Twitter Sentiment

D.S.S.

December 10, 2019

COURSE
CS 4970W - Capstone I

MEMBERS
Sara Caponi
Daniel Jaegers
Sarah Wooldridge

D.S.S. Team Information **3**
Mission Statement 3
Project Mentor 3

Team Members **4**

Introduction **5**
Problem Statement 5
Problem Resolution 5
Real World Benefits 5

Requirements **6**
User Requirements 6
Functional Requirements 6
Non-functional Requirements 7
Hardware Requirements 7

Research Introduction **8**
Server for Application 9
Server for Sentiment Algorithm 11
Sentiment Algorithm 13
Database 16

Flowcharts **18**
Search By Hashtag Flowchart 18
Search By Username Flowchart 19
Leaderboard Flowchart 20
About Us Flowchart 21

Use Cases **22**

Interaction Diagrams **23**

ERD **30**

System Architecture **32**

Sentiment Algorithm **33**
Training and Testing Data 34
Live Twitter Data using Tweepy 34
Preprocessing 34
Transformation 35

Negation Handling 35
Tokenization 35
Filtering 36
Normalization 36
Feature Extraction 37
Sentiment Classification using Support Vector Machines 38
Aggregation of Results 39
Pseudocode 40

Leaderboard Construction 42
Queries Pseudocode 42

GUI 43
Main Page 43
Leaderboard Page 50
About Us Page 54

Development Strategy 55

Development Timeline 56

Sprints 57

Work Delegation 58

Alternate Strategies 58

Possible Troubles 59

Testing Plan 59

Works Cited 60

D.S.S. Team Information

Mission Statement

The teams mission is to aggregate Twitter data into meaningful and insightful analyses and to give people an overhead view of a Twitter users' outlook.

Project Mentor

D.S.S has reached out to Dr. Jianlin (Jack) Cheng who is the Director of Bioinformatics, Data Mining, and Machine Learning Lab (BDM). He has contacted one of his PhD students in his group who works on text-related machine learning. The PhD student has not contacted the team back so far. The team will ask Dr. Cheng if he has any students who are willing to be a project mentor.

Team Members

Sara Caponi

I am a senior studying Computer Science and Mathematics. I am interested in front end development and NLP.

Daniel Jaegers

I am a senior studying Computer Science and Statistics and I am interested in NLP, Python development, and interactive visualizations.

Sarah Wooldridge

I am a senior studying Computer Science and Information Technology. I am interested in databases, user interface, and digital image processing.

Introduction

Problem Statement

The problem at hand today is there is no easy way to summarize a person's twitter account or a trending hashtag. There can be hundreds of thousands of tweets in one person's account or hashtag and it would be impossible to read all of them to get and understanding of the mindset of the user or idea. This is an issue when users are trying to get factual opinions about a topic or person. Misunderstanding of information present in tweets can lead to incorrect assumptions being made.This is a problem for society because while one Tweet may steer one's belief in one way, the aggregate may steer them in another direction.

Problem Resolution

The team plans on training/developing a machine-learning algorithm that assigns a numerical value to each Tweet. D.S.S. will then use an undetermined function to create total scores for a specific user or trend. This score will represent how positive or negative a handle or hashtag is. The information will then be displayed on an interactive webpage.

Real World Benefits

This project has multiple applications in the real world. First and foremost, this project can be used to give unbiased information to the users because the algorithm will be consistent in scoring for all searches. The information our users will receive is coming directly from the algorithm and won't be manipulated in order to sway opinions. Another application of our project would be for job recruiters to search for possible job candidates. This project would be able to give them an idea of how the person interacts with other people. For example, if a possible job candidate received a very low sentiment score this could indicate to the recruiter that they might be a negative person who would not be enjoyable to work with. Lastly, an important benefit to our project is the entertainment aspect. Users can search for themselves to see their score and compare it to their friends and/or celebrities.

Requirements

User Requirements

Functional Requirements

Identifier	Priority	Description
REQ1	5	User will be able to search for a Twitter handle.
REQ2	5	User will be able to search for a Twitter hashtag.
REQ3	5	User will be able to see a numerical value based on the attitude of the handle or hashtag.
REQ4	1	User will be able to view an About page containing information about how the scores were calculated.
REQ5	4	User will be able to see the highest and lowest scoring Tweet relating to the handle or hashtag.
REQ6	3	User will be able to view separate leaderboard for highest score for handle and hashtag.
REQ7	2	User will be able to view separate leaderboard for lowest score for handle and hashtag.
REQ8	3	User will be able to view separate leaderboard for top searched handle and hashtag.
REQ9	2	User will be able to view basic visualizations regarding the distribution of the sentiment of their tweets
REQ10	3	User will be able to select any hashtag or handle in any leaderboard and a pop-up modal will appear allowing them to view the sentiment score.

Table 1. This table includes a list of the functional system requirements that must be executed in the system to have the full system working in effect. These requirements are numbered, briefly explained and given a priority score 1-5 with 5 being the highest priority.

Non-functional Requirements

Identifier	Priority	Description
REQ11	5	The machine-learning algorithms shall provide a response in a reasonable amount of time.
REQ12	3	The system shall give appropriate feedback to user input (loading spinners, progress bars, etc.).
REQ13	2	All leaderboards will be cached on a periodic basis.

Table 2. This diagram includes a list of the nonfunctional system requirements that will be implemented in the system. These requirements are numbered, briefly explained and given a priority score 1-5 with 5 being the highest priority.

Hardware Requirements

- Server to host (ex. AWS Elastic Beanstalk)
- Server to run machine-learning algorithm (ex. AWS SageMaker)

Research Introduction

This project is a web-based sentiment analysis tool for Twitter. Users have the ability to search for any public Twitter profile or hashtag. The application collects Tweets that match the given search and then applies the Tweets to a sentiment analysis algorithm. The algorithm will compute the sentiment for each Tweet, aggregate the results, and display the results for the user. In addition, the application will store all search queries and sentiment scores in order to build a leaderboard system.

Given the broad description of this project, there are a few key decisions to make. First, there must be a defined general flow of the entire application. Then, for each step in the flow, all applicable tools must be noted. Finally, each tool in each part of the application must be compared and contrasted. From this work, justifications for each design decision can be made at ease.

Server for Application

Problem Statement

In order for the application to run, it needs a server. It is important to pick a server that will interact well with the language and that is reliable. There are three solutions for a server to host the application on and those include AWS Elastic Beanstalk, AWS EC2, and Microsoft Azure.

AWS Elastic Beanstalk	
Pros	Cons
Easy-to-use service for deploying and scaling web applications and services developed with Flask [1].	Many times it happens that there is a failure in deployment [2].
Only pay for the AWS resources needed to store and run the application [1].	AWS Elastic Beanstalk comes with a new stack update but fails to inform you what is new [2].
Automatically handles the deployment details of capacity, auto-scaling, and application health monitoring which will be very useful in the application [1].	Deployment is slow as it takes a minimum of 5 minutes and a maximum of 15 minutes just for two front-end servers [2].

AWS EC2	
Pros	Cons
7x fewer downtime hours than the next largest cloud provider [3].	If one instance fails, then the entire system fails [4].
Reliable, scalable, infrastructure on demand and can increase or decrease capacity within minutes which will be helpful as the project expands. [3].	Instance options are restricted such as specifications for the instance are decided and have to be the same for all instances on the host [4].
Lockdown security eliminates possibility of human error and tampering [3].	Once the instance is up the team must start paying for it which can add up quickly over a few months [4].

Microsoft Azure	
Pros	Cons
Scale computing and storage resources instantly which will be needed as the project grows [5].	It has no support for bugs as you have to pay for a support account which will slow down the development process of the project. [6].
Only pay for services you use which will be helpful considering the $200 budget [5].	It is expensive as you have to scale up quite a bit before the instances gets usable even for small loads [6].
Efficiently develop and manage the applications with nearly unlimited cloud computing resources [5].	The quality of services such as API's documentation and interfaces are just not extremely great [6].

Justification

The server for the application needs to be one that can handle the capacity of the application. AWS Elastic Beanstalk is an easy-to-use service for deploying and works well with services developed with Flask, which is the application's web framework. The group is unfamiliar with AWS Elastic Beanstalk, but since it claims it is easy-to-use, there will be no problem getting familiar with how to use it. Elastic Beanstalk automatically handles deployment and will scale up or down based on the application's need. With Elastic Beanstalk, you only pay for the services needed to store and run the application. This will put the team well below the $200 limit. If there is a failure in deployment, Elastic Beanstalk has application health monitoring. Elastic Beanstalk is more beneficial since it will interact well with the other AWS services for the rest of the project. Additionally, AWS Elastic Beanstalk allows the team to take over any resources powering the application.

Server for Sentiment Algorithm

Problem Statement

In order for the algorithm to run efficiently, it will need to be hosted it on a server. The main issue will be efficiency. The server needs to be able to sustain the complexity of the algorithm in order to return results in a timely manner. There are three options for this AWS SageMaker, Microsoft Azure Machine Learning Studio, and the same server as the backend is hosted on..

AWS SageMaker	
Pros	Cons
Very easy to prepare and collect training data with data labeling and pre-built notebooks included [12]	More expensive than AWS EC2 [13]
Provides a full end-to-end workflow that the team will need for this project [14]	Requires much more knowledge of the data engineering and the storage of data than Microsoft Azure [18]
Simple to set up training environment and to deploy model with 'one-click training' and 'one-click deployment' that will be very beneficial to the project because no team members have experience in this field. [12]	Also requires more knowledge of cloud operational complexities like cloud instance management — knowing the size of a cluster to choose, locations and spinning down clusters when done working [18]
Has many built in algorithms as well as supports the most popular ML frameworks [13]	

Microsoft Azure Machine Learning Studio	
Pros	Cons
Fully managed service that covers the entire machine learning workflow to build, train, and deploy machine learning models quickly. [19]	Does not break down data as comprehensively [18]
Has a drag and drop UI where the machine learning modeling process is architected on a canvas without code[19]	More difficult to create an algorithm using code, which the project is based around one main algorithm. [18]
Little code or cloud computing knowledge is needed[18]	

All in one	
Pros	Cons
Simple, having all in one will keep it easy to maintain,pay for, and get familiar with how to use.	Could run out of space on the server causing the team to have to add another one
Easiest to implement, since it is just one implementation, not multiple ones.	Efficiency could be slower than if seperated which would negatively affect the user's experience
Allows the whole application to be in one place easily accessible to the team.	

Justification

For this project the team has decided to use AWS SageMaker to create, train, and deploy the machine learning algorithm. D.S.S picked this service for many reasons one of them being they provide a full end-to-end workflow for the machine learning algorithm meaning everything can be done in one place. This service also has many built in algorithms that already exist but also supports the main machine learning frameworks. Although no one on D.S.S has experience with it before, from the research gathered it claims to be easy to use and understand.

Sentiment Algorithm

Problem Statement

The foundation of this project is the sentiment algorithm. It is important to pick an algorithm that calculates sentiment accurately and consistently. However, it is also very important to consider the efficiency of the algorithm. For example, an algorithm which is extremely accurate but takes a significant amount of time to produce results is not acceptable. The algorithm must have a balance between accuracy and efficiency. There are three solutions for the sentiment algorithm and these include Naive Bayes Classification, TextBlob, and Support Vector Machines.

Naive Bayes Classification	
Pros	Cons
High accuracy when applied to social media posts [9]	Requires an existing dataset with sentiment values. It may be difficult to find a balanced dataset of tweets with sentiment values. [9]
If implemented correctly, allows for bulk processing which increases efficiency and time to results	Difficult to implement
Ability to increase accuracy as you increase training data [9]	At its basic form, it does not allow for a neutral classification. A neutral classification is necessary because not all text is positive or negative. [9]

TextBlob	
Pros	Cons
Easiest solution to implement	Trained on movie reviews rather than tweets. This is not necessarily bad; however, there is room for improvement. [10]
Does not require an existing dataset with sentiment values [10]	Does not allow for any creativity; this solution is plug and play.
Positive, negative, and neutral classifications [10]	Does not allow for bulk processing by default. It is possible to employ some parallel processing within Python; however, other solutions provide batch processing by default. [10]
Does not require testing which would allow a much quicker development process.	

Support Vector Machine	
Pros	Cons
Most accurate solution when applied to social media posts [11]	Requires an existing dataset with sentiment values. It may be difficult to find a balanced dataset of tweets with sentiment values. [11]
If implemented correctly, allows for bulk processing which increases efficiency and time to results	Difficult to implement
Ability to increase accuracy as you increase training data [11]	
Positive, negative, and neutral classifications [11]	

Justification

When considering which sentiment algorithm fits the scope of the project best, it is important to consider accuracy, efficiency, and implementation effort. TextBlob, is the easiest solution to implement; however, it does not allow for bulk processing, and it was trained on movie reviews rather than Tweets. Therefore, TextBlob is not the best choice.

The next two options, Naive Bayes Classification and Support Vector Machines, are both standards in the NLP community. They both require existing datasets with sentiment values. However, they both have high rates of accuracy, and the accuracy can be improved as you add more training data. One problem with Naive Bayes Classification, at its most simple form, is that it does not allow for a neutral classification. It is important to have a neutral classification as not all text is clearly positive or negative. Consider, “the weather is hot.” This sentence is neither positive or negative. In fact, it lacks sentiment completely. Therefore, it is important not to force it into a positive or negative classification. Since Naive Bayes Classification does not support a neutral classification (by default), it is not the best option for this project.

Support Vector Machines are the best fit for this project. SVMs provide the highest accuracy, the ability to support bulk processing, and a neutral classification.

Database

Problem Statement

In order to rank the highest and lowest scored hashtags and users that have been searched a database system will be used to keep track of this information. To do this the team needs a fairly simple system but want it to be reliable and easy to use. The databases being discussed for possible use are MongoDB, MySQL, and AWS RDS.

MongoDB	
Pros	Cons
Faster query speed because information is held in JSON-like documents allowing results to be returned faster. [20]	Data size is typically higher due to document having key names stored in it. [17]
Very flexible because documents are able to have their own unique structure and new fields can be added at any time and contain any type of value. This is helpful when actually creating the project. [15]	Less flexibility when querying the database because does not have joins but this is not a huge dilemma for this project [15]
Free service and when hosted on an EC2, is much cheaper than AWS RDS. Cost will be below $200 budget. [20]	Limited data size, document size can not exceed 16M. This could be an issue in the future. [17]

MySQL	
Pros	Cons
Offers complete control and flexibility over the database [15]	Does not manage backups automatically compared to AWS RDS which does [16]
Free service and when hosted on an EC2 is much cheaper than AWS RDS. Cost will be below the $200 budget.[15]	More difficult to scale than AWS RDS[16]

Easy to use and team members have experience using it	Less flexibility than MongoDB, have to have a defined schema [15]

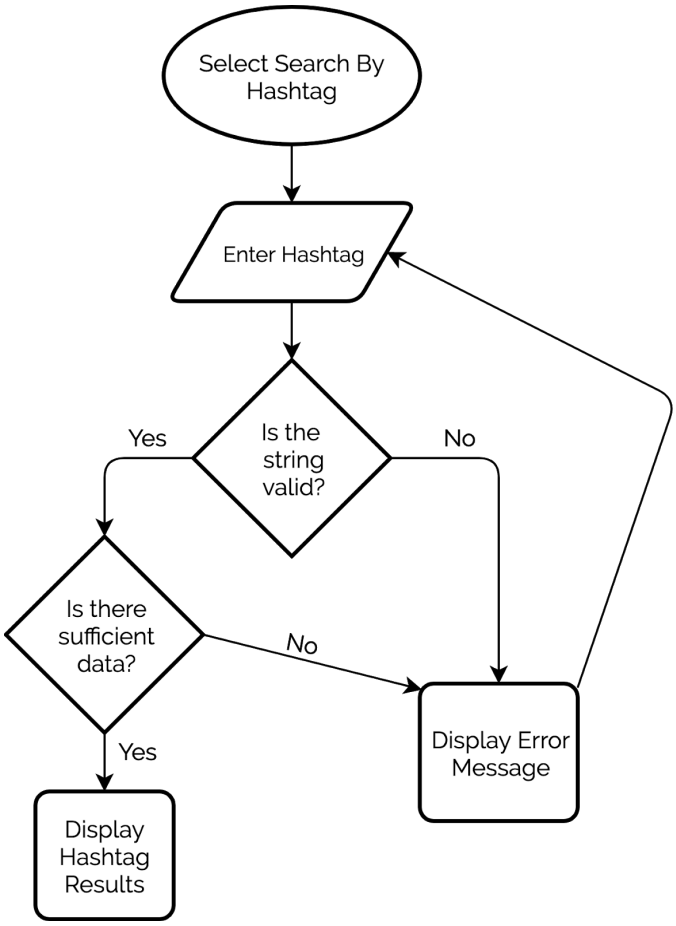
AWS RDS	
Pros	Cons
Easy to administer and to access the capabilities of a relational database in minutes [7].	AWS RDS is much more expensive than MySQL+EC2 instance [8].
Available, durable, and reliability for critical production with automated backups and host replacement. [7].	Provides many more features than the project requires,.
Scale the database's compute and storage resources with a few clicks, often with no downtime [7].	

Justification

The team decided that to use MongoDB as the database system for this project. The main reason the team went with this system is that some of the group members have experience using it and the JSON structure of the documents will be useful for storing the small amount of information needed for each search. The flexible nature of this system will be useful when deciding what information will be displayed on the leaderboards. The free monetary cost associated with MongoDB is also another reason D.S.S. decided to use it. The fast query speed that is associated with MongoDB as well as the ability to cache the results means the users will have the best experience with the fastest results.

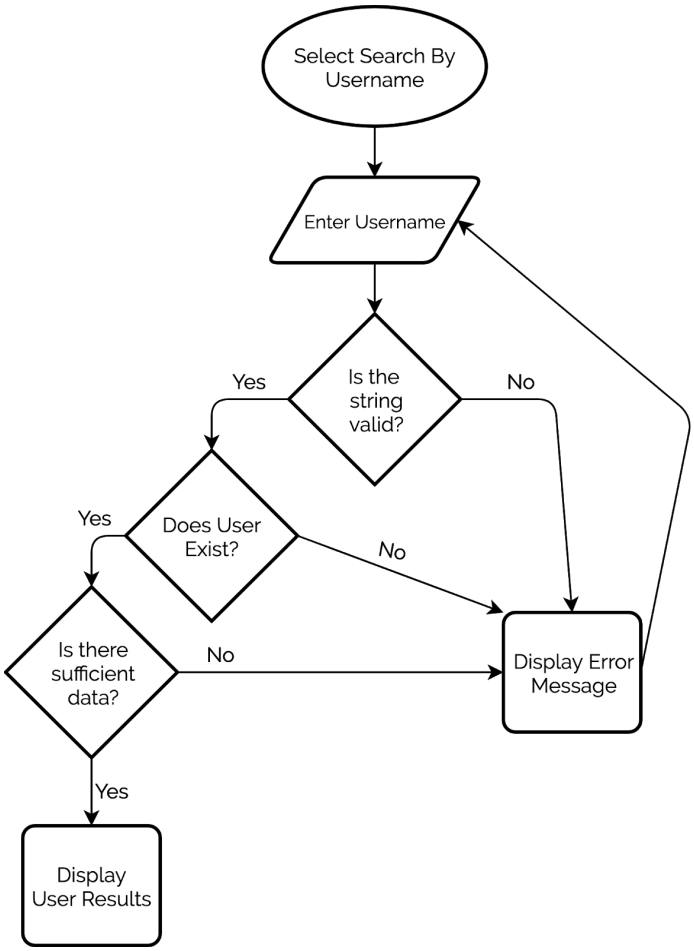
Flowcharts

Search By Hashtag Flowchart



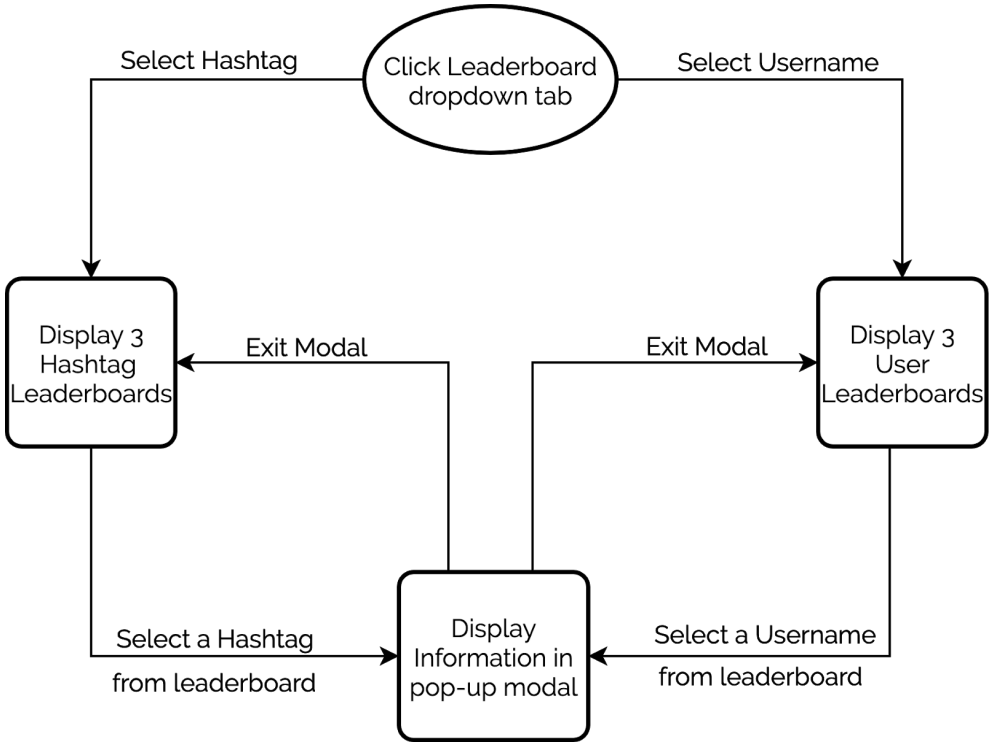
Description: From a high level perspective, the above flow chart demonstrates the steps the system takes when a user searches by hashtag. It begins when the user enters a string into the search bar and selects search by hashtag. The system will then verify that the entered string is valid. A valid hashtag is one that is under 140 characters and has no spaces or invalid characters. If the entered string is invalid, an error message will be displayed and the user will be prompted to correct their input. If it is valid, the system will then check if there is sufficient data associated with this hashtag. Sufficient data means that there is at least one tweet containing this hashtag. If that is not the case, an error message will once again be displayed and the user will be prompted to enter a new string. If data exists, the system will display this data to the user.

Search By Username Flowchart



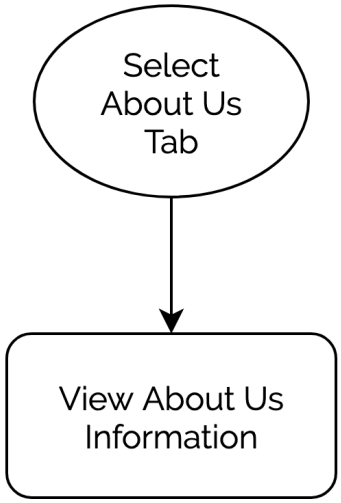
Description: The flowchart above shows the high level decisions that are made when a user searches for a Twitter user by their username. They begin by entering a string into the search field. The system then has to determine if the entered string is valid or not. A valid string in this situation is one that follows Twitters username creation guidelines of having no special characters or spaces and is under 15 characters. If the string is not valid, an error message will be displayed to the user and they will have to enter a new username. If the string is valid, the system will then check if the user exists. If the Twitter user does not exist, an error message will be displayed and the user will be redirected to enter a new string. If the user does exist, the next step will be checking if there is sufficient data related to this username. This is checking that the Twitter users profile is public, and therefore able to access their tweets. If the account is private, an error message will once again be displayed and the user will be prompted to enter a new search. If the user is public, meaning there was sufficient data, the system will display the search results.

Leaderboard Flowchart



Description: From a high level perspective, the above flow chart demonstrates the steps the system takes when a user views the leaderboard page. This flow begins when the user selects the leaderboard dropdown tab located in the navigation bar at the top of the screen. After selecting the drop down tab, they will have two options to choose from. They will be able to view leaderboards of searched usernames or hashtag. If the user selects the hashtag leaderboard, they will be directed to a page with three leaderboards holding the most negative hashtags, the most positive, and the most frequently searched. From here the user will be able to select a hashtag to view more information regarding this hashtag. Once they do that, a modal will pop up displaying this information. The user can then exit the modal which will return them to the hashtag leaderboard page. The same flow happens for the username leaderboards. The user selects the username leaderboard from the drop down leaderboard tab. They then can view the three leaderboards associated with usernames which is most positive, most negative and most frequently searched. They can also select a username from any of the three leaderboards to see more information. Once they do this, the modal will pop-up displaying the information. They can then exit the modal and return to the username leaderboards.

About Us Flowchart

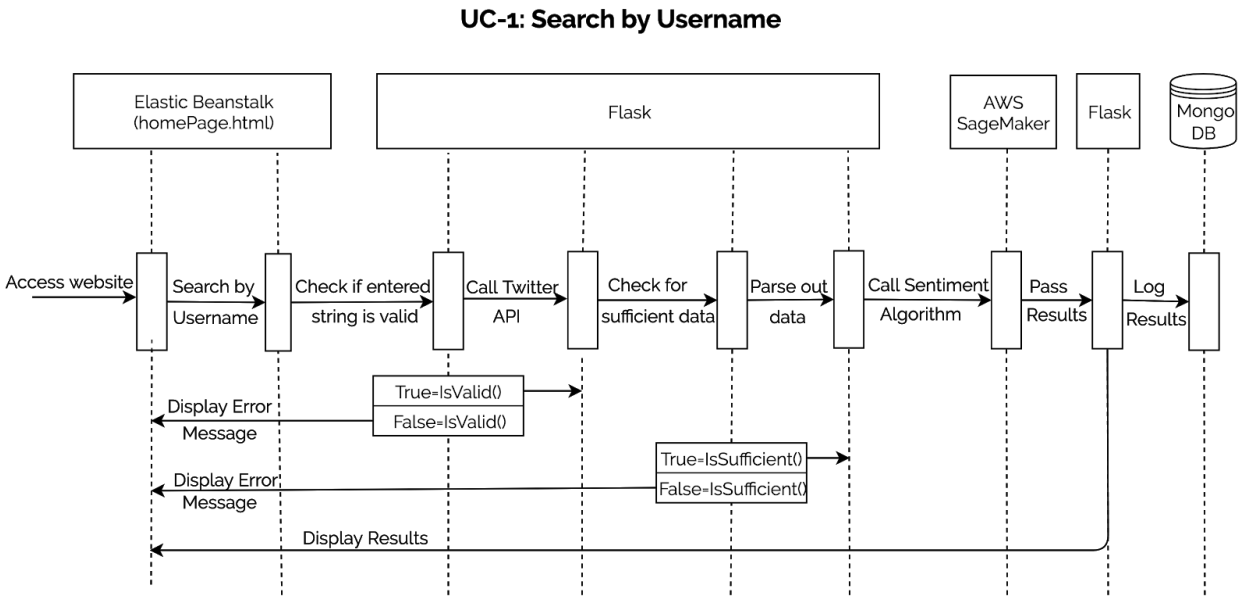


Description: The flow chart above demonstrates how a user would view the About Us page on the website. They will begin by selecting the About Us tab in the navigation bar at the top of the screen. They will then be redirected to an About Us page where they can view information about D.S.S.'s capstone project.

Use Cases

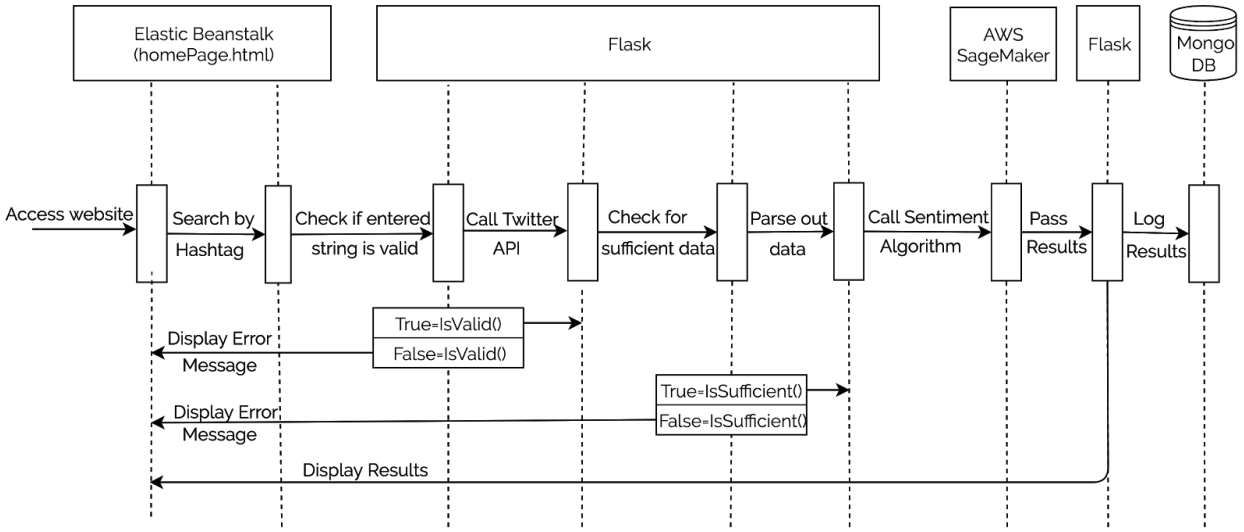
Use Case	Description
UC-1: Search by Username	The user will be able to search for a Twitter username. If the searched username is valid the user will be able to view the sentiment results.
UC-2: Search by Hashtag	The user will be able to search for a hashtag on Twitter. If there is valid data the user will be able to view the sentiment results.
UC-3: View Username Leaderboards	The user will be able to view a leaderboard page displaying three leaderboards based on the most negative, most positive, and frequently searched Twitter usernames.
UC-4: Display Username Information in Modal	The user will be able to select a username in one of the leaderboards and view the sentiment results in a modal.
UC-5: View Hashtag Leaderboards	The user will be able to view a leaderboard page displaying three leaderboards based on the most negative, most positive, and frequently searched Twitter hashtags.
UC-6: Display Hashtag Information in Modal	The user will be able to select a hashtag in one of the leaderboard and view the sentiment results in a modal.
UC-7: View About Us	The user will be able to view a separate page that describes the inner workings of the sentiment analysis and D.S.S.

Interaction Diagrams



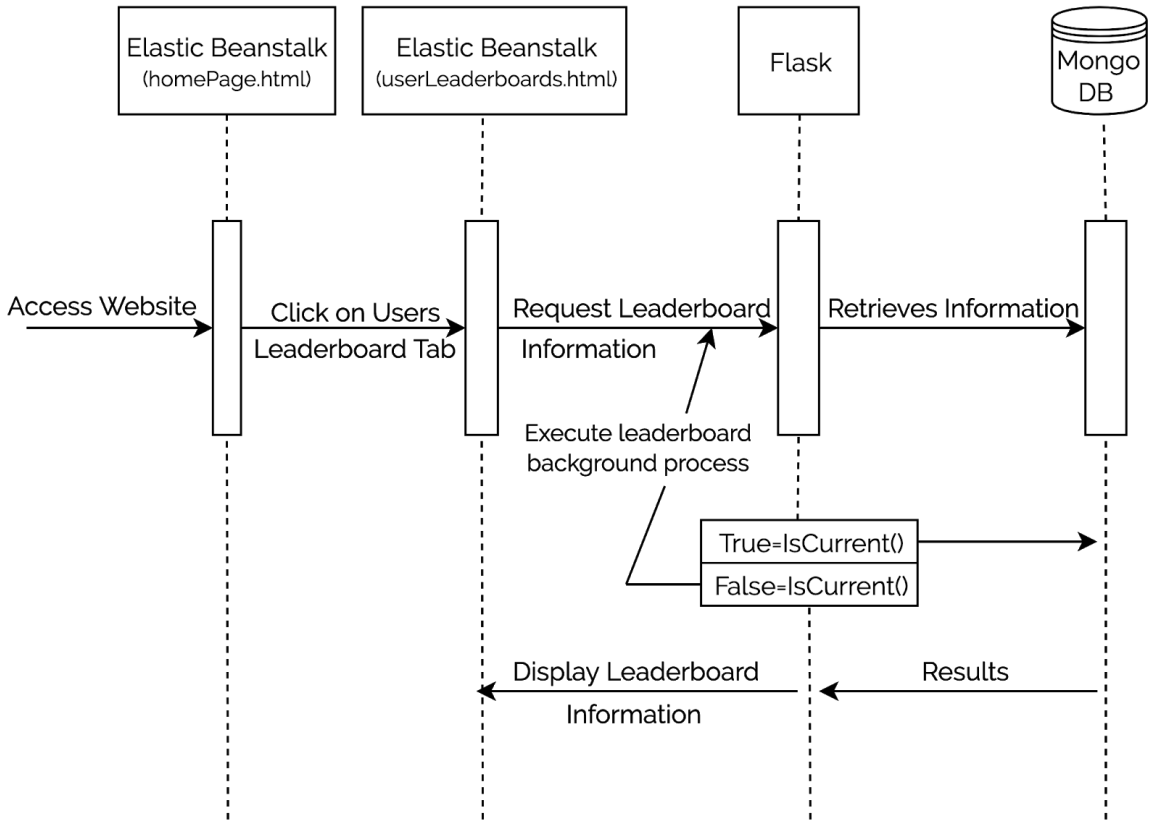
Description: The diagram above illustrates how parts of the system will interact with each other when implementing the first use case, search by username. To begin, a user will access the website through a webpage that will be deployed on AWS Elastic Beanstalk. From there, they will select that they want to search based on username, and enter a string into the search box. Then the backend, which is Flask, will see if the entered string is valid. This function, isValid() will check to make sure there are no illegal characters in the input. If this function fails, it will return an error message to the user and they will be prompted to enter another string. If isValid() returns true, Flask will then call the Twitter API. Another function will then check if the data returned from the Twitter API is sufficient enough to run the sentiment algorithm on. This function, isSufficient(), makes sure that the username entered is a valid Twitter user and that the profile is public to ensure access to the tweets. If isSufficient() returns false, an error message will be displayed to the user prompting them to enter a different search query. After the system has determined there is data to run the algorithm on, it will remove unneeded data that is returned from the Twitter API. For example, the location of the tweet or the time of tweet is not needed in the algorithm, so it will be discarded in this step. From there Flask will call the sentiment algorithm that will be held in AWS Sagemaker. Once the algorithm run is complete, it will pass the results back to Flask which will then log the results into MongoDB and also display the results back to the user on the homepage.html frontend.

UC-2: Search by Hashtag



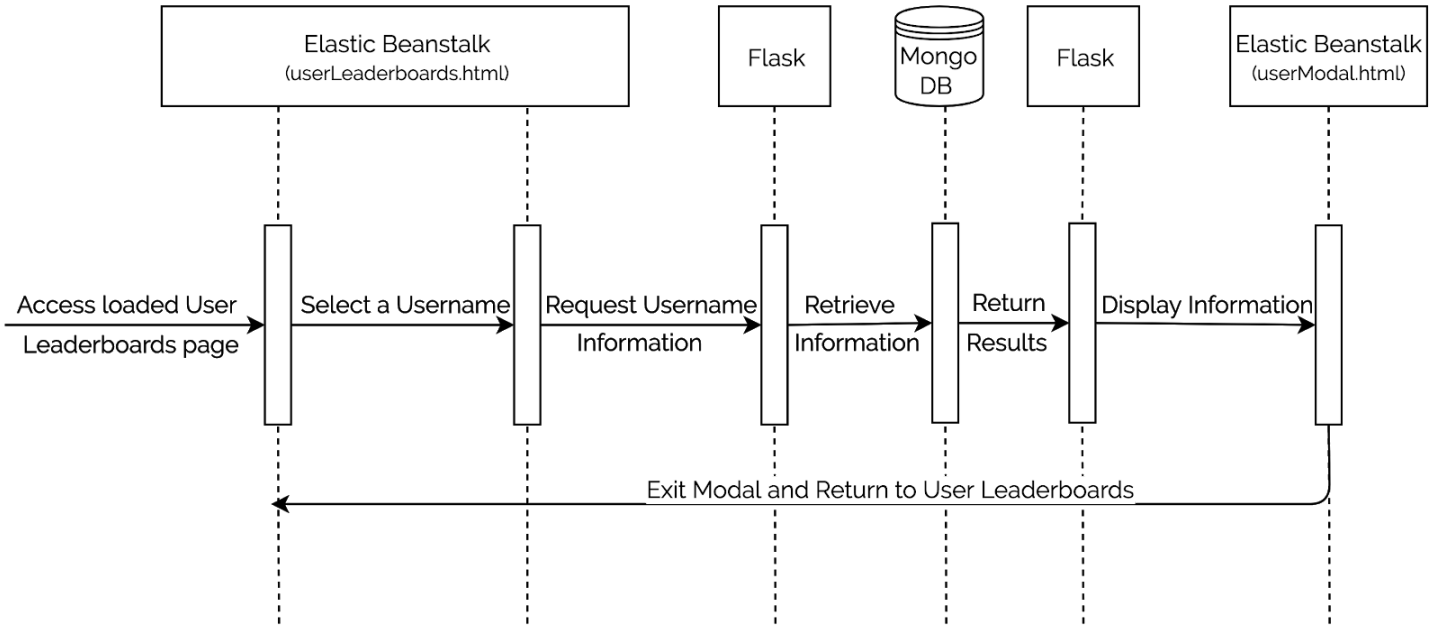
Description: In this diagram, the use case shown is UC-2, search by hashtag. Here the user starts at the website homepage. They then select search by hashtag and enter in the hashtag into the search bar. Once they press search, the string is sent to the backend, which is Flask, where a function called isValid() checks to see if the input is a valid hashtag. This function is checking the string to make sure there are no special characters or spaces inside the hashtag. If isValid() returns false, an error message is displayed and the user is prompted to enter a new string. If isValid() returns true, Flask then calls the Twitter API. The data retrieved after calling the API is then sent to a function to determine if there is enough data to run the sentiment algorithm. In this use case, isSufficient() is checking that there are tweets under the specific searched hashtag. If there are no Tweets available, the function will return false and an error message will be displayed back to the user on the homepage where they will be prompted to enter a different hashtag. If there are tweets under the searched hashtag, isSufficient() will return true and then parse out the unneeded information. After the data is cleaned, Flask will then call the sentiment algorithm on AWS SageMaker. This algorithm will run and send the results back to Flask. After receiving the sentiment algorithm results, the backend will log the results in the MongoDB as well as display the results back to the user on the homePage.html frontend.

UC-3: View Username Leaderboards



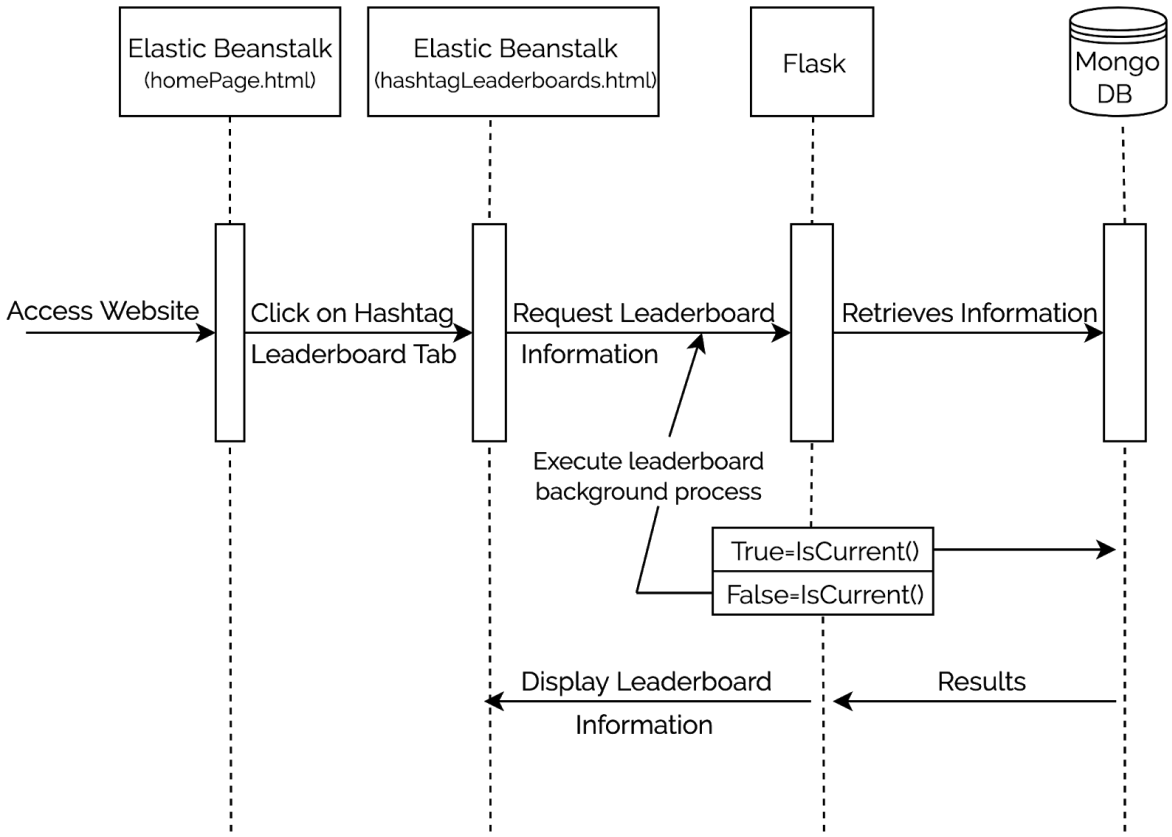
Description: UC-3 Viewing Username Leaderboards begins when the user accesses the website at the homePage.html. From there, the user will navigate to the userLeaderboards.html page by clicking on the Users Leaderboard Tab in the navigation bar at the top of the screen. This will request the leaderboard information from the backend Flask server. During this step the backend will run the function isCurrent() to determine if the leaderboard database information is up to date. If the information has been calculated in the past 12 hours isCurrent() will return true and will move to the next step. If it has not been calculated in the decided amount of time, it will execute the leaderboard background process to calculate the leaderboard standings for previous searched Usernames. After this process runs, the isCurrent() function will then return true. From here, Flask will retrieve the leaderboard information from MongoDB. The results will be sent back to Flask which will then display the information back to the user on the userLeaderboard.html page.

UC-4: Display Username Information in Modal



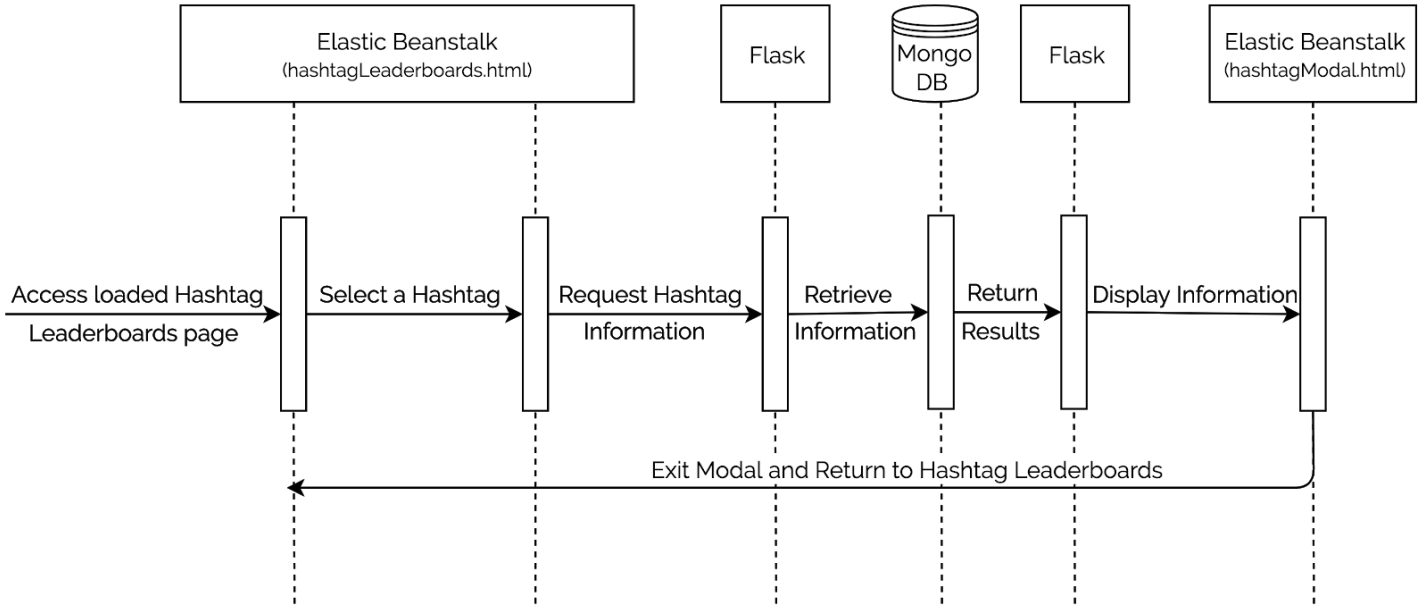
Description: This use case, displaying username information in modal, starts when the user accesses the already loaded user leaderboards page. From this page, userLeaderboards.html, hosted on an Elastic Beanstalk server, the user can select a username that is listed on any of the three leaderboards. This selection will trigger a request to be sent to the backend to retrieve the information associated with this user. The backend will then ask MongoDB for the data. These results will be returned to the backend where Flask will then display this information in a popup modal, userModal.html, in front of the user leaderboards. After the user is done observing the data they can exit the modal which will return them to the loaded user leaderboards page where they can then select a different username or go to another page on the website.

UC-5: View Hashtag Leaderboards



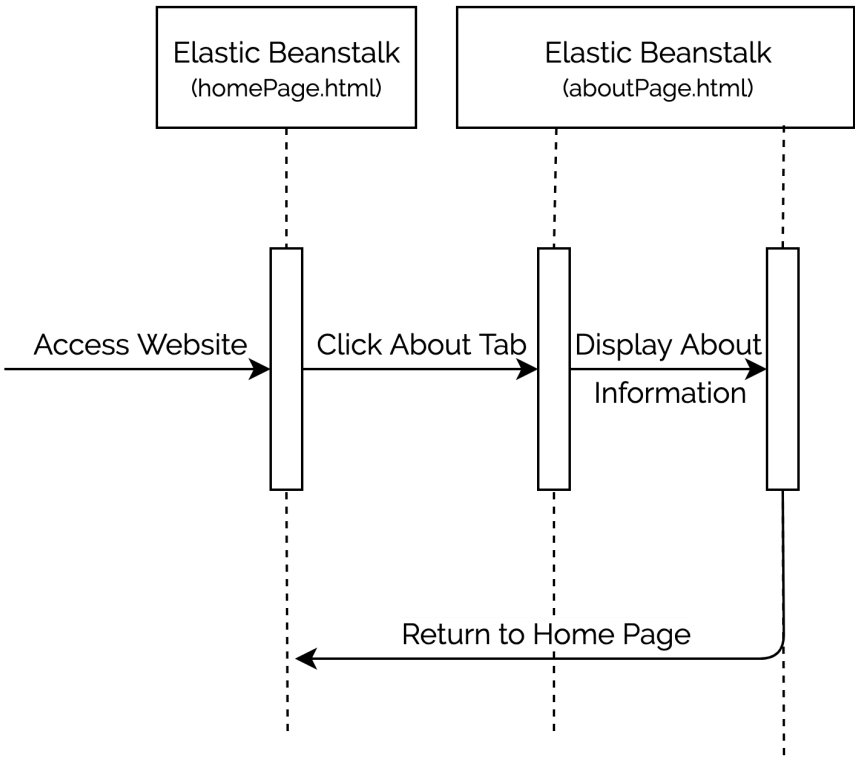
Description: The above diagram shows how the system will interact with each other during use case 5, viewing the hashtag leaderboards. To begin, the user will access the homepage.html deployed on AWS Elastic Beanstalk. From there, they will select the hashtag leaderboard tab taking them to the hashtagLeaderbaords.html page also hosted on AWS Elastic Beanstalk. This will trigger a request to the backend on a Flask server to get the leaderboard information to display to the user. During this step, the Flask server will run the isCurrent() function to see if the leaderboard information in the leaderboard table in the database is up to date. IsCurrent() will return false if the information was last updated over 12 hours age. When this happens it will execute the leaderboard background process to re-calculate the hashtag leaderboard standings. After this, the IsCurrent() function will return true because the data will be less than 12 hours old. From here, the backend will make a request to the database to retrieve this information. MongoDB will then return the results back to the backend which will then display the information back to the user on the hashtagLeaderboards.html page.

UC-6: Display Hashtag Information in Modal



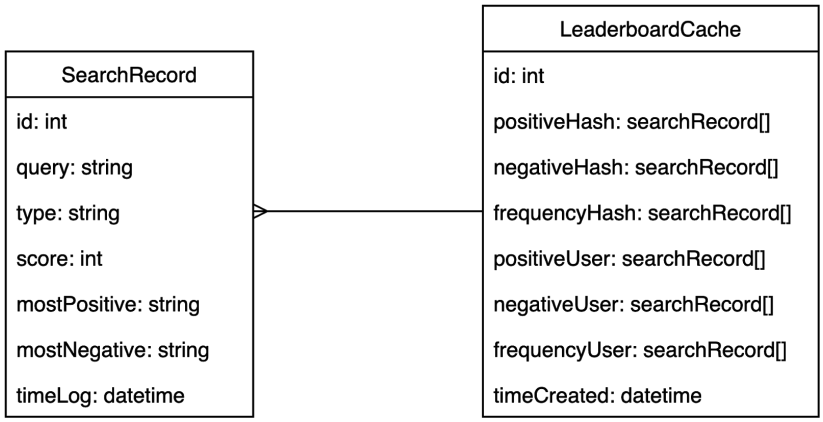
Description: The above diagram shows how parts of the system interact with each other when implementing use case 6, displaying hashtag information in pop-up Modal. This process begins when a user accesses an already loaded hashtag leaderboards page called hashtagLeaderboards.html which is deployed one AWS Elastic Beanstalk. From here they will select a hashtag they wish to see more information about. This will send a request to the backend. From here the backend will make a request to the MongoDB for the information on the selected hashtag. The MongoDB will then return the results back to the backend. From here, the backend will display this information back to the user in a pop-up modal. The user can then view the data and then exit the modal which will return them back to the populated hashtag leaderboards page.

UC-7: View About Us



Description: The above diagram illustrates UC-7, viewing the About Us page. This use case begins when the user accesses the website from the homePage.html page hosted on an Elastic Beanstalk server. From here they will select the About Us tab on the navigation bar which will direct them to the aboutPage.html page which is deployed on AWS Elastic Beanstalk. From here, the About Us information will be displayed for the user to view. When they are finished they can press the Home button on the navigation bar to return them to the home page or click on the Leaderboards drop down tab to view leaderboards.

ERD



SearchRecord - This is the database object that will hold all of the information associated with an individual search query. Its fields are further explained below.

Query: string - This will be the actual search that was entered into the system.

Type: string - This field represents the type of the search. The type will either be username or hashtag. This will be used when separating the searches into username or hashtag when displaying the data in the leaderboards.

Score: int - The score field will hold the numerical response from the algorithm. The score will be used when querying the most negative and most positive leaderboards.

mostPostive: string - Here the most positive tweet, the tweet that generated the highest score from the sentiment algorithm, will be stored in the searchRecord. This will be used to give users more information on their search. It will also be displayed in the pop-up modal on the most positive hashtag and username leaderboards.

mostNegative:string - This field will hold the mostNegative tweet associated with the searched hashtag or username. This will be used to give users more information as well as being displayed in the pop-up modal on the most negative hashtag and username leaderboards.

timeLog: datetime - This will keep track of the time the search was logged into the database. In cases where there are duplicate searches, this field will allow us to select the most recent searchRecord.

LeaderboardCache - This holds all of the information needed to display a leaderboard. Its fields are further explained below.

positiveHash: searchRecord[] - This field is a searchRecord array that will contain the top 10 most positive hashtags that have been searched. This information will be displayed on the hashtag leaderboard page.

negativeHash: searchRecord[] - This field is a searchRecord array that will contain the top 10 most negative hashtags that have been searched. This information will be displayed on the hashtag leaderboard page.

frequencyHash: searchRecord[] - This field is a searchRecord array that will contain the most frequently searched hashtags as of the timeCreated. This information will be displayed on the hashtag leaderboard page.

positiveUser: searchRecord[] - This field is a searchRecord array that will contain the top 10 most positive usernames that have been searched. This information will be displayed on the username leaderboard page.

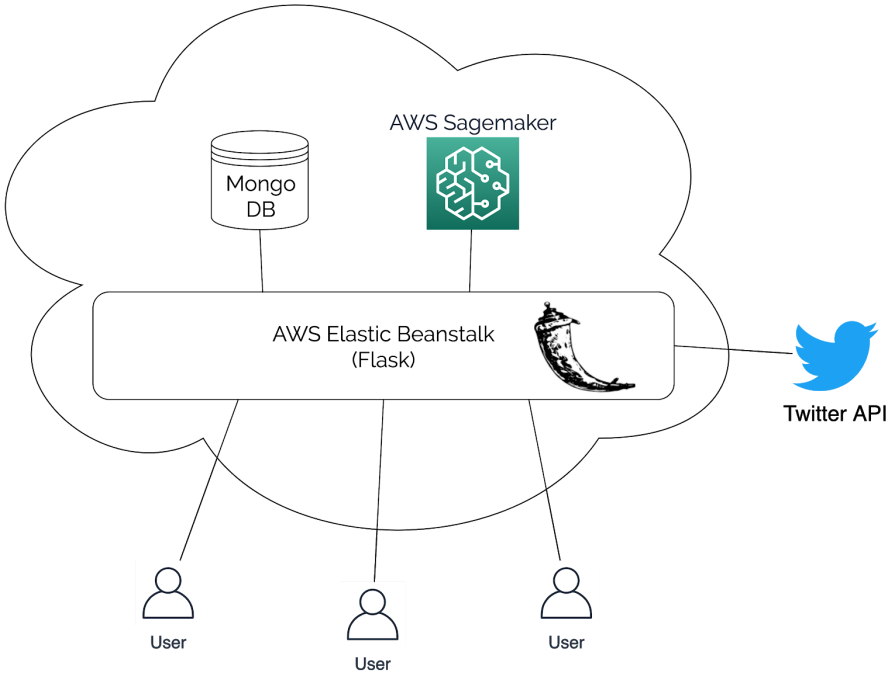
negativeUser: searchRecord[] - This field is a searchRecord array that will contain the top 10 most negative usernames that have been searched. This information will be displayed on the username leaderboard page.

frequencyUser: searchRecord[] - The field is a searchRecord array that will contain the most frequently searched usernames as of the timeCreated. This information will be displayed on the hashtag leaderboard page.

timeCreated: datetime - This will keep track of the time the current leaderboard was created. This will be used to determine if the leaderboard needs to be updated or not.

How SearchRecord and LeaderboardCache are related - There are many searchRecords in one LeaderboardCache. This is shown by the crow's foot notation.

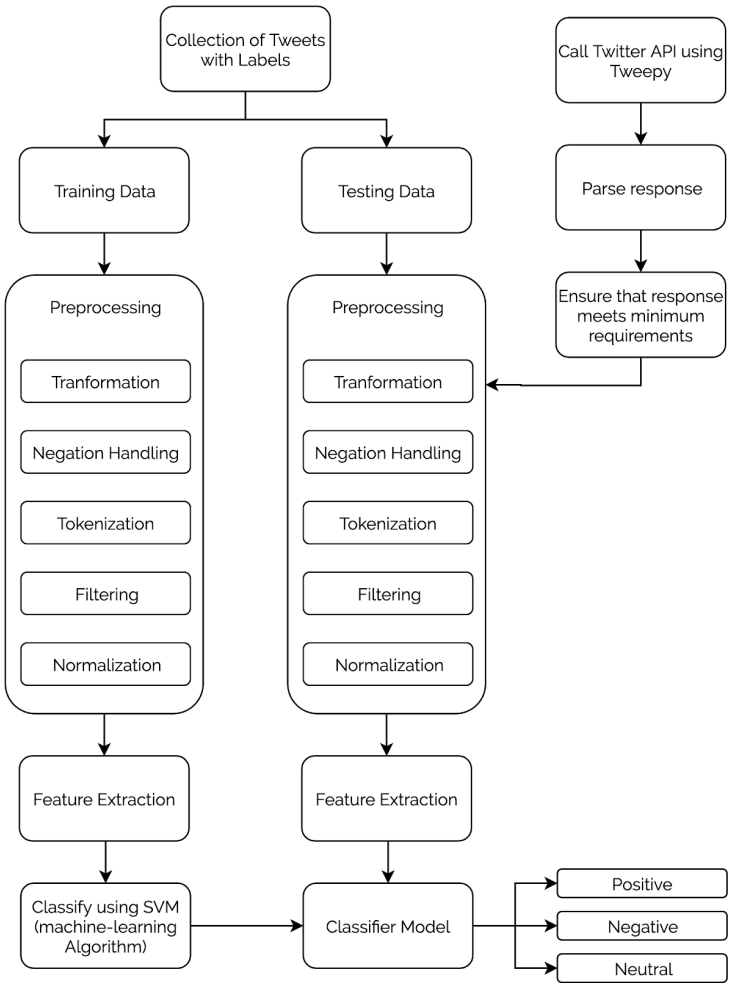
System Architecture



Description: This system relies on AWS Elastic Beanstalk to deploy the Python Flask application. It is the central point of the system. All users will interface at this point of the system. The Flask application communicates with Twitter using Tweepy, calls the sentiment algorithm which is hosted on AWS Sagemaker through an API, and logs search queries and leaderboard history on MongoDB.

Sentiment Algorithm

The proposed architecture for the sentiment algorithm contains three major groups of work where each group is one column in the figure below. Within the first group, the first step is building a collection of Tweets with labels that can then be separated into training and testing sets. The next step is preprocessing the Tweet's. After the Tweet's are preprocessed, it is time to extract the features. Once the features have been extracted, classification will be done using a Support Vector Machine. The second major group of work differs from the first major part only in that it is testing the machine-learning algorithm rather than creating it. The last major part of work involves placing the machine-learning algorithm into production where it will be given Tweets originating from the homepage of the web application and return a score for each Tweet. The last step of the process involves aggregating a group of scored Tweets into one numerical value to be presented to the user.



Training and Testing Data

SVMs fall under supervised learning. This means that the collection of Tweets must come with labels. The Twitter US Airline Sentiment dataset on Kaggle provides all of the necessary fields to train the SVM such as the sentiment, sentiment confidence, and tweet content. There are around 15,000 tweets in the dataset. In order to properly train the SVM, an equal number of positive, negative, and neutral tweets will be randomly selected from the dataset and split into training and testing dataset.

Live Twitter Data using Tweepy

When a user enters a search on the home page, the query will be checked for validity. If the query is valid, then it will be passed to a function which utilizes the Python package, Tweepy, to interface with Twitter's API. The Tweepy package provides results as JSON. This allows for quick and easy parsing of the tweet objects which will then be passed to the preprocessing stage.

Preprocessing

Data preprocessing is an important step in all types of analyses. It is especially important, though, in text-analysis and machine-learning projects. If there is too much irrelevant and redundant information or noisy and unreliable data, then the training phase will perform poorly. In this project, there are five applicable preprocessing steps: transformation, negation handling, tokenization, filtering, and normalization. Each step builds upon the previous step. Consider the following Tweet during each step

Tweet
@fake_person this is an example of the five preprocessing steps. This is not a real Tweet. #fake

Transformation

The transformation step involves cleaning Tweets of URLs, hashtags, and mentions. In this step, Tweets will be converted to lowercase, URLs will be replaced with a generic word (URL), mentions will be replaced with a generic word (AT_USER), and hashtags will be stripped of the hash. Along with the previous Twitter related transformations, all punctuation will be removed and multiple whitespaces will be reduced to a single whitespace.

Step	Tweet
Transformation	AT_USER this is an example of the five preprocessing steps this is not a real tweet fake

Negation Handling

Negations are words that affect the sentiment orientation of other words in a sentence. These words include no, not, never, cannot, should not, would not, etc. Negation handling is an automatic way of determining the scope of negation and inverting the polarities of opinionated words that are affected by a negation. Consider the sentence 'I think that it is not worth going to school.' After negation handling the sentence should resemble 'I think that it is not not_worth not_going not_to not_school.' In this way, the machine-learning algorithm explicitly knows each word's polarity if it depends on a previous negation.

Step	Tweet
Negation Handling	AT_USER this is an example of the five preprocessing steps this is not not_a not_real not_tweet fake

Tokenization

Tokenization involves breaking up a sequence of strings into smaller pieces called tokens. For this project, it is best to split each sequence of strings (one Tweet) into unigrams. A unigram is one word.

Step	Features
Tokenization	this, is, an, example, of, the, five, preprocessing, steps, this, is, not, not_a, not_real, not_tweet, fake

Filtering

The filtering step involves removing stop words. Stop words are commonly used words ('the', 'a', 'an', 'in', etc.) that a search engine has been programmed to ignore. Stop words are useless data and because of this it is best to remove them from the data. This will improve performance without losing accuracy. It will also reduce noise which may increase accuracy. The Python package NLTK contains a list of stopwords that will aid in implementing this step.

Step	Features
Filtering	example, five, preprocessing, steps, not, not_a, not_real, not_tweet, fake

Normalization

The last preprocessing step is normalization. In this step, lemmatization is performed. In Linguistics and NLP, lemmatization is the process of stripping words of their inflectional endings to return to the base or dictionary form of a word which is known as the lemma. Lemmatization does consider the part of speech of the token within the text when deciding how to alter the token.

Step	Features
Normalization	example, five, proprocess, step, not, not_a, not_real, not_tweet, fake

Feature Extraction

Term Frequency - Inverse Document Frequency (TF-IDF) will be used to extract features from each preprocessed Tweet.

Term Frequency is calculated for each term within each document as below.

$$TF(t,\ d)\ =\ \frac{number\ of\ times\ term(t)\ appears\ in\ document(d)}{total\ number\ of\ terms\ in\ document(d)}$$

Inverse Document Frequency measures how important a word is to differentiate each document by following the calculation as below.

$$IDF(t,\ D)\ =\ log(\frac{total\ number\ of\ documents(D)}{number\ of\ documents\ with\ the\ term(t)\ in\ it})$$

Once the values for TF and IDF exist, it is possible to calculate TF-IDF by multiplying the two values together. From here, it is time to select the features as terms with the highest TF-IDF values.

Sentiment Classification using Support Vector Machines

A support vector machine is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In the application, there will be two hyperplanes to signify the difference between positive, negative, and neutral. A linear kernel will be used in this model.

Find the biggest margin, M

$$\underset{\beta_0,\ \beta_1, ...,\ \beta_{p,}\ \varepsilon_1, ...,\ \varepsilon_n}{max} \quad M$$

Subject to

$$\sum_{j=1}^{p} \beta_j^2 = 1$$

And

$$y_i(\beta_0 + x_i^T \beta) \geq M(1 - \varepsilon_i),\ \forall\, i = 1, ..., n.$$

$$\varepsilon_i \geq 0,\ \ \sum_{i=1}^{n} \varepsilon_i \leq C$$

C is a tuning parameter which controls how much the model will "budget" for anomalies. When C is 0, this is the maximal margin classifier. Many different cross-validation techniques will be implemented in determining which C is best. One must always be careful and consider the bias-variance tradeoff.

The SVM will be implemented in Python using functionalities provided by sklearn.

Aggregation of Results

When you reach this point in the process, many different tasks have already been completed. A user has entered a query on the homepage. The result (a collection of tweets) of that query was sent to Sagemaker to be classified. Sagemaker returned the classification of each Tweet. Now, it is time to aggregate the results.

The team has defined three different results: the most positive tweet, the most negative tweet, and the overall score. The most positive and negative tweet is self explanatory; however, the overall score is not. In order for this score to be comparable across many different queries, it must be adjusted in some way. First, add up all the positive tweets. Then subtract all of the negative tweets. Lastly, divide this number by the total number of tweets that the search query returned. This number will represent the over positivity/negativity of the user or hashtag.

Pseudocode

```
def search(query, type):
    if type == user:
        results = use Tweepy to collect the user's tweets.

        if the user is not public:
            return error

        collectionOfTweets = retults.tweets
        if the user does not have sufficient tweets
            return error

        preprocessedTweets = preprocessing(collectionOfTweets)
        classifications = call Sagemaker API with preprocessedTweets
        aggregate = aggregateResults(classifications)
        return aggregate to be displayed
    if type == hashtag:
        results = use Tweepy to collect tweets with the matching hashtag

        collectionOfTweets = retults.tweets
        if the hashtag does not have sufficient tweets
            return error

        preprocessedTweets = preprocessing(collectionOfTweets)
        classifications = call Sagemaker API with preprocessedTweets
        aggregate = aggregateResults(classifications)
        return aggregate to be displayed
    else:
        return error
```

```
def preprocessing(collectionOfTweets):
    preprocessedTweets = []

    for tweet in collectionOfTweets:
        # Transformation
        replace URLs with 'URL'
        replace mentions with 'AT_USER'
        strip hashtags of the hash
        remove punctuation
        reduce multiple whitespaces to single whitespace
        reduce unnecessarily repeated letters in words

        # Negation Handling
        use NLTK to apply negation handling

        # Tokenization
        use NLTK to tokenize the tweet

        # Filtering
        use NLTK to remove stop words

        # Normalization
        use NLTK to lemmatize each word

        add to preprocessedTweets

    return preprocessedTweets

def aggregateResults(classifications)
    totalScore = 0

    for classification in classifications:
        if classification == positive
            totalScore++
        if classification == negative
            totalScore--

    return totalScore / len(classifications)
```

Leaderboard Construction

This system will keep track of six total leaderboards. Most positive, most negative, and most frequently searched for both usernames and hashtags. The leaderboards will have a limit of ten entries. The leaderboards will be constructed from querying the SearchRecord table.

Queries Pseudocode

Most Positive Username/Hashtag

```
SELECT * FROM SearchRecord
WHERE type = [username or hashtag]
ORDER BY score DESC
LIMIT 10;
```

Most Negative Username/Hashtag

```
SELECT * FROM SearchRecord
WHERE type = [username or hashtag]
ORDER BY score ASC
LIMIT 10;
```

Most Frequent Username/Hashtag

```
SELECT *, count(query) as count
FROM SearchRecord
WHERE type = [username or hashtag]
GROUP BY query
ORDER BY count DESC
LIMIT 10;
```

GUI

The Graphical User Interface images listed below are roughly how a user will interact with the website. Below each image is a caption that describes the basic functionality.

Main Page

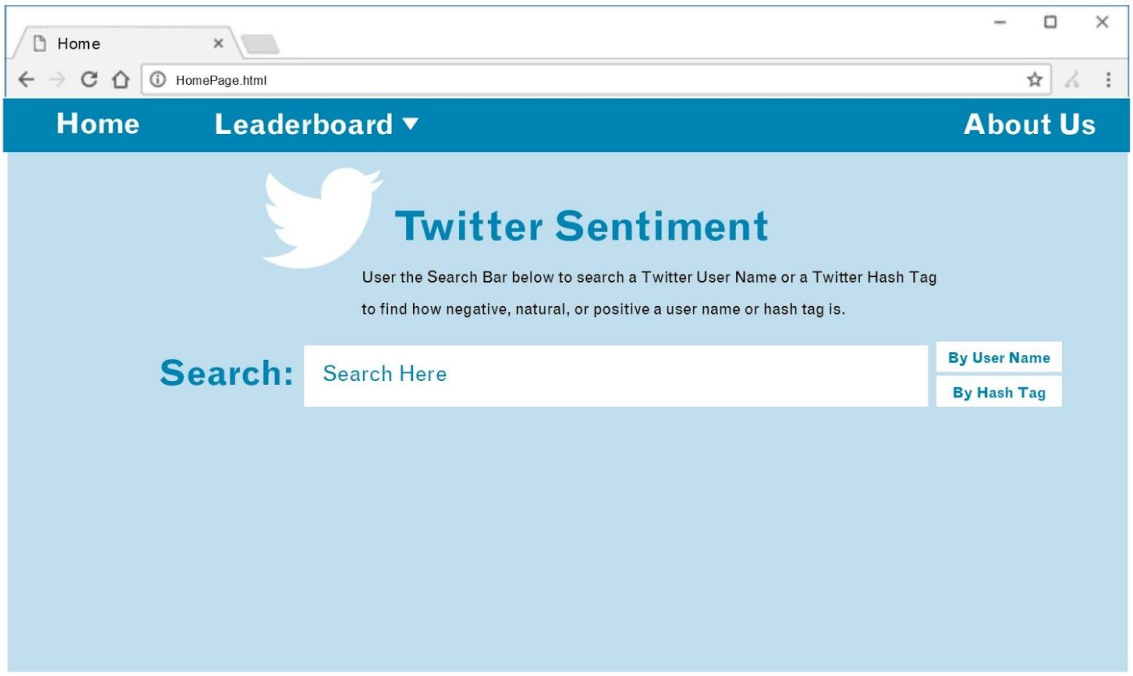


Figure 1. This is the landing page of the website. It will have a menu bar across the top allowing users to view what tabs are available. It will have a brief description of how to search and where to search a username or hashtag.

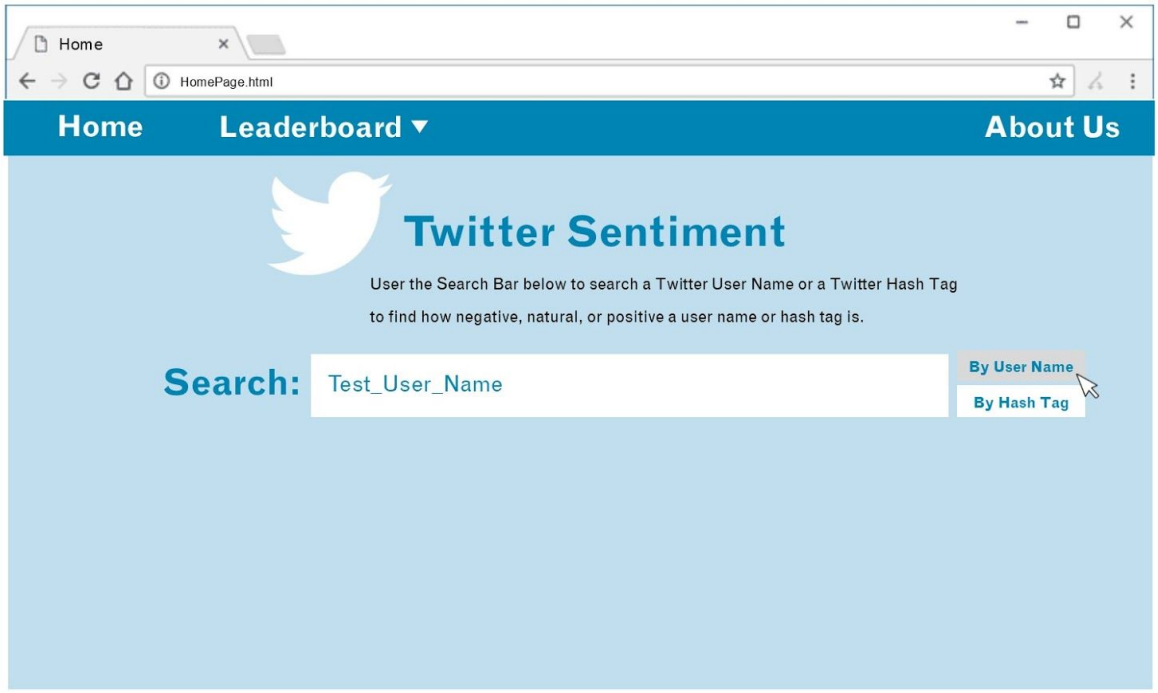


Figure 2. A user can type in either a username or hashtag in the search bar. Then they can search it by username or hashtag. When they hover over either the username or hash tag box it will turn gray so they know which one they click on.

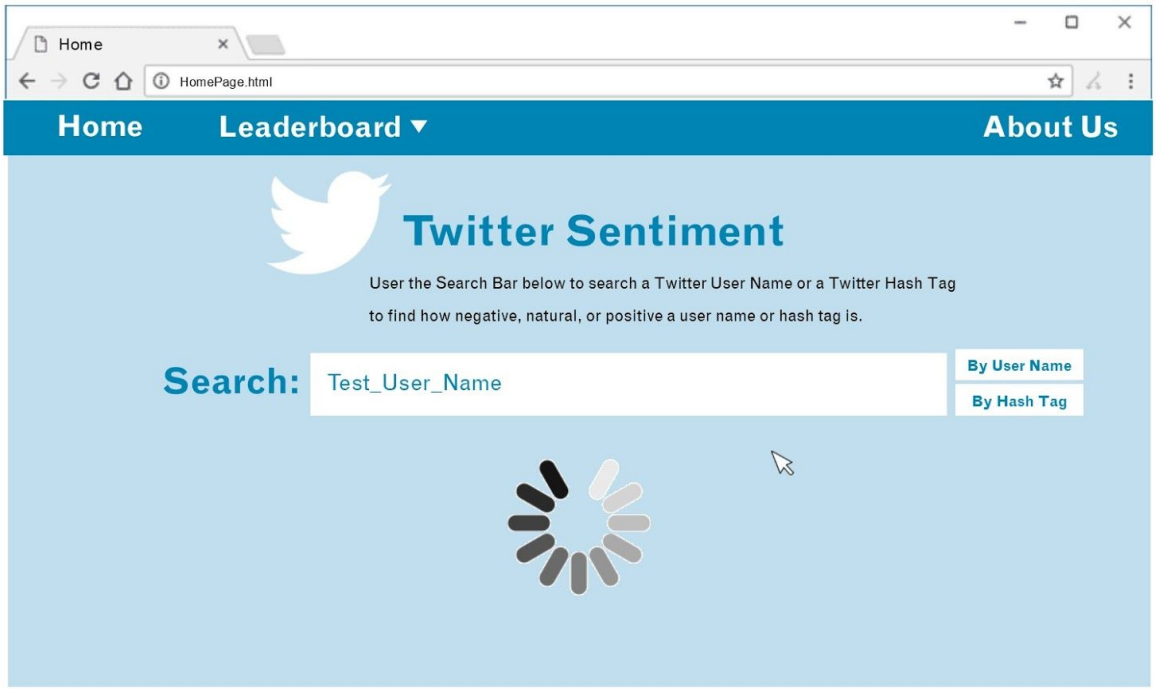


Figure 3. When a user clicks on the username box to search, a loading symbol will appear so the user can get dynamic feedback.

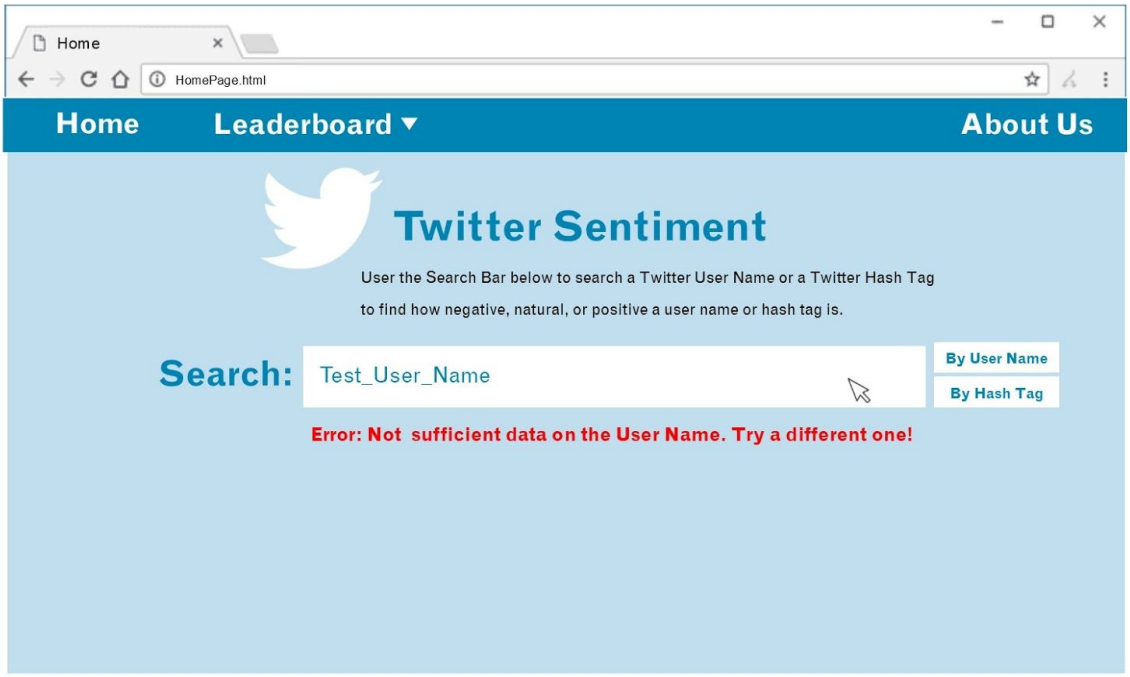


Figure 4. If there is not sufficient data, meaning there is no such user, an error will appear to let the user know to try a different username or hashtag.

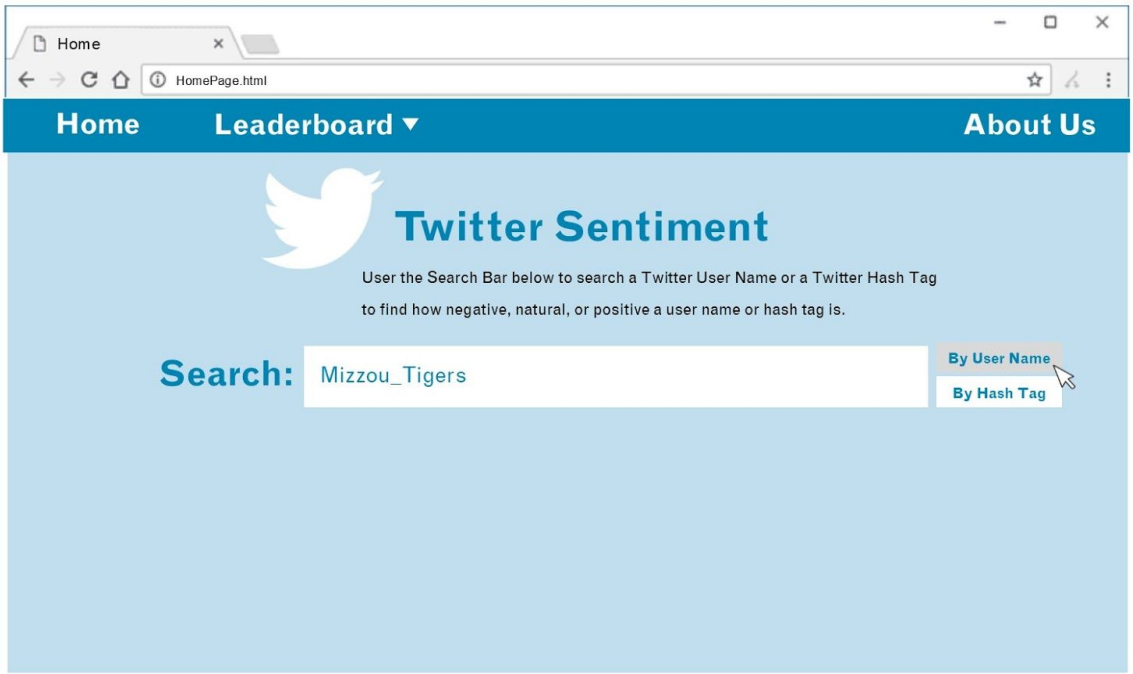


Figure 5. A user can type in either a username or hashtag in the search bar. Then they can search it by username or hashtag. When they hover over either the user name or hash tag box it will turn gray so they know which one they click on. This user is searching by a username.

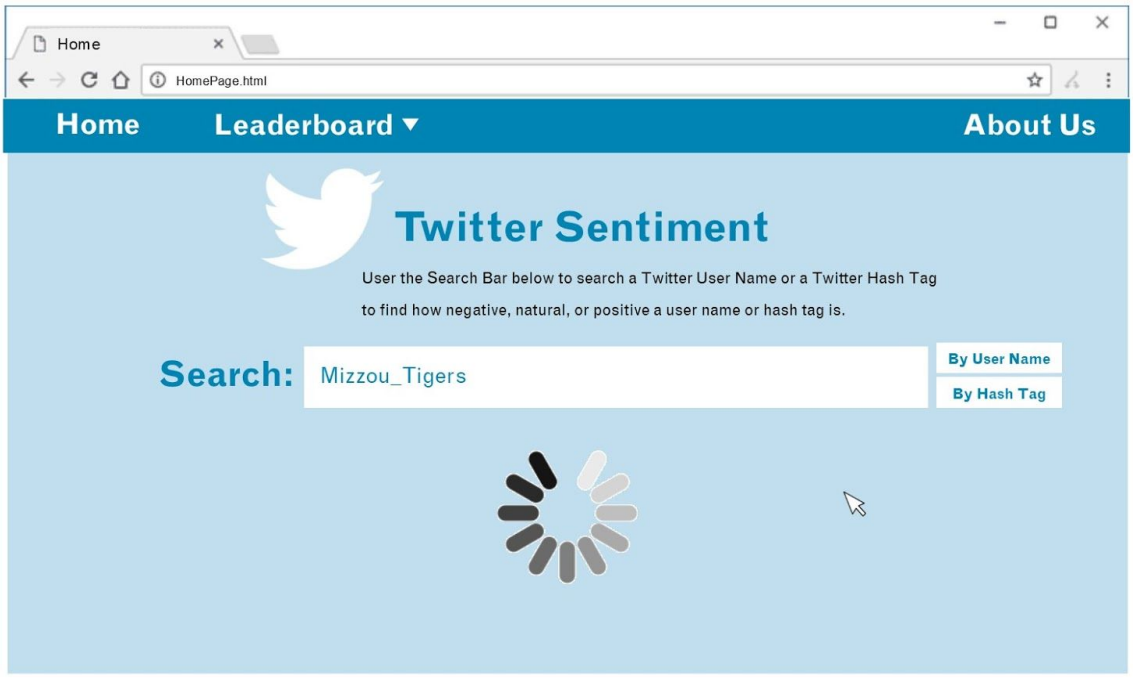


Figure 6. When a user clicks on the username box to search, a loading symbol will appear so the user can get dynamic feedback.

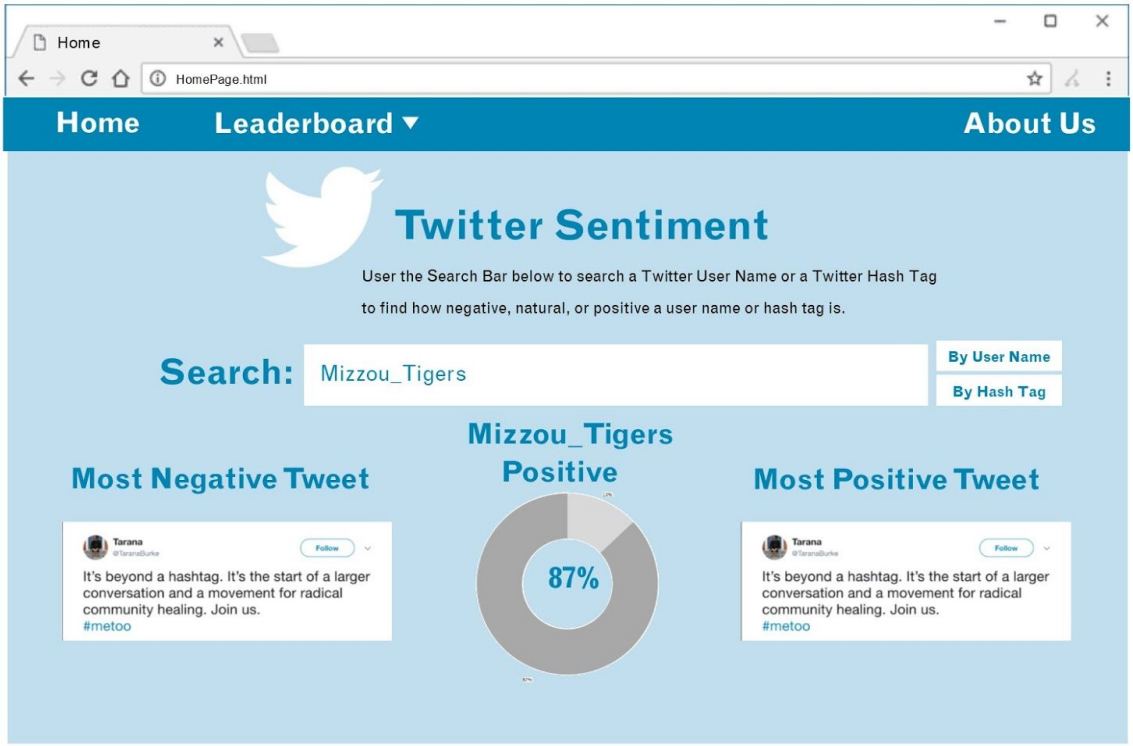


Figure 7. If the search is successful, results will appear. A donut chart that displays the score will appear in the middle. The most negative tweet from the user will appear on the left, and the most positive tweet of the user will display on the right.

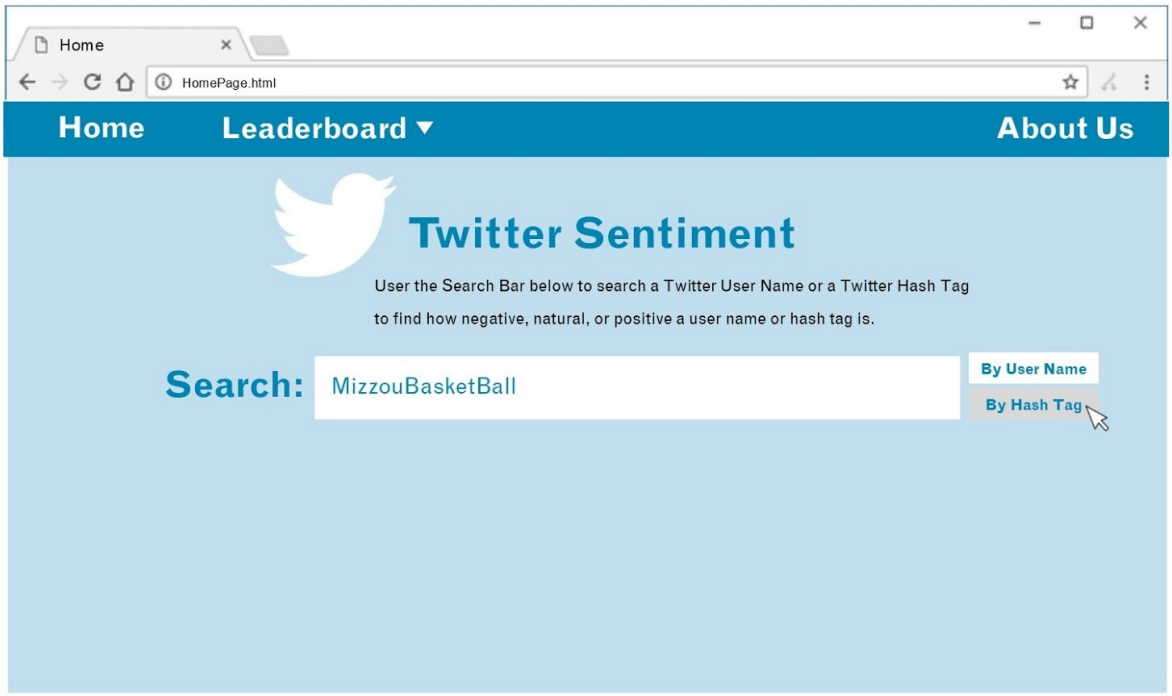


Figure 8. A user can type in either a username or hashtag in the search bar. Then they can search it by username or hashtag. When they hover over either the username or hashtag box it will turn gray so they know which one they click on. This user is searching by a hashtag.

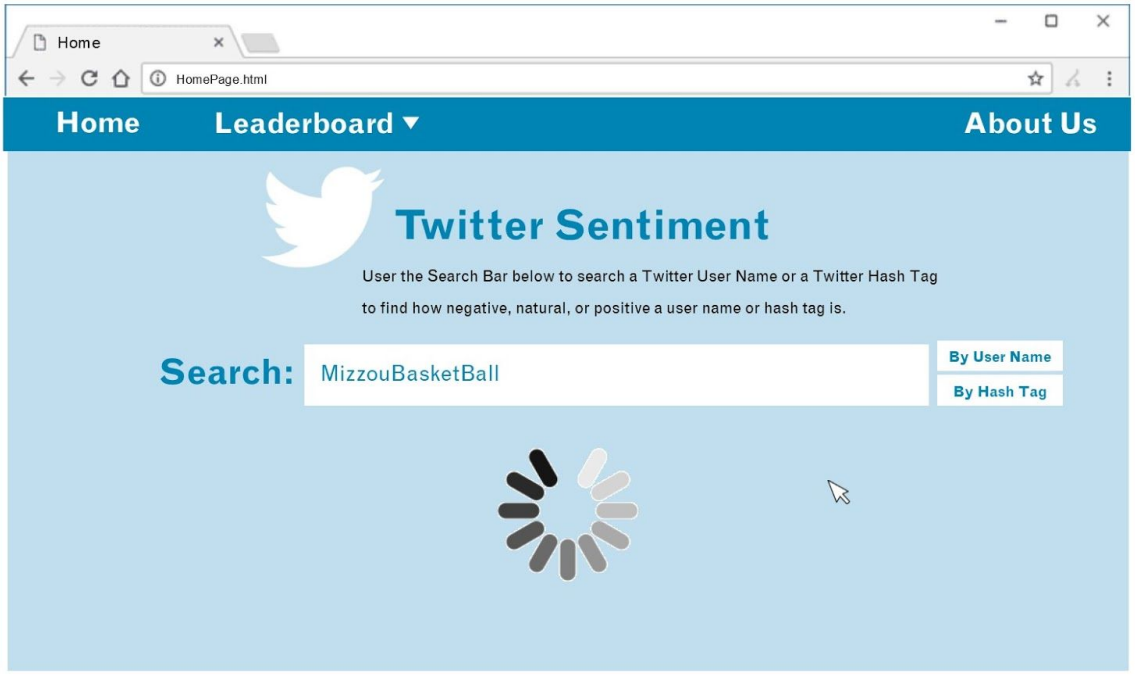


Figure 9. When a user clicks on the hashtag box to search, a loading symbol will appear so the user can get dynamic feedback.

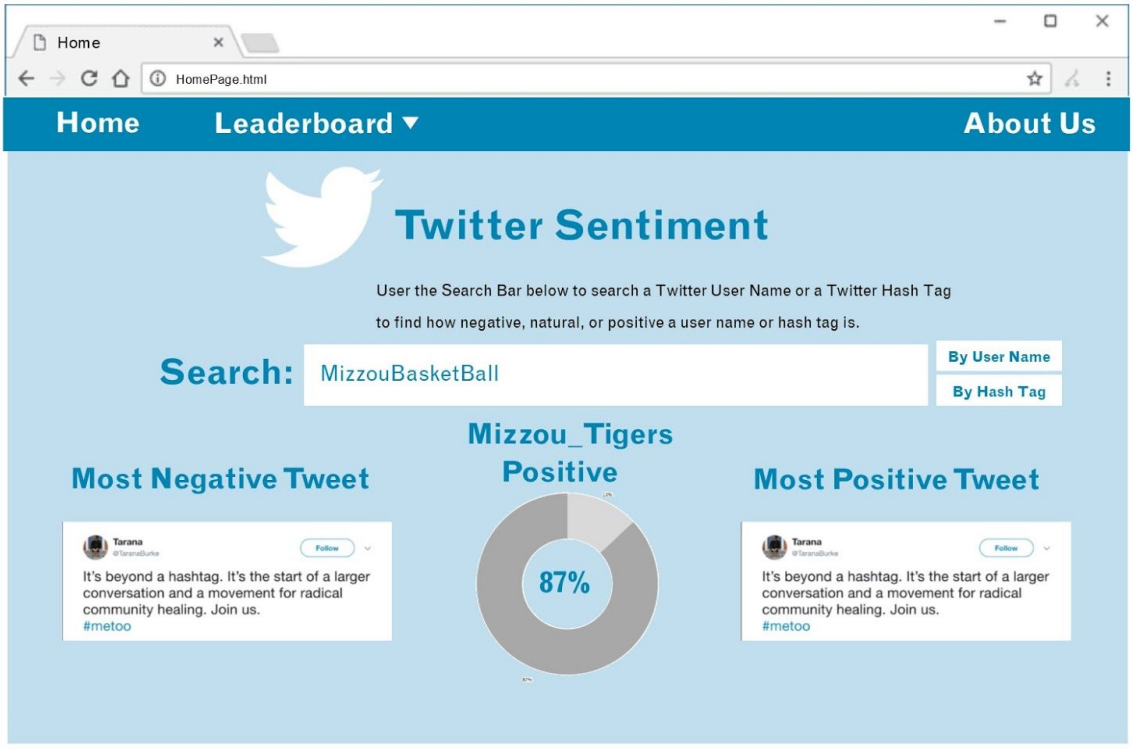


Figure 10. If the search is successful, results will appear. A donut chart that displays the score will appear in the middle. The most negative tweet from the user will appear on the left, and the most positive tweet of the user will display on the right.

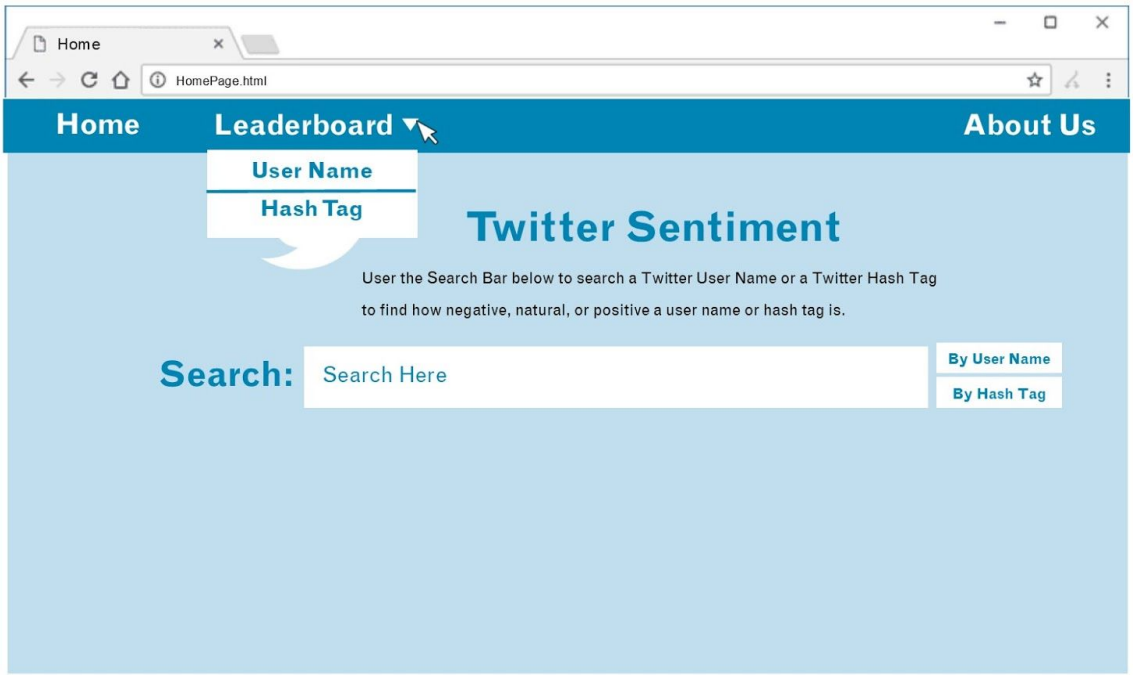


Figure 11. A user can hover over the leaderboard menu bar and it will give two different options for leaderboards.

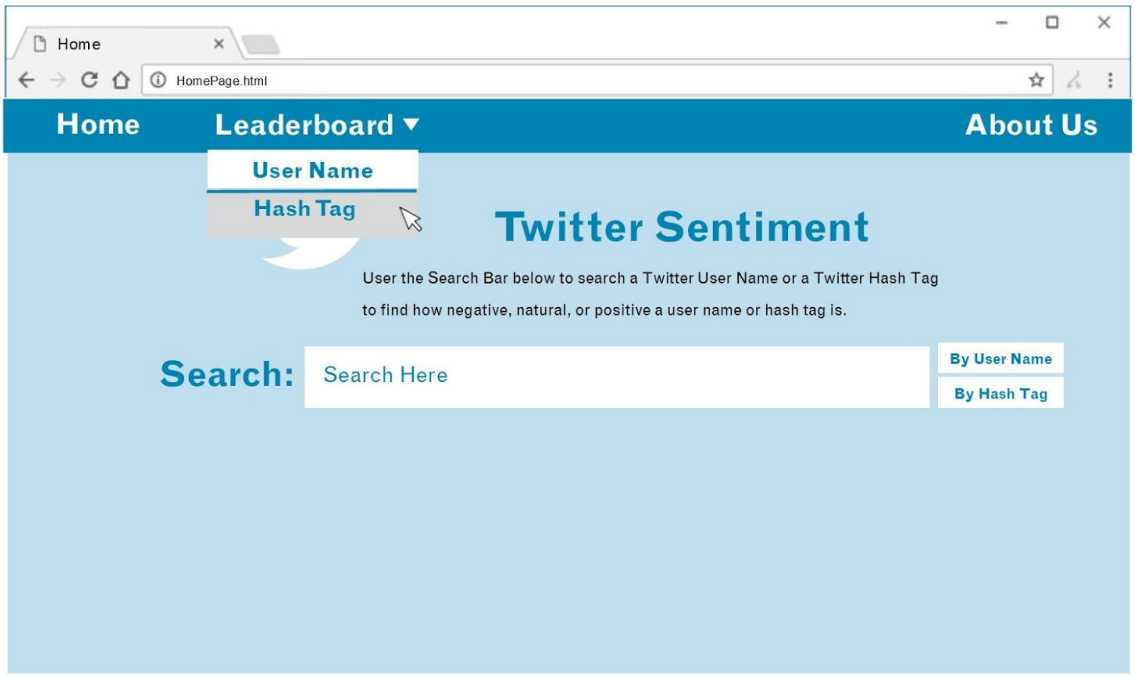


Figure 12. A user can hover over the leaderboard menu bar and it will give two different options for leaderboards. It will turn gray indicating which leaderboard the user will click on.

Leaderboard Page

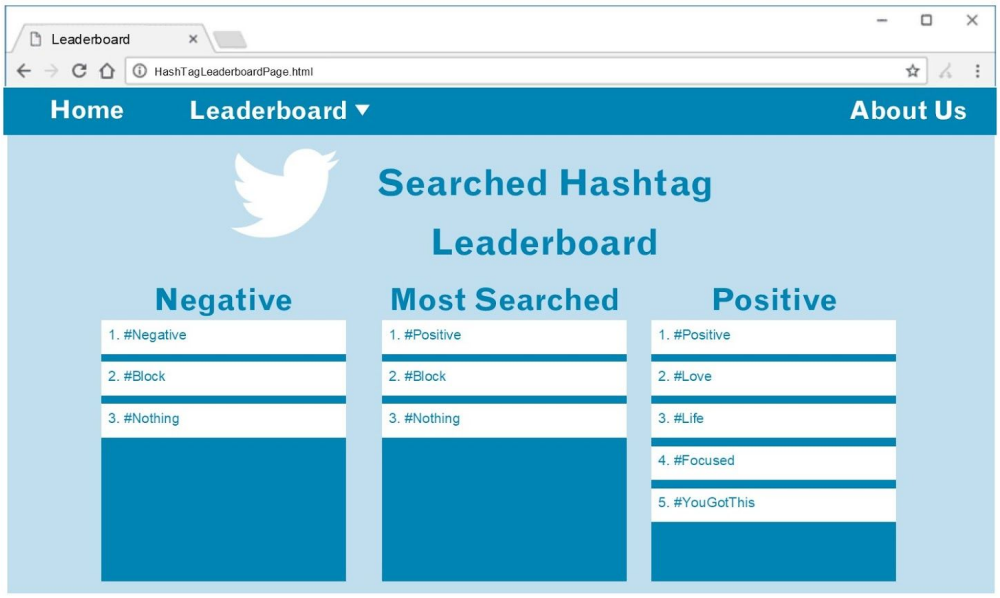


Figure 13. When a user clicks on the Hashtag Leaderboard three different leaderboards will appear. On the left the most negative searched hashtags will appear. In the middle, the most searched hashtags will appear. On the right, the most positive scoring hashtags will appear.

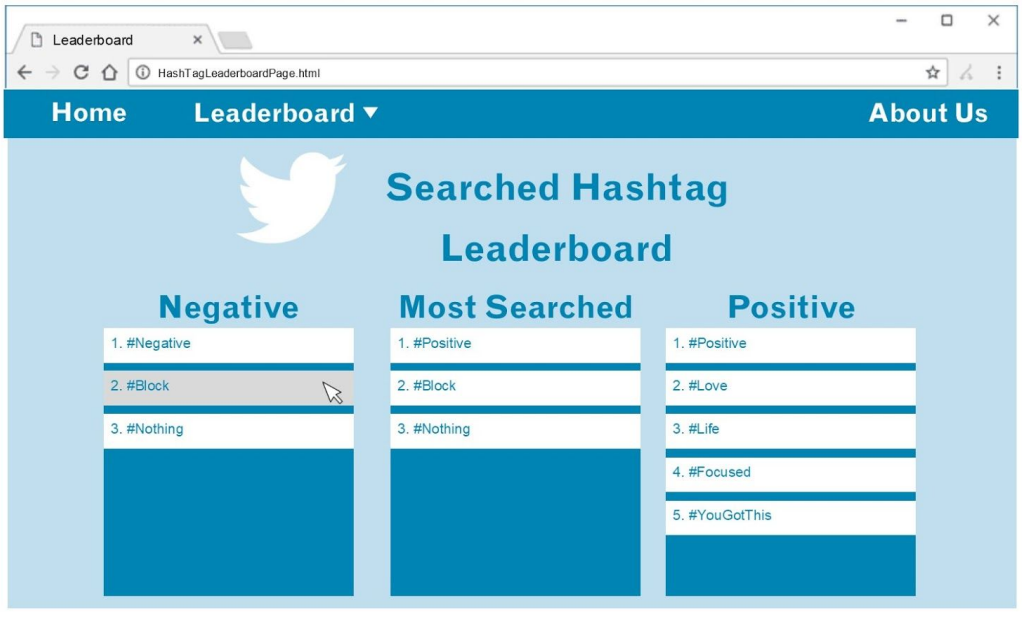


Figure 14. A user can hover over any hashtag on the leaderboards and it will display gray indicating they can click on it.

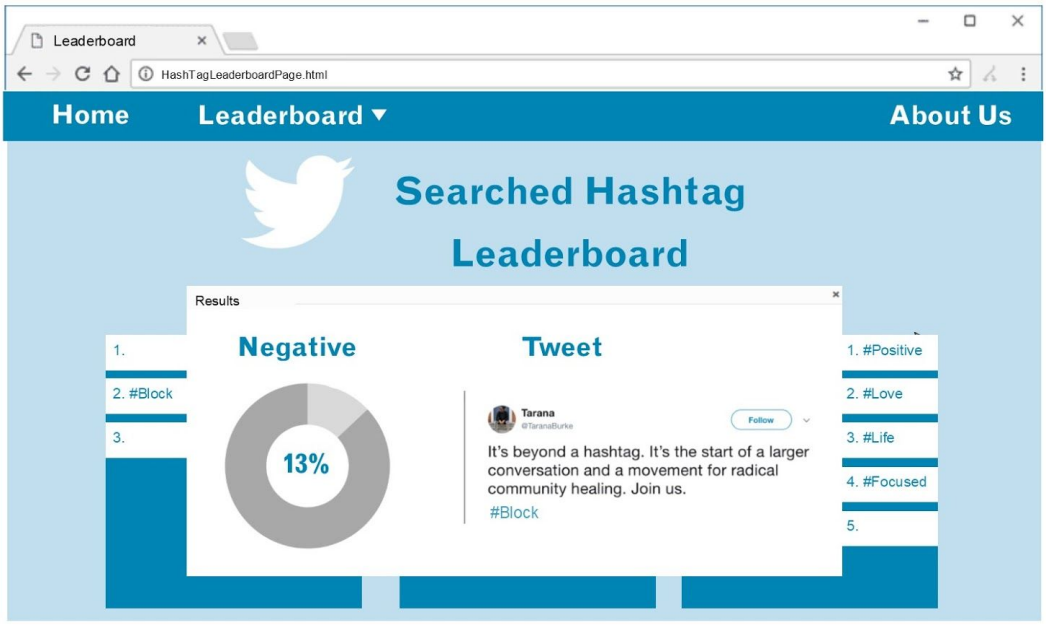


Figure 15. When a user clicks on any hashtag, the results display in a modal that were displayed when first searched. A donut chart will display on the left showing their score and their most negative tweet will display since they clicked on a hashtag in the negative leaderboard.

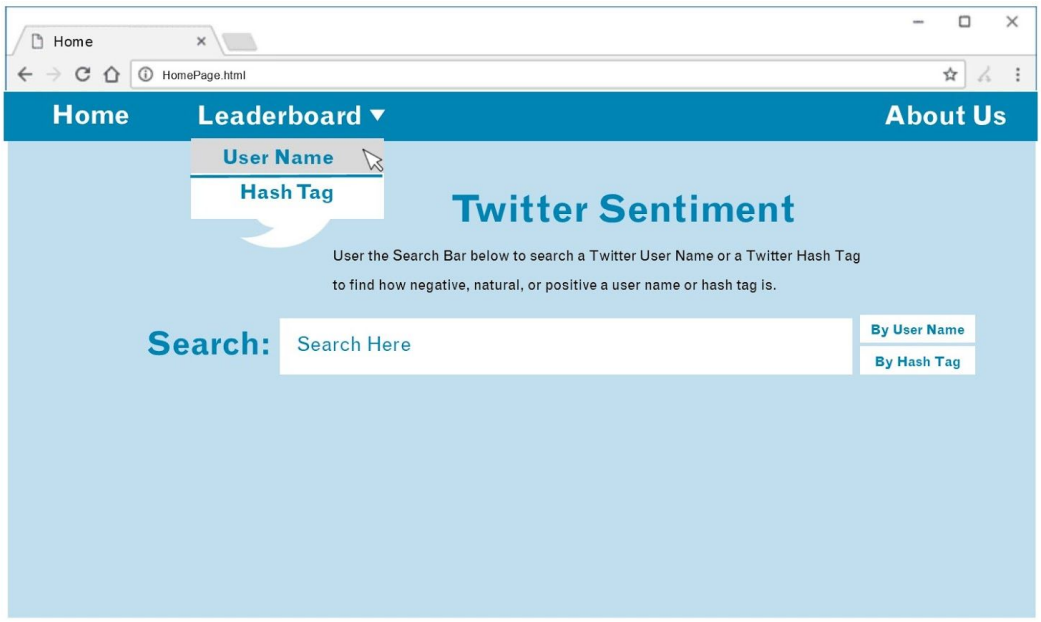


Figure 16. A user can hover over the leaderboard menu bar and it will give two different options for leaderboard. It will turn gray indicating which leaderboard the user will click on.

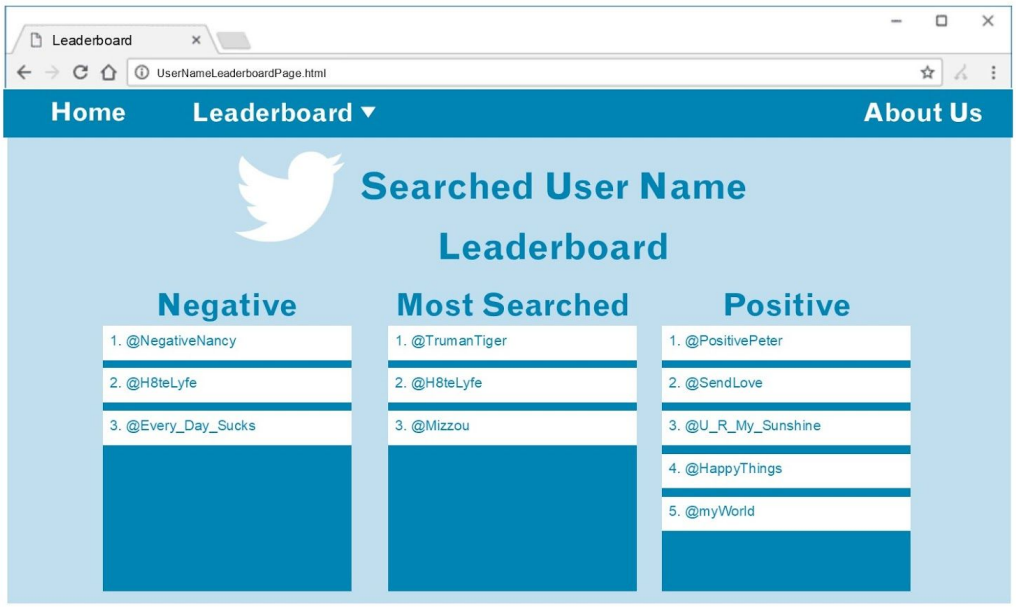


Figure 17. When a user clicks on the Username Leaderboard three different leaderboard will appear. On the left, the most negative searched usernames will appear. In the middle, the most searched usernames will appear. On the right, the most positive scoring usernames will appear.

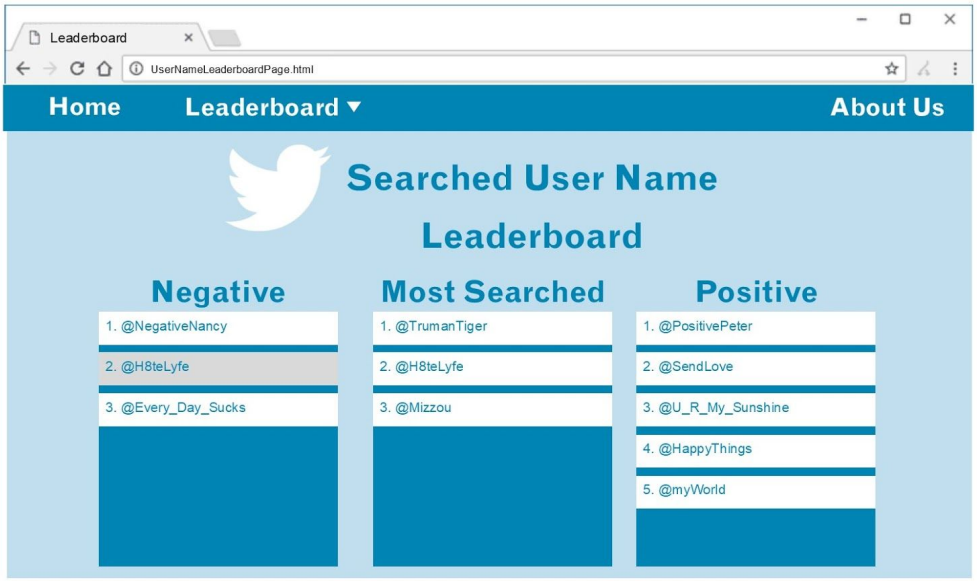


Figure 18. A user can hover over any username on the leaderboards and it will display gray indicating they can click on it. This user will select the username.

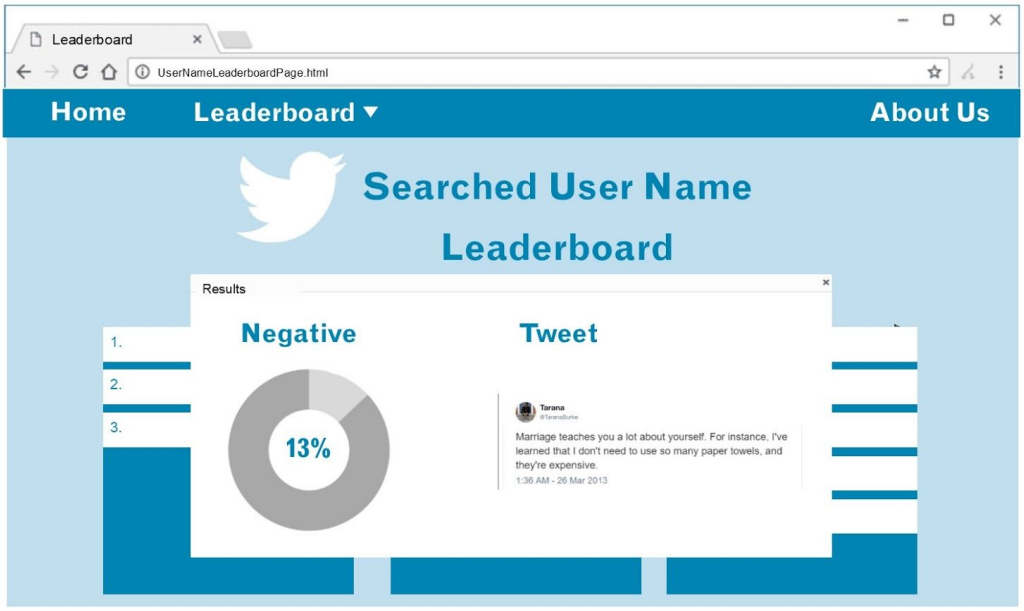


Figure 19. When a user clicks on any username, the results display in a modal that were displayed when first searched. A donut chart will display on the left showing their score and their most negative tweet will display since they clicked on a hashtag in the negative leaderboard.

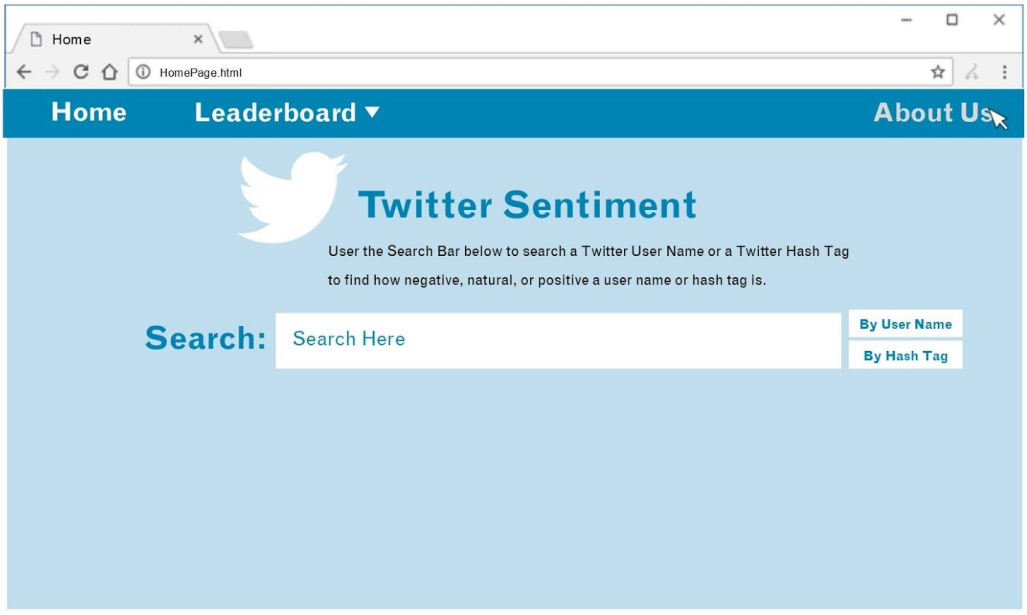


Figure 20. A user can hover over the About Us menu bar and it will turn gray indicating they can click on it.

About Us Page

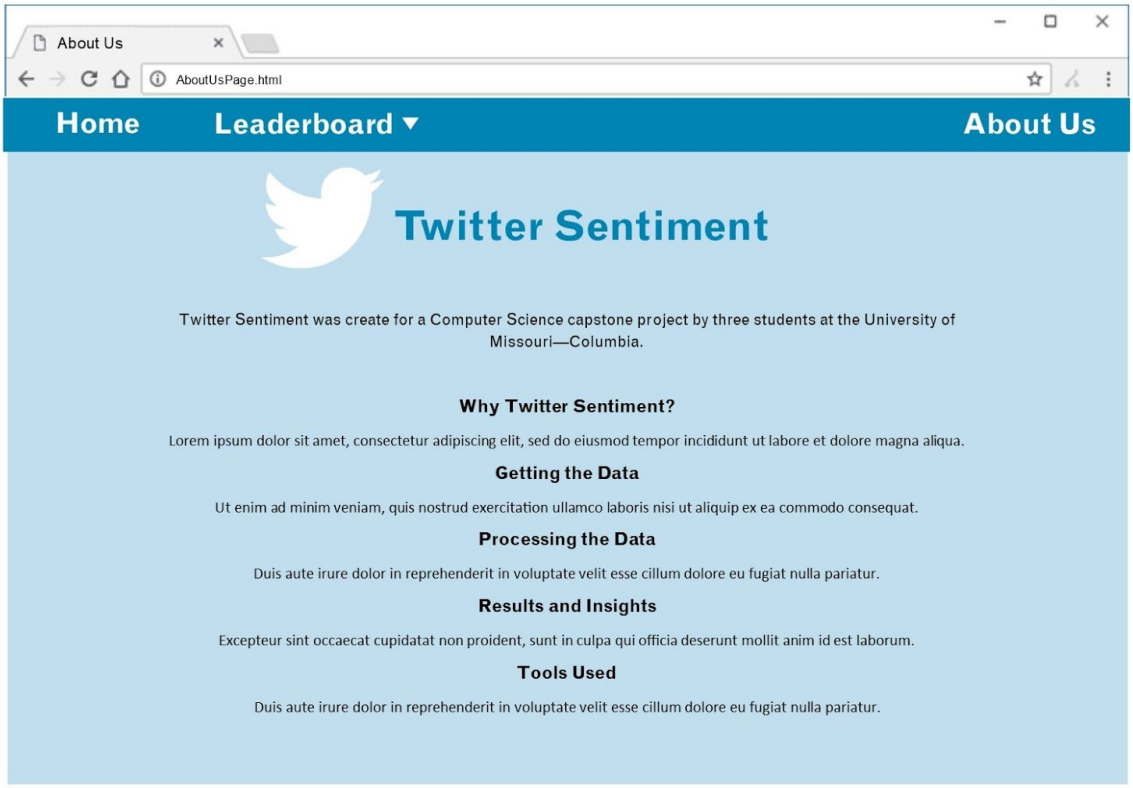


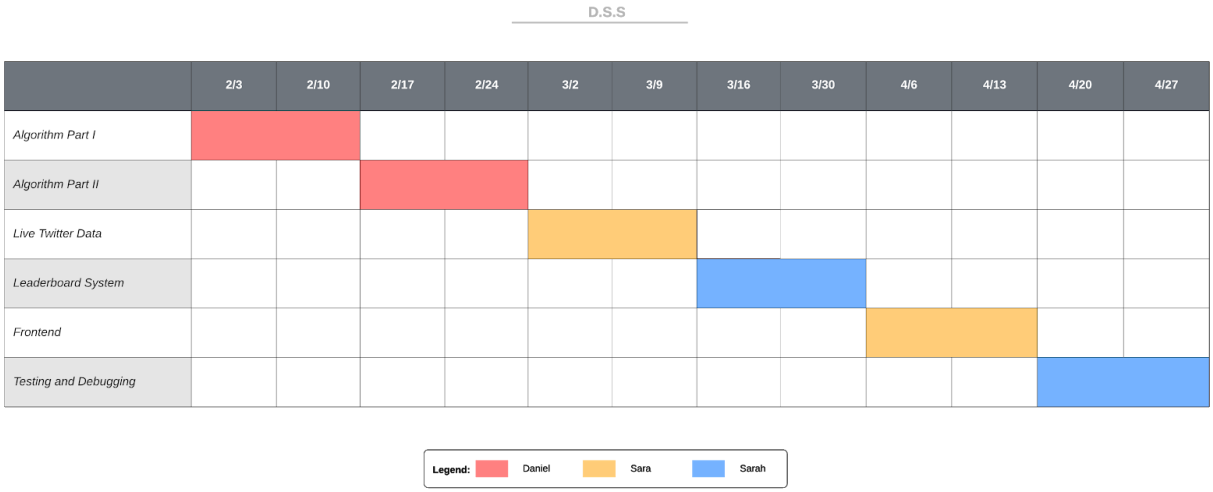
Figure 21. Once a user clicks on the About Us tab, it will bring them to a new page with information on D.S.S.'s capstone project.

Development Strategy

D.S.S. chose to use the agile development strategy because it is not as rigid as spiral or waterfall. This development strategy is flexible and will allow changes to be made more quickly throughout the development process. The sprint process will be two weeks long with six sprints to make sure the tasks are accomplished on time. D.S.S. will communicate as frequently as possible in order to make sure the tasks get accomplished on time according to the development timeline listed below. The best communication process for D.S.S. is group texting. Also, communication by word of mouth, such as verifying upcoming meeting dates and times. There has not been any problems with the group texting process so far and there are no foreseen issues that will arise with this communication process. The group will plan on meeting weekly to have scrum and discuss the progress on the sprints. Here the team will share any roadblocks that are happening and how to overcome them. Additionally, the group will share updates on the sprint via the group text to keep all members up to date and busy with work.

Development Timeline

D.S.S. chose to do a twelve week schedule with six sprints. The sentiment algorithm will be the most intense, so the group decided to split it into two sprints. The twelve week schedule will give the group enough wiggle room to catch up if a team member falls behind.



Sprints

Sprint	Tasks	Work Delegation
1. Algorithm part 1	• Create preprocessing functions • Create feature extraction functions • Train/Build SVM	Feb 3rd - Feb 16th Daniel
2. Algorithm part 2	• Test/cross-validate and adjust as needed • Deploy to AWS Sagemaker • Build aggregation function	Feb 17th - Mar 1st Daniel
3. Use Tweepy to query live Twitter data and send to algorithm	• Build database and table for search records • Implement Tweepy search and validation	Mar 2nd - Mar 15th Sara
4. Leaderboard System	• Create queries to build leaderboards • Build database to store leaderboards • Leaderboard background process	Mar 16th - April 5th Sarah
5. Frontend	• Homepage/search page • Results page • Leaderboard page with modals • About Us page	April 6th - April 19th Sara
6. Testing and debugging	• User interface • Leaderboard system • Sentiment algorithm	April 20th - May 3rd Sarah

The chart above describes the teams development timeline, what each sprint will consist of, and who is responsible for completing it.

Work Delegation

The work delegation is covered in the table above. Each team member will be assigned to a specific sprint. There are six sprints in total, hence each team member will do two sprints. Team members are responsible and expected to help out if an issue aries or a person finds it is too much work to get done in the duration of the spirit.

Alternate Strategies

For this project the team had to make decisions on what server to host the frontend application. The team decided to use an AWS Elastic Beanstalk server. This decision was covered in the research document. If issues are encountered, the team will be flexible and switch to using an AWS EC2 server. The team had to also decide where to host the sentiment algorithm. After researching options available, D.S.S. chose to use AWS SageMaker. As of now there are no foreseeable issues with this decision, but if an issue arises, the group will reevaluate available options. The database that is being used in this project is MongoDB. An alternative to this would be using MySQL. The reasoning behind choosing MongoDB over MySQLwas done in the research portion of the project. Lastly, the main decision for this project was what algorithm to use. D.S.S. chose to use the Support Vector Machine algorithm. There are multiple alternatives to this but if there is a roadblock encounter while using this algorithm the team plans to pivot and attempt to use a different algorithm such as TextBlob. Support Vector Machine was chosen because it provides the highest accuracy, the ability to support bulk processing, a neutral classification. Although more difficult to implement, SVM will give the best results and therefore was the best decision for this project.

Possible Troubles

One possible pitfall is that the Support Vector Machine (SVM) algorithm will be inaccurate. There are many different solutions to fall back on. One of which is TextBlob, a simplified text processing library in Python. Another alternative solution is AWS Comprehend, which is an NLP service that uses machine-learning to find insights and relationships in text. Another pitfall D.S.S. could run into is that the testing data set is not large enough or diverse. To resolve this issue, the group will have to spend countless hours finding more test data. A potential pitfall is the chosen methodologies, such as AWS Elastic Beanstalk, AWS SageMaker, and MongoDB will not interact as easily as originally thought. A way to work around this, is to keep everything on one methodology such as AWS EC2.

Testing Plan

This will be completed next semester.

Works Cited

[1] N.A. “AWS Elastic Beanstalk.” *Website.* AWS.Amazon.com. N.D.

[2] DataFlair Team. “AWS Elastic Beanstalk (AWS EBS) - Unique Benefits & Advantages.” *Website*. Data-Flair.Training. 15 September 2018.

[3] N.A. “Amazon EC2.” *Website.* AWS.Amazon.com. N.D.

[4] Andrew Froehlich. “Amazon EC2 Dedicated Hosts: Pros And Cons.” *Website.* InformationWeek.com. 15 December 2015.

[5] N.A. “Get to Know Azure.” *Website.* Azure.Microsoft.com. N.D.

[6] Robert Bengtsson. “Pros and Cons of Microsoft Azure.” *Website.* CodeAddiction.net. 15 April 2016.

[7] N.A. “Amazon Relational Database Service (RDS).” *Website.* AWS.Amazon.com. N.D.

[8] Ananth Kumar. “Is Amazon RDS relational database service good? What are the advantages and disadvantages of using it compared to running MySQL on an EC2 box yourself?” *Website.* Quora. 27 June 2018.

[9] Anas Al-Masri. “Creating the Twitter Sentiment Analysis Program in Python with Naive Bayes Classification” *Website. Towards Data Science.* 13 February 2019.

[10] N.A. “TextBlob: Simplified Text Processing” *Website.* https://textblob.readthedocs.io/en/stable/ N.D.

[11] Niketan Jivane. “Twitter Sentiment Analysis of Movie Reviews using Machine Learning Techniques.” *Website. Medium.* 20 November 2018.

[12] N.A. “Amazon SageMaker” *Website.* https://aws.amazon.com/sagemaker/ N.D.

[13] N.A. “Amazon SageMaker FAQs” *Website.* https://aws.amazon.com/sagemaker/faqs/ N.D.

[14] Enias Cailliau “Is SageMaker Worth It” *Website.* https://medium.com/radix-ai-blog/is-sagemaker-worth-it-4b78a2082ca9 Nov 13, 2018

[15] Matan Sarig “MongoDB vs MySQL: The Difference Explained” *Website.* https://blog.panoply.io/mongodb-and-mysql Nov 27, 2017

[16] Arun Bansal “Pros and Cons of ‘RDS vs EC2’ for MySQL with AWS” *Website.* https://serverguy.com/comparison/pros-cons-rds-vs-ec2-mysql-aws/ Aug 26, 2019

[17] N.A. “Advantages of MongoDB | Disadvantages of MongoDB “ *Website.* https://data-flair.training/blogs/advantages-of-mongodb/ Sept 4, 2018

[18] Steve Dille “How to Decide Between Amazon SageMaker and Microsoft Azure Machine Learning Studio” *Website.* https://towardsdatascience.com/how-to-decide-between-amazon-sagemaker-and-microsoft-azure-machine-learning-studio-157a08af839a May 15, 2019

[19] Samuel Greengard “Microsoft Azure Machine Learning Studio: Product Overview and Insight” *Website.* https://www.datamation.com/artificial-intelligence/microsoft-azure-machine-learning.html Feb. 14, 2019

[20] N.A. “The Database for Modern Applications“ *Website.* https://www.mongodb.com/ N.D