ADM deliverable 2

by Diego Garcia-Olano

March 30, 2014

1 2012 US election continued: Data cleaning and merging of other data[1]

The following is a listing of the sources used in this analysis:
1. Election data
  url: www.theguardian.com/news/datablog/2012/nov/07/us-2012-election-county-results-download
2. Measure of America 2013-14 report, using only 2010 data. Socio-Economic data
  url: www.measureofamerica.org
3. 2012 County Health Rankings National Data. Public health and some socio-economic data.
  url: www.countyhealthrankings.org/rankings/data

The election data set contained columns such as Party1, Candidate1, Party2, Candidate2, etc whose values could change per county, so I first needed to create a little script which would pull out and store in a standardized way how many votes Democrats and Republicans got per county. This analysis doesn't take into consideration third party candidates which on a national level makes little difference, but can influence particularly smaller county results.

Additionally, Alaska for voting purposes doesn't have the same concept of counties as the rest of the US, and thus the election data for it was summarized only at the state level, and hence also not used for this analysis of counties.

All the three data sets are linked by their Federal Information Processing Standards (FIPS) code.

The election results for the states in the northeast of the US (CT, MA, ME, NH, RI, and VT) are reported slightly different than other states because the FIPS codes within the state don't map one to one to with counties as they do in other states. Thus in our analysis, we need to combine and sum up the voting results for counties with the same FIPS code. The necessity for this step actually went unnoticed for the first analysis done on the data using Naive Bayes classifiers, and as the northeast is fairly solidly "blue", it in effect over counted the number of democrat counties.

The socioeconomic data from the Measure of America report was straightforward to merge using the FIPScode, as was the County Health data, but whereas the Measure data was for the most part "complete", having few missing values, the County Health data included some columns which were unfortunately too sparse and needed to be removed completely (particularly, Aids and Homicide rates). Fortunately in the County data, many of the variables for instance "smoking" contain a related variable "smokingq" which is its state quartile ranking. Thus we were able to leverage that prior knowledge and do regressions per state to imputate the missing variables: it should be noted that although there was missing data in variables such as "smoking", there was no missing data in the quartile variables.

For the few remaining variables which had no corresponding quartile, we used the 'mice' package in R to imputate them. Our cleaned and merged data set is now 3113 counties with 90 variables each.

2 Decision Trees with Julia

For the purposes of this task, I decided to try out the Julia language (http://julialang.org/),

[1]The analysis here builds upon the Blue Islands project found at http://www.diegoolano.com/electionmap/ and the first analysis paper is located at www.diegoolano.com/electionmap/analysis1.pdf

which is a relatively new language, not unsimilar to R and Matlab, but built on top of C and offerring speed improvements, easy leveraging of parallelism and among others. It is used at MIT and has an active, albeit small, development community. Because it is relatively new, it does not have the sheer number of libraries available in R or other more mature languages, and at times, there can be issues with packages, Julia's version of libraries, but most issues with that were solved via Pkg.update() and help().

I downloaded the Mac os x dmg, and installed it, and run it locally:

```
> cd /Applications/Julia-0.2.1.app/Contents/Resources/julia/bin/
> ./julia
```

Now Julia is opened and it looks similar to Ipython,
I first install the DataFrames and DecisionTree packages, read in my data, and look at my target.

```
> Pkg.add("DataFrames"); Pkg.add("DecisionTree")
> using DecisionTree, DataFrames
> md = readtable("electionmap/2012votes_health_merged_cleaned_final_version.csv", makefactors=true)
> by(md, "isdem", nrow)
        isdem   x1
[1,]     "no" 2427
[2,]    "yes"  686
> sum(md["dem"])             #62210233
> sum(md["rep"])             #58788428
```

The target variable is "isdem" and we can see that 686 counties voted democrat while 2427 didn't in the 2012 US election where the democratic party won with 62.2 million vs 58.7 million.

First we create a simple decision tree, a stump, consisting of only one split and two leaves. The split generated is the one with the most predictive power. This is done by traversing all the features and splitting the dataset into two subsets for every unique value of the feature, then returning the split with the highest information gain.

We get rid of the following from our feature matrix: X, fips, isdem, state.y, county

```
> features = matrix(md[:,[4:5,7:32,35:96]]);
> labels = vector(md[:, "isdem"]);

> stump = build_stump(labels, features);
> print_tree(stump)
Feature 11, Threshold 50.7
L-> yes : 250/336
R-> no : 2341/2777
```

Feature 11 is "white", and the tree says that if the percentage of white voters in a given county is less than 50.7, there is a 74.4 %(250/336) chance it voted Democrat. Subsequently if the amount of white voters is greater than 50.7, there is an 84.29 % (2341/2777) chance that the county voted Republican. Of the 3113 counties under study, only 10.79% (336 / 2777+336) are less than 50% white and in only 22% of counties (686/3113) did the Democrats win the vote.

One way of verifying this split is by making predictions based on the sample features using the generated decision stump and then comparing the predictions to the actual classes using the confusion_matrix function in julia.

```
> predictions = apply_tree(stump, features)
> confusion_matrix(labels, predictions)
Classes:  {"no","yes"}
Matrix:   2x2 Array{Int64,2}:
 2341   86
  436  250
Accuracy: 0.8323160938001928
Kappa:    0.40268715902332447
```

Thus our tree correctly identifies 250 democrat counties and misidentifies 86 Republican ones as being democrat. It has an accurracy of 83.3%
If we want better accuracy we can build out the full tree as such:

```
> automated_tree = build_tree(labels, features);
> print_tree(automated_tree)
```

This tree is pretty huge (14 levels deep) so I decided that I wanted to create an interactive visualization with d3.js to get a feel for the tree itself.

```
> featuresdf = md[1,[4:5,7:32,35:96]]
> dfnames = colnames(featuresdf)

function tree_to_d3(tree::Node,dfname::Array)
  print("{'name':'$(dfname[tree.featid]) < $(tree.featval)', 'children':[")
  if isa(tree.left,Leaf)
      matches = find(tree.left.values .== tree.left.majority)
    ratio = string(length(matches)) * "/" * string(length(tree.left.values))
      print("{'name':'$(tree.left.majority) - $(ratio)', 'size':''},")
  elseif isa(tree.left,Node)
      tree_to_d3(tree.left,dfname)
else
      print("")
  end
  print("")
  if isa(tree.right,Leaf)
      matches = find(tree.right.values .== tree.right.majority)
  ratio = string(length(matches)) * "/" * string(length(tree.right.values))
      print("{'name':'$(tree.right.majority) - $(ratio)', 'size':''}")
  elseif isa(tree.right,Node)
      tree_to_d3(tree.right,dfname)
else
  print("")
  end
  print("]},")
end

> tree_to_d3(automated_tree,dfnames)
```

I then took the json produced from the function and use it as input for a relatively simple visualization I made largely based the Collapsable Tree (http://bl.ocks.org/mbostock/4339083). It can be viewed here: www.diegoolano.com/electionmap/initial-decision.html and a partial view of it is below in Figure 1. Going up from a node means the label condition is true.

As can be observed the tree is quite large and unwieldy for a few reasons.
The inclusion of the "dem" and "rep" variables which pertain to the number of people who voted for Democrat and Republican in a given county is uninteresting and leads to some nonsensical paths. Additionally, the results of how well a branch does in predicting is only shown at the leaf level. Thus we decide to first rerun the tree but this time without the "dem" and "rep" variables, and then prune the tree itself.
Pruning is a method for making decision trees less prone to over-fitting while reducing their size and complexity. Splits with little predictive power are removed, producing more flexible, yet still accurate models. In Julia, the pruning parameter is a combined leaf purity threshold where by if the purity of the combined samples of two adjacent leaves is greater than the threshold, then the two leaves are merged into one leaf.

```
> features_without_dem_reps = matrix(md[:,[7:32,35:96]])
> automated_tree_v2 = build_tree(labels, features_without_dem_reps);
> featuresdfr = md[1,[7:32,35:96]]
> dfnamesv2 = colnames(featuresdfr)
```

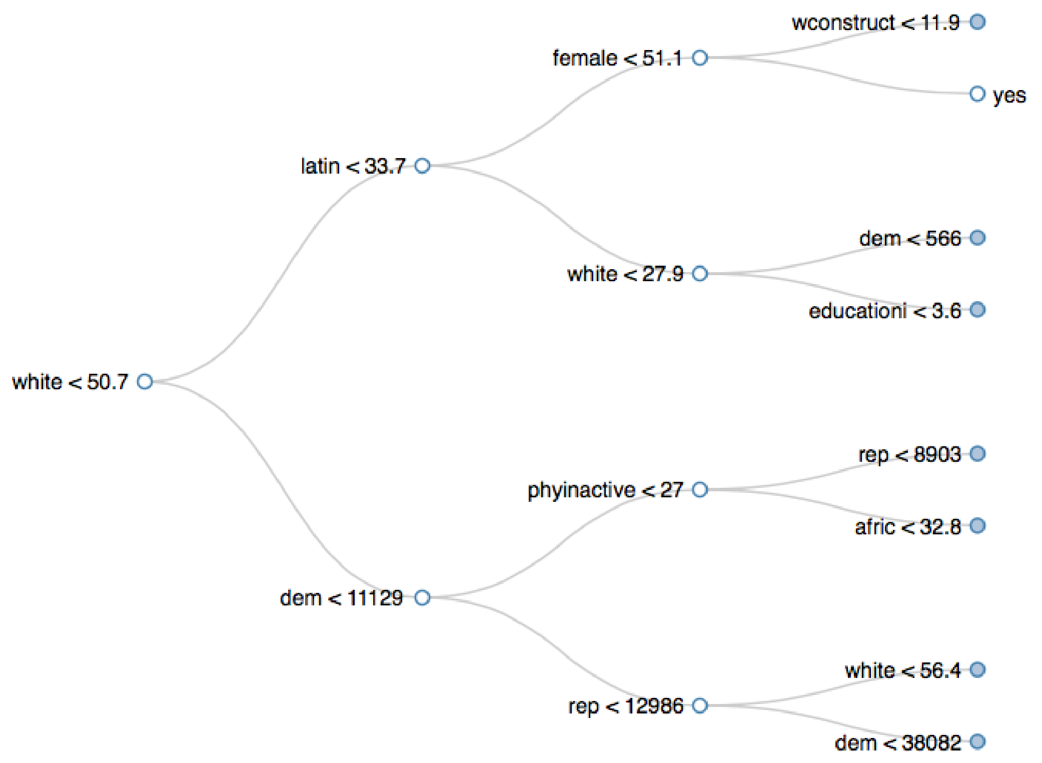

Figure 1: initial collapsable tree view of decision tree. only expand to depth 4 of 15

```
> tree_to_d3(automated_tree_v2,dfnamesv2)
> length(automated_tree_v2)    #176
```

```
Additionally, we decided to go with a visualization style which is less interactive,
but allows the user to see the entire expanded tree to get an idea of the numbers
( and to avoid implementing another function into Julia to get the counts).  The visualization
is based on the Elbow Dendogram (http://bl.ocks.org/mbostock/2429963) and can be seen here:
www.diegoolano.com/electionmap/initial-elbow-decision.html and a partial view is provided:
```

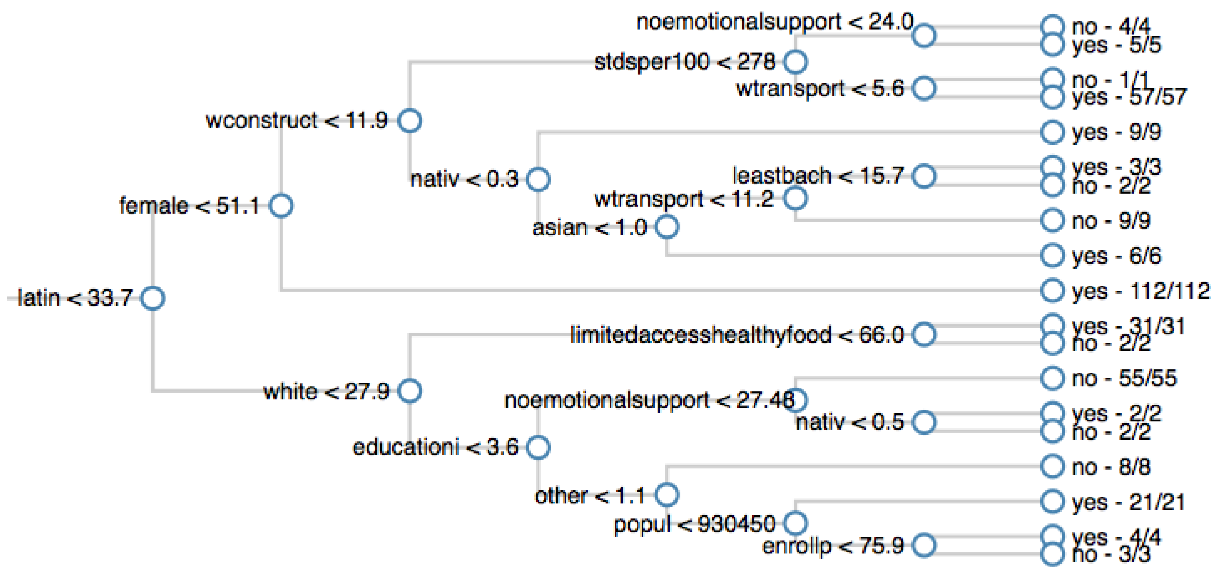

Figure 2: partial view of initial elbow dendogram of decision tree

This version is easier to use than the fully expanded prior tree , but is still too big to make sense of. We now apply pruning with a ninety percent threshold and visualize it again.

```
> prunedv2 = prune_tree(automated_tree_v2, 0.9);
> length(prunedv2)  #164
```

This looks virtually identical so we decide to go for a lower threshold of .75.

```
> prunedv3 = prune_tree(automated_tree_v2, 0.75);
> length(prunedv3)   #145
```

This one has markedly less nodes than the first two trees but still has the same depth so we decide to go for more drastic pruning just to make interpretability simple.

```
> prunedv4 = prune_tree(automated_tree_v2, 0.6);
> length(prunedv4) #98
> depth(prunedv4) #14
```

Still not drastic enough! So we continue even more.

```
> prunedv5 = prune_tree(automated_tree_v2, 0.53);
> length(prunedv5)              # 55
> depth(prunedv4) #still 14!
```

This is as low as we can set the threshold because otherwise we get a 1 length, 0 depth tree. Its a good enough general representation and provide insights into what are the most defining variables (ie, those higher up in the tree). It can be viewed at www.diegoolano.com/electionmap/final-decision.html and below in Figure 3.

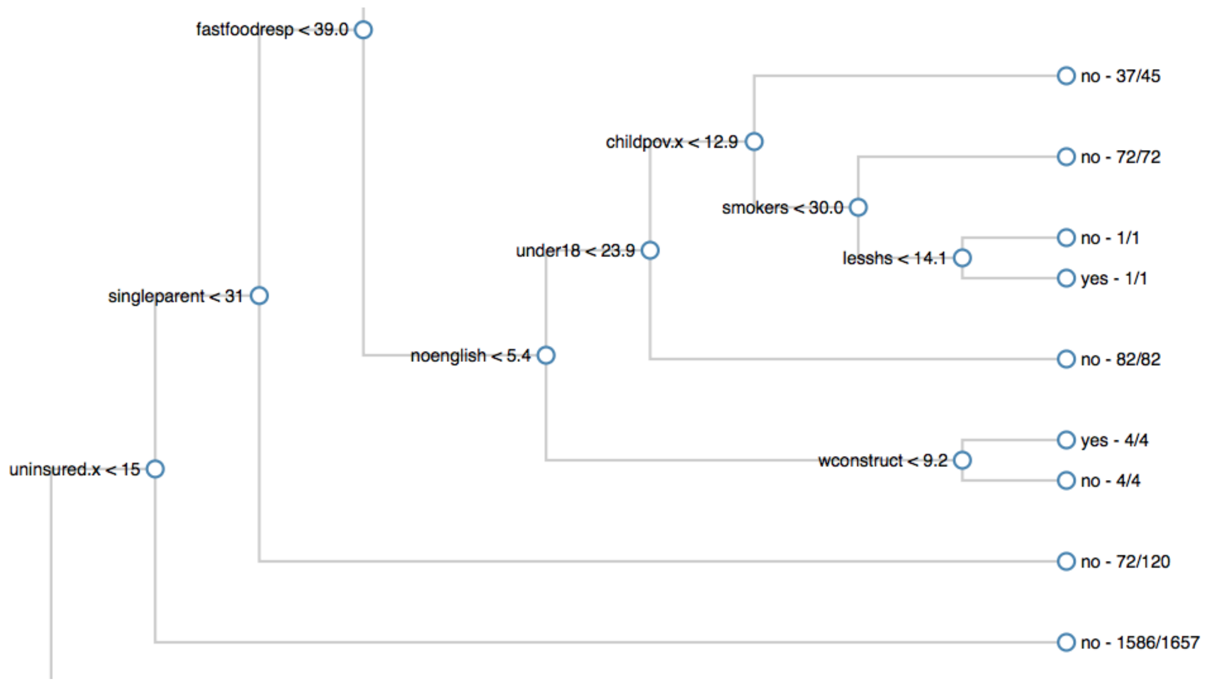

Figure 3: partial view of highly pruned decision tree

To get an idea of some of the paths from this tree, we look at the leaves with higher values. For instance counties:

```
  * less than 50.7% white and
    less than 33.7% latino
    voted Democrat 192 out of 208 times, 92.3% of the time.

  * more than 50.7% white
    with less than 8.3% of the population having graduate degress
    and whose uninsured population is less than 15%
    voted Republican 1568 out of 1657 times, 94.6% of the time!
```

* Additionally, it should be noted that the ratio variables where run as categorical and thus on the visual depections there operators should be considered as equality ones.
For future use, it may be better to either take these ratio variable and split them into two seperate variables representing the two integer values, or to treat them as numerical.

3 Random Forests and more

In addition, to creating a single tree, we can bag together multiple trees to create an ensemble tree learner known as a random forest. The basic idea behind a random forest is creating multiple different fully expaned trees on some test data, but introducing a component of randomness (by choosing only a random subset of features at each split) and then using the majority vote of their individual predictions as the forestâĂŹs overall prediction.

This approach give us the advantage of being less prone to over-fitting, hence generalizing better than individual decision tree, but at the expense of losing the transparency and interpretability offered by decision trees. We will build a forest which searches through 2 randomly selected features per split, consisting of an array of 10 trees:

```
> forest = build_forest(labels, features_without_dem_reps, 2, 10);
```

We can use the model to make predictions using apply_forest(), and again construct a confusion matrix of our predictions vs reality.

```
> predictions = apply_forest(forest, features_without_dem_reps);
> confusion_matrix(labels, predictions)
Classes:  {"no","yes"}
Matrix:   2x2 Array{Int64,2}:
 2416   11
   35  651
Accuracy: 0.9852232573080629
```

The accuracy is unrealistically optimistic since it was done on the same data, so we perform 3-fold cross validation using the nfoldCV_forest() routine to get a more realistic view.

```
> nfoldCV_forest(labels, features_without_dem_reps, 2, 10, 3);
Fold 1
Classes:  {"no","yes"}
Matrix:   2x2 Array{Int64,2}:
 776   40
  95  126
Accuracy: 0.8698167791706847
Kappa:    0.5731169975636308
Fold 2
Classes:  {"no","yes"}
Matrix:   2x2 Array{Int64,2}:
 767   37
 121  112
Accuracy: 0.8476374156219865
Kappa:    0.49848178757269657
Fold 3
Classes:  {"no","yes"}
Matrix:   2x2 Array{Int64,2}:
 776   29
 109  123
Accuracy: 0.866923818707811
Kappa:    0.5632751464843749
Mean Accuracy: 0.861459337833494
```