Problem 2: Introduction to Spatial Databases (Relational) and Python Geo-processing Utilities

Problem Submission Guidelines: For this problem, you may work in groups of up to 4. Please include the name of your group members when submitting your code and writeup.

Deliverables are defined in Tasks 1.6 and 2 and Extra credits. Please submit:
1. Your code
2. A write up in PDF

Alternatively, you can include all your code and writeup in a single ipython notebook (if you have non-python code, e.g. bash codes, put them in a mark down cell). When making submission, **you must also submit a PDF version of the ipython notebook.**

Problem Overview and Goals

Many utilities have been written in Python to simplify mundane GIS tasks. While libraries come and go, it is important to understand the aims of an open source GIS project and where it can be useful in simplifying your job working with spatial data.

This problem introduces a powerful tool for spatial data analysis: *PostGIS*. This *free and open source software* (FOSS) product has been extensively developed and supports a variety of large-scale spatial data persistence operations and analytic workflows.

By the end of this assignment your should be able to install PostGIS, understand *relational schema*, load *geospatial data* into the database, and perform simple *spatial queries*. Additionally, you will learn how to find a variety of free geospatial datasets as well as useful ancillary libraries for pre- and post- processing data.

Note: This problem includes a great deal of material; however, the main goal is to provide exposure to the variety of tools that may be employed in solving geospatial problems. **In Task I and Task II, you are required to use PostGis to perform the tasks, it is not optional.**

Problem Tasks

Task I: Introduction to Relational Databases and PostgreSQL Installation (40 pts)

PostgreSQL is a relational database system. A *relational database* defines *relationships* between *tables*, which are collections of *records* (rows) and associated *fields* (columns). Relational databases (also known as *Relational Database Management Systems or RDBMS*) are used for structured data persistence and are manipulated using Structured Query Language (*SQL*). Other popular relational database systems include MySQL and SQLite. In the relational model, data is often *normalized*, which is to say that redundancy within tables is reduced in order to maintain extensibility. This may be in contrast to the *NoSQL* (aka. Not Only SQL) design of database architecture. Examples of NoSQL databases include MongoDB, CouchDB, and BigTable. We will revisit this other style of database design in the next problem

While comparable SQL and NoSQL have been extended to support spatial data operations, thus far, complex spatial queries and efficient spatial data transactions remain within the domain of the RDBMS. PostGIS is the premiere open source spatial database system, and supports many web-scale applications.

Task Instructions:

PostGIS is notoriously tricky to install; however, many PostGIS installation issues may be avoided by first installing the appropriate dependencies for PostGIS. **Please complete the following subtasks sequentially within one week after the homework is released. That would give you more time to actually work with the tools than fuzz around with installation. To enforce this, GSI will not respond any installation questions after this first week.** Though all tools are available for modern OS versions, the specifics will vary by OS and from machine to machine (Some OS-specific recommendations are provided below.

Please read all associated documentation carefully. Understanding the different tools will be very useful in the subsequent problem as well as much of your near-term GIS work!

1. Install PostgreSQL version 9.3.5 for your operating system (9.3.x will do if already installed). Installation instructions may be found at http://www.posgresql.com. It is also recommended to install pgadmin (found at http://pgadmin.org), which can be used to visually inspect and edit your databases as well as make queries and install extensions.

    - For windows, a good tutorial for installation of pgadmin and PostGIS for Windows (with useful guidance as to how to use Spatial SQL for Windows and non-Windows users alike) can be found here: http://workshops.boundlessgeo.com/postgis-intro/.
    - For mac, you may want to use postgressapp or OpenGeo Suite as your first choice. They include PostGis and plugins.

2. Prior to installing PostGIS, you will need to install several Python and C/C++ libraries that serve as either dependencies or fluent interfaces to PostGIS

    - GEOS and Shapely: Python-based manipulation of geometric objects (installation instructions at at https://pypi.python.org/pypi/Shapely and http://trac.osgeo.org/geos/). GEOS comes with Anaconda, if you installed Anaconda, no need to install it separately.
    - GDAL and Fiona: GDAL is used for working with *raster* (spatially discrete) geospatial data. GDAL is OGR *vector* data equivalent. http://www.gdal.org/. Fiona (https://pypi.python.org/pypi/Fiona) [https://pypi.python.org/pypi/Fiona] provides python bindings to GDAL/OGR.
    - PROJ.4 and pyproj: pyproj is a wrapper for the *projection* library PROJ, which is now in version 4. Depending on your operating system, you may need to install the two libraries separately.

3. Preferably, you may want to access PostGis in python. Install PsycoPg2: a Python interface to user data. Installation instructions at: http://initd.org/psycopg/docs/install.html. The installation of PsycoPg2 might be a little bit trickier. If the methods in previous link failed,

    - For windows user, you can try installing the package with conda:
        - conda install -c https://conda.binstar.org/topper psycopg2-win32-py27 (for 32-bit)
        - conda install -c https://conda.binstar.org/topper psycopg2-windows (for 64-bit)
    - For mac users, there are a few options to try:
        - Use Synthicity repository:
          conda install -c https://conda.binstar.org/synthicity psycopg2
        - Use fink or MacPorts.
        - Run `pip install psycopg2`. This might require setting some environment variables and might not be compatible with Anaconda.

    If all those method failed, you may contact GSI for help.

4. Assuming 1-3 were complete successfully, create a template geospatial database by running the following in a terminal. (Note: per common practice, the prompt is represented by a `$` or `>` here. Your prompt may differ. Do not type these characters literally. All syntax following the prompt should be executed exactly as written):

```
$ createdb template_postgis
$ psql template_postgis
```

Your prompt should have just changed. You have launched the `psql` console with access to your `template_postgis` database as the root user for your RDBMS (Login roles and credentials were determined during PostgreSQL installation. You may need to enter the above commands by prefixing `sudo`. See the documentation, and then ask the GSI *re:* troubleshooting).

Still in the psql console, enter the following to equip the database with a spatial template (caps are optional, but traditional for distinguishing SQL keyword commands):

```
> CREATE EXTENSION postgis;
> CREATE EXTENSION postgis_topology;
```

Note the `;`, which ends each statement in SQL. You may wish to change the user or use this database as a template to create other PostGIS-enabled databases. The following commands will do the job (the words in angle brackets are not literal, but the quotes for the password and encoding are):

```
> CREATE USER <username> password <password>;
> CREATE DATABASE <db_name> OWNER <username> TEMPLATE template_postgis ENCODING utf8;
```

Quit the `psql` prompt by entering `\q`.

5. Additional visual demo of QGIS. You may want to use it for some visualizations of some shape files in this assignment but **it is not necessary to know how to use QGIS to complete this assignment**. Note that you may need to have some dependencies including GDAL, and some python packages installed prior to install QGIS. Follow the guide.

6. Please refer to the documentation for shapely, Fiona, and PostGIS and provide a response to the following (bullet points and tables are OK, just cover the major points): **For what sorts of geospatial tasks and data analysis situations would you use one library vs. another? In your response, consider use cases involving ad-hoc data analysis, long-term data storage, as well as processing speed, data distribution, and data format issues.**

Task II: Spatial Relational Databases and Spatial Queries (60 Points)

If you have followed the installation instructions, then you hopefully now have a working PostGIS installation and a spatially-enabled database. To verify that you can actually access PostGis from python, open a Python console of your choice and import the psycopg2 library, connect to the database you have previously created, and run some commands; verifying error-free execution.

```
>>> import psycopg2
>>> conn = psycopg2.connect("dbname=<db_name> user=<username> host=localhost password=<password>")
>>> cur = conn.cursor()
>>> cur.execute("""SELECT srtext FROM spatial_ref_sys WHERE srid = 32610;""")
>>> rows = cur.fetchall()
>>> for row in rows:
        print "    ", row[1]
```

This should print the spatial reference identifying (SRID) information for Northern California in the NAD83 Datum. Spatial references are used for *projecting* geospatial data. Essentially, for validity within a planar Cartesian framework of relational algebra operations, projections permit topologically sound transformations. Different projections are used for a variety of cartographic and geospatial analytic purposes over a range of geographic extent. For the tasks in this assignment, we will be using the SRID designated EPSG:32610 for Northern California (Universal Transverse Mercator: UTM 10)(WGS84 datum) and EPSG:32611 for Southern California (UTM 11). The site http://spatialreference.org/ is, perhaps, the definitive reference for SRID

information; however, PostGIS comes with a number of spatial references pre-installed in the table `spatial_ref_sys`, as illustrated above. For more info about coordinate system check out this ppt

Alternatively, you may also use EPSG: 4326, which is WGS 84 coordinate for the assignment.

Once the coordinate reference system is set for a table, spatial queries may be performed.

Task Instructions

Your system should now be configured to complete this task. You are expected to consult external documentation to complete this task. Any available tutorials, how-to books, or Stack Overflow entries are valid resources that may need to be consulted. You are encouraged to use any additional helpful python libraries you may come across. If you find a particularly useful resource, please post on Piazza.

1. Create a new database for this assignment using the `"template_postgis"` as the template.
2. Parse the geographic coordinates from the 100K Twitter feed used in the previous task into a format that can be inserted into the PostGIS database. *Hint:* consider using the GeoJSON (http://geojson.org/) specification as an intermediary transformation.
3. Insert the 100K Tweets together with associated user information, timestamp, location and content into the spatial database, using the geographic coordinates from previous question.
    - Example: `cur.execute("INSERT INTO Tweets(id, userid, loc, time, text) VALUES (%s, %s, ST_GeomFromText(%s',SRID='%s), %s, %s)", (_id, user_id, 'POINT(-122,37)', '4326', datetime.datetime(), text))`
    - Note that `'time'` entry should be of type timestamptz, and you can directly give a datetime.datetime python object to it.
    - A great datetime parser from string is dateutil.parser.parse.
4. Download county tracts for the State of California from http://www.census.gov/cgi-bin/geo/shapefiles2010/main
    - Selected Counties (and equivalent).
    - Select California;
    - Downloaded the census population data (P1):
5. Convert these into a form suitable for insertion into your database and do so.
    - Example:`shp2pgsql -I -W "latin1" -s 4326 <yourDeirectory>/County_2010Census_DP1.shp public.ca_census_tract | psql -d <dbname>`
    - Or you can use the pgshapeloader GUI comes with OpenGeo suite.
6. Calculate the number of tweets inside of Contra Costa County.
7. How many Tweets fall 100 miles outside of Alameda County? (i.e., fall outside of a 100 mile polygon surrounding the Alameda County).

8. Insert the 2010 Census population per county into your database. You can get the data from http://census.ire.org/data/bulkdata.html?state=11&sumlev=140 for County as `.csv`. Please fixed the `.csv` file to remove any `'.'` in the headers.

    - Create a new table to store this population data

```
$ psql dbname
dbname-# CREATE EXTENSION postgis;
dbname-# CREATE TABLE ca_census_data
(GEOID varchar(11),
SUMLEV varchar(3),
STATE varchar(2),
COUNTY varchar(3),
CBSA varchar(5),
CSA varchar(3),
NECTA integer,
```

```
CNECTA integer,
NAME varchar(30),
POP100 integer,
HU100 integer,
POP1002000 integer,
HU1002000 integer,
P001001 integer,
P0010012000 integer);
ca_census-# \q
```

- Insert the csv data to the table

```
$ cat all_140_in_06.P1.csv | psql -
dbname -c 'COPY ca_census_data FROM
STDIN WITH CSV HEADER'
```

9. Using this information, provide a visualization of tweets per-capita for California counties. **To get full credit, you are only allowed to perform one query to the database**. (Hint: use hierarchical join). More sql information can be found at this tutorial. For this assignment, it is fine to provide a histogram or bar chart using matplotlib or any other visualization tool that you like; however **5 bonus points will be awarded for an appropriate cartographic representation**.

10. **Bonus (10 points)**: Find the centroids of the DBSCAN tweet clusters from Problem 1 (10,000 Tweets). Prepare a table that shows the minimum radii necessary to fully contain these clusters.

Task III (Extra Credit): Introduction to NonSQL Database (20 pts)

This problem provides you with an experience of using a modern document-based database with (limited) spatial data processing capabilities.

The database used in this assignment is MongoDB . MongoDB is a cross-platform NoSQL document-oriented database allowing for JSON-like documents with dynamic schemas. MongoDB was build with scalability in mind. MongoDB can be installed from the official website .

MongoDB also has a Python API: the pymongo module contains the code and examples needed to connect to and query the database.

MongoDB supports 3 types of GeoJSON objects: points, line strings and polygons. The newest spatial index, the 2dsphere index, supports queries that calculate geometries on an earth-like sphere. This is the recommended index for this assignment.

Using this index you can make the following queries:

- `$geoWithin`
  queries for location data found within a GeoJSON polygon
- `$geoIntersects`
  queries for locations that intersect a specified GeoJSON object
- `$geoNear`
  return the points closest to the defined point and sorts the results by distance

This assignment deals with creating, inserting data into and querying a Mongo database. Each tweet should be augmented to contain its cluster id as well as all original attributes saved in the GeoJSON format. In Parts 1 and 2 of the assignment, you will create a database and populate it with tweets. You will then perform spatial\temporal queries on the database to explore and find interesting clusters. You will be querying the data using a spatial index. Indexes allow for faster data retrieval (at the cost of additional writes in the database). An example of creating a spatial index is given in the appendix below.

Note: This assignment relies on the results obtained in HW1. The tweets and corresponding cluster id found in Part 3 of HW assignment 1 are to be added to the database. A spatial index will then be added to the database and queries will be performed using spatial and temporal features.

It is recommended that you use `$geoWithin` when finding points within a polygon. `$geoIntersects` is a more general query for dealing with geometries.

Task Instructions

Part 1. The database (5 points)

1. Follow the on-line instructions and install MongoDB and pymongo on your working machine.
2. Load the tweets from file and augment them to contain the appropriate cluster id and GeoJSON coordinates (http://geojson.org/) coordinates.
3. Write a script that inserts all 1M tweets into a mongo database. It is important to note that if you add two identical tweets to the database one will not overwrite the other. Instead both will be treated as unique documents and will be added to the collection. The mongo generates an attribute `'_id'` used as a primary key to indicate unique documents in the collection and this value should not be changed manually. Note that if you create and populate the database twice without deleting any of the old documents, you will have a collection of 2 million tweets.
4. Create a mongo spatial index

Part 2. Querying the database (15 points)

Write Mongo queries enabling to:

1. Find all tweets of the user with id 1138308091
2. Find the 10 closest tweets to the tweet with id 378189967014379520
3. Find the number of tweets that are contained within the polygon: `{'type':'Polygon','coordinates': [[[-122.412,37.810],[-122.412,37.804],[-122.403,37.806],[-122.407,37.810],[-122.412, 37.810]]]}`

Appendix for MongoDB

Connecting to database:

```
from pymongo import MongoClient
mongo_client = MongoClient()
mongo_db = mongo_client.san_francisco_db
tweets = mongo_db.tweets
mongo_client.close()
```

Add document to collection:

```
tweet = { 'id': 1234 ,
                'text' : 'hello_world',
                'user_id':1234,
                'location' : {'type':'Point','coordinates':[-122.4167,37.7833]}}
tweets.insert(tweet)
```

Adding Geospatial index:

```
tweets.create_index([('location','2dsphere')])
```

Get all tweets from the user with id 1234

```
user_tweets = tweets.find({'user_id':1234})
for tweet in user_tweets:
    print(tweet)
```

Update the tweet with id 1234 to have a new attribute called cluster_idvwith a value of 12

```
tweets.update({"id":1234},{"$set":{"cluster_id":12}})
```

Delete the tweet with id 1234 (Removed documents cannot be recovered)

```
tweets.remove({"id":1234})
```

Additional Useful Resources:

- **Command Line Help:** Do not be intimidated by the title. This covers the basics and then some: http://cli.learncodethehardway.org/book/.
- **TIGER/Line Data Access:** US Census-maintained political boundaries, roads, and other geographic features maintained in a variety of formats. http://www.census.gov/geo/maps-data/data/tiger.html. Find the tl_2015_06_tract.zip, which contains the cencus tract information for California (whose state code is 06).
- **Census API:** Convenient, pythonic census data access https://github.com/sunlightlabs/census.
- **DataScience Toolkit**: An API for a variety of useful statistical and geospatial APIs http://www.datasciencetoolkit.org/.
- **UC Berkeley GIF Resources**: While you should check out the GIF for all of your UC Berkeley GIS needs (workshops, free commercial software such as ArcGIS, and computer labs), the resources link provided here is a portal to a variety of geodata you might find useful or interesting http://gif.berkeley.edu/resources/data.html.
- **GeoPandas**: Easily load, manipulate, plot, and visualize geospatial data. Familiarity with the Pandas framework, in general, is very helpful for all sorts of data manipulation activities. It does have a slightly steep learning curve, but is well worth the effort http://geopandas.org/.