STAYFE

Stay safe, wherever you are.

Seung Woo Son, Ki Hyun Won, Avery Yip, Bilal El-Halabi

Problem Statement

It is generally understood that Berkeley is not the safest place to live in. Shootings, violence demonstrations, clown sightings are just a few among many incidents that threatened many Berkeley students living on campus. The greater problem is that there isn't a medium where crime event information is collectively visualized for people to be aware of crime that is happening around their area. Police reports are updated monthly or bi-weekly at best. Not only that, Google Maps works well for cars, but not necessarily for pedestrians, as their algorithm do not take into account pedestrians' safety.

Hence we have built a web app that comprises of two main features - 1. A real-time crime visualizer, which visualizes where crime is happening around the world from crawling and parsing news article texts from various credible news sources, and 2. A navigation tool that suggests the safest path for pedestrian desiring to walk from point A to B.

Real-time Crime Visualization

In order to build a real-time crime visualizer, we first need to crawl data from the various news media sources, such as abc7news.com/tag/crime. Luckily, we were able to find that many news sources already have crime tagged section so that we do not need to classify an article whether it is related to crime or not.

As we explored using various crawling tools such as Scrapy and BeautifulSoup, we realized that different crawlers are needed to match the differently structured web pages. For instance, some websites might have "show more" buttons to retrieve more articles and hence we looked into using Selenium to catch Javascript-based button features. Some might have infinitely scrolling bars. Others might have simple numbered pages. We successfully built different sets of crawlers for different structured pages, and made them real-time by setting the crawlers on Google Cloud and running them at scheduled times every day. We have built a temporary cache that keeps track of the most recently crawled URL so that once crawler is re-run and hits that URL, the crawling will terminate.

With the news articles successfully crawled from various sources, we need to parse and extract address (location), time and crime type (murder, rape, etc.) of the crime event. We first tokenized body_text into a list of sentences using NLTK library, after which we used keyword filtering techniques to classify a sentence describing a crime event into a crime type. Then, we used regular expression to catch and filter out all the addresses within those sentences. These addresses were then used as inputs into the Google Geocoding API, which spit out a pair of coordinates mappable onto a map.

These coordinates were then loaded onto the CARTO database. We have designed our own map on CARTO, and were able to build our final product - which visualizes more than 50,000+ crime events that

happened in Bay Area and beyond for the last 5 years, just from 6+ news sources (https://seungwooson.carto.com/builder/9de37341-04e4-47d7-89d9-f9602d27a11a.)

Safest Path Algorithm - Theory

To generate a custom safest path, we would need to acquire coordinate information of every sidewalks and crossroads in the bay area. We looked into open source database such as OpenStreetMaps; however, it did not contain fine-grained coordinate information such that it was infeasible to create a walking path. Instead of constructing a path from scratch, we thought about ways to modify obtainable shortest path. Intuitively, we would need to place coordinates away from crime zone. The idea was that our algorithm would model impact radius of each crime data point and find waypoints to circumvent crime zone.

We represented each crime data point as a Gaussian distribution with different amplitude and widths depending on types of crime.

$$\text{Weight} = \sqrt{\sum_{i=1}^{n} Ae^{-Mean\left(\left(\frac{pathpoint_i - crimepoint_i}{width}\right)^2\right)}} \quad ,where\ width\ depends\ on\ type\ of\ crime$$

For simplicity, we assigned larger width to crimes that contains keyword ‘bomb’ or ‘gun.’ Our assumption about crime data was that its impact on neighborhood would decay as time passes. To calculate the impact or amplitude, we modelled it as an exponential decay function. In math,

$$A(\text{Amplitude}) = e^{\left(-\frac{currentTime - crimeTime}{\beta}\right)} \quad ,where \beta\ is\ some\ large\ constant$$

To actually retrieve the path, we first grab the crime data between a desired source and destination point. Then, our algorithm weighes each data point as a Gaussian distribution and select points that are above fixed threshold. We added fixed threshold to reduce the number of points required for computation. By iterating through each selected points, we compute its gradient. The heavier its weight is, the steeper its gradient becomes. We repeat this process until all the points are less than the threshold or we have 4 or more points. Essentially, these points are the waypoints that are passed as a parameter in Google Direction API to compute the path.

Safest Path Algorithm - Implementation

With the safest path algorithm created, the final stage is to convert what we have into a product for our end users. We pondered on whether we wanted to create a mobile application or web application and ultimately decided on creating a web application for a minimum viable product. At an abstract level, we created a user interface which displays a map and input boxes for users to query their designated start point and destination. After the user has decided on where they want to go, our application would show the safest path in green and provide a toggleable feature showing nearby crimes in the past as markers on

the map. Intuitively, our safest path algorithm consistently avoids routes with high crime density and provide users a sense of safety when traveling to their destination.

Heading into the technicalities of our project, we used Flask, a web development micro framework in Python, to provide a way for users to interact with our map. With the data preprocessed, we cached the data for quicker access when using it to compute The user would input their location into the application and our backend would utilize the Google Maps API to geocode the start and destination. Using these geocodes, we would get the safest path as a list of waypoints. Through these waypoints, we would tweak these points to make sure that it accounted for nearby crimes density using the algorithm explained in the previous section. When connected, these tweaked waypoints represent our safest path. Finally, all we need to do now is to visualize the path through our front end built using HTML and CSS. Using a module in python named Flask-GoogleMaps, we were able to display the map, crime markers, and the route as a javascript object. With our backend and frontend in place, Stayfe is now launched onto Heroku for others users to use in the future(https://stayfe.heruokuapp.com).

Team Collaboration

We had a very strong collaborative team where everybody was constantly learning and improving upon each other's idea. Stayfe was Seungwoo's brainchild and he had a strong vision of how the product was going to be built. He worked heavily on scraping, regex, and visualization to deliver users with a consistent and flawless experience. With Ki's mathematical background, he worked on a large part of the algorithm to provide the safety that the users desired. Lastly, Avery and Bilal worked on scraping data for use with the safest path algorithm and web development to convert all of our concepts into an interactive reality. We all collectively agree this was one of the better teams we have worked in at Berkeley and look forward to working together again in the near future.

Future Improvement

We had couple of ideas for future improvement. First of all, we would like to transition our web app into a mobile app, where users can actually use our safety path navigation features on the street. Not only that, we would love to incorporate user-based reporting to make it even closer to real-time. The reason why we did not start with user-based reporting from the very beginning was because we assumed that news-based reporting was more credible. When we do incorporate user-based data, we can color-code them to be red (flashing like a beacon), and once it is reported in various news sources (we can set a threshold such as 5 news sources), then change the color of the data point to be blue and permanently mine it onto the map. Otherwise, it will disappear after some time.

Thank you IEOR 135 teaching staff for providing an opportunity to build this product - stay safe!