**Pre processing**

Due to db size and our need to have a dynamic and responsive environment we performed the following preprocessing steps:

1. We download with the mongoexport command each item in the newsItems\_Events table as a single object.
2. We wrote a script that preforms the following
   1. We loaded the ReverseGeoCode package with appropriate coordinates (allCountries.txt ,source http://www.geonames.org/)
   2. For each object from the newsItems\_Events we ran the ReverseGeoCode package on the first georss:point.
      1. If it had a georss:point , we added the following key pairs:

Country (e.g. “contry:russia”,” contry:”japan”)

State Code (e.g. “stateCode:AZ”-Arizona ,”stateCode:AL”-Alaska)

* + 1. If it didn’t had at least 1 georss:point we ignored the object.
  1. If the country code for the object is the US, then we consider this a valid object.
  2. Each group of 2000 valid objects (i.e. with a georss:point field and in the us) we group into a json array.
  3. Each array we save in a json file named part+ # of array .json (e.g. the third array is kept in part3.json file located in the data/newsItemsparts folder).
  4. The scripts splits the files because both Git and Firefox had problem handling very large json files (Git does not allow more then 1.g file and Firefox kept crashing).

All code for preprocessing is located in geo.zip, our preprocessing program is in the mainGeo file.

Remark: Due to Firefox slow down we use only the first 30 files (60000 records) but it is programmatically possible to use all files.

**Data set**

**newsItems\_Event :**

* $oid (key): 56c469e45adbab1a826c062f
* emm:entity list: FBI, Global Positioning System, Tim Cook.
* PERSON list: Cook","Tashfeen Malik","Farook","Tim Cook","Syed Farook
* Updated: 2016-02-17T12:04:00Z
* Polarity: 1.340000033378601
* Country(calculated): US
* state Code(calculated): CA

**state table(constant,size=52):**

* id: 0
* ShortName: AL
* FullName: Alabama

**Visual Mapping**

* Map type: Choropleth map
* geographic area mapped (background)
* average polarity of state for period 1 mapped to color 1 (rgba) –see algorithm explanation
* average polarity of state for period 2 mapped to color 2 (rgba) –see algorithm explanation

**Color mapping algorithm**

* Input: time period 1, time period 2, keyword, color scheme (2 colors), number of data classes, min and max opacity value.
* The algorithm receives two fixed colors (the user may chose his own)

Color1 and Color2.

* The algorithm calculates the average polarity of each state with the given keyword for each time period.
* The algorithm finds the max and min average polarity for all states.
* The algorithm calculates according to the number of data classes ,ranges for each value(number of ranges will be equal to the root of number off data classes)

Example:

Min average polarity=0

Max average polarity= 1.5248863629319451

Number of data classes =9 root of number of data classes =3

Size of each range= (Max average polarity-Min average polarity) /root of number of data classes=0.5082954543106484

Range 1:0-0.5082954543106484

Range 2:0.5082954543106484-1.0165909086212968

Range 3:1.0165909086212968-1.5248863629319451

Remark: This is simplified example the algorithm can also handle negative average polarity values.

* The algorithm calculates the opacity of each range

Example:

Min opacity =0.10

Max opacity =0.40

Number of data classes =9 root of number of data classes =3

Opacity for range 1=0.10

Opacity for range 2=0.25

Opacity for range 3=0.40

* The algorithm calculates the colors that will be used (number of colors is equal to Number of data classes) , where each color is superimposed from the two colors with the calculated opacity value(see diagram) (i.e. the algorithm calculates an RGBA color)



Y- Average polarity

On Date period 2

X- Average polarity on

Date period 1

R0

R1

R2

R3

R1

R0

R2

R3



Example:

Min opacity =0.10 Max opacity =0.40, Min average polarity=0

Max average polarity= 1.5248863629319451, Number of data classes =9

Color 1(RGB) :( 1, 133, 113)

Color 2(RGB) :( 166, 97, 26)

Opacity for range 1=0.10

Opacity for range 2=0.25

Opacity for range 3=0.40

X=0.6

Y=1.4

New opacity = Opacity for range 2+( Opacity for range 3)\*(1- Opacity for range 2)

New opacity =0.25 +(0.40)\*(1-0.4))=0.49

r = round ((color1(r) \* Opacity for range 2)+ color2(r) \* Opacity for range 3\*(1- Opacity for range 3))/new opacity);

r = (1\*0.25) +166\*0.40\*(1-0.4)/0.49=82

if (r>255) {

r=255;

}

New RGBA color (82,G,B,0,49)

Features

* Color-blind mode - basing our work on color brewer, we added a color-blind mode, that uses the same colors, with only access to one color-blind person (color blinds type bla bla), we can report that color recognition for the colors in the color-blind mode was complete, but due to small test sample this is still not a good indicator.
* Legend - we added a map legend where you can see what each color means, the legend also contains help, which explains how the values were reached.
* Highlighting - in the legend a user may choose a color or colors which he wishes to highlight. When pressed this will cause all other colors to black out. This was added due to request by user testers who reported the map was “too busy” and it was difficult to identify trends.
* Color testing - this option was added in order to test the ability of users to identify all colors in a single map. It randomizes color distribution and uses all color hues in a single map.
* Hover info –when hovering over a sate
* Word selection
* Word exclusion
* Help
* Range mimaztion
* Max and min opacityseeting
* Random test

Development process and changes

Color testing

Color randomizer

Worth (formula)

Human feedback

Colorblind mode

Abilities (is it scalable)

Mission types completion