Lincoln Swaine-Moore
Astrostatistics Final Project
May 9, 2016

FEC Individual Campaign Donations Data

Introduction

On its website, the Federal Election Commission (FEC) releases data on, among other things, individual contributors to 2016 Presidential Campaigns (http://www.fec.gov/disclosurep/PDownload.do). This data spans all 50 states, and other territories (foreign, military, etc.).

This data is available as a CSV-formatted file, with one row per donation action (more on this later). It was originally 2578106 rows long. Each row has values for each of eighteen columns, which are as follows:

cmte_id   cand_id   cand_nm   contbr_nm   contbr_city
contbr_st   contbr_zip   contbr_employer   contbr_occupation
contb_receipt_amt   contb_receipt_dt   receipt_desc   memo_cd
memo_text   form_tp   file_num   tran_id election_tp

For brevity's sake, I will only describe the columns that were relevant to my analysis.

cand_id: A letter/number identification (e.g. "P60006723") that is unique to the candidate, which corresponds to the candidate who received the donation.
cand_nm: A string containing the candidate's name (e.g. "Rubio, Marco"), which corresponds to the candidate who received the donation.
contbr_nm: The name of the donor (e.g. "BLUM, MAUREEN").
contbr_city: The city of the donation (e.g. "ANCHORAGE").
contbr_st: The state of the donation (e.g. "AK").
contbr_zip: The zip code of the donation, either in zip-5 or zip-9 format (e.g. "99760", "997095327").
contbr_employer: The employer of the donor (e.g. "STRATEGIC COALITIONS & INITIATIVES LL").
contbr_occupation: The occupation of the contributor (e.g. "TEACHER").
contb_receipt_amt: The amount of the donation (e.g. 388.25).
contb_receipt_dt: The date fo the donation (e.g. "15-Mar-16").
receipt_desc: A description of the receipt of the donation (e.g. "REATTRIBUTION FROM SPOUSE").

I also used a second dataset for an auxiliary purpose. The dataset (found at https://boutell.com/zipcodes/) has 43204 rows, and maps zip codes to state, city, and geographic coordinates. It has the following columns:

zip	city	state	latitude	longitude	timezone	dst

These are all very self-explanatory, but:

zip: 5 digit zip code (e.g. 13654).
city: city corresponding to that zip code (e.g. “Limerick”)
state: state corresponding to that zip code (e.g. “NY”)
latitude: latitude coordinate corresponding to that zip code as a float (e.g. 44.921678).
longitude: longitude coordinate corresponding to that zip code as a float (e.g. -75.13664).
timezone: timezone corresponding to that zip code, as a difference from GMT (e.g. -5).
dst: I believe this describes daylight savings time usage corresponding to a zip code, but I didn’t use it, so I didn’t spent much effort investigating (e.g. 0,1)

Preparing the Data

I read the data in via Pandas, and had to do some cleaning. First I needed to get rid of some rows with zip codes that for whatever reason, were not actually zip code (just letters). I also dropped rows that did not have an empty string for the receipt_desc. I did this because most of the rows with receipt descriptions appeared to correspond to reallocations money between different people’s names (e.g. a person would donate on behalf of both them and their spouse, and then some of that money would be reallocated to their spouse). This was leading to double counting of money in contb_receipt_amt, and some negative values of contb_receipt_amt, so I dropped it.

Next I pulled out from the FEC data the first five digits from the zip codes (some of which were already only five digits long), and used these merge the FEC dataset to the zip code dataset. I also merged on the criteria that the states matched between the two datasets, but I didn’t require that the cities match, because I was concerned that doing so would require cleaning the dataset’s respective cities columns to make them fit, and that seemed like overkill. Out of the 2533442 remaining rows in the FEC dataset, 2496660 found a match in the zip codes, which is remarkably high. The ones that didn’t just got NaN values for latitude and longitude.

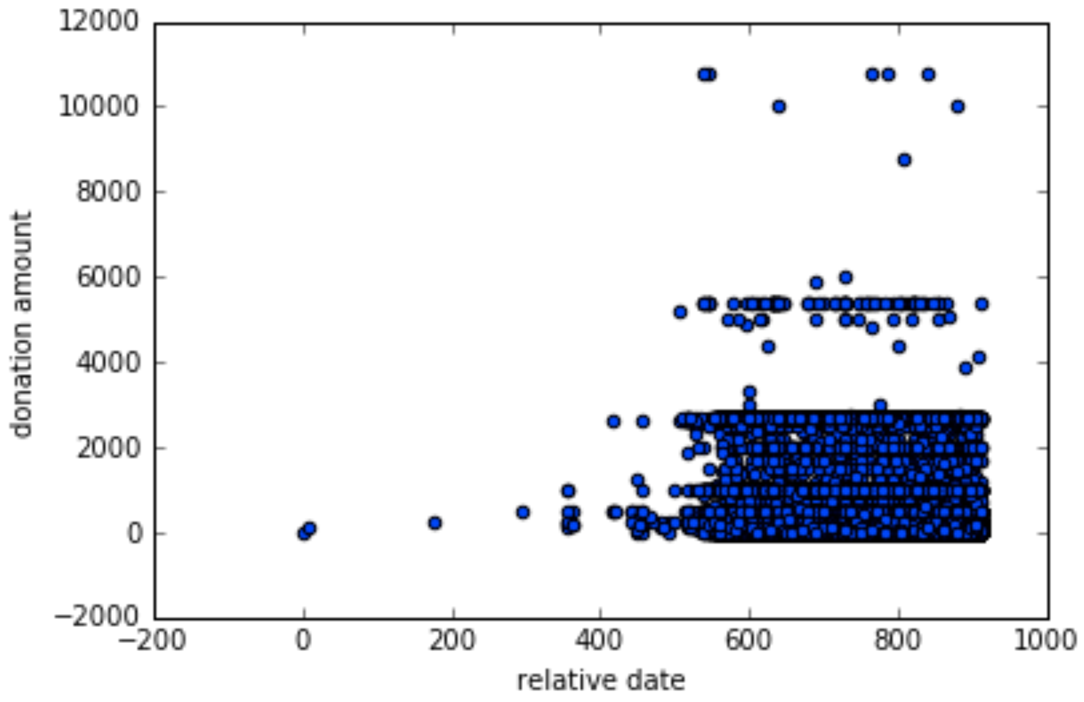
Next I parsed out the date information to make it more sensical to a computer. I extracted the month, day of month, and year, and converted that to an absolute date (days since the computer’s year 1), and a relative date (days since the earliest date in the dataset).

I cleaned up in a few other ways as well: replaced candidate names with last names for simplicity, and coerced the types of some variables to be numeric when for some reason they weren’t. I also dropped negative donations, also for simplicity, and because I couldn’t figure out why negative donations were remaining.

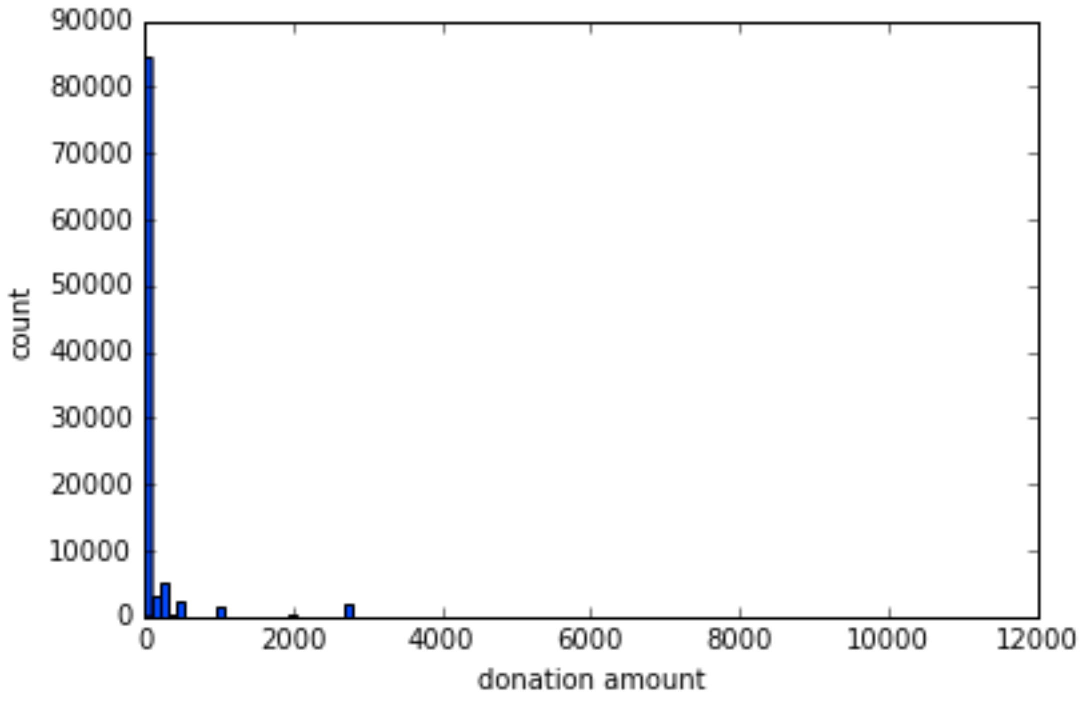
Finally, I created some subsets of my remaining dataset that I would use later. I created a dictionary that maps candidates’ names to a DataFrame with only their donations. I also created another dictionary that did that for only the candidates who had more than 5000 donations (5000 chosen because it seemed to eliminate some of the irrelevant candidates while keeping big players like Donald Trump). I created a DataFrame that only has the candidates who got more than 5000 donations as well. I also created a DataFrame that contained 5000 randomly selected donations from each candidate who had that many, and another DataFrame that contained 100000 randomly sampled donations from each of Hillary Clinton and Bernie Sanders. These last two were to be used for modeling purposes.

Exploratory Analysis

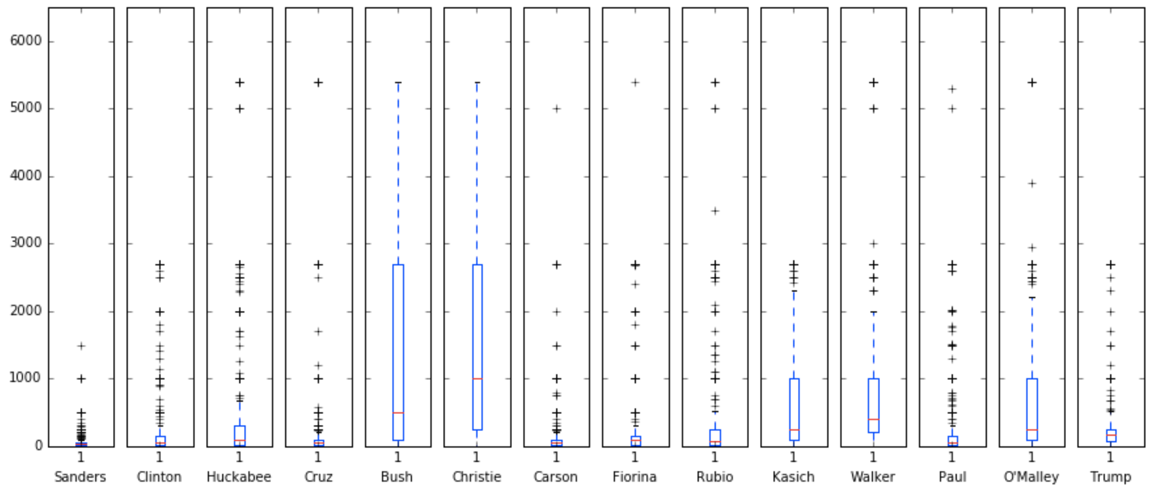
The first thing I did was explore the basic data a bit. To make plotting feasible, I randomly sampled 100000 values from the dataset, and made the following figures.



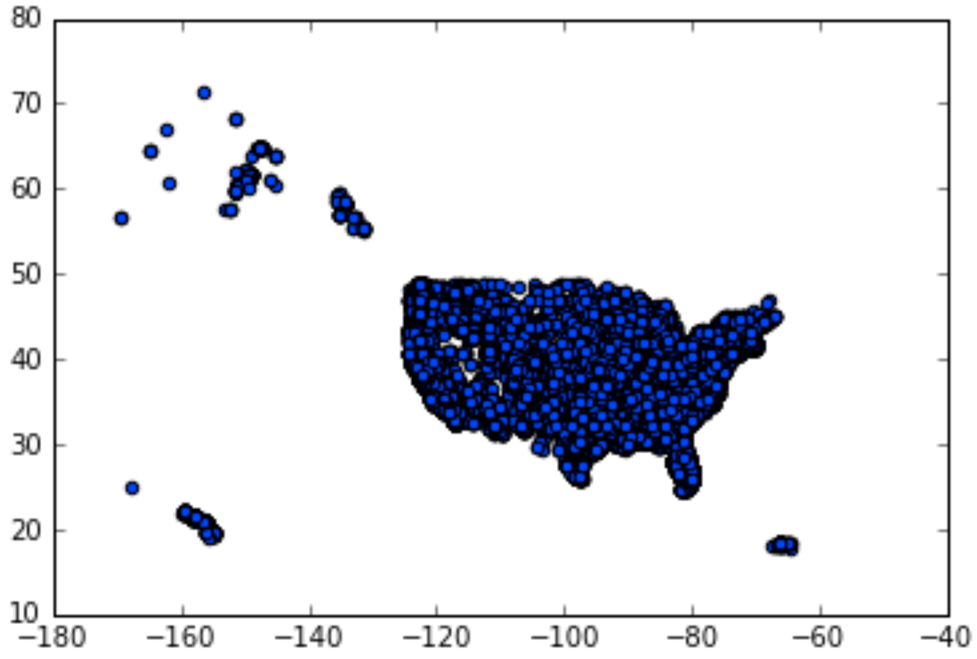
Above we see the relationship between the date and amount donated. Interestingly, though not particularly surprisingly, we see that there appear to be bands at which people donate. These probably have to do with campaign finance regulations. One such band is around 3000, another around 6000, and another around 10000. This plot shows that it may be hard to model the donation amount given the time, unfortunately.



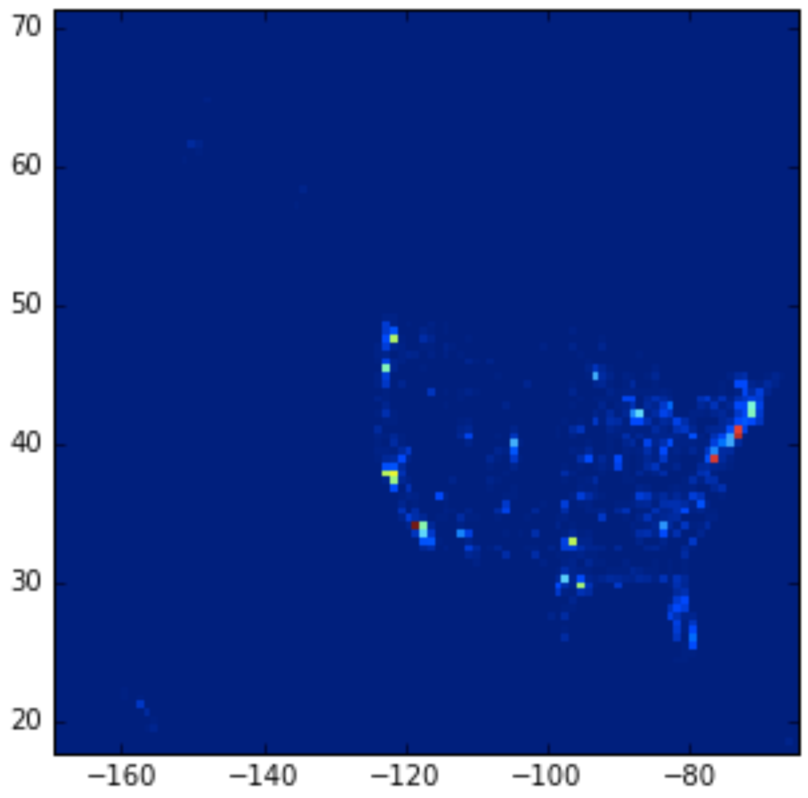
Above we see that the vast majority of donations are small and close to 0, with small spikes at higher values.



Above we see some notion of the distribution of different candidates' donations in terms of amount. Unsurprisingly, establishment candidates like Bush and Christie have high means and larger donations while candidates like Cruz and Carson have have much lower donation amounts. True to his word, Bernie Sanders appears to have the lowest donation amounts, though perhaps not by as large a margin as he would like to suggest.



Above is a simple longitude/latitude scatter of donations. We can see a pretty picture of the continental US, and even Hawaii/Puerto Rico/Alaska. This to me is a good demonstration that the zip code to lat/long is working (though the places where there was no match of course had to be dropped).



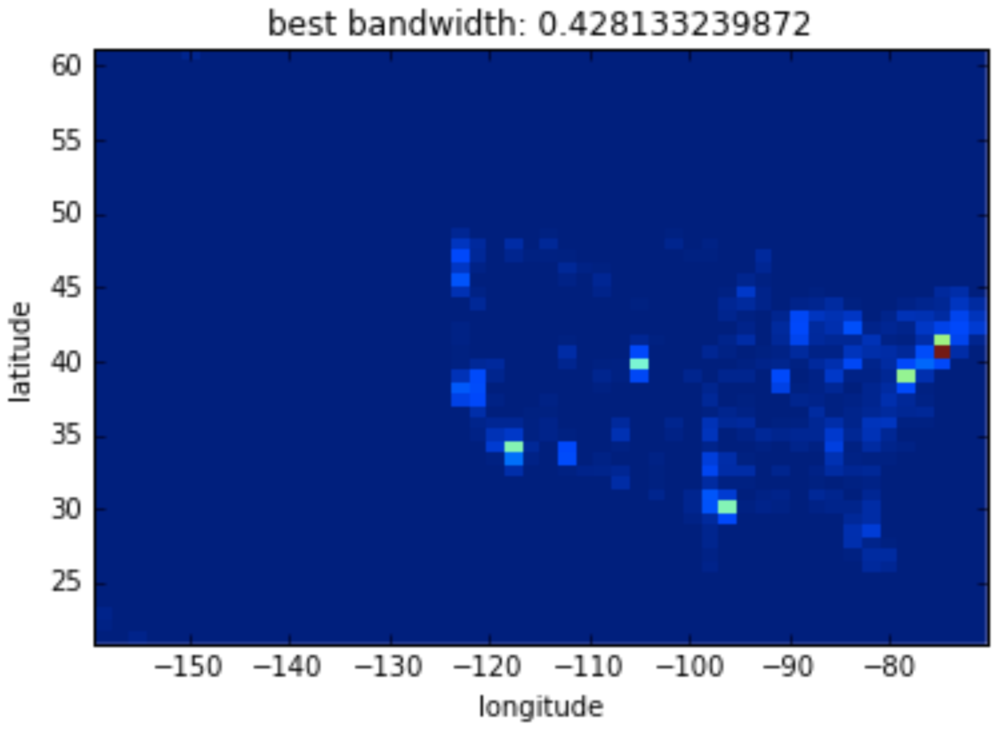
Above is a histogram which shows a little better the distribution of the donations in georgraphic space. New York is highlighted, as is what looks to be Los Angeles. But this brings me to clustering!

Clustering

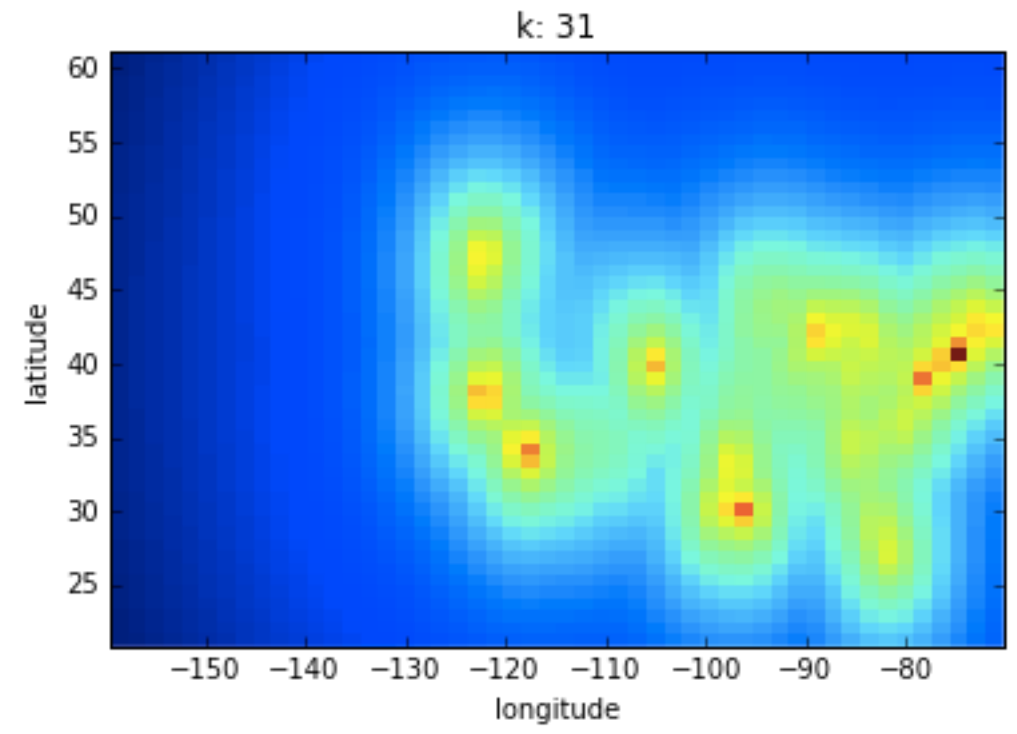
I remained interested in the geographic clustering of donations, so I decided to do clustering analysis using the latitude/longitude data.

As in Problem Set 4, I first tried KDE and KNN to get a better look at the densities, and then tried a couple of other clustering approaches.

Here is the density plot I found with KDE:



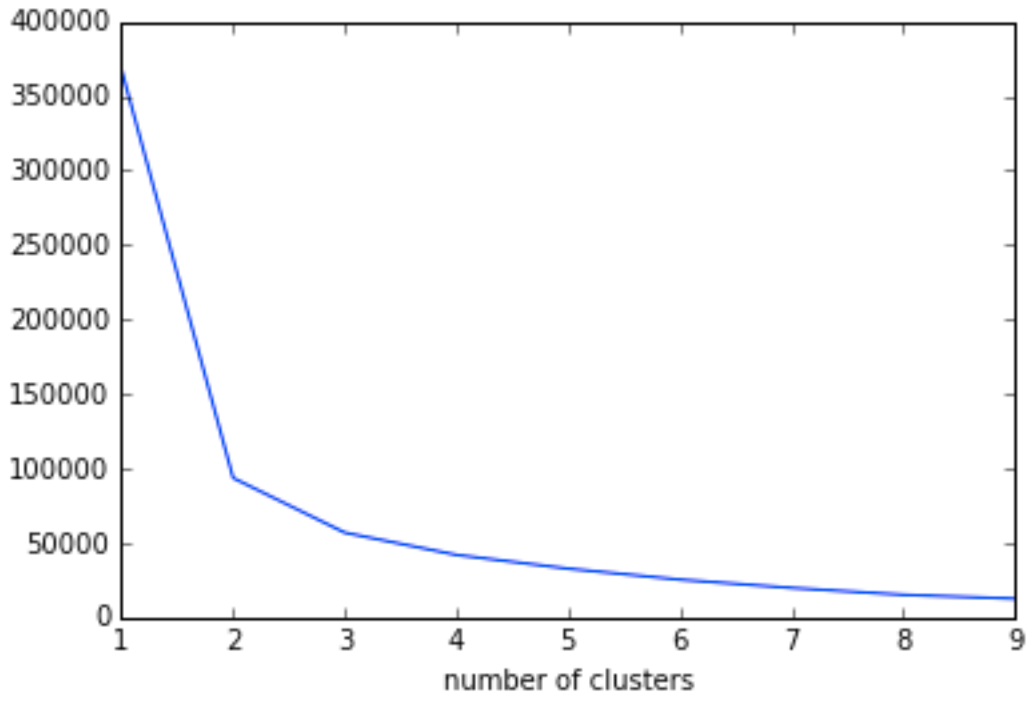
And here is the density plot I found with KNN:



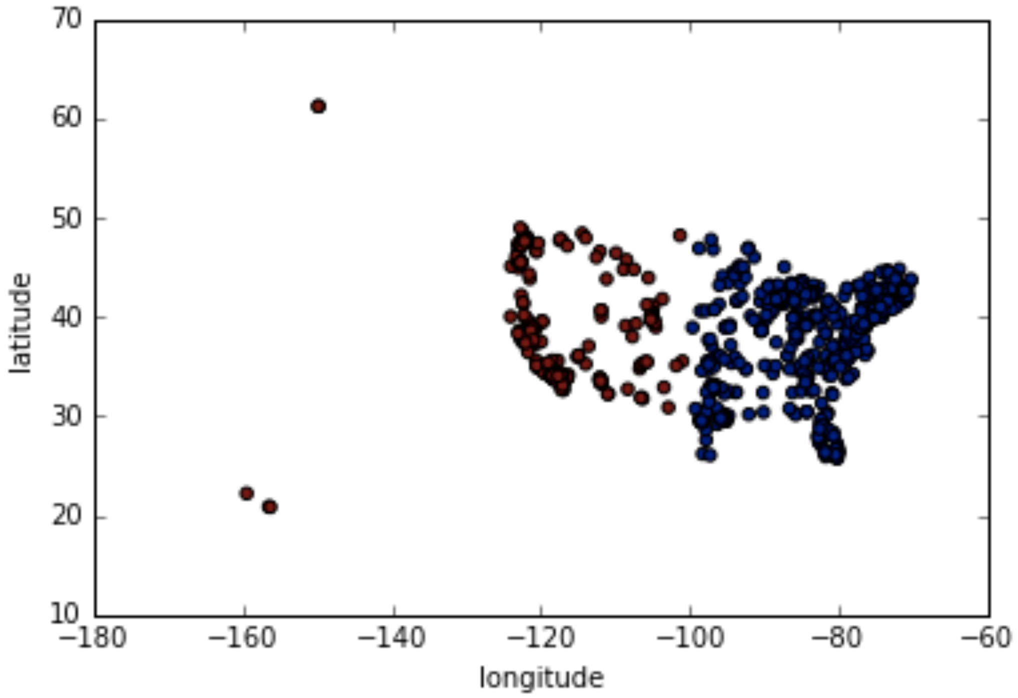
My explanation for how I chose bandwidth and k is pretty much the same as in problem set 5: I chose the value for the bandwidth using cross-validation (based on these links: http://scikit-learn.org/stable/auto_examples/neighbors/plot_digits_kde_sampling.html , https://jakevdp.github.io/blog/2013/12/01/kernel-density-estimation/). GridSearchCV allows you to specify the range for the parameters, and the number of folds--I used 20, like Jake Vanderplas!)
For KNN, I chose the value of K using a rule of thumb that sets k=sqrt(n).

Between these two methods, I think that the plot for KNN is much more visually pleasing—for some reason, I don't believe that the GridSearchCV found the bandwidth for KDE that is optimal—and k = 31 seems to work well. Again we can see major clustering around New York and LA, but also this time we can see Texas and the middle of the country also using their own clusters.

Next I tried K Means. Again, my method was the same as in Problem Set 5: To choose k, I created tried values between 1 and 10 (because 10 clusters would clearly be too many), and got the value of their "inertia"--"Sum of distances of samples to their closest cluster center" (http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)--from the model object. Then, I plotted these inertias on the y axis with number of clusters on the x axis and looked for the most dramatic change in slope (the elbow--see part (e)). The elbow appeared to be at 2, so that's what I chose. The value for the elbow was even the same as in Problem Set 5. A plot of the inertias is below:

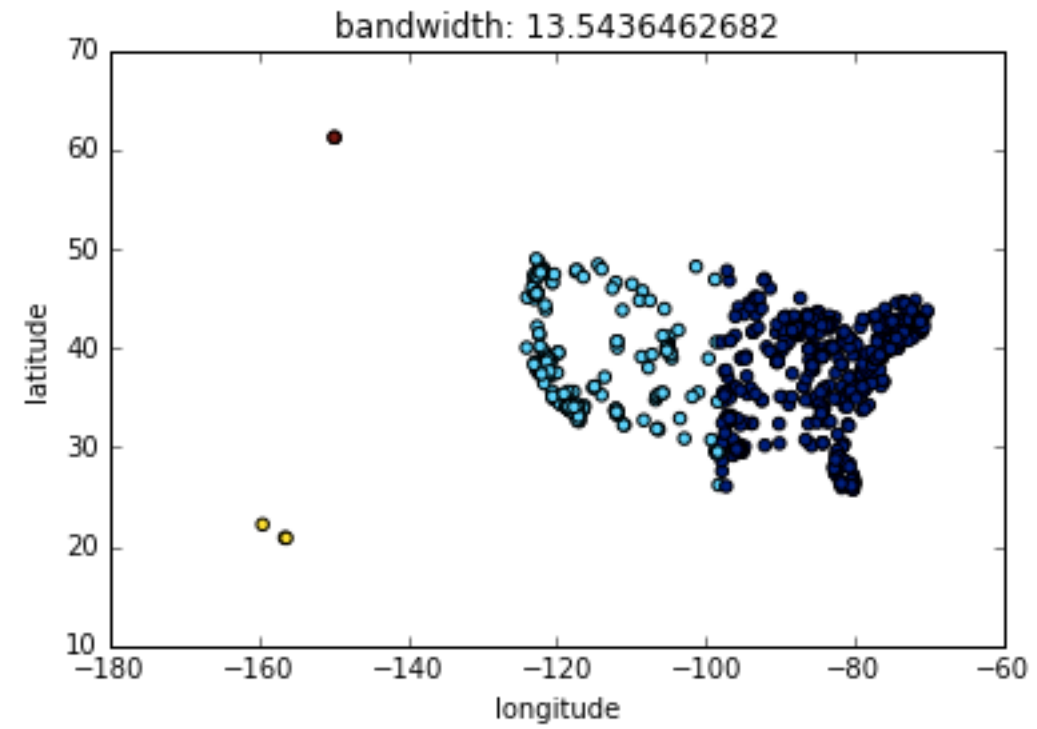


Choosing k = 2:



This splits the country, naturally, into east and west. This, to me, is not very satisfying.

Finally, I tried Mean Shift. Again, I picked my bandwidth similarly to Problem Set 5: I picked my bandwidth using the built-in function estimate_bandwidth from sklearn.cluster. This gave a value of 13.5436462682. Here is the plot:



This is a much more satisfying plot, because we can see the country split into more geographically sensible partitions: instead of just east and west, we now appear to have two more clusters for Alaska and Hawaii.

These additional clusters (which make more clear the difference between mainland US and its other parts) make MeanShift the clear winner for me between MeanShift and K Means.

Predicting 14 Candidates via Classification

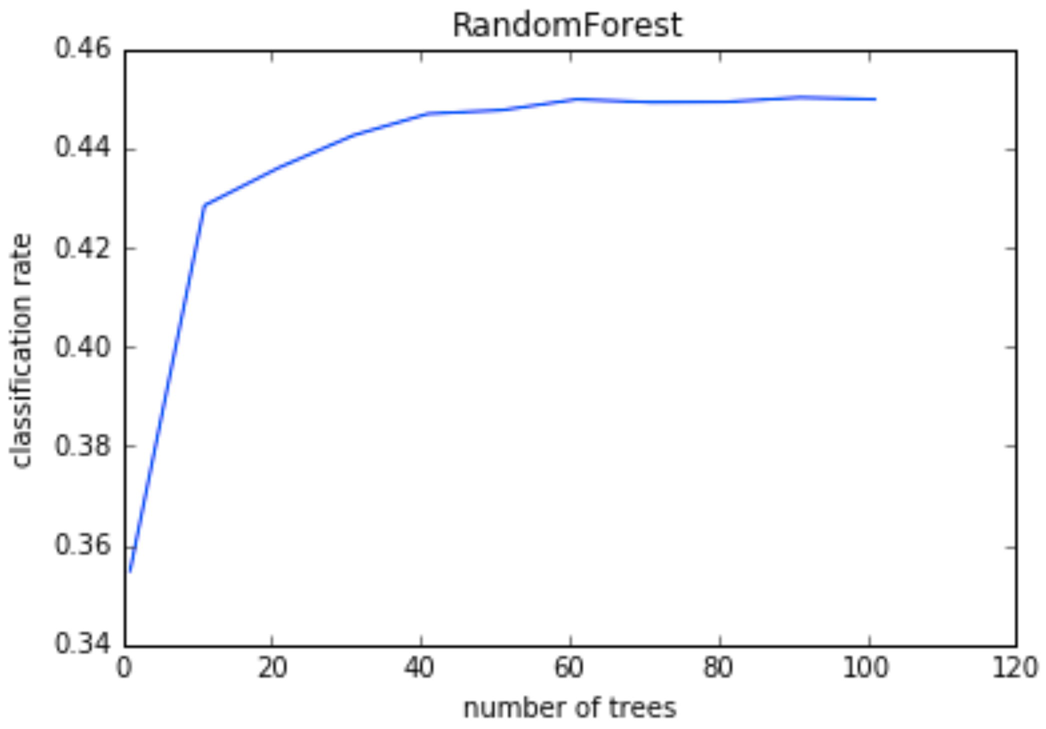
Next, I made it my goal to try to predict the candidate using numerical information from the dataset. The idea here is that candidates differ significantly enough in terms of who donates to them, when, and how much, that we may be able to pick out who someone was donating to, given that information about them and their donation.

In this classification model building, I attempted to predict as class the names of the candidates, using as predictors: longitude, latitude, contb_receipt_amt, and relative_date_int. I had originally been interested in also using the occupational and employer data, but these are of course non-numeric values, and I had trouble getting the sklearn models to accept these as inputs. So I stuck to the numerical data.

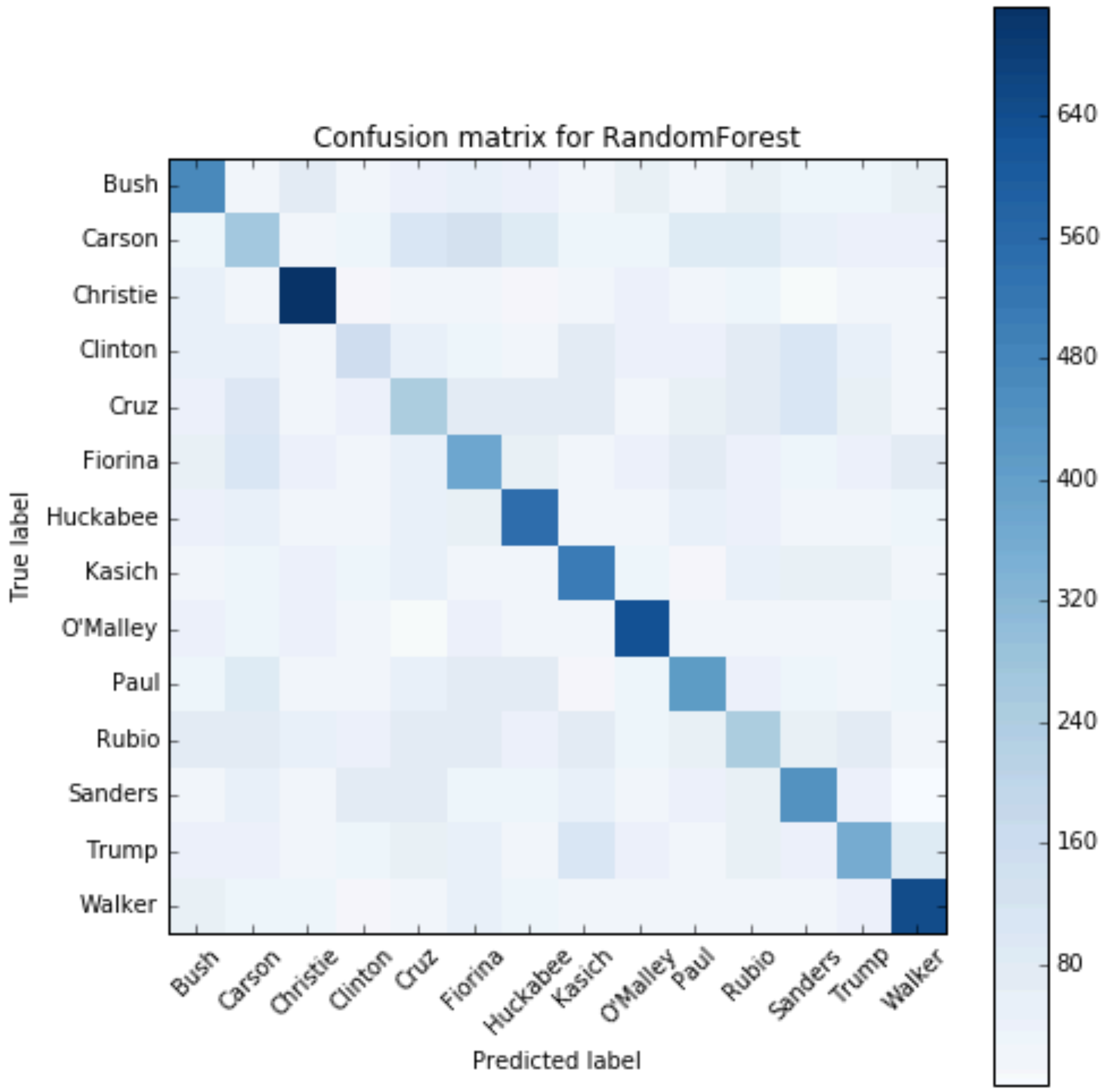
I split the data into 80% training set, and 20% test set. All classification rates reported below are calculated on the test set.

I would characterize my success as moderate. I tried a number of different techniques, and had difficulty getting my classification rates above 50%. Notably, when I used the full dataset of big donors (that is, before I started using a pared down dataset that had only 5000 donations for each candidate), I was able to break 50%--in fact getting up to about 55%. But I suspected these results, because Bernie Sanders had by far the most donations in the input data set, and I suspected this was skewing the model. When I made the number of donations for each candidate even, the classification dropped, but the confusion matrix looked much improved.

The first model I used was Random Forests, and these were the most successful. To tune the parameter n_estimators, I fit a number of models using values of n_estimators between 1 and 101 (steps of 10), and plotted the classification rate:

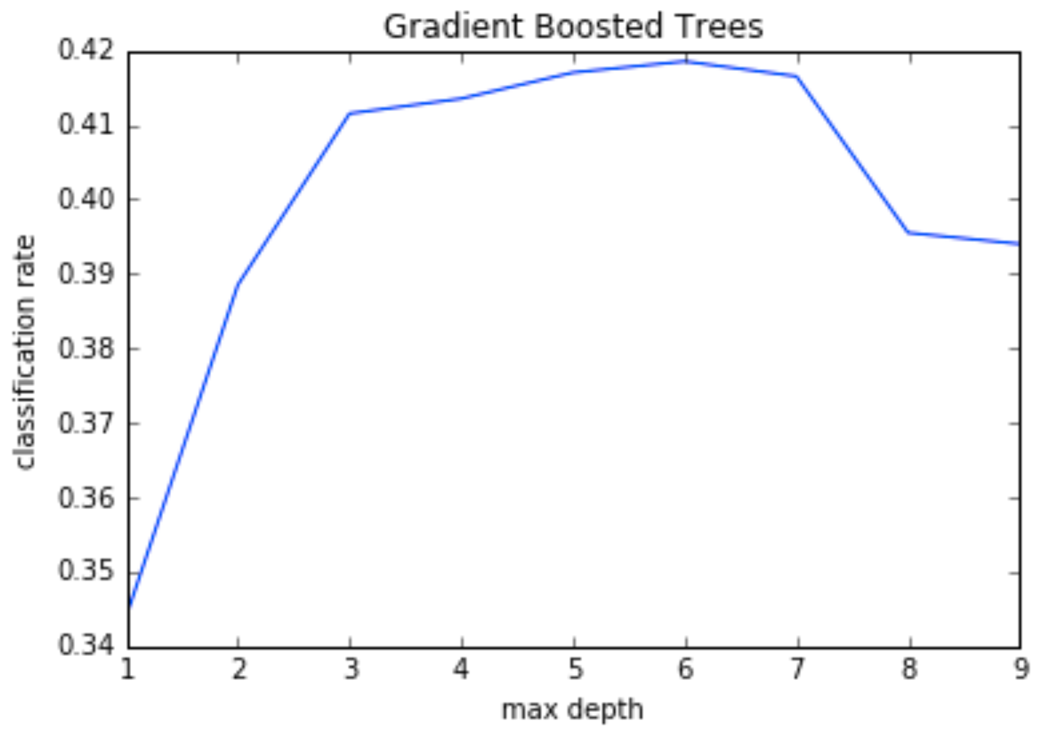


From this, it seemed to make sense to choose 100. Doing so got me a classification rate of 0.449, and produced the following confusion matrix:

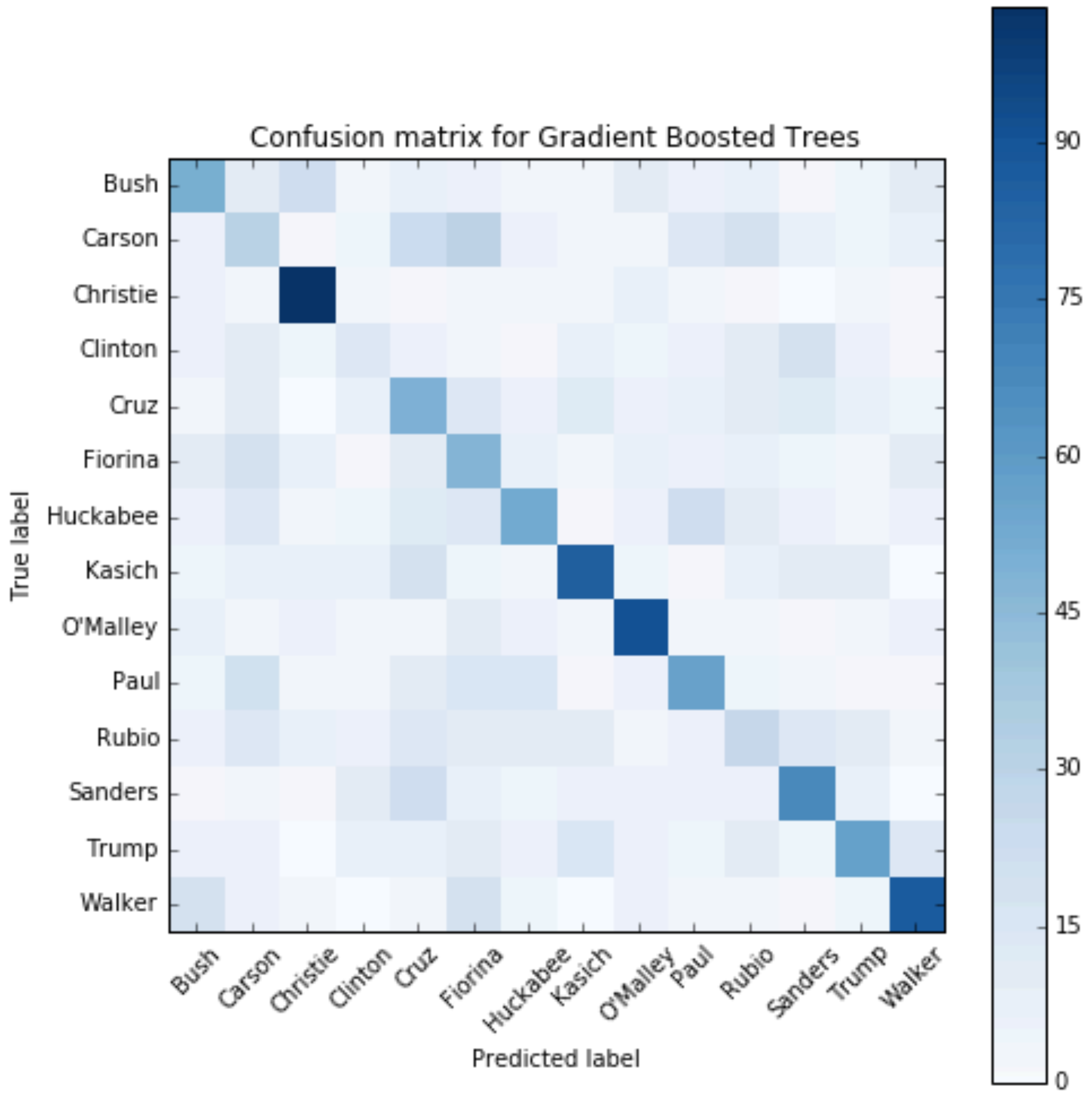


From this, we can see that at the very least, for just about every candidate (with the possible exception of Clinton, who's donations seemed to get classified as Sander's soemtimes), the model guessed that candidate more than any other.

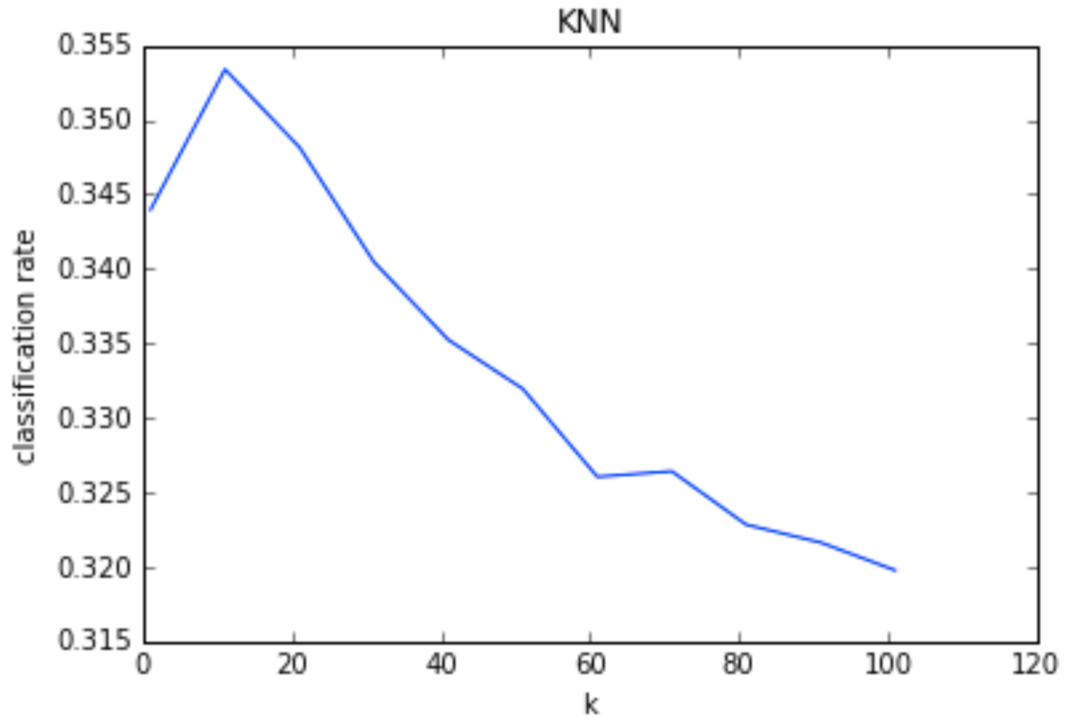
Next I tried Gradient Boosted Trees. To get this to run a bit, I had to downsample the size of my data a bit. I tuned the parameter max_depth, because the sklearn website says, "Tune this parameter for best performance" (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html). To do so, I let it vary between 1 and 10, and got the following plot:



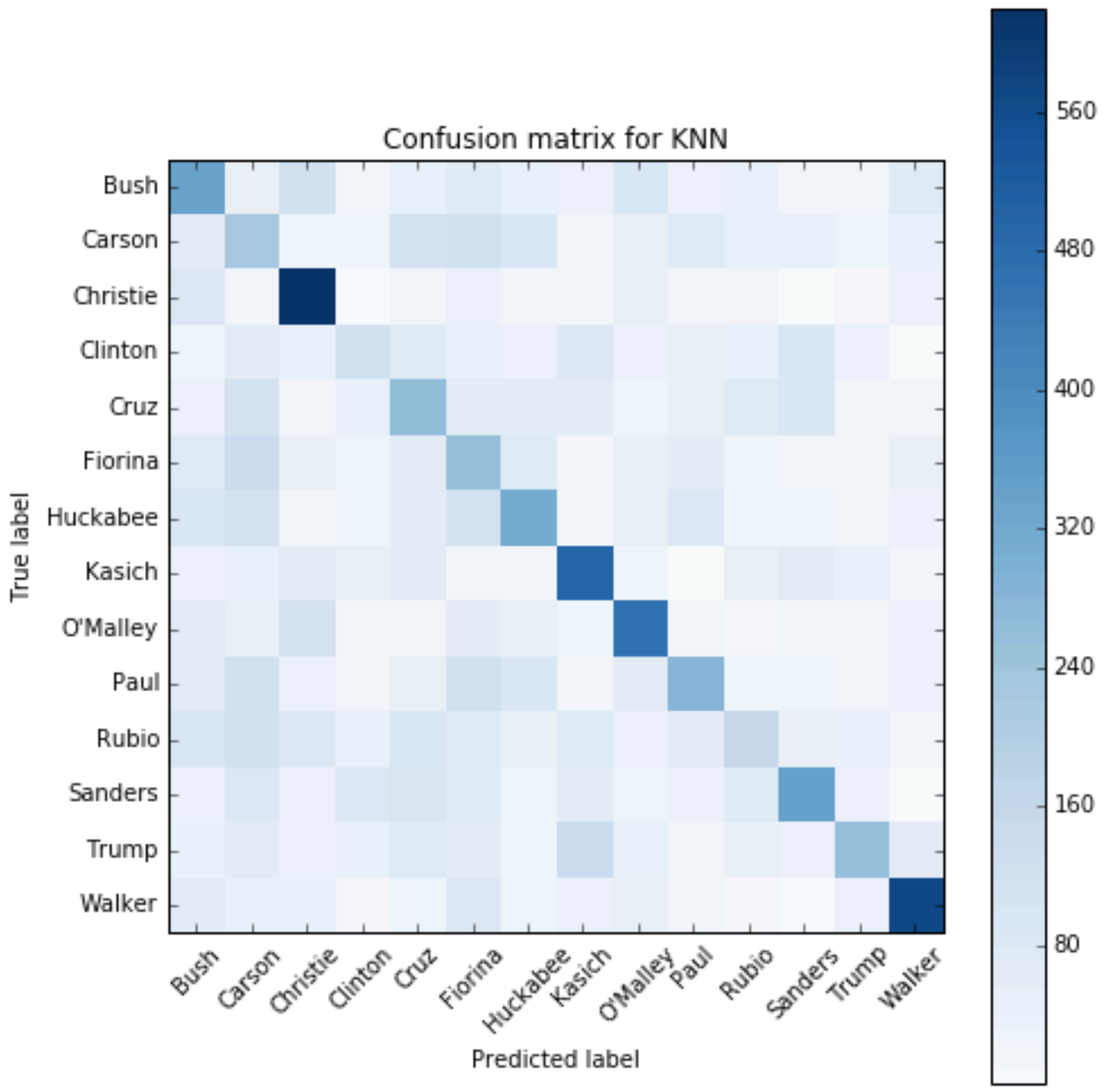
max_depth = 6 appears to give the best test classification rate, so I choe this, and got a classification rate of .4095, and a confusion matrix looking like:



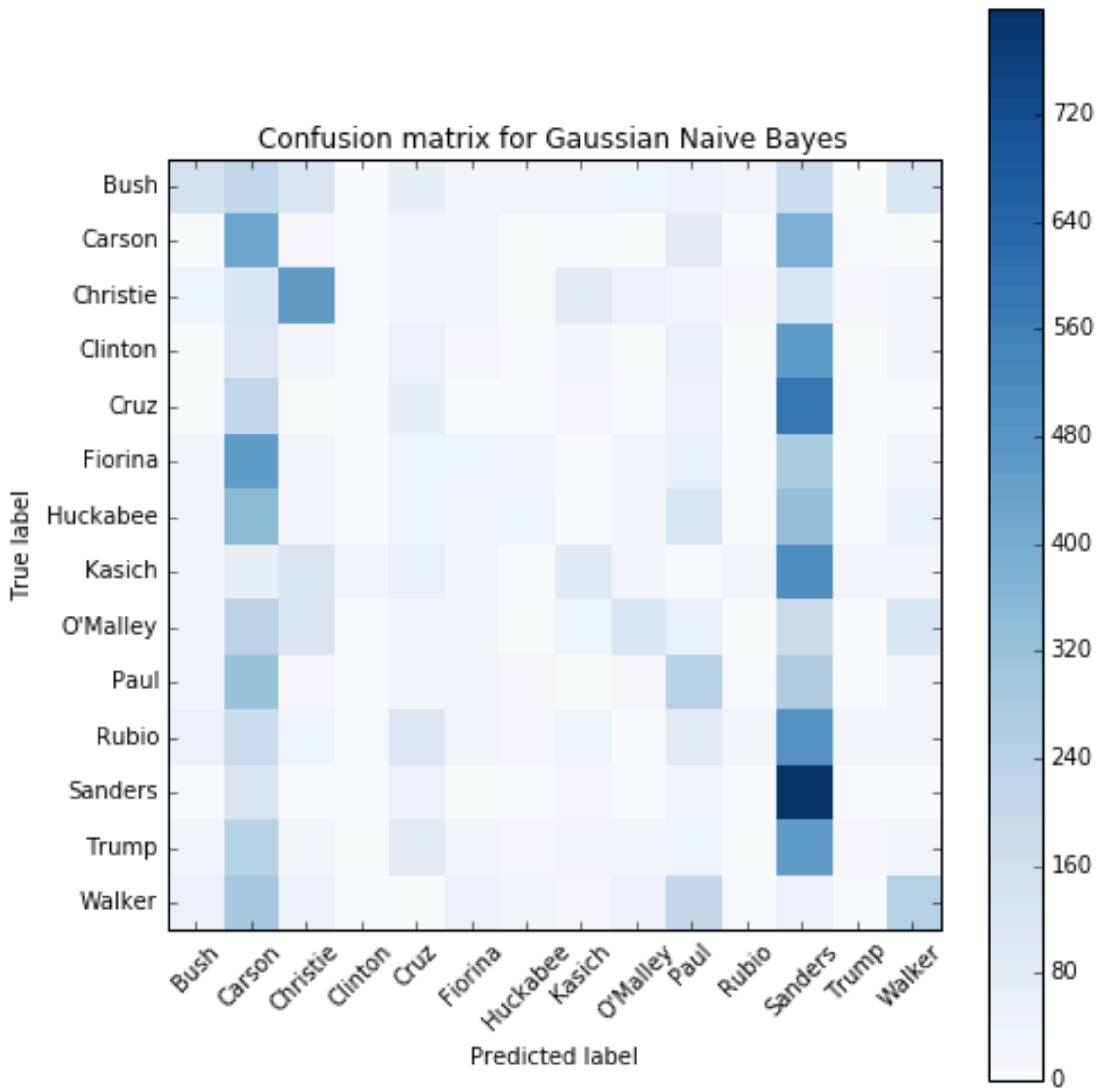
I also tried KNN classification, and did the sort of analysis to tune k, by varying between 1 and 100:



k = 10 appeared to work, best so I used it, and got a classification rate of 0.351, and confusion matrix:

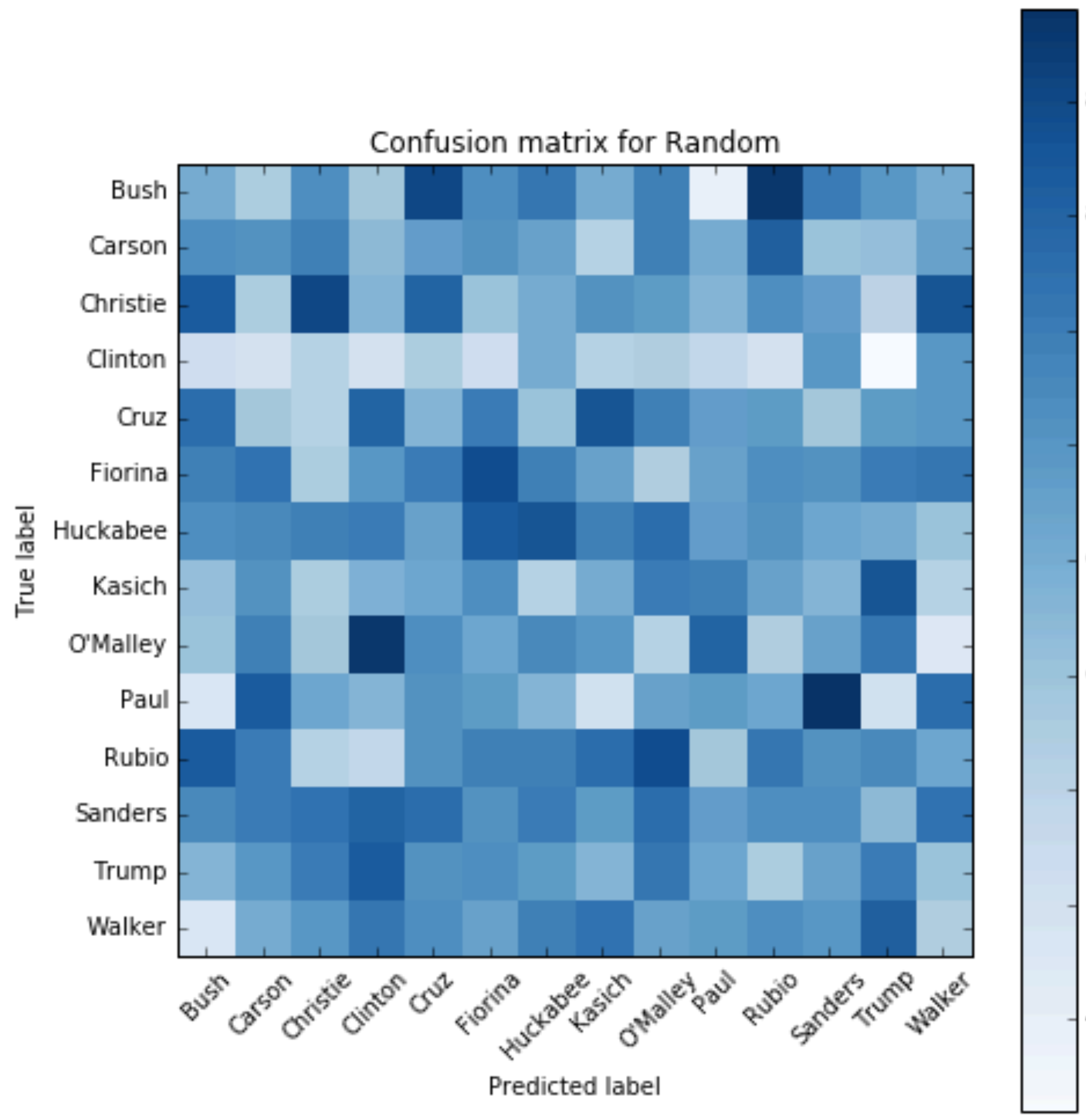


Next I tried Gaussian Naïve Bayes, just for fun, and got a classification rate of 0.206, with a much stranger looking confusion matrix:



This matrix is certainly diagonal, and seems very intent on classifying things as Bernie Sanders donations. In fact, despite there being 14 candidates to choose from, Sanders gets labeled on 38% of them, and Carson on 25%. This weirdness is probably due to the fact that this dataset majorly violated the Gaussian naïve Bayes estimator's assumptions: "The Gaussian naive Bayes estimator of eq. 9.21 essentially assumes that the multivariate distribution p(x|yk) can be modeled using an axis-aligned multivariate Gaussian distribution." (from the textbook, p. 372). It is unsurprising that the model performs poorly.

Overally, the best classification rate found here was for Random Forests, and it was a little under 45%. This sounds remarkably poor, until you consider that there are 14 candidates. While 45% would be worse than random if we were classifying between 2 candidates, for 14, if we were to classify randomly (with replacement), we would get a classification rate of between 7-8% (I tried this), and a confusion matrix like this:



Clearly, at least some of the models we produced are much better than this.

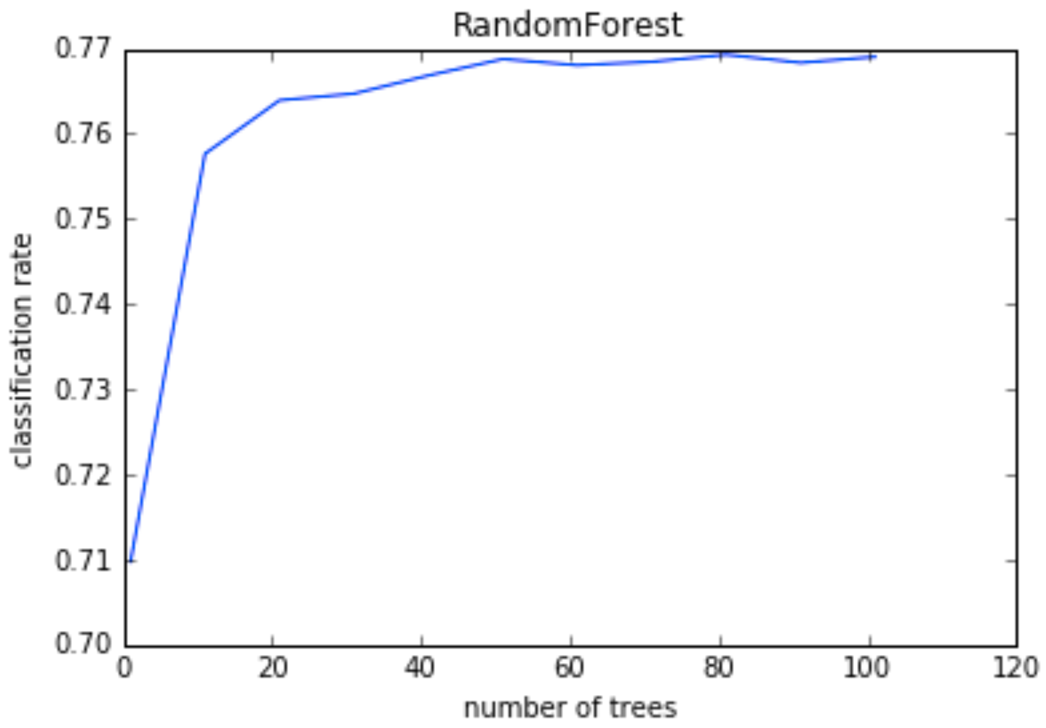
Unfortunately, multi-class classification problems do not lend themselves well to ROC curves by virtue of not having well defined concepts of positives and negatives in the context of "true positive" and "false positive.' But we can do them when we have a binary classification, which brings me to…

Predicting 2 Candidates via Classification

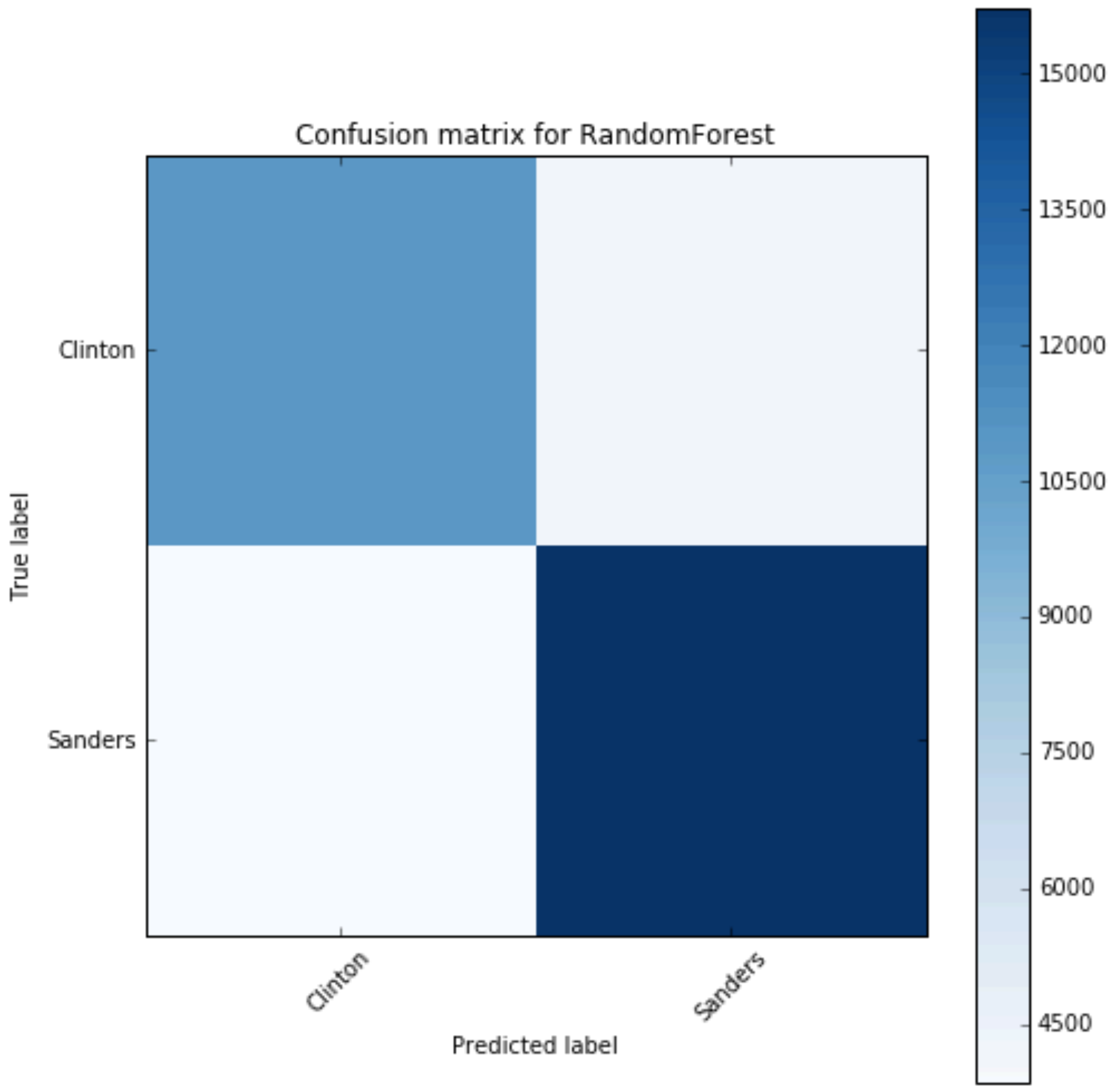
I was somewhat unsatisfied by my results classifying 14 candidates, so I decided to see how good I could get a model to do at classifying between two candidates. I chose Hillary Clinton and Bernie Sanders because Sanders consistently claims to receive different kind of donations than Clinton, and I wanted to test his proposition. Choosing to classify between one Democrat and one Republican would probably yield better results in terms of classification rate, but I thought this would be a particularly interesting question.

I fit the models in much the same way, except I didn't bother with the Naïve Bayes this time around.

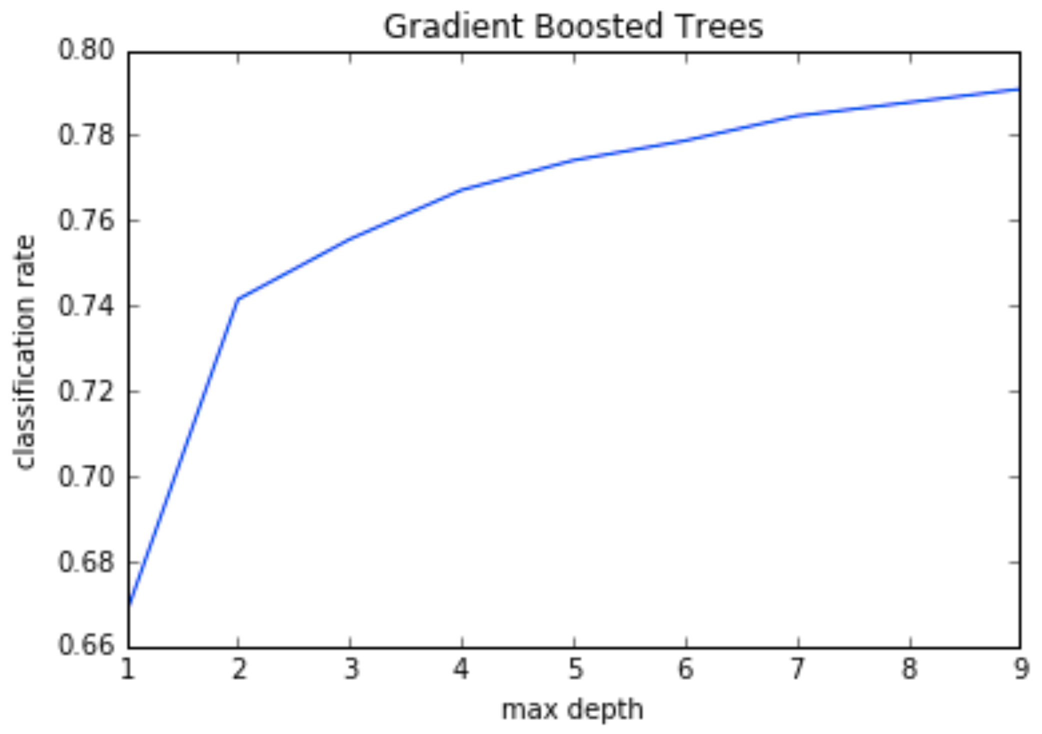
First, Random Forest. I tuned the n_estimators parameter:



I picked n_estimators = 80, for a classification rate of about 0.769, and a rather boring confusion matrix:

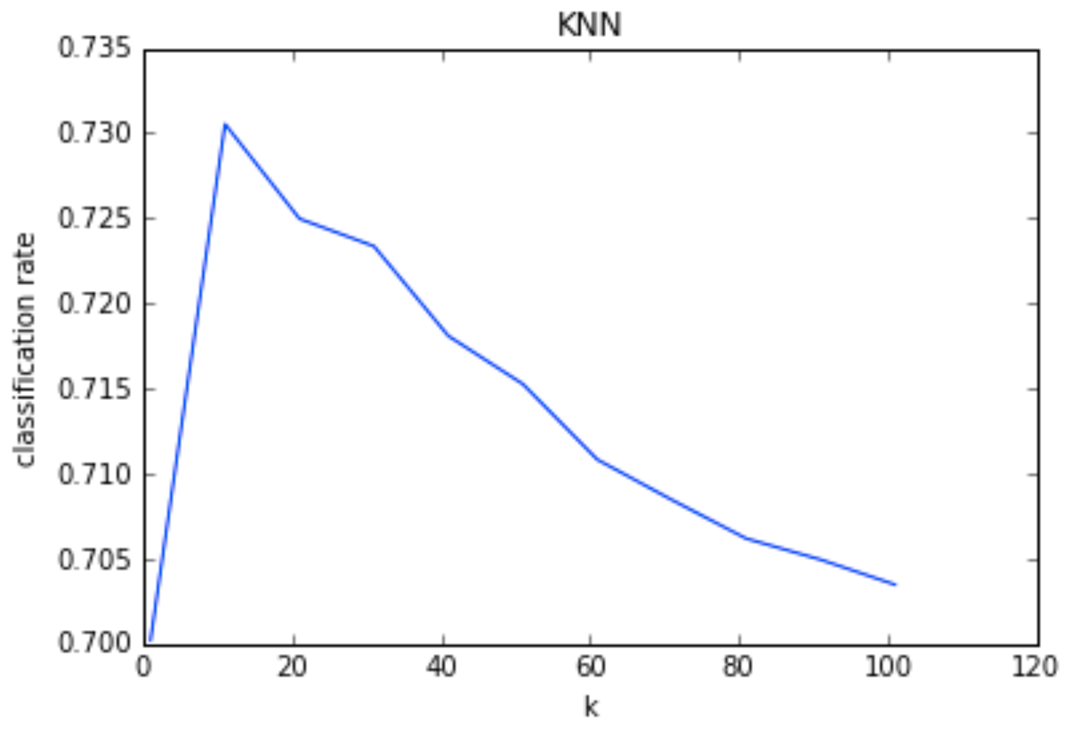


Next, I tried Gradient Boosted Trees (this time I didn't seem to need to downsample). I tuned max depth:



I chose max_depth = 9 (though there was an unpward trend still happening, it was pretty gradual, and the higher max_depth models were taking a long time to fit) for a classification rate of 0.79, and a classification matrix nearly indistinguishable from the above one (omitting for brevity's sake).

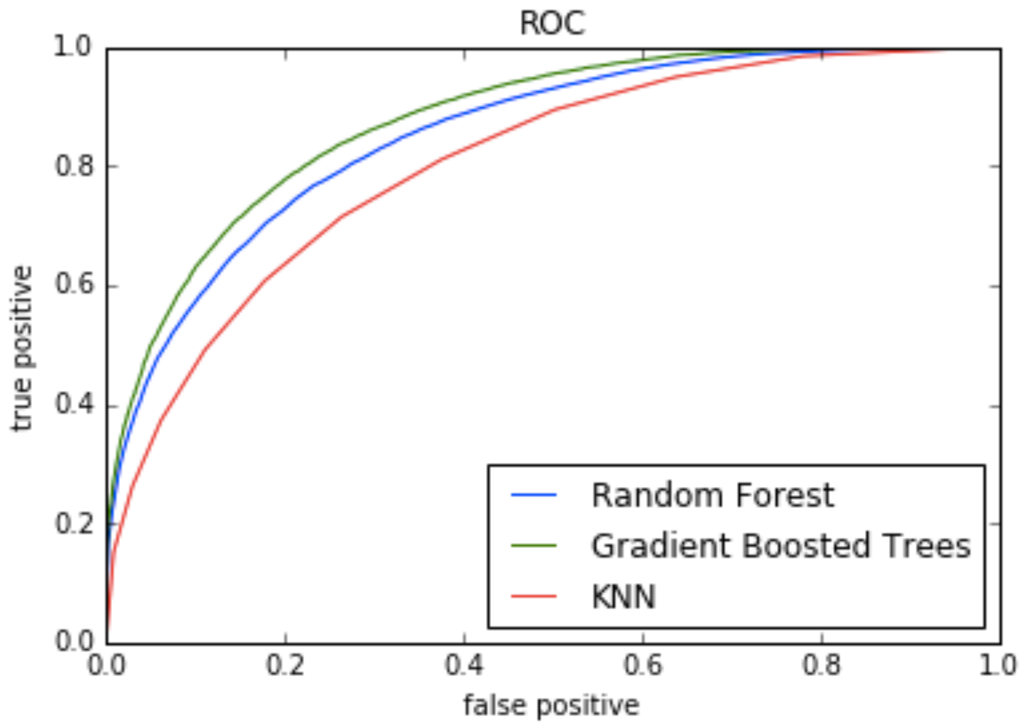
Finally, I tried KNN. I tuned k:



I picked k = 10 for a classification rate of 0.727, and another pretty similar confusion matrix (again, omitting).

Based on these results, if I were to pick via classification rates, the Gradient Boosted Trees would be the best model, followed by Random Forest, followed by KNN.

But the best part about doing binary classification is that you don't need to rely on classification rates. Here is a ROC plot of the three compared against one another:



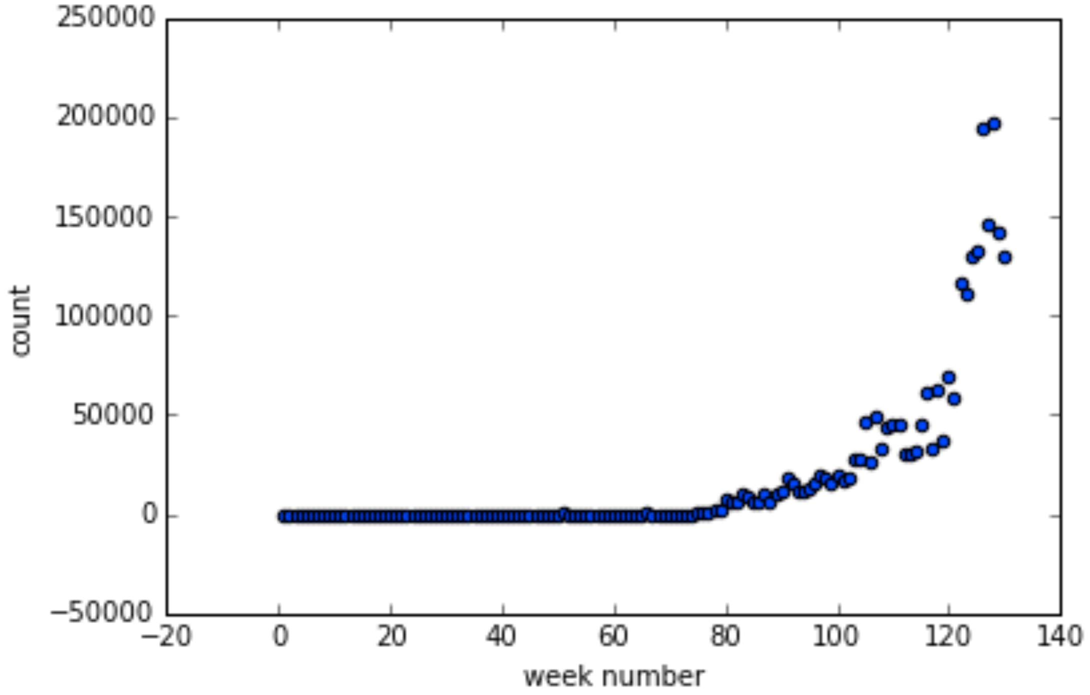
It looks like the ROC curve confirms the ordering done above using classification rates: Gradient Boosted Trees wins!

On the whole, I am much more pleased with this classification. The classification rate on the test set approaches 80% which is pretty impressive, given that all it was trained on was latitude, longitude, amount donated, and timing.

Time Analysis

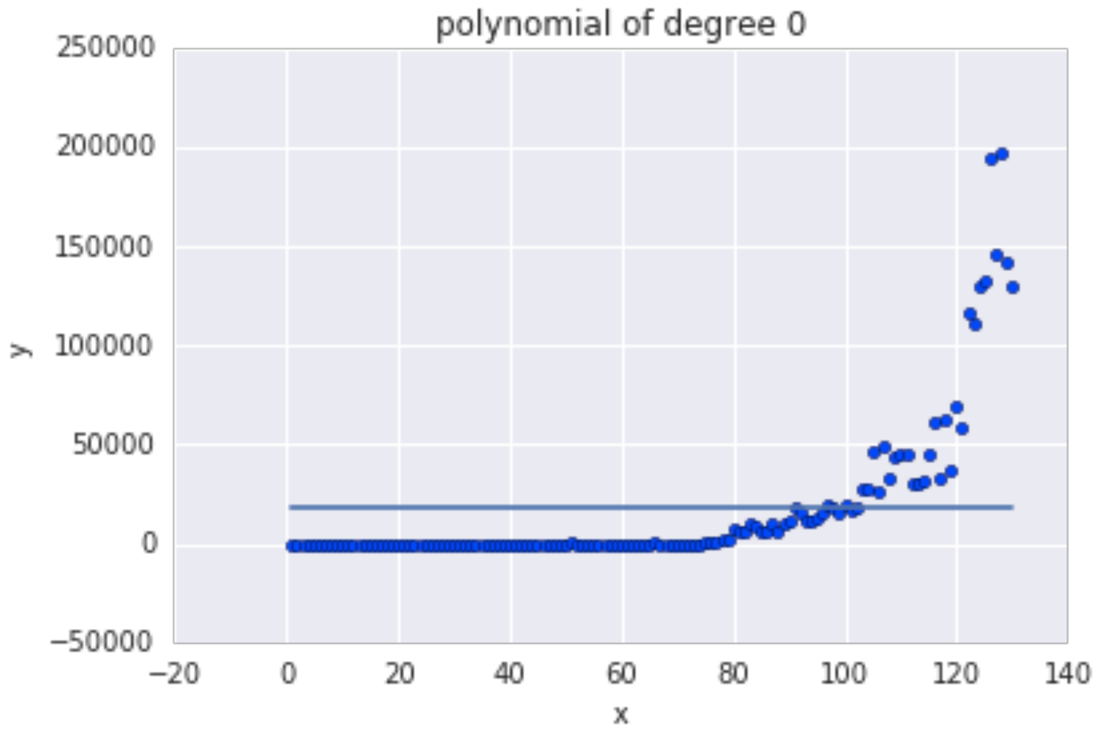
I had originally intended to model donation amounts as time progressed, but the plot showed earlier between date and donation amount disenchanted me a bit with the notion that there was much of a relationship there. Instead, though, there seemed to potentially be a relationship between time and the number of donations, which I was interested in exploring.

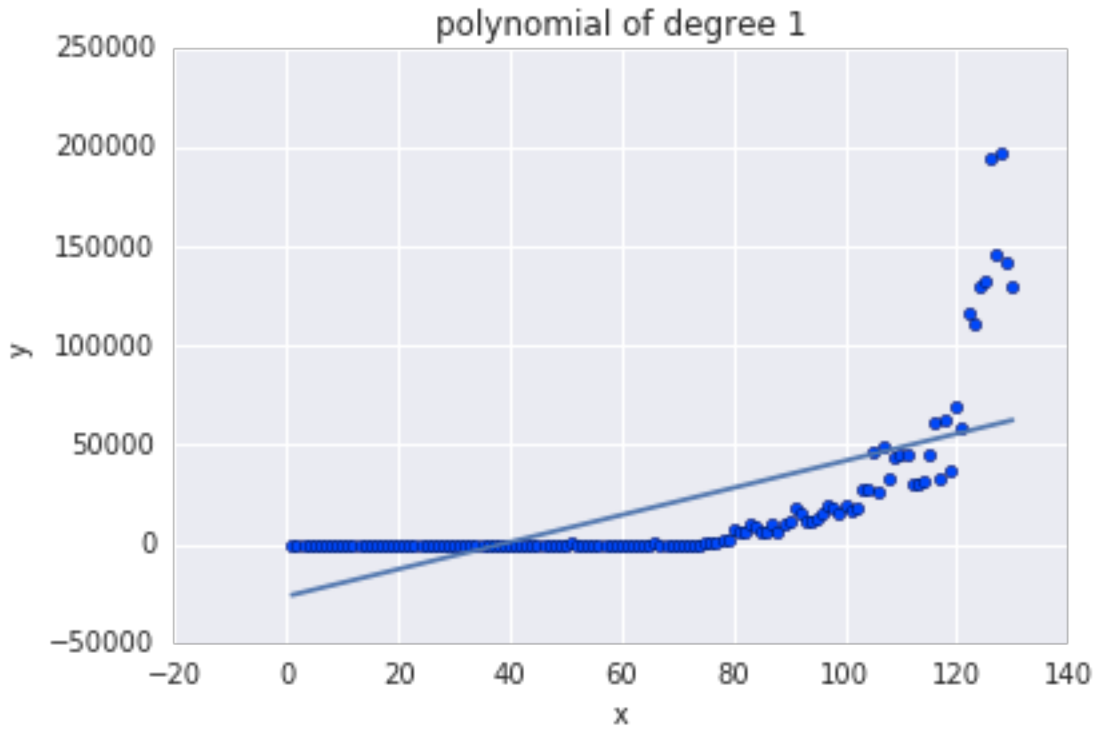
So I gathered the data into weeklong bins (so as to avoid fluctuations day to day), and totaled the number of donations per week. This produces a data set that looks like this:

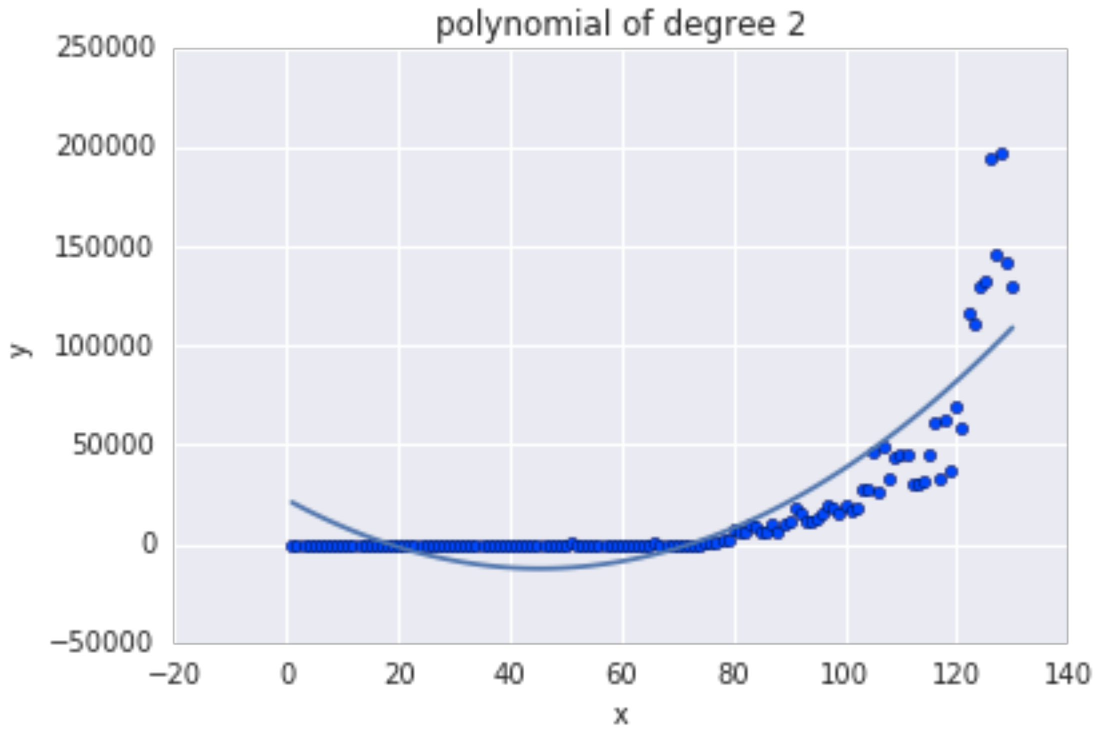


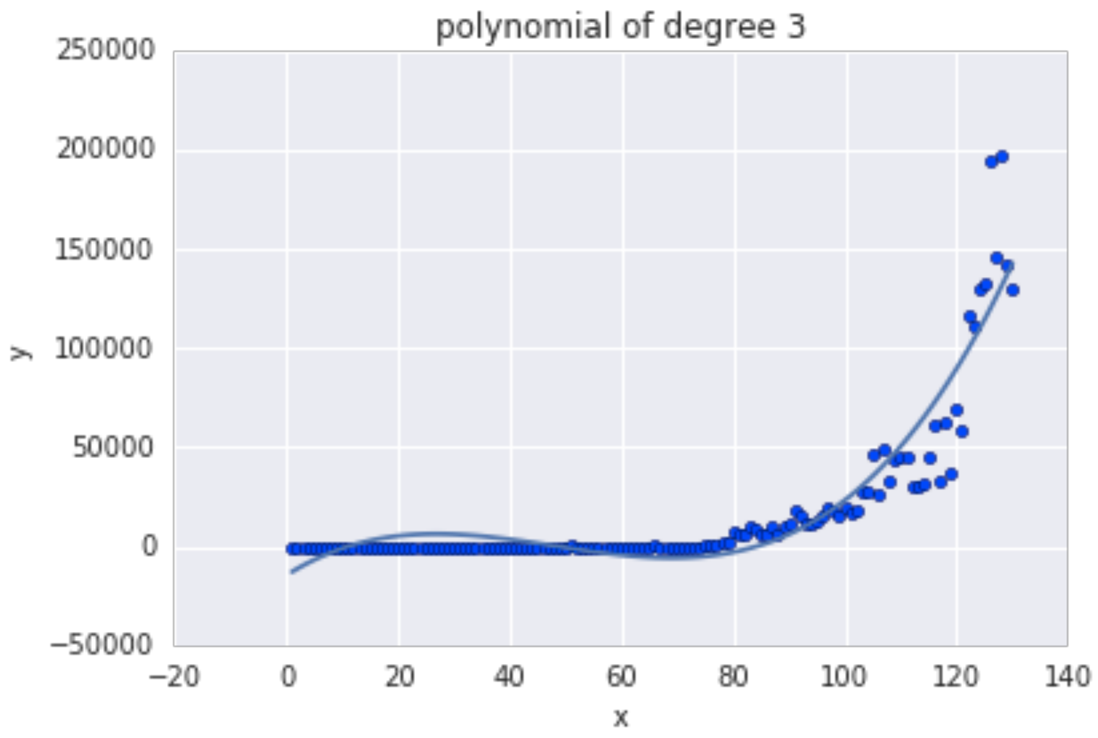
Clearly, there is some sort of relationship here, and ideally we can model it.

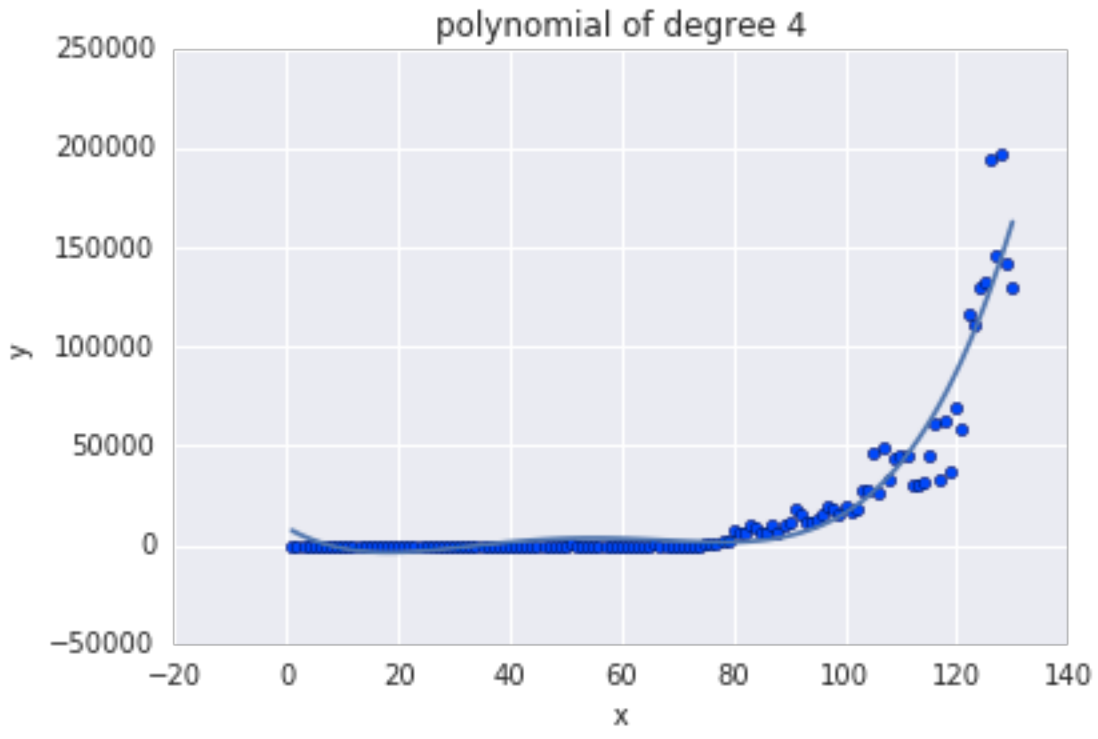
The numpy functions polyfit and poly1d make it trivial to create polynomial fits to data of arbitrary degree. Here are a few of the fits:

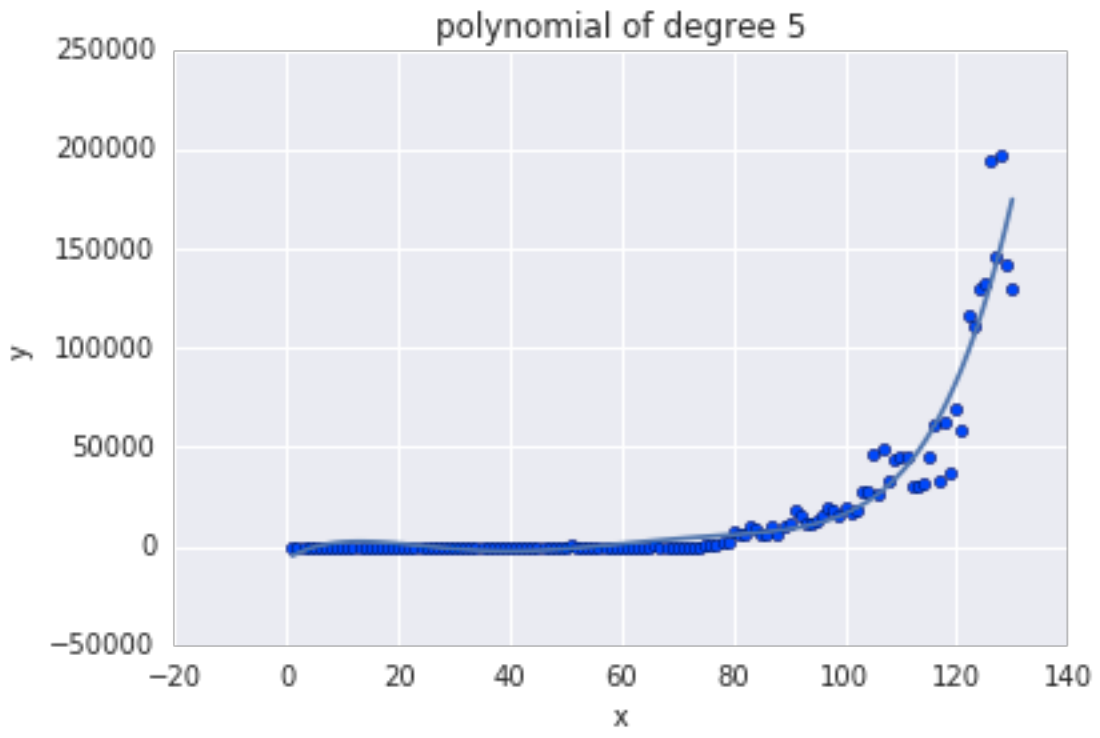


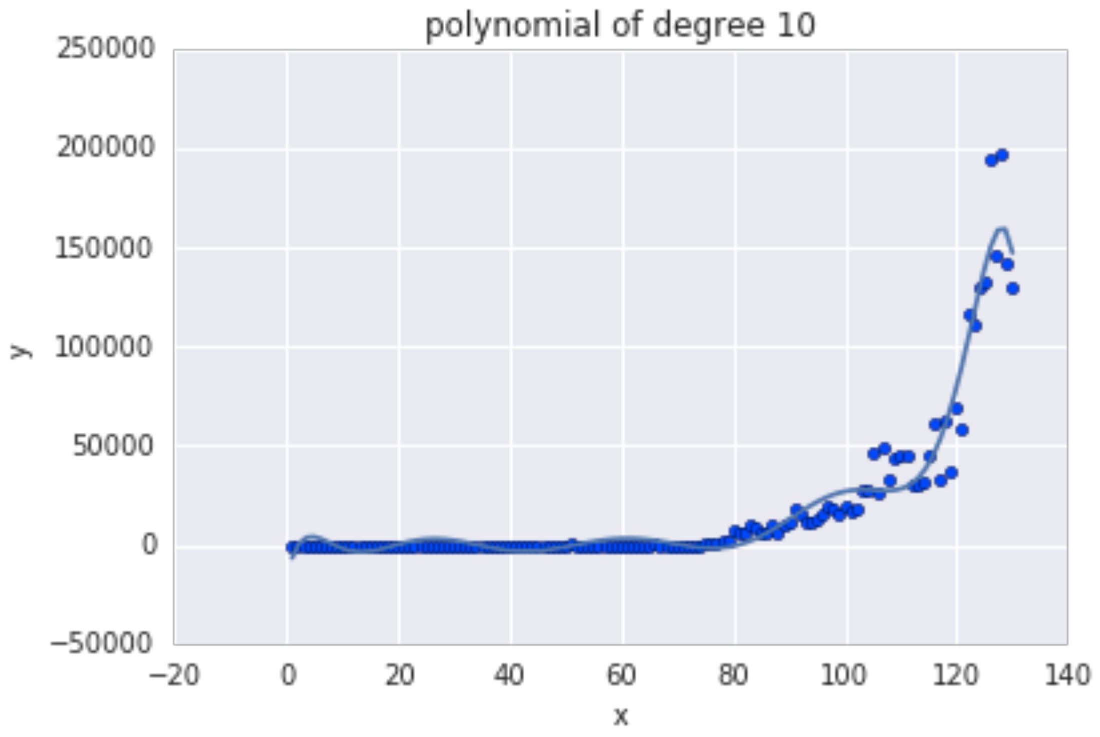

polynomial of degree 1
y
x
250000
200000
150000
100000
50000
0
−50000
−20
0
20
40
60
80
100
120
140


polynomial of degree 2
y
x
250000
200000
150000
100000
50000
0
−50000
−20
0
20
40
60
80
100
120
140


polynomial of degree 3
250000
200000
150000
100000
50000
0
−50000
y
−20
0
20
40
60
80
100
120
140
x


polynomial of degree 4
250000
200000
150000
100000
50000
0
−50000
y
−20
0
20
40
60
80
100
120
140
x

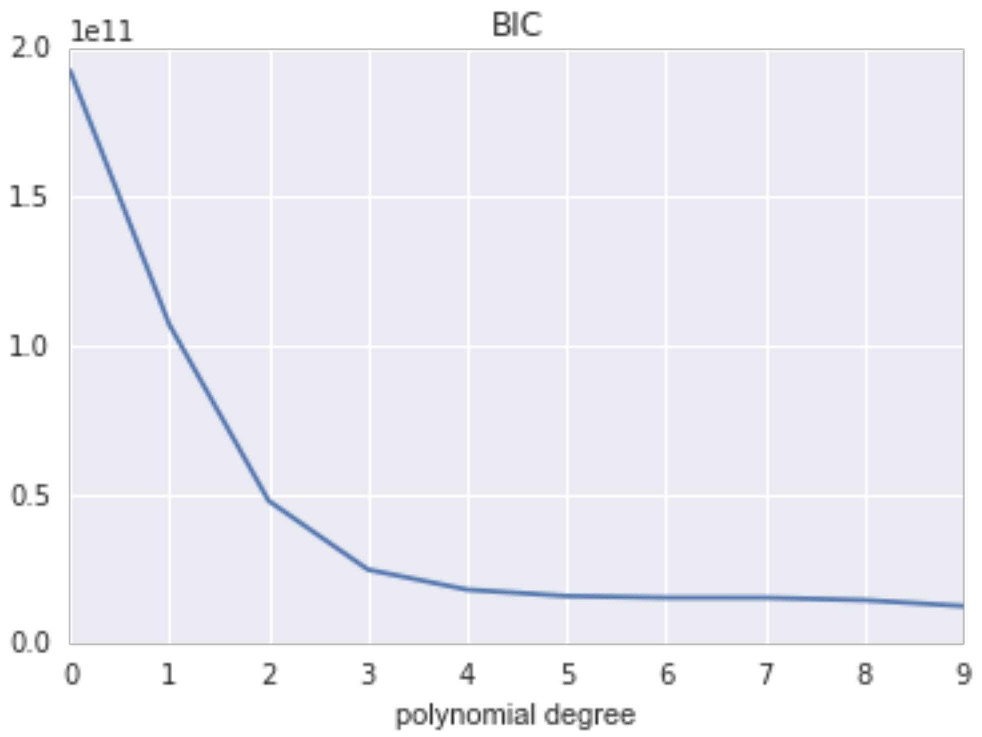


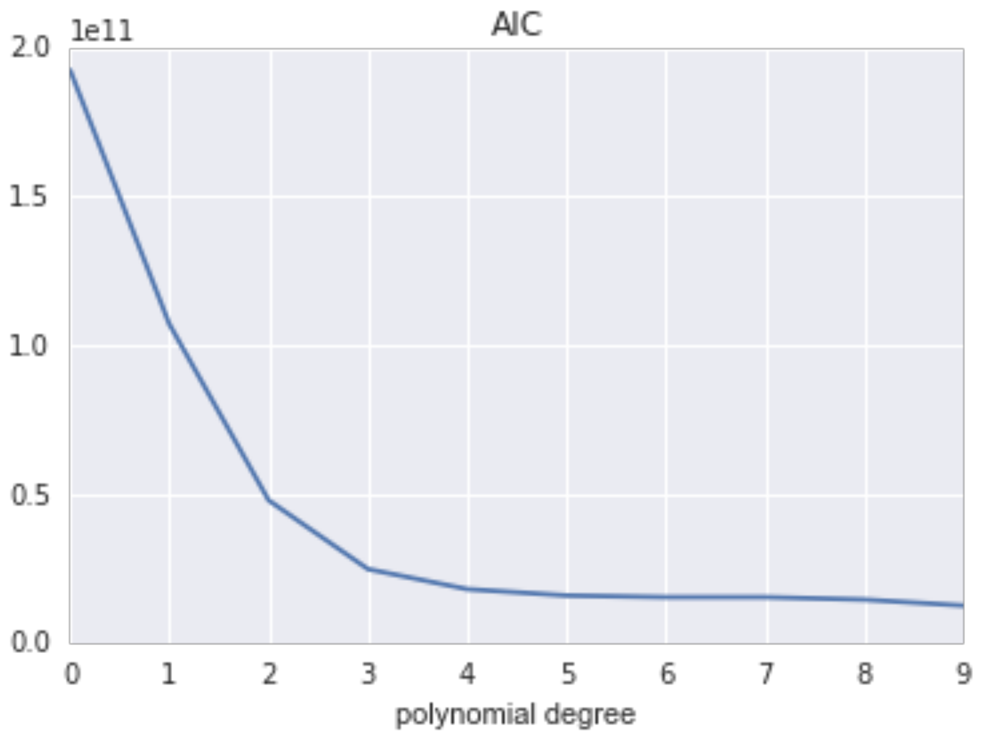


It is clear that degree of 0 severely underfits the data, and degree 10 seems to overfit it (note the bend at the far right around week 130). The question then becomes, where is the balance in this tradeoff. We can use Bayesian and frequentist approaches to weigh in on this question.

Following Jake Vanderplas's example in
http://jakevdp.github.io/blog/2015/08/07/frequentism-and-bayesianism-5-model-selection/ quite closely (which I also used in Problem Set 4), we can construct functions to compute the log prior, log likelihood, and log posterior. Because the polynomials results produced by numpy.polyfit are created so as to maximize likelihood, it is then trivial to compute the AIC and BIC. Vanderplas considers AIC a frequentist approach, and says BIC "quickly approximates the Bayes factor" in that same blog post.

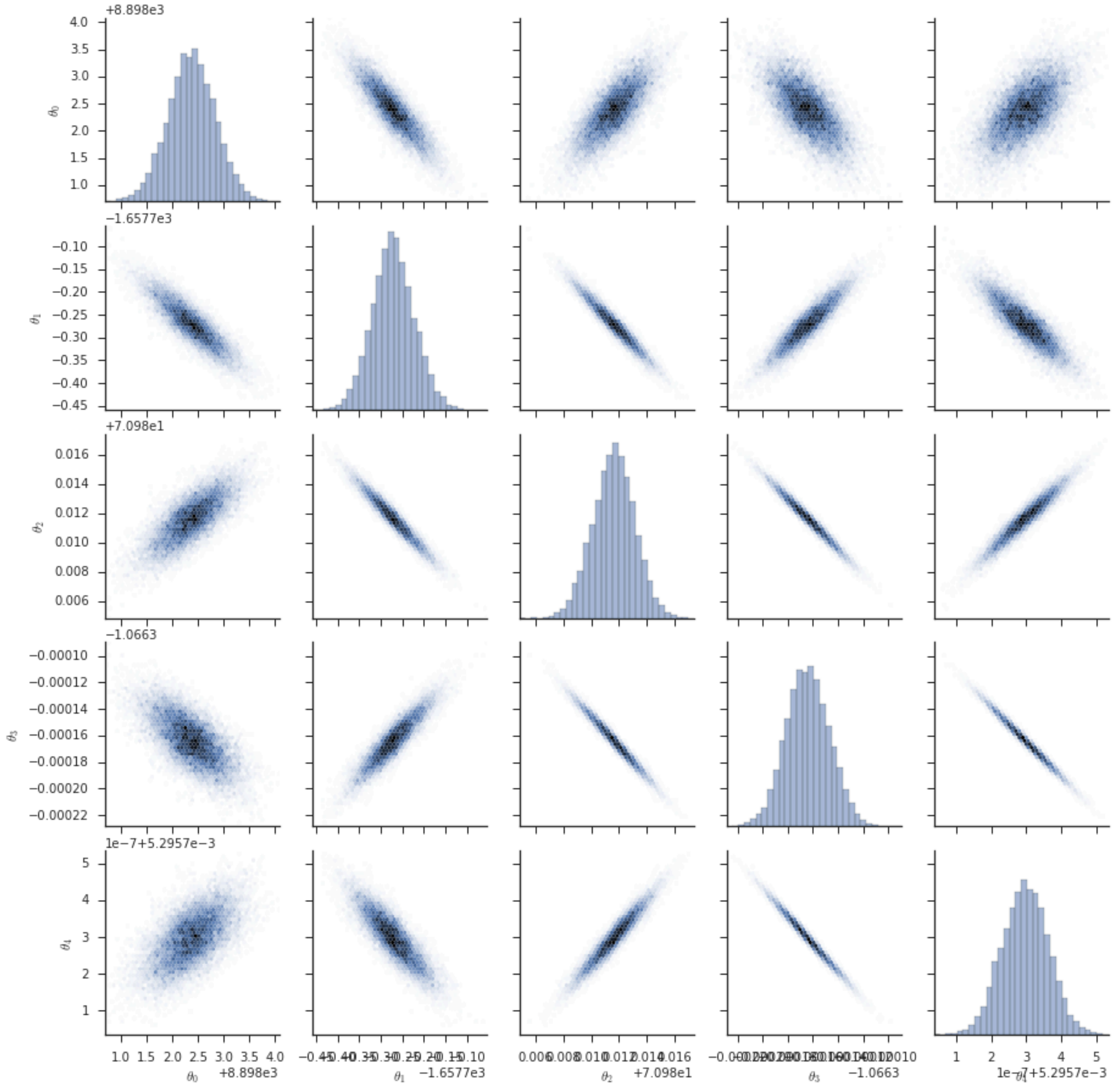
We can then produce the following plots:



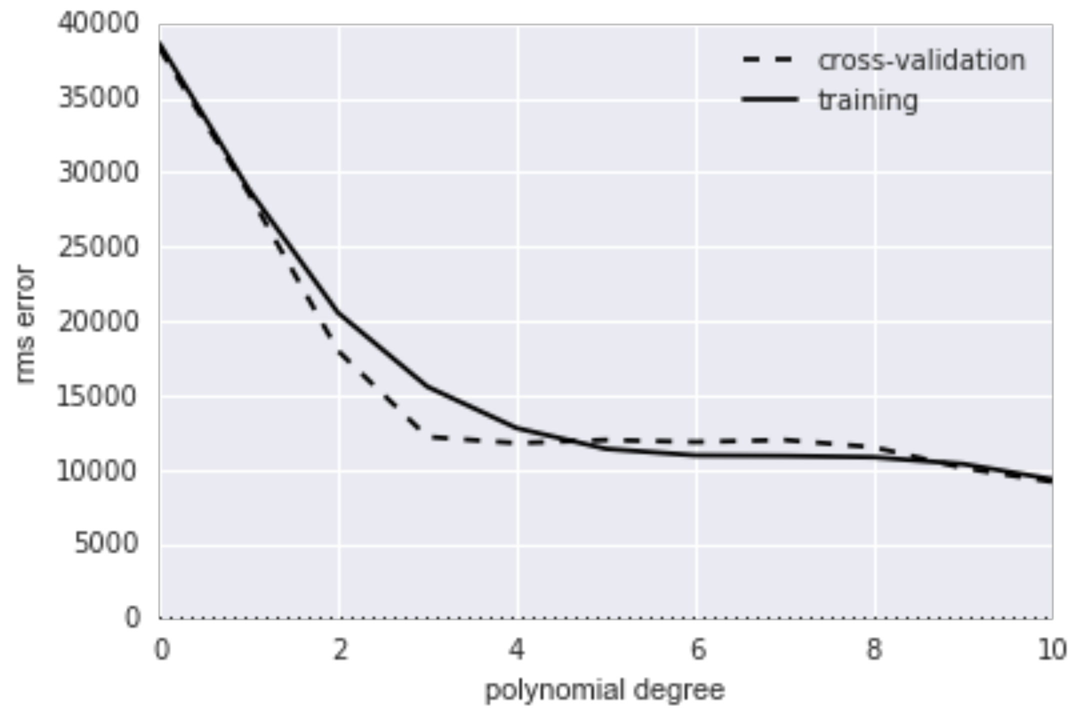


The BIC and the AIC approaches seem to agree quite well: beyond degree 4 or 5, there is very little advantage to increasing the number of parameters. Looking back at our original plots of the polynomials, this seems very plausible. Thus, in this situation, the Bayesian and the frequentist approaches produce very similar results.

If we wish to take the Bayesian approach further, we can go further down the road of Jake Vanderplas's post, and conduct some MCMC to find the posterior of parameters. Doing so produces a plot like so (for the degree 4 polynomial):

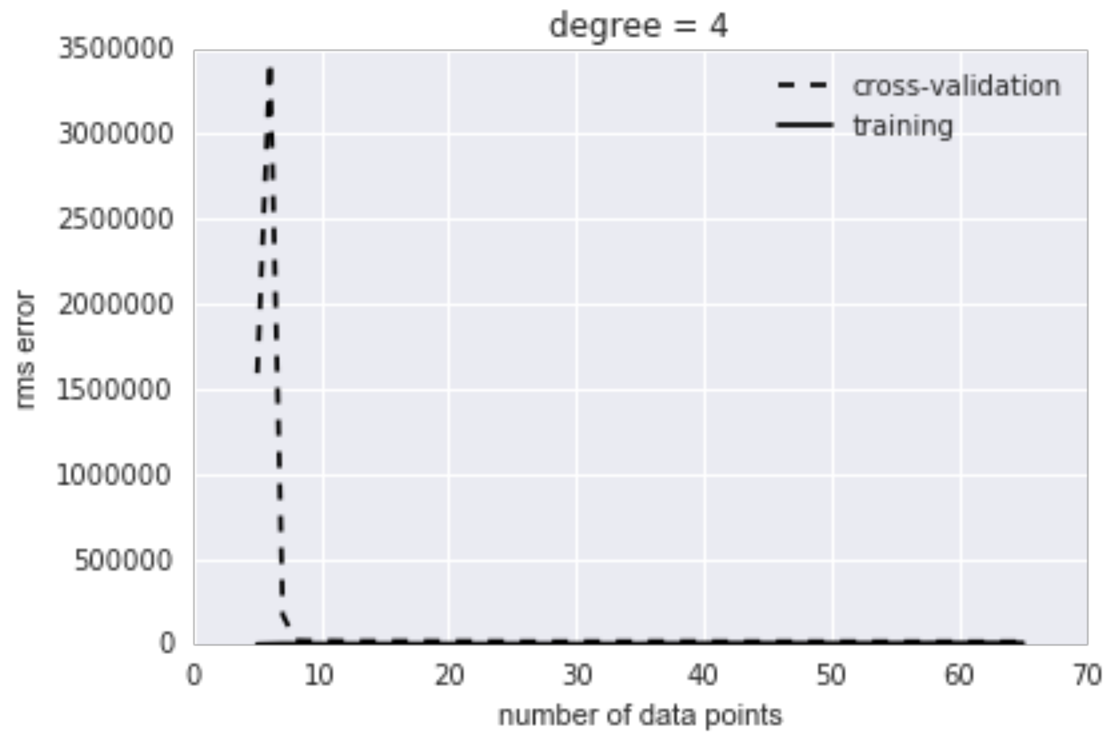


We can also closely follow problem 2 of Problem Set 3 (as recommended), and produce the following plot demonstrating how training and validation errors interact:



This offers further evidence that we are interested in a polynomial degree of about 4. Beyond that point, we stop decreasing the error for both the training and cross-validation sets.

Finally, here we have a learning curve for that demonstrates how the polynomial of degree 4 learns:



The initial cross-validation error is massive, as you might expect, because higher degree polynomials can jump around. But it quickly stabilizes in a very promising way.

Polynomial of degree 4 is the way to go!

Discussion/Conclusion

The results I found in each part of this project are documented pretty thoroughly alongside their diagrams and descriptions, so I just wanted to to dedicate a few words to saying briefly how much potential there is in a data set like this one. 18 columns offers the ability to analyze the data in so many different way. The above—geography, time, and classification—are just a few of the lenses that could've been used. I found myself wishing I had more time to dive into the possibilities.

In terms of general findings from this analysis, I'll make a few brief notes in summary:

1) There is a clear geographic nature to the donations, as demonstrated by the clustering. They are unambiguously not distributed evenly throughout the country, though of course, we would not expect them to be.
2) Geographic, time, and donation amount can tell you a lot about who a donation might be going to. In particular, it is possible to train a model to be relatively successful at distinguishing between the donations of two candidates.
3) There is clear variation in the number of donations made. The relationship is not linear, but is higher dimensional.

Future Improvements

As alluded to earlier, I originally wanted to also use the occupation and employment data as part of my classification modeling, but was stymied by sklearn. One thing that could be done would be to split that data into dummy variables, and to try using that data also as inputs.

I would like do more binary classification, perhaps pairwise on all the candidates—this would let me see which are most similar (and hence hardest for the model to distinguish), and which are least similar (and hence have good classification rates). It would also be very interesting to split data into Democrats and Republicans, and do binary classification trying to predict those labels.

N.B. I've made mention of certain blog posts and tutorial I used to do my analysis and create my plots in this document, but I have not done so exhaustively here. I've tried to document my sources as comments in the code or markdown notes (the nice looking confusion matrices, for instance, were styled based on sklearn documentation, which is linked to in the notebook).