

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 paper Auto-Tagging for Massive Online Selection Tests: Machine Learning to the Rescue.
- Used data points from database with DB connector sqlalchemy ORM tool for python.
- Calculated all the features including weighted features given in paper.
- K-Means Clustering algorithm was applied on the normal features and accuracy was calculated.
- K-Means Clustering algorithm was applied on the normal features 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.



## 1.1. SOFTWARE USED

---

- 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



## 1.1. SOFTWARE USED

---

- 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](#)



## 1.2. SOFTWARE AND CODE

- 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.
  - First activate a virtual environment and go to the **Auto-Tagging-Algorithm-2.0-2019/RESTAPI** folder and open shell/command prompt and 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
4     training the model
5     Arguments: year(int): to check which year table is to be
6     created/fetched
7     conn(sqlalchemy connection object): Used by
8     the pandas to fetch data from the database
9
10    We calculate features for the model training and analysis
11    of the questions such as :
12    1. No. of People who solved the given question correctly
13    2. No. of People who solved the given question
14    incorrectly
15    3. No. of People who did not attempt the given question.
16    4. Average Marks of People who solved the given question
17    correctly.
18    5. Average Marks of People who solved the given question
19    incorrectly.
20    6. Average Marks of People who did not attempt the given
21    question.
22    7. Weighted Feature F1.
23    8. Weighted Feature F2.
```



## 1.2. SOFTWARE AND CODE

```
16     """
17     query = "show tables like \"features_%d\""%year
18     a = conn.execute(query)
19     if(len(a.fetchall())==0):
20         f1 = conn.execute(text("select question_id,count(
21             answer_option) as correctly_answered from for_features
22             where marks = 3 and marked =1 group by question_id"))
23         f2 = conn.execute(text("select question_id,count(
24             answer_option) as incorrectly_answered from for_features
25             where marks = -1 and marked =1 group by question_id"))
26         f3 = conn.execute(text("select question_id,count(
27             answer_option) as not_answered from for_features where
28             marked =0 group by question_id"))
29         f4 = conn.execute(text("select question_id,avg(
30             marks_scored) as avg_marks_correct from for_features where
31             marks = 3 and marked =1 group by question_id"))
32         f5 = conn.execute(text("select question_id,avg(
33             marks_scored) as avg_marks_incorrect from for_features
34             where marks = -1 and marked =1 group by question_id"))
35         f6 = conn.execute(text("select question_id,avg(
36             marks_scored) as avg_marks_na from for_features where
37             marked =0 group by question_id"))
38
39         feature1 = pd.DataFrame(f1,columns = f1.keys())
40         feature2 = pd.DataFrame(f2,columns = f2.keys())
41         feature3 = pd.DataFrame(f3,columns = f3.keys())
42         feature4 = pd.DataFrame(f4,columns = f4.keys())
43         feature5 = pd.DataFrame(f5,columns = f5.keys())
44         feature6 = pd.DataFrame(f6,columns = f6.keys())
45         weighted = weighted_features(conn)
46
47         feature_list = [feature1,feature2,feature3,feature4,
48             feature5,feature6]
49         ques=[x for x in range(1,1801)]
50         i=0
51         features = pd.DataFrame(ques,columns = ["question_id"
52             ])
53         while(i<=5):
54             features = pd.merge(features,feature_list[i],on =
55                 "question_id")
56             i = i+1
57             features = pd.merge(features,weighted ,on = "
58                 question_id")
59             features.to_sql("features_%d"%year,conn)
60             return features
61         else:
62             table_name = "features_%d"%year
63             table = conn.execute("select * from "+table_name)
```



## 1.2. SOFTWARE AND CODE

```
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 calculating the weighted feature F1
4     Arguments: weights_map(dictionary): a dictionary which
5     maps different students to the weights assigned to them
6     marked(dictionary): to check who are all
7     students who have answered a particular questions
8     sum_weight(float): sum of all weights of the
9     students
10    '''
11    feature_f1=[]
12    sum_ =0
13    for i in marked:
14        if(i==1):
15            for j in marked[i]:
16                if(j[1]==3):
17                    sum_ = sum_+float(weights_map[j[0]])
18                    feature_f1.append(sum_/float(sum_weight))
19
20            elif(i>=2):
21                sum_ =0
22                for j in marked[i]:
23                    if(j[1]==3):
24                        sum_ = sum_+float(weights_map[j[0]])
25                        feature_f1.append(sum_/float(sum_weight))
26    return feature_f1
27 def f2(weights_map, marks_map, marked, sum_weight):
28     '''
29     This function is for calculating the weighted feature F1
30     Arguments: weights_map(dictionary): a dictionary which
31     maps different students to the weights assigned to them
32     marks_map(dictionary): a dictionary which
33     maps students to their marks
34     marked(dictionary): to check who are all
35     students who have answered a particular questions
36     sum_weight(float): sum of all weights of the
37     students
38    '''
39    feature_f2=[]
40    sum_ =0
```

## 1.2. SOFTWARE AND CODE

```
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: -

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

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

For

$$w_s \quad (1.3)$$

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

For

$$m_s \quad (1.4)$$

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 *
```





## 1.2. SOFTWARE AND CODE

```
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])
```



## 1.2. SOFTWARE AND CODE

```
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
```



## 1.2. SOFTWARE AND CODE

```

    vectors for the neural network to be used.
    uses the no of features to initialize a vector.
    '''
8     y = np.random.random()*(2.0/np.sqrt(total_data))
9
10    return 0.5 - (1/np.sqrt(total_data)) + y
11
12
13    def compute_distance(w,x):
14        '''
15        function for computing the distance between the x(data)
16        and w(Weight) vector
17        takes in two arguments
18        w: weights
19        x: features
20        '''
21        distance=0
22        for i in range(len(w)):
23            distance = distance + (w[i] - x[i])*(w[i] - x[i])
24        distance = np.sqrt(distance)
25        return distance
26
27    def find_closest_to_x(W,x):
28        '''
29        function to calculate the closest x vectors to the w
30        vectors
31        takes in two arguments
32        w: weights
33        x: features
34        '''
35        w = W[0]
36        dist = compute_distance(w,x)
37        i = 0
38        i_n = i
39        for w_ in W:
40            if compute_distance(w_,x)<dist:
41                dist = compute_distance(w_, x)
42                w = w_
43                i_n = i
44            i = i + 1
45        return (w,i_n)
46
47    def fit_predict(data):
48        W=[]
49        n_clusters = 3
50        features = len(data[0])
51        total_data = len(data)
52        for i in range(n_clusters):
53            W.append(list())
54            for j in range(features):
55                W[i].append(get_weights(total_data) * 0.5)
```



## 1.2. SOFTWARE AND CODE

```
54     la = 0.3      #      coefficient
55     dla = 0.05    #
56     '''
57     This code applies the training process defined above for
58     every data point given in the dataset.
59     We run a loop till la is equal to 0. In that we take 10
60     iterations and find closest datapoint of the neuron and
61     then
62     updates the value of the wn as in the above equation.
63     '''
64     while la >= 0:
65         for k in range(10):
66             for x in data:
67                 wm = find_closest_to_x(W, x)[0]
68                 for i in range(len(wm)):
69                     wm[i] = wm[i] + la * (x[i] - wm[i])
70             la = la - dla
71     prediction=[]
72     for x in data:
73         i_n = find_closest_to_x(W,x)[1]
74         prediction.append(i_n)
75     return prediction
```

Listing 1.4: Self Organizing 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 initializing 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.

```
1  router.get('/login',function(req,res){
2      res.render('login')
3  });
4  router.post('/auth',auth_controller.login)
5  router.get('/sorry',(req,res)=>{
6      res.redirect('/login');
7  })
8  router.get('/',function(req,res){
9      if (req.session.loggedin) {
10         if(req.session.user.role=='admin'){console.log("Inside
11         admin");return res.redirect('/admin')}
12         else{console.log("Inside user");return res.redirect('/
13         user')}
14     } else {
15         res.redirect('/login');
16     }
17 });
18 router.get('/logout', function(req, res, next) {
```



## 1.2. SOFTWARE AND CODE

```
18   if (req.session) {
19       // delete session object
20       req.session.destroy(function(err) {
21           if(err) {
22               return next(err);
23           } else {
24               return res.redirect('/login');
25           }
26       });
27   }
28   });
```

Listing 1.5: Login Routes

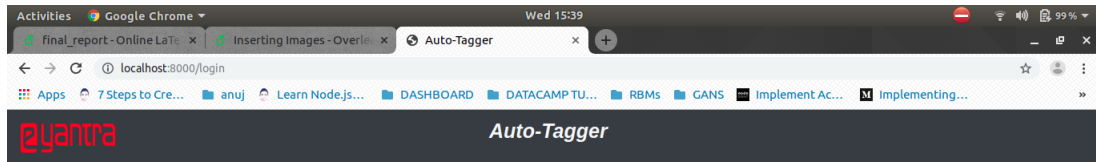
```
1 login:
2   (req, res, next)=> {
3       console.log('Inside POST /login callback')
4       auth.authenticate('local-login', (err, user, info
5   ) => {
6       if(info) {return res.send(info.message)}
7       if (err) { return next(err); }
8       if (!user) { return res.redirect('/sorry'); }
9       req.login(user, (err) => {
10           console.log(user);
11           if (err) { req.session.destroy();
12               return next(err);
13           }
14           req.session.loggedin = true;
15           req.session.user = user;
16           // req.session.role=
17           return res.redirect('/');
18       })
19       })(req, res, next);
20   }
```

Listing 1.6: AuthController

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.



## 1.2. SOFTWARE AND CODE



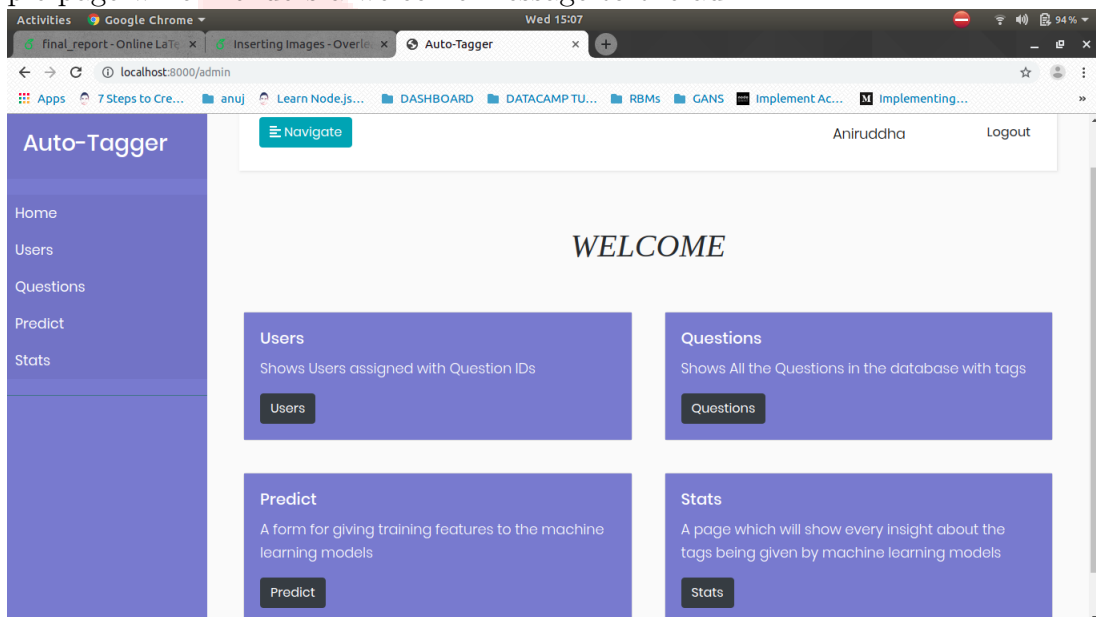
### Log in

  
  
Log in  
[Forgot Password?](#)

```
1 router.get('/', admin_middleware, admin_controller.home)
2 //admin_controller.home
3 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.



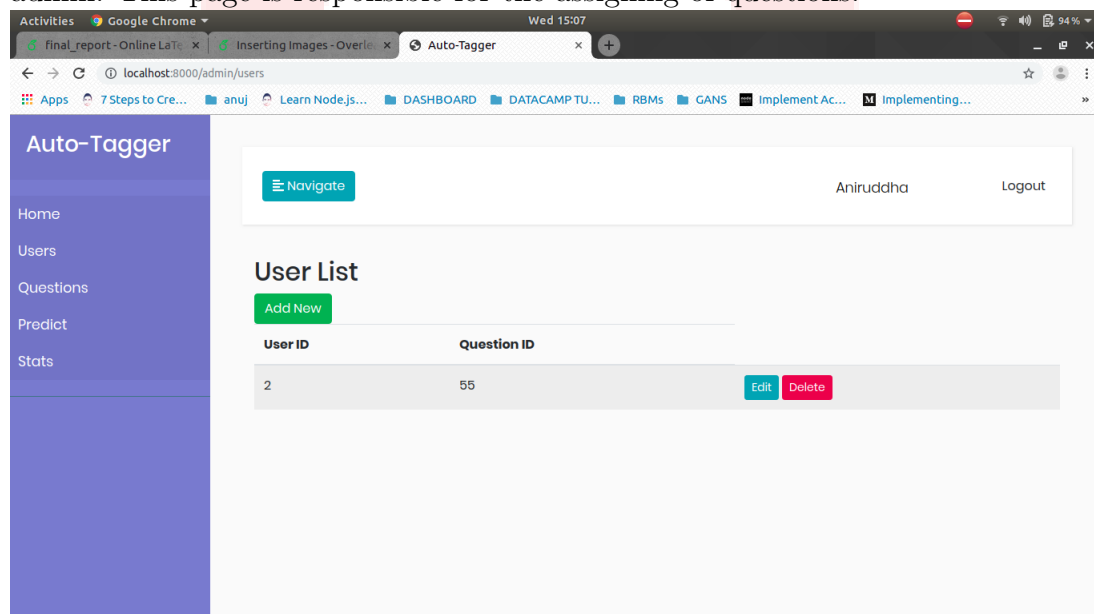


## 1.2. SOFTWARE AND CODE

```
1 //route for showing the user data
2 router.get('/users',admin_middleware,admin_controller.users);
3 //admin_controller.users
4 users: (req, res) => {
5     let sql = "SELECT * FROM user_question";
6     let query = conn.query(sql, (err, results) => {
7         if (err) throw err;
8         res.render('./admin/users', {
9             results: results
10        });
11    });
12 }
```

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.



```
1 //route for ahowing the question
2 router.get('/questions',admin_middleware,admin_controller.
  questions)
3 //admin_controller.questions
4 questions: (req, res) => {
5     sequelize2.query("select * from question_master").
6     then((results) => {
7         res.render('./admin/admin_ques', { results:
8             results[0] });
9     })
10 }
```



## 1.2. SOFTWARE AND CODE

8 },

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.

question_id	category_id	answer_option	difficulty_level	pre_tag	post_tag
1	1	4	1	2	1
2	1	1	2	2	1
3	1	4	2	2	2
4	1	2	1	2	1
5	1	1	2	2	2
6	1	3	1	2	1
7	1	5	2	2	2
8	1	1	2	2	2
9	1	3	0	2	0
10	1	4	0	2	0

Showing 1 to 10 of 1,800 entries

Previous 1 2 3 4 5 ... 180 Next

```
1 //route for showing the form
2 router.get('/predict',admin_middleware,admin_controller.
  predict);
3 //admin_controller.predict
4 predict: (req, res) => {
5     var message = ""
6     message = req.query.message
7     res.render("./admin/Predict", { message: message });
8 },
9 //route for getting the results from the flask REST API via
  form data
10 router.post('/predict',admin_middleware,admin_controller.
  predict_request);
11 //admin_controller.predict_request
12 predict_request: (req, res) => {
13
14     var features = [];
15     var message = "Accuraacy";
16     var dummy = [req.body.year,
17     req.body.correctly_answered,
18     req.body.incorretly_answered,
```





## 1.2. SOFTWARE AND CODE

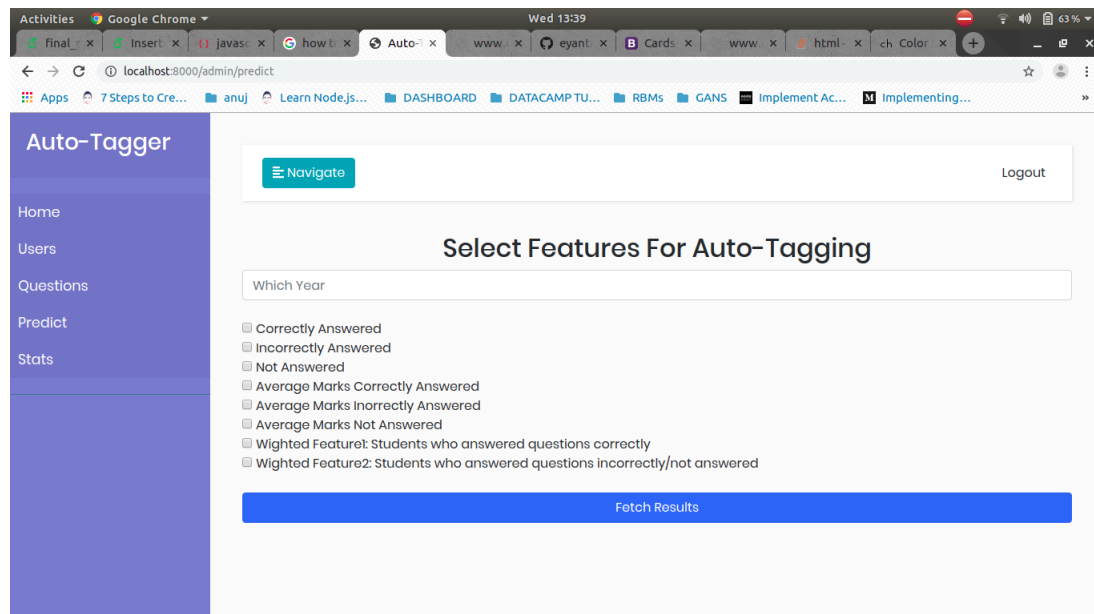
```
19     req.body.not_answered,
20     req.body.avg_marks_correctly_answered,
21     req.body.avg_marks_incorrectly_answered,
22     req.body.avg_marks_not_answered,
23     req.body.F1,
24     req.body.F2];
25     for (var i = 0; i < dummy.length; i++) {
26         if (dummy[i] != undefined) {
27 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



## 1.2. SOFTWARE AND CODE



```
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(([
6         results, metadata]) => {
7         // Results will be an empty array and metadata will
8         // contain the number of affected rows.
9         let array=[]
10        return results;
11        //console.log(array);
12
13        // console.log(metadata[0].KMeans_labels);
14    }).then(results=>{
15        sequelize2.query("select * from features_2018").
16        then([stats, metadata]) => {
17            // Results will be an empty array and metadata
18            // will contain the number of affected rows.
19            let array=[]
20            res.render("./admin/stats",{data: JSON.stringify(
21                results),stats: JSON.stringify(stats)})
22        })
23    })
24 }
```

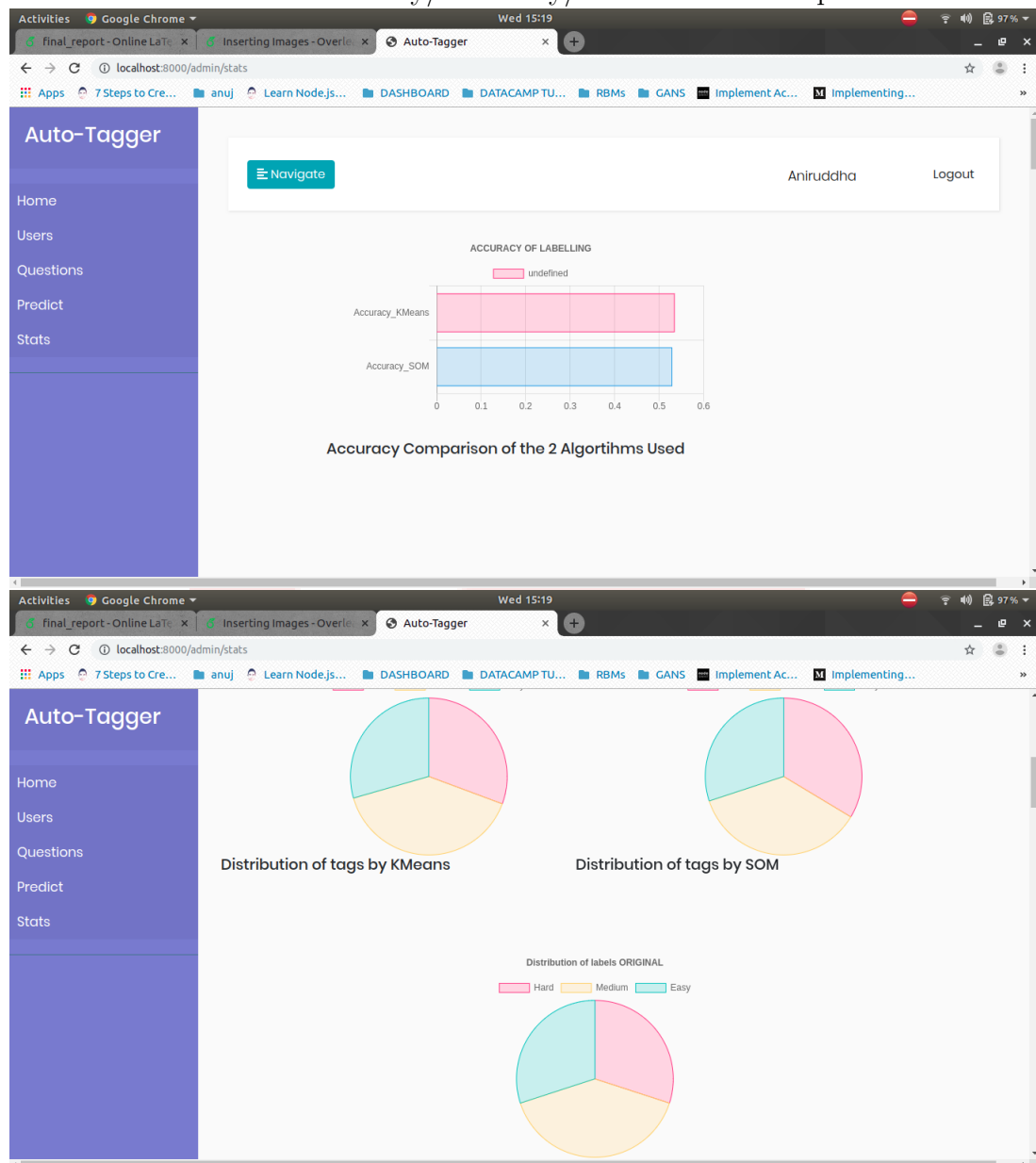
Listing 1.11: Stats Route

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



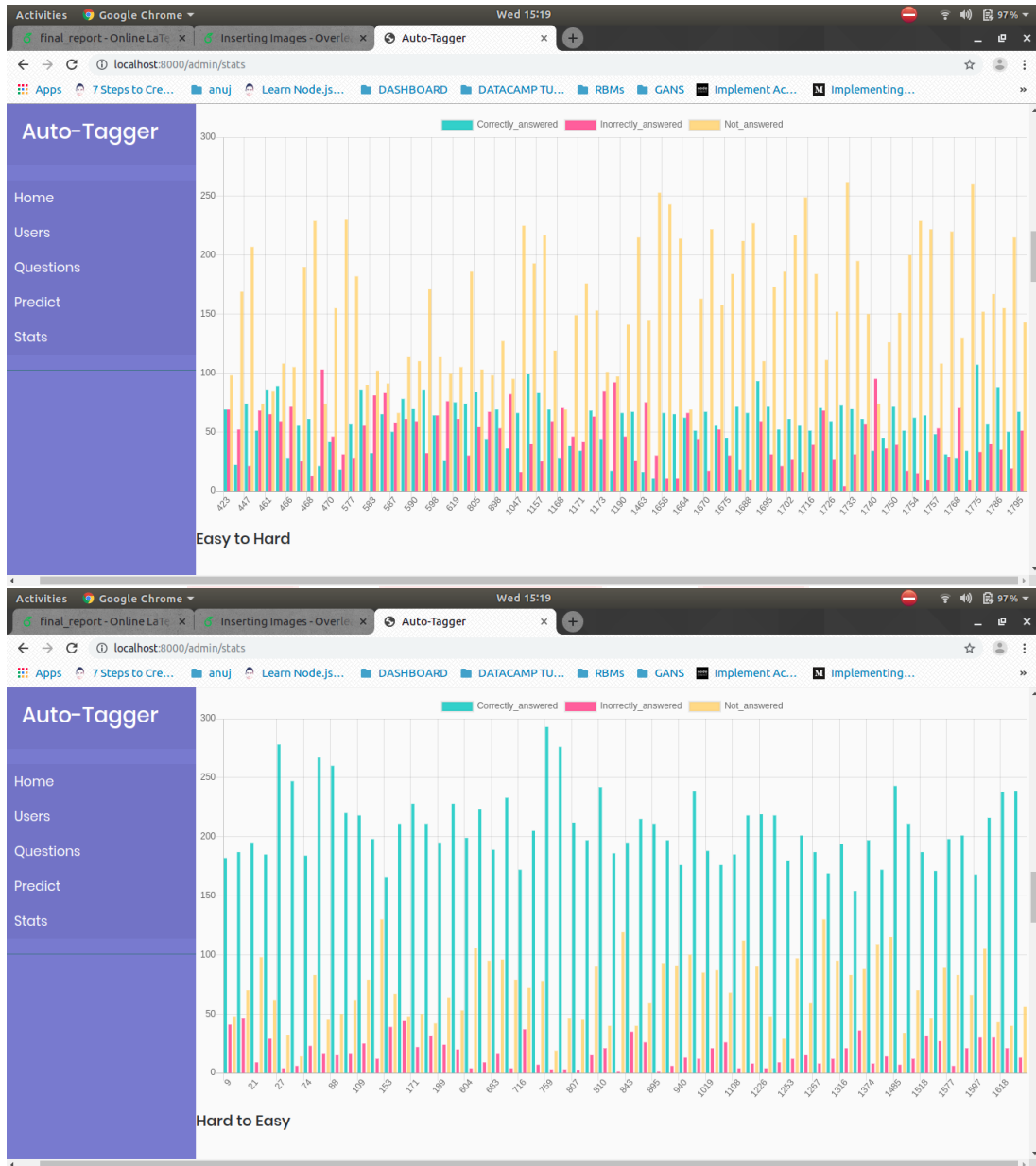
## 1.2. SOFTWARE AND CODE

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/incorrectly/not answered the questions.





## 1.2. SOFTWARE AND CODE



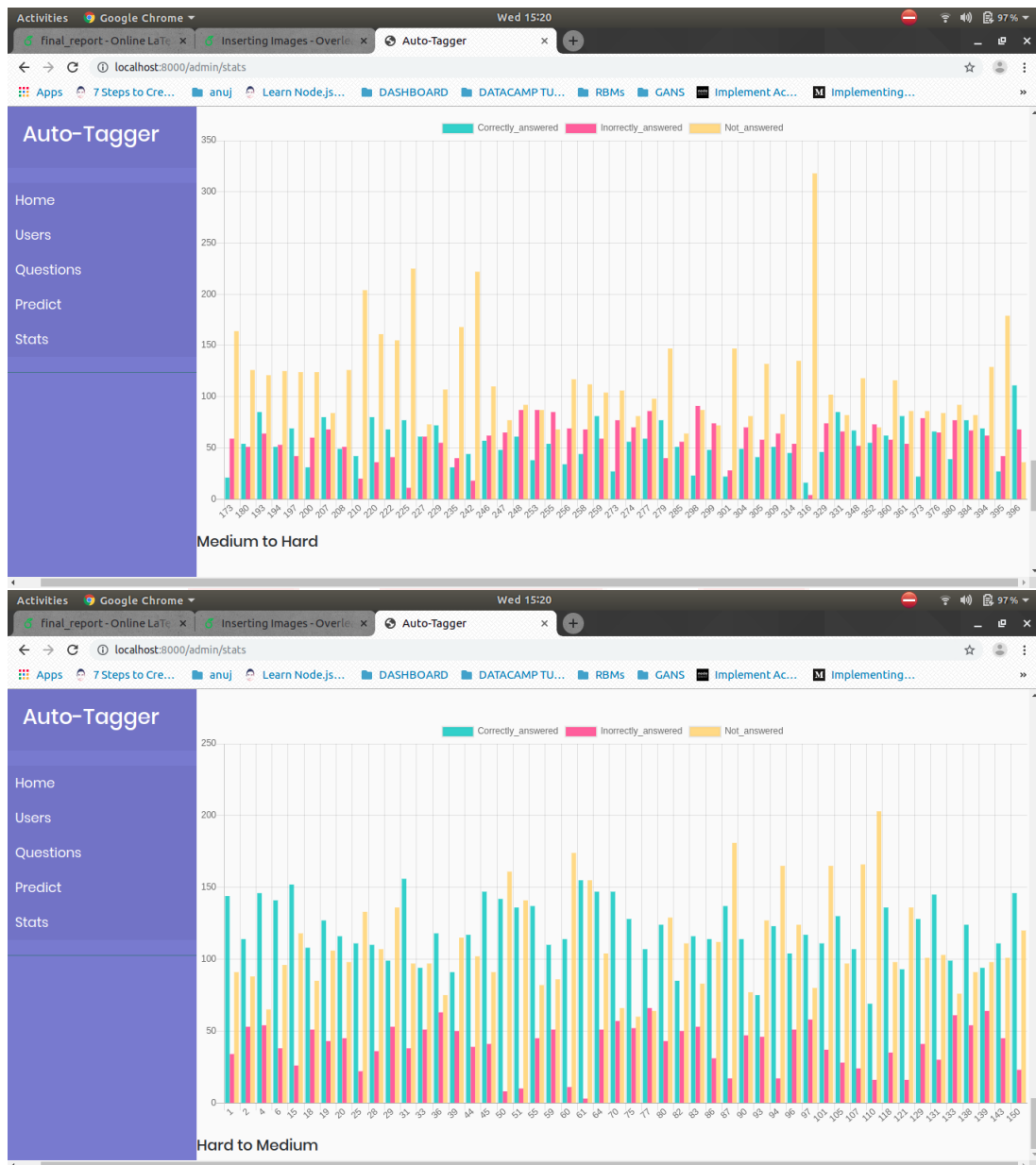


## 1.2. SOFTWARE AND CODE





## 1.2. SOFTWARE AND CODE



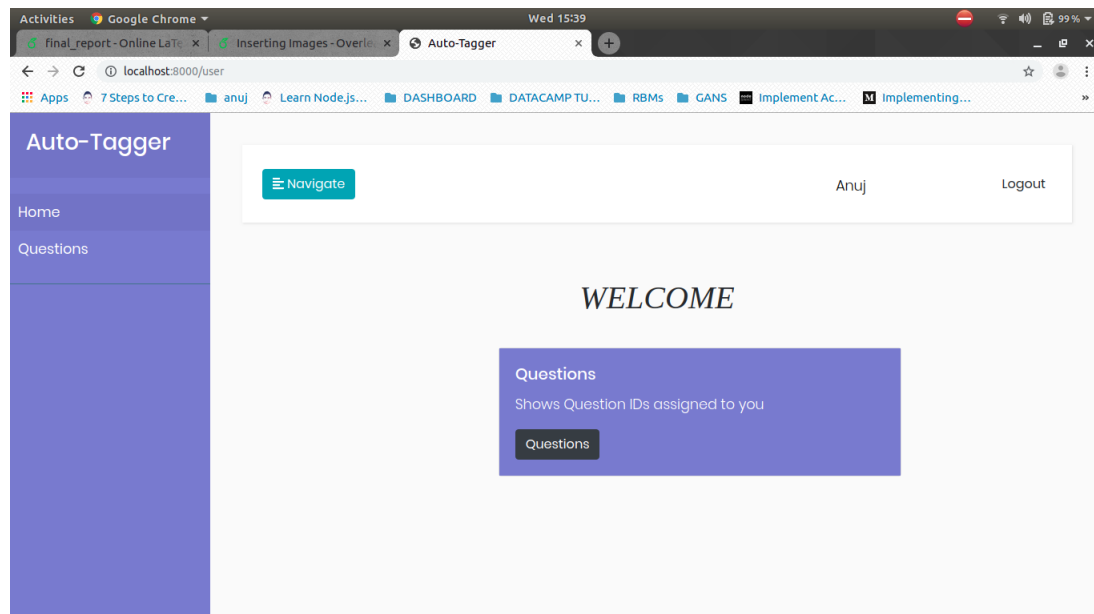
```
1 //home route
2 router.get('/',user_middleware,user_controller.home)
3 //user_controller.home
4 home: (req,res)=>{res.render("./user/home_user",{username:req
    .session.user.name})},
```

Listing 1.12: User Home Route

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



## 1.2. SOFTWARE AND CODE

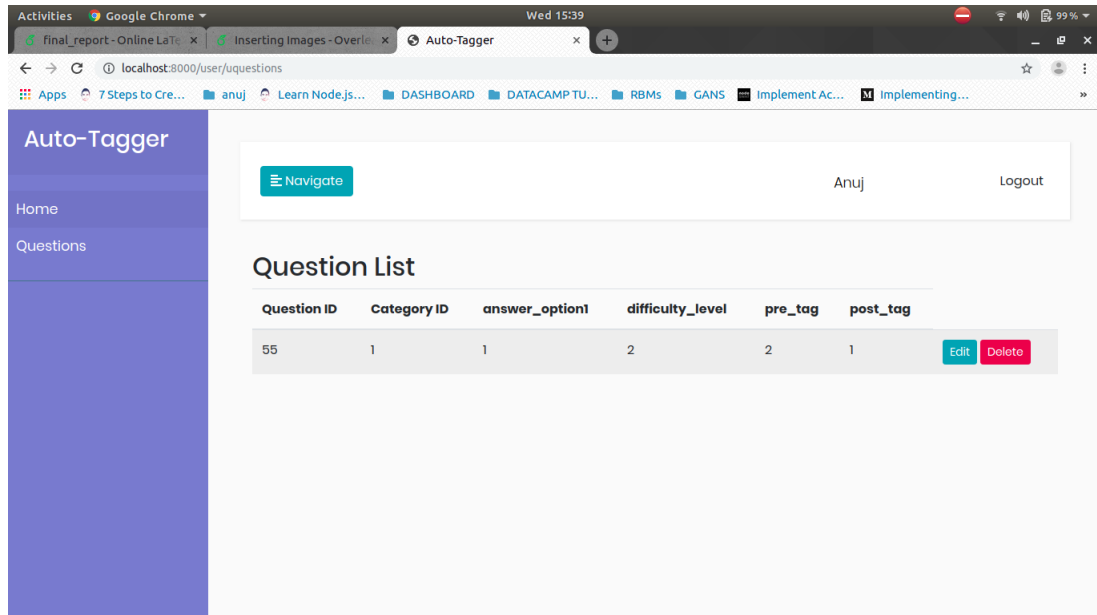


```
1 //route for showing data
2 router.get('/uquestions',user_middleware,user_controller.
   questions);
3 //user_controller.questions
4 questions:(req, res) => {
5     let sql = 'select question_id from user_question
6     where user_id = ${req.session.user.userid}';
7     let query = sequelize3.query(sql).then((results0)
8     =>{
9         var results = []
10        for(var i= 0;i<results0[0].length;++i){
11            results.push(results0[0][i].question_id)
12        }
13        question_master.findAll({
14            where:{
15                id:results
16            }
17        }).then((results1)=>{
18            res.render("./user/questions",{results:
19            results1})
20        })
21    })
22 }
```

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



### 1.3 Use and Demo

Please follow the link to see the demonstration of the setup and web app.  
[DEMO video for web app](#)

### 1.4 Future Work

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

- Discovering about other algorithms like Autoencoders and other deep learning algorithms.
- Making the frontend in React/Vue or any other frontend library for better design.
- Improve the performance of the REST API to scale better.
- Make two different API for both the models.

### 1.5 Bug report and Challenges

#### BUG REPORT





## 1.5. BUG REPORT AND CHALLENGES

- 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.
- If as user you go to a route which only admin has access you may get a message like **cannot (get/post) /page or You are not authorised Personnel**

### 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 :-

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

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

- 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.
- [2] S.Krithivasan, S.Gupta, S.Shandilya, K.Arya, K.Lala, *Auto-Tagging for Massive Online Selection Tests: Machine Learning to the Rescue*, 2016
- [3] S. Krithivasan, S. Shandilya, K. Arya, K. Lala, P. Manavar, S. Patil, and S. Jain, Learning by competing and competing by learning - experience from the e-yantra robotics competition, In IEEE Frontier In Education (FIE), 2014, October 2014.
- [4] Blog on Self Organizing Maps, <https://towardsdatascience.com/self-organizing-maps-ff5853a118d4>
- [5] Blog about passport authentication, <https://medium.com/gomycode/authentication-with-passport-js-73ca65b25feb>
- [6] Blog about Autoencoder Clustering, <https://www.dlology.com/blog/how-to-do-unsupervised-clustering-with-keras/>
- [7] Documentation for FlaskSQLAlchemy, <https://flask-sqlalchemy-session.readthedocs.io/en/v1.1/>