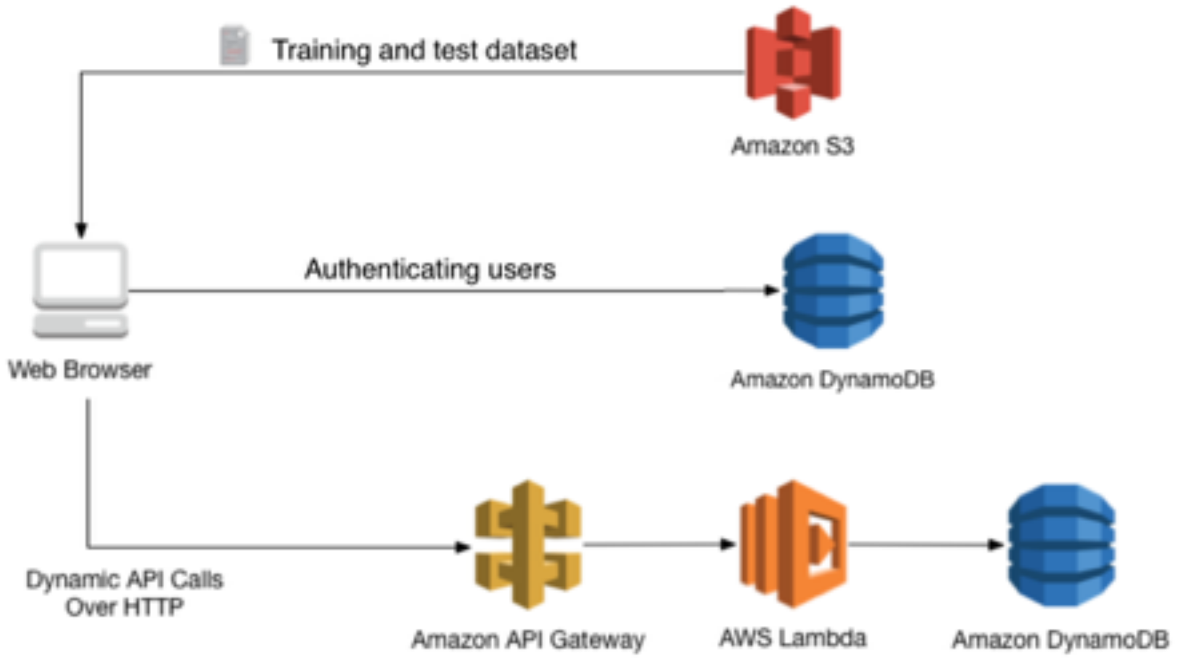
AN APPROACH TO SOLVE CLASSIFICATION PROBLEM

The general architecture of the web application is as follows :



DEVELOPER DOCUMENTATAION:

This developer documentation tells us about the code from the developer’s perspective i.e how the code works according to the inputs from the user. It gives us information about the flow of data through the various lines of the code. It also sheds light upon the uses of various functions that have been used (for authentication, image upload etc) and explains their internal workings.

We have created the following Lambda functions :

1. deletedataset: used to delete dataset of users/email on s3.

    parameters : email, dataset_name

    function returns : success or fail

    gateway api : deletedataset

2. getdataset: return all dataset under users/email account

    parameters : email

    function returns : a list of dataset_names

    gateway api : getdataset

3. knn : machine learning algorithm

    parameters : training file location on S3, test file location on S3

    function returns : results in string format

    gateway api : knn_api

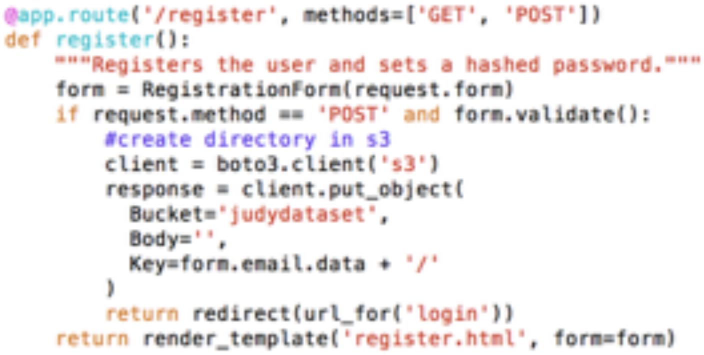
4. register_login : connect to dynamodb 'users' for user login and register

    parameters : func = "register" OR "login" , email , password

    function returns : success / fail

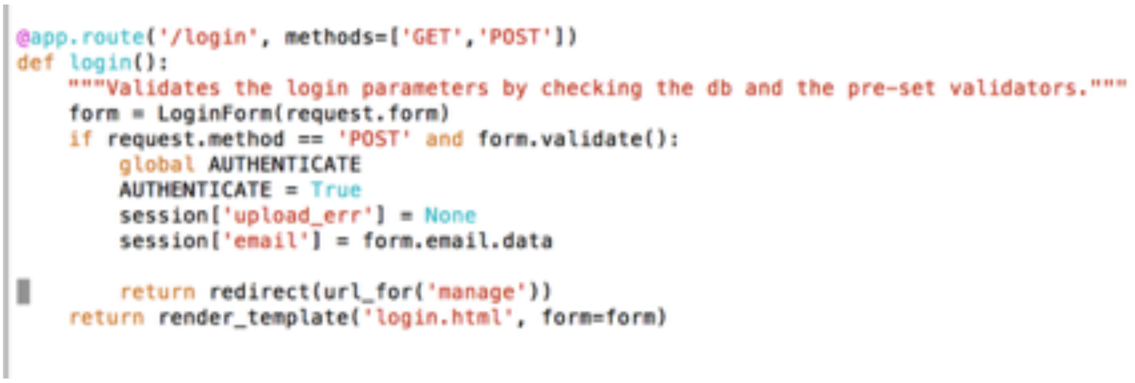
    gateway api : dynamodb_users_api

When any user registers himself into the application, his email address and password is stored in the Dynamodb.



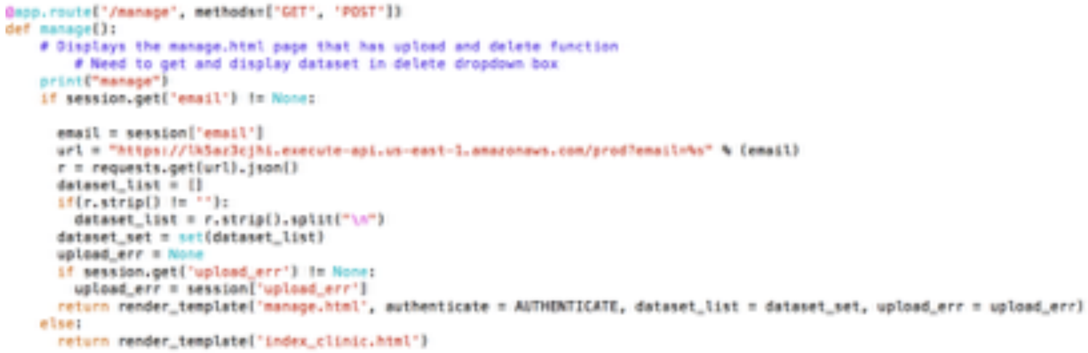
```
@app.route('/register', methods=['GET', 'POST'])
def register():
    """Registers the user and sets a hashed password."""
    form = RegistrationForm(request.form)
    if request.method == 'POST' and form.validate():
        #create directory in s3
        client = boto3.client('s3')
        response = client.put_object(
          Bucket='judydataset',
          Body='',
          Key=form.email.data + '/'
        )
        return redirect(url_for('login'))
    return render_template('register.html', form=form)
```

Authentication process takes place when a returning user enters his credentials. If the values match in the database, user is allowed to access the application.



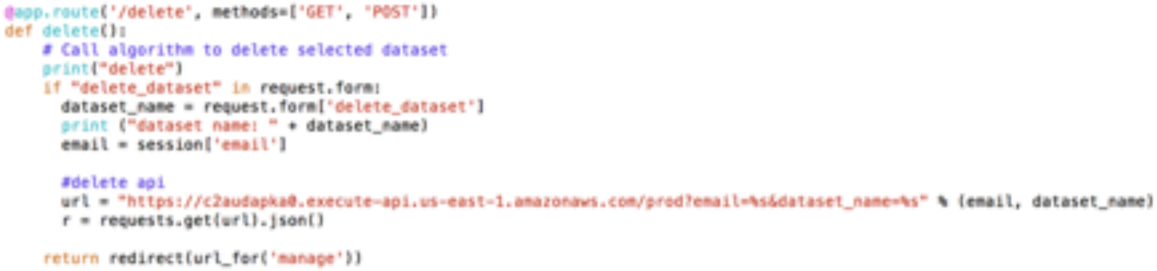
```
@app.route('/login', methods=['GET','POST'])
def login():
    """Validates the login parameters by checking the db and the pre-set validators."""
    form = LoginForm(request.form)
    if request.method == 'POST' and form.validate():
        global AUTHENTICATE
        AUTHENTICATE = True
        session['upload_err'] = None
        session['email'] = form.email.data

        return redirect(url_for('manage'))
    return render_template('login.html', form=form)
```

When the user clicks on the 'Manage Data' tab, the application redirects the user to a form. Here the user selects the training and the test dataset and uploads it. The uploaded file then gets saved in the S3 bucket. You can even delete any dataset..



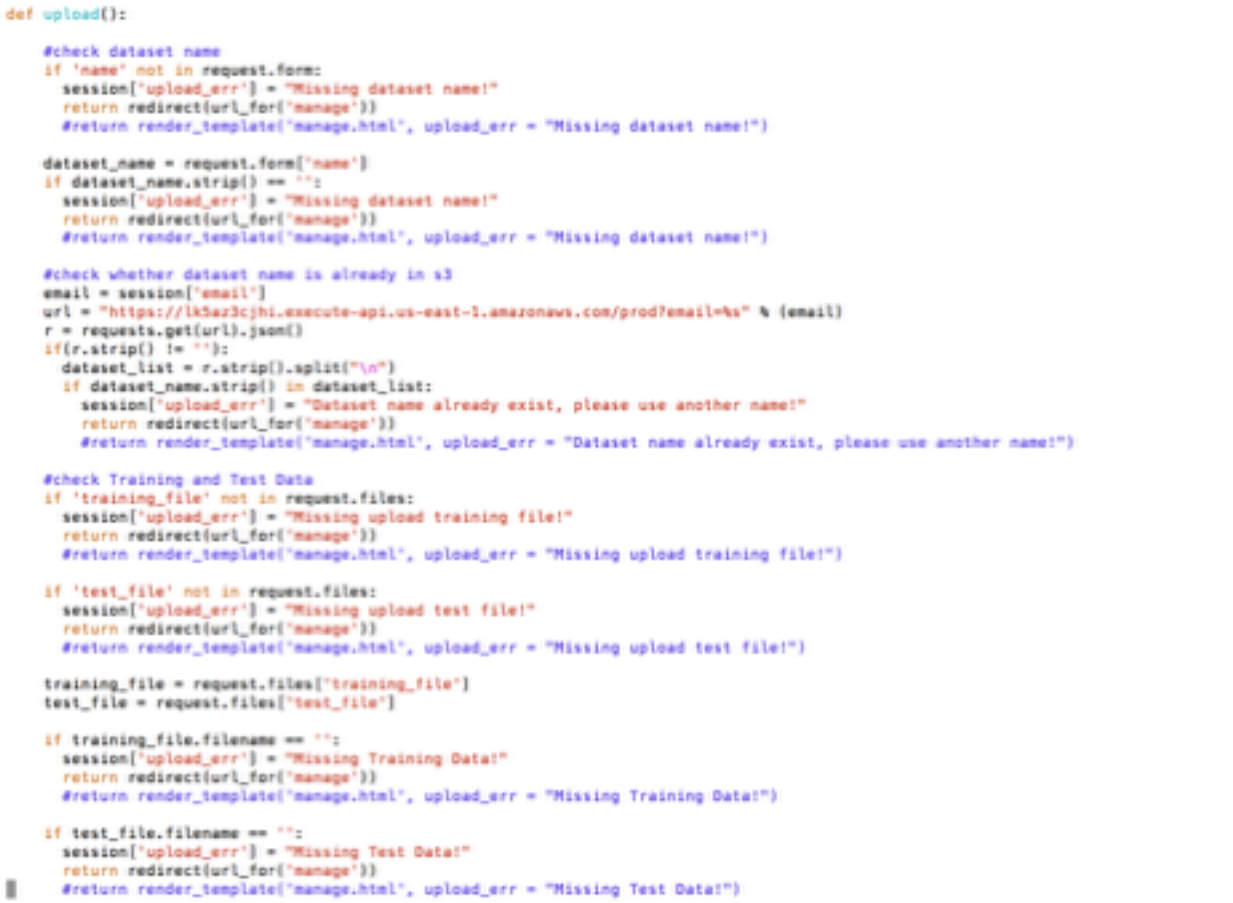
```
@app.route('/manage', methods=['GET', 'POST'])
def manage():
    # Displays the manage.html page that has upload and delete function
        # Need to get and display dataset in delete dropdown box
    print("manage")
    if session.get('email') != None:

      email = session['email']
      url = "https://lk5ar3cjhi.execute-api.us-east-1.amazonaws.com/prod?email=%s" % (email)
      r = requests.get(url).json()
      dataset_list = []
      if(r.strip() != ""):
        dataset_list = r.strip().split("\n")
      dataset_set = set(dataset_list)
      upload_err = None
      if session.get('upload_err') != None:
        upload_err = session['upload_err']
      return render_template('manage.html', authenticate = AUTHENTICATE, dataset_list = dataset_set, upload_err = upload_err)
    else:
      return render_template('index_clinic.html')
```

To delete a specific dataset, the following method has been implemented.

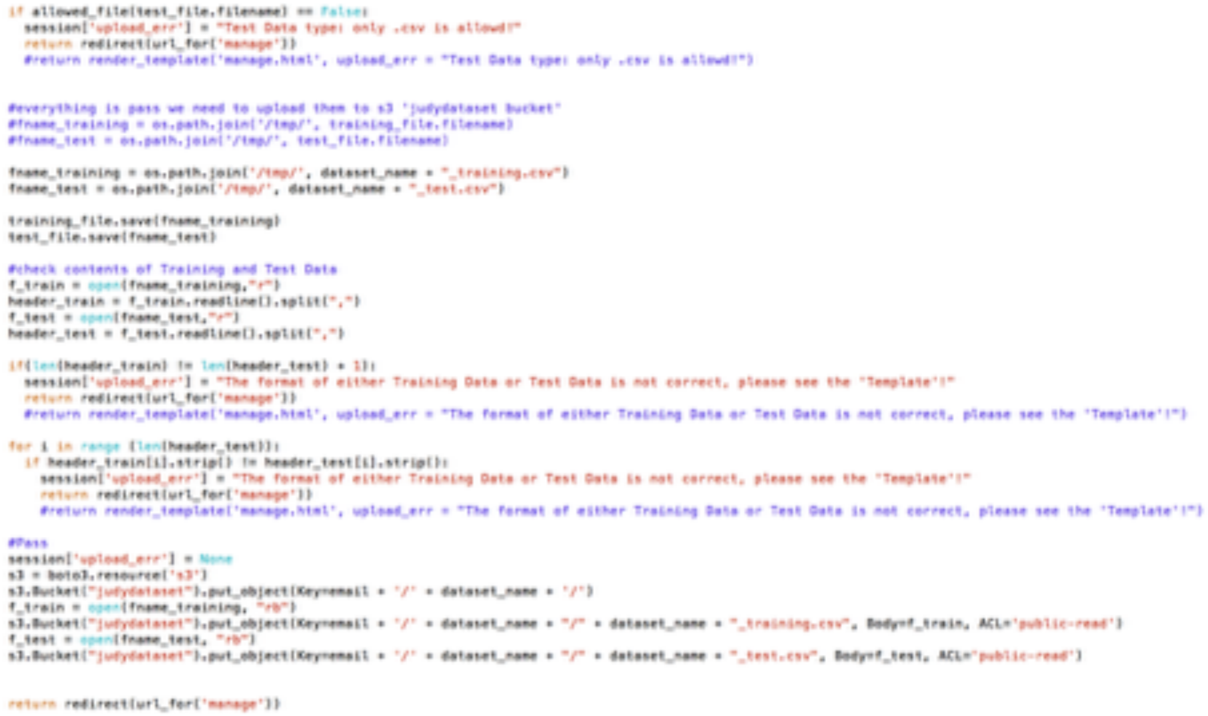


```
@app.route('/delete', methods=['GET', 'POST'])
def delete():
    # Call algorithm to delete selected dataset
    print("delete")
    if "delete_dataset" in request.form:
      dataset_name = request.form['delete_dataset']
      print ("dataset name: " + dataset_name)
      email = session['email']

      #delete api
      url = "https://c2audapka0.execute-api.us-east-1.amazonaws.com/prod?email=%s&dataset_name=%s" % (email, dataset_name)
      r = requests.get(url).json()

    return redirect(url_for('manage'))
```

The uploading dataset invokes the following upload method.

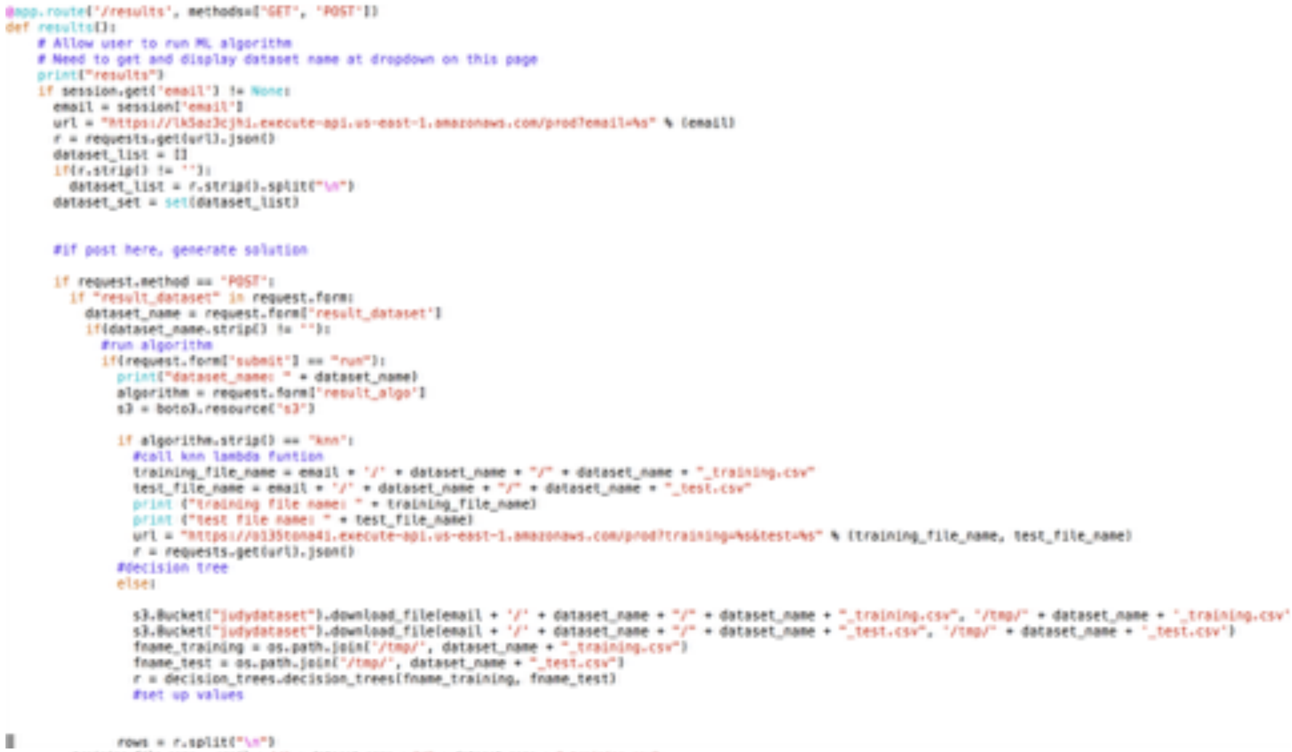


```
def upload():

    #check dataset name
    if 'name' not in request.form:
      session['upload_err'] = "Missing dataset name!"
      return redirect(url_for('manage'))
      #return render_template('manage.html', upload_err = "Missing dataset name!")

    dataset_name = request.form['name']
    if dataset_name.strip() == "":
      session['upload_err'] = "Missing dataset name!"
      return redirect(url_for('manage'))
      #return render_template('manage.html', upload_err = "Missing dataset name!")

    #check whether dataset name is already in s3
    email = session['email']
    url = "https://lk5ar3cjhi.execute-api.us-east-1.amazonaws.com/prod?email=%s" % (email)
    r = requests.get(url).json()
    if(r.strip() != ""):
      dataset_list = r.strip().split("\n")
      if dataset_name.strip() in dataset_list:
        session['upload_err'] = "Dataset name already exist, please use another name!"
        return redirect(url_for('manage'))
        #return render_template('manage.html', upload_err = "Dataset name already exist, please use another name!")

    #check Training and Test Data
    if 'training_file' not in request.files:
      session['upload_err'] = "Missing upload training file!"
      return redirect(url_for('manage'))
      #return render_template('manage.html', upload_err = "Missing upload training file!")

    if 'test_file' not in request.files:
      session['upload_err'] = "Missing upload test file!"
      return redirect(url_for('manage'))
      #return render_template('manage.html', upload_err = "Missing upload test file!")

    training_file = request.files['training_file']
    test_file = request.files['test_file']

    if training_file.filename == "":
      session['upload_err'] = "Missing Training Data!"
      return redirect(url_for('manage'))
      #return render_template('manage.html', upload_err = "Missing Training Data!")

    if test_file.filename == "":
      session['upload_err'] = "Missing Test Data!"
      return redirect(url_for('manage'))
      #return render_template('manage.html', upload_err = "Missing Test Data!")
```

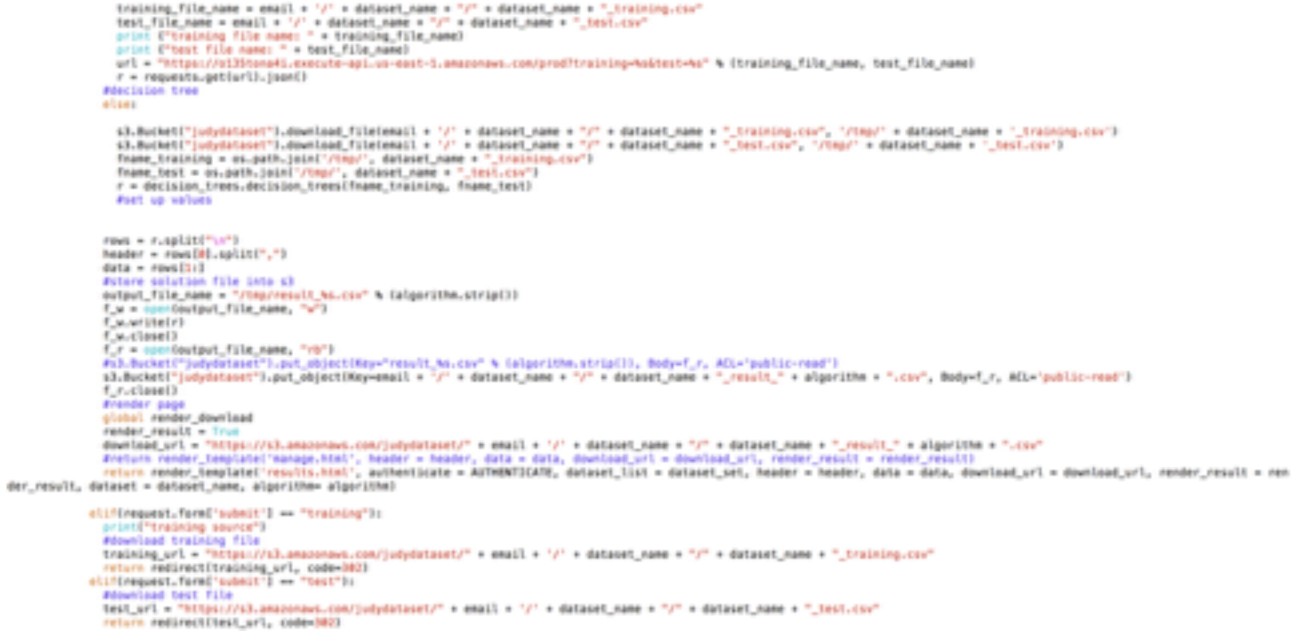


```
    if allowed_file(test_file.filename) == False:
      session['upload_err'] = "Test Data type: only .csv is allowd!"
      return redirect(url_for('manage'))
      #return render_template('manage.html', upload_err = "Test Data type: only .csv is allowd!")


    #everything is pass we need to upload them to s3 'judydataset bucket'
    #fname_training = os.path.join('/tmp/', training_file.filename)
    #fname_test = os.path.join('/tmp/', test_file.filename)

    fname_training = os.path.join('/tmp/', dataset_name + "_training.csv")
    fname_test = os.path.join('/tmp/', dataset_name + "_test.csv")

    training_file.save(fname_training)
    test_file.save(fname_test)

    #check contents of Training and Test Data
    f_train = open(fname_training,"r")
    header_train = f_train.readline().split(",")
    f_test = open(fname_test,"r")
    header_test = f_test.readline().split(",")

    if(len(header_train) != len(header_test) + 1):
      session['upload_err'] = "The format of either Training Data or Test Data is not correct, please see the 'Template'!"
      return redirect(url_for('manage'))
      #return render_template('manage.html', upload_err = "The format of either Training Data or Test Data is not correct, please see the 'Template'!")

    for i in range (len(header_test)):
      if header_train[i].strip() != header_test[i].strip():
        session['upload_err'] = "The format of either Training Data or Test Data is not correct, please see the 'Template'!"
        return redirect(url_for('manage'))
        #return render_template('manage.html', upload_err = "The format of either Training Data or Test Data is not correct, please see the 'Template'!")

    #Pass
    session['upload_err'] = None
    s3 = boto3.resource('s3')
    s3.Bucket("judydataset").put_object(Key=email + '/' + dataset_name + '/')
    f_train = open(fname_training, "rb")
    s3.Bucket("judydataset").put_object(Key=email + '/' + dataset_name + "/" + dataset_name + "_training.csv", Body=f_train, ACL='public-read')
    f_test = open(fname_test, "rb")
    s3.Bucket("judydataset").put_object(Key=email + '/' + dataset_name + "/" + dataset_name + "_test.csv", Body=f_test, ACL='public-read')


    return redirect(url_for('manage'))
```

Once you upload datasets you can go to Run and run a specific data. As output you get results of the test data, in which classification has been efficiently done.



```
@app.route('/results', methods=['GET', 'POST'])
def results():
    # Allow user to run ML algorithm
    # Need to get and display dataset name at dropdown on this page
    print("results")
    if session.get('email') != None:
      email = session['email']
      url = "https://lk5az3cjhi.execute-api.us-east-1.amazonaws.com/prod?email=%s" % (email)
      r = requests.get(url).json()
      dataset_list = []
      if(r.strip() != ''):
        dataset_list = r.strip().split("\n")
      dataset_set = set(dataset_list)


      #if post here, generate solution

      if request.method == "POST":
        if "result_dataset" in request.form:
          dataset_name = request.form['result_dataset']
          if(dataset_name.strip() != ""):
            #run algorithm
            if(request.form['submit'] == "run"):
              print("dataset_name: " + dataset_name)
              algorithm = request.form['result_algo']
              s3 = boto3.resource("s3")

              if algorithm.strip() == "knn":
                #call knn lambda funtion
                training_file_name = email + '/' + dataset_name + "/" + dataset_name + "_training.csv"
                test_file_name = email + '/' + dataset_name + "/" + dataset_name + "_test.csv"
                print ("training file name: " + training_file_name)
                print ("test file name: " + test_file_name)
                url = "https://o135tona41.execute-api.us-east-1.amazonaws.com/prod?training=%s&test=%s" % (training_file_name, test_file_name)
                r = requests.get(url).json()
              #decision tree
              else:

                s3.Bucket("judydataset").download_file(email + '/' + dataset_name + "/" + dataset_name + "_training.csv", '/tmp/' + dataset_name + '_training.csv'
                s3.Bucket("judydataset").download_file(email + '/' + dataset_name + "/" + dataset_name + "_test.csv", '/tmp/' + dataset_name + '_test.csv')
                fname_training = os.path.join('/tmp/', dataset_name + "_training.csv")
                fname_test = os.path.join('/tmp/', dataset_name + "_test.csv")
                r = decision_trees.decision_trees(fname_training, fname_test)
                #set up values


              rows = r.split("\n")
```



```
                training_file_name = email + '/' + dataset_name + "/" + dataset_name + "_training.csv"
                test_file_name = email + '/' + dataset_name + "/" + dataset_name + "_test.csv"
                print ("training file name: " + training_file_name)
                print ("test file name: " + test_file_name)
                url = "https://o135tona41.execute-api.us-east-1.amazonaws.com/prod?training=%s&test=%s" % (training_file_name, test_file_name)
                r = requests.get(url).json()
              #decision tree
              else:

                s3.Bucket("judydataset").download_file(email + '/' + dataset_name + "/" + dataset_name + "_training.csv", '/tmp/' + dataset_name + '_training.csv')
                s3.Bucket("judydataset").download_file(email + '/' + dataset_name + "/" + dataset_name + "_test.csv", '/tmp/' + dataset_name + '_test.csv')
                fname_training = os.path.join('/tmp/', dataset_name + "_training.csv")
                fname_test = os.path.join('/tmp/', dataset_name + "_test.csv")
                r = decision_trees.decision_trees(fname_training, fname_test)
                #set up values


              rows = r.split("\n")
              header = rows[0].split(",")
              data = rows[1:]
              #store solution file into s3
              output_file_name = "/tmp/result_%s.csv" % (algorithm.strip())
              f_w = open(output_file_name, "w")
              f_w.write(r)
              f_w.close()
              f_r = open(output_file_name, "rb")
              #s3.Bucket("judydataset").put_object(Key="result_%s.csv" % (algorithm.strip()), Body=f_r, ACL='public-read')
              s3.Bucket("judydataset").put_object(Key=email + '/' + dataset_name + "/" + dataset_name + "_result_" + algorithm + ".csv", Body=f_r, ACL='public-read')
              f_r.close()
              #render page
              global render_download
              render_result = True
              download_url = "https://s3.amazonaws.com/judydataset/" + email + '/' + dataset_name + "/" + dataset_name + "_result_" + algorithm + ".csv"
              #return render_template('manage.html', header = header, data = data, download_url = download_url, render_result = render_result)
              return render_template('results.html', authenticate = AUTHENTICATE, dataset_list = dataset_set, header = header, data = data, download_url = download_url, render_result = ren
der_result, dataset = dataset_name, algorithm= algorithm)

            elif(request.form['submit'] == "training"):
              print("training source")
              #download training file
              training_url = "https://s3.amazonaws.com/judydataset/" + email + '/' + dataset_name + "/" + dataset_name + "_training.csv"
              return redirect(training_url, code=302)
            elif(request.form['submit'] == "test"):
              #download test file
              test_url = "https://s3.amazonaws.com/judydataset/" + email + '/' + dataset_name + "/" + dataset_name + "_test.csv"
              return redirect(test_url, code=302)
```

DATABASE SCHEME:

The database used in this project is Dynamodb. We have the table named users (control user authentication). It has two fields i) Email- which stores every user's email and is the primary key for the table. ii) Password- it stores user password which is based using salt.

DEPLOYMENT INSTRUCTIONS:

After connecting to the instance (you can connect by executing

“ ssh -i .ssh/ece1779.pem ubuntu@PublicIP-Of-Instance”) , site can start by giving the following commands:

1. First you need to set the virtual environment. This can be done by the following command:

    - **cd Desktop/ece1779/aws/venv/bin**
    - **source activate**

2. After this going back to the ece1779 folder, do ' **cd a3 '** and if starting from the root execute the command
   cd Desktop/ece1779/a3

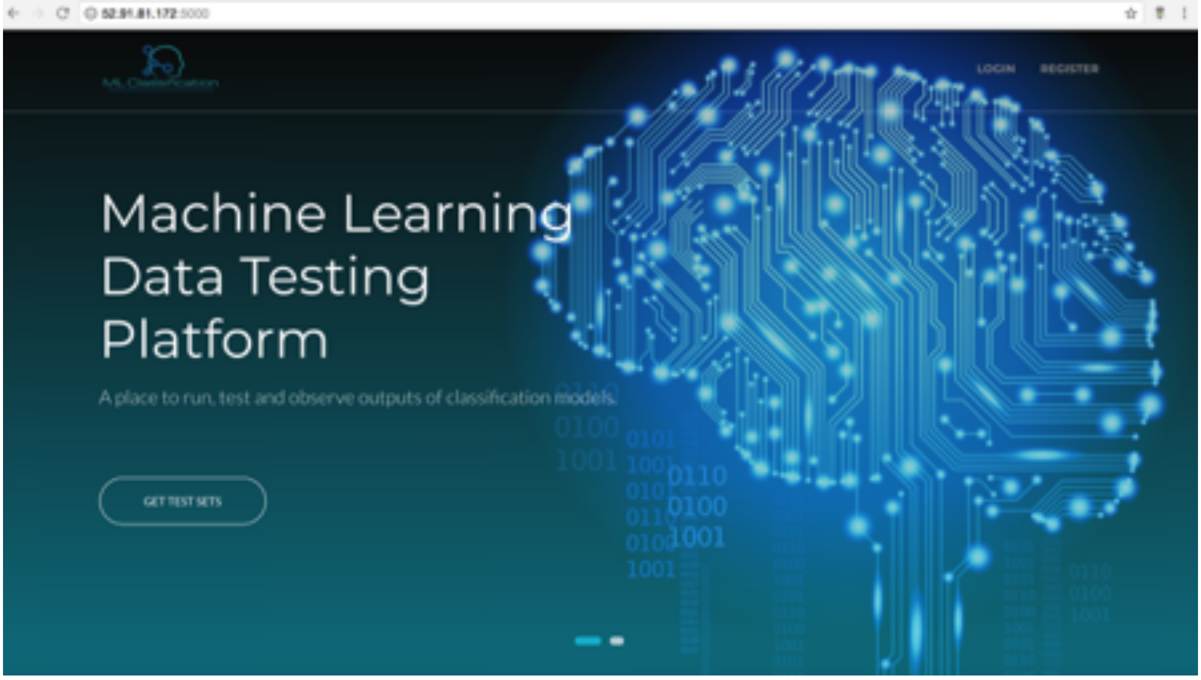
3.Then running the server.py file by : **python server.py**

4.After this using the instance’s publicIP and the port for running the application is 5000 (i.e publicIP:5000) , you can access the application.

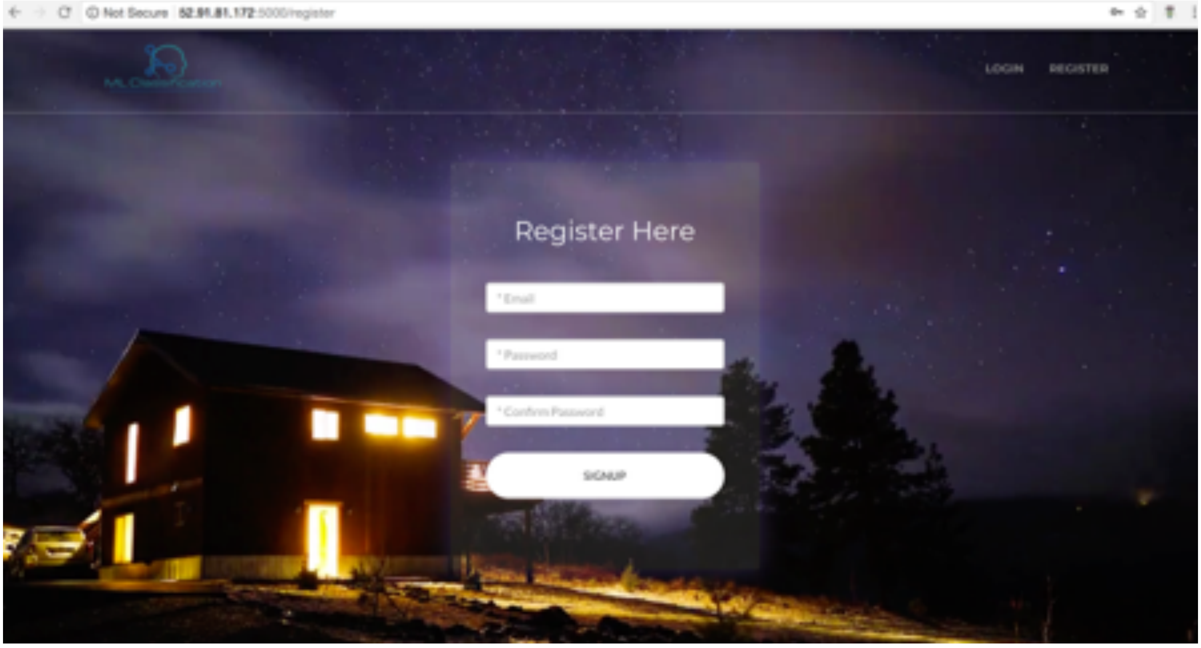
User Documentation:

We have attempted to solve the classification problem through the ML algorithm. This user friendly and easy to use application along with being extremely time efficient, efficiently solves the classification problem.

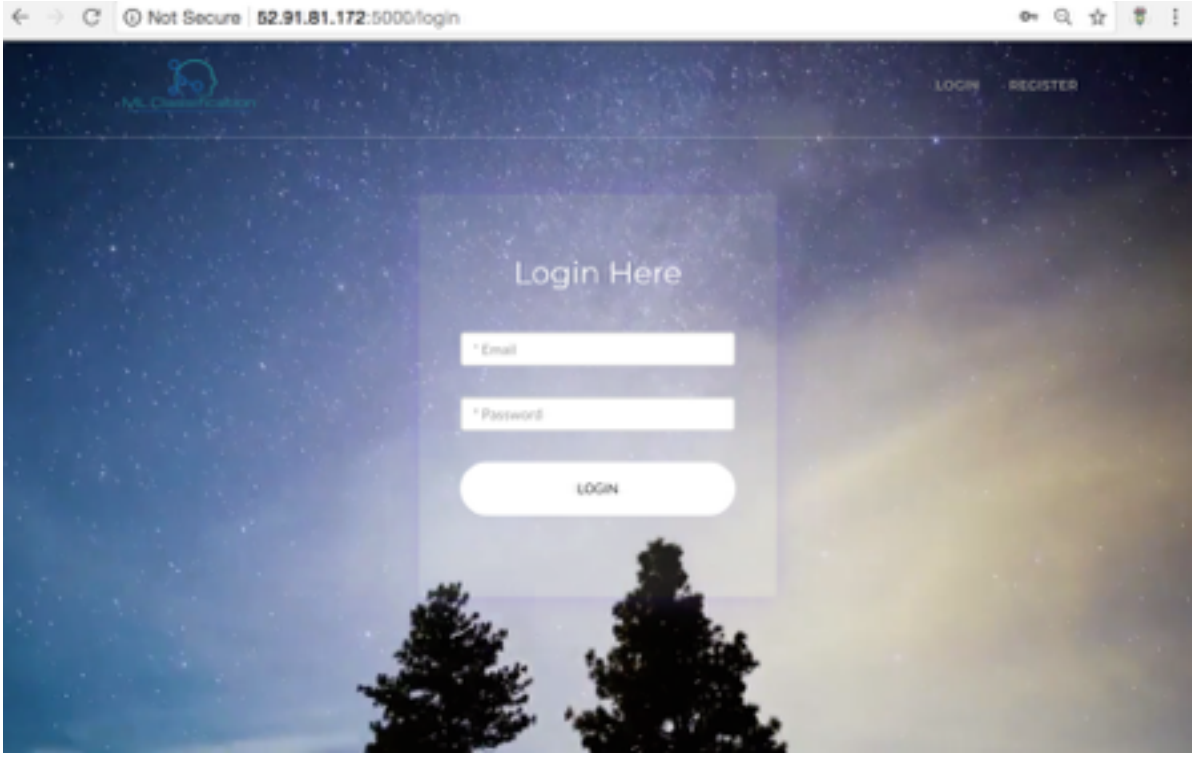
You can get a general idea regarding ml using the videos provided in the website.



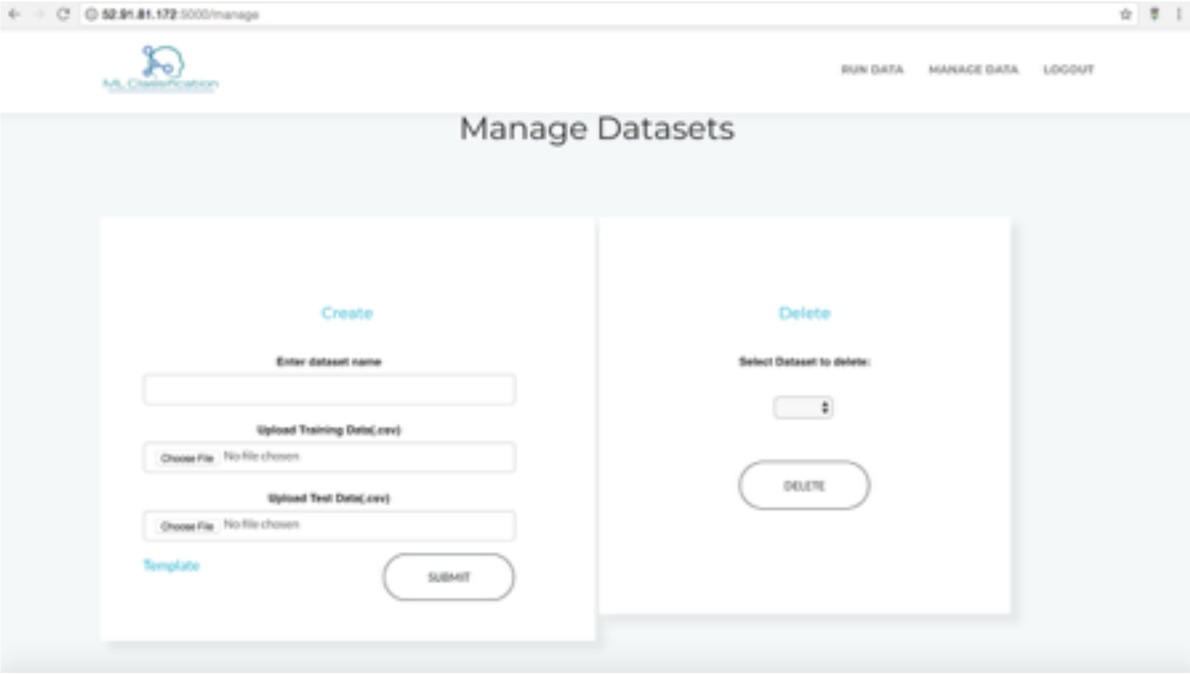
Register yourself if you are using the website for the first time.



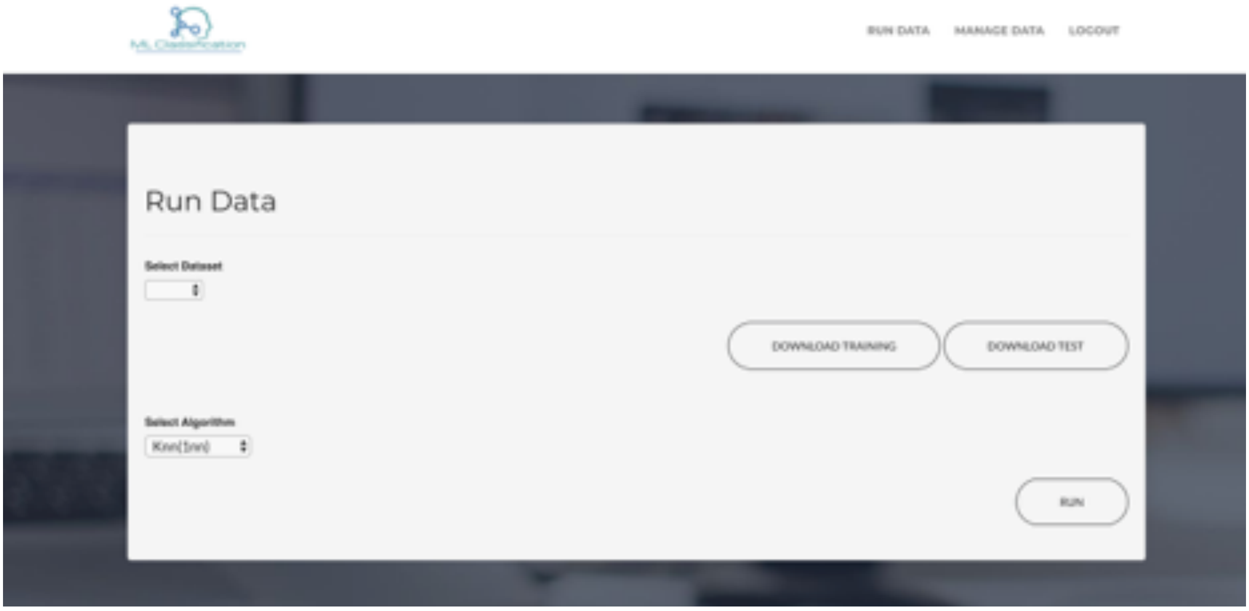
After registering yourself you can login into the website



After logging in, you can go to MANAGE DATA and add any datasets(training data and test data) you want. You can also delete them. You can use these datasets for training the machine.



Once you have done this, you can go to RUN DATA and select a dataset and algorithms and run to begin the classification.



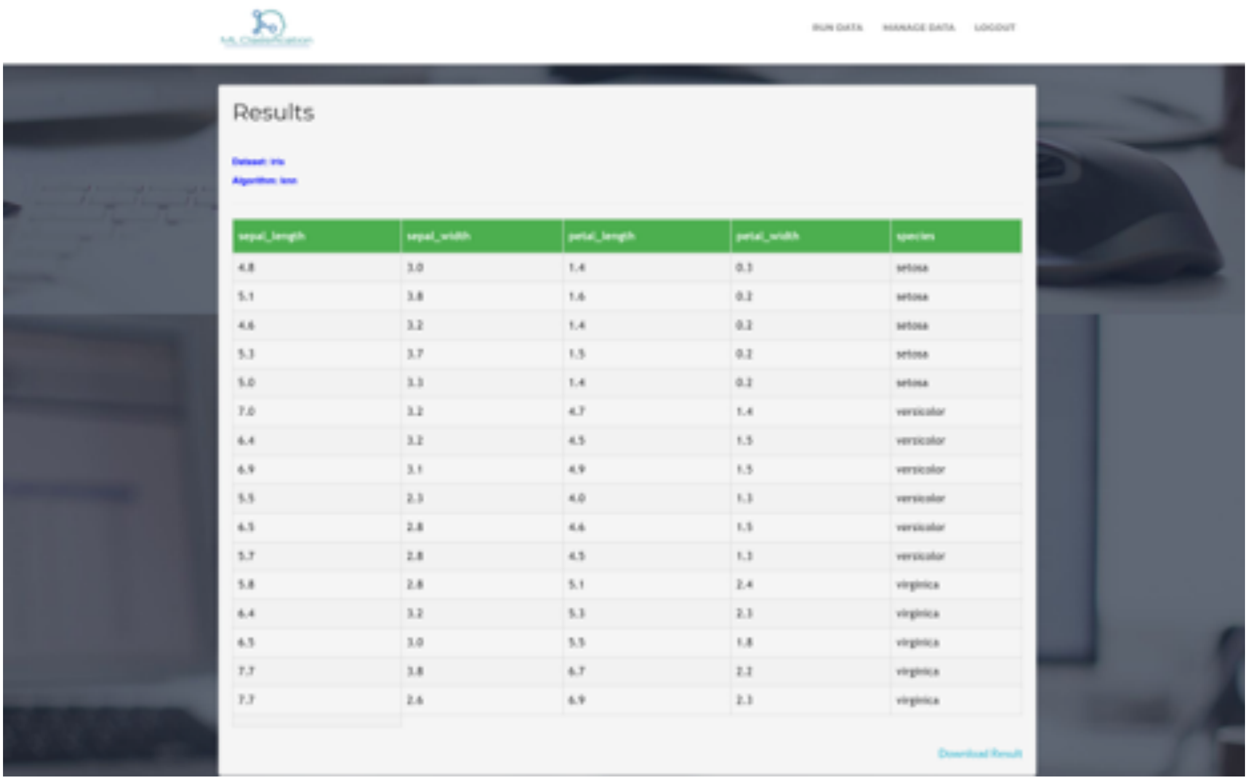
Run Data

Select Dataset

Select Algorithm

Knn(1nn)

After the Machine is trained using the training data, it classifies the test data into respective classes. You can download the results.



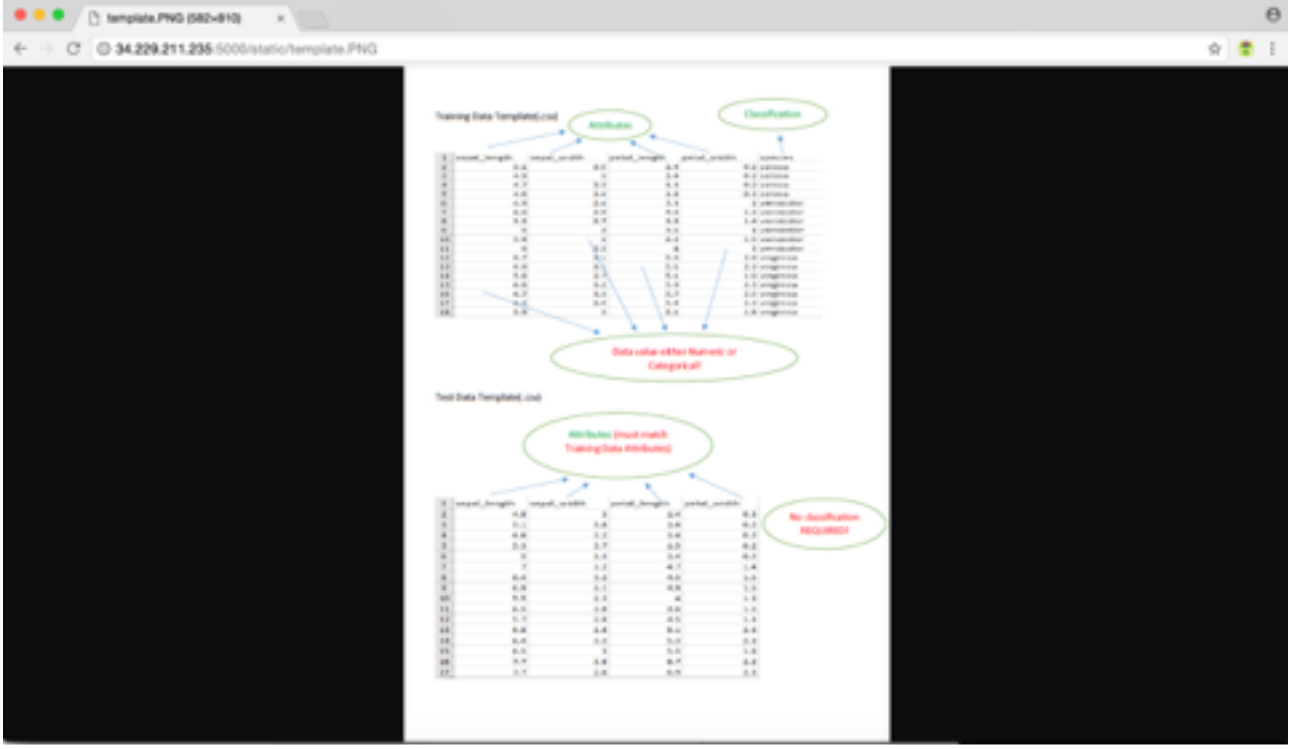
Results

Dataset: iris

Algorithm: knn

sepal_length	sepal_width	petal_length	petal_width	species
4.8	3.0	1.4	0.1	setosa
5.1	3.8	1.6	0.2	setosa
4.6	3.2	1.4	0.2	setosa
5.3	3.7	1.5	0.2	setosa
5.0	3.3	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.9	3.1	4.9	1.5	versicolor
5.5	2.3	4.0	1.3	versicolor
6.5	2.8	4.6	1.5	versicolor
5.7	2.8	4.5	1.3	versicolor
5.8	2.8	5.1	2.4	virginica
6.4	3.2	5.3	2.3	virginica
6.5	3.0	5.5	1.8	virginica
7.7	3.8	6.7	2.2	virginica
7.7	2.6	6.9	2.3	virginica

A sample template can be viewed in Data Template .



template.PNG (582×810)
34.229.211.235:5000/static/template.PNG

Training Data Template(.csv)

Attributes
Classification
Data value either Numeric or Categorical!

1	sepal_length	sepal_width	petal_length	petal_width	species
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Test Data Template(.csv)

Attributes (must match Training Data Attributes)
No classification REQUIRED!

1	sepal_length	sepal_width	petal_length	petal_width
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]