Result Prediction of Wikipedia Administrator Elections based on Network Features

Nikhil Desai, Raymond Liu, Catherine Mullings

12 December 2014

Abstract

The collaborative encyclopedia Wikipedia has a small set of moderators, known as "administrators," who are elected via a public voting and discussion process called a request for adminship (RfA). RfAs are subjective; they have no vote quotas and are closed and "evaluated" by high-level Wikipedians. In this paper, we attempt to predict results of Wikipedia RfAs using historical voting data and the social network structure of the Wikipedia community. Using votes as representative of voter-on-candidate sentiment, we search for a small subset of "influential" voters whose sentiments might serve as bellwethers on a candidate's electability. To do this, we treat prediction as a binary classification problem, utilize feature selection algorithms such as recursive feature pruning and L1-penalized regression, and apply standard classification algorithms such as Naive Bayes, logistic regression, and support vector classification. From a dataset with 2,761 elections and 6,210 voters, we generated subsets of fewer than 40 voters whose votes yielded 95% accuracy in predicting elections. We also calculated structural metrics on the Wikipedia talk-page communication graph to potentially boost predictive accuracy on these reduced vote sets.

Introduction

Wikipedia is an open-source encyclopedia that can be edited by anyone. However, certain activities on Wikipedia, such as blocking users or deleting pages, are restricted to a small group of trusted users known as "administrators" (or "admins"). Admins are elected through non-anonymous elections in which any registered user may choose to express positive, neutral, or negative sentiments (both in vote and in comment) about a particular candidate.

Voting is not strictly a numbers game; there are no quotas for any particular voting category, and the result of the election is ultimately at the discretion of the bureaucrat (an even higher class of editor) who closes the election. The rationale and identity of the voter behind the vote may therefore be more important than the vote itself, and in practice a large proportion of the votes (perhaps around 80%) must be positive in order for the candidate to be guaranteed adminship.

We hypothesized that there were two main factors that influence the outcome of a Wikipedia admin election: first, how intrinsically "qualified" a candidate running for election is for the Wikipedia admin role, and second, the "influence" of the voters who voted for the candidate. In this paper, we focus primarily on the latter, using machine learning algorithms to identify a potential subset of the most influential voters. As elections are binary classification problems (the candidate is either elected or is not), we initially used Naive Bayes to determine voters' ability to predict election outcomes. We employed dimension reduction / feature selection techniques -- logistic regression and variations of support vector machines -- to detect the most influential voters among the 6,210 voters in the Wikipedia community. We additionally attempted to use unsupervised learning techniques like PCA and matrix factorization to search for hidden categories of voters, but did not obtain meaningful results. Nevertheless, we were able to identify subsets of fewer than forty voters on whose votes we could predict elections with almost 95% accuracy - compared to 98% when using all 6,210.

We did also consider the former hypothesis, attempting to approximate "reputation" in the social graph via various vertex-centric structural metrics. We attempted to use this "auxiliary" dataset both in combination with reduced-dimension voting features and in isolation. We found little to no improved accuracy when we combined features, but reported approximately 65% accuracy when learning on the auxiliary dataset independently.

Sources of Data

We obtained our underlying datasets from the Stanford SNAP network data repository. In particular, we used SNAP's *wiki-Elec* dataset [1], which lists every vote made in a Wikipedia RfA before January 2008. Votes are listed as +1, 0, or -1 depending on whether they were marked "Support," "Neutral," or "Oppose", along with the eventual result of the RfA, the username and id of the candidate, the username and id of the voter, and the time the vote was cast. We noted that some candidates stood for election multiple times, and some voters thus voted multiple times on a candidate (even during the same election), introducing some noise. In total, the dataset contained 2,761 candidacies and 6,210 voters.

We initially obtained the Wikipedia talk-page social graph from the *wiki-Talk* [7] dataset provided on the SNAP website. However, we quickly realized that this dataset did not provide metadata such as usernames, and vertices in the graph could not be identified with users in the voting network. We thus created our own version of the talk-page social graph manually, using the "User talk" segment of the *wiki-meta* dataset [2,3,6], culled from the 3TB January 2008 Wikipedia data dump. This dataset approximated the structure of the Wikipedia community by drawing an edge between user A and user B if A edited the talk page of B. We noted that due to the complexity of the Wikipedia database, some usernames had multiple IDs associated with them, and some IDs corresponded to multiple usernames (as in the case of a name change). After processing, we obtained a directed graph with 257,516 vertices and 6,335,333 edges.

Features and Preprocessing

We treated the *wiki-Elec* dataset as an adjacency list of the bipartite voting graph, with disjoint node sets representing voters and "candidacies" (a tuple of candidate username and election date). Edges link voters to candidacies and carry a tag representing vote sentiment (+1, 0, or -1). We used a table pivoting process to turn this list into a bipartite adjacency matrix, with rows representing candidacies and columns representing voters' decisions (with "holes" where a voter did not vote on a candidacy). For simplicity, if a voter did not vote on a candidate, we assumed they held a neutral sentiment towards that candidate.

The resulting matrix contained 2,761 samples $x^{(i)}$, with each sample defined as a 6,210-dimensional vector such that $x^{(i)}_j$ is the vote voter *j* cast in candidacy *i*. Each vector is consequently very high-dimensional, but as any given user does not vote in every election, ends up being fairly sparse. Each vector comes with a label $y^{(i)}$, the eventual outcome of the candidacy (1 for success, 0 for failure).

In our second dataset, each sample vector $x^{(i)}$ is a low-dimensional vector capturing the values of several different node-centric graph structural metrics. These metrics include in-/out-/total degree, PageRank value, HITS scores, and triangle count. We picked these as they were well-known metrics with importance in network analysis and were implemented out of the box in large graph analysis toolkits [5].

Models

Baseline

We should hope that the "influential" voters we select give better predictive power than a random subset. As a baseline, we picked one hundred (1.5% of total) random voters and trained classifiers on them.

Voter "quality" in predicting outcomes

We first attempted to predict outcomes from the complete voting dataset we generated. To improve the accuracy of a straightforward classifier, and to ensure the discrete features presented to a Naive Bayes implementation would be nonnegative integers, we decided to weight voter sentiment by a voter's "accuracy" in predicting candidate outcomes. Given that elections are binary classification problems, we felt employing a Multinomial Naive Bayes classifier would be a reasonable start.

- **Multinomial Naive Bayes:** To predict the outcome $y_k \in \{0, 1\}$ of the *k*th candidate Y_k, we calculated:

$$P(Y = y_k \mid \vec{V}) = \arg\max_{y_k} P(Y = y_k) \prod_{i=1}^{m} P(V_i = v_i \mid Y = y_k)$$

  where
  (since the numerator of this probability has multiple outcomes, we use multinomial naive bayes) and

$$P(V_i = v_i \mid Y = y_k) = \frac{P(Y = y_k, V_i = v_i)}{P(V_i = v_i)} = \frac{\#\{\text{times } V_i \text{ voted } v_i \text{ in an election with outcome } y_k\}}{\#\{\text{elections with outcome } y_k\}}$$

  $v_i = \{-1,\ 0,\ 1\}$ is the vote of the ith voter V_i; $i \in \{1 \ldots m\}$ and *m* is the number of voters who voted for candidate Y_k.

- **SVM with Linear kernel:** As a reference point for the performance of our Naive Bayes classifier, we chose to train a support vector classifier on the same data. We used a linear kernel for the underlying SVM since it would be faster to train and test on a large feature set. We found that our Naive Bayes classifier performed better with respect to test error.

Feature Selection

To determine the most "consequential" voters in the voting network, we need to find a subset of voters whose collective sentiment yields a predictive accuracy almost equivalent to that reached on the entire dataset. Since each feature in our dataset represents the sentiments of a single voter, finding this subset would be equivalent to reducing the dimensionality of our prediction problem. We thus decided to indirectly obtain the "most influential" voters by using several techniques for feature selection. We picked the following methods because they offered the possibility of yielding a list of significant features (and thus influential voters) and were readily available in our machine learning package of choice.

Univariate methods. As a naive first approach, we tried selecting the features that scored highly on a statistical test of correlation with election outcomes; we chose our test to be ANOVA (analysis of variance), in which the variance of the feature within each label is compared to the overall variance of the feature value (the smaller their ratio, the more likely there is a correlation). We chose a *k* and selected the top *k* features. We did not expect this method to yield highly accurate results, as it does not take into account correlations between different features, and thus may yield groups of features highly correlated with each other, and thus providing redundant information.

L1-penalty ("lasso") methods. We then attempted to select features by searching for linear classifiers that incorporated only small subsets of features in their models. To do this, we induced an *L1 regularization penalty* on various linear models: this adds a penalty of

$$\|\theta\|_1 = |\theta_0| + |\theta_1| + \ldots + |\theta_n|$$

(the "L1 norm" of the hypothesis vector), rather than the standard Euclidean or L2 norm, to the objective function of the underlying optimization model. Due to the convexity of our underlying problem, this penalty will ensure sparseness. We used two standard linear classification models for the L1 feature selection - linear regression and support vector classification with a linear kernel.

Recursive feature elimination. As a final method of feature selection, we chose a standard linear classifier, trained it on the full feature set, identified the component with the lowest model coefficient, and eliminated it from the feature set. We chose a *k* and repeated this process until we were left with *k* features. Unlike L1-regularized methods, this method is more amenable to cross-validation, and thus it could be used to find the "optimal" number of features for a given classifier.

Results

Random Feature Selection (averaged over 20 runs)

100 Features (random)	Avg. Test Error	Min. Test Error	Avg. Train Error	Min. Train Error
SVM (RBF)	0.3587	0.2731	0.3469	0.2649
SVM (Linear)	0.2822	0.2222	0.3030	0.2434

All Voters, weighted by predictive power

6,210 Features	Avg. Test Error	Min. Test Error	Avg. Train Error	Min. Train Error
Multinomial N.B.	0.0272	0.000	0.0215	0.0167
SVM (Linear)	0.0369	0.000	0.000	0.000

Feature-Selection (Univariate ANOVA) on Voters

40 Features	Avg. Test Error	Min. Test Error	Avg. Train Error	Min. Train Error
Logistic Reg.	0.0756	0.0612	0.1002	0.0435

SVM (RBF)	**0.0696**	0.0592	**0.0973**	0.04348
SVM (Linear)	0.0761	0.0628	0.1021	**0.0399**

18 Features	**Avg. Test Error**	**Min. Test Error**	**Avg. Train Error**	**Min. Train Error**
Logistic Reg.	0.1265	0.0974	0.1328	**0.0399**
SVM (RBF)	**0.1041**	**0.0918**	**0.1303**	0.0435
SVM (Linear)	0.1278	0.0978	0.1354	**0.0399**

Feature-Selection (Linear SVC with L1 Penalty) on Voters

38 Features	**Avg. Test Error**	**Min. Test Error**	**Avg. Train Error**	**Min. Train Error**
Logistic Reg.	0.0595	**0.0507**	0.0742	**0.0362**
SVM (RBF)	**0.0581**	**0.0507**	0.0756	**0.0362**
SVM (Linear)	0.0600	0.0978	**0.0731**	**0.0362**

Feature-Selection (Logistic Regression with L1 Penalty) on Voters

7 Features	**Avg. Test Error**	**Min. Test Error**	**Avg. Train Error**	**Min. Train Error**
Logistic Reg.	0.1419	**0.1163**	**0.1470**	0.0616
SVM (RBF)	**0.1355**	0.1180	0.1473	**0.0543**
SVM (Linear)	0.1454	0.1211	0.1477	0.0616

Conclusions and Discussion

As our results indicate, using users' past votes as features and feature selection to ascertain "influential" voters can yield very accurate election predictions. Using our modified Multinomial Naive Bayes on all the voters, we were able to obtain around 97.3% accuracy; in some cases we had 100% accuracy, Interestingly, the linear-kernel SVM on all voters performed slightly worse, despite the large number of training examples provided. This may be due to the fact that we have many more features than training examples, or due to the format of the aggregate counts, which are naturally well-suited to Naive Bayes.

For all types of feature selection implemented (best results bolded), RBF-kernel SVMs resulted in the lowest average test error. L1-penalized linear-kernel SVM yields better results than univariate feature selection with ANOVA on the dataset, as we obtained a higher accuracy with 38 features selected via the former than with 40 features selected via the latter. In general, the results were surprisingly accurate given that 40 features, the highest number used in our feature selection results, is only around 0.5% of the total number of features. Especially surprising is our L1-Penalty Logistic Regression feature selection; we were able to get an accuracy of around 86.5% using only 7 voters in conjunction with an RBF-kernel SVM.

Recursive feature elimination (data not shown) proved ineffective compared to either approach. When run using cross-validation to optimize the number of features for both accuracy and size, we found models trained on the resulting dataset to average an error of approximately 15%. The "optimal" number of features computed by this approach was 18; however, using ANOVA to naively select 18 features yielded higher accuracy than using RFE.

While vote metrics have reasonable predictive accuracy, they ideally could be higher given that we are currently running our models over existing data and that elections are a binary classification problem. As mentioned in the "Data" section, noisiness in the data may have impacted the accuracy of our predictions.

Leveraging Network Features

As noted previously, we also attempted to improve our model by incorporating structural information about the Wikipedia social graph. We computed eight metrics: indegree, outdegree, total degree, PageRank value, vertex triangle count, size of connected component, HITS authority score, and HITS hub score. We trained three standard classifiers on this dataset in isolation, then attempted to concatenate it to the smallest (7-feature) voter dataset to see whether it would boost classification accuracy.

Classification Using Graph Structural Metrics

8 Features	Avg. Test Error	Min. Test Error	Avg. Train Error	Min. Train Error
Logistic Reg.	0.3805	0.3775	0.3864	0.2717
SVM (RBF)	0.4455	0.4262	0.4483	0.2862
SVM (Linear)	0.3494	0.3426	0.3556	0.2717

Training on graph metric values in isolation yielded a best-case average error of 35%; while significant, this is significantly lower than in any of our voter-based models. Combining graph metrics with the smallest voter set, meanwhile, did not result in any noticeable increase in accuracy (data not shown).

There are many potential reasons for this. First, we had very few graph metric features. We lacked the time to compute more useful network statistics, such as betweenness-centrality or Girvan-Newman communities, whose computation was cubic or worse in the graph size. Additionally, our graph represents a static snapshot of the Wikipedia talk network at a particular time, but we used it to predict elections at various previous points in time. This leads to several confounding variables; for example, some users with high HITS authority scores may have gained that score due to their actions and influence as admins, or even from notoriety gained from their candidacy. Furthermore, any user who ran unsuccessfully before a later successful candidacy will confound the model, since the two points will have the same coordinates but different labels. There are also more fundamental possibilities for error: our approximation may not be indicative of the overall Wikipedia community structure, our metrics may not be accurate predictors of influence, or the network structure may have little to do with election results.

Future

Given more time, we would first collect more data; we currently have vastly more features than training examples. Ideally, we would build an exhaustive, self-updating community graph using all Wikipedia metadata since conception, resolve all username/ID ambiguities, and recompute graph metrics (and perhaps generate additional ones), but even loading this graph would require supercomputer or cluster access (until 2008, the metadata is already >3 terabytes).

We would like to improve upon our models which use past voter data by allowing them to store data from the results of recently-closed elections; this lends itself to an unsupervised learning model where the value function is whether the prediction was correct. We might even be able to incorporate temporal components into this model, or our community graph metrics by generating new graphs/metrics with upper and lower bounds on time. Leveraging previous work by Leskovec et al. that attempts to predict single-voter behavior in the Wikipedia RfA network [2], we may even be able to dispense with directly measuring voter data and instead use such an approximation to generate voter features.

In addition, we are also considering the implementation of decision-tree models that would allow us to better choose which features are most relevant for any particular election. We are also considering ensemble learning techniques that would allow us to model more robustly features from different sources that may have different distributions (for us, graph metrics and vote counts), or perhaps combine the two (as through random forests).

References

[1] J. Leskovec *et. al.* (2010). *Wikipedia adminship election data* [Online]. Available: http://snap.stanford.edu/data/wiki-Elec.html
[2] J. Leskovec *et. al.* (2010). *Governance in Social Media: A case study of the Wikipedia promotion process* [Online]. Available: http://cs.stanford.edu/people/jure/pubs/voting-icwsm10.pdf
[3] J. Leskovec *et. al.* (2010). *Signed Networks in Social Media* [Online]. Available: http://cs.stanford.edu/people/jure/pubs/triads-chi10.pdf
[4] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.
[5] J. Leskovec, R. Sosi (2014). *Snap.py: SNAP for Python, a general purpose network analysis and graph mining tool in Python* [Online]. Available: http://snap.stanford.edu/snappy
[6] G. Kossinets. *Processed Wikipedia Edit History.* [Online]. Available: http://snap.stanford.edu/data/wiki-meta.html
[7] J. Leskovec *et. al.* (2010). *Wikipedia talk network graph.* [Online]. Available: http://snap.stanford.edu/data/wiki-Talk.html