Geographic Audience Segmentation for Real Time Bidding - Documentation

Blaž Dolenc (63120184)

May 28, 2015

1 About

This document contains documentation and results. All needed files (additional data sources, KML) are available for download on `https://drive.google.com/file/d/0B9VLz7VBi4yFWXhVS2k1ekVOYkE/view?usp=sharing`. Code is also published on `https://github.com/bdolenc/Zemanta-challenge`

2 How to test provided code

The code is divided in four files. The easiest way to test the code is with Anaconda python package `https://store.continuum.io/cshop/anaconda/`, which contains all needed libraries. 64bit python should be used for running code on whole dataset.

FileHandler.py - this script transform input data to shape ready for clustering. All files needed for running this script are available on download link provided.

ExploratoryDataAnalysis.py - After all data is transformed to one csv file (FINAL.csv), this script can be executed to get clusters. Results of clustering are saved to .csv file. After clustering averages function should be called, to get average statistics for each cluster. Function create csv file, needed for creating KML for visualisation of points.

KMLCreator.py - This script generate KML file. For coloring points, style tags with color had to be added manually. KML file is also provided on the link for download.

StatisticalModelling.py - after all files are prepared, this script perform random forest and stacking, reporting AUC on training set, split in training and test set.

3 Dataset

The first step was organizing dataset in shape useful for applying data-mining techniques. First column of my dataset contains ZIP, while corresponding row contains all the data I decided to include. Every row contains ZIP, and all the features extracted from provided data. Not all establishments data were included - for establishments by sector I keep only Total number of establishments by sector, because most of the features are 0, and to maintain my dataset easier to manage performance wise. Even though I still experienced some performance problems.

3.1 Additional data sources

Because first clustering attempts were not satisfying I included additional data found on census website.

First was demographic data, from which I extracted columns describing percentage of residents of certain race (Caucasian, Afro-American, Hispanic etc.) Another data I included was about age (percent of residents in from 10-20 for example) and males per females ratio. This, merged with already provided data improved clustering results.

All additional data sources are included in folder on the link provided.

4 Used libraries

Because provided data was pretty large, mostly because of inefficient data organization, I decided to use Pandas library, which is well supported, and can handle large amount of data pretty well.

Pandas performed well, especially on joining data from different sources. Bellow is an example code for joining unemployment and population data frames.

```
merge = pd.merge(df_unemployment, df_population,
how='left', left_on='ZIP', right_on='Zip/ZCTA')
```

For clustering and statistical modelling I used Scikit-learn library. The advantages are good documentation, with plenty of examples, and a lot of available classifiers.

Matplotlib was used for plotting points for first overview, and xml minidom library for creating KML file.

5 Exploratory Data Analysis

5.1 DB Scan

First attempt for finding related ZIPs was with DB Scan clustering algorithm. I used scikit learn implementation of DB Scan. The trouble with DB Scan was, that it clustered almost all zips in only two groups - outliers and one cluster. Even with different normalized or scaled data, and different distance between points the results was odd, as shown on figure bellow:

There are 320 clusters, all on the right, marked with different colors, while all other black points are outliers. The visualisation was done with Principal component analysis, which allow us to plot multidimensional data in two dimensional space.

5.2 Hierarchical clustering

Second attempt on clustering was greater success. After some attempts with different linkages, I got more balanced clusters using Agglomerative clustering from scikit library, with ward linkage. Instead of distance metrics, ward is measuring similarity as an analysis of variance. Visualisation of clustering is on Figure 2.

Bellow is listed code, with which labels for each zip code are obtained. Colors used for visualisation are also calculated.

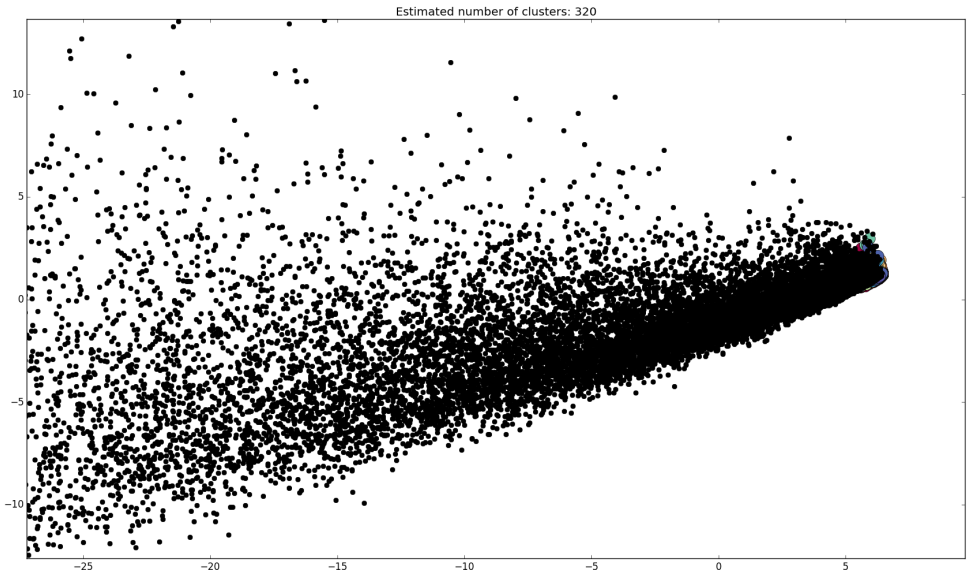

Figure 1: Visualisation of DB Scan clusters using PCA

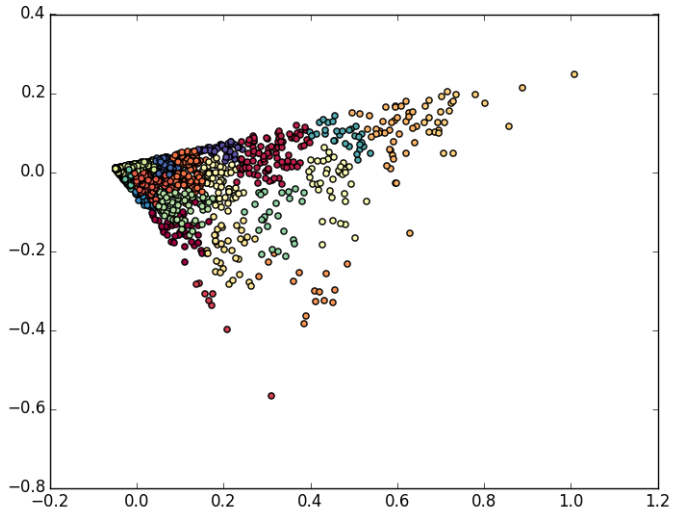

Figure 2: Visualisation of ward clustering using PCA

```
hc = AgglomerativeClustering(n_clusters=40, linkage='ward').fit(data)
labels = hc.labels_
unique_labels = set(labels)
colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_labels)))
```

Clustering results are included in provided folder, in file hc results.csv.

5.3 Visualisation of clustering results on map

Visualisation above (using PCA) aren't as informative and intuitive as they should be. They are also highly depending on used normalization or scaling of the data. The best way to visualize geographic data is, of course, plotting them on a map.

After some research of different options (Google maps, Bing maps), one problem appear. There is around 30 000 zips to plot, but both Google maps and Bing maps have restrictions for geocoding that large set of addresses. Luckily, I found database of ZIP codes, already having latitude and longitude for each ZIP. Please note, that some coordinates are not 100% accurate, so some ZIP codes are not plotted accurately.

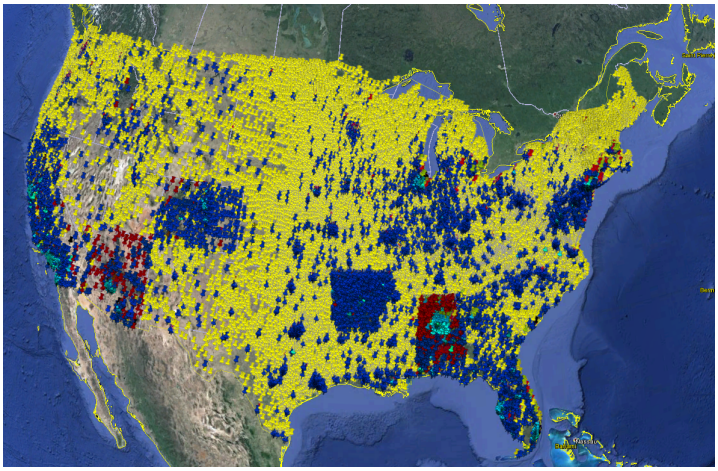

Figure 3: ZIP codes of USA

KMLCreator.py contains code, that create KML file, which can be imported in Google Earth. Every ZIP code is shown as push pin, membership in cluster is marked with different colors, ie. push pins of the same color belong to same clusters, so we can assume they are similar.

KML file is easily imported in Google Earth, using Open command in the application. Quick look on a map reveal, that clustering results make sense. To make exploring the map and clusters easier, I also included pop-up window on each zip (push pin). Click on push pin reveals some data (Demographic, unemployment, size, density) for ZIP code, and also average for mentioned data for whole cluster in which this ZIP belong. This quickly give as an idea, if ZIP is similar to its cluster or not.

5.4 Clusters

In this section five most outstanding clusters are described in detail. Other clusters are only briefly categorized.

5.4.1 Cluster 1, areas near centers of big cities

Typical map view:

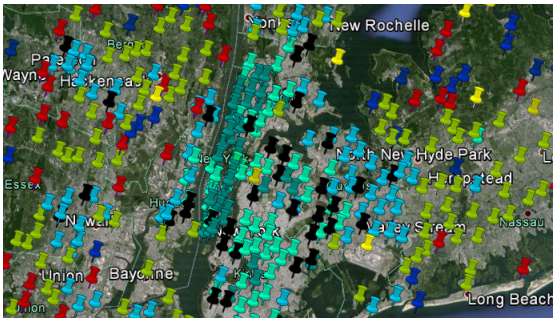

Figure 4: Diverse ZIP codes in New York

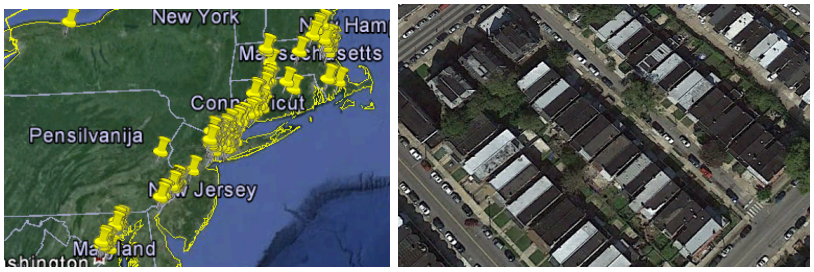

Figure 5: House settlements, typical for cluster 1

Low unemployment, high average percent of blacks (20 %), people in these ZIP codes are young, with average income. On the map we can see that the city center is always near, people live in houses or multi unit rentals. This cluster is well represented (a lot of ZIP codes in it), and very homogeneous.

5.4.2 Cluster 3, Metropolitan cities, middle class

Typical map view:

Figure 6: Chicago and New York - All ZIPs are in the center of the city

American metropolitan areas (New York, Miami, Chicago, Phoenix, etc), high density of population, mostly high rise renters, but this are still areas with lower average income than the richest metropolitan ZIP codes, average unemployment is above average. Statistics for this cluster also reveal high density of population and that small area is covered by them in square miles.

5.4.3 Cluster 10, Large rural areas on south

Major part of this ZIP code belong to the south. Typical ZIP in this clusters covers large area with low number of residents. Density per square mile is bellow 100. Unemployment is over 10 percent, average ZIP has almost 500 square miles of area.

Figure 7: Rural areas near Mexican border

5.4.4 Cluster 7, Highly populated, race diverse areas

High density of population, low percent of white residents (52 percent). This ZIPs are mostly present in big cities. Average unemployment is 9 percent, which is above average (5.6 percent). Average area covered by ZIP code in this cluster is only 3 square miles.

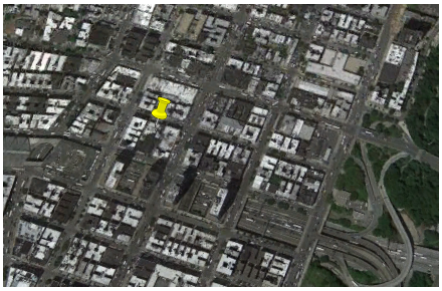

Figure 8: Over 100 000 people live in this ZIP code.

5.4.5 Cluster 8, American upper class, urban neighborhoods

Big houses with pools are the view we get when zooming on Zip codes in this cluster. Urban lifestyle, high income, this areas are found near metropolitan center.

5.4.6 Other clusters

- Cluster 0, rural areas with high unemployment
- Cluster 2, outdoors of Alaska
- Cluster 4, Suburb houses

Figure 9: Typical mansion in cluster 8

- Cluster 5, farms and fields
- Cluster 6, mixed, but more ZIPs found in countryside
- Cluster 9, small villages in the countryside
- Cluster 10, low populated outdoors
- Cluster 11, suburbs around the country
- Cluster 12, cities on west coast
- Cluster 13, countryside, 90% of white residents
- Cluster 14, urban areas, racially diverse with high percent of black and hispanic.
- Cluster 15, Suburbs
- Cluster 16, Industrial areas
- Cluster 17, Alabama towns
- Cluster 18, Suburbs and shopping malls
- Cluster 19, Rich neighborhoods, low unemployment

6 Statistical Modelling

6.1 Preparing training set

When I had good enough clustering results, I first prepare training set for machine learning. All ZIP codes were changed with corresponding cluster number, and all domains were changed with numeric value. The column with site info was removed.

I was using multiple classifiers, from two libraries - Scikit and Orange. Scikit was faster (In python 2.7), later I found out that I should use Orange 3, which perform even better in this version than scikit.

6.2 Random forest

First simple test to further validate clustering results was perform Random forest classification with data prepared as mentioned above, and the other dataset with ZIP column left intact. The results were encouraging. AUC was improved using clustering labels from 0.58 to 0.725.

Multiple combination of clusters number and scaled / non scaled data were also tested. The results were slightly better using normalization and scaling of the data before clustering.

6.3 Stacking

The best score was achieved with stacking, inspired by example on `https://github.com/log0/vertebral/blob/master/stacked_generalization.py`. For base classifiers I used random forest, extra trees and gradient boost classifier. A lot of other classifiers were also tested (Naive bayes, SVM, K Neighbours), but AUC was lower. Base classifiers were stacked with Logistic regression, with low regularization (Low parameter C means stronger regularization). With default regularization (C=1) AUC was around 0.02 lower.

6.4 Results with internal validation

In the table bellow are results with Random Forest classifier compared with stacking. Training set was separated to train (60 percent) and test (40 percent) set. Increasing number of trees in classifiers also improved AUC significantly.

Table 1: Results with Random Forest and Stacking

Classifier	Number of estimators	AUC
Random Forest	10	0.67
Random Forest	100	0.72
Stacking	10	0.74
Stacking	100	0.78