Richard Bao and Shiva Mudide

Dr. Alvarez

PS 120

June 8, 2020

Collaborative Filtering on the General Election

Introduction:

There is a lot of potential for machine learning in political science. Election statistics are already plentiful, and as the world becomes increasingly data-driven, we expect that the availability of election data will only continue to grow into the future. We are especially interested in collaborative filtering, as we could not find any current well known application in the realm of political science. Collaborative filtering is a powerful and relatively new machine learning technique with useful applications to problems that involve user preferences. The underlying intuition behind collaborative filtering is that users who have similar preferences in the past will continue to have similar preferences in the future. As such, it is frequently used in product industries to predict user preferences for new products, by analyzing ratings of other users who have rated items similarly in the past, or with some sort of consistent correlation.

Traditionally, collaborative filtering has been used to create recommender systems. Perhaps the most famous example of collaborative filtering is the Netflix Competition, which challenged programmers to create models that could predict if a user would like a new movie based on how users with similar movie preferences rated that particular movie.

We were inspired by this Netflix Competition, and we realized that conceptually, there are many similarities between election voting and product rating. In fact, these seem to be analogous problems, as there is a very interesting direct mapping between users (and ratings for movies) to counties or congressional districts (and votes for candidates). Of course, there are external factors as well, such as incumbency and political advertising. But these features are also analogous to brand recognition and product advertising, both as a function of time. Companies like Netflix have been able to account for these features in their algorithms without much concern. Since collaborative filtering considers data at the highest level of user preference, the hope is that these factors are implicitly included as abstract features. In fact, this is the "black box" aspect of collaborative filtering in general, which operates with the belief that all the data needed to predict a user's preferences is already encoded in the preferences of the other users, and the assumption that these external factors affect the other users in similar ways and therefore already manifest in their reported ratings. So we decided to explore the applications of collaborative filtering to the election process. We believe this is a relatively underexplored and possibly novel approach to election forecasting.

Initially, we wanted to focus only on Presidential primary election data. Unlike the Presidential general election, the Presidential primary election is staggered, in that states vote on different dates, so the results slowly trickle in. As a result, the presidential primary takes place over an extended period of time, and in between, with only a handful of states and their counties reporting results, there is a lot of uncertainty. We thought it would be an interesting problem to try to use collaborative filtering on this sparse data, to "fill in the

blanks" for the counties that have not voted yet. By doing so, we can hope to predict the results of the election in real time, by updating and retraining the model every time a new county reports results. Of course, since there is no ongoing competitive primary election, we would have to simulate one by removing the candidate columns corresponding to the most recent election year, and then incrementally adding their values back, county by county, to show that at a certain point, the model will converge on a prediction for an overall winner. The primary election is a very complex system, and it is certainly worthy of rigorous machine learning techniques. This is an important problem because it can help candidates focus campaign efforts and allocate funding more effectively. Accurate preference-based primary predictions will also allow candidates to better understand the preferences of the general electorate at a county level. Businesses and stock market speculators can also benefit from these predictions, in order to prepare for potential political and economic consequences. Moreover, such a model has the potential to be an important diagnostic tool for our democracy, by evaluating how different counties's voting preferences may converge or diverge over time.

As we continued to learn about the current problems in political science, we also realized that collaborative filtering has some potential in the general election as well. Since the general election popular vote is decided practically overnight, we cannot use incoming county level data to predict the results of an ongoing general election. However, we may be able to predict in advance using polling data as a substitute. Currently, polling data is either too general, taken over the entire country, or too specific to a particular locality. The most

common practice is to conduct broad, sweeping polls across large regions, but these fail because they assume a uniform distribution of voters, while in reality there are localized pockets of voting preferences and priorities attached to specific communities. Moreover, these sweeping polls fail to give any useful information about swing counties, as the crucial difference required to swing a county is lost in the overall average. Instead of conducting broad, sweeping polls, a better approach would be to conduct polls on the county by county level, but there is no polling service that can realistically poll every single county or congressional district in the United States.

With the risk of being too optimistic, we propose that collaborative filtering may offer a new approach. If we can obtain sufficiently accurate polling results for only a small subset of counties prior to the general election, we may be able to train a collaborative filtering model to "fill in the blanks" for the rest of the counties, effectively extrapolating their most likely voting preferences based on how they voted in the past relative to that small subset of known counties.

Data:

The primary dataset we will explore is the David Leip election atlas. This includes county level voting data for both the primary and general elections of every election year from 1789 to 2016. Of course, for the sake of consistency, we will only use the years after the annexation of Hawaii, the 50th and last state in the union. Although territories like Puerto Rico can vote in certain primaries, we will limit our scope to only consider the 50

states. In order to obtain this data, we need to parse the website, as it is not freely available in csv format. However, the process of parsing is not straightforward.

The data is scattered across a series of nested web pages. The main web page is a user interface with HTML elements that allow the user to select parameters for election type, year, and state. Upon submission, these HTML elements send a POST request query through an ajax object, which performs an asynchronous call to retrieve data from a protected SQL database. This data is then returned to a callback function that dynamically renders it onto the current webpage. To get to the county level voting statistics, we need to repeat this process for every state for every year we wish to retrieve. Then, once we get to the state level, the county level query options are revealed. At this point, we can submit a query request for each county. Each county query will load an html page with the county level voting statistics displayed within the first html table element.

Conveniently, the necessary POST parameters for each step in this process can be parsed from the map display, which partitions the United States into shapes representing states, and further partitions these states into shapes representing counties. These parameters are exposed because the web master chose to make each shape an HTML link object, with the POST parameters encrypted as URL parameters, instead of using a responsive element to send JSON serialized data to the backend PHP route controller. We can therefore parse the necessary POST parameters from the page source at each step, and submit the corresponding queries by simply navigating the corresponding URLs.

We used the python package urllib with a modified user agent to send over a hundred thousand requests to the leip atlas server, retrieving a total of 50850 observations for democratic primaries and 65021 observations for republican primaries from the years between 2000 and 2016. To extract the necessary content from the html markup at each level of the parsing, we used the Beautifulsoup library to exhaustively iterate through all the html file and identify the elements of interest. Specifically, we searched for href tags in link elements to get url parameters, and row tags on table elements to get the observations at the finals step. In order to identify all the counties, we first loaded a csv file of county fips codes beforehand, and read entries from that file as inputs into the parsing script, to query individual counties and make sure we covered all of them.

We attempted to implement multithreading to split the counties list into subsets handled by separate threads, but this functionality is not compatible with the url request library, as a result of inherent limitations on how many requests a machine can send out. Originally, due to the nested structure of the website and the exhaustive search design of Beautifulsoup, the parsing algorithm was O(n^5) in time complexity, but were able to reduce this to O(n^3) by realizing we could skip some steps by a basic template for the url parameters. Regardless, the entire process took several hours. Unfortunately, the data before 2000 is harder to access, as it is not stored in the same format on the website, so we decided to proceed without it. However, it is still possible to parse, so given more time we would develop a second parser for this data in order to train on longer time horizons.

A secondary, freely accessible dataset is available on the MIT Election Lab website. Until we complete the parsing scripts for the Leip data, we will likely use this dataset, as it is easier to obtain. The format of this dataset is as follows: each row corresponds to a county and a year, with the counties sorted alphabetically by state and numerically fips code, which is a unique identifier used by the U.S. Census Bureau. For each row, there are additional column features for office, candidate, party, candidate votes, total votes, and version. For the scope of this project, we only care about the candidate, candidate votes, and total votes features for each county and year. However, this dataset only contains general election data from 2000 to 2016. Its dataset on primary elections is even more limited, so we will only use the general election data.

To ensure that the voting statistics are weighted equally across all counties, we normalize by population. For each county, we convert the total vote counts for each candidate to a ratio from 0 to 1. We also get rid of observations with missing values, as well as observations with ambiguous candidate labels, such as “other” or “undecided.” All this data is saved to a csv file. Using pandas, we are able to load it for use in our training scripts, where we can perform additional preprocessing, involving actions such as applying min-max scaled normalization, or replacing the continuous range of numerical vote ratios by a one hot encoding on the discrete classifications of loss, victory, and draw, within a certain uncertainty.

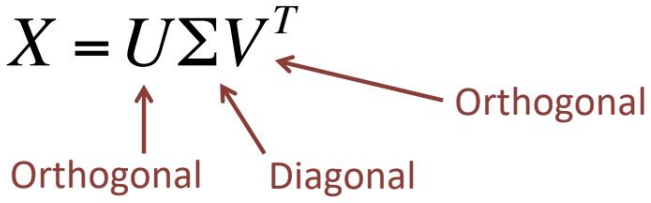
With the Surprise SVD package, we were able to use the observations as inputs directly. However, we also experimented with a custom numpy implementation of SVD in

the early phases, which we developed ourselves. This numpy implementation required a different format. In order to use the parse data for training SVD using the numpy method, we must reformat the dataset to map each row to a county and each column to a candidate. The mapping from rows to counties will be one to one, unlike the formatting in the raw dataset, which contains different rows for the same county for different years. To achieve this one to one mapping, we will transpose the data for different years as additional columns in the reformatted dataset. For every county, for every election year, we added a separate column for each candidate running that year, containing information about the number of votes obtained. In the interest of studying time evolution of voting preferences, we will sort the candidate columns from left to right by year, but we do not need the explicit years as features in the dataset. To implement these reformatting schemes, we loaded the original dataset into a pandas dataframe and performed the necessary matrix and element by element manipulation using pivot methods. The final format of our dataset should look like a two dimensional matrix with N rows and M columns, where N is the number of counties in the United States, and M is the number of total candidates throughout all the years. Correspondingly, the entry for each cell in the final data frame represents the proportion of the voting electorate in the county corresponding to that row who supported the candidate corresponding to that column.

Methods:

SVD:

SVD takes a data matrix, X, where every column of X is a data point (election candidate) and every row is a feature value (county), and factorizes it into an orthogonal matrix (U) , diagonal matrix (Σ), and another orthogonal matrix (V). It generalizes the spectral decomposition, well known in fields like quantum mechanics but only works on hermitian matrices, to any non-sparse matrix. The first k columns of U is used to define a low dimensional subspace of the data points. This subspace is the k-dimensional subspace with the smallest residual. What this means is that SVD gives a handle on finding the low dimensional subspace of dimensionality k that captures as much of the variability in the data as possible in a k-dimensional subspace of the data.



For our data $X_{i,j}$ is the percent of people that voted for candidate j in county i. If X were completely observed then the factorization is the standard SVD. The U matrix would then correspond to a set of k latent factors that describe any given county and the V matrix corresponds to the set of k latent factors that describe any given candidate. It is important to note that none of these features is interpretable.

Collaborative Filtering:

In actual implementations the matrix X is usually very sparse so a simple SVD can not be applied. The problem then becomes formally known as a collaborative filtering problem. In a sense this is like the "reverse" of unsupervised learning. We only have labels

for our data, but we have no features. We aim to fill in the missing values of our matrix X. In our case the missing values are the percentage of votes each candidate in the new election gets from each county. Without any assumptions about the ground truth matrix, X, it is impossible to recover the missing entries. They could really be anything, and any algorithm would have no idea what they are. This notion does not daunt us though. It often comes up in many fields of science, like compressive sensing. In compressive sensing, we know that there is no way to recover an unknown, arbitrary signal of length n by using less than n measurements. In compressive sensing we often assume the signal is sparse. Can we do the same for matrix completion? Then the matrix X would be filled with all zero values for its missing entries. That does not make much sense. Instead, we make the key assumption that the unknown matrix, X, has low rank (technically we assume that the matrix is approximately low rank). For example, if X were rank one, then all the rows and all the columns would be multiples of each other.

Convex Optimization:

Given that all we know about the ground truth matrix is that it agrees with the data matrix on its known entries and that it has low rank, we attempt to recover X by solving the following optimization problem:

min rank (χ) *such that* χ *agrees with X on its known entries*

Here, χ is our best guess of the ground truth matrix, X. Sadly this rank problem is NP-hard and there are no good algorithms known to solve it. The objective function is even non-convex! To understand why, try proving that the mean of n rank-k matrices does not

need to be a rank-k matrix itself. There is still hope though. If we can see the above rank-minimization problem in the light of an l_0 minimization problem, we can relax the problem into an l_1 minimization problem. The SVD we described above allows us to do this. The rank of X is the number of non-zero singular vectors in the diagonal Σ matrix. We therefore simply want to minimize the support of the diagonal Σ matrix, subject to χ agreeing with X on its known entries.

We then convert the l_0 minimization problem to an l_1 minimization problem by requiring that we minimize the total sum of the singular vectors in Σ. This optimization is known as nuclear norm minimization. We are essentially minimizing the sum of the singular values subject to consistency with all our known information about the matrix X.

Surprise Python Scikit:

The above convex optimization methods were difficult to implement from scratch. We did not find any python packages viable for our data. This is probably because the methods were only discovered very recently. After attempting to write all the code from scratch, we found a python package that used a slightly worse optimization method, which could still be of use to us. The Surprise package alleviates the pain of dataset handling. It allows users to easily load in their own datasets. It also provides several prediction algorithms based on neighborhood models and matrix-factorization based models. It is slightly different from convex optimization though.

It maps both the counties and candidates to a join latent factor space, such that county-candidate interactions are modeled as inner products in that space. Each county is

associated with a vector and each candidate is associated with a vector. For a given candidate its vector represents the measure of the extent to which the candidate possesses certain unknown latent factors. These could be anything from how left or right leaning their policies are to the color of their hair to a linear combination of both. For a given county, the elements of its vector represent the extent of interest the county has in the candidate factors. The dot product between each county and candidate vector represents the interaction between them. Here the challenge is to calculate the feature vector for each of the counties and candidates. To learn the vectors, the system minimizes the regularized squared error on the set of known entries. More specifically, we can express this in the learning formulation below:

$$\underset{U,V}{\operatorname{argmin}} \frac{\lambda}{2}\left(\|U\|_{Fro}^2 + \|V\|_{Fro}^2\right) + \sum_{ij}\left(y_{ij} - u_i^T v_j\right)^2$$

We aim to gradually learn the matrices U, and V simultaneously. This is opposed to the closed form solution we had for U and V in a standard SVD. The percent votes given to candidate j by county i is given by $u_i^T v_j$. The system learns the model by fitting the previously observed ratings. Yet, we also want our model to generalize by predicting data we have not seen yet, the missing values. To ensure this we regularize to prevent overfitting. We regularize in the first term in the above expression, by the sum of the squares of the frobenius norms of U and V. The second term in the expression is the plain loss function, squared loss.

We optimize with alternating stochastic gradient descent. We keep some subset of the parameters to optimize, like a single column, while keeping all the other parameters fixed. Since we are learning V in addition to U, in each iteration of gradient descent the values for v_j will have changed.

Since this model makes predictions based on the relative voting behaviors of different counties, this should account for external factors like incumbency. However, we can bolster our model by separately learning these external features (like GDP growth, economic condition, incumbent party data, inflation rate) through a simple regression. We can then create an ensemble with our SVD model to create a general bilinear model.

In this model we have features of both the counties and the candidates, which we previously learn. These features are represented as a vector, in addition to the county and candidate ID. Both these are projected onto a low dimensional space, and how much a specific county favors a specific candidate is represented by a linear inner product in this low dimensional space.

The Surprise package also allows us to add bias terms to account for biased counties or biased candidates. We found that biased counties do not really exist in our formulation, since we normalize the number of votes by the total number of votes to get a percentage. Certain candidates could be particularly appealing though, which would make them biased to gaining more votes from counties in that sense. The overall score we predict a candidate will get from a county is not just the interaction between the county and the candidate, but rather the sum of the interaction with the relevant biases.

Implementation:

When we began to think about how to design our model, we realized that the Presidential primary dataset will have more nuanced complexities than we initially imagined. The primary election data is sparse, as most primary candidates drop out during the primary before reaching every county. This means that there will be empty values for these candidates in the training data, corresponding to the counties they failed to reach. Consequently, training on the primary election data would be much more time intensive than expected, but still well within acceptable parameters. This is no issue for the model, because modern SVD algorithms are designed specifically for sparse data, and SVD prediction already involves validating guesses for missing values and minimizing error. These empty entries will simply add new degrees of freedom to the gradient descent, as each step in the error minimization would involve validating guesses for these missing values in the matrix factorization as well.

But interestingly, this may cause the model to associate a kind of time dependence for country predictions. For each candidate that drops out, there will be two distinct regimes in the training data, corresponding to the counties that voted before the candidate dropped out, and the counties who voted after the candidate dropped out. The latter subset of counties will have all zeros for that particular candidate, and this shared feature may force the model to learn to strongly cluster the preferences of counties by the date of their respective primaries. However, this may also cause the model to make other, less intuitive associations, and therefore it will be harder to diagnose unexpected results. If this is a

problem, we can choose to remove the candidates who dropped out early and only keep the later stage candidates for each year in the training data, and renormalize the voting ratios accordingly. We can also do this for the ongoing election prediction columns. But there is also a possibility that this implicit time correlation may be beneficial. To determine how the model will react, we would need to train it exhaustively across many different scenarios. Moreover, it is likely that candidates who drop out early were unpopular, and would have had very low values near zero in the dataset anyways, and thus the impact of them dropping out would be minimal.

General Election Visualization

To first establish proof of concept that SVD can effectively train on our data, it is useful to show that there are intrinsic patterns in the data. Although it is hard to define the specified "abstract features" that the SVD algorithm identifies and uses as templates, or prototypes to describe the actual feature space, we are able to identify which abstract features are the most important, as the decomposition sorts them by their contributions to the covariance matrix. We can extract the first two columns of the left unitary matrix, representing and plot these two columns against each other on a two dimensional plot. This essentially visualizes similarity in "abstract feature preference" among the counties. Each point in the resulting plot corresponds with a county, and the distance between two points in the plot correlates with similarity in voting preferences, based on the data on what candidates they have favored in the past. We can label these points in a variety of ways, but we decided to use a simple labeling that colored each county point by its respective state,

using the first two digits of its FIPS code as the state id. We can also do the same with the first two columns of the right matrix to visualize how similar candidates are by their abstract feature expression. We can also feed these results into more advanced clustering algorithms to discover any more subtle patterns. Furthermore, we can explore an additional dimension by creating a 3D plot with the third column as well.

We decided to begin by visualizing the general election first. The general election data is a complete data set, with no missing values, so we simply perform SVD decomposition algebraically. To facilitate the visualization of distinct patterns in the data, we used a preprocessing procedure to encode definitive wins in a county as a value of 1 for the candidate, definitive losses in a county as a value of -1 for the candidate, and uncertain results like those between 0.45 and 0.55 as a value of 0, as close margins can be swung easily. We believed that this encoding would make the data less volatile. We tried a variety of hyper-parameters for the number of training epochs, the number of factors, and the bias term. We were able to see definitive patterns in the data, especially around the range of less than 10 but more than 2 factors in the SVD decomposition. The number of ranks corresponds to the rank of the SVD approximation. As we increase the number of factors, the approximation becomes very fuzzy, and the clusters that are apparent at lower numbers of factors merge together to form a single point cloud, which is not very informative. We tried visualizing three columns in a three-dimensional plot, and it is clear that the third column adds a lot less variance in the point cloud compared to the first two columns, as the main patterns in the visualization were still those in the plane of the first two columns.

Interestingly, we also see a decomposition of singular values in the SVD visualization, as expected. The bias term also had a surprisingly significant effect on the results, and we concluded that the patterns were more well defined without a bias term

Primary Election Visualization:

Next, we visualized the primary election data, separately by party. We did not encode the results like we did for the general election data, as each county can have a different number of candidates, as many candidates drop out during the primary. This would require us to encode separately for each county, based on the number of total candidates, but we did not think to save information on the total number of candidates per county in the data parsing procedure detailed above. Given more time, we would explore this option more thoroughly.

To train the visualization, we only used the surprise SVD implementation, as the primary election data is much sparser than the general election data, so an algebraic SVD decomposition is not possible. We tried a variety of hyper-parameters for the number of training epochs, the number of factors, and the bias term. We were able to see definitive patterns in the data, especially around the range of 4 factors in the SVD decomposition. The number of ranks corresponds to the rank of the SVD approximation. As we increase the number of factors, the approximation becomes very fuzzy, and the clusters that are apparent at lower numbers of factors merge together to form a single point cloud, which is not very informative. We tried visualizing three columns in a three-dimensional plot, and it is clear that the third column adds a lot less variance in the point cloud compared to the first two

columns, as the main patterns in the visualization were still those in the plane of the first two columns. Interestingly, we also see a decomposition of singular values in the SVD visualization, as expected.

For the Democratic primary, there were definitive patterns, and we saw that there was a large degree of clustering by state as well. This essentially created a well defined "point cloud" of counties for each state, demonstrating that counties in the same state have more closely aligned preferences. This relationship is likely due to the fact that residents in a state largely rely on the same institutions and industries, but may also be explained by any time dependency on the favorability of candidates. As the primary progresses, certain candidates may become more popular due to increased exposure, or alternatively certain candidates may become less popular due to controversy, or drop out entirely due to a lack of success. This essentially clusters counties by the date of their respective voting days, which groups counties together by state. Moreover, this relationship may also be caused by the tendency of certain candidates to focus heavily on specific states. Interestingly, compared to the visualization of the general election, which revealed clusters classifying states into distinct regions, the clusters in the visualization of the Democratic primary had a lot more mixing between different states. In other words, the Democratic primary clustering, as mentioned before, clustered counties within the same state together, but if we were to compare the individual centers for each state in the clusterings, the variations in distances would not be as high as those for the general election. This makes sense, as the Democratic primary candidates are much more similar to each other than general election candidates,

since democratic primary candidates come from the same party, and to some extent have the same policy agendas.

For the Republican primary, there were definitive patterns, and we saw that there was an even larger degree of clustering by state than for the democratic primary. Again, this demonstrates that counties in the same state have more closely aligned preferences. We similarly hypothesize that this relationship is likely due to the fact that residents in a state largely rely on the same institutions and industries, but may also be explained by any time dependency on the favorability of candidates. As the primary progresses, certain candidates may become more popular due to increased exposure, or alternatively certain candidates may become less popular due to controversy, or drop out entirely due to a lack of success. This essentially clusters counties by the date of their respective voting days, which groups counties together by state. Moreover, this relationship may also be caused by the tendency of certain candidates to focus heavily on specific states. Interestingly, the Republican primary visualization revealed a clustering in which individual states "point clouds" occupied more distinct regimes compared to the Democratic visualization clusters, but still less distinct than than the regimes apparent in the general election visualization. The higher degree of distinctness (and the lesser degree of mixing across different states) for the Republican primary in comparison to the Democratic primary may suggest that either the candidates are more diverse, or that certain states are more well unique preferences.

Implementing SVD:

We trained on data from 2000 to 2012. We then simulated the election in 2016 in the order that states vote in the primaries. We used the results of the simulated ongoing 2016 election to fill in the predictions for counties that have not yet voted yet. We initially chose to approximate the matrix by a rank 2 matrix, based on the decay of singular values from our visualization. We also chose standard values for regularization (0.6) and the learning rate (0.2). Both these values are unitless. This gave us poor results, where we were only able to predict the winner of a county about half the time. We realized we needed to validate the results of our training.

We partitioned the dataset into training and testing splits. For this, we experimented initially with a K-fold cross-validation procedure with 3 splits. Cross validation is a resampling procedure that partitions the data in several different splits, and evaluates performance on each split. But for a better control, we can also instantiate a cross-validation iterator, and make predictions over each split using the split() method of the iterator, and the test() method of the algorithm. These are functions that are built into the Surprise library.

In order to optimize the results of training, we implemented hyperparameter tuning. SVD algorithms like those implemented in the Surprise Library have a wide range of possible values for hyper parameters like the number of epochs of training, the type of regularization, and the learning rate of the algorithm. To search the space of possible hyper-parameter sets, we used a method inspired by the Grid Search, which exhaustively

tries every combination of hyper-parameters within set bounds. We optimized this grid search by training and validating each hyper-parameter set with the cross validation procedure described earlier, in order to avoid overfitting. Once we were able to somewhat optimize the hyper-parameters and achieve reasonably good validation scores, we were ready to train on the entire data set.

Reducing Sparsity:

Matrix completion algorithms in general perform worse the more sparse the matrix is. Since many smaller candidates dropped out mid race, our matrix was extremely sparse. We therefore trimmed our matrix so that it would only include the major candidates from the years 2000 - 2012. Of course in any real election you would not be able to tell who will be the major candidates beforehand, so we were forced to keep all the candidates in the 2016 election. This did not end up affecting our algorithm too much because we trained on states in order that they vote in the 2016 election, so even the small candidates who dropped out later still had training data for the specific counties we were training on. For 2000 to 2012 the politicians we kept were Albert Gore, Jr., Bill Bradley, John Kerry, John Edwards, Howard Dean, Wesley Clark, Barack Obama, Hillary Clinton, and Bernie Sanders.

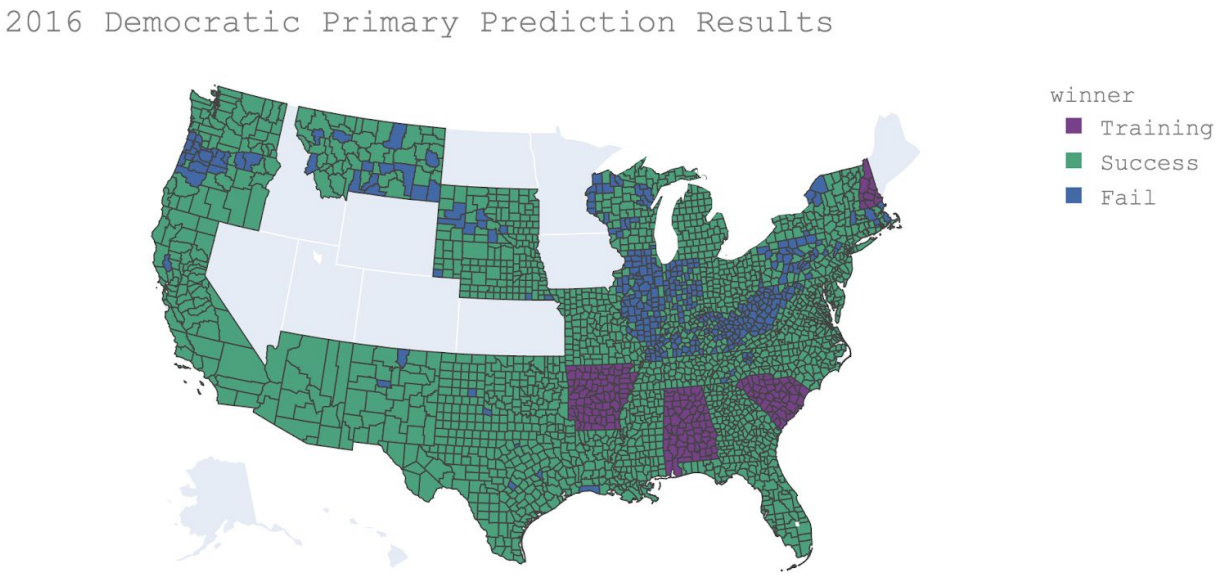
Reformatting Data:

We attempted several methods of reformatting the data. We wanted to reformat the data to reduce the overall variance of the matrix entries. We thought the percent votes that each candidate gets per county might be too noisy. We tried three methods of formatting the data. First we tried rounding the data to the nearest ten's place, so a value of 67% of the votes

would become 70%. This method did not significantly change the performance of our algorithm. We then tried a one-hot encoding scheme where the winner of a county would be labeled "1" and the rest of the candidates would be given zeros. The performance of this method was similar to our final method of reformatting the data, in which we gave each candidate a rank from 1 to n, where n is the number of candidates participating in that election. Both of these schemes gave drastically lower results. We were predicting around 30% of the counties correctly (we suppose this is the same as predicting 70% of the counties correctly, but we discarded this method since it was extremely stochastic, producing a 30% success rate on some runs and a 50% success rate on others). We think the ranked-encoding did not work so well because different counties would have a different upper limit of rank since candidates were dropping out. The upper limit of the rank was also not consistent year to year. In the end we discovered that the best form of encoding was merely to leave the labels untouched.

Results:

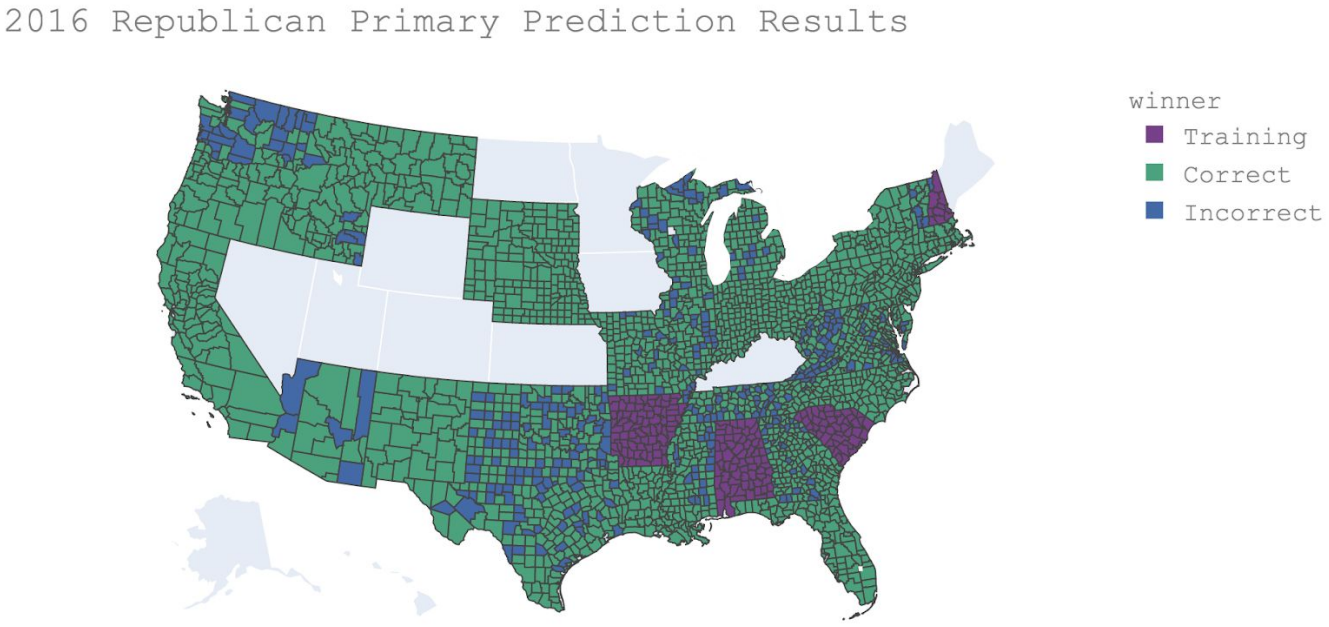
We predicted the results of the simulated 2016 election for both the democratic and republican primaries. Below are plots of the accuracy of our best algorithm.



The counties are individually colored. The gray states are sections of missing data. The purple counties are those which were used for training in the 2016 election. The green counties are counties for which we successfully predicted who would win the election in that county. The blue counties are counties for which we were incorrect. The optimal parameters were: n_factors=3, reg_all = 0.6, lr_all = 0.001, n_epochs=10. Interestingly the best approximate rank of the matrix was one more than we predicted. It is still incredible how low the best rank of the matrix was. This means that our visualization efforts have merit in that the rapid decay of singular values and clustering we see are indicative of the similar latent features each county possesses.

The best success rate for the democratic primary was 85.7% as shown above. This value was sadly somewhat stochastic, ranking from 68.2% to 85.7%. This most likely means that our loss function has multiple local minima, and during our stochastic gradient descent, our algorithm is getting stuck in one of the local minima. This may be possible to fix in the future by changing the loss function to better suit our application. We could reduce the amount of terms in the loss function by eliminating those that are irrelevant. For example, the county bias term is a term that is provided by the Surprise package, but is not relevant for our application. We would have to code the algorithm from scratch in the future if we wished to make our own custom loss function.

We noticed similar stochasticity with the repulblican primary elections as well. The results from our algorithm applied to the 2016 republican primaries are shown below.



Our best results here were an 82.4% success rate, but we were achieving values as low as 59% at some points.

Interesting Points about Results:

Examining the counties which we predicted incorrectly reveals that for the Democratic primary we were mostly incorrect in states that Bernie Sanders ended up winning, and for the Republican primary were mostly incorrect in states that Ted Cruz ended up winning. For the Democratic primaries it was a hit or miss between states. We correctly predicted that Bernie would win Washington and Vermont, but we failed to predict that he would win West Virginia. In all three of those states, our county level predictions were almost 100%, even predicting the specific counties that Hillary would win in states that Bernie won overall.

The story was different for the Republican primaries in which failed to predict that Ted Cruz would win Texas, but our county level predictions were not 100% incorrect. We instead correctly predicted half the counties in Texas. This comparison is strikingly clear by quickly examining both of the above graphs. We think this has to do with the bias term we used. Our algorithm may have been biased for Hillary or Trump winning their respective elections, thus making us fail sometimes in predicting when Bernie or Ted would win.

Finally, even though the results above seem pretty amazing, the actual values we predict for the percent of votes each county will give to the candidates was often quite off. In some cases a county would vote 55% for one candidate and 45% for the other, but our algorithm would predict a ratio of 70% to 30%. In the above graphs “correct” simply means that we correctly predicted the winner of each county. We believe we were not able to achieve more accurate results for the specific votes each county would give because we only trained on three election cycles, one of which wasn’t really a true election (like when Obama ran for reelection in 2012 he was basically uncontested).

We believe that with more election data SVD may be able to predict votes more accurately, but merely predicting which candidate will win each county with the decent success rate we see above is a good step in that direction.

Advantages of SVD

Currently, many election forecasting models use a combination of polling and regression. According to fivethirtyeight.com[1], a widely trusted election forecasting company, their latest model, developed for the most recent Democratic primary earlier this year, attempts to uncover relationships between a state's voting results and their demographic and geographic features, then combines these results with polling results to predict the results for certain states in advance. They perform numerous regressions on different combinations of these features in order to capture more possibilities, but even so, the issue with a simple regression is that there may be many additional plausible features beyond the limited set of demographic and geographic features considered that are harder to quantify or test due to a lack of data, which would cause a regression model attempting to incorporate them to overfit on the data. Moreover, it is hard to make county-specific predictions with this method of regression and polling, as the data for county specific demographic and geographic data may be unreliable, or may not adequately capture more county specific differences, which are often the result of more subtle features that are even harder to quantify.

However, with SVD, we are training on the highest level of voting preference, which captures the expression of all the possible features that may determine a county's voting results. Therefore, to some degree, SVD implicitly considers the effects of all the features that a regression may not consider. But perhaps the main advantage of SVD is that it can

learn from historical patterns by performing a general analysis on the correlation of voting results by county, while a regression is limited to a specific year, as it only attempts to find a direct relationship between its defined feature space and voting results in the current year.

Anomaly and Fraud Detection:

We have also begun using collaborative filtering for election anomaly and fraud detection. At first it seemed like collaborative filtering was not suitable for anomaly detection. Collaborative filtering works well for predicting how certain people will rate variables by looking at their past rating history and comparing it with the rating history of others who voted for the variable of interest. Fraud detection seemed best determined by a model that could learn to find anomalies in data, like a random forest.

After much thought we came up with our own model that uses SVD for fraud detection. The input data is like that in Dr. Alvarez's paper, *Election forensics: Using machine learning and synthetic data for possible election anomaly detection*. We would have information on the percent of votes each candidate received from each county, the number of invalid/blank votes, and the number of eligible voters. Each $X_{i,j}$ of our matrix is then a vector of this information, so our matrix X becomes a tensor. We train our model using data from a previous year's election. We simulate the election by gradually incorporating more and more rows into our tensor. This simulates the counties voting for candidates in the election. After a certain threshold, we run SVD on our tensor to predict how the rest of the counties will vote.

Next we create a synthetic data set, similar to how Dr. Alvarez created his data. In the synthetic data we use our a priori knowledge to generate synthetic examples of both ballot-box stuffing (BBS) and vote stealing (VS) in select counties. We then create errors between our predictions and the synthetic data subject to voter fraud. We learn to classify errors with a label: clean, BBS, VS through a support vector machine.

The result is a classification model for any future iteration of our collaborative filtering system. For any future election, we run our SVD to predict how counties will vote for candidates. We then use the support vector machine (built from analysing the errors between our SVD and the synthetic data) to classify counties as clean, affected by BBS, or affected by VS. We would also be able to determine the extent of voter fraud from the magnitude of error. We can use the same model to detect anomalies in past elections simply by simulating the election as described above, and using the support vector machine to classify counties according to fraud.

Future:

Before we can try anomaly detection our SVD model needs to work with relatively high accuracy. If it works well, then applying it to anomaly detection will be somewhat straightforward, but also somewhat tricky. The most difficult thing we believe will be properly creating the synthetic data. We also think that turning SVD, a matrix decomposition, into a tensor decomposition might be difficult. There do not seem to be any libraries for sparse tensor decomposition for machine learning in the way we described above, but Shiva has experience coding this from scratch. Shiva has worked on tensor

decompositions for simulating lowly entangled quantum systems on a classical computer, so hopefully we can code the model we described from scratch if needed.

First though, we must gain an accurate prediction for SVD. This in itself was all we set out to originally do, but the anomaly detection would be a great bonus. We think we can get some results on our SVD predictions, as we have a good handle on using SVD for the general election data. After we overcome our issues with our parser, we should be seeing some results soon after.

Bibliography:

1. https://fivethirtyeight.com/features/how-fivethirtyeight-2020-primary-model-works/