Location Prediction Based Action

Deekshith Allamaneni
Dept. of Electrical & Computer Engineering
Missouri University of Science and Technology, Rolla, Missouri 65409
Email: daqnf@mst.edu

***Abstract*—The aim of this project is to predict the current location of users based on their past location history and take an action depending on the location match. In this project, the GPS sensor in the mobile phones is used to track the current location of the user and the data is logged onto a server at a defined interval. The server processes location coordinates logged over time using a neural network and estimates the current location of the user. The client requests the predicted location from server and takes an action like notifying or alerting the user if the user is not found to be in an incorrent location at that time.**

I. Introduction

Many a times we miss a schedule due to forgetfulness. Reminders and alarms solve this issue to some extent but there can be times when we even forget to set an alarm. This project aims to solve this problem by keeping track of users' location and then generates a prediction of their current location by using neural networks. The current location and the predicted location is compared and necessary action is taken by the client side application.

This projects consists of two components, the client which is a mobile application and a server which processes and returns the data sent from the client. The main scope of this application is to predict the user's current location. The methods and functions to get an estimation as well as for location comparison are built into the client application and are demostrated. However, the applications of this can be extended easily using the existing client and server infrastructure.

II. Client Application

Most smart phones are equipped with a GPS sensor and also provide an API for applications to query its current geographical location. For this project I have designed a mobile application in Java for Android operating system that acquires the current location of the user by using the GPS sensor equipped with the smartphone.

The client has two main functions.

1) Data Logging
2) Location prediction based action

A. Data Logging

The client application acquires the current location of the user via Android API and uses the REST API [2] provided by the server side application to send the location and the time at which the location is acquired to the server.

The Table I shows the data that the clinet saves periodically to the server.

	Latitude	Longitude	Weekday	Hours	Minutes
Range	-90 to +90	-180 to +180	0 to 6	0 to 23	0 to 59

TABLE I
Data stored to server by the client

B. Location prediction based action

The server also provides a HTTP REST API to query the predicted location of the user. The client requests the current predicted location of the user using HTTP GET request. The client also gets the current GPS location from the sensor and takes an action depending on the comparison.

1) Action 1: Report known location but prediction not matched: When the user is at a known location but not at the right time, it notifies the user as shown in Figure 1 and provides an interface for user acknowledgement.

2) Action 2: Unidentified location: When the user is at an unknown location for a certain amount of time, the client side application requests the user to enter the name of this current location. If no name is provided but if the prediction reports that location, it just identifies that location by the coordinates. Figure 2 shows the unidentified location notification and Figure 3 shows the interface to enter a new location.

III. Server Application

Server side application is written in Python using Flask web framework. Most of the heavy load is handled by the server as the client runs on a mobile application, so doing major processing on the server side can save the battery life on client side.

The server application provides a RESTful API for the clients to interact with it. It saves the data sent by the server in a SQLite database and processes it to estimate the current location of the user.

A. Neural Network

The server application uses Levenberg-Marquardt backpropagation [1] [3] neural networks to process the data sent by the client to predict the user location and returns to the client upon request.

We are using a supervised learning model in which the input is the time information and the output is the location information at that corresponding time.

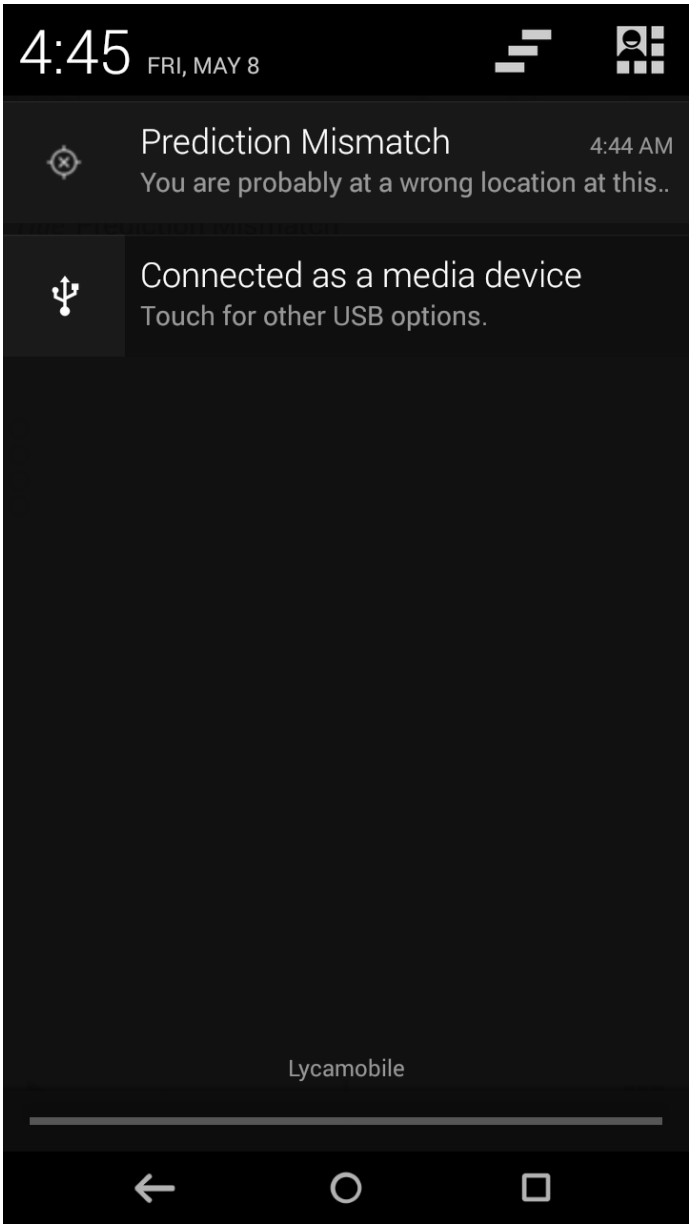

Fig. 1. Screenshot of the client app displaying prediction mismatch notification

Fig. 2. Screenshot of the client app notifying about unknown location

1) Neural Network Training Inputs and Their Representation: The inputs for the neural network is the time information including the weekday, hours and minutes. The neural network has a total of ten inputs out of which 3 inputs are used to represent the weekday, 5 inputs to represent hour and 2 inputs for quantized minutes information.

a) Weekday: Weekday is stored in the database in the range 0 to 6 where 0 is Monday, 1 is Tuesday and so on upto 6 for Sunday. But while giving as an input for the neural network, we are using a binary form of the weekday so that there are three neural network inputs representing it.

b) Hours: Hours are stored in 24-hour format ranging from 0 to 23 where 0 represents 12:00 AM and 23 represents 11:00 PM. While passing it to the neural network, I am converting it into binary format to represent the hours information with five neural network inputs.

c) Minutes: The client sends the exact minutes information ranging from 0 to 59 but that detail is not necessary for this application. So when giving it as an input for neural network, I am quantizing it to the lower fifteen minutes and representing the range 0 to 60 as just 0 to 3 converted to binary. The minutes 0 to 14 is represented as 0 (0,0), 15 to 29 as 1 (0, 1), 30 to 44 as 2 (1, 0) and 45 to 59 as 3 (1, 1). So the minutes information is represented using 2 inputs. This form of representation improves the performance of the netowrk by a great extent.

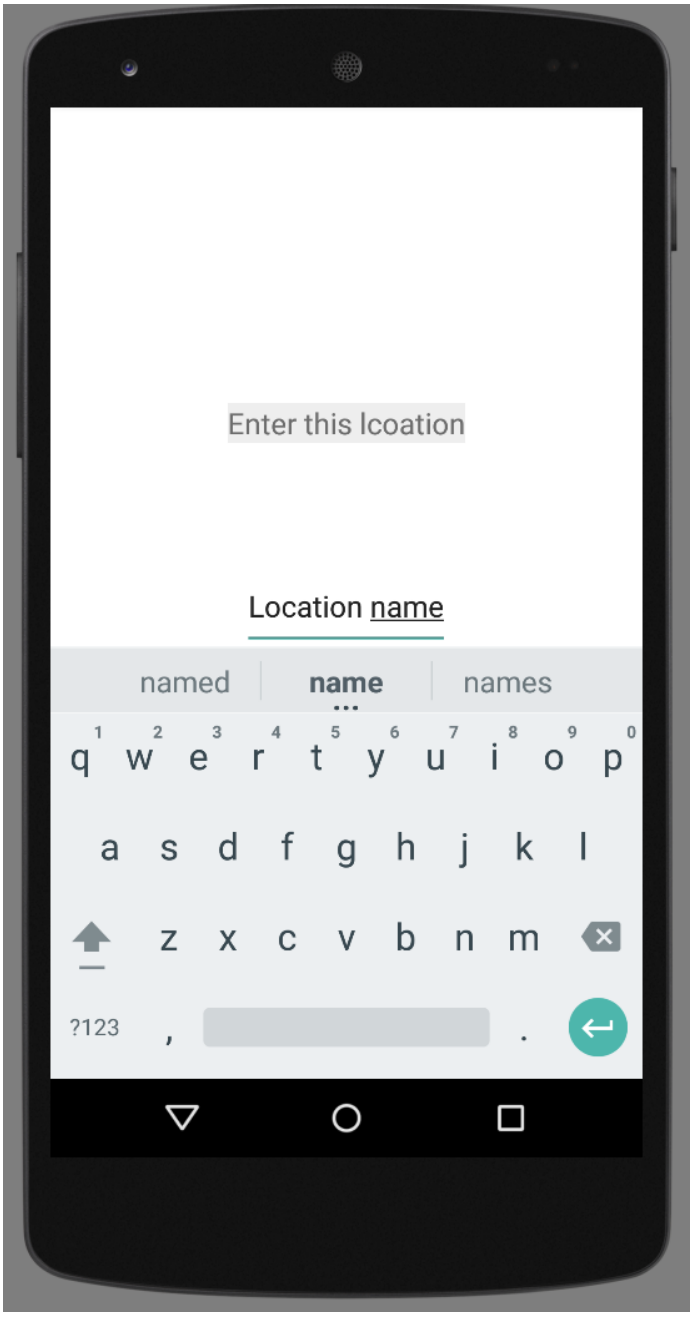

Fig. 3. Screenshot of the client app interface to enter location name

2) Neural Network Training Targets and their Representation: The targets for the neural network are the latitude and longitude corresponding to the time data given at the input.

Latitude ranges from -90 to +90. The altitude data is preprocessed by using the formula

lat_pre[] = (lat[] max(lat[])+90

We are basically subtracting the latitude values with the maximum value of the latitude and then adding 90 to it which makes it a positive and relative distance from the user location rather than absolute geo-coordinates. This reduces the magnitude of the coordinates and improves the performance of the network.

Longitude has a range of - 180 to +180 and it is preprocessed similar to the latitude as shown above

lon_pre[] = (lat[] max(lon[])+180

3) Neural Network Architecture: The neural network for this project makes use of Levenberg-Marquardt backpropagation algorithm using Python's PyBrain module. The architechture has an input layer with 10 inputs, a hidden layer with 8 hidden neurons and an output layer with two outputs. The input to the neural network is the time information and the output is the geographical coordinates.

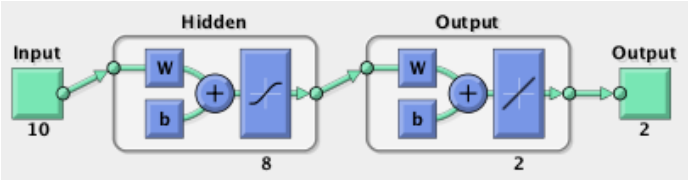

Fig. 4. Block diagram of neural net architecture generated using MATLAB

I am usually using 20 epochs and a learning rate of 0.2 for training. 70% of the training dataset is used for training, 15% for testing and remaining 15% for validation.

IV. APPLICATIONS

The client and server provides fully functional methods and API's to configure it to any application desired. Some of the possible applications are discussed as below.

A. Auto Alarm

Imagine a situation when the user forgot to leave to office and still on his bed. The client side application detects that the predicted location at that time should be his office location but it is still his home location. So when there is a mismatch between the office location and the home location, the app alerts the user. This can work as an auto alarm in that way.

B. Auto Lock and Theft Protect

When the user location is found to be unknown and does not match either the prediction or the logged location data, it can be configured to auto lock so as to ensure it has not been stolen.

V. RESULTS

The performance of the network varies drastically with the sample data of different users even with the same sample size. Best performance can be expected when there is a regular pattern in the user location over time. The performance is also better if the user is confined to a small geographical area as the input to the neural network is the relative coordinates, the magnitude is less in this case therefore boosting performance. A minimum of 2 weeks of data is needed to for it to predict the location with minimum error. The server automatically removes the data older than 60 days so as to adapt to newer locations as well as to reduce load on the database.

The test user with performance plot shown in Figure 5 has a location history as shown in Figure 6. We can observe that the

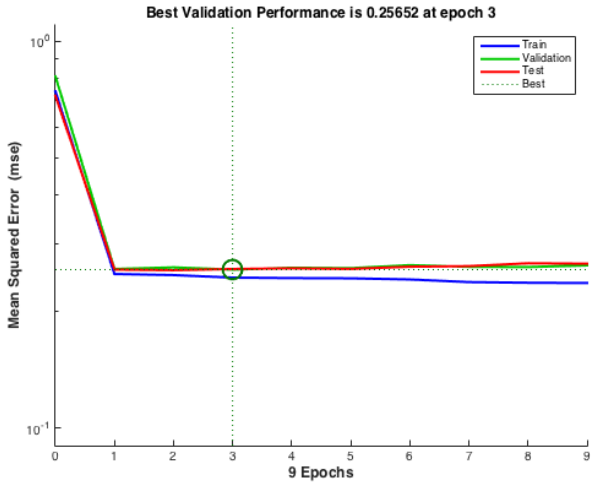

Fig. 5. Performance plot of the network for a test user

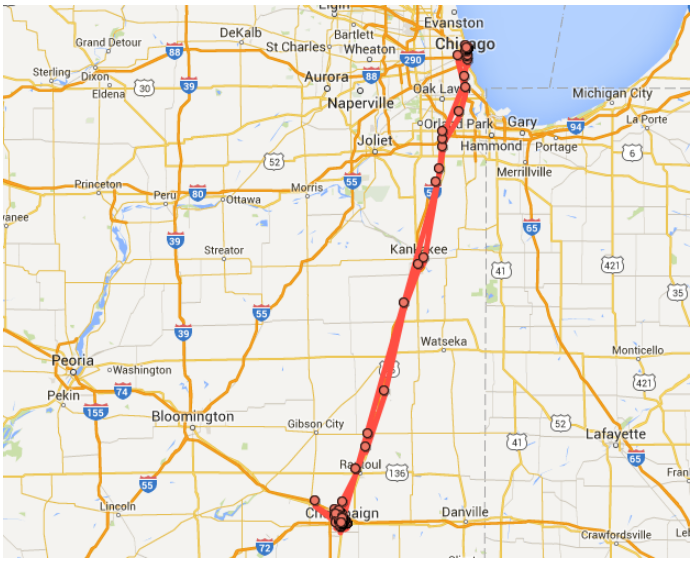

Fig. 6. Graphical ploting of a test user's geographical loaction

user is confined to two cities most of the time and travelling back and forth regularly at almost the same time on weekdays. As there is less complexity in the user's pattern it is trained well within jsut a few epochs as shown in Figure 5 and the performance is good even with lesser samples compared to other cases.

References

[1] Paul J. Werbos, *Backpropagation Through Time: What it Does and How To Do It*, Proceedings of the IEEE, VOL. 78, NO 10, OCTOBER, 1990.

[2] Li Li, Wu Chou, *Design and Describe REST API without Violating REST: A Petri Net Based Approach*, 978-0-7695-4463-2/11 IEEE 2011.

[3] Reynaldi, A. ; Lukas, S. ; Margaretha, H, *Backpropagation and Levenberg-Marquardt Algorithm for Training Finite Element Neural Network*, 978-1-4673-4977-2 IEEE 2012.

Appendix A
Client/MainActivity.java

```
/*
This is the main interface for the Android
    client
 */
package com.example.locationupdater;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(
    Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Intent intent = new Intent(this,
      LocationService.class);
    //intent.putExtra("ActivityStatus",
        "true");
    startService(intent);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu
      menu) {
    // Inflate the menu; this adds items to
    // the action bar if it is present.
    getMenuInflater().inflate(R.menu.main,
        menu);
    return true;
  }

  @Override
  public boolean onOptionsItemSelected(
    MenuItem item) {
    // Handle action bar item clicks here.
    // The action bar will
    // automatically handle clicks on the
    // Home/Up button, so long
    // as you specify a parent activity
    // in AndroidManifest.xml.
    int id = item.getItemId();
    if (id == R.id.action_settings) {
      return true;
    }
    return super.onOptionsItemSelected(item);
  }
}
```

Appendix B
Client/LocationService.java

```
/**
 * Runs a background service to monitor
 * the location and update it on the
 * server periodically.
 */
package com.example.locationupdater;
```

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;
import java.util.Calendar;
import java.util.TimeZone;
import java.util.UUID;

import org.json.JSONException;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import
    android.content.SharedPreferences.Editor;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;

public class LocationService extends Service {
  // in Meters
 private static final long MIN_DIST = 0;
  // requireed 5 mins in Milliseconds
 private static final long MON_TIME = 300000;
 //DBController dbcontroller;
 Calendar c;

 private static String uniqueID = null;
 private static final String PREF_UNIQUE_ID
  = "PREF_UNIQUE_ID";

 //public static final String Stub = null;
 protected LocationManager locationManager;
  // LocationListener mlocList ;
  private Context mComtext;
  Location location; // location

  //GPSTracker mGPS;

 @Override
 public void onCreate() {
  // TODO Auto-generated method stub
  super.onCreate();
  Log.i("LocationService", "onCreate");
  mComtext = this;
   // US OR CST Time zone
  TimeZone tz =
      TimeZone.getTimeZone("GMT-06:00");
  c = Calendar.getInstance(tz);
 }


 @Override
 public int onStartCommand(
  Intent intent,
  int flags,
  int startId) {
  Log.i("LocationService", "onStartCommand");

  //mlocList = new MyLocationListener();

  locationManager =
    (LocationManager) getSystemService(
     Context.LOCATION_SERVICE);

     locationManager.requestLocationUpdates(
           LocationManager.GPS_PROVIDER,
           MON_TIME,
           MIN_DIST,
           new MyLocationListener()
     );

  return START_STICKY;
 }

 @Override
 public IBinder onBind(Intent arg0) {
  // TODO Auto-generated method stub
  return null;
 }


 private class MyLocationListener implements
    LocationListener {

     public void onLocationChanged(Location
         location) {
        int dayOfWeek =
            c.get(Calendar.DAY_OF_WEEK) - 1;
        Log.i("LocationService dayOfWeek ",
            "" + dayOfWeek);
        int hour =
            c.get(Calendar.HOUR_OF_DAY);
        Log.i("LocationService hour ", "" +
            hour);
        int minutes = c.get(Calendar.MINUTE);
        if(minutes < 15){
         minutes = 0;
        }else if(minutes < 30){
         minutes = 1;
        }else if(minutes < 45){
         minutes = 2;
        }else{
         minutes = 3;
        }
        Log.i("LocationService minutes ", ""
            + minutes);
        //dbcontroller.logLocationData(location.getLati
          , location.getLongitude(),
              dayOfWeek, hour, minutes);

        String uuid = getUUID(mComtext);
        String url =
            "http://parishod.com/logdata/" +
          uuid + "/" +
          location.getLatitude() + "/" +
          location.getLongitude() + "/" +
          dayOfWeek + "/" +
          hour + "/" + minutes;

        Log.i("LocationService URL ", "" +
            url);
        new MyAsyncTask().execute(url);
     }

  }

 private class MyAsyncTask extends
    AsyncTask<String, Integer, Double>{
```

```
  //String result1 = "";
  @Override
  protected Double doInBackground(String...
      params) {
    // TODO Auto-generated method stub
    try {
     postData(params[0]);
    } catch (JSONException e) {
     // TODO Auto-generated catch block
     e.printStackTrace();
    } catch (Exception e) {
     // TODO Auto-generated catch block
     e.printStackTrace();
    }
    return null;
  }

  public void postData(String
      valueIWantToSend) throws Exception {
    //String resultString = null;
    String content =
        getResponse(valueIWantToSend);
       System.out.println(content);

       String uuid = getUUID(mComtext);
       String content1 = getResponse(
         "http://parishod.com/logdata/" +
         uuid + "/predictlocation");
       System.out.println(content1);
  }

}

/*Function to get UUID*/
public String getUUID(Context context) {
   if (uniqueID == null) {
      SharedPreferences sharedPrefs =
        context.getSharedPreferences(
            PREF_UNIQUE_ID,
                Context.MODE_PRIVATE);
      uniqueID =
          sharedPrefs.getString(PREF_UNIQUE_ID,
          null);
      if (uniqueID == null) {
         uniqueID =
             UUID.randomUUID().toString();
         Editor editor = sharedPrefs.edit();
         editor.putString(PREF_UNIQUE_ID,
             uniqueID);
         editor.commit();
      }
   }
   return uniqueID;
}

/*Gets Json Response*/
public static String getResponse(String url)
    throws Exception {
     URL website = new URL(url);
     URLConnection connection =
         website.openConnection();
     BufferedReader in = new BufferedReader(
                       new InputStreamReader(
                          connection.getInputStream()));

     StringBuilder response = new
         StringBuilder();
         String inputLine;

         while ((inputLine = in.readLine()) !=
             null)
            response.append(inputLine);

         in.close();

         return response.toString();
    }

}
```

APPENDIX C
CLIENT/GPSTRACKER.JAVA

```
/**
 * Gets the current location information
 * from the GPS using native android API
 */
package com.example.locationupdater;


import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.provider.Settings;
import android.util.Log;

public final class GPSTracker implements
    LocationListener {

   private final Context mContext;

   // flag for GPS status
   public boolean isGPSEnabled = false;

   // flag for network status
   boolean isNetworkEnabled = false;

   // flag for GPS status
   boolean canGetLocation = false;

   Location location; // location
   double latitude; // latitude
   double longitude; // longitude

   // The minimum distance to change Updates
       in meters
   private static final long MIN_DIST = 1; //
       10 meters

   // The minimum time between updates in
       milliseconds
   private static final long MIN_TIME_UPD =
       1; // 1 minute
   // Declaring a Location Manager
   protected LocationManager locationManager;
```

```
public GPSTracker(Context context) {
   this.mContext = context;
   getLocation();
}

/**
 * Function to get the user's current
     location
 *
 * @return
 */
public Location getLocation() {
   try {
      locationManager = (LocationManager)
          mContext
             .getSystemService(
                Context.LOCATION_SERVICE);

      // getting GPS status
      isGPSEnabled = locationManager
            .isProviderEnabled(
               LocationManager.GPS_PROVIDER);

      Log.v("isGPSEnabled", "=" +
          isGPSEnabled);

      // getting network status
      isNetworkEnabled = locationManager
            .isProviderEnabled(
               LocationManager.NETWORK_PROVIDER);

      Log.v("isNetworkEnabled", "=" +
          isNetworkEnabled);

      if (isGPSEnabled == false &&
         isNetworkEnabled == false) {
         // no network provider is enabled
      } else {
         this.canGetLocation = true;
         if (isNetworkEnabled) {
            location=null;
            locationManager.requestLocationUpdates(
                  LocationManager.NETWORK_PROVIDER,
                  MIN_TIME_UPD,
                  MIN_DIST, this);
            Log.d("Network", "Network");
            if (locationManager != null) {
               location = locationManager
                 .getLastKnownLocation(
                  LocationManager.NETWORK_PROVIDER);
               if (location != null) {
                  latitude =
                      location.getLatitude();
                  longitude =
                      location.getLongitude();
               }
            }
         }
         // if GPS Enabled get lat/long
             using GPS Services
         if (isGPSEnabled) {
            location=null;
            if (location == null) {
               locationManager.requestLocationUpdates(
                     LocationManager.GPS_PROVIDER,
                     MIN_TIME_UPD,
                     MIN_DIST, this);
               Log.d("GPS Enabled", "GPS
                   Enabled");
               if (locationManager !=
                   null) {
                  location =
                      locationManager
                         .getLastKnownLocation(
                            LocationManager.GPS_PROVID
                  if (location != null) {
                     latitude =
                         location.getLatitude();
                     longitude =
                         location.getLongitude();
                  }
               }
            }
         }
      }
   } catch (Exception e) {
      e.printStackTrace();
   }

   return location;
}

/**
 * Stop using GPS listener Calling this
 * function will stop using GPS in the app
 * */
public void stopUsingGPS() {
   if (locationManager != null) {
      locationManager.removeUpdates(
       GPSTracker.this);
   }
}

/**
 * Function to get latitude
 * */
public double getLatitude() {
   if (location != null) {
      latitude = location.getLatitude();
   }

   // return latitude
   return latitude;
}

/**
 * Function to get longitude
 * */
public double getLongitude() {
   if (location != null) {
      longitude = location.getLongitude();
   }

   // return longitude
   return longitude;
}

/**
 * Function to check GPS/wifi enabled
 * @return boolean
 * */
public boolean canGetLocation() {
```

```
    return this.canGetLocation;
}

/**
 * Function to show settings alert dialog
 * On pressing Settings button will
 * lauch Settings Options
 * */
public void showSettingsAlert() {
    AlertDialog.Builder alertDialog =
        new AlertDialog.Builder(mContext);

    // Setting Dialog Title
    alertDialog.setTitle("GPS is settings");

    // Setting Dialog Message
    alertDialog
          .setMessage(
             "GPS is not enabled.
             Do you want to go to settings
                 menu?");

    // On pressing Settings button
    alertDialog.setPositiveButton("Settings",
          new
              DialogInterface.OnClickListener()
              {
             public void onClick(
                DialogInterface dialog, int
                    which) {
                Intent intent = new Intent(
                  Settings.ACT_LOC_SETT);
                mContext.startActivity(intent);
             }
          });

    // on pressing cancel button
    alertDialog.setNegativeButton("Cancel",
          new
              DialogInterface.OnClickListener()
              {
             public void onClick(
                DialogInterface dialog, int
                    which) {
                dialog.cancel();
             }
          });

    // Showing Alert Message
    alertDialog.show();
}

@Override
public void onLocationChanged(Location
    location) {
}

@Override
public void onProviderDisabled(String
    provider) {
}

@Override
public void onProviderEnabled(String
    provider) {
}

    @Override
    public void onStatusChanged(String
        provider,
        int status, Bundle extras) {
    }

}
```

APPENDIX D
SERVER/__MAIN__.PY

```
# API calls implemented here
from flask import Flask
from flask import render_template
import json
import os
import logdata.incoming
import logdata.predict
app = Flask(__name__)


@app.route('/')
def hello_world():
    return render_template('index.html')


# Depricated
@app.route('/logdata/<userid>/'+
    '<float:latitude>/<float:longitude>/'+
    '<int:weekday>/<int:hour>'+
    '/<int:minutesQuant>')
def logInputData(userid, latitude,
    longitude, weekday,
    hour, minutesQuant):
    # Log the user data
    inputData = logdata.incoming.Data(
        userid, latitude,
        longitude, weekday,
        hour, minutesQuant)
    responseJson =
        inputData.generateResponseJson()
    return str(responseJson)

# Depricated


@app.route('/logdata/<userid>/predictlocation')
def predictedLocationData(userid):
    responseJson =
        logdata.predict.locationPredict(userid)
    return str(responseJson)


@app.route('/location-predict/api/v1/'+
    'logdata/<userid>/<float:latitude>'+
    '/<float:longitude>/<int:weekday>/'+
    '<int:hour>/<int:minutesQuant>')
def logInputDataV1(userid, latitude,
    longitude, weekday,
    hour, minutesQuant):
    # Log the user data
    inputData = logdata.incoming.Data(
        userid, latitude,
        longitude, weekday,
        hour, minutesQuant)
```

```
    responseJson =
        inputData.generateResponseJson()
    return str(responseJson)


@app.route('/location-predict/api/v1/predict-res/'+
    '<userid>/<float:latitude>/<float:longitude>/'+
    '<int:weekday>/<int:hour>/<int:minutesQuant>')
def predictedLocationDataV1(userid,
    latitude, longitude,
    weekday, hour,
    minutesQuant):
    responseJson =
        logdata.predict.locationPredict(
       userid, latitude,
       longitude, weekday,
       hour, minutesQuant)
    return str(responseJson)

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

APPENDIX E
SERVER/INCOMING.PY

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import json
import sqlite3


class Data:

    'Data class. Verifies input data, saves
        and generates output.'

    def __init__(self, uuid, latitude,
        longitude, weekday, hour,
        minuteQuantized):
       self.uuid = str(uuid)
       truncateToDigits = 6
       self.latitude = round(latitude,
           truncateToDigits)
       self.longitude = round(longitude,
           truncateToDigits)
       self.weekday = weekday
       self.hour = hour
       self.minuteQuantized = minuteQuantized
       self.inputValidity =
           self.validateInput()
       if self.inputValidity == 'valid':
          self.saveDataToDb()

    def validateInput(self):
       if self.latitude > 90.0 or
          self.latitude < -90.0:
          return "Latitude {} is not \\\
          in the range between
              +/-180.0".format(
             self.latitude)
       elif self.longitude > 180.0 or
          self.longitude < -180.0:
          return "Longitude {} is not \\\
             in the range between
                 +/-180.0".format(
                    self.longitude)
       elif self.weekday > 6:
          return "Weekday {} is not \\\
             in the range (0,
                 6)".format(self.weekday)
       elif self.hour > 23:
          return "Hour {} is not \\\
             in the range (0,
                 23)".format(self.hour)
       elif self.minuteQuantized > 3:
          return "Quantized minute {} is not
              \\\
             in the range (0,
                 3)".format(self.minuteQuantized)
       else:
          return "valid"

    def generateResponseJson(self):
       responseData = {}
       responseData['error'] = {}
       if self.inputValidity == 'valid':
          responseData['error']['code'] = 0
       else:
          responseData['error']['code'] = 1
       responseData['error']['comment'] =
          self.inputValidity
       responseJson = json.dumps(
          responseData, indent=4,
              sort_keys=True)
       return responseJson

    def saveDataToDb(self):
       print "Entered saveData"
       conn =
           sqlite3.connect('db/locationdata.db')
       print "Connected to database"
       insertSQL = 'INSERT INTO locationlog '+
          '(uuid, latitude, longitude,
              weekday, '+
             'hour, minute_quant,
                 repeated_count)'+
             ' VALUES ("{}", {}, {}, {}, {},
                 {}, 1);'
       insertSQL = insertSQL.format(
          self.uuid,
          self.latitude,
          self.longitude,
          self.weekday,
          self.hour,
          self.minuteQuantized)
       updateSQL = 'UPDATE locationlog SET '+
       'repeated_count= repeated_count+1 '+
       'WHERE EXISTS (SELECT * FROM
           locationlog '+
          'WHERE uuid = "{}" AND latitude={}'+
          ' AND longitude={} '+
          'AND weekday={} AND hour={} AND '+
          'minute_quant={});'
       updateSQL = updateSQL.format(
          self.uuid, self.latitude,
          self.longitude, self.weekday,
          self.hour, self.minuteQuantized)
       try:
          conn.execute(insertSQL)
       except:
          conn.execute(updateSQL)
       print "Table created successfully"
```

```
    conn.commit()
    conn.close()
    return 0
```

APPENDIX F
SERVER/PREDICT.PY

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import json
import sqlite3
from pybrain.tools.shortcuts import
    buildNetwork
from pybrain.datasets import SupervisedDataSet
from pybrain.supervised.trainers import
    BackpropTrainer


def getDataFromDB(uuid,
  weekdayCurrent,
  hourCurrent,
  minuteQuantCurrent):
    conn =
        sqlite3.connect('../db/locationdata.db')
    sqlGetUserData = 'SELECT uuid, latitude, '+
       'longitude, weekday, hour, '+
       'minute_quant, repeated_count '+
       'FROM locationlog WHERE uuid = "{}" '+
       'AND weekday = {} AND hour = {} '+
       'AND minute_quant = {};'
    locationEntryDB = conn.execute(
      sqlGetUserData.format(
        uuid,
        weekdayCurrent,
        hourCurrent,
        minuteQuantCurrent) )
    locationEntryList = list(locationEntryDB)
    return locationEntryList


def generatePredictedJson(uuid,
    predictedLat,
    predictedLon,
    statusSituation,
    statusAction):
    responseData = {}
    responseData['prediction'] = {}
    responseData['prediction']['latitude'] =
        predictedLat
    responseData['prediction']['longitude'] =
        predictedLon
    responseData['status'] = {}
    responseData['status']['situation'] =
        'normal'
    responseData['status']['action'] = 'none'
    responseJson = json.dumps(
      responseData,
      indent=4, sort_keys=True)
    return responseJson


def locationPredict(uuid, latitudeCurrent,
    longitudeCurrent, weekdayCurrent,
    hourCurrent, minuteQuantCurrent):
    net = buildNetwork(3, 4, 2)
    ds = SupervisedDataSet(3, 2)
    userLocDataList = getDataFromDB(uuid,
       weekdayCurrent, hourCurrent,
       minuteQuantCurrent)
    print userLocDataList
    for userLocData in userLocDataList:
       latitudeDB = userLocData[1]
       longitudeDB = userLocData[2]
       weekdayDB = userLocData[3]
       hourDB = userLocData[4]
       minuteQuantDB = userLocData[5]
       repeatedCountDB = userLocData[6]
       for iter in range(repeatedCountDB):
          ds.addSample(
             (weekdayDB,hourDB,minuteQuantDB),
             (latitudeDB, longitudeDB,))
    print("Dataset length: {}".format(len(ds)))
    trainer = BackpropTrainer(net, ds)
    # trainer.trainUntilConvergence()
    trainer.train()
    [predictedLat, predictedLon] =
        net.activate(
       [weekdayCurrent,
       hourCurrent, minuteQuantCurrent])
    statusSituation = 'normal'
    statusAction = 'none'
    print "uuid: "
    responseJson = generatePredictedJson(
       uuid,
       predictedLat,
       predictedLon,
       statusSituation,
       statusAction)
    return responseJson

print locationPredict('apr0041',
 17.45, 78.35, 1, 11, 2)
```