Allergy Prediction Using Artificial Intelligence

DESIGN DOCUMENT

Team Number: 48
Client: Ashraf Gaffar
Advisors: Ashraf Gaffar, Ashfaq Khokhar
Client Lead: Joseph Trembley
Team Lead: Noah Ross
Minute Taker: Ella Godfrey
Research Lead: Xerxes Tarman
Quality Assurance Lead: Alex Ong
Email: sdmay24-48@iastate.edu
Website: https://sdmay24-48.sd.ece.iastate.edu

Revised: 27/04/2024

Executive Summary

Development Standards & Practices Used

- AI / Machine Learning
- Cloud Development/Deployment
- Version Control via Git
- Clean Code practices and maintainability
- User Interface design with Figma
- Agile Development
- Sprint Planning - Determine next set of prioritized work

Summary of Requirements

- Model predicts allergic reaction accurately
- Model is deployed and available on a cloud platform
- Has clean user interface with wide-scale availability
- Simple to use: clients should be able to easily present data to the model and be returned a result
- Backend can accurately process data from front-end

Applicable Courses from Iowa State University Curriculum

COM S/SE 309 - Software Development Practices: Comprehensive course in learning project management, working in a team, and using Git.

DS 301 - Applied Data Modeling and Predictive Analysis: Teaches many machine learning techniques that can be used to create the model.

COM S 319 - Construction of User Interfaces: Introduction to developing front-end user interfaces.

SE 422X - Cloud Computing Software Development: Introduction to cloud computing services and how to deploy applications using them.

New Skills/Knowledge acquired that was not taught in courses

Working in a professional work environment that teaches agile workflow is beyond what formal education usually covers. We also needed to learn about AI as most of our group has minimal experience with AI development.

Table of Contents

Table of Figures

Table of Tables

1. Team

1.1. Team Members

Ella Godfrey, Joseph Trembley, Noah Ross, Xerxes Tarman, Alex Ong

1.2. Required Skill Sets for Your Project

AI modeling: understanding of how to begin building out and testing the model

Front-end GUI: creating a user interface to interact with our model

Backend: sending and processing data from front-end, integrating model into function

Communication (within a team and with a client): ensures high intra-team collaboration and understanding of requirements from client and advisor

Cloud computing: deploying front-end, backend, and model

1.3. Skill Sets covered by the Team

Ella has experience with front-end development working for a company that focused on web development.

Joseph is working on a minor in data science and has a basic understanding of machine learning/AI modeling. He also has 6 summers of interning with a Java development team working with backend applications and Amazon Web Services.

Noah Ross has 2 summers of experience developing User Interfaces on construction/agriculture products. His strengths include human/product interactions and defining customer wants/needs.

Xerxes has two summer internships working as an embedded software developer. He has experience collaborating and working on an agile team.

Alex has two summer internships with one carrying into the school year part time. Between the two internships, he contributed to both front-end and backend projects, giving a range of full-stack industry experience. He has also built personal and ecommerce full-stack websites as well.

1.4. Project Management Roles

Noah: Team Organization, covers creating team meetings and anything to do team wise

Joseph: Client Organization, sets meetings with the client and handles all emails between the team and the client.

Ella: Minutes Taker, Records the minutes in a shared google doc.

Xerxes: Research, determines best technologies to complete the project

Alex: Quality Assurance, ensures all work meets requirements

2. Introduction and Background

2.1. Problem Statement

The goal of this project is to predict whether a patient would have an allergic reaction to a medicine. It will use machine learning to make a decision based on factors about the individual along with the medicine itself. This will allow for faster and easier tests, as the model will have a rapid response time, and no additional patient information is required to run the model. In addition, we built the system on Google Cloud Platform and Amazon Web Services to see if one cloud provider has an advantage over the other.

2.2. Intended Users and Uses

Some of our intended users are medical professionals, who could see if a patient might have an allergic reaction to a medicine; patients themselves, who can enter their information and determine if one of their medicines is causing allergic reactions; and medicine manufacturers, who can find whether their upcoming medicine could cause widespread allergic reactions.

2.3. Related Products and Literature

Allergen testing is already done, but it is mostly done on test animals and in some cases humans. Monitoring skin pricks or digestive reactions to new products in animals like guinea pigs and humans gives us most of our current insight into allergic reactions. Not much exists in predicting allergic reactions like we are trying to do.

It is only in recent years where the idea of using artificial intelligence to predict any sort of medical event has become popular. While some of these models exist, none seem to have gained a large-scale adoption in the medical field. An article [1] published in August 2023 states that even though artificial intelligence is not at the point where it can currently be used in the industry, it will approach that point and is an overall benefit to healthcare.

We are not knowingly following in the footsteps of any other projects or programs as Machine Learning is pretty new, and has yet to be widely adopted in the medical field.

The article [1] states that the biggest hurdles to overcome with creating an AI model that could be used in the healthcare industry are the wide scale data requirements, lack of ability to see how the model processes data, and difficulty in validating the model. In order for a machine learning model to begin predicting data correctly, it will need a significant amount of data.

3. Revised Design

3.1. Requirements & Constraints

Backend Requirements:

- Backend can send data from front-end to model
- Backend validates data and ensures it follows correct formatting
- Backend can return results to front-end
- Predicts whether a patient would have an allergic reaction to a medicine
- Able to process large number of input variables

UI/front-end Requirements:

- Clear display of prediction and confidence level
- Location for user to upload information to test the model
- The remaining UI should be visibly appealing to the user
- Web accessibility from anywhere

Legal Requirements:

- Data usage does not violate any health privacy laws

Testing Requirements:

- The model should be tested for an overall accuracy percentage to report
- Each component has multiple iterations of tests for any type of error
- System has tests covering entire scope of the project
- Logs should be implemented to catch faults or errors

Maintainability Requirements:

- File structure should be clear
- Code should be well documented

3.2. Design Complexity

Our final design consists of two major components: the front-end and the backend. The front-end will be the interface seen directly by the user and requires a careful design to ensure users can navigate it properly. The backend will be built using cloud functions to call off to the model and return the result to the front-end. The model, as part of the backend, will create a prediction for whether the patient has an allergy based on the input data received. Some of the applicable principles are keeping the model secure, as well as creating a transparent view of how data is used. Developing an ML model for our project presents a significant challenge as we strive to maintain high accuracy. Since the model will be trained with a large dataset with many varying input types, there may be numerous iterations before one begins to work as intended. Additionally, it is crucial to create a robust testing suite to safeguard against overfitting the model. Since the model will be hosted on a cloud-based platform, we will need to ensure that it can be accessed by anyone without compromising the integrity of the system to follow artificial intelligence engineering principles.

3.3. Modern Engineering Tools

- Model Creation: Keras API
  - Keras serves as a high-level API for the TensorFlow machine learning library, known for its emphasis on providing a fast and user-friendly interface for creating production-ready ML models. We used this library to build out a model to test.
- Model Hosting: Cloud Providers
  - Since we needed to evaluate the performance of our model between Google Cloud Platform and Amazon Web Services, we looked into both options for hosting the different components to the web application. While they are different, they both provide options to host our model and site, so our design was implemented in both and compared.
- Model Workbook: Jupyter Notebook
  - Jupyter Notebook is a browser-based tool for developing and presenting data science projects. It is specifically designed to visualize and execute Python code, which is useful in developing machine learning models. Our group used Jupyter Notebook as our primary development tool for our machine learning model. This saved us from having to use overly complex systems to run the model training code.
- Backend framework: Serverless functions
  - Serverless functions create small endpoints that can be used to set up an infrequently accessed application. It is a great cost saver as the costs are based on the time it takes to compute the calculations as opposed to a continually running instance. Since this application should not be high-traffic, this was the optimal method of running the backend.
- Front-end framework: React.js
  - React.js is a front-end JavaScript framework designed for building responsive single-page web applications. We will be using this library to develop the user interface for inputting data and displaying results from our ML model.

3.4. Design Decisions

One of the major design decisions we needed to make in the project was choosing whether to use Amazon Web Services or Google Cloud to use computing power and host our machine learning model. To make this decision, we had to carefully compare the features offered by both such as pricing, ease of use, and functional capabilities. After comparing them both by seeing available resources and using our own experiences with them, we elected to choose Amazon Web Services as our primary cloud platform for the project.

Another major decision we made was choosing which language to build our front-end with. We chose to use React, since many of us have experience working with the language, and its flexibility will be a great benefit to us as we implement our design.

The last major decision for the project was choosing how to create our backend. After some thoughtful discussion, we decided that we wanted to use a cloud function which would call our model and return the data back to the front-end. This method requires less coding overall, and it allows us to build the backend using the same cloud platform as the machine learning model uses.

3.5. Design

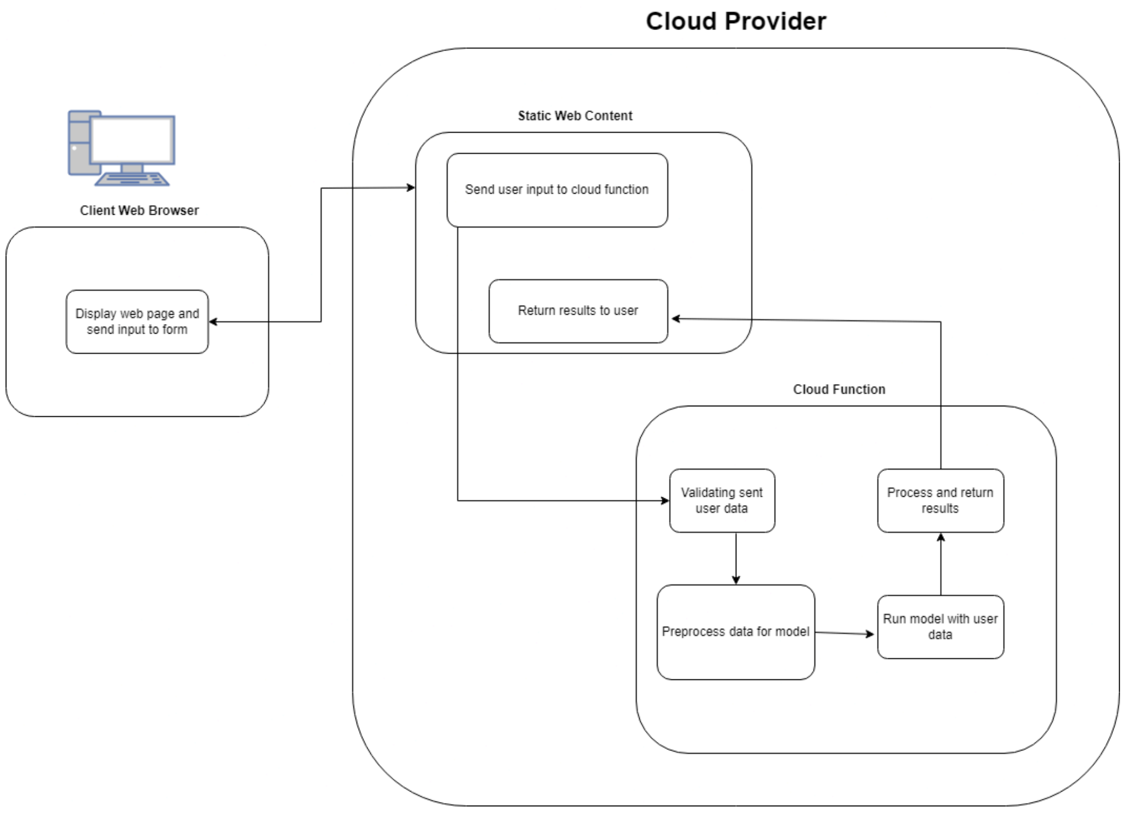

Cloud Provider
Static Web Content
Send user input to cloud function
Return results to user
Client Web Browser
Display web page and send input to form
Cloud Function
Validating sent user data
Process and return results
Preprocess data for model
Run model with user data

Figure 1: Final Design of Our Project With Respect to Both Cloud Providers

Front-End

The user interface of this design is a desktop web application. The user interface is responsible for presenting a user-friendly interface for entering patient information along with relevant predictors. Additionally, it facilitates the input and listing of skin conditions to be sent to the allergy prediction model and returns the results to the user.

Backend

The back end handles all the operations of the program. Our backend architecture uses a singular cloud function to handle validation and preprocessing of data, running the model prediction using the data, and returning the results to the front-end. This is a significant change from our last design, as we originally separated the model from the rest of the backend.

3.6. Engineering Standards

Since our project would take place in the medical field we should follow the IEC 62304 standard [2]. The IEC 62304 standard [2] specifies the life cycle for medical software devices such as risk management, software requirement analysis, software system testing, etc.

IEEE 11073 [3] is the "Health informatics - Point-of-care medical device communication," this provides a framework for compatibility between various medical devices and systems. It [3] defines communication protocols and data formats for exchanging information between medical devices and healthcare information systems, such as electronic health records or AI systems. When developing an AI model for allergy prediction, adherence to this standard can facilitate easy integration with other medical devices and systems, ensuring that relevant patient data can be efficiently shared and utilized for accurate predictions and informed healthcare decisions.

IEEE 2801 [4] is a standard built around the management of data in medical artificial intelligence. It [4] places a high emphasis on how data should be used and controlled once it has been gathered. Following this will help us ensure that we are using data in an ethical manner.

The model will be built using Python, because this language supports a wide variety of libraries, including ones with a focus on machine learning. One of the key libraries we will use is Keras, which is used to create a neural network system to train models.

3.7. Technology Considerations

By automating the technical aspects typically handled by a medical professional, our project has the potential to significantly decrease costs associated with drug prescription and development. However, a major challenge we face is our heavy reliance on obtaining high-quality data for our predictions.

One drawback associated with employing a machine learning model for allergic prediction is its reliance on the quality of the training data. The model may encounter challenges with chemical components that are not well-represented in our training dataset.

3.8. Design Evolution

As we continued to build out the application, we noticed that there were many aspects in our original design that needed to be changed:

- Building our app on both cloud providers: when we reviewed the requirements of our project, we realized that one of the aspects was to build a comparison between GCP and AWS, meaning that we needed our design to work for both, even if the implementation may be slightly different.
- Removal of SageMaker from AWS design: as we started working with SageMaker in our project, we realized that it was overly complex for what we needed. We did some research and found that most of what we use would work better as an addition to the cloud function instead of separating them out.
- Removal of database: when we evaluated our requirements with those of engineering standards, we decided that the database should not be included. Since we had planned on putting the data that gets sent to our function there, there was the possibility of a HIPAA violation. We determined it would be safer not to include that.
- Transference of front-end to cloud provider: in our original design, we intended to communicate that the front-end is also a part of the cloud provider, but the schematic had it listed on the client side.

4. Implementation Details

4.1. Detailed Design

Front-end

The front-end has a relatively simple design and tech stack. The client side portion of our application was built using React JS, with styling assured by the CSS framework, Tailwind CSS. The client is responsible for gathering patient information, which is bundled into a JSON payload to fit the schema expected by our model comprising Gender, Birth Year, Fitzpatrick, Skin Tone, and array of Skin Conditions. This payload is then sent off to the cloud functions to input into our model, and then the client will display the prediction message contained within the response body.

For hosting, we use two cloud providers, AWS and GCP. For both, we have elicited to store the React application file as static content within the cloud object storage of both providers. On the AWS side, we store the react app file within a S3 bucket, which provides its own url to the app. Using this url, we can access the entire application relatively easily.

For GCP, it is slightly more complicated as we need to provide a .yaml file in order to run the application via the object storage url. The .yaml file allows us to create an application via the object from Google's Cloud Storage.

Backend

The first step of making the backend was building out the model in Python. To do this, we had to encode our training data so that the values were numbers that could be taken in by a neural network. Most of our data was divided into two sections, the smaller categories and the larger ones. The smaller categories used a process called one-hot encoding, where each possible value was given a separate position, and used a 1 to denote the presence of that value, backfilling the others with a 0. This is a common process, as it ensures that each value is uniquely stored and does not affect the other parts of that category. The larger categories were processed through text vectorization, where each individual value is given a numerical key. While not as efficient as one-hot encoding, the vectorization allowed for multiple values in certain categories, which was needed for the skin conditions.

After we made the data conversion, we had to build the model. Through many months of refining and tweaking, we were able leverage Keras libraries to build a model that would take the data in two sets, one where the order mattered, the one-hot encoded values, and one where the order did not matter, the text vectorized values. Each was passed through a few hidden layers before merging and outputting its predictions. These predictions were given a threshold and put through a converter which transformed the binary values to their numeric counterparts. Due to data and time constraints, we were only able to get predictions for the 10 most common allergies in the data set in our model.

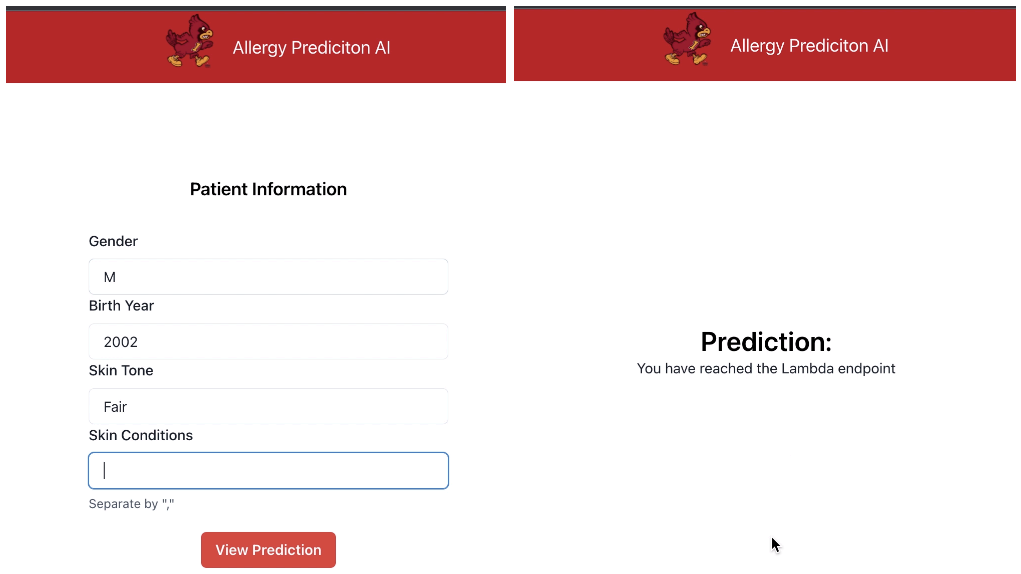
When our model was complete, we exported it and the objects we used to encode the inputs and outputs to a separate Python file. In that file, we added a method to preprocess incoming data, pass the transformed data into the model, and return the result. To run this on the cloud platforms, we packaged our source code along with certain dependencies in a Docker image and uploaded it. From there, we build a cloud function on both GCP and AWS, citing our image as the source code that should be used whenever the endpoint is reached.

4.2. Description of Functionality

Front-end

The simple React client takes in basic patient information that the AI model expects. These fields contain the Gender, Birth Year, Skin Tone, Fitzpatrick, and list of Skin Conditions. Once the user wishes to view the prediction, the information is bundled into a JSON payload, and sent off to the cloud function via POST request. The response will be a prediction from the AI model which is in the form of Ingredient ids that may act as potential allergens for the patient. This could either be empty or 1 to many different allergen ids. As mentioned previously, the prediction is displayed as ingredient ids due to a limitation of the provided data set. The ingredients were provided as a id only, with no correlation or relation to any labels or names. With more data and labeled products or ingredients, we could be more direct with the allergens.

The figures below showcase the initial design and final iteration of the user interface.



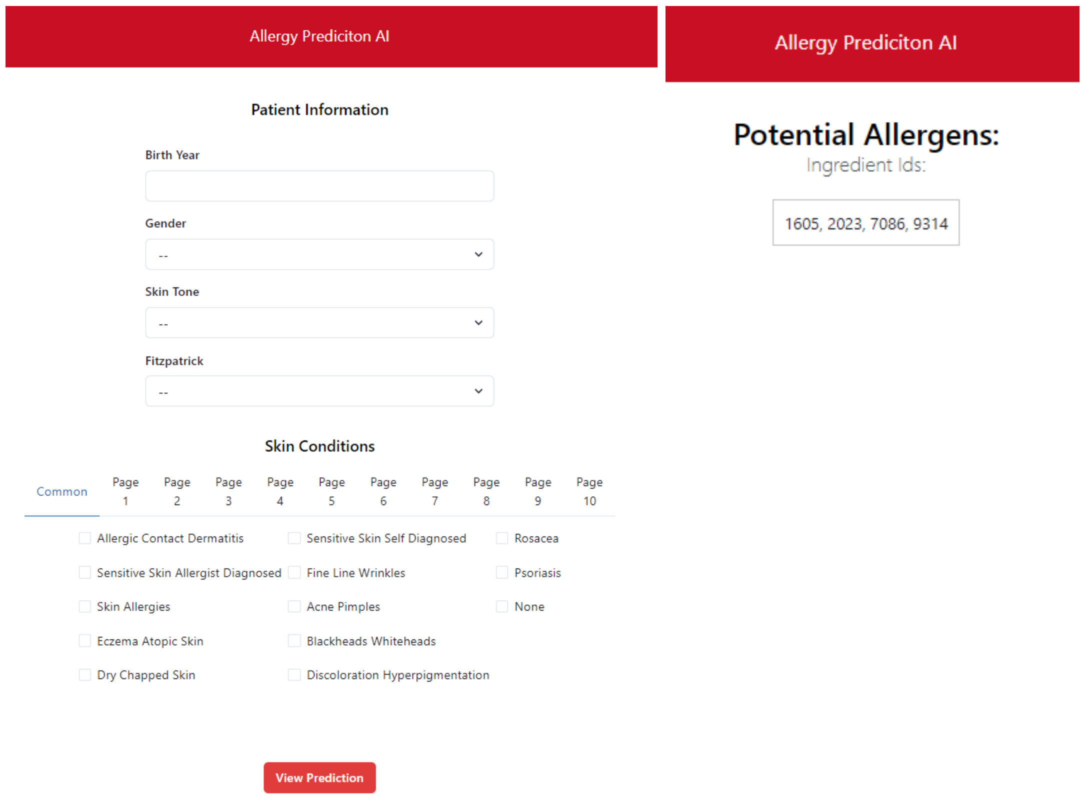
M

2002

Fair

You have reached the Lambda endpoint

Figure 2: Initial iteration of the client, simple text fields with an established response from the cloud functions.


Allergy Prediciton AI

Patient Information

Birth Year

Gender

--

Skin Tone

--

Fitzpatrick

--

Skin Conditions

Common | Page 1 | Page 2 | Page 3 | Page 4 | Page 5 | Page 6 | Page 7 | Page 8 | Page 9 | Page 10

Allergic Contact Dermatitis	Sensitive Skin Self Diagnosed	Rosacea
Sensitive Skin Allergist Diagnosed	Fine Line Wrinkles	Psoriasis
Skin Allergies	Acne Pimples	None
Eczema Atopic Skin	Blackheads Whiteheads	
Dry Chapped Skin	Discoloration Hyperpigmentation	

View Prediction

Allergy Prediciton AI

Potential Allergens:

Ingredient Ids:

1605, 2023, 7086, 9314

Figure 3: Final iteration of the client. Restricted input fields and multi-selectable skin conditions. Upon submission, a response from the backend and model is displayed

Backend

The backend endpoint is a simple function that receives request data from the front-end. After getting the data, it uses various methods of encoding the data as numbers such as one-hot encoding and text vectorization. This is necessary so that the data format matches what the model was trained off of. After the encoding, the data is passed through a neural network, which gives a prediction as to what allergies the patient has. These values are returned as a response back to the front-end.

4.3. Notes on Implementation

Our implementation strategy focuses on ensuring data integrity, system reliability, and compliance with regulatory standards. Robust data validation mechanisms have been implemented on both the front end and backend to maintain the integrity of patient information and ensure adherence to the required data format for the model input. Error handling mechanisms are put in place to gracefully manage any issues that arise, with comprehensive logging to capture faults or anomalies for effective troubleshooting. Security measures include encryption during data transmission and compliance with health privacy laws to safeguard patient information. Scalability considerations guided the design to accommodate large volumes of input variables efficiently, with load balancing and horizontal scaling options to handle increased demand. A thorough testing strategy was adopted, covering unit tests, integration tests, and end-to-end tests to ensure system reliability and accuracy.

Documentation has been maintained to facilitate ease of maintenance and future enhancements, following engineering standards to ensure compliance with medical device regulations and ethical data usage practices.

5. Testing

5.1. Unit Testing

AI Model

To test the model, we created a split of our data between training and testing, and used several k-folds (splits of the data using a different iteration of training and testing) to get good baselines for the performance. When we tested the AI model, we found that accuracy was not a good baseline, as we had many different results that could be returned. Instead, we focused on different metrics like precision (the rate of true positives over marked positives), recall (true positives over all actual positives) and F1 score (measure of precision and recall together). With these values, we were able to see the actual behavior of our model. For example, an extremely high precision and low recall indicated that the model was predicting almost no allergens, and the inverse indicated that it was outputting a large number of allergens every time. Because they are looking for almost opposite parts, creating a balance between these two metrics is a common problem in machine learning.

Individual Cloud Functions

On both AWS and GCP, we created multiple test cases to ensure it worked as intended. In both iterations of the backend, we were able to use Postman to send requests and receive an accurate response from each. For AWS specifically, there is a built-in method of using a test case directly from the console using an input string. In our iterations for both of our cloud functions, we have the functions working as intended.

Front-end

For our front end, we manually tested the inputs and ensured they did what they intended. The React component itself needed little testing, as the changes show up visually, and the project won't compile with errors. We created a Postman endpoint to test the request functionality for the Javascript JSON request from the React component to the backend. Once we communicated with a Postman endpoint, we created a test AWS Lambda function to receive and send a response back that the front end could display. This allowed us to test the ability of the front end to send and receive from the model without actually going through the model. Once we established the model worked on its own and the front end worked on its own, we changed the React component JSON request to send the request to the model instead of the test Lambda function.

5.2. Experimentation with Random Forests

When analyzing the data, there was an extreme class imbalance with the 731 unique ingredients. The histogram below shows the 300 most common ingredients in the dataset.

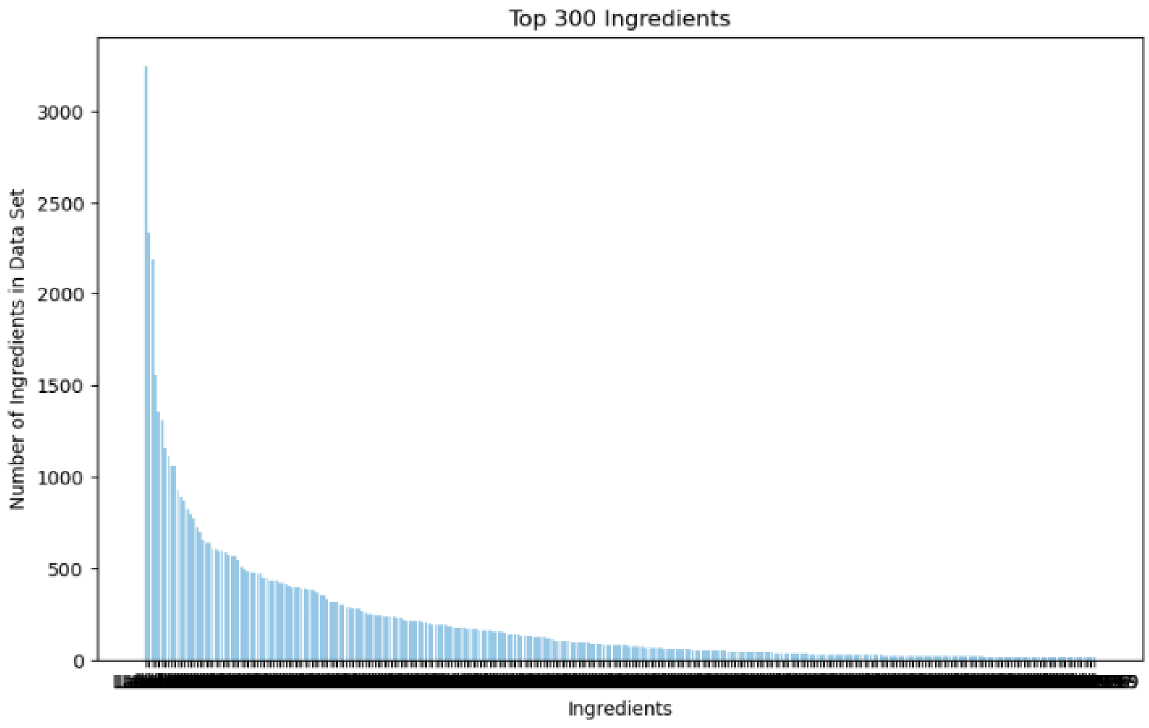


Figure 4: Histogram of the frequency of ingredients in our dataset.

The graph illustrates a clear imbalance with ingredients that appear, so we established a cutoff for the number of ingredients we can predict given there is not enough data to proceed. After further research and comparing multiple options, we decided that around 9 ingredients was the best trade-off between the number of ingredients predicted, the F1 score for each ingredient, and the time required to train and predict.

Current Results with Random Forest

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.36	0.52	0.42	664
1	0.28	0.58	0.38	456
2	0.25	0.56	0.34	426
3	0.19	0.54	0.29	328
4	0.14	0.53	0.23	249

5	0.14	0.47	0.22	276
6	0.11	0.44	0.18	225
7	0.15	0.56	0.23	235
8	0.12	0.54	0.20	218
Micro Avg	0.20	0.53	0.29	3077
Macro Avg	0.19	0.53	0.28	3077
Weighted Avg	0.23	0.53	0.31	3077
Samples Avg	0.19	0.40	0.23	3077

Table 1: Testing results for the Random Forest

Overall, based on experimentation with Random Forests, this technique offers a good baseline model in addition to deep neural networks. Given our testing, if there is a sufficient sample of a particular ingredient, our model is able to predict an individual's allergy to it fairly.

5.3. Integration Testing

Once we connected our front-end to the backend, we tested how the system worked together. With teams working from both sides of the path, we made sure that the front-end was correctly sending requests and that the backend was receiving and processing them. There were minor differences in the cloud platforms due to how their cloud platforms work, but both are able to connect to each other.

5.4. System Testing

After the integration testing, we tested the system as a whole manually. We created a test plan based on the expected functionality on both systems. On the front-end of the system, we used end-to-end testing to test user interactions. For the backend we used Postman to validate the communication between the front-end and backend.

In our development, we built out new components and tested them with the old components before merging them to main to ensure functionality worked as intended at every stage.

On the ML side, since our model underwent multiple training iterations on the same data, we aimed to prevent overfitting by employing a subset of the data for testing. This allowed us to assess whether the model is overfitting or not. Our strategy to prevent overfitting was a notebook with k-fold cross-validation. This enabled automatic assessment of whether the model is improving or succumbing to overfitting for our dataset. Moreover, this method was useful in the beginning of our project for selecting the ML approach that best suits our data.

5.5. Acceptance Testing

After we built out our project, we compared the requirements of the project with our test results. While there were some areas that did not exactly match, such as directly using accuracy for our model, we determined that we had met the requirements with our work.

Because we had to change how we measured what constituted a good model, we needed to create a new baseline. We evaluated the performance of the model by comparing it with a support vector machine, an algorithm designed to separate between the different classifications we had. The process is not how predictions are usually done in the medical field, but it has given us an idea of what evaluation parameters demonstrate acceptable performance from our model. In that case, we needed our precision to be greater than 0.25, our recall greater than 0.13, and our F1-score to be greater than 0.17 to have a good model. Our final iteration of the model beats all of those metrics by a clear margin, we had a precision score of 0.35, a recall score of 0.86, and an F1-score of 0.65.

Model Type	Precision	Recall	F1-Score
Random Forest	0.20	0.53	0.29
Neural Network	0.35	0.86	0.65

Table 2: Comparison between our baseline SVM Model and our final Neural Network.

6. Broader Contexts

6.1. Global

With our project relying heavily on AI and sensitive patient information, we need to ensure that data is secure and that our design follows HIPAA regulations as this can impact the Global, Cultural, and Social areas. With the data being run through a third party library with Keras to create and test the model, we made sure to verify the integrity of the program. In addition, we did not put the data in a place where it could be compromised and removed any permanent data collection from the design.

6.2. Economic

If our product is used by real medical professionals and facilities, an beneficial economic impact could be the resources, time, and overall cost saved by being able to predict patient allergies faster and more accurately. This would allow doctors to spend less time having to come up with the diagnosis and reduce the chance of inaccurate diagnoses.

For our organization, some of the economic impacts to consider for us would be the third party sources we use and if we need to cover any costs for the operations if they were to exceed the free plans. Some examples are that cloud services such as AWS and Google Cloud support free services up to a certain amount of bandwidth / requests. The same could be said for the Keras libraries and is something we may have to consider if our design scales. The majority of the costs we incurred were small charges that can be easily covered by users.

The model prevents the need for expensive allergy testing kits. Both cloud services provide free tiers which should suit all of our request needs during development.

6.3. Environmental

There was not much to consider outside of the energy and resources we need to consume to run our servers and AI model. Since most of our services will be through the Cloud, the main environmental considerations would be how much resources and energy is required to run our host machines. One way that we have minimized the impact is restricting computing power and energy consumption during the creation of the model.

6.4. Societal

Our project used AI in the cloud for allergy prediction. The main societal areas affected by our design will be the Public health, safety, and welfare area. By creating a tool that allows medical professionals to input patient data and symptoms, the goal of our design was to be able to predict specific allergies a patient may have. This can help provide faster and more accurate results, while reducing human error. This can lead to better treatment that benefits both the patient and medical professional.

7. Conclusion

7.1. Progress

We were able to complete our major objectives. As stated earlier in the testing, not only did we make the model, but it had great metrics with a precision of 35% and a recall of 86%. Despite this success, we were not able to get all of the possible allergens in our model, as we did not have the time to make the model complex for all of them while maintaining our high metrics. In addition, some of the allergies appeared much less frequently than the subset we selected, making it difficult for the model to make its predictions.

We were able to run the service both on GCP and AWS, as well as evaluate the differences between the two. Below is a comparison of the services based on some important metrics. The cost is the approximate cost for how many times we have called the function, and the memory usage is the approximate amount of memory it uses per request. Because cloud functions are provisioned as-needed, we have two separate times listed for completion. The cold start time is a request that is the first when the service is running, or the first if it has not been run in a while. The warm start time is the time it completes after the service has been started, which is often much faster.

Cloud Provider	Cost	Memory Usage	Cold Start Time	Warm Start Time
GCP	$0.02	> 512MB	11 seconds	.6 seconds
AWS	$0.00	700 MB	10 seconds	.4 seconds

Table 3: Comparison between GCP and AWS as cloud providers

Based on the data above, we are choosing GCP as the better option for our use case. It is simpler to use and provides greater transparency to the services being used and their costs. In addition, the times to get a response were not that much different from each other, and the costs were very similar.

7.2. Value of our Design

While our model does not fully capture all of the allergens like we intended at the beginning, it shows what is possible. Our high precision and recall scores prove that using artificial intelligence is a good way to make allergy predictions without needing the invasive technology. It is not a perfect replacement, but it brings medical professionals closer to seeing the connections causing allergic reactions before an individual has one.

7.3. Future Steps for Development and Providing Value

The next step for the project is finding a way to integrate the other allergens back in. Because of our time constraints, we were not able to get a model that would consistently get us accurate results. If we had more time, we would go back and bring those values in so that our model would be able to make better decisions. Once our model was able to predict any of the allergens, we would spend more time refining it. What we have works, but it does not capture all of the allergens like we would want it to.

7.4. References

[1] M. van Breugel et al. “Current state and prospects of artificial intelligence in allergy”. *Allergy*. 2023; 78: 2623-2643. https://doi.org/10.1111/all.15849.

[2] *Medical device software — Software life cycle processes*, IEC Standard 62304, 2006.

[3] *Health informatics–Device interoperability–Part 20601: Personal health device communication–Application profile –Optimized exchange protocol*, IEEE Standard 11073, EMB/11073 - IEEE 11073 Standards Committee, 2022.

[4] *IEEE Recommended Practice for the Quality Management of Datasets for Medical Artificial Intelligence*, IEEE Standard 2801, EMB/Stds Com - Standards Committee, 2022.

8. Appendices

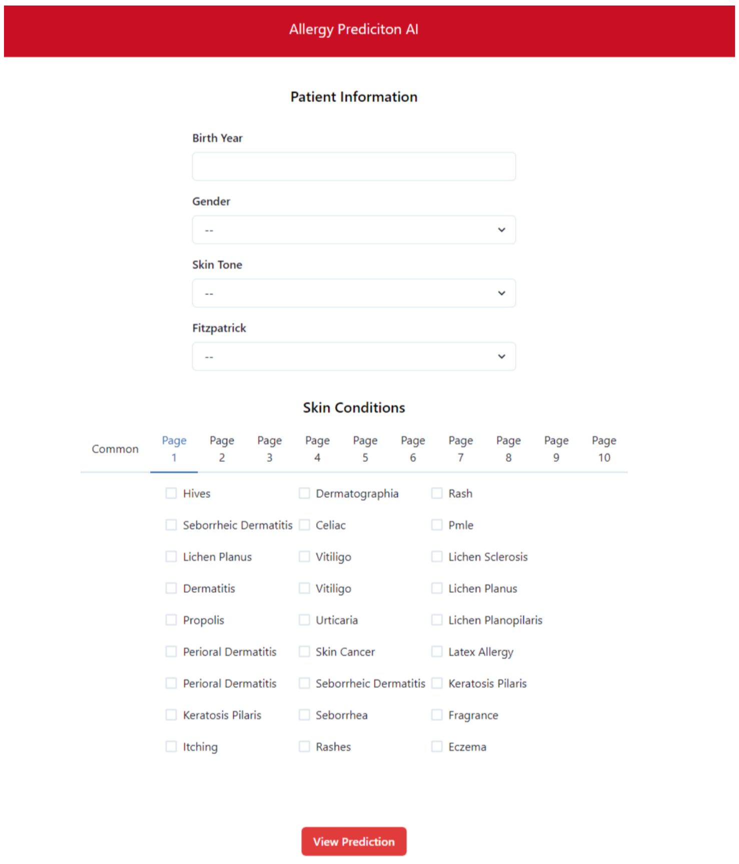
8.1. Appendix 1 - Operation Manual

Since our application is a web-based app, there is no setup required. To test and demo the application for yourself, follow the instructions below.

1. Follow the link to one of our applications.
   a. App on AWS: http://seniordesignteam48.s3-website.us-east-2.amazonaws.com/
   b. App on GCP: https://team-48-402621.uc.r.appspot.com/

    c. If neither of these links are live when you attempt this, view the screenshots in the next steps

2. Enter your birth year, gender, skin tone, fitzpatrick skin phototype, and any skin conditions to the best of your ability. Note that there is no account or linking of any personal information to you. This is an intended advantage of our application that it only takes non-personally-identifiable information in its predictions.



Allergy Prediciton AI

Patient Information

Birth Year

Gender
--

Skin Tone
--

Fitzpatrick
--

Skin Conditions

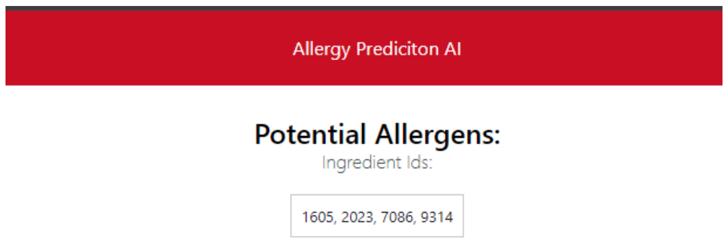
Common | Page 1 | Page 2 | Page 3 | Page 4 | Page 5 | Page 6 | Page 7 | Page 8 | Page 9 | Page 10

☐ Hives	☐ Dermatographia	☐ Rash
☐ Seborrheic Dermatitis	☐ Celiac	☐ Pmle
☐ Lichen Planus	☐ Vitiligo	☐ Lichen Sclerosis
☐ Dermatitis	☐ Vitiligo	☐ Lichen Planus
☐ Propolis	☐ Urticaria	☐ Lichen Planopilaris
☐ Perioral Dermatitis	☐ Skin Cancer	☐ Latex Allergy
☐ Perioral Dermatitis	☐ Seborrheic Dermatitis	☐ Keratosis Pilaris
☐ Keratosis Pilaris	☐ Seborrhea	☐ Fragrance
☐ Itching	☐ Rashes	☐ Eczema

View Prediction

Figure 5: Demonstration of the front-end of the application.

3. Verify your information and click Submit.
4. Wait for the results to return. When they return, the results will appear in a small box like shown below.


Allergy Prediciton AI

Potential Allergens:

Ingredient Ids:

1605, 2023, 7086, 9314

Figure 6: Demonstration of a successful prediction

8.2. Appendix 2 - Alternative/Initial version of the design

Initial Design

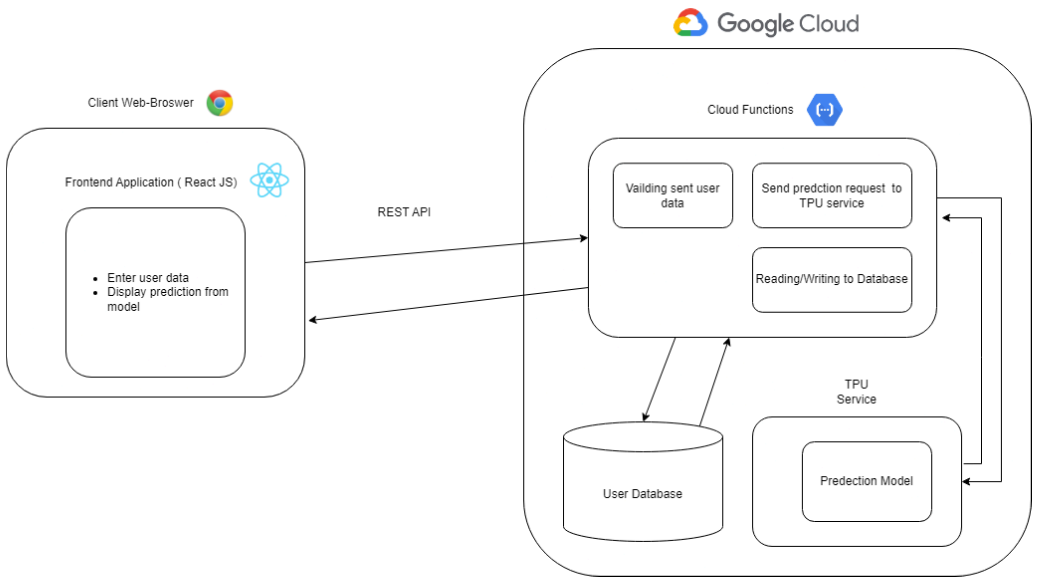


Figure 7: Initial design of the project with the React front-end, a backend utilizing cloud functions, and the prediction model.

The first portion of the design is the React front-end. It allows the user to enter the data and display the prediction. This component does not need much itself but needs to connect to the backend, which is possible with the given design. Since both the cloud function and model will be using the same cloud platform for their operations, they are placed under the same category. We used Google Cloud as a cloud provider option to illustrate how the data would flow. Within the cloud platform, we have the cloud functions, the database, and the TPU service, a circuit designed for neural networks.

Functionality

In this current design, the user enters data through the front-end to get a response. The data is passed into the cloud, entering the cloud function, where the data is added to the database and sent to the model. The model, which at this point has already been trained, looks at the data and makes a prediction. The model sends its prediction to the cloud function. The function sends the data back to the front-end and adds the prediction to the database where the data is stored. The front-end displays the prediction to the user.

Reasons for Change

Our initial design did not correctly illustrate how the different components interact within the cloud provider. We had not fleshed out how each of the different components would interact or how we would build them out. We expanded on what these services were when we decided to switch completely to AWS in the second design.

Second Design

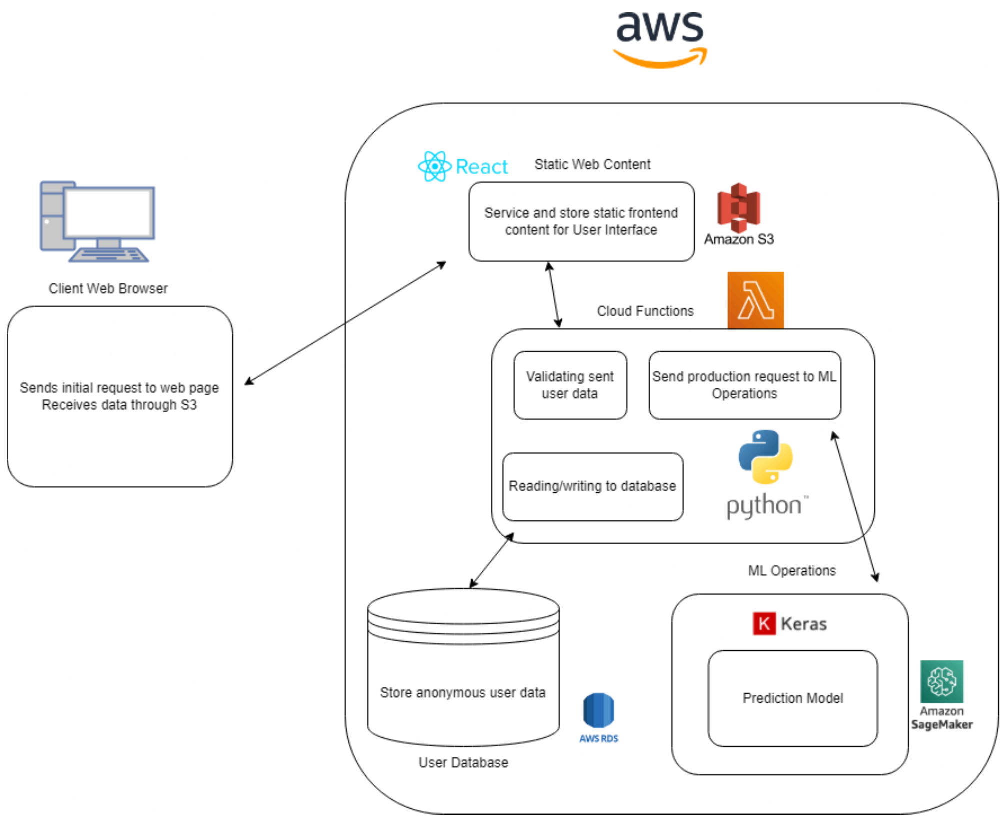

aws
Client Web Browser
Sends initial request to web page
Receives data through S3
React
Static Web Content
Service and store static frontend content for User Interface
Amazon S3
Cloud Functions
Validating sent user data
Send production request to ML Operations
Reading/writing to database
python
ML Operations
Keras
Prediction Model
Amazon SageMaker
Store anonymous user data
AWS RDS
User Database

Figure 8: Second design of the project, including specific services within AWS to be used in the project.

Front-End

The user interface of this design is a desktop web application. The user interface was responsible for presenting a user-friendly interface for entering patient information along with relevant predictors. Additionally, it facilitated the input and listing of skin conditions to be sent to the allergy prediction model. The front-end handled the display of the prediction results generated by the model.

Backend

The back end played a crucial role in managing requests from the user-facing web application. This involved updating a database to store patients' information and making prediction requests. Our backend architecture comprised multiple AWS services. The front-end was stored and delivered through an S3 bucket. The API, managed by AWS Cloud Functions, were responsible for forwarding protection requests to our ML model, authenticating users, validating user input, and handling database queries for storing user and patient information. Additionally, we planned to host our prediction model using AWS SageMaker, ensuring accessibility through our Cloud Function.

Reasons for Change

Once we started developing the project, we realized that some of the components we had would not work as intended. The biggest change was including both GCP and AWS as separate services while building the project out on each. This is because part of the project is evaluating the costs of the cloud and showing the differences, which was not part of the previous designs. In the time we worked with AWS, we found that working with SageMaker proved to be more difficult than intended. Since the backend was already using Python, we made the decision to combine the two into a singular serverless function. We made the same change when we deployed the application to GCP. Our last major change was removing the database and all related operations, because there may have been possible HIPAA violations with storing the test and actual data in places of potential security risk.

8.3. Appendix 3 - Other considerations

Set	Class	Precision	Recall	F1-Score	Support
Set 1	0	0.36	0.52	0.42	664
Set 1	1	0.28	0.58	0.38	456
Set 1	-	Micro Avg	0.32	0.54	0.40
Set 1	-	Macro Avg	0.32	0.55	0.40
Set 1	-	Weighted Avg	0.33	0.54	0.40
Set 1	-	Samples Avg	0.23	0.26	0.23

Set	Class	Precision	Recall	F1-Score	Support
Set 2	0	0.36	0.52	0.42	664
Set 2	1	0.28	0.58	0.38	456
Set 2	2	0.25	0.56	0.34	426
Set 2	3	0.19	0.54	0.29	328
Set 2	-	Micro Avg	0.27	0.55	0.36
Set 2	-	Macro Avg	0.27	0.55	0.36
Set 2	-	Weighted Avg	0.29	0.55	0.37
Set 2	-	Samples Avg	0.25	0.34	0.26
Set 3	0	0.36	0.52	0.42	664
Set 3	1	0.28	0.58	0.38	456
Set 3	2	0.25	0.56	0.34	426
Set 3	3	0.19	0.54	0.29	328
Set 3	4	0.14	0.53	0.23	249
Set 3	5	0.14	0.47	0.22	276
Set 3	6	0.11	0.44	0.18	225
Set 3	7	0.15	0.56	0.23	235
Set 3	8	0.12	0.54	0.20	218
Set 3	-	Micro Avg	0.20	0.53	0.29
Set 3	-	Macro Avg	0.19	0.53	0.28
Set 3	-	Weighted Avg	0.23	0.53	0.31
Set 3	-	Samples Avg	0.19	0.40	0.23

Table 4: Extended testing of the Random Forest model

8.4. Appendix 4 - Code

Our source code can be found at a GitHub repository here:
https://github.com/trembleyjr/senior_design_may24-48