Public, Web-Accessible, Searchable Database of Protein Spectra

Capstone Project: CSCI 6838.11

July 29th 2017

Mentor:

Dr. Michael G. LaMontagne

Asst. Professor of Microbiology

Email: LaMontagne@uhcl.edu

Instructor:

Dr. Pradeep Buddharaju

Asst. Professor of Computer Science

Email: Buddharaju@uhcl.edu

Team: Hunter Hawks

Ravi Chandra Reddy Are (AreR0248@uhcl.edu)

Pranay Dheeraj Vajhala (VajhalaP0133@uhcl.edu)

Aditya Thodimela (ThodimelaA9636@uhcl.edu)

Satya Naga Sai Bhargavi Kolli (KolliS9373@uhcl.edu)

Acknowledgements

This project is a result of continuous and combined efforts of the team, Hunter Hawks. However, it would not have been possible to accomplish it in such a short span without the cooperation of the faculty and staff at the University of Houston-Clear Lake.

We acknowledge the contribution of:

Dr. Michael G. LaMontagne

For taking time out of his busy schedule, attending our team meetings and giving us all the required knowledge and resources to start with and incrementally build the project.

Dr. Pradeep Buddharaju

For guiding us throughout the project: giving his insightful feedbacks, reviewing the project progress on a regular basis, and mentoring us. We would like to mention a special thank you for the moral support he provided when we hit several roadblocks during the development of this project.

University of Houston- Clear Lake

We appreciate the efforts of the staff at the University of Houston-Clear Lake for providing us 24-hour access to computer labs, study areas and collaborative zones. Also, all the required softwares and applications were arranged within a short span, if not immediately.

Other Teams

In our journey towards learning and exploring various concepts and ideas, we were accompanied by 4 other teams. We value the contribution of our fellow teams: Tech Dynamites, Tech Geeks, Hawk Riders and Code Crackers. The ideas put forward by the members of these teams during our project presentations fueled our progress.

Table of Contents

1. Introduction

1.1. Background

Bacterial identification and classification is a very complex task and requires a lot of human intervention as well. Therefore, it is important to discover effective methods and procedures to simplify this task and automate it in a way so as to reduce the human intervention. Bacterial Spectra classification is one of the broadly utilized techniques to examine properties of these microscopic organisms [1]. Every bacterial species produces novel spectra because of the properties within them. Using those properties, we can differentiate between the types of the bacteria.

In this procedure, bacteria collected for testing purposes are subjected to a laser beam. This results in emission of a protein spectra from the bacteria, which can then be used for testing and investigative purposes.

Most of the water bodies around us are being toxified, resulting in the extinction of many aquatic animals. Also, there are many endangered species that are at the risk of extinction. If not acted upon immediately, future generations would become unfamiliar with many of the aquatic species living in these water bodies today. One way to do this is by identifying the amount of pollutants water bodies carry in them because of constant exposure. We can discover the extensive variety of features that these microscopic organisms display in them and decide the measure of risk they cause to the ecosystem around them. It can be resolved in view of the bacterial elements of every species. This can help in sparing and treating water bodies to a substantial extent.

1.2. R Language

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS [2]. It's free, open-source, powerful and highly extensible. It comes with many prepackaged tools that are already available, so it becomes easier for beginners to learn the language and professionals to develop sophisticated applications.

Since it is a programmable environment that uses command-line scripting, one can store a series of complex data-analysis steps in R, which lets the user re-use and analyse work on similar data more easily than if one were to use a point-and-click interface [3]. One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed.

1.2.1. Where is it available?

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form [4]. This is available on the website https://cran.r-project.org/mirrors.html

1.2.2. Downloading and Installing R

To download R, please choose the preferred Comprehensive R Archive Network (CRAN) mirror.

1. Open a browser and visit the website www.r-project.org.
2. Click the "download R" link in the middle of the page under "Getting Started."
3. Select a CRAN location (a mirror site) and click the corresponding link.
4. Click on the "Download R for Windows/ (Mac) OS X" link at the top of the page.
5. Click on the file containing the latest version of R under "Files."
6. Save the .pkg file, double-click it to open, and follow the installation instructions.
7. Now that R is installed, you need to download and install RStudio.

1.2.3. Downloading R Studio

R Studio can be downloaded by following the below steps:

1. Go to www.rstudio.com and click on the "Download RStudio" button.
2. Click on "Download RStudio Desktop."
3. Click on the version recommended for your system, or the latest Mac version, save the .dmg file on your computer, double-click it to open, and then drag and drop it to your applications folder.[5]

1.3. Packages Used

The following packages were used in order to achieve the integration of the database with the web application.

1.3.1. R Shiny Package

Shiny is a web application framework for R. It is an R package that provides an elegant and powerful web framework for building web applications using R. Shiny helps you turn your analyses into interactive web applications without requiring HTML, CSS, or JavaScript [6].

RShiny makes it incredibly easy to build interactive web applications with R. Automatic "reactive" binding between inputs and outputs and extensive prebuilt widgets make it possible to build beautiful, responsive, and powerful applications with minimal effort.

1.3.2. shinythemes

Themes for use with Shiny include several Bootstrap themes from http://bootswatch.com/, which are packaged for use with Shiny applications.

```
install.packages(“shinythemes”)
```

1.3.3. Matrix-Assisted Laser Desorption/Ionization (MALDIquant)

MALDIquant provides a complete analysis pipeline for MALDI-TOF and other 2D mass spectrometry data. This vignette describes the usage of the MALDIquant package and guides the user through a typical preprocessing workflow [7].

```
install.packages(c("MALDIquant", "MALDIquantForeign"))
```

1.3.4. Matrix-Assisted Laser Desorption/Ionization (MALDIquantForeign)

MALDIquantForeign provides routines for importing/exporting different file formats into/from MALDIquant. This vignette describes the usage of the MALDIquantForeign package [7].

1.3.5. P-Values via Multiscale Bootstrap Resampling (pvclust)

An implementation of multiscale bootstrap resampling can be used for assessing the uncertainity in hierarchical cluster analysis. It provides approximately unbiased (AU) p-value as well as bootstrap probability (BP) value for each cluster in a dendrogram.

```
install.packages("pvclust")
```

1.3.6. Classification and Regression training (caret)

The caret package (short for classification and regression training) contains functions to streamline the model training process for complex regression and classification problems. The package utilizes a number of R packages but tries not to load them all at package start-up. The package "suggests" field includes 27 packages. Caret loads packages as needed and assumes that they are installed. Install caret using

```
install.package("caret", dependencies = c("Depends", "Suggests"))
```

1.3.7. Amazon Web Services: Simple Storage Service (aws.s3)

A simple client package for the Amazon Web Services (AWS) Simple Storage Service (S3) REST API. While other packages currently connect R to S3, they do so incompletely (mapping only some of the API endpoints to R) and most implementations rely on the AWS command-line tools, which users may not have installed on their system.

To use the package, we need an AWS account to enter the credentials into R. The keypair required can be generated on the *IAM management console* under the heading *Access Keys*. The access to the secret key is restricted to only one-time use. After it is generated, it is preferable to store it in a secure location. In the event of a lost, stolen or forgotten key pair, a new one can always be generated.

By default, all **cloudyr** packages look for the access key ID and secret access key in the environment variable.

1.4. Algorithm used

The machine learning algorithm that is implemented to perform the spectra matching is the Cosine Similarity algorithm.

1.4.1. Cosine Similarity

Cosine Similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any other angle. It is thus a judgment of orientation and not magnitude; two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. Cosine similarity is particularly used in positive space, where the outcome is neatly bounded in [0,1]. The name derives from the term "direction cosine"; in this case, note that unit vectors are maximally "similar" if they're parallel and maximally "dissimilar" if they're orthogonal (perpendicular). This is analogous to the cosine, which is united (maximum value) when the segments subtend a zero angle and zero (uncorrelated) when the segments are perpendicular.

The cosine of two non-zero vectors can be derived by using the Euclidean dot product formula:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 \cos\theta$$

Given two vectors of attributes, A and B, the cosine similarity, cos(θ), is represented using a dot product and magnitude as

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Where A_i and B_i are components of vector A and B respectively[8].

For information retrieval, the cosine similarity of two documents will range from 0 to 1, since the term frequencies (tf-idf weights) cannot be negative. The angle between two term frequency vectors cannot be greater than 90°. Therefore, the similarity value for two .csv files when compared will range between 0 and 1, Where 0 indicates no similarity at all and 1 indicates the exact match.

2. Requirements

2.1. Requirements Specification

Requirements gathering and analysis is very much important because it drives us towards the right path and focuses on what we need to do. This in turn guides us towards proper development and testing phases of the project. Requirements are classified into three categories: Business Requirements, Functional Requirements, and System Requirements. Business Requirements define specific goals that must be achieved to reach an objective. Functional Requirements are the detailed break-down steps for reaching those described goals mentioned in Business Requirements. System requirements define the hardware and software that are required in developing the project.

2.2. Business Requirements

BR1.	Storing the peak values of the data fetched into a persistent database.
BR2.	Comparing the unknown spectra with the existing spectra.

The business requirements include extracting the peak values from the data imported and then storing those extracted values in a persistent database.

Secondly, the user should be provided with the ability to compare the unknown spectra with the data that is already present in the database. The functional requirements for each of these business requirements are mentioned below.

2.3. Functional Requirements per each Business Requirement

BR1.	Storing the peak values of the data fetched into a persistent database.
FR1.1.	The format of the fetched data should be storable in the database.
FR1.2.	Type of database that is apt for connectivity

	with R Studio.
FR1.3.	Establish connection between the web application and the database.
FR1.4.	Store the data in the database.

Data imported by the user into the web application should have a format that is easily storable in the database. Also, the database that will be selected should be compatible with R Studio and allow a smooth communication between the web-application and the database. Since the R Studio has a built-in package for Amazon Web Services, the simple storage service (S3) is selected. The S3 database accepts files of different formats. However, the .csv files are much easier to store.

BR2.	Comparing the Unknown spectra with the existing spectra.
FR1.1.	Importing this unknown spectra data into R studio.
FR1.2.	Smoothing the unknown spectra.
FR1.3.	Comparing these smoothed unknown spectra with the existing spectra in the database.
FR1.4.	Display if any matches are found.

Since CRAN is included in R studio, we can install all the packages present within it. Two additional packages "lsa" and "SnowBallC" are used for performing the matching. These two packages are external packages which are not present in CRAN, and hence, the user needs to install them manually. Once a good match is obtained, the results are displayed in the user interface.

2.4. System Requirements

System requirements include hardware and software that are required for the project implementation.

- Hardware Requirements:
  - Windows machine or MacBook.
- Software Requirements:
  - Operating System: Windows or MacOS
  - Software: R Studio, R Shiny.
  - Programming Language(s): R programming
  - Amazon Web services: Amazon S3 (Database)
  - AWS.S3 Package installation in R Studio

3. Design

The design phase is composed of the architectural diagram, work-flow diagram and sequence diagram.

3.1. Architecture Diagram

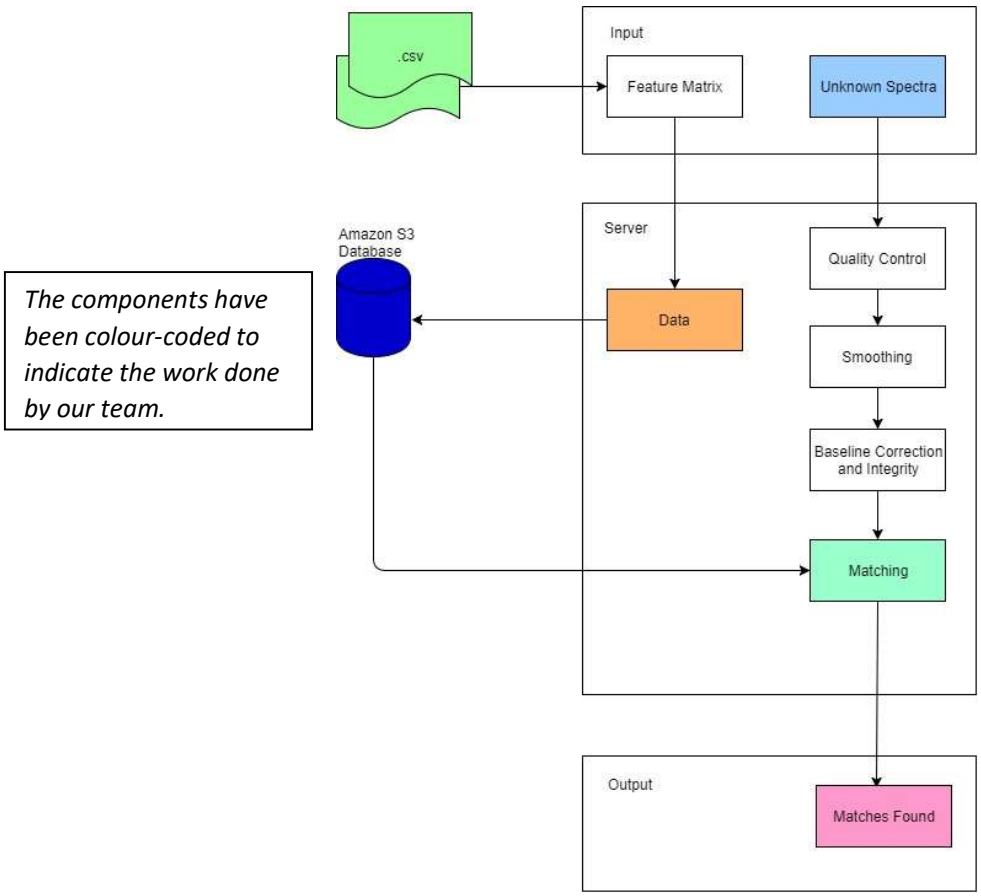

Fig. 3.1. Architecture Diagram

The web application is very user friendly and it defines each step in MALDIquant, which is a quantitative analysis of mass spectrometry data. The Fig. 3.1 shows the incorporation of a series of operations that can be clustered into input/output functionality, pre-processing functionality, and analysing functionality.

The above figure shows the series of pre-processing steps involved in the processing of raw data consisting of Quality Control, Smoothing, Baseline Correction and Integrity. Once the raw input is fed to the web application and after the pre-processing steps, it is converted into a .csv file and stored in the Amazon Web Services S3 bucket. Then, the matching algorithm compares the file with the already existing data on the S3 cloud. Once the matching is done, the results are displayed under the 'matching' tab on the web application. The user then has the option of saving the matched results to the database by clicking the "save results" option beside the file with the best match.

3.2. Work-Flow Diagram

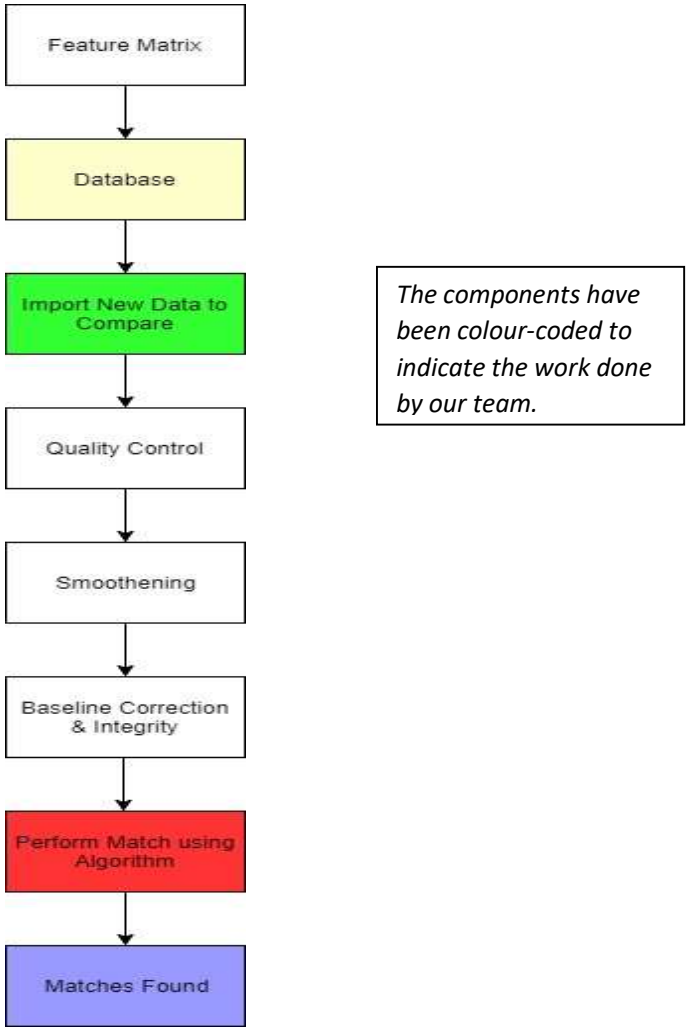

Fig. 3.2. Work-Flow Diagram

Fig. 3.2 depicts the Work-Flow diagram of the application. Every time a .zip/.rar/tar.gz/tar.bz2 file is uploaded into the web application, it is subject to the pre-processing steps and the feature matrix is obtained by extracting the peak values from the output. The feature matrix can then be downloaded by the user. The application then imports the new data to compare with the existing data.

Once the pre-processing steps are completed, the Cosine Similarity machine learning algorithm will perform the matching operation to identify the category of the protein spectra it belongs to, and then displays the matching results on the User Interface of the web application.

3.3. Sequence Diagram

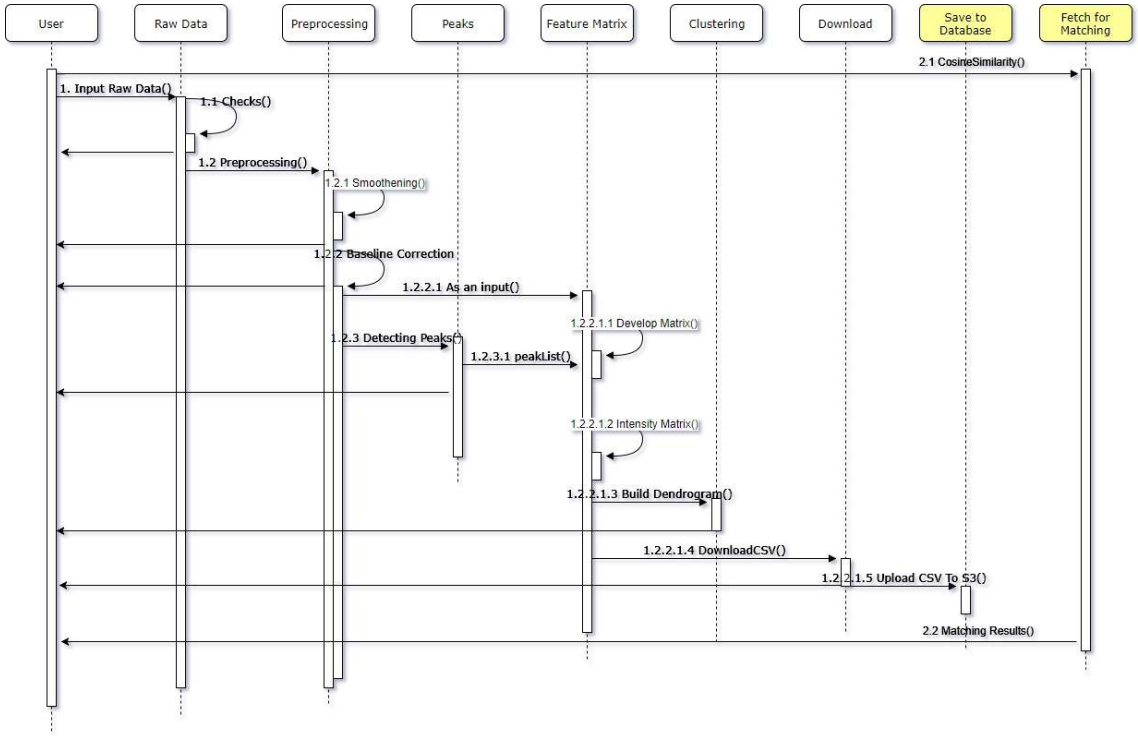

Fig. 3.3. Sequence Diagram

The following figure shows the sequence of operations involved in the project, the operations can be categorized as pre-processing steps, database integration and application of the matching function. The user has complete control of the flow of the application throughout the whole process.

4. Implementation

4.1. Experimental Setup

To begin with the project, R should be installed followed by the installation of MALDIquant and MALDIquantForeign. These packages can be directly installed from CRAN as,

```
install.packages(c("MALDIquant", "MALDIquantForeign"))
```

Since Shiny was used to build the interactive web application, in order for us to be able to launch the application, we require to install Shiny package as,

```
install.packages("shiny")
```

Apart from the above packages, we need the following packages:

```
install.packages(“shinythemes”) for shinythemes
install.packages(“pvclust”) for pvclust
install.packages(“caret”, dependencies = c(“Depends”, “Suggests”))  for caret
install.packages(“aws.s3”) for Amazon Web Services S3 package
install.packages(“lsa”) for lsa package
install.packages(“SnowBall C) since lsa package is dependent on SnowBallC
```

Upon successful installation of these packages, the following libraries needed to be loaded as,

```
library(“MALDIquant")
library(“MALDIquantForeign”)
library(“shiny”)
library(“caret”)
library(“aws.s3”)
library(“lsa”)
library(“SnowBall C)
```

4.2. Procedure

The already existing web application is connected to Amazon S3 database using the below code:

```
s3BucketName <- "hunterhawks"

Sys.setenv("AWS_ACCESS_KEY_ID" = "AKIAJRVLXNHWIV3HRC6A",

"AWS_SECRET_ACCESS_KEY" =
"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",

"AWS_DEFAULT_REGION" = "us-east-2") [9]
```

Then the feature matrix that is generated is uploaded into the database:

```
put_object(file = file,bucket = s3BucketName, object = paste("FeatureMatrix-
", Sys.time(), ".csv", sep="") ) [10]
```

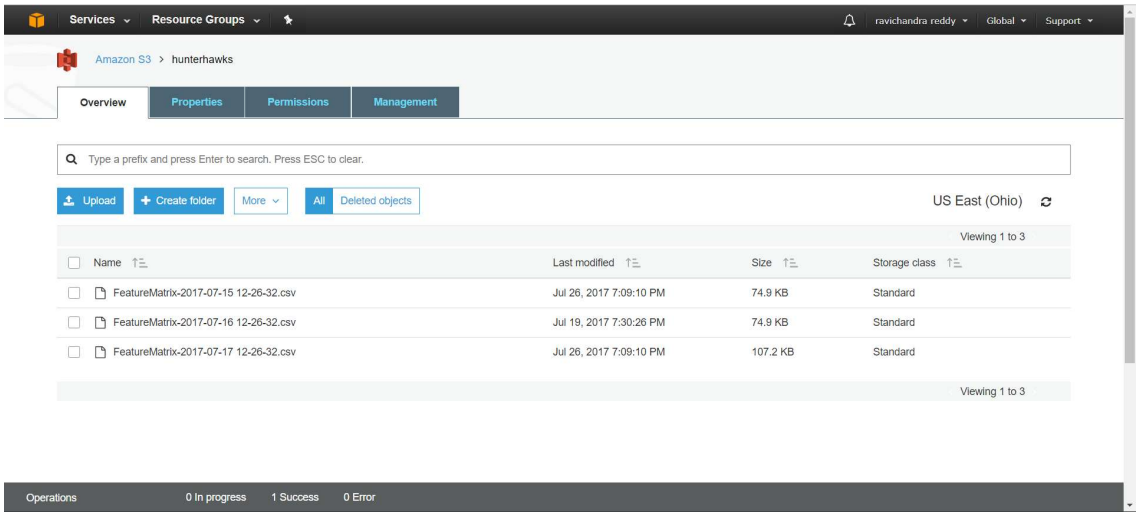

Fig 4.1 Connecting and uploading the data to the database

For matching, we again fetch the data from AWS S3 database using the below code:

```
getbket <- get_bucket(s3BucketName)
```

```
con1<-capture.output(print.simple.list(getbket))

con2 <- textConnection(con1)

data1 <- read.csv(con2,sep=" ")

close(con2)

filter1<-filter(data1,X == 'Contents.Key',grepl('.csv',X.11) | grepl('.csv',X.10))

filter1$object_list=paste(filter1$X.10,filter1$X.11)

obj_list<-subset(filter1,select=c(object_list))

conf <- 0

obj_Name <- " "

j<-1

for( val1 in obj_list$object_list){

getcsvobj <-get_object(object = val1, bucket = s3BucketName )
```

We perform matching between the user provided data with the already existing data which is in the AWS S3 database. Below is the code for performing the same:

```
csvobj <- rawToChar(getcsvobj)

con <- textConnection(csvobj)

aws_csv_df <- read.csv(con)

k<- 0

for (i in 1:379 ) {

        x1 <- aws_csv_df[2:16, 2:380]

        y1 <- fm[2:16, 1:379]

        res <- lsa::cosine(x1[,i], y1[,i]) [11]

        k <- k + res

}

p<-k/379
```

4.3. Results

Matching is performed and the result is obtained as a confidence level between the two feature matrices. A list of confidence levels between the user's feature matrix and each of the feature matrices that are present in the AWS S3 database is displayed. Also, the user would be able to download the feature matrix that was generated in the web application.

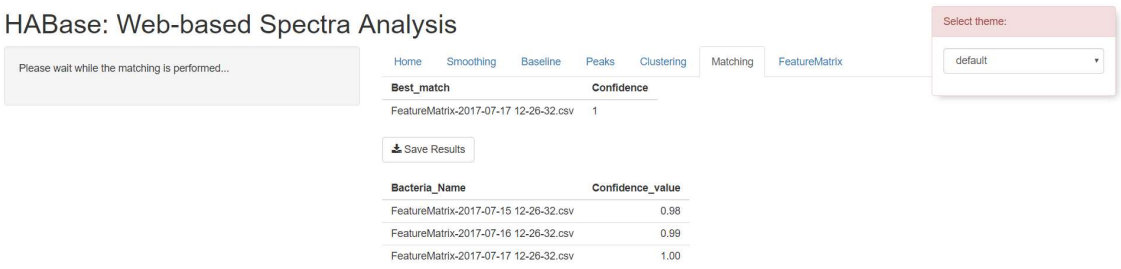

Fig. 4.2. Matching results

5. Testing and Validation

For testing and verification, Hunter Hawks team used two datasets namely fielder 2009 subset and 16-12-01-MLM. Fielder dataset is a subset of MALDI-TOF data and has a list of 16 MassSpectrum objects.A feature matrix is generated as an output through the web application. Manipulation of this feature matrix is done to develop three different feature matrices, with the same number of rows and columns.

Looking at Fig. 4.2, the results obtained are 0.98, 0.99, and 1.00. The value 0.98 and 0.99 are the result of matching different matrices from the one the user uploads. The value 1.00 is a result of matching the same feature matrices.

6. Project Management

Team “Hunter Hawks” constituted of 4 members. In order to simplify the task and to ensure timely achievement of the goals that were set, each member lead a specific area of the project and ensured that all other members of the group stick to the set timelines. The following are the roles in which each member of the team lead:

- *Project Manager*: Ravi Chandra Reddy Are
- *Technical Lead*: Pranay Dheeraj Vajhala
- *Presentation and Website lead*: Aditya Thodimela
- *Documentation lead*: Satya Naga Sai Bhargavi Kolli

6.1. Means of Team Communication

- *Code sharing*: Bit Bucket was used by our team to save and share various versions of the code that was developed by each of the team members. Commits were done after debugging and testing at regular intervals following agile methodologies.
- *Meetings (Skype, In-person):* The mentor and the instructor were kept updated about the progress of the project. Various ideas emerged during these meetings and effective ways to tackle the roadblocks were suggested by **Dr. Buddharaju** and **Dr. Michael LaMontagne** during these meetings that were held online via Skype and face-to-face on Tuesdays and Thursdays of the week throughout the duration of the project.

6.2. Project Timeline

The following tasks contributed to the project timeline

- *Requirement Analysis*: Since this was a completely unfamiliar topic for each of us, it took 5 days for us to understand it and then analyze the various requirements for the database integration and performing the matching.
- *Design*: The design phase took us 8 days to complete. We developed the architecture design initially and then designed the work-flow. Once we had a clear idea of the architecture and the flow, we developed the sequence diagram. Consequently, we entered the implementation stage.

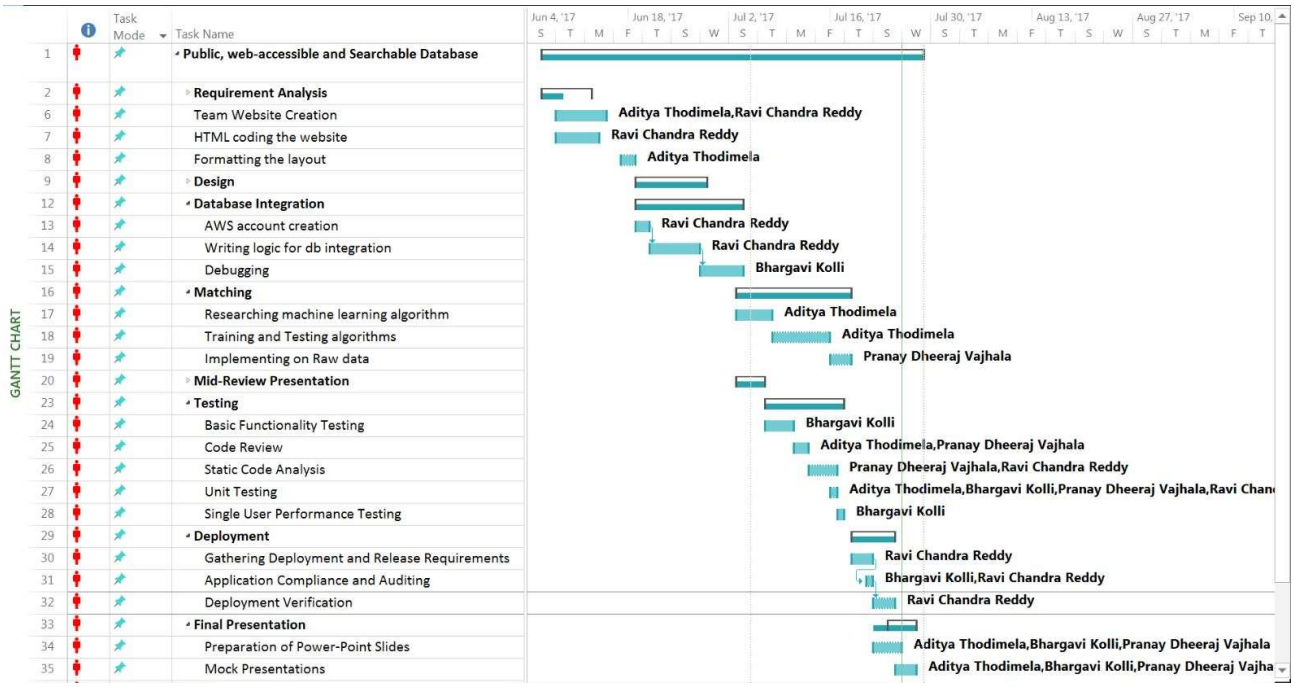

Fig. 6.1. Project Timeline

- *Database Integration*: This was one of the two major parts of the project, though it did not take much time to create an Amazon Web Services Simple Storage Service (AWS S3) account, it took us 11 days to figure out a way to integrate it with the web application and then writing the code logic. Also, we encountered many roadblocks in the process and needed severe efforts of all the team members to overcome them.
- *Matching*: The matching was the other major task involved with the development of this project. This involved tedious research, training & testing and implementations on the real-time data. This phase took a total of 12 days for completion.
- *Testing*: Compared to database integration and matching, the testing phase took lesser time. Mainly, because the raw data that we had access to was very limited. Nonetheless, this phase uncovered several short comings in the developed model and gave scope to further development. We took 2 days to complete the testing phase.
- *Deployment*: The project deployment took us 4 days. It mainly involved gathering deployment and release requirements, Application compliance and Auditing, and Deployment verification.

Apart from the above tasks, the project timeline also consisted of other minor tasks like team website creation and constant updating, Mid-review presentation, Final presentation creation and project documentation.

6.3. Task Division

Since the project needed a lot of efforts and resources, it was very important to divide the larger task of development into several sub-tasks. Also, there was a severe need for someone to take up the initiative to ensure that the deadlines were met without compromising on the outcome. To ensure this requirement was met, we divided larger tasks into smaller tasks and then worked on the solutions and finally all the tasks were integrated.

6.3.1. Database Integration

The database integration involved creation of amazon s3 account, uploading data to the cloud, and retrieving data from the cloud. This phase was led by the project manager, **Ravi Chandra Reddy Are** and rest of the team members coordinated with him in the integration. Many roadblocks were successfully overcome during this phase by the combined efforts of all the team members.

6.3.2. Matching Algorithm

We chose trial-and-error method to complete this task. Matching algorithm was a very challenging task and involved testing of various algorithms like K-nearest neighbors, Decision tree and Cosine Similarity. It was led by **Pranay Dheeraj Vajhala** and **Aditya Thodimela**, and supported by Ravi Chandra Reddy and Bhargavi.

6.3.3. Training and Testing using Raw Data

The machine-learning algorithms that were short-listed by the team members looked very promising initially. However, the effectiveness of each algorithm was not clear until these algorithms were trained and tested repeatedly. This task was led by **Bhargavi Kolli** and other team members helped her in testing using the raw data that was made available to the team.

7. Road Blocks

In the process of achieving what was required, we as a team faced some road blocks:

- Understanding the data was a bigger road block in the initial stages of the project. As everyone in the team belong to the Computer Science background, it was hard to understand all the Micro-Biological terms.

- Integrating the web application with the Amazon S3 Database. Initially, we were not able to save the .csv file with the data in it.

- Coming up with the required packages that are needed for the implementation and installing them.

- Finding out which machine-learning algorithm is suitable for performing the matching. At the beginning, we tried to implement the matching function using KNN-algorithm. But we were not able to do so, since the KNN-algorithm requires labeled data and a target variable.

- Once the machine-learning algorithm is decided, integrating it with the web application was a harder task.

8. Conclusion

The team Hunter Hawks has successfully developed and deployed the required functionality of connecting the web application to the AWS S3 database. Also, performing the matching between user data and the data present in the database using Cosine similarity calculation.

9. Lessons Learned

- Learned how to work with Amazon S3 and how to integrate it into R environment in R-Studio.

- Learned how to deploy the application into the R-Shiny cloud server.

- Got a hands-on experience of using a machine-learning algorithm in R-Studio and how to use its power in Data Analysis.

10. Future Work

- Deploying the application into AWS cloud server which would increase its performance and a faster result of matching confidence level.

- Saving the bacterial spectra data into AWS with naming conventions. We were able to save the data into the database and the name starts with “feature matrix”, continued by the date and time. It would be better if the names of the files start with the name of the bacteria to which the data in the file belongs and any extra parameters required.

- For now, we used Cosine Similarity to compare the feature matrices and get the confidence levels. Incorporating more machine-learning algorithms for matching would be better, since the user would be able to select the algorithm according to his needs.

- We are able to save the data into the database and the whole data is saved into a single bucket instead of saving it to different buckets. As there would be many bacterial species of the same type, clustering them into a single bucket would be awful in the coming future. If there are other type of bacterial species, they should be placed in a different bucket together .

11. References

[1] https://www.r-project.org/

[2] https://cran.rstudio.com/

[3] http://www.computerworld.com/article/2497143/business-intelligence/business-intelligence-beginner-s-guide-to-r-introduction.html

[4] https://www.rstudio.com/products/rstudio/download/

[5] https://courses.edx.org/courses/UTAustinX/UT.7.01x/3T2014/56c5437b88fa43cf828bff5/

[6] https://www.rstudio.com/products/shiny-2/

[7] Gibb, S. and Strimmer, K. (2012). MALDIquant: a versatile R package for the analysis of mass spectrometry data. Bioinformatics, 28(17):2270-2271

[8] https://en.wikipedia.org/wiki/Cosine_similarity

[9] https://aws.amazon.com/s3/

[10] https://shiny.rstudio.com/articles/persistent-data-storage.html

[11] https://stackoverflow.com/questions/34045738/how-can-i-calculate-cosine-similarity-between-two-strings-vectors-in-language-r/