e-YSIP 2019

Auto Tagging Algorithm 2.0

Anuj Trehan
Sharad Mishra, Tenzin Dhekyong
Duration of Internship: 03/06/2019 – 15/07/2019

2019, e-Yantra Publication

Auto Tagging Algorithm 2.0

Abstract

Main task of any Organisation conducting the online/offline Exams is to address the challenges faced in ensuring authenticity and fairness for Test-Takers to who attempt Online Exams. In this project I have came-up with an Web-application which will ensure that questions are automatically tagged with correct difficulty level for next iteration of Exam, using various Unsupervised Machine Learning and Deep Learning analysis on the performance of test-takers for previous exam.The Web-app will have a full admin dashboard and users to edit the questions and apply analysis on difficulty level of the questions

Completion status

- Read the reference paperAuto-Tagging for Massive OnlineSelection Tests: Machine Learning to the Rescue.
- Used data points from database with DB connector sqlalchemyORM tool for python.
- Calculated all the features including weighted features given in paper.
- K-Means Clustering algorithm was applied on the normalfeatures and accuracy was calculated.
- K-Means Clustering algorithm was applied on the normalfeatures and accuracy was calculated.
- Self Organizing Maps Algorithms was read and applied on the dataset.
- Results were analyzed.
- Integrated the Models into a Flask REST API.

- Dashboard with user and admin authentication was developed.
- Debugging of the Stack for better performance.

1.1 Software used

- The following is the software requirement:-
  - Operating System used: Ubuntu(18.04LTS).
  - MySQL Server.
  - Programming languages used: Python& NodeJS.
  - NumPy.
  - SciPy.
  - Pandas.
  - Matplotlib.
  - PyMySQL.
  - SQLAlchemy,
  - Scikit-learn.
  - Tensorflow.
  - Flask.
  - Flask-SQLAlchemy.
  - Flask-Cors.
  - Express JS
  - Passport JS
  - Express-Handlebars
  - JQuery
  - Bootstrap
  - JQuery-datatables
  - Sequelize
  - mysql
  - mysql2
  - body-parser
  - bcrypt

  - virtualenv

- Versions of different software are:-
  - MySQL Server: This will install latest version for Ubuntu 18.04 follow this for other OS
  - Python:v3.7.3 (for windows and mac only)
  - NodeJS: v12.4.0
  - NumPy: v1.16.4
  - SciPy: v1.3.0
  - Pandas: v0.24.2
  - Matplotlib: v3.1.0
  - PyMySQL: v0.9.3
  - SQLAlchemy: v1.3.4
  - Scikit-learn: v0.21.2
  - Tensorflow: v1.13.1
  - Flask: v1.0.3
  - Flask-SQLAlchemy: v2.4.0
  - Flask-Cors: v3.0.8
  - Express JS: v4.17.1
  - Passport JS: v0.4.0
  - Passport-local: v1.0.0
  - Express-Handlebars: v3.1.0
  - JQuery: v3.3.1
  - Bootstrap: v4.3.1
  - JQuery-datatables: v1.19.1
  - Sequelize: v5.9.2
  - mysql: v2.17.1
  - mysql2: v1.6.5
  - body-parser: v1.19.1
  - bcrypt: v3.0.6
  - virtualenv: v16.5.0

- Follow below mentioned steps for complete installation:-
  - Follow the steps given in this link to install mysql server in ubuntu or follow the guidelines given here for other OS
  - Follow the links given for Python and NodeJS installation in your respective system.
  - Install Virtualenv in your system from links given above.
  - Then clone the Github Repository into your system
  - GO to the **Auto-Tagging-Algorithm-2.0-2019/Auto-tagger** folder and open shell/command prompt and run **npm install package.json** to install all node modules.
  - GO to the **Auto-Tagging-Algorithm-2.0-2019/REST_API** folder and open shell/command prompt and run **source bin/activate for Ubuntu** to activate virtual environment.
  - Then run **pip install -r requirements.txt**.

1.2 Software and Code

This is the Github Link for the project

The following snippets will walk you through the code for python:-

```
1  def make_features(year,conn):
2      '''
3      This is the function for making the feature table for
       training the model
4      Arguements: year(int): to check which year table is to be
       created/fetched
5                  conn(sqlalchemy connection object): Used by
       the pandas to fetch data from the database
6
7      We calculate features for the model training and analysis
       of the questions such as :
8      1. No. of People who solved the given question correctly
9      2. No. of People who solved the given question
       incorrectly
10     3. No. of People who did not attempt the given question.
11     4. Average Marks of People who solved the given question
       correctly.
12     5. Average Marks of People who solved the given question
       incorrectly.
13     6. Average Marks of People who did not attempt the given
       question.
14     7. Weighted Feature F1.
```

```
    8. Weighted Feature F2.
    '''
    query = "show tables like \"features_%d\""%year
    a = conn.execute(query)
    if(len(a.fetchall())==0):
        f1 = conn.execute(text("select question_id,count(answer_option) as correctly_answered from for_features where marks = 3 and marked =1 group by question_id"))
        f2 = conn.execute(text("select question_id,count(answer_option) as incorrectly_answered from for_features where marks = -1 and marked =1 group by question_id"))
        f3 = conn.execute(text("select question_id,count(answer_option) as not_answered from for_features where marked =0 group by question_id"))
        f4 = conn.execute(text("select question_id,avg(marks_scored) as avg_marks_correct from for_features where marks = 3 and marked =1 group by question_id"))
        f5 = conn.execute(text("select question_id,avg(marks_scored) as avg_marks_incorrect from for_features where marks = -1 and marked =1 group by question_id"))
        f6 = conn.execute(text("select question_id,avg(marks_scored) as avg_marks_na from for_features where marked =0 group by question_id"))

        feature1 = pd.DataFrame(f1,columns = f1.keys())
        feature2 = pd.DataFrame(f2,columns = f2.keys())
        feature3 = pd.DataFrame(f3,columns = f3.keys())
        feature4 = pd.DataFrame(f4,columns = f4.keys())
        feature5 = pd.DataFrame(f5,columns = f5.keys())
        feature6 = pd.DataFrame(f6,columns = f6.keys())
        weighted = weighted_features(conn)

        feature_list = [feature1,feature2,feature3,feature4,feature5,feature6]
        ques=[x for x in range(1,1801)]
        i=0
        features = pd.DataFrame(ques,columns = ["question_id"])
        while(i<=5):
            features = pd.merge(features,feature_list[i],on = "question_id")
            i = i+1
        features = pd.merge(features,weighted ,on = "question_id")
        features.to_sql("features_%d"%year,conn)
        return features
    else:
        table_name = "features_%d"%year
        table = conn.execute("select * from "+table_name)
```

```
48          return pd.DataFrame(table,columns = table.keys())
```

Listing 1.1: Making Features

This code snippet is for making the features from the tables in database.It makes features such as no of students who correctly answered the question etc.After making the features pandas store the table in database.(Right now only one database is connected so the year has to remain constant as 2018).

```
1  def f1(weights_map,marked,sum_weight):
2      '''
3      This function is for caluclating the weighted feature F1
4      Arguements: weights_map(dictionary): a dictionary which
       maps different students to the weights assigned to them
5                  marked(dictionary): to check who are all
       students who have answered a particular questions
6                  sum_weight(float): sum of all weights of the
       students
7      '''
8      feature_f1=[]
9      sum_ =0
10     for i in marked:
11         if(i==1):
12             for j in marked[i]:
13                 if(j[1]==3):
14                     sum_ = sum_+float(weights_map[j[0]])
15             feature_f1.append(sum_/float(sum_weight))
16
17         elif(i>=2):
18             sum_=0
19             for j in marked[i]:
20                 if(j[1]==3):
21                     sum_ = sum_+float(weights_map[j[0]])
22             feature_f1.append(sum_/float(sum_weight))
23     return feature_f1
24 def f2(weights_map,marks_map,marked,sum_weight):
25     '''
26     This function is for caluclating the weighted feature F1
27     Arguements: weights_map(dictionary): a dictionary which
       maps different students to the weights assigned to them
28                 marks_map(dictionary): a dictionary which
       maps students to their marks
29                 marked(dictionary): to check who are all
       students who have answered a particular questions
30                 sum_weight(float): sum of all weights of the
       students
31     '''
32     feature_f2=[]
33     sum_ =0
```

```
34      for i in marked:
35          if(i==1):
36              for j in marked[i]:
37                  if(j[1]!=3):
38                      sum_ = sum_+(float(weights_map[j[0]])*
    marks_map[j[0]])
39              feature_f2.append(sum_/float(sum_weight))
40
41          elif(i>=2):
42              sum_=0
43              for j in marked[i]:
44                  if(j[1]!=3):
45                      sum_ = sum_+(float(weights_map[j[0]])*
    marks_map[j[0]])
46              feature_f2.append(sum_/float(sum_weight))
47      return feature_f2
```

Listing 1.2: weighted features

This code snippet is for making the weighted features suggested in the referenced paper as follows: -

$$F_k^{(1)} = \frac{\sum_{s \in S} w_s f_k^{(1)}(D_k^{(s)})}{\sum_{s \in S} w_s} \tag{1.1}$$

$$F_k^{(2)} = \frac{\sum_{s \in S} w_s m_s f_k^{(2)}(D_k^{(s)})}{\sum_{s \in S} w_s} \tag{1.2}$$

For

$$w_s \tag{1.3}$$

in equations 1.1 and 1.2 we have taken a dictionary weights_map which maps students to their weights.

For

$$m_s \tag{1.4}$$

in equation 1.2 we have taken a dictionary marks_map which maps students to their weights.

```
1  from flask import Flask,render_template,request
2  from flask_sqlalchemy import SQLAlchemy
3  from Feature_functions import *
4  from SOM_functions import *
5  from accuracy_calc import *
```

```
6  from stats import *
7  import pandas as pd
8  import numpy as np
9  from sklearn.preprocessing import StandardScaler,MinMaxScaler
10 from sklearn.cluster import KMeans
11 import pickle
12 from flask_cors import CORS
13
14 app = Flask(__name__) #instantiating the app object
15 cors = CORS(app, resources={r"/*": {"origins": "*"}}) #used
       for handling calls from nodejs
16 ss = StandardScaler()
17 kmeans = KMeans(n_clusters = 3,max_iter = 900)#unstantiating
       KMeans object for clustering
18 app.config["SQLALCHEMY_DATABASE_URI"] = "mysql+pymysql://anuj
       :Anuj@21101998@localhost/auto_tagging_data"
19 app.config["SQLALCHEMY_BINDS"] = {
20     'stats' : 'mysql+pymysql://anuj:Anuj@21101998@localhost/
       stats',
21
22 } #configuring the app with the database
23 db = SQLAlchemy(app)
24 query = "select pre_tag from question_master"
25 a = db.engine.execute(query)
26 df = pd.DataFrame(a,columns = a.keys())
27 tags = list(df["pre_tag"])#Making the list for pre_tags
28 @app.before_request
29 def log_request():
30     print(request.headers)
31     return None
32 '''
33 This is the route for post request in the app.
34 Here the machine learning models will be trained on the fly
       and result will be stored in database
35 '''
36 @app.route("/",methods = ['GET','POST'])
37
38 def predict():
39     '''
40     This is the function for predicting the results from the
       features given throuh the form from node js.
41     This will return the message about accuracy score of the
       models.
42     The results for labels will be stored in stats database.
43     '''
44     names = request.get_json()
45     message = 'ok'
46     print(names)
47     print(names['features'][0])
```

```
48      if request.method == "POST":
49          print("inside post call")
50          features = make_features(int(names['features'][0]),db
        .engine)
51          data = features[names['features'][1:]]
52          data_pp = ss.fit_transform(data)
53          pred_kmeans = kmeans.fit_predict(data_pp)
54          pred_SOM = fit_predict(data_pp)
55          acc_kmeans,kmeans_tag = map_labels(np.array(tags),np.
        array(pred_kmeans))
56          acc_SOM,SOM_tag = map_labels(np.array(tags),np.array(
        pred_SOM))
57          print(acc_kmeans,acc_SOM)
58          to_statistics(int(names['features'][0]),kmeans_tag,
        SOM_tag,acc_kmeans,acc_SOM,tags,db.get_engine(app,"stats")
        )
59          if(acc_kmeans>acc_SOM):
60              for i in range(1,1801):
61                  query = "update question_master set post_tag
        = "+str(kmeans_tag[i-1])+" where id = "+str(i)
62                  db.engine.execute(query)
63          else:
64              for i in range(1,1801):
65                  query = "update question_master set post_tag
        = "+str(SOM_tag[i-1])+" where id = "+str(i)
66                  db.engine.execute(query)
67          message = "The Accuracy of Kmeans is "+str(acc_kmeans
        )+" and accuracy of SOM is "+str(acc_SOM)
68      return message
69  if __name__ == "__main__":
70      app.run(port = 5000,debug = True)
```

Listing 1.3: REST API

This code snippet is for the REST API made in flask. It uses all the functions made up until now to make features,predict and store results from the machine learning models.We configured two databases for the app. One is for the questions and other is for the statistics to be used by express app.It takes in input from NodeJS hosted form for features and uses pandas to extract the database.

```
1  import numpy as np
2  import pandas as pd
3  import matplotlib.pyplot as plt
4
5  def get_weights(total_data):
6      '''
7      function for initialize random values in the weight
```

```
    vectors for the neural network to be used.
    uses the no of features to initialize a vector.
    '''
    y = np.random.random()*(2.0/np.sqrt(total_data))
    return 0.5 - (1/np.sqrt(total_data)) + y

def compute_distance(w,x):
    '''
    function for computing the distance between the x(data)
    and w(Weight) vector
    takes in two arguments
    w: weights
    x: features
    '''
    distance=0
    for i in range(len(w)):
        distance = distance + (w[i] - x[i])*(w[i] - x[i])
    distance = np.sqrt(distance)
    return distance

def find_closest_to_x(W,x):
    '''
    function to calculate the closest x vectors to the w
    vectors
    takes in two arguments
    w: weights
    x: features
    '''

    w = W[0]
    dist = compute_distance(w,x)
    i = 0
    i_n = i
    for w_ in W:
        if compute_distance(w_,x)<dist:
            dist = compute_distance(w_, x)
            w = w_
            i_n = i
        i = i + 1
    return (w,i_n)
def fit_predict(data):
    W=[]
    n_clusters = 3
    features = len(data[0])
    total_data = len(data)
    for i in range(n_clusters):
        W.append(list())
        for j in range(features):
            W[i].append(get_weights(total_data) * 0.5)
```

```
    la = 0.3    #    coefficient
    dla = 0.05  #
    '''
    This code applies the training process defined above for
   every data point given in the dataset.
    We run a loop till la is equal to 0. In that we take 10
   iterations and find closest datapoint ot the neuron and
   then
    updates the value of the wn as in the above equation.
    '''
    while la >= 0:
        for k in range(10):
            for x in data:
                wm = find_closest_to_x(W, x)[0]
                for i in range(len(wm)):
                    wm[i] = wm[i] + la * (x[i] - wm[i])
        la = la - dla
    prediction=[]
    for x in data:
        i_n = find_closest_to_x(W,x)[1]
        prediction.append(i_n)
    return prediction
```

Listing 1.4: Self Oganizing Map's Functions

This code snippet shows the workflow of making the Self Organizing Map which is made on the fly.We have function for intializing the weights,calculating the distance,calculating the shortest distance and for the main algorithm which changes the shape of the map according to the distribution of the data.

```
router.get('/login',function(req,res){
    res.render('login')
  });
router.post('/auth',auth_controller.login)
router.get('/sorry',(req,res)=>{
    res.redirect('/login');
  })
router.get('/',function(req,res){
  if (req.session.loggedin) {
      if(req.session.user.role=='admin'){console.log("Inside
   admin");return res.redirect('/admin')}
      else{console.log("Inside user");return res.redirect('/
   user')}
      } else {
          res.redirect('/login');
      }
});

router.get('/logout', function(req, res, next) {
```

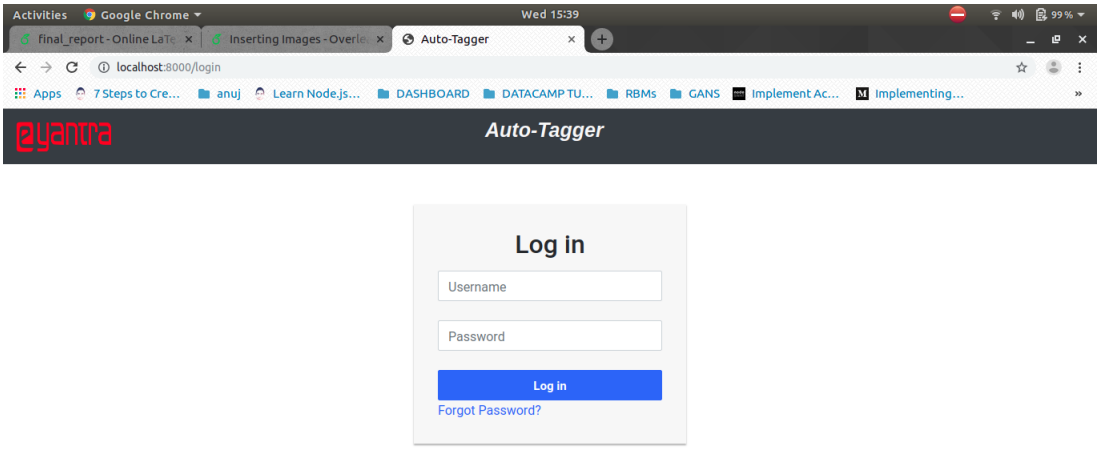
```
    if (req.session) {
      // delete session object
      req.session.destroy(function(err) {
        if(err) {
          return next(err);
        } else {
          return res.redirect('/login');
        }
      });
    }
  });
```

Listing 1.5: Login Routes

```
login:
        (req, res, next)=> {
            console.log('Inside POST /login callback')
            auth.authenticate('local-login', (err, user, info
    ) => {
              if(info) {return res.send(info.message)}
              if (err) { return next(err); }
              if (!user) { return res.redirect('/sorry'); }
              req.login(user, (err) => {
                console.log(user);
                if (err) { req.session.destroy();
                  return next(err);
                }
                req.session.loggedin = true;
                req.session.user = user;
                // req.session.role=
                return res.redirect('/');
            })
            })(req, res, next);
          }
```

Listing 1.6: Auth$_{C}$*ontroller*

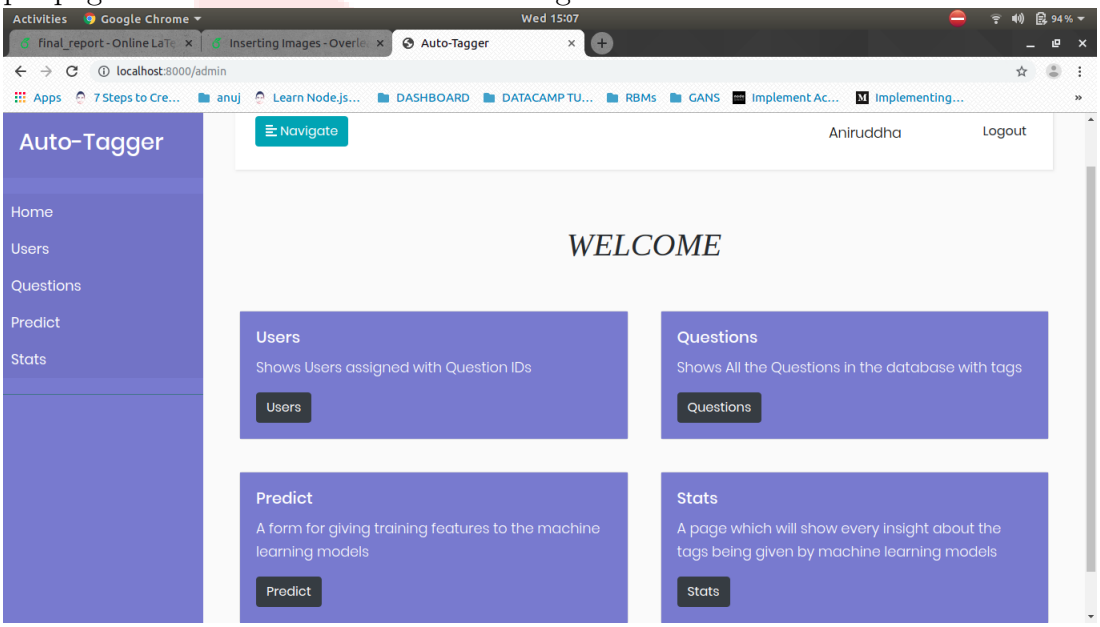
The above two code snippets walk you through the workflow of authentication and authorization.This all code handles user login and checks if the user is logged in or not and also checks whether the person is normal user or admin.




```
router.get('/',admin_middleware,admin_controller.home)
//admin_controller.home
home: (req,res)=>{res.render("./admin/home",{username:req.
    session.user.name})},
```

Listing 1.7: Admin Home

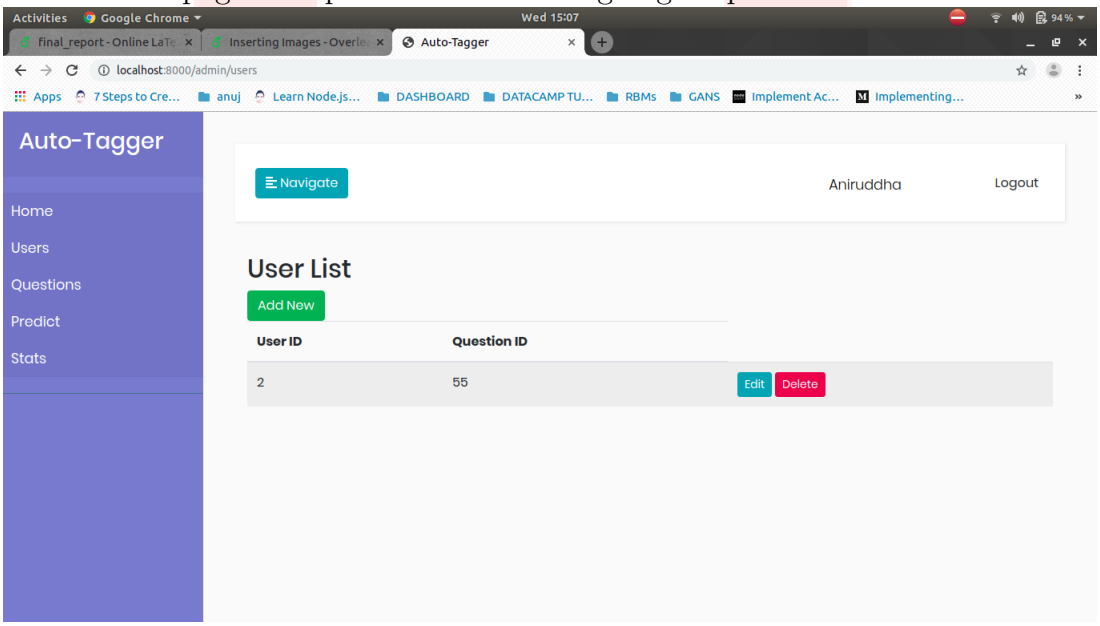
The above code snippet is for the home page route for the admin.It is a simple page which renders a welcome message to the admin.



```
//route for showing the user data
router.get('/users',admin_middleware,admin_controller.users);
//admin_controller.users
users: (req, res) => {
        let sql = "SELECT * FROM user_question";
        let query = conn.query(sql, (err, results) => {
            if (err) throw err;
            res.render('./admin/users', {
                results: results
            });
        });
    },
```

Listing 1.8: Users List

The above code snippet walks you through the backend of the users page at admin. This page is responsible for the assigning of questions.

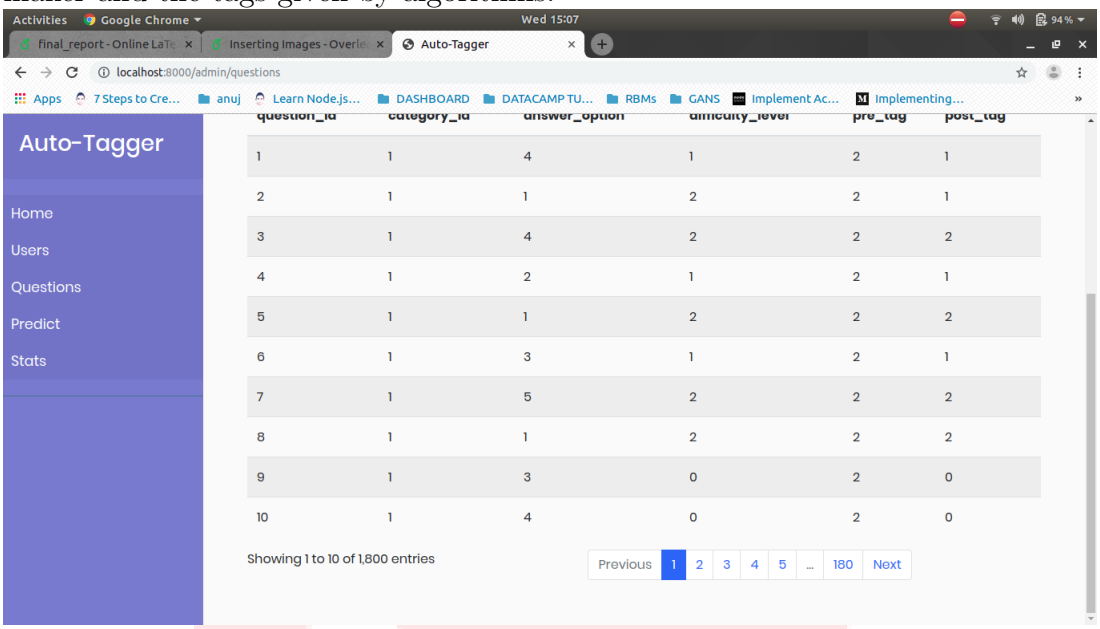



```
//route for ahowing the question
router.get('/questions',admin_middleware,admin_controller.
    questions)
//admin_controller.questions
questions: (req, res) => {
        sequelize2.query("select * from question_master").
    then((results) => {
            res.render('./admin/admin_ques', { results:
    results[0] });
        })
```

```
},
```

Listing 1.9: Questions list

The above code snippet walks you through the backend of the questions page.Here the admin can see the questions with the tags given by question maker and the tags given by algorithms.



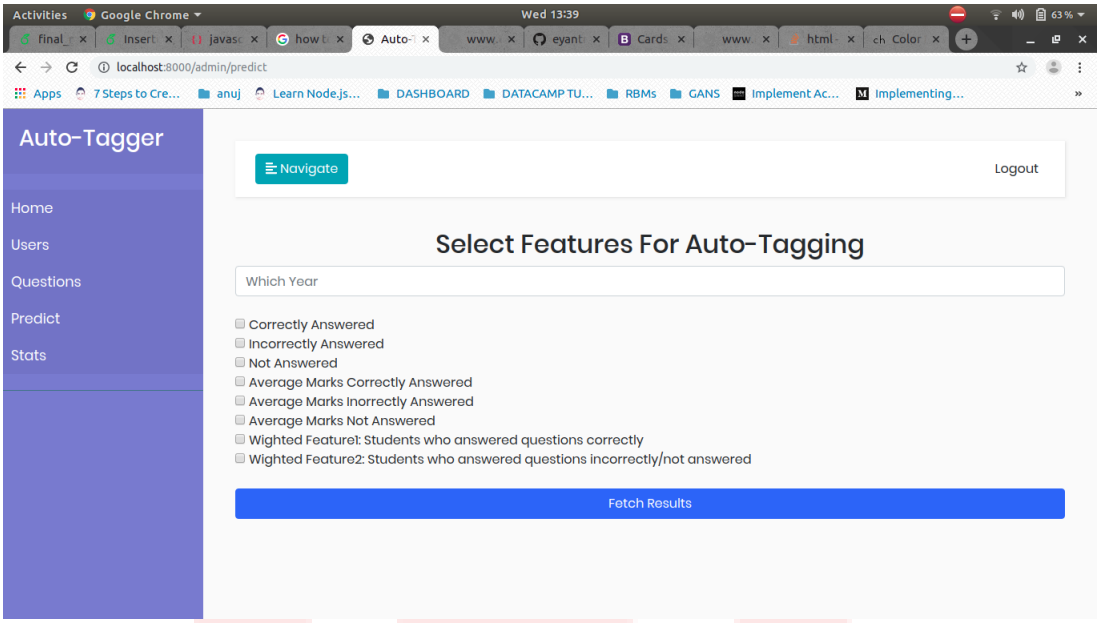

```
//route for showing the form
router.get('/predict',admin_middleware,admin_controller.
    predict);
//admin_controller.predict
predict: (req, res) => {
        var message = ""
        message = req.query.message
        res.render("./admin/Predict", { message: message });
},
//route for getting the results from the flask REST API via
    form data
router.post('/predict',admin_middleware,admin_controller.
    predict_request);
//admin_controller.predict_request
predict_request: (req, res) => {

        var features = [];
        var message = "Accuraacy";
        var dummy = [req.body.year,
        req.body.correctly_answered,
        req.body.incorretly_answered,
```

```
19          req.body.not_answered,
20          req.body.avg_marks_correctly_answered,
21          req.body.avg_marks_incorrectly_answered,
22          req.body.avg_marks_not_answered,
23          req.body.F1,
24          req.body.F2];
25          for (var i = 0; i < dummy.length; i++) {
26              if (dummy[i] != undefined) {
27                  features = features.concat(dummy[i]);
28              }
29          }
30          if (features.length === 1) {
31              res.render("./admin/Predict", { message: "Please
    select atleat 1 feature" });
32          }
33          else {
34
35              axios.post("http://127.0.0.1:5000/", { features
    }).then(function (response) {
36                  console.log(response.data);
37                  messsage = response.data;
38
39              }).catch((err) => {
40                  console.log(err.message)
41                  return res.redirect('back')
42              }
43              )
44              return res.render("./admin/Predict", { message: "
    Please head to stats page for insights in result" })
45          }
46  },
```

Listing 1.10: Prediction Form

The above code walks you through the backend of the Predict page which is responsible for communicating with the flask API at backend for retrieving the results from Machine Learning Models.

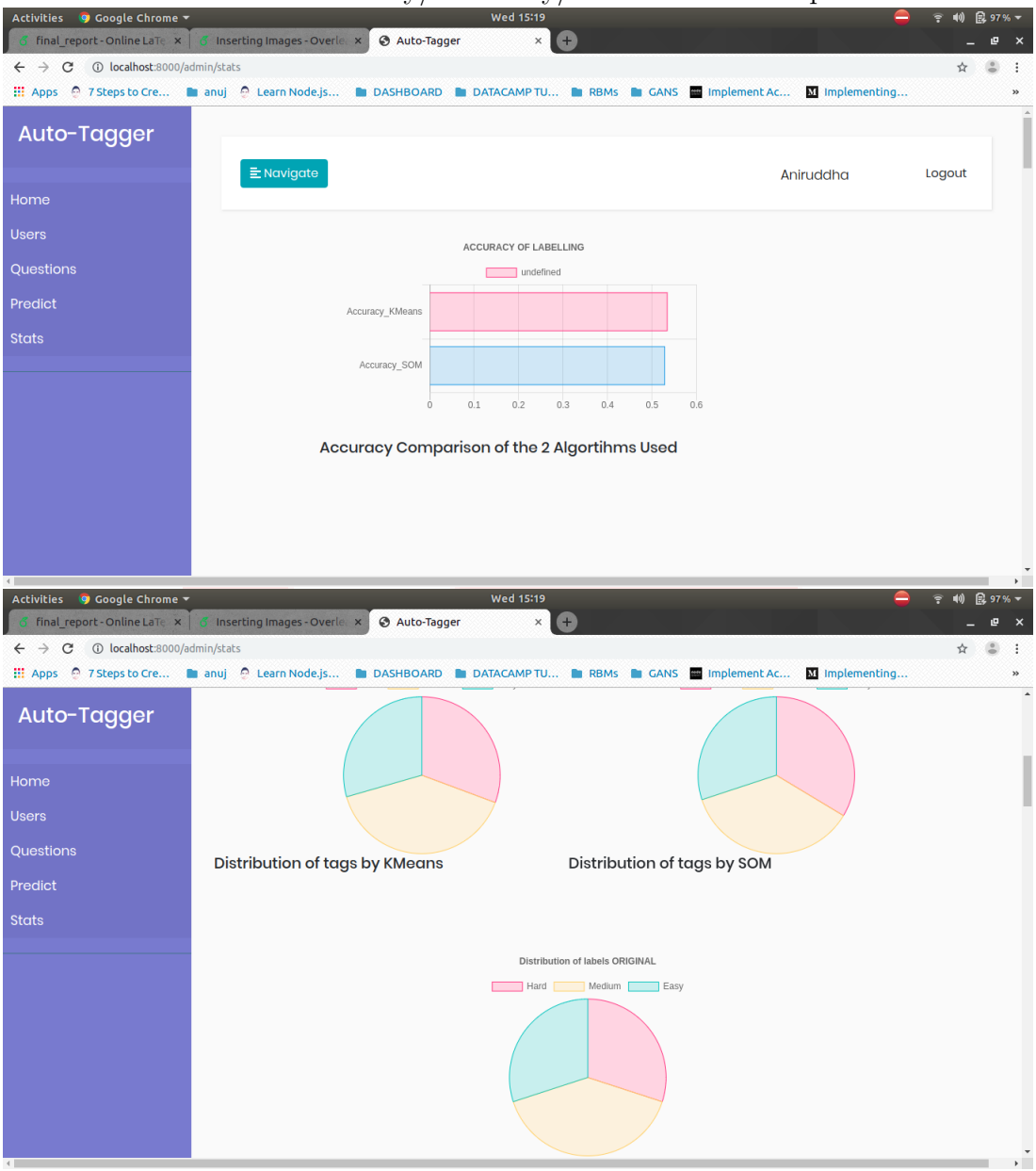


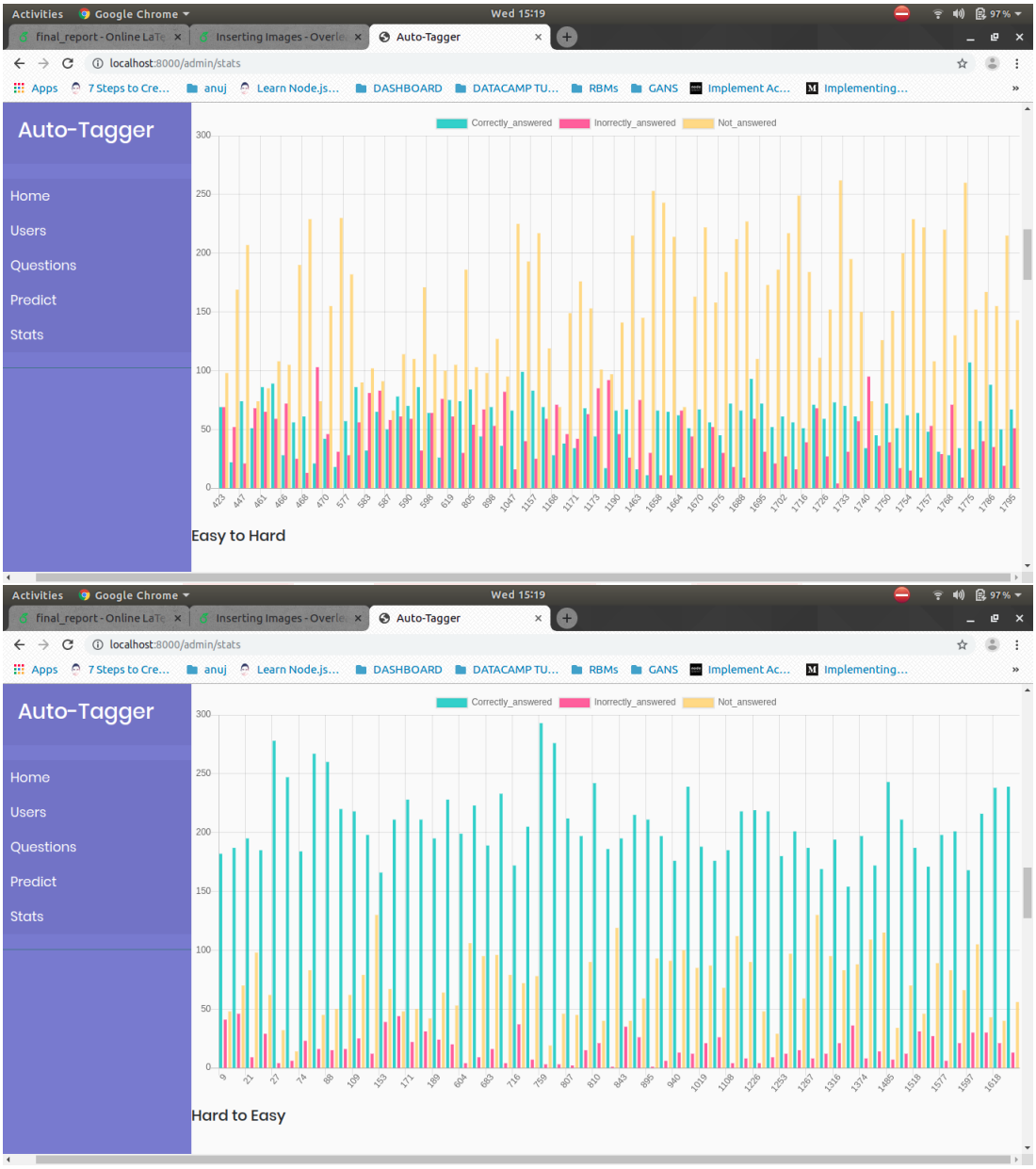

```
1  //route for showing the statistics
2  router.get('/stats',admin_middleware,admin_controller.stats);
3  //admin_controller.stats
4  stats:(req,res)=>{
5          sequelize1.query("select * from stats_2018").then(([
       results, metadata]) => {
6            // Results will be an empty array and metadata will
        contain the number of affected rows.
7            let array=[]
8            return results;
9            //console.log(array);
10
11          // console.log(metadata[0].KMeans_labels);
12         }).then(results=>{
13           sequelize2.query("select * from features_2018").
       then(([stats, metadata]) => {
14             // Results will be an empty array and metadata
       will contain the number of affected rows.
15             let array=[]
16             res.render("./admin/stats",{data: JSON.stringify(
       results),stats: JSON.stringify(stats)})
17           })
18
19       })
20
21   }
```

Listing 1.11: Stats Route

The above code walks you through the backend of the stats page which shows

stats about model's performance and how the questions are tagged.It shows the accuracy of the models used,the distribution of tags over questions predicted by models and graphs for showing how the tags have changed with the no of students who correctly/incorretly/not answered the questions.



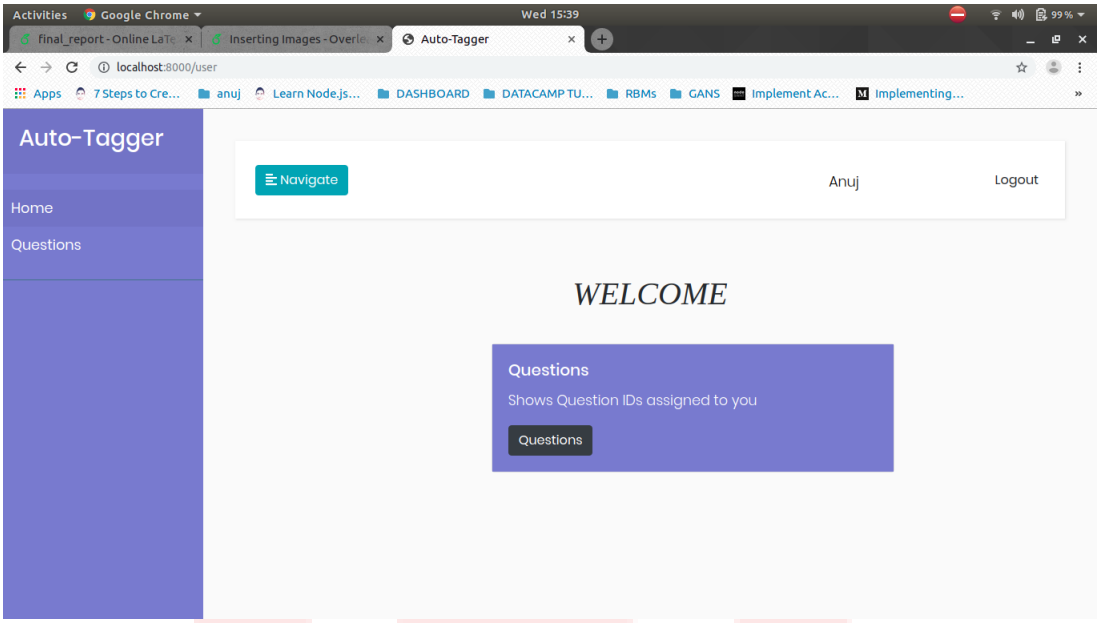

Activities
Google Chrome
Wed 15:19
Auto-Tagger
localhost:8000/admin/stats
Auto-Tagger
Home
Users
Questions
Predict
Stats
Correctly_answered
Inorrectly_answered
Not_answered
Easy to Hard
Hard to Easy


Activities
Google Chrome
Wed 15:19
Auto-Tagger
localhost:8000/admin/stats
Auto-Tagger
Home
Users
Questions
Predict
Stats
Correctly_answered
Inorrectly_answered
Not_answered
Easy to Medium
Medium to Easy


```
//home route
router.get('/',user_middleware,user_controller.home)
//user_controller.home
home: (req,res)=>{res.render("./user/home_user",{username:req
    .session.user.name})},
```

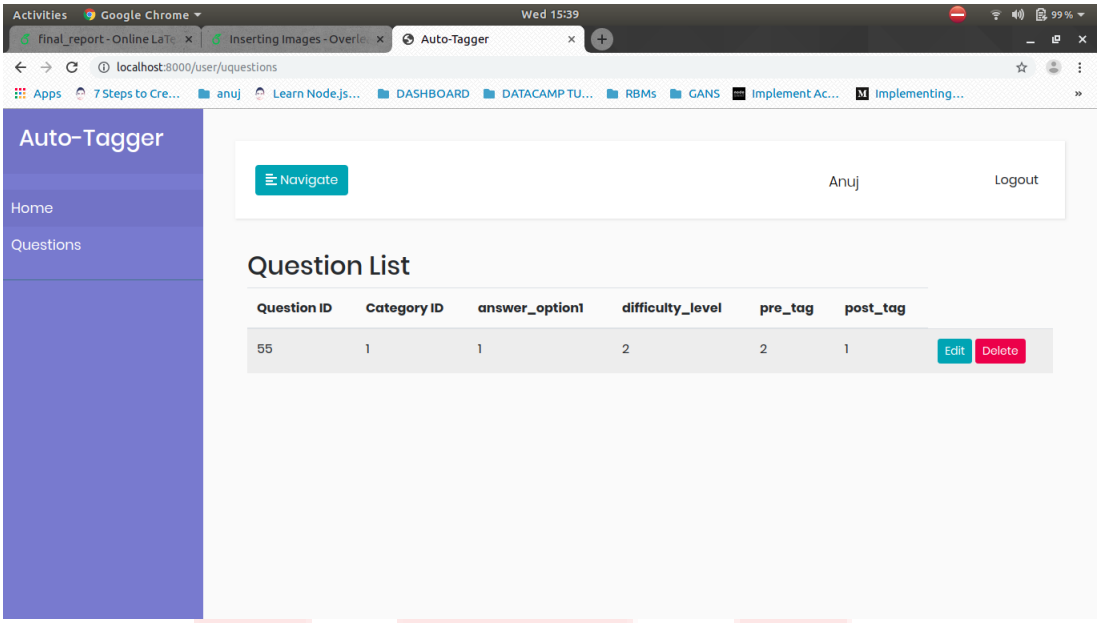
Listing 1.12: User Home Route

The above code walks you through the backend for the user home page.




```
1  //route for showing data
2  router.get('/uquestions',user_middleware,user_controller.
       questions);
3  //user_controller.questions
4  questions:(req, res) => {
5          let sql = `select question_id from user_question
       where user_id = ${req.session.user.userid}`;
6              let query = sequelize3.query(sql).then((results0)
       =>{
7                var results = []
8                for(var i= 0;i<results0[0].length;++i){
9                  results.push(results0[0][i].question_id)
10               }
11               question_master.findAll({
12                 where:{
13                   id:results
14                 }
15               }).then((results1)=>{
16                 res.render("./user/questions",{results:
       results1})
17               })
18             })
19       }
```

The above code walks you through the backend for questions page at user side where he/she can edit/delete the questions assigned.



1.3 Use and Demo

Final Setup Image

User Instruction for demonstration

Youtube Link of demonstration video

1.4 Future Work

The following can be tried in future to make the project better:-

- Discovering about other algorithms like Autoencoders ans other deep learning algorithms.
- Making the frontend in React/Vue or any other frontend library for better design.
- Imporve the performance of the REST API to scale better.

1.5 Bug report and Challenges

BUG REPORT

- Since the App is based on one database which belongs to questions of 2018 the predict form will work only when 2018 as year will be given.
- The Database which stores the User ID and Question ID does not have a primary key. So redundancy can occur there.
- If as admin you entered User ID which is not in the User's Database no check functionality will be performed.

Challenges Faced

- Understanding the whole database to find out important tables.There were 10 tables in the database but only **4** of them were needed.
- Making Features from different tables and import it into python.This was the most important step as it forms the base for our model.**Good data promises good models.**
- Calculating the features from research paper was time consuming. Firstly we have to **calculate weights for each student and also map each question to the students who answered them** and then calculate the features.Shown below are equations :-

$$F_k^{(1)} = \frac{\sum_{s \in S} w_s f_k^{(1)}(D_k^{(s)})}{\sum_{s \in S} w_s} \tag{1.5}$$

$$F_k^{(2)} = \frac{\sum_{s \in S} w_s m_s f_k^{(2)}(D_k^{(s)})}{\sum_{s \in S} w_s} \tag{1.6}$$

- Very little knowledge of web development was a hurdle when making the Web App.

Bibliography

[1] Ad Kamerman and Leo Monteban, *WaveLAN-II: A High-Performance Wireless LAN for the Unlicensed band*, 1997.