Optimal Stoke

Rowan Fitch
COMP 390
Occidental College
Los Angeles, United States
rfitch@oxy.edu

***Abstract*—Using a supervised machine learning model, Optimal Stoke aims to solve a problem faced by inland/urban surfers in Southern California of where the best place to go surf is. Taking into account live surf and traffic conditions, Optimal Stoke will decide for a user which beach is the best for them to surf at, and display the results in a web app. This project will combine machine learning, web scraping, and full stack development.**

I. Introduction

One of the beauties of computer science is that it can be applied to almost any sport, hobby, or pastime to make some aspect of it better. Surfing is unique in that the playing field, the ocean, varies from day-to-day, and two different beaches can yield two totally different surfing experiences. With these differences in beaches, it can be difficult and somewhat stressful for surfers to choose which beach to surf at. This project aims to make that decision for a surfer with machine learning, and present the results in a web app.

This project implements both a webscraping and linear regression algorithm. The webscraping algorithm is used to scrape data from Surfline.com, which provides daily surf reports for almost every beach on Earth. Implementing a webscraper into a web app, that can scrape data in real time will prove challenging, as the algorithm must be optimized for efficiency.

The linear regression algorithm takes a user's current location, and ranks beaches within a certain distance of the user, based on how close each beach is to the user, and how good the waves are at that beach. This algorithm will be a supervised model, meaning that it needs labeled data in order to learn. This presents a unique challenge, as a way to numerically represent how good the waves are at a certain beach is needed. Not only that, but data needs to be compiled in a large training data set with labeled outcomes.

Finally, this project requires learning application of programming skills to industry by building a web app. This will require an understanding of concepts such as HTTP requests, REST APIs, and micro service architecture.

II. Background

A. Related Work

In relation to this project, attempting to quantify ocean waves with respect to surfing has been done before, but in a different context. "Classification of Surf Breaks in Relation to Surfing Skill" quantified a wave by its height, speed, and what the authors described as "peel angle" [1]. The "peel angle" referred to the angle the face of the wave broke at, where a wave with a high "peel angle" was considered slow and unenjoyable to surf, and conversely a wave with a low "peel angle" was too fast and equally unenjoyable to surf. However, this metric was used to pair different surf breaks with surfers based on their skill level, and is not applicable to this project. The numerical metrics used to classify a wave will be explained in the methods section.

In terms of software, the Surfline mobile app is able to pair a user to surf breaks near them. However, this mobile app does not provide directions, nor does it choose which breaks would be best for the user. In this way, the two projects are different.

B. Timeline

The following is a time line for the major milestones of this project. Each milestone will be completed by the specified date

- September 15th, 2020: create the machine learning model
- September 30th, 2020: complete the backend framework of the web app with both the ranking algorithm and webscraper built in as functions
- October 21st, 2020: finalize front end
- November 21st, 2020: finish testing the finished product
- December 1st, 2020: have the finished product deployed on AWS EC2 with Nginx and Gunicorn installed

C. Required Skills

Some of the skills such as data collection, data labeling, building and training machine learning models, and optimizing algorithms are all necessary for this project to be built successfully. These skills are taught through the coursework of the Occidental College Computer Science Department. However, there are other skills necessary for this project that will require further research such as:

- Building a webscraper in Python
- Building a REST API server with Django (Python framework)
- Implementing an API endpoint that interbally implements a mcahine learning model
- Creating a webscraping microservice with Python
- Building frontend components and making API calls with React.js
- Deploying a web app on an AWS server
- Testing a web app with actual users

III. Methods

A. Data Collection

The first step in implementing any machine learning algorithm, is to prepare the data in a logical fashion. This first step is crucial, and if not well planned can lead to an ineffective model later down the road, one must avoid "high level[s] of redundant and irrelevant information that can bring down the performance of the learning algorithms" [8].

One challenge unique to this project is collecting past numerical data of ocean waves. The website noaa.org has several oceanography datasets available for the California coast. The data gathered will be:

- swell size (in feet)
- swell direction (in degrees)
- swell period (in seconds)
- wind speed (in mph)
- wind direction (in mph)
- water temperature (in Fahrenheit)

After this numerical data has been gathered beaches will be paired with mock distances (in miles) and driving time (in minutes). These mock distances and driving times can be paired with wave data, to create mock situations that a user would be presented with in choosing a beach to surf at. 1,000 of these mock situations will be created, with the choice of beach recorded. This will serve as the labeling of the data. The resultant data is an outer array consisting of 1,000 data points. Each data point is also an array, consisting of a X portion of 3 beach instances also represented as arrays, and a Y portion represented as a single array. The data will be used to train the ranking algorithm with a 70/30 split, meaning 70% of the data will be used to train the algorithm, and 30% to test it.

One issue with this approach, is that the decisions from the mock situations are prone to bias from the person choosing the beach. The person will most likely choose their favorite beach each time, so to avoid this, all beaches will be unnamed in the mock situations. This way, the decisions are being made purely off of numerical data.

B. Linear Regression Algorithm

A Linear Regression model implements **supervised learning** in which the model takes in a feature set X and a labeled output Y. The model then learns the relation between the dependent $y_i \in Y$ variables and the independent $x_{i,j} \in X$ independent variables.

In the case of Linear Regression, the relation between X and Y is assumed to be linear, so the relation can be expressed as

$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + ... + \beta_p x_{i,p} + \epsilon_i \quad (1)$$

Which can be generalized to matrix form as

$$Y = \beta X + \epsilon$$

where $Y = [y_1, y_2, ..., y_p]$

where $\beta = [\beta_1, \beta_2, ..., \beta_p]$

where $X = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & ... & x_{1,p} \\ 1 & x_{2,1} & x_{2,2} & ... & x_{2,p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n,1} & x_{n,2} & ... & x_{n,p} \end{bmatrix}$

where $\epsilon = [\epsilon_1, \epsilon_2, ..., \epsilon_p]$

In equation (1), β_j represents the partial derivative of y_i with respect to $x_{i,j}$. Also, ϵ_i represents the error value of the labeled output y_i compared with the calculated output of $\hat{y}_i$. The Linear Regression model implemented uses an **Ordinary Least Squares** approach, which means that $\epsilon_i = (y_i - \hat{y}_i)^2$.

The goal of a Linear Regression model is the minimize the ϵ value by finding the most accurate β vector to represent the relationship between Y and X. After each training iteration, the model calculates the error, ϵ_i of its predicted output $\hat{y}_i$ and the labeled output y_i. The elements of β are then adjusted accordingly in an attempt to decrease ϵ.

With respect to this project, the Linear Regression model is being used to score beaches based on their features defined in the Data Collection section. Beaches with a higher score receive a higher ranking, and therefore the exact score does not need to be accurate for each beach, rather the overall ranking via score does. For this reason, the metric for how accurate this model performs must be tailored to the problem its addressing. For a predicted output $\hat{y} = \hat{y_1}, \hat{y_2}, \hat{y_3}$ and a labeled output $y = y_1, y_2, y_3$, a successful prediction will be one in which the maximum, median, and minimum values of y are in the same indices as the maximum, median, and minimum values of $\hat{y}$. For example, a successful prediction would be

$$y = \{10, 50, 100\}$$
$$\hat{y} = \{-1, 11, 18\}$$

because the maximum value of y is at index 2, and the maximum value of $\hat{y}$ is also at index 2, etc. An unsuccessful prediction would be

$$y = \{1, 50, 100\}$$
$$\hat{y} = \{10, -10, 50\}$$

because the minimum value of y is at index 0, while the minimum value of $\hat{y}$ is at index 1. The percentage of correct predictions using the method described above will be used to determine the effectiveness of the Linear Regression model.

C. Building the Webscraper

One common road block in web scraping is the discrepancies among different web sites [2] [3] [5]. The challenge is that web scrapers are often tailored to fit a specific format of HTML code, and thus if the desired information is not in the exact format that the web scraper is searching for, then it will not grab the information. Fortunately, pertaining to this

project, a web scraper is being used on one specific website, Surfline.com. This web site offers surf reports for nearly every beach in the United States, many with live web cams as well. Because Surfline.com provides updated surf reports for almost every beach, it is the perfect candidate for web scraping. And, since every surf report web page is formatted exactly the same, this project's web scraper will bypass the issue of information presented in a different HTML format.

The most common programming language for implementing web scrapers among the sources was Python [2] [3]. Python's numerous open source libraries and packages make it an ideal programming language for a web based program, as an engineer can take advantage of existing tools instead of having to build from scratch. For this project, the webscraping algorithm will be built with the beautifulsoup Python library. This web scraper is aimed to collect the real time surf conditions of beaches in California. It will get the surf conditions of every beach in California every 3 hours, as that is how often Surfline.com updates their conditions. A successful webscraper will be one that can do this for every beach in the database of beaches.

The webscraper was built using the Python library beautifulsoup. The beautifulsoup library provides a useful tool that allows the program to target HTML ¡div¿ tags with a specific class attribute. This tool was used to target the ¡div¿ tags that contained the swell, wind, and tide data on a Surfline webpage. Conveniently, ¡div¿ tags that contained the desired information had consistent naming across pages, so the program could be easily abstracted to target any Surfline page. The only thing different was the specific URL that accessed a specific beach in Surfline's website. The beaches and their respective URLs were stored in a dictionary for clear organization.

D. The Database

Any software that stores data must do so using a database. The database of a web application acts like a system of dynamic spreadsheets that store different pieces of information about the objects involved in the web application. Due to its easy syntax and relatively good performance, SQL systems are popular databases among programmers [14] [16]. The SQL system MySQL offers a simple syntax, as well as compatibility with multiple programming languages [15]. Additionally, MySQL is supported by the Django back end framework [9], making MySQL a prime candidate for the database of this project.

E. The Backend

An important step to developing a web app is building the backend, which is how the web app handles and moves data. One useful feature common among back end development is the existence of frameworks. Back end frameworks act as a template for how the web application will interact with its database, how the web application accepts HTTP requests, and how it sends HTTP responses. There are numerous programming languages with professionally developed back end frameworks. Specifically pertaining to Python, Django is a popular back end frame work. Dong et. all describe Django as "a high-level Python web framework which allows for the rapid development and clear design" [10]. Django's main advantage is its files automatically generated upon installation that all serve a specific purpose. The files automatically generated are

- the "model" file, which is made to structure data [9]
- the "view" file which handles a request from a user and returns the response [9]

clearly, Django offers an organized framework that can streamline the back end development process. For this reason, Django will be the back end framework of choice for this project.

The specific type of backend architecture is a RESTful architecture, which means the server responses are in JSON format. the backend receives an HTTP request at a specific URL, called an API endpoint. Once the backend receives this request, it returns either an error or an HTTP response depending on the parameters of the request it received. This response is sent as a JSON object, and the data contained within the JSON object is then displayed in the frontend.
The backend's endpoints perform the following actions

- user authentication
- user registration
- returning a list of optimal surf spots
- returning a list of all surf spots

In order to authenticate a user, the backend receives a HTTP POST request that contains a username, password, and email address. The backend then encrypts the password, and checks if a user exists with the specific username, encrypted password, and email address within the database. If a user with these three parameters exists, then the backend returns a HTTP response with a unique token corresponding to the user. Otherwise, the backend returns a response that states the user does not exist.

In order to register a new user, the backend performs a function very similar to authenticating an existing user. This endpoint also receives a HTTP POST request with a username, password, and email address. The backend first checks if a user with that username or email address already exists. If a user with either of those fields exists, then the backend returns a response stating that a user with atleast one of those credentials already exists. Otherwise, the backend then encrypts the password, and creates a new row in the users database table with the sent username, password, and email address, and then returns a response containing a unique token corresponding to the new user.

The main API endpoint of the app is the one that returns a list of optimal surf spots for a user. This endpoint also receives a HTTP POST request, which contains a latitude point, longitude point, and the maximum distance a user is willing to drive. The backend then takes this information, and performs two calculations:

1) first, the backend finds the beaches that are within that maximum distance to the user, using the (latitude,

longitude) point sent by the user
2) second, for all beaches within the maximum distance the user is willing to drive, the linear regression model is ran on these beaches to determine what the top three are.

The name, and (latitude, longitude) points of the top beaches are then returned via a HTTP response to the user.

The final endpoint, which gives a list of all the spots saved within the database, accepts a HTTP GET request. The request does not need any data inside of it. This endpoint is fairly simple, the backend simply performs a SQL query that returns all the beaches saved in the database, and then returns a HTTP response containing the (latitude, longitude) points of all said beaches.

F. The Frontend

Equally important to the backend, is how a webapp displays information and interacts with a user. This is called the frontend. Similar to backend development, front end development also has numerous frameworks that streamline the process of building a user interface for a web application. Among these frameworks, the most popular frameworks operate as a modified form of the JavaScript programming language [11] [12] [13]. Kaluza, Troskot, and Vukelic compared three JavaScript frameworks

- Angular.js
- Vue.js
- React.js

and found that React.js to be the best overall framework based on its low overhead, and inclusion of useful outside packages [12]. For this reason, React.js will be the front end framework of choice for this project.

The frontend is divided into pages, where each page renders React.js components. React components have two major aspects, props and state. Props are inherited from a parent component that renders the component of concern. Props are immutable, and can be either data or a function. A components state is defined within the component, and is mutable. The state of a component can only be data as well. The main pages of the frontend are:

- the register page
- the login page
- the spot map page
- the home page

The register page displays a react component which allows a user to input a username, email, and password. This component then sends that user input to the /register API endpoint in the backend with a HTTP POST request, and awaits a response. If the component receives a valid response, then the frontend redirects the user to the home page, otherwise it displays an error message to the user explaining why they could not register with the information they input.

The log in page is similar to the register page, with a component that allows a user to input their username, email, and password. The difference is that this component sends a HTTP POST request to the /login API endpoint in the backend. If this component receives a valid response, it also redirects the user to the homepage, otherwise it displays what input was invalid.

The spot map displays a component which first checks if the user is authenticated or not. If the user is not, then it redirects back to the log in page. After the authentication check, the spot map page sends a HTTP GET request to the /all-spots endpoint in the backend, and receives a list of (latitude, longitude) points as a response. The page then sends this list as a prop to a map component. This map component sends a HTTP GET request to an external Google Maps API, and uses the list of (latitude, longitude) points to display markers on the Google Map it renders.

The home page displays several components, and also uses a concept in React.js called **conditional rendering**. Before anything is rendered, the home page checks if the user is authenticated, and like the spot map page, redirects to the login if the user is not. If the user is authenticated, the home page renders a component with instructions on how a user can get their optimal surf spots, as well as a component that gives a brief introduction to the developer of the app. Upon click of a specific button, another component is rendered which accepts an address as the users starting point, as well as a maximum distance they are willing to drive. If both of these things are filled out, the user is then allowed to click a button which sends a HTTP POST request containing the user input. This POST request is sent to the /spots endpoint in the backend. If the page receives a valid response from this API call, then an entirely new component is rendered which displays the top surf spots for the user, each on their own unique map component. Otherwise, the existing components stay rendered, and an error message is shown to the user.

G. Deploying the Web App on AWS

During the development phase, any application or service is only accessible to the machine that its being developed on. However, for that application to be accessible to the world wide web, it must be hosted by a web service. This step in the production process is where the systems architecture of an application can take different paths as well. For example, an application could have both the frontend and backend deployed into the same environment, or they can be separate.

The backend itself can be divided into smaller separate independent environments that each serve a specific purpose. This architectural style is called a **microservice** architecture and is what Optimal Stoke most closely follows. The three main environments are:

- the frontend
- a backend service for authenticating users and performing the Linear Regression model
- a backend service that hosts the webscraper, and updates the database

the main benefit of a microservice architecture is its scalability. By dividing the load of HTTP requests across multiple independent services, each service has less work to do.

The frontend is deployed using AWS Amplify, which is a web hosting service provided by Amazon Web Services. AWS Amplify will be used for the frontend because of its ability to autoscale in response to web traffic, and its easy to use backend connecting tools.

The main backend environment that will authenticate users and also run the Linear Regression model will be hosted on an AWS EC2 server. This line of servers is another service provided by Amazon Web Services, and like AWS Amplify, EC2 autoscales in response to incoming HTTP requests. Also notable is that EC2 servers run the Linux Operating System, and to deploy an environment onto an EC2 instance, knowledge of the Linux command line is required.

Aside from an EC2 server, software must also be included in order for this main backend environment to handle a large number of incoming HTTP requests. In order to do this, Gunicorn and Nginx can be used. Nginx is a service called a **proxy server** which handles incoming requests and directs them to appropriate location [18]. Gunicorn accepts requests routed from Nginx, and acts directly with the Python code that executes on the EC2 instance [17]. Gunicorn also uses multi threading to split up the number of incoming requests, allowing a web app to handle more traffic.

The final service, the webscraper environment that updates the database of beaches, will be deployed as a **cronjob**, which means it runs continually on a timed schedule. As stated earlier, this service will run every three hours, and update the database of beaches so that the surf conditions are continually up to date.

H. Testing

While testing an application during the development phase is important, there are always going to be errors and bugs that pass through the local testing and make it through to production. Often times this happens because the person developing the app understands the product so well, so they do not provide faulty inputs or click wrong buttons or click outside the intended area, etc. This is why user testing is equally important. Specifically for optimal stoke, **naive user testing** will be implemented. Naive user testing is the practice of having users that are completely unfamiliar with the application use it, and document the issues/errors they encounter. Test users will also need to be surveyed on whether or not they felt that the beach(es) recommended to them were good suggestions. This provides a unique challenge, as it means that the users testing this project must be surfers with enough experience to discern the difference between surfing at two different beaches.

Aside from user testing, the project must also go through **load testing**. Load testing is a practice where a large amount of requests are sent to a server, to test whether or not the server can handle that amount of requests all at once. To implement load testing, a command line interface (CLI) tool built with Go can be used. This CLI tool takes in a URL and a number as arguments, and then sends the specified number of requests to that specific URL.

I. Evaluation

In total, the project will be evaluated on the following criteria

- The percentage of correct predictions made by the Linear Regression model
- All API calls return successful responses in the frontend, both for external and internal APIs
- All three environments are successfully deployed onto AWS
- The complete web app is accessible to external users
- The main backend service can handle at least 1000 concurrent requests at each endpoint
- At least 50% of naive users provide positive feedback pertaining to the recommended beach for them

if these criteria can be met, this project will be considered a success.

Results section?

IV. Additional Resources

Some components of this project involve outside services, such as implementing the Google Maps API, and deploying the web app on AWS. To implement the Google Maps API, an API key is needed, which gives access to the API. This API key does not cost money to register, however the pricing is currently set up at $5 per 1000 requests. During the testing of this web app, there will most likely be several thousand requests, which could accrue a significant cost.

In terms of AWS, a free tier server can be used to host the web app, along with a free tier database and storage unit. However, an AWS account with a credit card on file is still necessary to rent a server.

V. Ethical Considerations

The two main ethical considerations regarding this project are taking a user's location, and scraping data from Surfline.com. A user's location is personal information, and getting that information without consent is an invasion of privacy. Thus, the user will be asked if their location can be accessed. Another consideration is the ethics and legality of scraping information from websites. Since the owners of the targeted web site do not explicitly consent to having their displayed information taken, nor are they compensated in any way, this raises the question of whether or not web scraping is a legal practice. However, "a federal district court decided that it is not a violation of copyright to scrape publicly available data" [3]. For scraping data off of Surfline.com, a small disclaimer can be displayed that all surf data is taken from Surfline.com, this way Surfline.com is given credit for their work. Another thing to consider is that my project could increase foot traffic to otherwise uncrowded beaches. Increased traffic can increase littering and degrade the natural beauty of the beach, for this reason, there are some beaches in state parks that I will not include in my database.

VI. Future Plans

After evaluation of this project, the web app will continue to be hosted on AWS, and hopefully acquire daily active users. Depending on the number of users the web app is able to acquire within one year post-production, it will either continue to run and be expanded upon with a mobile app using Google's Flutter code base, or it will be terminated and kept only as a repository on Github.

References

[1] Hutt, Black, Mead, "Classification of Surf Breaks in Relation to Surfing Skill", Journal of Coastal Research, SPECIAL ISSUE NO. 29. Natural and Artificial Reefs for Surfing and Coastal Protection (WINTER 2001), pp. 66-81, Coastal Education & Research Foundation, Inc.

[2] Kamran Shaukat et. all, "ANovelApproachtoDataExtractionon Hyper-linkedWebpages ", volume 1, page 1-10, November 25, 2019

[3] Geoff Boeing, Paul Waddell, "New Insights into Rental Housing Markets across the United States: Web Scraping and Analyzing Craigslist Rental Listings ", Journal of Planning Education and Research, vol 1, page 1-20, 2016

[4] "Introducing an expert system for prediction of soccer player ranking using ensemble learning",Maanijou, Mirroshandel,Neural Computing and Applications,31, page 9157-9174,2019

[5] "Deep Web crawling: a survey", Hernandez, Riviero, Ruiz, page 1-34, 2019

[6] "Machine learning for neuroimaging with scikit-learn",Abraham et. all,Frontiers in Neuroinformatics,8,2014

[7] "Mastering Machine Learning with scikit-learn",Hackeling,Packt Publishing Ltd,2017

[8] "Improvement in Hadoop performance using integrated feature extraction and machine learning algorithms",Sarumathiy, Geetha, Rajan,Springer Nature 2019, page 1-10,2019

[9] "Django documentation", https://docs.djangoproject.com/en/3.0/

[10] "ADMETlab: a platform for systematic ADMET evaluation based on a comprehensively collected ADMET database",Dong et. all,Journal of Cheminformatics,10, page1-29, 2018

[11] "COMPARISON OF FRONT-END FRAMEWORKS FOR WEB APPLICATIONS DEVELOPMENT", Kaluža, Troskot, Vukelić, Zbornik Veleučilišta u Rijeci, 6, page 261-282, 2018

[12] "From requirements to source code: a Model-Driven Engineering approach for RESTful web services", Zolotas, Diamantopoulos, Chatzidimitriou, Symeonidis, Automated Software Engineering, 24, page 791-838, 2017

[13] "Json-GUI—A module for the dynamic generation of form-based web interfaces", Galizia, Zereik, Roverelli, Danovaro, Clematis, D'Agostino, SoftwareX, 9, page 28-34, 2019

[14] "Big SQL systems: an experimental evaluation", Aluko, Sakr, Cluster Computing, 22, page 1347-1377, 2019

[15] "MySQL :: MySQL Documentation", https://dev.mysql.com/doc/

[16] "YourSQL: a high-performance database system leveraging in-storage computing", Jo, et. all,Proceedings of the VLDB Endowment,9, page 924-935,2016

[17] "Gunicorn - WSGI server", https://docs.gunicorn.org/en/stable/index.html

[18] "nginx documentation", https://nginx.org/en/docs/

Overall, good draft.
- You need to really think about what goes in background & what goes in methods.
- start putting in results! (eg, screenshots from app, etc)