Risk Prediction Sentiment Analysis

Authors:
Tom Himler, Evan Joaquin, Eriq Taing, Akshay Akula, Pranav Sharma

Instructor: Edward A. Fox
Client: Christian Johnson

CS 4624 Multimedia, Hypertext, and Information Access
Department of Computer Science
Virginia Tech, Blacksburg VA 24061

December 8, 2021
# Contents

List of Figures .................................................. 4

1 Executive Summary ........................................... 5

2 Introduction ..................................................... 6
   2.1 Background and Objective .................................. 6
   2.2 Team and Client ........................................... 6
   2.3 Report Organization ....................................... 6

3 Requirements .................................................. 7
   3.1 Generate Sentiment from Written Narrative Reports ......... 7
   3.2 Web Interface That Allows Upload, Processing, and Output .... 7
   3.3 Enable Feedback Training Results to Model to Improve Accuracy .... 7

4 Design/Implementation ......................................... 8
   4.1 Frontend .................................................. 8
   4.2 Frontend Implementation .................................... 9
   4.3 Backend .................................................. 10
      4.3.1 Overview ............................................ 10
      4.3.2 Deep Dive ........................................... 11
      4.3.3 Hosting .............................................. 12
   4.4 Backend Implementation .................................... 13

5 Testing .................................................................. 15

6 Developer Manual ............................................... 16
   6.1 Frontend .................................................. 16
   6.2 Comprehend ................................................ 17
   6.3 Backend .................................................. 18
   6.4 Starting the Hosted Website ................................ 19
   6.5 Deploying .................................................. 20
      6.5.1 Frontend ............................................ 20
      6.5.2 Backend .............................................. 20

7 User Manual ..................................................... 22

8 Work Completed ................................................ 24
   8.1 Application ................................................ 24
   8.2 Costs ........................................................ 24

9 Lessons Learned ................................................ 26
   9.1 Timeline .................................................. 26
   9.2 Problems and Solutions .................................... 27
   9.3 Future Work ................................................ 28
List of Figures

1. Line graph component showing trends of sentiment .................................................. 8
2. Table component showing specific sentiment entries .................................................. 8
3. The upload dropzone UI .............................................................................................. 9
4. The page for human review .......................................................................................... 9
5. The AWS architecture of our webapp .......................................................................... 10
6. An example subset of a human-review CSV ................................................................ 11
7. An example output of the custom model JSON .............................................................. 12
8. The upload page ........................................................................................................... 22
9. The home page app ...................................................................................................... 23
10. The human review page .............................................................................................. 23
11. Cost table for AWS ..................................................................................................... 24
1 Executive Summary

In the workplace, especially in a factory setting, there is a risk associated with handling equipment and interacting with other employees. In order to alleviate the potential hazards, a proposed solution is to survey employees, and gauge their reactions and feelings of their workplace environment. Using these descriptions, an ideal solution would be to quantify some sort of “sentiment” to generate a report about the conditions. Then, based on the sentiment, actions can be taken to ameliorate or mitigate the risks.

Our Risk Sentiment Analysis web application is designed to interact with companies to assess this workplace sentiment and interpret the results into a meaningful analysis. The website takes reports from employees and sends it through a pipeline of cleaning and processing to predict a sentiment based on descriptions. The model generates a percentage of sentiment using four categories – positive, negative, neutral, or mixed – for each employee’s description. Following that, users can view statistics based on the reports to determine trends, allowing them to reflect and make actions grounded in the data.

Another goal for this project as proposed by our client, Christian Johnson, was to test a cloud provider’s natural language processing model for sentiment analysis. As there were many options, we decided to go towards AWS, since based on a preliminary analysis, its software seemed to achieve relatively good accuracy. Since AWS was selected, we used an infrastructure that was heavily based on AWS; we used DynamoDB to store individual entries of employee reports, S3 to store CSVs of the reports after sentiment analysis, and AWS Comprehend to predict sentiment from employee descriptions. AWS Comprehend offers a built-in solution (no training required) and a custom model solution (training required), so we decided to try both.

We found that the built-in solution for AWS Comprehend achieved an overall accuracy of about 37.4%. This underperformed more than we wanted, so we looked into the custom sentiment analysis model solution that AWS offers. As part of our custom model solution, we built a pipeline for users to classify employee descriptions, thus providing training data for the model. We found that the custom model solution from AWS Comprehend achieved an overall accuracy of 65.52%. We expect that with more data, it could easily achieve a higher accuracy. In the frontend the custom model required a page of our webapp dedicated to collecting human assessments.

We believe that, if given an accurate model, our webapp can be a meaningful solution to risk management in the workplace. Quantifying employee sentiment can help the employer grasp the safety of each employee’s environment and react accordingly.
2 Introduction

2.1 Background and Objective
For a company, it is essential to assess and mitigate potential risks in the workplace. Furthermore, a hazardous work environment can be costly for the company and draining for the employees working there. Seeking ways of observing and collecting data of the conditions and evaluating them is essential to avoid these risks. A proposed solution is to generate reports from the employees. These reports consist of descriptions of their day including incidents and positive events. With the reports, an employer can analyze and detect the sentiment of the descriptions to see if action need be taken. Our web application proposes a solution to account for increased scale in this approach. Employers have the ability to submit batches of reports to be analyzed for sentiment, and the results are displayed for the user in charts and tables. Further, our web application has the ability to improve the sentiment analysis model by having the user manually determine the sentiment of given descriptions.

2.2 Team and Client
Our team consists of five Computer Science majors: Tom Himler, Eriq Taing, Evan Joaquin, Pranav Sharma, and Akshay Akula. We decided to pursue this project because of our combined interests in web development and AI/ML along with having a client that is interested in using cloud technologies.

Our client, Christian Johnson, is a Safety Health and Environmental Global Director for AstraZeneca. He also pursues interests in AI/ML solutions for assessing risk using cloud services. This project is a prototype for these interests.

Tom is our AWS lead. He is responsible for setting up and connecting our backend to AWS. He also worked to set up routes to connect our frontend to our backend.

Pranav is our data scientist. All of the data from Mr. Johnson went through a process of cleaning that Pranav developed. He also was responsible for analyzing the output from AWS Comprehend.

Eriq is our programming lead. He oversaw the development of the frontend and backend. He also developed the backend routes for the custom AWS Comprehend model.

Akshay is one of our frontend developers. He was responsible for choosing the styling of the frontend, as well as developing key features like the charts and tables.

Evan is our other frontend developer. He wrote the code for uploading data to the backend and the frontend for human review for the custom model.

2.3 Report Organization
The report begins with an introduction to the project, including background and team information. Following that, it details the requirements as posed by our client. Then, the report describes the design and implementations of our frontend and backend, including our reasoning for decisions like the UI and its layout and the architecture of the backend.
3 Requirements

3.1 Generate Sentiment from Written Narrative Reports

This is the cornerstone of our app. The web application needs to be able to handle batches of user reports to be submitted to a sentiment analyzer [13]. A sentiment analysis model results from a type of natural language processing [8]. Natural language processors can use machine learning [7] models that process language and translate it into meaningful data. In our case, sentiment analysis takes textual data and transforms it into percentages that describe whether the text was positive, negative, neutral, or mixed.

As per a request from our client, the natural language processor that quantifies the sentiment from the reports should be one from a cloud service [14]. In the case of our project, we went with AWS [30].

3.2 Web Interface That Allows Upload, Processing, and Output

As those who will utilize the service will presumably not be technologically literate, we cannot assume that they will know how to interact directly with AWS Comprehend [28] to upload data to be processed or assign a model to be retrained through the AWS Console [22]. As such, we need to simplify how the users can interact to perform their needed tasks. The web interface frontend is designed to facilitate easy access to AWS services, only leaving what is most important to the users: data upload and graphical analysis. The processing and reading is performed in the backend without need for the user’s interaction. Thus the users would not have to interact directly with AWS Comprehend to analyze their data.

3.3 Enable Feed Back Training Results to Model to Improve Accuracy

One type of machine learning is supervised learning [9]. This is when the model receives training data used to improve quality. For the built-in sentiment analysis model on AWS Comprehend, we are essentially using it as a black box. We put the data in, and it spits out a corresponding sentiment. While this is convenient, we noticed that the accuracy is not as high as we hoped (it achieved 37.4% accuracy). Another reason we chose AWS was because it offers a custom model solution [17]. With AWS Comprehend custom models, we can create classifiers (positive, negative, neutral, and mixed) and input training data. Using this training data we can improve the model’s accuracy. After implementing the custom model, we achieved a peak of 65.52% accuracy; however, we believe this can be improved with a much larger sample of data.

Now, as a part of the project, there needs to be some sort of feedback loop that takes in new human-assessed data and processes it through the custom model. The human-provided data serves as training data for the custom model, allowing it to develop and become more accurate than the built-in AWS Comprehend model.
4 Design/Implementation

4.1 Frontend

The design of the frontend web application was based on a free React / CoreUI template we selected online. Figures 1 and 2 show the template’s main page, which included card components containing tables, graphs, and other informational cards. These were re-purposed and integrated with the data sent to AWS Comprehend. A line graph was selected to display the sentiment values of the data over time. This allows the user to have a clearer view of the trends in sentiment.

Figure 1: Line graph component showing trends of sentiment

<table>
<thead>
<tr>
<th>Sentiment Data Table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>No.</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

Figure 2: Table component showing specific sentiment entries

The website includes an upload page with a drag and drop style file selector for uploading new data sets for analysis, as shown in Figure 3. The dropzone element allows for multiple files to be uploaded at once. These files are limited to ten megabytes, approximately 10 times the size of the largest file we received from our client as example data.

Finally, a review page was created to allow for human review of small sections of data, as
Figure 3: The upload dropzone UI seen in Figure 4. This data is compiled in the backend into new datasets that can be used to train custom AWS Comprehend Models.

4.2 Frontend Implementation

The frontend of our project was developed using React.js [31] with CoreUI [5] assets. The custom elements on each page made use of several CoreUI HTML components and were built in the React.js framework. Multiple open-source React components were added to polish the functionality of the UI.

In the data selection card, a Date-Time-Picker was used to streamline the selection of start and end dates for data retrieved. Additionally, a custom dropzone was used in the upload file card to allow users to easily select multiple CSV files to send in for analysis.

The review page for human data analysis uses a table to display the content of ten entries followed by a series of buttons to select the proper sentiment value for the given data. There is a button at the bottom of the page that will send the reviewed data to the back-end. Before sending the data, however, the frontend will check to see if each entry has been reviewed and notify the user if they have forgotten any of them.

To communicate with our backend on AWS, we used the Axios [19] npm [18] library to send and receive HTTP requests. These requests included sending raw data for analysis, retrieving analyzed data from AWS DynamoDB [26], and compiling human-reviewed data for future custom AWS Comprehend model training.
4.3 Backend

The main design decisions made for the backend were using AWS Comprehend [28], AWS DynamoDB [26], AWS EC2 [20], and AWS S3 [24]. We’ll first give an overview of how we utilized each service.

4.3.1 Overview

Coming into the project, our client had tested out the Microsoft Azure Text Analytics [1] and Natural Language AI from Google [2]. The results from both were underwhelming, and he was looking for a new solution. After an inspection of the custom model of the previous group working on this project, we decided that a cloud solution would be ideal, specifically AWS Comprehend. AWS Comprehend allows for easy real-time analysis without any previous model but can also allow training of new models using custom classifiers. Another reason AWS was desirable is because one of our team members had previous experience using AWS Comprehend, AWS DynamoDB, AWS EC2, and AWS S3.

In conjunction to AWS Comprehend, we used AWS S3 to store CSV data that was used for our custom Comprehend and our final output of the data. AWS S3 functions as a cloud storage solution for arbitrary files, so CSV data fits well into this category.

The next service from AWS that we used was AWS DynamoDB. It is a noSQL database [16] service offered by Amazon that we used to store records of our outputs. This made it easy for us to serve the data to the user and allow quick and easy querying. We originally tried using AWS RDS [21] as our backend database but we ran into difficulties and quickly found it that it was more expensive.

For hosting, we opted into using the service EC2. EC2 functions as AWS’s solution to server hosting. We ran an instance (a generic server) on EC2 and use it to host our backend. For our frontend we used the static hosting capabilities of S3 [27] to serve our frontend files.

Figure 5 provides a visual representation of the connections between the services.

Figure 5: The AWS architecture of our webapp.
4.3.2 Deep Dive

Now, we’ll describe in more detail how we configured and used each service.

Our original design used the built-in model for AWS Comprehend. This relieved us of configuring and monitoring the training and classification jobs, but did not achieve the accuracy we hoped for. Using the built-in AWS Comprehend model was a simple API call to AWS with a list of the documents (strings to be analyzed). It then returned the strings and their associated sentiment. In our implementation, we used AWS Comprehend batch analysis. This allows for a larger amount of documents to be analyzed at once. While batch analysis did increase the throughput of documents through the model, it had a maximum of 20 per API request, so we needed to make multiple calls to it in order to process all of the data. We found that if we were to do this, we would get over-provisioning errors from AWS. To fix this we had to use small delays to offset the amount of calls to the API. It’s not an ideal solution since CSVs that had 1000+ documents would take minutes. Alongside that, we found the overall accuracy of the built-in AWS Comprehend model was only 37.4%.

To try and resolve the issues presented by the built-in model of AWS Comprehend, we turned to the custom model option it offers. We found the custom model (partially) solved both of our major issues. It can process large CSV files without having to worry about over-provisioning. A downside is that processing the data through the model can take upwards of 10 minutes, but we can do it asynchronously. Alongside that, the custom model achieved peek accuracy of 65.52%. While this is still not up to what we want, it is much greater than the built-in solution. We hope our approach ultimately will be successful, since we achieved 65.52% accuracy when using only 284 training documents.

There are two separate pipelines for the custom model: training and processing data. On our webapp we have a page for human-reviewing data. After sample data is reviewed, it gets sent to the backend, converted to a CSV file, and stored in a bucket dedicated to holding training data for the model. The CSV file contains two columns: the first column holding the sentiment and the second column holding the document the sentiment was derived from. See Figure 6 for an example.

<table>
<thead>
<tr>
<th>Negative</th>
<th>Concern regarding slippery surfaces due to extreme temps. after rain.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>I observed a LO/TO in 1300 area. All safety procedures were followed. Permit was filled out properly and job was completed safely.</td>
</tr>
</tbody>
</table>

Figure 6: An example subset of a human-review CSV.

After receiving 20 CSVs of human-review data, the model retrains itself, creating a new version. Training the model can take about 40 to 60 minutes. Comprehend takes all of the CSVs, compiles them, trains off of 90% of the data set, and tests with the remaining 10%. The testing achieved 65.52% accuracy. Comprehend then outputs a tar file with the confusion matrix describing the model’s performance. The other pipeline involves processing data. The upload page of our webapp allows for users to upload company data to be processed through the model. Assuming the CSV file is properly formatted, it first gets put into an S3 bucket. This is required because the custom model only works with CSVs in an S3 bucket, in contrast to the built-in model just needing an API request with the documents. The CSV data then goes through the custom model to be analyzed; this can take up to 10 minutes.
The model outputs a tar of a JSON file \([6]\) with the sentiments for the documents. As you can see in Figure 7, each line of the CSV file gets a score for the different labels. If you look carefully, each line only gets three labels even though there are four possible options, so to get the last label, we take the complement of the three scores’ probabilities. Once untarred, the output can be paired with the input documents to get the final CSV file with all of the data. We add the records from the CSV to DynamoDB and upload the final CSV file to S3.

Figure 7: An example output of the custom model JSON.

The next service we used was AWS DynamoDB. We used it to serve data to the frontend for display. When using DynamoDB, it is imperative that we set it up properly to avoid unnecessary costs. For example, we set up three secondary indices: organization-date-index, organization-site-index, and organization-id-index. Secondary indices allow for cheaper queries based on a partition key (which is organization in our secondary indices) and a sort key (the second item in their names). If we didn’t use secondary indices, it would be necessary to scan the entire database for the keys we want, which is extremely costly.

The secondary indices, organization-date-index and organization-site-index, are used for the chart and table functionality on the dashboard. Users can search, given an organization and a date range, and DynamoDB will query based on the organization and date range. Thus, a user can provide an organization and a site, and all appropriate records will be returned.

The last secondary index, organization-id-index, is used to provide random entries from the database for the human-review page of the webapp. As part of the training pipeline for the model, the human-review page serves as an interface for the user to improve the model. To avoid bias in the training data, we chose to query random entries from the database. To get the entries, we choose a random organization and two random UUID \([32]\) (which are unique identifiers for each entry in the database) and make a range query with the keys. The query also has a limit of 10 items since the review page requires that. One quirk about this method is that there is a chance that the query does not return anything since no record’s ID may fit the UUID range. To alleviate this, we do up to 50 queries to avoid this case, as this decrease the change we return no entries from the database. In testing, we needed a maximum of 3 queries to get 10 items, so a maximum of 50 queries is more than enough.

4.3.3 Hosting

Two services, AWS S3 and AWS EC2, were used to host our website.

The frontend is hosted on S3. We have a public bucket that serves our frontend using the static hosting feature S3 provides. It can be accessed by [15].
The backend is hosted on an EC2 instance. Currently we use a t2.micro instance since it was created for our client, but in the future more can be done to grant auto-scaling and load balancing. Our instance is only started to access the app, so the frontend will be useless unless the instance is running. The instance has port 8080 open for the backend.

There were several options for hosting the website, but we ended up choosing EC2 since it was the easiest to set up. Fortunately, we had someone experienced in using EC2. Another option included AWS Lambda. That is much more cost effective, as it charges based on the number of operations instead of up-time, but it is harder to set up. There are other external services we could use, but we felt that using an entire AWS suite of services was more elegant.

4.4 Backend Implementation

We used Node.js and Node Package Manage (npm) as a foundation for our backend. The npm packages used on top of Node.js were Express.js, CORS, Multer, jQuery (and jQuery-csv), AWS-SDK, and UUID. Express is used for routing purposes. CORS is used to enable CORS on the backend. Multer is middleware that handles parsing the CSV. jQuery-csv is a Node.js library for CSV parsing. AWS-SDK is Amazon’s Software Developer Kit which is needed for interacting with AWS Comprehend, S3, and DynamoDB. UUID is to create a primary key/unique ID for values in the DynamoDB table.

For the backend, the goal is to support the functionality of uploading CSVs to AWS S3, training and processing data through AWS Comprehend, storing data on AWS DynamoDB, and supporting queries made by the frontend.

The web application is going to be used by a multitude of users, and needs to ensure consistency among them, so the program is not personalized for each user. The CSVs uploaded must contain three columns: Organization, Site, and Description. This is just to identify the Organization and Site to figure out the specific location of the review and then the Description, on which we will run sentiment analysis. The only other columns that will be allowed are date and sub-organization. Date helps keep track of a sentiment over time to aid management’s assessment of risk over time, and sub-organization can be used for enhanced specificity on the location of the review. All other columns will be disregarded, and if the required columns are not there, then it will respond with an error.

Once the CSV file is uploaded and verified, the web application will process it through AWS Comprehend with the “description” column, returning the sentiment of the description. The output returns four categories (positive, negative, neutral, and mixed) each holding a percent value that is added to the CSV file.

Originally, AWS Comprehend was run in batches of 25 records at a time for the built-in model. In order to combat AWS’s rate limits, we had to sleep the program for a second after each batch of 25. For the custom model, we did not have to worry about rate limits, as the custom model natively supported large-scale processing.

Then the data for each record is stored in a NoSQL database, DynamoDB, and the final output is exported as a CSV file and stored in an S3 bucket.

In order to facilitate the charts and tables needed for the frontend, the backend has corresponding querying capabilities. The backend can handle two types of queries: organization and site, or organization and date. Both queries also support filtering based off a
filename. From the frontend, the backend obtains the parameters from the get request and then retrieves them from DynamoDB, using a corresponding secondary index for the table.
5 Testing

Our frontend was tested manually by us, and through review with our client. For each feature, the common use cases were tested, for instance CSV data submission of reports to perform sentiment analysis, visualization graphs of recent data, and data querying or filtering. Communication with the backend was also tested in common cases to make sure the system would be able to serve the client’s needs.

As we established the basic functionality of the frontend and backend of the webapp, we also began to upload our client’s sample data to begin testing the efficacy of the product we intended to use. For initial testing, we performed a manual review where the data was retrieved and sections were given to each of the team members to independently determine what they perceived the sentiment to be. Afterwards, our client checked over our review to make sure that it was up to their standards, making corrections as needed. Afterwards we compared the results, deriving the accuracy of the results from how Comprehend classified the data compared to the manual “gold-standard” classification.

The backend currently supports a custom Comprehend model which needs to be trained and tested. The data is sent to the frontend to be sent for human review. After human review, the data is then automatically characterized as train or test data by AWS Comprehend. 10% of all data is randomly selected and becomes the test data; the rest becomes training data. AWS Comprehend then trains the model using the training data, and then we can test the accuracy of the model against the human verified test data.
6 Developer Manual

6.1 Frontend

Requirements and Libraries: The frontend utilizes the React.js and CoreUI libraries. It is also recommended to install a Git [12] compatible terminal, such as Git Bash, or install Git for Windows. The following commands are for Git Bash on your terminal.

Setup: After installing Node.js (for npm) and Git, in a terminal perform the following command:

```
git clone https://github.com/akshayakula/Risk-Sentiment-Analysis-FrontEnd.git frontend
```

This line will clone the repository to a folder ./frontend located in your current directory. Afterwards you should move into the directory and change the branch to the latest branch, master, and then perform the following commands:

```
cd ./frontend
git checkout master
```

Now that the main branch is loaded, install all the required libraries to run the server for the frontend through the following command:

```
npm install
```

Now the app is set up with its required libraries (modules). Run the frontend server by entering the following:

```
npm start
```

Now, the server should be running on the localhost on port 3000, accessible with the URL http://localhost:3000.

Methodology: The purpose of the frontend server is to act as a middle man to serve websites to the devices and allow communication between the devices and the backend server (i.e., when sending CSV files, requesting data, and sending human reviewed data). Following the ideology of CoreUI and React.js, all components are separated into templates to minimize redundancy for the users. The frontend server should solely interact with the backend server and client browsers and devices. The responsibility of data processing (sentiment analysis), data storage, and actual data querying falls to the backend.

Inventory:

*Inside src folder:*

Folders/files unchanged from the COREUI assets
• assets/
• reusable/
• scss/
• app.text.js
• index.js
• polyfill.js
• setupTests.js
• store.js
• views/
• users/

Folders edited:

• containers/ - Contains JavaScript files of common elements, i.e., header, sidebar and other navigation
• views/charts - Contains JavaScript files of components to assemble a bar and line chart.
• dashboard/ - Contains scripts that pertain to the dashboard, as well as re-rendering the chart/visuals.
• pages/ - Contains pages pertaining to error pages and login/registering.
• review/ - Contains elements pertaining to the review page to send human reviewed data to the model.
• upload - Contains elements pertaining to the upload page for sending data to be processed by Comprehend and included into the database.

6.2 Comprehend

Requirements: In order to interact with the AWS services associated with the webapp (i.e., AWS Comprehend, S3 buckets, and DynamoDB), it is necessary to possess an account with access and edit permission to the services listed above, likely as a child account of the owner AWS account. Creating a dependent account tends to have a username along with password, access key ID, and secret access key provided. It is then required to set up a “credentials” file for AWS. A tutorial can be found at: https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html

Inventory:
s3://custom-model - S3 container that contains the training data.

s3://risk-sentiment-analysis-custom-model-unprocessed-data - S3 container that holds the input and output of processing CSV files through the custom model.

s3://uploaded-risk-csvs - S3 container that holds the data after classification (not utilized afterwards)

CleanedCRData.csv - Cleaned Critical Report Data in .csv format (data points separated by commas and new lines).

CleanedSOData.csv - Cleaned Safety Observations in .csv format.

Both of these files contain a similar format, just organized in a different order. Their categories and data are as follows:

- Organization - Major group
- Sub-organization - Minor group
- Site - Which of the physical locations the report concerns
- Date - Time when the report was made
- Description - The actual response of the worker
- Positive, Negative, Neutral, and Mixed - Confidence value from AWS Comprehend for the class

6.3 Backend

Requirements and Libraries: The backend utilizes the following libraries and modules: Node.js, Express.js, CORS, Multer, jQuery, jQuery-csv, AWS-SDK, and UUID. Despite the large number of libraries, they can all be installed easily with npm. As with the frontend, it is recommended to use a terminal that can utilize Git such as Git Bash.

Setup: The setup for the backend is similar to the frontend but will differ in how to launch. Like the frontend, it is required to clone the repository, change to the latest branch, and use npm to install the other required libraries. Utilize the following commands to perform these actions:

git clone https://github.com/thimler9/Risk-Sentiment-Analysis-Backend.git backend
cd ./backend
git checkout main
npm install

Afterwards, to launch the backend server, utilize the following command:
The server is now launched, listening on port 8080 and accessible through the URL http://localhost:8080. The port can be changed in index.js on line 11, const port = 8080, by changing the number to the desired port.

Methodology: The reason for having a separate server for the backend is to separate responsibility from the frontend, allowing for the backend to be changed to interface with other databases and machine learning models while the frontend can still operate cleanly with little to no change, so long as the two servers follow the same protocols. The backend itself will not serve websites (HTML files) but rather will accept and supply data.

Inventory:

- src/index.js - The main JavaScript file that contains instructions for all of the responsibilities of the server.
  - File upload: when a post request with a file comes in it parses the data and sends it to Comprehend to be analyzed (through comprehendText), sending the received results to S3 and DynamoDB (sending a 200 status back if completed; will send 400 should the .csv file have improper format).
  - Search: Org and site; Returns data related to asked organization and site
  - comprehendText: Helper method: performs Comprehend on the description column of the CSV files
  - uploadToS3: Helper method: uploads the CSV files to S3
  - addResultsColumn: Helper method: adds columns for the classes of Comprehend for recording responses
  - addToDB: Adds item to DynamoDB

6.4 Starting the Hosted Website

To have the website function we need to run the backend.

1. First, log into the AWS console.
2. In the search bar, type EC2 and select it.
3. Go the “Instances” tab on the left and find our backend instance named “Risk-Sentiment-Analysis-Backend”.
4. Select the instance and under the “Instance State” dropdown, select “Start Instance”. This will automatically start the backend.
5. Since the frontend is statically hosted on AWS S3, nothing needs to be done other than going to the link: http://sentiment-analysis-frontend.s3_website-us-east-1.amazonaws.com
6. Once you are done using the webapp, make sure to shut off the EC2 instance. Do this by selecting the instance EC2 and click “Stop instance” under “Instance State”. DO NOT select “Terminate instance”.

6.5 Deploying

Currently deploying the app is not easily done, so we will explain the process of doing so. As discussed in the backend design, we host our frontend using the static hosting capabilities of AWS S3. Alongside that, we use AWS EC2 to host the backend.

6.5.1 Frontend

Updating the frontend is very easy.

1. Get a build of the frontend with all of the assets; run the command

   `npm run build`

   while in the root directory of the frontend. This will create an optimized version of the frontend in a folder labeled “build”.

2. Now, log onto the AWS Console and head to the S3 service.

3. Under the buckets tab select “sentiment-analysis-frontend”. This is where our frontend is held.

4. To update the frontend, delete the contents of the “sentiment-analysis-frontend” bucket and copy all of the contents of the “build” folder to the empty bucket.

5. You are done! Simply go to the link [http://sentiment-analysis-frontend.s3-website-us-east-1.amazonaws.com](http://sentiment-analysis-frontend.s3-website-us-east-1.amazonaws.com) to check the frontend is updated.

6.5.2 Backend

Our backend is hosted in EC2, so updating it requires logging into the server.

1. Before doing anything, download the “dev-sentiment-analysis.pem” file from the shared Google Drive folder, “VT CS 4624 Sentiment Analysis Project Fall ’21”. You should have access to the shared folder as a developer. This file contains access keys needed to connect to the instance.

2. Next, log into the AWS console and go to the EC2 service.

3. Go to the “Instances” section and find our backend server named “Risk-Sentiment-Analysis-Backend”.

4. Select it, and click the “Instance State” dropdown and select “Start Instance”.

20
5. After a couple of seconds, the server will start up. Once it has started, select the instance and click the “Connect” button on the top right.

6. Then, select the tab that says “SSH Client” and copy the command under where it says “Example”.

7. Now, open up a terminal with ssh-ing capabilities in the folder where you saved the “dev-sentiment-analysis.pem” file.

8. Execute the command that you saved to your clipboard. This should log you in.

9. If prompted with anything respond with “Yes”.

10. Now that you’re in the server, go to the “Risk-Sentiment-Analysis-Frontend” folder.

11. From there pull whatever commit you want to have the backend use.

12. Once you are done, make sure to shut off the EC2 instance. Do this by selecting the instance EC2 and click “Stop instance” under “Instance State”. DO NOT select “Terminate instance”.
7 User Manual

This manual will describe the basic usage of our React.js frontend web application.

The platform has four primary capabilities: send, view, review, and query data that is stored on the backend database. For the send portion of our functionality there is a tab on the frontend called upload as seen in Figure 8.

The Upload page has a React.js dropzone that will only accept files of the proper format; in this case that is CSV. This is configurable and adaptable to future needs. It has a button to confirm the upload of a selected CSV file. We also have a button to clear the cache in case you uploaded the wrong file.

![Figure 8: The upload page](image)

The view portion of the web application is on the main dashboard, as shown in Figure 9. A query can be entered on the top bar. After clicking either the “Fetch Data by Site” or “Fetch Data by Date”, the chart will be updated with data from the DynamoDB database. This data will then be displayed on a chart and a table. The chart will provide insights on the data and some total numbers at the bottom, while the table provides specific information detailing every single data point. In addition, the user can also query based off of the filename of a file they have uploaded.
For human reviewing of data, the user can go to the human review page, as seen in Figure 10. This can be accessed by clicking “review” in the top bar. This page loads 10 data points to be manually reviewed by the user. A user will go through each of the 10 descriptions and select what sentiment they believe best reflects them. The user then presses the send button which sends the data only if all 10 have been reviewed. After 20 human review submissions, the model will update to improve accuracy.
8 Work Completed

8.1 Application

Our team has completed a full stack application for robust sentiment analysis on workplace safety data. The application leverages cloud technologies provided by AWS for the sentiment analysis and database capabilities. The project that we completed allows for a user to send, view, and review the sentiment analysis on their site workplace safety data. Our first objective was analyzing the results of the many cloud provider sentiment analysis systems and choosing the best overall. After this step, we cleaned the datasets and looked for insights. One of these led to the decision of removing a Dutch dataset, since Dutch is an unsupported language on AWS Comprehend. Our findings led us to choose AWS Comprehend with the approval of our client. The built-in sentiment analysis provided by the AWS Comprehend service did not yield the results that we were expecting; the overall accuracy was only 37.4%. In light of this setback, we decided to focus on the other aspects of the project such as the frontend human review which should help improve the system using the custom models on AWS Comprehend. Our goal was to set up the infrastructure for this project for future development. We were able to achieve the system using the custom model which can gave us a peak accuracy of 65.52%.

8.2 Costs

A major part of using AWS is keeping track of costs and using the best performing methods to minimize costs. Figure 11 shows the total costs for each of the services we used.

<table>
<thead>
<tr>
<th>Service</th>
<th>October</th>
<th>November</th>
<th>December</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comprehend</td>
<td>$5.75</td>
<td>$2.02</td>
<td>$15.94</td>
<td>$23.71</td>
</tr>
<tr>
<td>DynamoDB</td>
<td>$0.04</td>
<td>$0.03</td>
<td>$0.02</td>
<td>$0.09</td>
</tr>
<tr>
<td>Elastic Compute Cloud</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.19</td>
<td>$0.19</td>
</tr>
<tr>
<td>Relational Database Service</td>
<td>$7.45</td>
<td>$0.18</td>
<td>$0.02</td>
<td>$7.66</td>
</tr>
</tbody>
</table>

Figure 11: Cost table for AWS.

AWS S3 is not depicted, as we did not leave the free tier, which allows up to 100 GB of data transfer per month.

You will notice that AWS Comprehend was the most expensive of the services. In October and most of November, we were using the built-in model offered by Comprehend. In late November and early December, we implemented the custom model option for AWS Comprehend. With the custom model, there are two parts that accrue the most costs: training and detecting sentiment. To train the model, AWS charges $3 an hour which is quite costly in comparison to other AWS services. We trained the model 4 times, leading to $6.40 in costs with an average training time of 32 minutes per model version. The cost for the rest
of December was $9.54, i.e., for classifying all of the data. Classification costs are 5 times more expensive for the custom model than the built-in model. Note that the custom model classification costs that we used fell under asynchronous classification, but if we needed real time synchronous classification, the cost becomes much higher as it charges by the second instead of by the number of entries being classified.

DynamoDB was very cost effective. You can see that in total throughout the project, we only used $0.09. This was a result of using some key functions of the database service that alleviate the costs. For one, we did not use the “scan” function at all during the project. Scanning is a notoriously expensive cost, as it scans through the entire database for each query. Since DynamoDB charges by the number of reads, this can be expensive if not handled properly. Secondly, we utilized “Secondary Indices”. Secondary Indices allow more efficient query processing, by using a partition key and a sort key. By specifying these, DynamoDB can scrap all results that do not fit into the partition key, and filter the remaining based on the sort key.

We learned an important lesson using the RDS service. In contrast to DynamoDB, RDS charges by uptime. This means that the longer you have an RDS server running, the more costs you will obtain. In the early stages of our project, we were deciding on the database service we wanted to use. We first decided to go with RDS, but later changed to DynamoDB. Unfortunately, we left the server for RDS running throughout October, which led to a $7 charge.
9 Lessons Learned

9.1 Timeline

- **September 16** - Data cleaning, AWS set up, and complete presentation 1
  - Finish data cleaning so that it can be used with the AWS Comprehend API
  - Set up AWS account credentials with client and provide access management so developers can use the API easily and securely
  - Meet on 9/12/2021 to prepare for presentation 1, discussing our plans and ideas for the project

- **September 24** - Set up API with AWS Comprehend and start work on frontend
  - Sentiment Analysis on given Incident Reports
  - Get the AWS API key to use for AWS Comprehend
  - Assess the need for Multer to put the CSV files in S3 to be used with AWS Comprehend
  - Output data into a CSV file that is saved on S3 (or somewhere else)
  - Develop the basics of a frontend app to allow people to upload CSV files for data processing
  - Create a basic backend to route the CSV data to the AWS Comprehend API

- **October 1** - Assess performance, report to client, and improve the experience with the frontend
  - Check accuracy on Sentiment Analysis of AWS Comprehend
  - Do analysis on output to determine accuracy for AWS Comprehend
  - Make report to client to show progress and what steps need to be taken
  - Add styling to frontend

- **October 24** - Depending on Client, investigate other cloud providers or custom model and frontend authentication
  - Maintain and update AWS version of Sentiment Analysis
  - Discuss the thought of using a custom model with AWS Comprehend depending on the accuracy of the vanilla AWS Comprehend
  - Discuss with client about the interest of using other cloud platforms

- **November 2** - Presentation 2, start drafting report, statistical analysis
  - Meet on 10/29/2021 to prepare for presentation 2, discussing our plans and ideas for the project
  - Condense analysis of AWS Comprehend and compile into a report
9.2 Problems and Solutions

- Data may skew general results.
  The largest technical challenge to overcome for this project is getting high accuracy on the sentiment analysis (above 90%). This proved to be challenging, as much of the data is missing context (for example, being an answer to a question). Another challenge with the sentiment analysis is the usable reviews are set describing work in an certain environment which a general sentiment analysis solution might not be ready to handle. A possible solution may be filtering the data based on a character minimum threshold or adding the context of the question into the description. We learned that before analyzing and cleaning the data, observing undesirable trends can prove vital.

- Meeting times are difficult when coordinating with many different people.
  One of the initial challenges we ran into was scheduling our weekly meetings. As with any large group, it is very difficult to figure out a time when everyone is available. A major issue was our conflicting schedules and our client being busy during work hours. To find the best meeting time, we used WhenToMeet, a platform that lets everyone fill out their availability and find common openings. In the end, we compromised with 4:45 on Fridays.

- Working with AWS Permissions and Costs.
  While working on Amazon, setting permissions to team members to have access to S3 and other services offered by Amazon was difficult. The solution to this was sending emails to our client with step by step directions to give us permissions. If there was any confusion in the directions we would set up a meeting to clarify and go through it with him. One of the largest drawbacks to using cloud services is the underlying expenses; however, these services are central to our product. To mitigate the risk of incurring unexpected expenditures, we implemented rate limits and notifications. This allows both the client and the team to be informed about excess cloud service usage. The limit prevents us from going over the allotted $65 budget provided by Mr. Johnson. We actually caught an unnecessary expense of Amazon RDS which is a database service hosted on a server. We were exploring what we can use as the DB and we forgot to turn the service off, which cost us $7.

- Managing workflow across many different machines and developers:
  Some of the challenges with any software full-stack project are related to dependencies. Developing code for both the frontend and backend requires communication between two different teams. To alleviate miscommunication, we utilized Git liberally to ensure conflicts were mitigated. However, there were multiple times where our communication led to roadblocks. Miscommunicating outputs of the data from the backend and inputs
of the parameters from the frontend is an example. Furthermore, we used Trello \[33\] to formulate weekly sprints, so we stayed on schedule.

9.3 Future Work

9.3.1 Different Cloud NLP Models

Our client, Christian Johnson, has a desire to try different cloud sentiment analysis models. For our project, we decided to focus on AWS Comprehend. This leaves other cloud service providers like Azure’s “Text Analytics” and Google Cloud’s sentiment analysis. Future development on this project should try to get a breadth of sentiment analysis service’s offered. Christian Johnson wants to see the performance of each of the services and compare and contrast them.

9.3.2 Text to Speech

A stretch goal for our project that we did not achieve was text to speech \[3\]. The idea is to take employee audio to derive sentiment out of that too. Audio provides depth that text cannot give naturally. Our client believes that we can extract not only the text, but also data like inflection, and compute sentiment based on that.

9.3.3 HTTPS

Hosting was the last part of our project that was implemented. As a result, we did not have time to establish the certificates needed to get HTTPS \[4\] for our backend. For our frontend, by default, S3 static hosting automatically gives HTTPS, so we would only need to implement it for the backend.

9.3.4 Automatic Data Reevaluation

A stretch goal of our project was to have a feedback loop that took the data in the database and reran it through the custom model each time it was retrained. Unfortunately, we did not achieve this stretch goal. By doing this, we can keep the user updated with the latest results from the model. One consideration if this was pursued is cost for AWS. In order to retrain the data, we need to reprocess everything in the database. AWS DynamoDB charges based off the number of reads and writes, so as more users put data into the database, the cost will scale too.

9.3.5 Model Accuracy Improvements

With the custom model we achieved a peak accuracy of 65.52%. This was obtained with a sample of 266 documents and tested against a sample of 29 documents. With such a small sample size for training and testing, the variance for the accuracy is going to be high. Ideally, we want to increase the sample size as much as possible, to obtain higher accuracy and decrease the variance. Our professor, Dr. Fox, advised having 1000 samples for each of the different categories (positive, negative, neutral, and mixed). Unfortunately we could not get that many data points in the time span of our project.
9.3.6 Frontend Optimizations

There are some improvements to the frontend that did not make it into the final product. Currently, the graph is rendered anew for every change to the query fields. This can cause slow downs if the currently displayed data is large.
10 Acknowledgements

We want to acknowledge our professor Dr. Edward Fox for guiding us throughout this project, especially enlightening us on his experience with natural language processing and cloud services.

Dr. Edward Fox

We would also like to thank Mr. Christian Johnson for being a flexible and professional client. He was understanding about the challenges that arose and made accommodations for them.

Christian Johnson
11 References


