NationBrowse

Making sense of civic data

CS4970 Group 2

Graham Greenfield
Jeremy Howard
Nick Roma
Mike Tigas

Live URL

http://capstone.nationbrowse.com/

Source Repository

http://github.com/mtigas/cs4970_capstone

Table of Contents

List of figures & tables

Executive summary

NationBrowse is a Web application that provides a browsable front-end to civic data provided by the United States Government, through the Data.gov Web site. The primary goal of the Web application is to not only provide an easy-to-use system for browsing statistics, but to also offer statistical analysis and comparison functionality.

The sheer amount of data available from the 2000 United States Census presents a significant problem to those who may be interested in demographic information. (Note that the 2000 Census represents only a fraction of available datasets on Data.gov.) Providing an application interface to assist in analyzing the data would be extremely useful to research organizations, marketers, investigative reporters, and the public at-large.

Beyond tables of numbers and statistics, the application will support dynamic generation of charts and graphs of the stored information. By presenting data in graphical form, it is hoped that the application will be usable to a wider variety of non-research audiences. In addition, map-based browsing of the data will be implemented; for example, a dataset such as population size could rendered as a color-coded map of the respective geographic areas — states, counties, ZIP codes, etc.

For those performing research on the data, a graphical user interface to "build" queries, beyond those of the pages that are browsable, will also be provided. The ability to customize the acquisition of data, generate statistical analyses, and generate graphical representations of the data should prove useful in a research environment.

Problem definition

The overwhelming amount of information available via Data.gov makes it difficult for people and organizations to use. While releasing large amounts of government data presents a step toward transparency, making the data useful actualizes the goal of the government's Data.gov project.

Census data is not new — the United States Government digitally released the results in 2001 and these datasets have been widely used. However, only recently have Web technologies evolved to the point where a user-centric browsable presentation of the data is possible.

Few dynamic applications to civic data have existed until very recently. Since the launch of the Data.gov project in 2008, Sunlight Labs, a pro-transparency think tank, has sponsored an annual contest for applications (Apps For America) that use civic data from the Data.gov Web site.[1][2] Apps For America recognizes applications of Data.gov datasets that provide a useful tool to the public. Several apps from the Apps For America 2 contest appear to have goals that fall in-line with those of NationBrowse:

- ThisWeKnow[3] provides a browsable interface of many Data.gov datasets, but does not perform any analyses or provide any dynamic element to the browsing experience. There are no maps or alternative perspectives on the data, everything is simply presented as a list of trivia regarding the user's selected location.
- DataMasher[4] provides a way to compare states against one another by "mashing up" two disparate datasets. For example, one may divide crime rate by high school graduation rate, and create a color-coded map that displays that relationship in each state. DataMasher admits that the Web site's output is not statistically rigorous, but is simply a tool to drive conversations regarding the data.

In addition to Web sites featured on Apps For America, several other Web sites have inspired this project either through use of civic data or the use of specific open-source technologies:

- The Los Angeles Times launched a "neighborhood mapping" project[7] that provides lists of demographic information for each L.A. neighborhood, in a very similar vein to the ThisWeKnow project, above.

- EveryBlock[5] is a Web site developed by several former programmers in the news industry, which provides aggregation of local civic data (police records, building permits) in addition to news stories and other information. EveryBlock provides a custom map interface to much of it's data. Paul Smith, a primary developer for EveryBlock's mapping infrastructure, recently wrote about the open-source GIS stack that EveryBlock uses.[6] It is hoped that a implementation of this "stack" will drive some portions of NationBrowse's mapping applications.

- FiveThrityEight[10] is an poll-tracking Web site that follows state and national elections. During the 2008 U.S. Presidential Election, the site provided maps with projections of election results, based on data acquired from a large variety of pollsters. The current blog often displays charts and graphs of relevant data. It appears that these charts and graphs are manually created and not a result of dynamic generation from a database or data source.

- The *New York Times* releases a variety of mapping projects every few months, much of it based on civic data covering New York City. As with FiveThirtyEight, election maps were created during the 2008 election. A current example of their dynamic mapping projects is a map of homicide locations within New York City,[11] via a database updated from New York Police Department statements.

A quick look at the above sources shows the wide variety of tools available to browse demographic data. However, few of the above products creates any sort of statistical analysis regarding the wide variety of data supplied by the U.S. Government. Mapping and automated chart generation is also lacking — many maps and graphs are one-off creations in these products, while our goal is to provide coverage of these tools for *all* locations in the datasets. It is our hope that NationBrowse allows a more thorough and useful (in a significant research sense) look into the information that the United States Government has released.

Requirements analysis

System constraints

As a data-driven Web application, NationBrowse is inherently constrained by the availability and usability of data supplied by the source(s). As of October 2009, the data will be manually imported into a database internal to the application. Because the data model for demographic information is generic, the opportunity for automated imports and updates exists, but relies on consistent formatting from the Data.gov sources.

The project is intended to be a publicly-accessible web site. The resources required to render data, chart images and maps could affect usability of the application — usability is constrained by the performance of server and database components. Some portions of the map renderer may rely on external services, such as Google Maps or other accessible map tile sources.

Performance requirements

The project must be accessible by modern Web browsers, over HTTP. The server must dynamically serve map tiles to the client — performance overhead regarding this is alleviated by the use of TileCache software.

NationBrowse will require statistical analyses to be performed over various datasets. Some functions (mean, standard deviation) are built-in to the PostgreSQL database. Others are implemented in the SciPy Python library[1] that the project will use, while other algorithms may have to be developed.

Image generation for charts could produce another significant stress on the application server. Therefore, “simple” charts for non-comparative data (pie charts or bar charts for demographics) will

[1] SciPy provides a Matlab-like programming interface to many useful mathematic and statistical computations.

be delegated to the Google Charts API, while more complex charts dealing with comparative data (place-place or variable-variable correlations) will be rendered directly by the application.

Resource requirements

The application software requires an HTTP Web server with access to Python. The map rendering and GIS portions of the application are built toward the PostgreSQL database using PostGIS extensions.

The data that drives the application comes from the United States Government on Data.gov. Specifically, GIS data will be sourced from the Census Bureau's TIGER/Line shapefiles, which provide detailed shape/border data for all states and territories, all counties, and all ZIP codes within the United States. Other data (demographics) will be sourced from Census Bureau data and other datasets available on Data.gov.

The Django framework includes a caching framework so that rendered pages and data do not have to be processed as frequently. For best performance, the Django documentation recommends the use of the memcached cache server software, although the Django caching framework may use a disk-based cache if this is not possible.

Alternative solutions

We chose Django[12] as the overall application framework, because it provides an ORM (object-relational mapping) API that handles much of the database work. From a technical standpoint, the database, Web page, and statistical portions of the application could be implemented in any Web-ready language (such as PHP), but Django and Python provide much in terms of utility, so that many low-level tasks (such as interfacing with the database or serving templated pages) are made simpler. Using Django and Python allows us to concentrate our efforts on the statistical and mapping portions of the application.

SciPy[13] and MatPlotLib[14] are being used as the statistics backend and chart generation portions. These provide similar analysis and charting functionality to R and Matlab, but through the Python language. This provides excellent integration with the Django-powered front-end. MatPlotLib is not as well-known as R and Matlab, but users and contributors include researchers at NASA's Jet Propulsion Laboratory and the National Oceanic and Atmospheric Administration (NOAA)[15].

Mapnik was selected as the GIS backend because of it's free and open-source nature and compatibility with other portions of the application. Alternative solutions include MapServer, another free and open-source project, and ESRI's ArcGIS Server, which is a commercial product. All three provide similar levels of map image generation, but vary in levels of database backend support. MapServer and ArcGIS Server only support PostgreSQL and PostGIS (our database backend) through third-party adapters, while Mapnik provides native support for this database stack. In addition, Mapnik utilizes Python, the primary language for this project, while the alternatives are provided in either C(++) or binary form.

In terms of full implementation of color-coded maps, ESRI also provides a commercial product, InstantAtlas Server. This product, which implements a front-end in Adobe Flash, generats both maps and charts of given data. It is marketed primarily as a "visual presentation tool" and does not provide any statistical analysis nor the ability to interface with and compare multiple datasets at once.

Evaluation metrics

As a Web application, the project will mostly be evaluated on feature-completeness and performance metrics. Django and Python offer various methods to test performance: the primary performance metrics would be server response time (or page load time), CPU load, and database load (as measured in number of queries).

Statistical analyses should also prove “sound” based on accuracy and correct use of statistical algorithms. Dr. Lori Thombs, director of the Social Science Statistics Center at the University of Missouri, will assist the team in developing the statistical algorithms and analyses.

Design Specifications

Overview

The project will provide an interface and tools for people to browse and analyze civic data. The project will generate graphical representations of data, in the form of maps and charts, and allow the user to also perform basic statistical analyses on datasets given.

Software design

The program will be built in the Python programming language and use the Django application framework. The SciPy library will also be used to provide mathematical/statistical features in the application.

The front-end will be HTML/CSS-based and use several Javascript-based libraries, including jQuery (for basic effects and HTML manipulation) and OpenLayers (for generating maps).

Data requirements

The project will use PostgreSQL as the database server backend, because of the availability of the GIS extension, PostGIS, which allows the database to be queried based on geographic information.

The data is sourced from Data.gov and Web sites related to that catalog, including the United States Census. This data is provided in a CSV (comma-separated values) format, which is easily imported into the application database.

Geospatial data will also come from the United States government, in the form of Census TIGER/Line data. PostGIS provides tools for importing the data into the database, from the format that the TIGER/Line data is provided in (ArcGIS .shp files).

Hardware

The primary back-end of the application should conceivably run on any server capable of running a Web server (Apache, lighttpd, nginx, IIS, etc.), Python, and PostgreSQL. To take advantage of geospatial data, the server should also be capable of running Mapnik & TileCache.

In a large-scale deployment, the Web application server, the database server, the map/tile servers, and the hosting for "static" files (CSS, images, etc.) could (and should) be separated.[16] In a small, prototype environment (i.e., the scope of this class project), the scaling of resources is not necessary.

Testing methods

Django has built-in test suite support[9], based on Python's built-in unit testing support.

Implementation resources & costs

The primary costs underlying NationBrowse relate to the operational costs of the Web server. These costs vary from under $10 per month for a low-end shared server setup, to several hundreds of dollars for more elaborate setups. (The determination of server requirements will inevitably be based on the number of end users and server load.) Additionally, domain name registration fees apply.

Product implementation

In a general sense, our product implementation will consist of a web application. The project will be constructed using the Django open source framework, which uses the Python programming language. We will take advantage of the Javascript library, jQuery, for the implementation of effects for the front end interface that will make the application look more polished and professional to the end user.

A user would be presented with a frontend to browse locations available in the database: states, counties, and zip codes. Maps would be presented, displaying some of the demographic data — the map would also provide navigation to pages with specific data for the selected location.

The ability to perform statistical analyses will be available to the user. User-generated queries, implemented in jQuery, would provide a request to a custom API on the server side. This could be as simple as providing an HTTP request string to a particular URL, i.e. `/example/?region_1=MO&region_2=IL&datasets=total_population,race`. (This is only a theoretical example.) The browser would then show the user the query result on the page.

Chart generation would be performed by using similar HTTP request strings, which would return a PNG image to the browser. The generation of this type of request string could be handled within a template.

Evaluation of product performance

The evaluation of our products performance will be through the way of measuring the amount of time it takes for the users query to be fully processed and displayed graphically by the front end of our web application.

Discussion, future work, and conclusions

References

[1] "Apps For America." 1 April 2009. Retrieved 21 September 2009. <http://www.sunlightlabs.com/contests/appsforamerica/>

[2] "Apps For America 2." 8 August 2009. Retrieved 21 September 2009. <http://www.sunlightlabs.com/contests/appsforamerica2/>

[3] "ThisWeKnow." Retrieved 20 September 2009. <http://www.thisweknow.org/>

[4] "DataMasher." Retrieved 20 September 2009. <http://www.datamasher.org/>

[5] "EveryBlock." Retrieved 20 September 2009. <http://www.everyblock.com/>

[6] Smith, Paul. "Take Control of Your Maps." *A List Apart*. 8 April 2008. Retrieved 20 September 2009. <http://www.alistapart.com/articles/takecontrolofyourmaps>

[7]"Mapping L.A. Neighborhoods." *The Los Angeles Times*. Retrieved 20 September 2009. <http://projects.latimes.com/mapping-la/neighborhoods/>

[8] "Slicehost." Retrieved 23 September 2009. <http://www.slicehost.com/>

[9] "Testing Django applications." Retrieved 22 September 2009. <http://docs.djangoproject.com/en/dev/topics/testing/>

[10] "FiveThirtyEight: Politics Done Right." Retrieved 5 October 2009. <http://www.fivethirtyeight.com/>

[11] "New York City Homicides Map." Retrieved 6 October 2009. <http://projects.nytimes.com/crime/homicides/map>

[12] "Django." Retrieved 25 September 2009. <http://www.djangoproject.com/>

[13] "SciPy." Retrieved 8 October 2009. <http://www.scipy.org/>

[14] "matplotlib: python plotting." Retrieved 8 October 2009. <http://matplotlib.sourceforge.net/>

[15] "Credits." Retrieved 8 October 2009. <http://matplotlib.sourceforge.net/users/credits.html>

[16] "Ch. 20: Deploying Django: Scaling." Holovaty, Adrian and Jacob Kaplan-Moss. *The Django Book*. <http://djangobook.com/en/1.0/chapter20/#s-scaling>

Response to critique

The main critiques from Mohammed and Dr. Kazic related to finding further competitor products and more specifically defining the innovative portions of the project. More support for our arguments have been made over the past two drafts, particularly with regard to the use of statistical analysis and graphs in the project. In addition, more competitor or precursor products have been identified since the initial draft was submitted.

We have no issues with the critique from group 2. (Attached.)