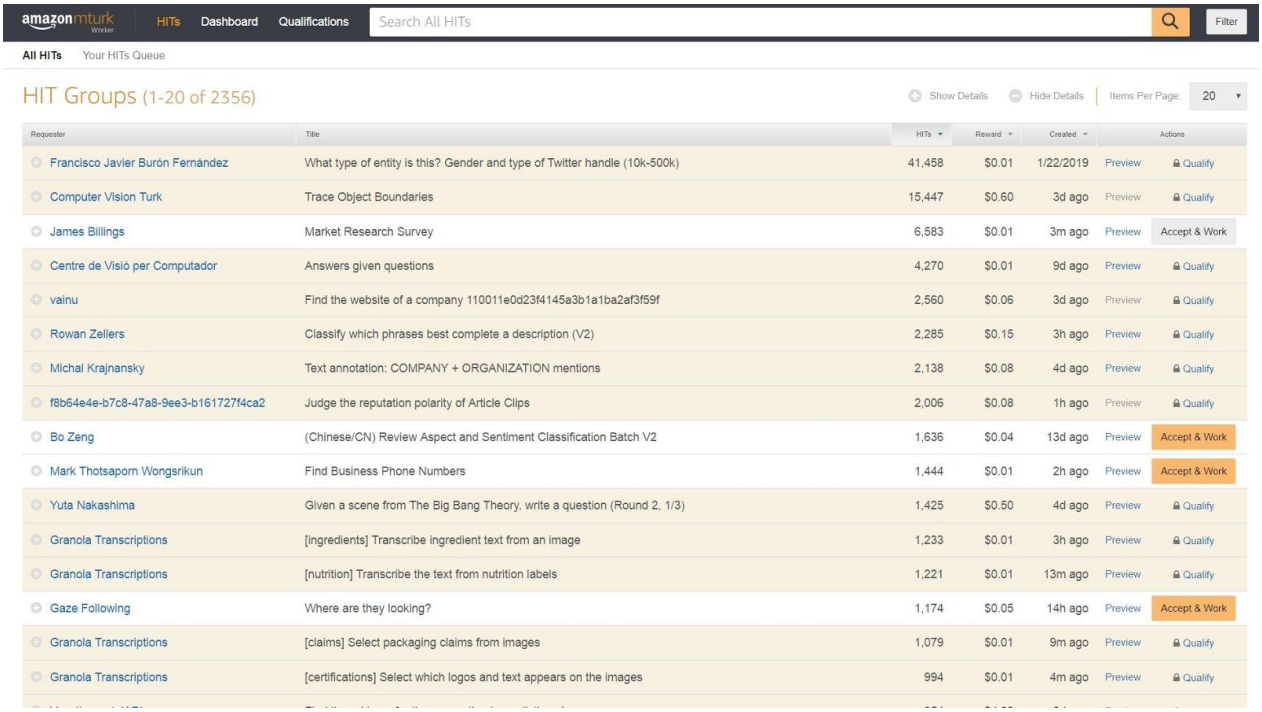

Figure 1: Large number of tasks on MTurk require pre-qualifications

Considering the widespread and possibly detrimental consequences of malicious campaigns, we found it crucial to come up with a tool to help mitigate the impact they have upon targeted websites, workers, and the web ecosystem as a whole. So far, there exists no research or product that compiles the findings of malicious campaigns into a comprehensive list where affected parties can utilize to moderate the crowdsourcing environment. This has led to our development of a malicious campaign blacklist, a mechanism that filters out proven and potential malicious campaigns from 450,000 campaigns listed on four popular crowdsourcing sites: Amazon's Mechanical Turk (MTurk), Microworkers, Rapidworkers, and Shorttask. We subsequently present this blacklist on a website, where users can query for malicious campaigns based on the targeted site, task description or requesters' information. Parties adversely affected

by malicious campaigns, such as targeted websites owners, legitimate workers, owners of the crowdsourcing platforms themselves, can therefore use this website as a tool to identify, moderate and eliminate potential malicious campaigns from the web.

3.1 Dataset

In this project, we use two datasets. The one used for baseline model development is a 23,220-row dataset of crowdsourcing campaigns, with legitimate and malicious campaigns already labeled. This dataset was crawled from MTurk, Microworkers, Rapidworkers, and Shorttask, using a crawler developed by Lee et al. in his aforementioned 2016 research. The crawler collected 23,220 campaigns, including detailed campaign descriptions within a period of three months between November 2014 and January 2015. Each campaign in the collected dataset was manually labeled as malicious or legitimate based on its description. Out of the 23,220 campaigns, 5,010 campaigns were identified as malicious and the rest were legitimate. In the same manner, the large unlabeled 450,000-row dataset was crawled from the four mentioned crowdsourcing platforms, in the 20-month period from July 2015 to February 2017 . This would be the dataset from which our blacklist for malicious campaigns were pulled out from.

4. Methodology

4.1 Goals and Objectives

The goal of this project is to build a classification model that achieves equal or higher accuracy compared with the model accuracy achieved in aforementioned Lee et al.'s paper, and with this new model, filter out the malicious campaigns within 450,000 campaigns crawled from four crowdsourcing sites. The training dataset would be a provided 23,220 rows labeled dataset, and true test would be the 450,000 entries unlabeled dataset. We subsequently present this blacklist on a website, where users can query for malicious campaigns based on the targeted site, task description or requesters' information. The result of this project, the blacklist website, will be a useful tool for website owners, especially in the social media area, in detecting malicious attempts targeted at their sites, as well as for workers to filter out malicious tasks and find legitimate tasks to complete.

4.2 Methodology

This section goes into detail on the classification models we developed, the metrics and criteria that we used to pick out the best model to run on the large unlabeled dataset. In addition, we will talk about the design process of the web application, the technology with which the product website was developed, as well as external resources that helped us deploy the final web service. Throughout the development process, we receive constant feedbacks from our advisor based on his perspective of the product.

The high level workflow of our project (as depicted in Figure 2) is as follows: First, we used the baseline dataset to develop a classification model. Next, we used this model to predict malicious campaigns in the large dataset and save these campaigns in the database. We then indexed the malicious campaigns with a search engine library and deployed the search engine through a web service. Users can use this web service to search through the malicious campaigns within our database.

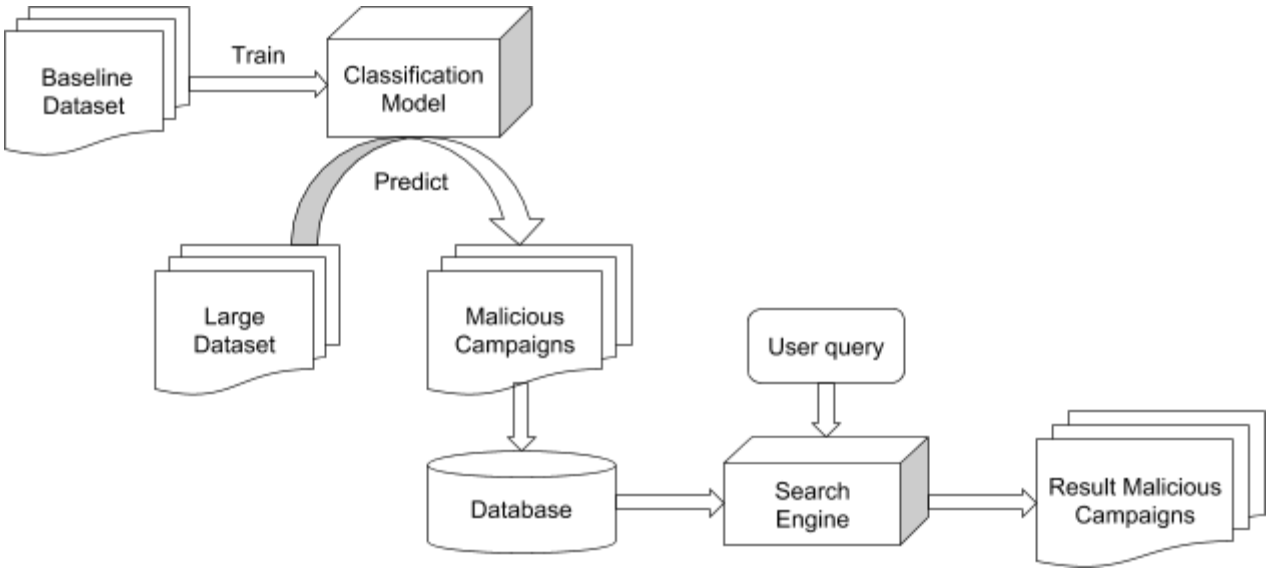

Figure 2: Project high level workflow

4.2.1 Creating classification models

4.2.1.1 Dataset

As we have mentioned in the previous sections, the 23,220-row labeled dataset would be used as the baseline dataset. For the purpose of creating classification models, this is the only dataset we were going to use. We believed the large number of instances in this dataset was

sufficient to overcome the problem of overfitting. Therefore, we made use of this dataset to make both training and testing sets. We randomly splitted this dataset into a 17,415-row training set and the remaining 5,805-row into the testing set, then saved these training and testing sets and used across all models for consistency.

4.2.1.2 Features

As Lee et al. has proposed in his aforementioned paper, the features used for building malicious campaign classifiers are "reward, number of tasks, estimated time to complete (ETC), hourly wage, number of URLs in task instruction, $\frac{Number\ of\ URLs\ in\ task\ instruction}{Number\ of\ words\ in\ task\ instruction}$, number of words in a task title, number of words in task instruction, and text features extracted from task title and task instruction." With these features, his team was able to develop a model with a high accuracy of 99.2%. Therefore, we decided to make use of these features in creating the classification models, as well as coming up with other features. However, we found out that while all other features were included in the given dataset, the correlation score between the campaign and the text features was missing. What was included in the dataset was only a list of the text features of all campaigns. As a result, we had to generate the scores ourselves. In order to do this, we went through the following steps:

1. Combine task title and task instruction, remove stopwords and apply stemming
2. Build a TF-IDF vectorizer with the sklearn library that uses the provided text features as its vocabulary and automatically extracts unigram, bigram, trigram features from the text

3. Generate TF-IDF vectors for each campaign based on its processed title and instruction text with the vectorizer

After the TF-IDF vectors were generated, we combined them with all other proposed features to get the final vectors for our model. The entire workflow of the feature extraction process can be summarized in Figure 3.

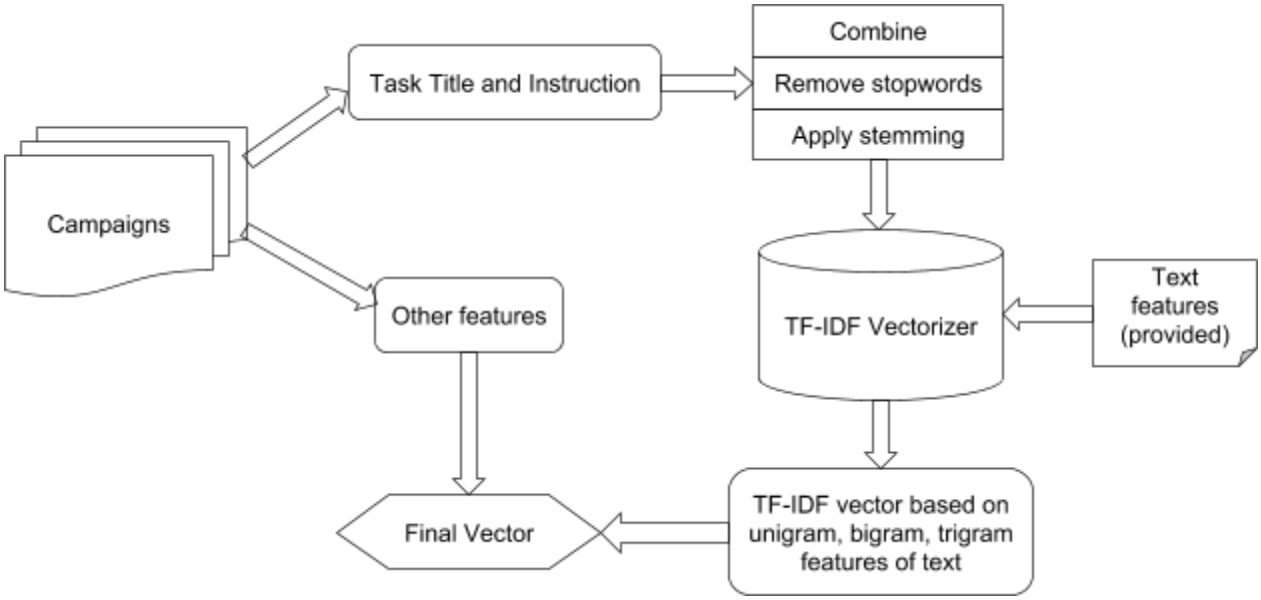

Figure 3: Campaign feature extraction workflow

4.2.1.3 Classification models

a. Features variations

Since the TF-IDF vectorizer that we built extracted unigram, bigram, and trigram features, word order within a text played a crucial part. To avoid oversight, we decided to use 3 slightly different feature combinations:

1. Without the text features to serve as baseline

2. Combine the task title with the task detail (in that order)
3. Combine the task detail with the task title (in that order)

Each type of model we utilize was run using these 3 sets of features. We then compared the performance of all models of all combinations and pick the best performing model to run against the unlabeled dataset.

b. Selected models

Choosing which model to use was an important decision, and we looked into the characteristics of several models in order to make our selections. Firstly, as Lee et. al.'s best performing model was Support Vector Machine (SVM), we selected SVM as one of our models and used its reported performance as our benchmark. Another model chosen to use on the baseline database is Decision Tree for its simplicity and easiness to interpret. We also selected Random Forest, Gradient Boosting Tree, and Xgboost to make the Decision Tree algorithm more powerful.

4.2.1.4 Final model selection

In order to pick the best performing model, we compared each model's accuracy score, false positive rate (FPR), false negative rate (FNR) and area under the receiver operating characteristic curve (AUC).

Accuracy score shows how accurate the model's prediction is. It is the ratio between the number of correct predictions and the total number of predictions. While useful, we cannot

confidently choose the best performing model on this metric alone. This is also one of the metrics used by Lee et. al.

FPR and FNR are metrics to measure errors in a model's results. FPR shows the rate of falsely predicting malicious campaigns as legitimate while FNR shows that rate of falsely predicting legitimate campaigns as malicious. These metrics would help us pick the model with the least error. These metrics are also the remaining 2 metrics of Lee et. al.

AUC shows how well a model can discriminate between positive and negative classes (21). AUC is useful for evaluating binary classification, especially with high bias data (one class is much more common than the other) (22). Since the number of malicious campaigns is significantly less than the number of legitimate campaigns in both of our dataset, this metric is extremely valuable for model evaluation.

We also used Lee et. al. best performing model as baseline, which is the SVM model with 99.2% accuracy, 0.019 FPR and 0.055 FNR.

4.2.2 Creating a web application

4.2.2.1 Frontend pages

a. Structure design

Our first task in creating the frontend UI for the web application was to come up with the overall structural design. Since the number of malicious sites to be displayed is relatively large (more than 9000 items), it would not make sense to present all of them on the main page. Instead, we thought of creating a landing page which includes a search bar for users to put in

their search criteria (it could be based on task name, targeted sites, task requesters, etc..). This page would only contain a title for the project, a search bar and a navigation bar for users to get to other parts of the website. This include links to the "Search", "Statistics", "Contact" and "About" page. The statistics page displays graphs and charts to inform users about the percentage of malicious tasks on crowdsourcing websites, the breakdown of malicious tasks rate in each platforms, and the typical pay rate for one malicious campaign, etc… "About" page gives a short description of the project, and "Contact" page provides contact information of this project's team members.

There are two ways that users can query malicious tasks. They can either fill in the search bar on the landing page and click the "See blacklist" button, or go straight to the "Search" page on the navigation bar. Either way, once the users fill in the search query (this can be the task name, requester, targeted site names, etc...), they will be taken to the search result page. The details on this page, along with initial design ideas will be further discussed in the next section.

b. Search results page

The design for this result page was also a priority for us in developing frontend UI, because this page will show the heart of our project: a searchable blacklist for malicious tasks.

We initially intended to display the blacklist in a table format, with each column as a field of the task (targeted site, task title, campaign link, etc.), and within each column there will be an option to sort the rows based on this criterion (Figure. 4). However, we found this table format extremely ineffective at showing large amount of text (Most task description and detail

are longer than the cell space). Moreover, based on our advisor's feedback, we also agreed that this format was too plain, outdated and would not attract the wanted attention of users.

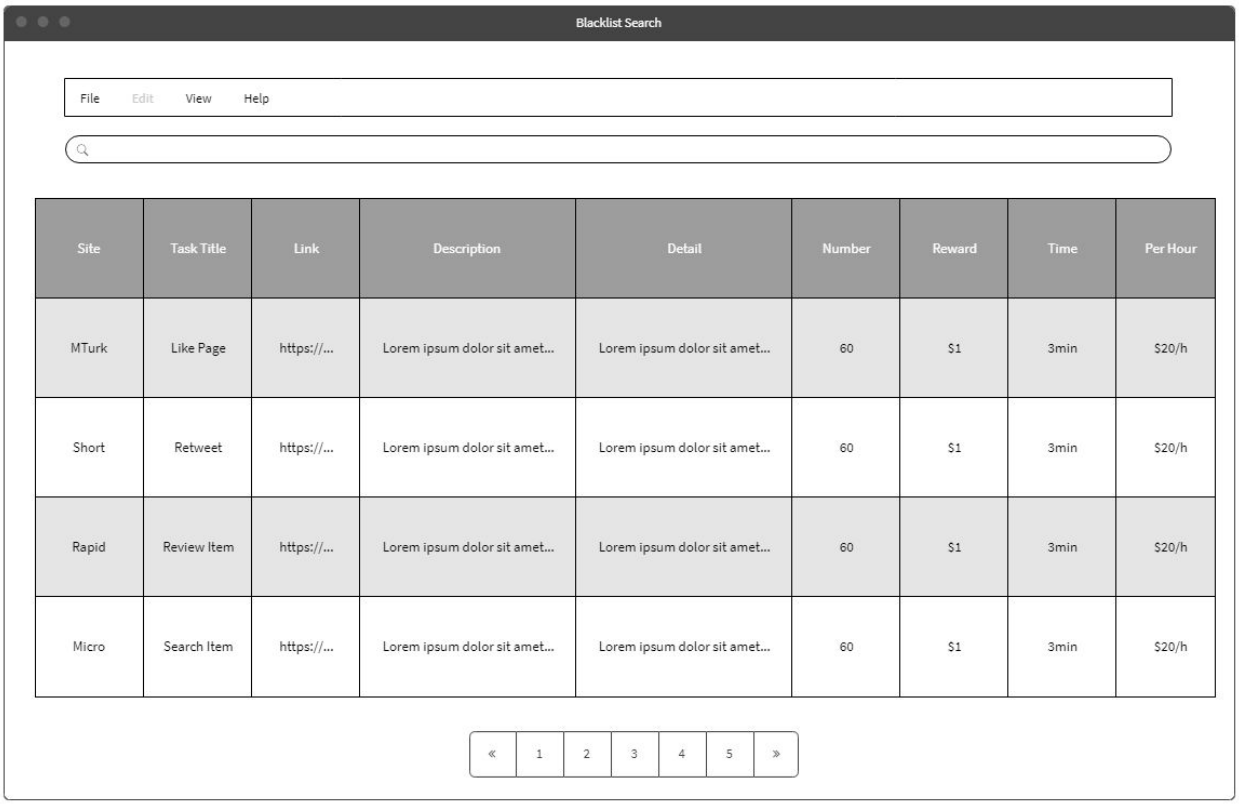

Figure 4: Initial design for search results page is in a table format

In an effort to make the page more modern and more appealing to users, we decided to study other websites. We wanted to learn from websites that were well-known, had a modern design, able display large amount of text and easy to navigate. We found that job searching sites matched all of our requirements. As a result, we decided to look at 3 popular job searching sites: Google Careers (Figure 5), Monster (Figure 6) and Glassdoor (Figure 7). Interestingly, these 3 sites had almost identical design format: a search bar at the top, an option bar that offered

capabilities such as filter and sort, a list of jobs in the small left panel, and information on the selected job in the large right panel. We decided to base our final design on this format.

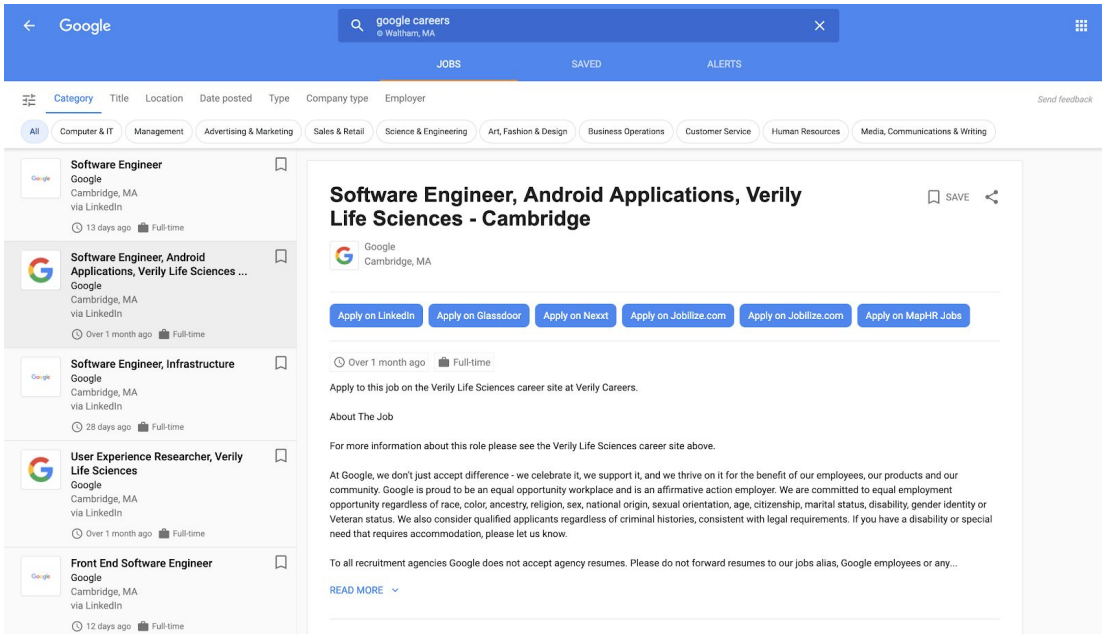

Figure 5: Google Careers web page

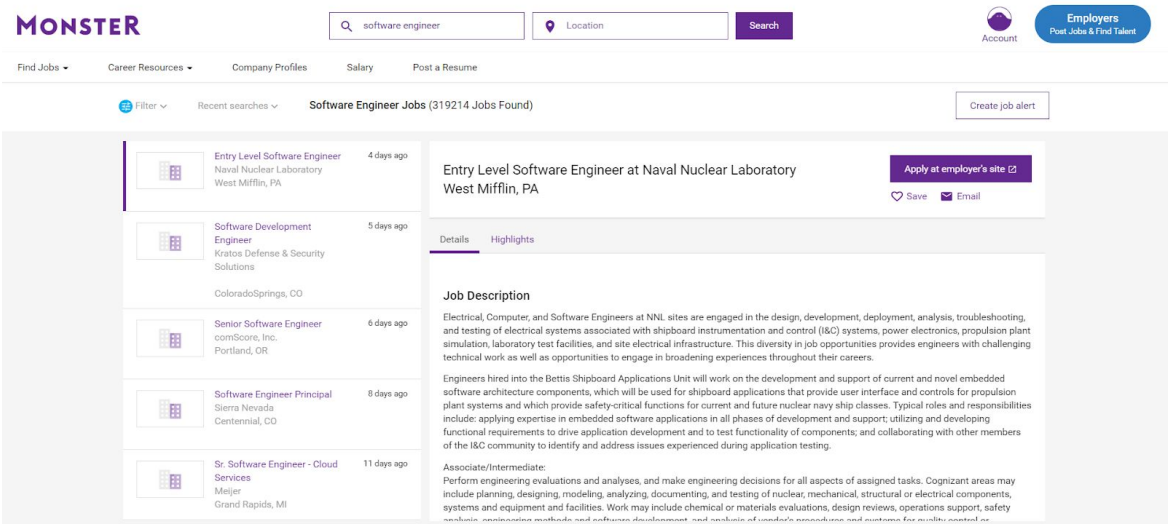

Figure 6: Monster web page

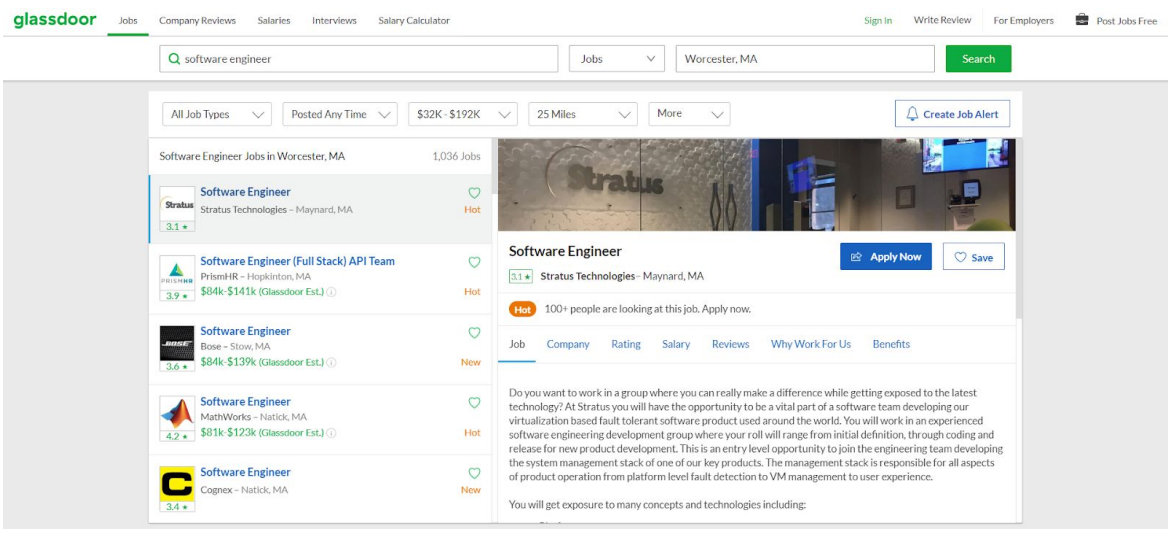

Figure 7: Glassdoor web page

Our advisor also suggested adding some statistics of the current query, such as the total number of results associated with the query or the distribution of targeted sites of all results. We decided to include both of these information in our search page final design.

The final search page design is portrayed in Figure 8. The site has a navigation bar at the top to allow users to navigate to other pages of the website. Next is an option bar that contains a filtering option button, sorting option button, a search bar, and the number of results associated with the current query. The page has 3 main panels. The left-hand panel contains a list of malicious tasks. 20 task are shown at a time in the list to avoid crowding the page and to improve page load time (by avoiding loading too many campaigns at the same time). Users can navigate the malicious task list with the page number navigator below the list. When a user chooses an item from this list, the details of the malicious task would appear on the central panel. The right-hand panel will display some statistics related to this particular task. This design template is employed by a number of websites because of its simplicity yet effectiveness.

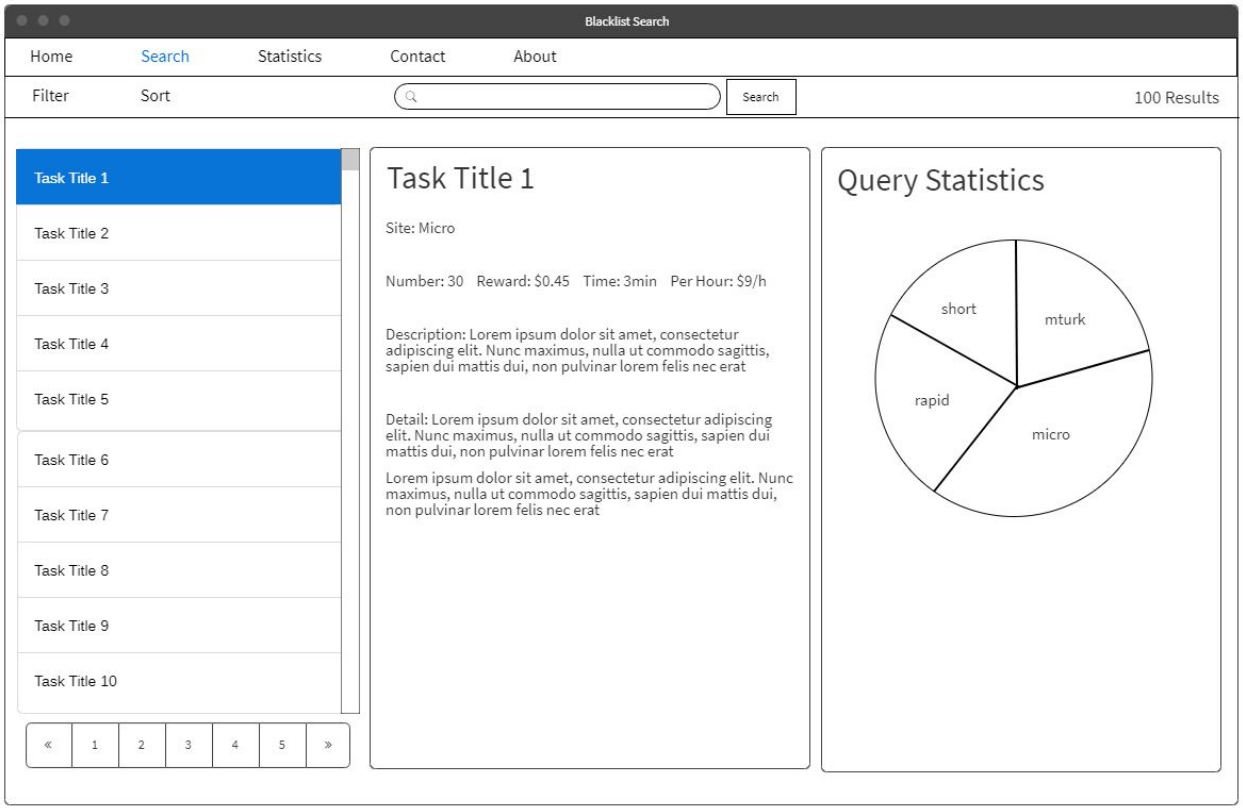

Figure 8: Final version design of search results page

4.2.2.2 Backend components

a. Web application framework

The first critical design decision to be made for our web application was the backend framework. There are a number of factors we needed to take into consideration to choose an appropriate backend framework. Firstly, since we are using Python for developing models, and the website is going to be the interface to present the results, we would prefer a framework that uses Python in order to mitigate the possibility of confusion when switching between different programming languages in the same project. Secondly, as this web service is going to fetch and

present data from the database, separation of concern became a priority, and so a well-established Model-View-Controller framework would be desired. Another important point to keep in mind is security. Like all other web applications that use private data and is going to be deployed to the public, security is one of the key criteria that the potential framework has to satisfy. Last but not least, since this is an ongoing effort to tackle harmful influences on the web ecosystem in general, and malicious campaigns on crowdsourcing sites in particular, it is our vision that this project is going to be continued and expanded in the future. Therefore, scalability is another aspect that would be nice for the framework to have.

With all the aforementioned concerns in mind, we decided to choose Django as our backend framework. Django is a popular Python-based backend web application framework which follows the model-view-template (MVT) architectural pattern (11). Some notable sites that use Django include Disqus, Instagram, and Mozilla (14). Django includes an object-relational mapper (or ORM) that arbitrates the data models (which are defined as Python classes) and the database (the "Model"). The “View” component is the Python callback function for a particular URL, and a template system for the user interface (the "Template"). Django is well-known for its emphasis on making sure that developers can avoid most common mistakes related to security. These common mistakes include SQL injection, cross-site request forgery (CSRF), clickjacking and cross-site scripting (XSS) (13). By using a strong user authentication system, Django reduces significantly security risks. Furthermore, by employing a separation-of-concerns method, Django projects are highly scalable. This means that developers can add hardwares, middlewares or caching servers at any point and still make sure the project is not corrupted.

b. Database management system

Since the data that we would use for training and testing were stored in SQL tables, we decided to use MySQL for database management. Today, MySQL is the world's largest open source relational database management system (RDBMS), indicating its maturity and availability for failure support (15). It also has the advantage of compatibility, as it runs on virtually all platforms, including Linux, Mac OS, Windows,... (16) This is important for our project in such a way that we would be using different operating systems in the development and deployment process. Details for the project deployment will be explained further in section 3.2.2.3.

c. Search engine

One of the main features of our web service is the search function. This is where users put in a search criterion and it will be used to filter out relevant malicious tasks. We used Django Haystack library as our search engine for its numerous advantages. First of all, as we had previously decided to use Django as the backend framework, we were leaning towards using a Django supported search engine. This would make the installation and usage of the search engine much easier and faster than any other searching modulars. Secondly, Django Haystack is a reusable app and relies only on its own code. This makes it work nicely with both first-party and third-party apps without requiring us to modify existing code (19). Furthermore, it has a large and active user and contributor community, which is extremely helpful for us in case we are stuck or have questions during installation and development.

Finally, Django Haystack is extremely flexible and supports some of the most popular search backends (Elasticsearch, Solr, Whoosh, etc...) The backend is also pluggable, meaning the

developer can change the search backend with minimal code changes. Initially, we opted for Elasticsearch as our search backend because of its performance, popularity, and scalability (20). However, we were not able to fully implement this backend. As a result, we switched to Whoosh, which is much simpler but less powerful.

4.2.2.3 Server machine

In the production process of the project, we needed a virtual machine to host the MySQL database, as mentioned above, and the final product - the website which would utilize this database and run machine learning models in the backend. Furthermore, with a view for this project to continue in the coming years, we preferred to set up a long-term infrastructure. After discussions with the resources team at WPI, we were given a basic centOS 7 virtual machine so that we could install various pieces as desired. We first set up the database management tool on the machine so that the web service could get access to it later. We then created a virtual environment with all the Python dependencies installed so as to avoid the scenario of confusions between different versions of packages. Inside this virtual environment, we put our Django project and manage it through a version control system - Git. Three users (members of this team and the instructor) have access to this machine. The last step in production of the web application is to publish the website by starting up the server inside this virtual machine. The URL for the website is *blacklist.wpi.edu*

5. Results

5.1 Models

As discussed in 3.2.2.4, we compared all models using Accuracy, FPR, FNR and AUC. We would also compare our models with Lee et. al. best performing model, which is the SVM model with 99.2% accuracy, 0.019 FPR and 0.055 FNR.

The first feature combination contains all proposed features except for the text features. This combination result is summarized in Table 1. All models with this feature combination perform worse than the baseline model in all metrics, with only the SVM has better FNR. In fact, this is the best FNR out of all models of all combinations.

Model	**Accuracy**	**FPR**	**FNR**	**AUC**
SVM	0.9080	0.4152	***0.0022***	0.7913
Decision Tree	0.9842	0.0372	0.0099	0.9764
Random Forest	0.9869	0.0333	0.0075	0.9796
Gradient Boosting	0.9804	0.0658	0.0068	0.9637
Xgboost	0.9805	0.0642	0.0070	0.9644

Table 1: Result of Feature Combination 1: No text feature

The second feature combination contains all proposed features, with the text features generated by combining the task title before the task detail. This combination result is summarized in Table 2. While the SVM and the Decision Tree in general perform worse than the baseline, the Random Forest, Gradient Boosting and Xgboost perform better than the baseline in 2 or more metrics.

Model	**Accuracy**	**FPR**	**FNR**	**AUC**
SVM	0.9686	0.1038	0.0112	0.9425
Decision Tree	0.9914	0.0214	0.0051	0.9868
Random Forest	0.9922	0.0174	0.0051	0.9888
Gradient Boosting	0.9921	0.0238	0.0035	0.9864
Xgboost	0.9922	0.0254	0.0029	0.9859

Table 2: Result of Feature Combination 2: Text feature taskTitle + taskDetail

The third feature combination contains all proposed features, with the text features generated by combining the task detail before the task title. This combination result is summarized in Table 3. Similar to Feature Combination 2, while the SVM and the Decision Tree in general perform worse than the baseline, the Random Forest, Gradient Boosting and Xgboost perform better than the baseline in 2 or more metrics.

Model	**Accuracy**	**FPR**	**FNR**	**AUC**
SVM	0.9686	0.1038	0.0112	0.9425
Decision Tree	0.9912	0.0206	0.0055	0.9869
Random Forest	***0.9940***	***0.0158***	0.0033	***0.9904***
Gradient Boosting	0.9920	0.0246	0.0035	0.9860
Xgboost	0.9922	0.0254	0.0029	0.9859

Table 3: Result of Feature Combination 3: Text feature taskDetail + taskFeature

Overall, the Random Forest model of Combination 3 has the best accuracy, FPR and AUC and second best for FNR (with 0.0011 difference of the best FNR). It also outperformed the baseline in all 3 metrics. Based on these comparisons, we considered this to be the best performing model and selected this as the model to predict the labels for the large dataset.

5.2 Website

As discussed in the previous section 3.2.2.1, after several design sketches, we have decided to keep an easy-to-navigate overall structure of the web page, which has a landing page as the main page, and from there users can get other pages on the navigation bar. This web

service was deployed to a CentOS 7 virtual machine, provided to us by the WPI's Academic Resources Center (ARC). The address for our website is "blacklist.wpi.edu".

For the search result page, the average query time that users have to wait before the page fully loads is about 7 seconds, which was longer than our expectation. This is partially due to the Highcharts query statistics graph being generated at runtime

Below are some snapshots of final version of the website.

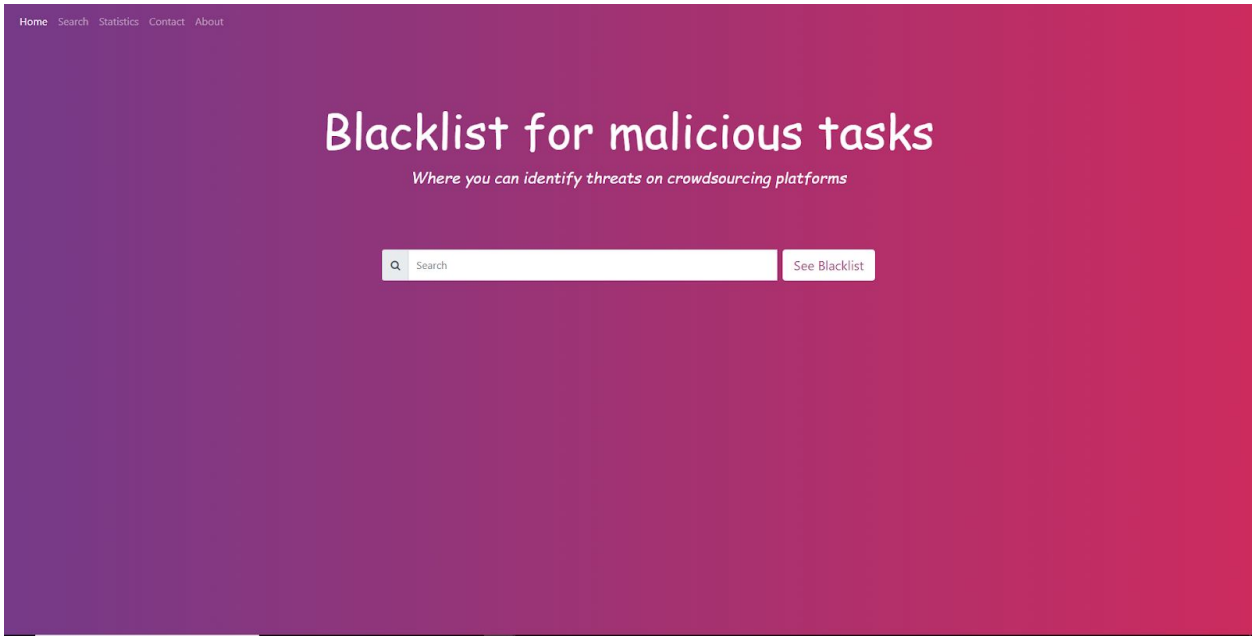

Figure 9: Landing page

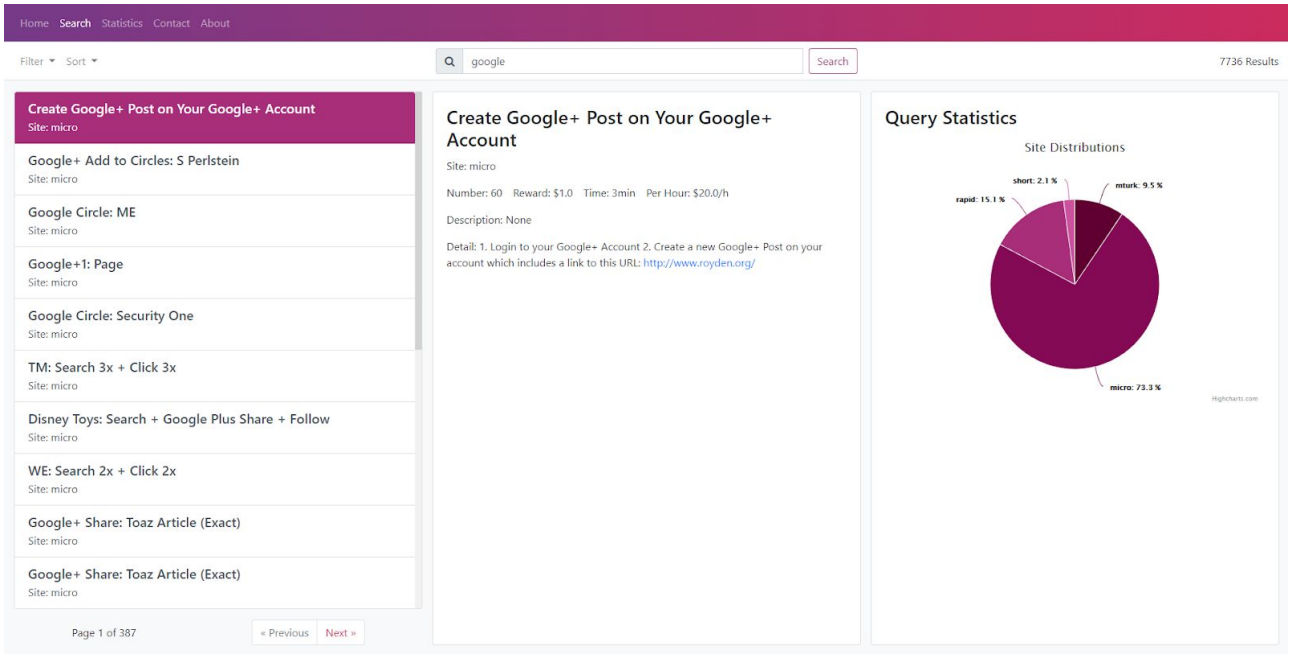

Figure 10: Search result page

Figure 11: Statistics page

6. Conclusion and Future works

6.1 Conclusion

In this project, we have developed a model which labels malicious campaigns based on a labeled 23,220-campaigns dataset. This was a Random Forest model, performing spectacularly on the baseline dataset with an accuracy of 99.4% and FPR of only 1.8%. This model was used to perform classification on the large, unlabeled nearly 450,000 campaigns database, which were crawled from several popular crowdsourcing platforms. It identified over 5000 malicious campaigns, which made our final blacklist.

We then went on to create a web service to present this blacklist, where users can query for malicious campaigns based on the targeted site, task description or requesters' information. This web service uses Django as the backend framework, which promotes separation of concerns, security and scalability. The database management system of our choice was MySQL for its compatibility with our provided datasets, ease of use and popularity. Django Haystack was the search modular used for the searching functionality for its compatibility with Django, our choice of backend framework. This web service was deployed to a server machine provided to us by the WPI's Academic Resources Center. The address for our website is "blacklist.wpi.edu".

Instructions on how to set up the server and running the website can be found in Appendix A. Likewise, all the Python packages that needs to be installed in order to run the scripts and replicate our development process is included in Appendix B.

6.2 Future works

In this project, the blacklist was generated only from the dataset provided by our advisor, which was collected in 3 months between November 2014 and January 2015 (3). In order to populate the database with more recent malicious campaigns, future researchers should develop a crawler to crawl crowdsourcing websites for new campaigns, use a classification model to predict the labels of these new campaigns, add the newly predicted malicious campaigns to the database, and index them with the site's search engine. Ideally these actions should be automatically performed at least once per day to keep the database up-to-date.

The current search engine backend is Whoosh. While it is sufficient for now, this backend is not optimal for scaling and performing more complicated tasks such as faceting. Future researchers should replace this backend with a more powerful one such as Elasticsearch or Solr.

Currently the runtime of the search page is slow (approximately 7 seconds for 1 query). This runtime can be improved by using a more powerful backend engine and running the server on a more powerful infrastructure.

References

[1] Crowdsourcing Week, 2018. *What is Crowdsourcing?* (n.d.) [online]. Retrieved from https://crowdsourcingweek.com/what-is-crowdsourcing/

[2] *Spigit's 2018 State of Crowdsourced Innovation Report* http://go.spigit.com/rs/123-ABC-801/images/2018_Spigit_State_of_Crowdsourced_Innovation.pdf

[3] Hongkyu Choi, Kyumin Lee, and Steve Webb. 2016. *Detecting malicious campaigns in crowdsourcing platforms*. In Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM '16). IEEE Press, Piscataway, NJ, USA, 197-202. https://web.cs.wpi.edu/~kmlee/pubs/Choi16ASONAM.pdf

[4] Wang, Gang & Mohanlal, Manish & Wilson, Christo & Wang, Xiao & Metzger, Miriam & Zheng, Haitao & Y. Zhao, Ben. (2012). *Social Turing Tests: Crowdsourcing Sybil Detection.* https://www.researchgate.net/publication/224973061_Social_Turing_Tests_Crowdsourcing_Sybil_Detection

[5] Merriam-webster.com. (2019). *Definition of CROWDSOURCING*. [online] Available at: https://www.merriam-webster.com/dictionary/crowdsourcing

[6] A. Fayazi, K. Lee, J. Caverlee, and A. Squicciarini, *"Uncovering crowdsourced manipulation of online reviews,"* in SIGIR, 2015. https://web.cs.wpi.edu/~kmlee/pubs/fayazi15sigir.pdf

[7] (Digit.hbs.org, 2015) *Waze – Crowdsourcing Maps and Traffic Information* (n.d.). https://digit.hbs.org/submission/waze-crowdsourcing-maps-and-traffic-information/

[8] Kearns, K. (2015). *9 Great Examples of Crowdsourcing in the Age of Empowered Consumers* - Tweak Your Biz. [online] Tweak Your Biz. Available at: https://tweakyourbiz.com/marketing/9-great-examples-crowdsourcing-age-empowered-consumers

[9] (Digit.hbs.org, 2018) *Building Together: How LEGO leverages crowdsourcing to sustain both innovation and brand love* https://digit.hbs.org/submission/building-together-how-lego-leverages-crowdsourcing-to-sustain-both-innovation-and-brand-love/

[10] Hansen, S., (2017, May 23). *Advantages and Disadvantages of Django*. Retrieved from https://hackernoon.com/advantages-and-disadvantages-of-django-499b1e20a2c5

[11] Django Documentation. (n.d.). Retrieved from https://docs.djangoproject.com/en/dev/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names

[12] Django (web framework). (2019, February 23). Retrieved from https://en.wikipedia.org/wiki/Django_(web_framework)

[13] Django Documentation. (n.d.). Retrieved from https://docs.djangoproject.com/en/2.1/topics/security/

[14] Poulton, R. (n.d.). DjangoSites Shiny websites, powered by Django. Retrieved from https://djangosites.org/

[15] MySQL (n.d.). Retrieved from https://www.mysql.com/

[16] Rouse, M., & Moore, L. (2018, July). What is MySQL? - Definition from WhatIs.com. Retrieved from https://searchoracle.techtarget.com/definition/MySQL

[17] Ning Su, Yiqun Liu, Zhao Li, Yuli Liu, Min Zhang, and Shaoping Ma. 2018. *Detecting Crowdturfing "Add to Favorites" Activities in Online Shopping*. In Proceedings of the 2018 World Wide Web Conference (WWW '18). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 1673-1682. DOI: https://doi.org/10.1145/3178876.3186079

[18] Wu, L., & Liu, H. (2018). *Detecting Crowdturfing in Social Media*. Encyclopedia of Social Network Analysis and Mining. 2nd Ed..
https://pdfs.semanticscholar.org/d6b1/5814783ca67207fbcfc5023c560e37c429b5.pdf

[19] Getting Started with Haystack. (2016). Retrieved from
https://django-haystack.readthedocs.io/en/master/tutorial.html

[20] Elasticsearch. (n.d.). Retrieved from https://www.elastic.co/products/elasticsearch

[21] Brownlee J. (2016, May 25). *Metrics To Evaluate Machine Learning Algorithms in Python.* Retrieved from
https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/

[22] Model Selection (n. d.) Retrieved from
https://frnsys.com/ai_notes/machine_learning/model_selection.html

Appendix A

Below lists the steps to start up the server and web service:

I. SET UP SERVER

1. *Open SSH connection to the server blacklist.wpi.edu, port 22*
2. *Log in with your credentials*
3. *Go to /opt/TaskBlacklistMQP_1819*
4. *Activate virtual environment*
5. *Go to blacklist/*
6. *Run sh runserver.sh*

The website should now be up and running on server

II. ACCESS THE WEBSITE

Open any browser, go to blacklist.wpi.edu:8000

Appendix B

Below lists the packages that were installed inside our virtual environment:

pip 18.1

backcall 0.1.0

bleach 3.1.0

construct 2.5.3

decorator 4.3.2

defusedxml 0.5.0

Django 2.1.5

django-connections 0.1

django-haystack 2.8.1

entrypoints 0.3

future 0.17.1

ipykernel 5.1.0

ipython 7.3.0

ipython-genutils 0.2.0

ipywidgets 7.4.2

jedi 0.13.2

Jinja2 2.10

joblib 0.13.2

jsonschema 2.6.0

jupyter 1.0.0

jupyter-client 5.2.4

jupyter-console 6.0.0

jupyter-core 4.4.0

MarkupSafe 1.1.0

mistune 0.8.4

mysql 0.0.2

mysql-connector-python-rf 2.2.2

mysqlclient 1.3.14

nbconvert 5.4.1

nbformat 4.4.0

nltk 3.4

notebook 5.7.4

numpy 1.16.1

pandas 0.24.1

pandocfilters 1.4.2

parso 0.3.4

pefile 2018.8.8

pexpect 4.6.0

pickleshare 0.7.5

prometheus-client 0.6.0

prompt-toolkit 2.0.9

ptyprocess 0.6.0

Pygments 2.3.1

PyMySQL 0.9.3

python-dateutil 2.8.0

python-dotenv 0.10.1

python-ptrace 0.9.3

pytz 2018.9

pyzmq 18.0.0

qtconsole 4.4.3

runipy 0.1.5

scikit-learn 0.20.2

scipy 1.2.1

Send2Trash 1.5.0

setuptools 40.8.0

singledispatch 3.4.0.3

six 1.12.0

sklearn 0.0

SQLAlchemy 1.2.18

terminado 0.8.1

testpath 0.4.2

tornado 5.1.1

traitlets 4.3.2

wcwidth 0.1.7

webencodings 0.5.1

wheel 0.33.1

Whoosh 2.7.4

widgetsnbextension 3.4.2