Indian Institute of Information Technology, Allahabad, Prayagraj

Software Engineering
Summary Report
Project 44
Real Time Medical Database Management System

Prateek Durgapal
IIT2019014

I IMPORTANCE

Our web application, PublicAgent (clickable link to web app), is a new self contained product which is aimed to increase the efficiency of updating scores for mental institutions in the USA to overcome the problems of current management faced by both, users as well as the Institutions. Online score updation, as well as score checking will reduce the paperwork, save a lot of time for users who have to waste time searching for the best institution at their place and for the institution to update scores offline.

II OBJECTIVE

The goal for development of this web based application is to provide people knowledge of the best hospital around them for a particular disease based on the feedback and experience of other patients who got treatment for the same in a particular hospital. The motivation behind this is to provide best treatment to a person for a healthy lifestyle without any useless trouble and travel.

III INTRODUCTION

This web application is for the automation of Mental Institution Management by maintaining two levels of users:

- Administrator Level
- User Level (Patients)

It maintains the mean score for all the patients in a database that attended the institution and is updated in real time if new scores are given by any patient. It also provides all the users information of the best mental institution in their state. There are also some side features like live news on the web application and feedback by the patient to the HQ.

This project is beneficial for the society as it will aid in the alleviation of people with addiction to drugs, alcohol and tobacco. In the Trends section we will see the growing number of these people, especially drug users, who are being treated by these hospitals. Our web application will not only help them or their well-wishers to locate the best hospitals for their treatment but will also allow them to score the treatment the receive so that other people who may undergo the treatment in the future will get unbiased opinion about it. They can also leave a query regarding their problems to us which we will try to rectify as soon as possible.

In terms of uniqueness, the web app has an extremely user-friendly user interface. This is evident from the fact the presence of helpful elements all over the website. For example, if a user doesn't the code for a treatment, there a dropdown menu for it. If a user doesn't know the code for a state, a table can be opened with click of a mouse. If a user wants to see the complete list of facilities with IDs while scoring, that provision is done as well. So the User Experience is boosted by all these elements. Also the technique used for data division is simple and fast allowing for instant results. There is also an additional feature of graph generation which sends 1000 random score request to the server and the database gets temporarily updated. The graph is shown to user and all the changes are reverted back. This is a way for any user to assess the speed of our technique.

IV DATASET

We have used the *Veterans Health Administration Behavioral Health*[1] dataset. This contains the following columns along with the how or where we have used them in our project:

- Facility ID: This an alphanumeric unique ID (primary key) for each hospital/facility present in the country. We use this in our s*core* and *search* features to identify hospitals.
- Facility Name: The full name of the facility. Used in the result of the *search* feature.
- Address: The local address of the facility. Used in the result of *search* feature.
- City, ZIP Code, County Name: As the name suggest, all the other address related information.
- State: The state in which the facility is present. It is 2 character long capitalized version. For example, New York is represented as NY, Florida is represented as FL, etc. Used as an input to *search* feature.

- Phone Number: The contact number of the facility. Used in the result of *search* feature.
- Measure ID: This is unique ID for 5 of the treatments that are offered at every hospital. Used as an input in the *score* feature as well as the *search* feature.
- Measure Name: This is a short description corresponding to the Measure ID. Used as a helper text for patients to know more about the treatments.
- Score: The mean score of a particular treatment and facility. Used in the result of *search* feature. Also, is updated real time by new scores.
- Sample: The total number of patients who have scored. Is updated real time by new scores.
- Location: The coordinates of the facility i.e. latitude and the longitude. Used in the result of *search* feature.

V BACKEND

For the database we used *PostgreSQL*[2]. First and foremost we created a schema in the database as identical to that of the database provided. Then we imported all the rows of the dataset into the database.
We have used *Django*[3] in our project as a web framework to handle both the frontend as well as the backend. The following subfolders in our main project folder are essential:

- PublicAgent (name of our project): The first step of our project was to create this subfolder which contains settings related to our Postgres database such as the username, password and the name of the database. It also contains settings related to Django itself like base directory, secret key, allowed hosts, installed apps, middleware, root URL, templates, password validators and email information. Finally, this subfolder contains all the URLs that will be required to access our web app over localhost:

- root: Next we created this subfolder. This contains code related to the admin. It also contains classes corresponding to tables in our database. These are the Element (stores information about each hospital) and the Profile (stores information about each user) class. Next are methods which act as an interface between the frontend and backend. There are 3 methods here:

    - index(request) - returns the home page by rendering index.html
    - about(request) - return the about page by rendering about.html
    - query(request) - used for feedback/query given by user in form of a text message and then sent to the HQ email.

```
→ if request method is 'POST' then
   → fetch name, data, username, message from request
   → create appropriate subject
   → initialize the sender and receiver email address
   → send the query through mail
   → return to home page
```

Finally, we have the sub URLs that will lead us to the query and about page over localhost.

- accounts: Thirdly, we created this subfolder for recording information for every account created and to also create one. Contains 6 methods:

  - register(request) - takes information about user and sends verification email to the email provided (calls send_email()). This puts him/her into the waiting list and so the user cannot login before verification.

```
→ if request method is 'POST' then
   → retrieve user information from request
   → check if password1 = password2
   → check if the username or email exists
   → if check fails then return to register page
   → create new user
   → save the user onto database
   → call send_email() with an auth token
   → return to login page
→ else return to register page
```

  - send_email(email, first_name, username, password, auth_token) - sends the email with appropriate subject, message which contains the user credentials as well as a link to to click on to verify the account. That link invokes verify().

```
→ create subject
→ create message including the auth token in
   verification email
→ initialize the sender and receiver email address
→ send email
```

  - verify(request, auth_token) - verifies the user by comparing the auth tokens with the database. If the tokens match, the user is verified and can now login and login() is called. Otherwise error() is called.

```
→ if profile exists then
   → retrieve user profile from database
   → if profile is verified then return to login
   → change the verification status of profile
   → save the profile
   → return to login
→ else return to error page
```

- login(request) - takes input the username and password and checks in the database for these credentials. Three cases are possible - user is there but not verified, user is there and verified, user is not there. In either case, appropriate message is displayed (and logged in in case it is verified).

```
→ if request method is 'POST' then
   → retrieve user info from request
   → authenticate user
   → if user authenticated then
        → if profile is not verified then
               return "Not Verified"
        → login the user
        → return to home page
   → else return "Invalid Credentials"
→ else return to login
```

- logout(request) - simply logs out the user.
- error(request) - renders the error page by rendering error.html

Finally, we have the sub URLs that will lead us to the login, logout, register, error and verify page over localhost.

- facility: Fourthly, we created the facility subfolder to handle our *search* feature. There is only 1 method here:

  - best_facility(request) - takes 2 inputs. One for the state and other for the measure ID. If any one of those is invalid (by checking in the database) or both fields are empty, appropriate dialog box with correct usage of the search bar is shown. If one input is missing and the other is valid, then also results are shown to the user with all the instances of the missing field taken into account. We fetch all the records from the database while taking into account the inputs given by user and output the table in descending order of score. If score is same, higher sample is given priority.

```
→ if request method is 'POST' then
   → get state, treatment from request
   → check if both of them are not empty
   → check if state and treatment are valid
   → if check fails the return to home page
   → filter the required record from the database
   → sort them according to score
   → return to best facility page
→ else return to home page
```

Finally, we have the sub URLs that will lead us to the best-facility page over localhost.

- score: Fifth, we created this subfolder to allow users to score their treatments. This also contains only 1 method:

  - score(request) - Takes input of facility ID, measure ID, score. First checks if score is in the range 0-100. Since the IDs were chosen from a predefined dropdown or radio buttons, no checking for existence is required. So, we fetch the records from the database with the given IDs. Update the score and sample values. Store them back in the database. And display a dialog box for the user to see the statistics of the treatment.

```
→ if request method is 'POST' then
   → get facility_id, measure, score from request
   → check if score is between 0-100
   → if not, return "Invalid Score"
   → get records from database given the id, measure
   → new_total = (old_score * old_sample + given_score)
   → new_score = new_total / (old_sample + 1)
   → new_sample = old_sample + 1
   → update the database with new_score, new_sample
→ else return to score page
```

Finally, we have the sub URLs that will lead us to the give-score page over localhost.

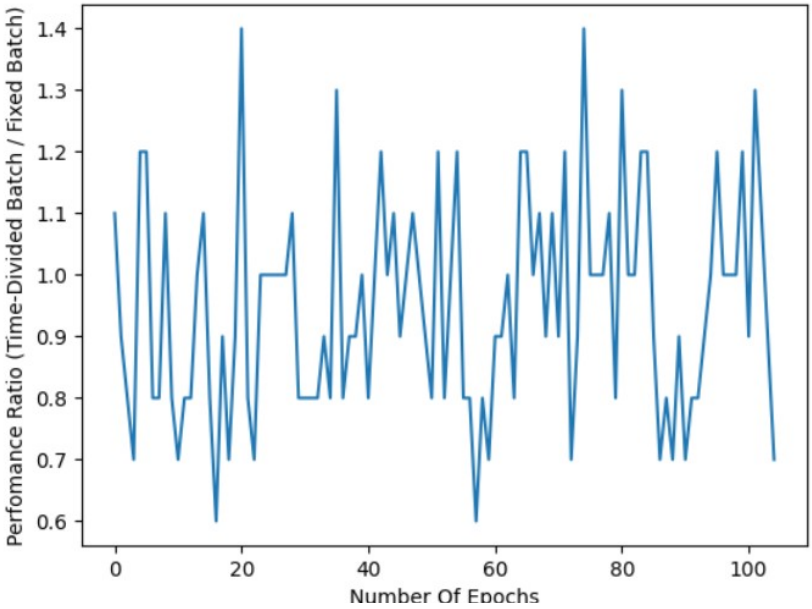
- Graph: Lastly, we created the graph subfolder which generates the graph of our data division technique. Is does this by dividing the data into equal batches and comparing the efficiencies of the two. This also contains one method:

  - graph(request) - used a list of *times* which appends 1000 random times between 0 to 1 sec, which tells our server when a user gets registered, and it tries to give a score. So to compare their efficiencies, we took 10 users per batch for equal batch performance and 5 sec time interval for time batches. So, the efficiency is basically how many users are logged in 5 sec, i.e ratio

of (users in equal batches) / (users in time batches successfully processed in 5 sec). So we get a graph for efficiency vs epochs, where epochs are basically number of times we are comparing the efficiencies.

VI TRENDS

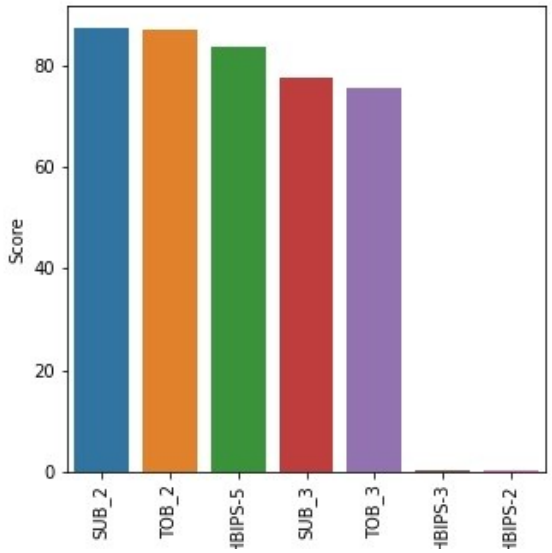
The following is the graph generated with the help of the graph() method mentioned above. Since the requests generated are random in nature, the graph will be different for different users:



As we can see the graph is very fluctuative and there is no certainty which method is better, so we can take anything of the two for a good performance. If there is heavy traffic on our side we will use the time divided batch and if the traffic is modulate than equal batches are doing fine.

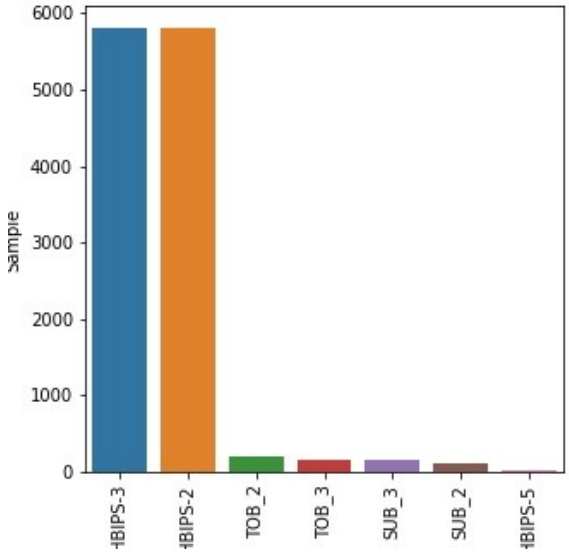
We also extracted some information from the dataset. For that, we first removed all the null values of Score and Sample from the dataset. Then, grouped it on the basis of Measures. After grouping, we found the mean Score and Sample and generated the plot. Finally, we also plotted coordinates of hospitals and superimposed them on the map of the USA to get an idea of where these hospitals lie. We have the following observations:

- First, we have the Measure vs. Score bar graph:



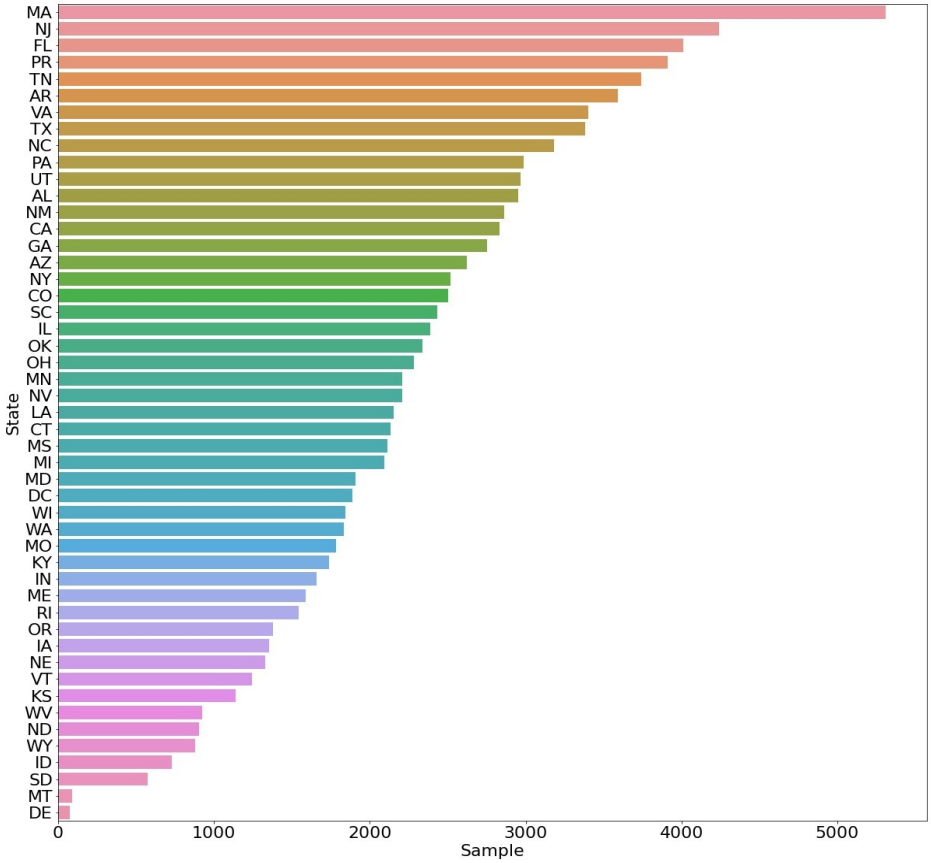
In this graph we see that, the score for the first 4 measures are in the range 70-95 but for the last two, it is almost 0. This maybe due to 2 reasons – either the people who undertook the treatment didn't like it and thus gave it a very low score or some kind of scaling has been done on the HB measures. After analyzing the dataset, the latter cannot be the case as there were still some scores with values like 70 or 80. So, I would assume the former to be true.

- Second, we will look at the Measure vs. Sample bar graph:



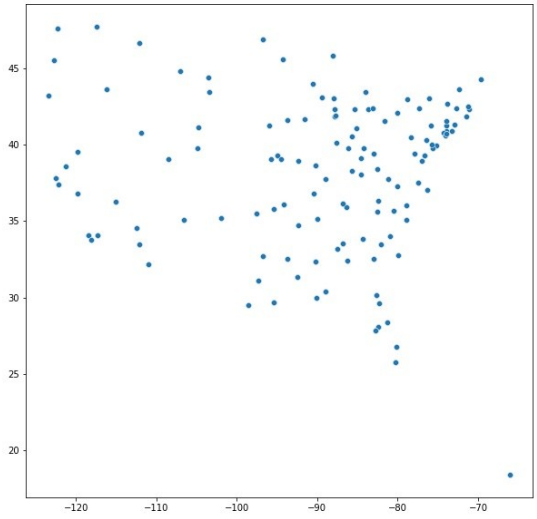
There is an astounding negative correlation here. The measures with highest scores have the lowest sample s and vice versa. This means the people in the USA were mostly treated for drug usage by seclusion (HBIPS-3) or by physical-restraint use (HBIPS-2). This also goes to show that people there don't have that much problems regarding smoking and alcohol usage[4].

- Thirdly, let's look at State vs. Sample graph:



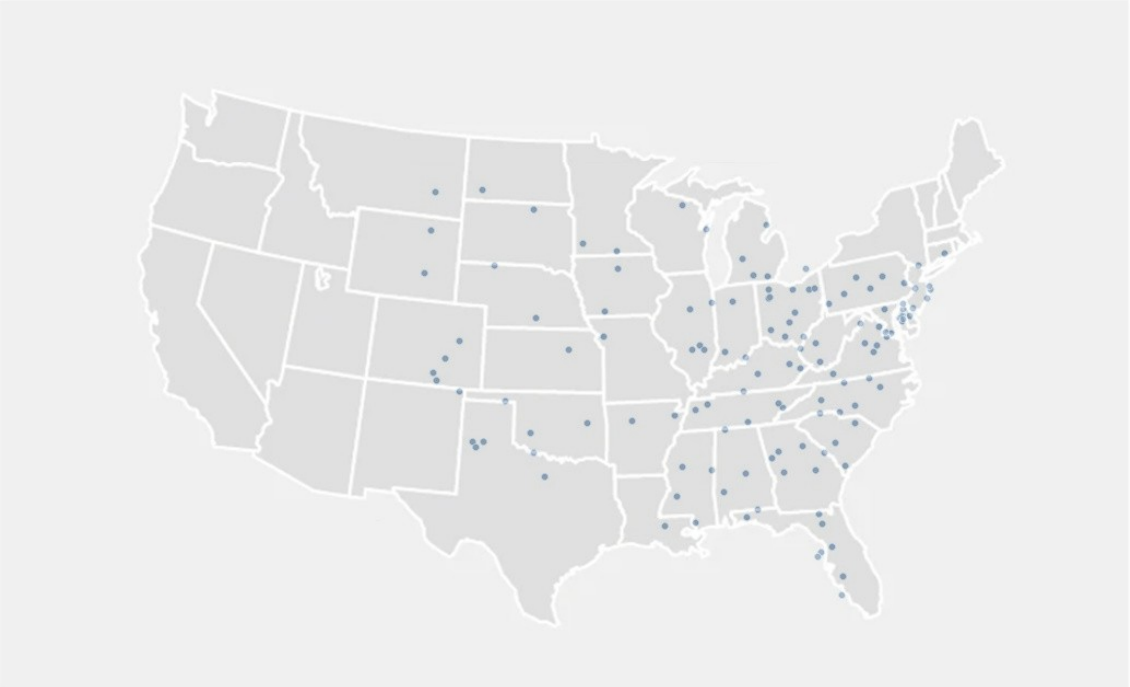
We see that the state MA (Massachusetts) has the highest patients leading by around 1500 from NJ (New Jersey) which is 2nd. At the last we have two states with almost identical patients. MT (Montana) and DE (Delaware). These have only 10-20 patients. The reason might be population. Both MT and DE are at 45, 46 (out of 51) positions in terms of population[5]. For MA, it is due to high drug use here[6].

- Finally, we have the coordinate plot:



This plot depicts that there is a dense area of hospitals towards the eastern side of the USA. After carefully observing the map we deduced that this part is actually near the East Coast of the United States[7] which is very densely populated and also a highly developed area. So it makes sense to have more hospitals here.

The following is the superimposed coordinates on the map of the USA to get a clearer picture:



VII FUTURE ADD-ON

We have decided to implement the following in future:

- To show information not only about the US hospitals but also about hospitals in other countries as well, primarily India.
- We can also add the feature of consulting doctors online w.r.t. the treatment they want and also booking appointments with their choice of hospital.
- Lastly, we wanted to add user profiles, where a user can see all the past scores he/she given.

VIII REFERENCES

1. https://data.cms.gov/provider-data/dataset/6qxe-iqz8
2. https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-18-04
3. https://docs.djangoproject.com/en/3.2/intro/tutorial01
4. https://www.addictioncenter.com/addiction/addiction-statistics
5. https://en.wikipedia.org/wiki/List_of_states_and_territories_of_the_United_States_by_population
6. https://www.newenglandrecoverycenter.org/drug-abuse-trends-in-massachusetts
7. https://en.wikipedia.org/wiki/East_Coast_of_the_United_States