Empirical and Computational Methods for Electoral Politics

Benjamin Haber Fifield

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Politics
Adviser: Kosuke Imai

January 2019

Abstract

This collection of three essays introduces new computational and empirical methods for data analysis and validation in political science, with specific applications to the practice and study of electoral politics.

In the first chapter, I introduce a set of graphical diagnostics that can be used to improve statistical models of heterogeneous treatment effects. I adapt the uplift curve, which has been used previously to visualize cumulative benefits from targeting treatments, and repurpose it as a tool for model selection to make model-building workflows for predicting treatment effects less ad-hoc. I also present a stack-ranking graphical diagnostic for heterogeneous treatment effect models to help visualize model fit, and I apply both diagnostics to a canonical experiment on voter mobilization using social pressure appeals. The chapter advances a more principled model-building approach for predicting individual-level treatment effects.

The second chapter builds on recent computational and simulation-based methods for studying the effects and consequences of congressional redistricting. As ensembles of simulated redistricting plans become more common in the academic literature and as legal evidence, validating the accuracy and representativeness of those algorithms has become increasingly important. This paper makes two advances — first, it applies a recently-developed enumeration algorithm to increase the size and complexity of simulation validation datasets, and second, it introduces a new validation diagnostic that can be used to ensure redistricting simulators are accurately representing the target distribution of plans. I then apply the new validation test to two competing redistricting simulation methods.

The third chapter (adapted from work coauthored with Ted Enamorado and Kosuke Imai) is a practical guide to data merging and record linkage using fastLink, a new open-source implementation of the canonical Fellegi-Sunter probabilistic record linkage model. I briefly review fastLink and the computational improvements it im-

plements, and I then walk through several applied data examples using the software that illustrate its preprocessing, merging, and inspection functionalities. Finally, I apply the entire proposed workflow and software to a validation exercise using data on local-level politicians in Rio de Janeiro, Brazil, where I use fastLink to analyze rates of party switching across election cycles.

Acknowledgements

This dissertation, at many points, looked like it would never be completed. I still don't quite believe that it actually exists, and I'm incredibly grateful for the people in my life who have supported me along the way.

I owe Kosuke Imai immeasurably for his mentorship throughout my time at Princeton. It's fair to say that my life changed when Kosuke asked me to precept Quant II back in Spring 2013. I'm consistently in awe of his devotion to his students, his extraordinary collaborative drive, and his incredible optimism and excitement for research. I've learned so much from Kosuke about how to ask questions, how to think about evidence, how to be a better mentor, and how to demand the best from myself. It's also due to Kosuke that I had any faith in my ability to return to Princeton after a nearly two-year absence and actually finish a dissertation. I am incredibly proud to call myself his student, and I'll always be grateful to him for what I've learned and the opportunities I've had that are a direct result of his mentorship.

I've experienced more dramatic highs and lows alongside David Nickerson than any advisor should have to deal with in a student. Most of these moments came long before he agreed to join my dissertation committee — during my time at HFA, his rigor, thoughtfulness, depth of knowledge, and humor motivated my work and guided me through major professional and personal decisions. He was an incredible resource when I was deciding to return to graduate school, and he has given me consistently excellent advice on a project that is frequently difficult to sell to political scientists. I'm lucky to call him both an advisor and a friend.

I'm also grateful for Nolan McCarty for his steadfast support throughout my time at Princeton. Nolan has seen me whiplash through dissertation topics, enrollment statuses, and employers, and he has always been there without judgement to read anything I sent his way, no matter how long it had been since we had last talked.

One of my biggest mistakes at Princeton was not asking for his advice more often, but I am incredibly thankful for his patience and thoughtfulness throughout.

While Chuck Cameron is no longer on my committee, I owe him a special thanks for being a consistent source of support throughout my health issues in 2015. I always appreciated his insistence on asking how I was doing before diving into real work, and for his excitement whenever I had good news on successful surgeries and treatments.

To every one of my friends at Princeton — Kabir, Katie, Marcus, Liz, Rebecca, Ted, Yuki, Winston, Asya, Chantal, Adam, Scott, Carlos, Xander, Ryan, Wouter, Christoph, Joan, Naoki, James, Steven, Mike, and many more who I apologize for leaving off — you made graduate school a warmer, more fun, and more supportive place than I ever expected. I'm amazed thinking about how many lifelong friends I've met here, and how many huge milestones we've celebrated together. And to Jacob, Gordon, Zack, Mark and Christian — thank you for always being there to get me out of Princeton when I needed it most, for running up to Boston and Vermont together for weekends, for always having a couch or floor to crash on, and for being the strongest and longest friendships of my life.

In September 2015, I left grad school to join Hillary Clinton's presidential campaign — easily the best and most consequential decision of my professional life. Thank you to Kit Rodolfa and Elan Kriegel for taking a chance on me back when I had no idea what kind of chaos was going to hit my life when I joined. To all of model-poll — Alissa, Alex, Bill, Hana, Hannes, Hasan, Jason, Meg, and Sarah — I am so grateful that I got to live this alongside all of you and gain so many amazing friendships along the way. You were all a source of refuge, strength, and humor throughout endless week, query costing, the not-always-metaphorical flies, and countless tag-ups, and I am immensely thankful that we continued to be there for each other after it ended. I'll be proud to call myself Dr. Bentern once this dissertation is defended.

To my family — my sister Jocie, and my parents John and Sandi — thank you for being a constant source of support, encouragement, distraction, and warmth throughout my entire life, and particularly throughout the entire graduate school experience. I can't imagine making it to Princeton in the first place without the values of empathy, curiosity, and compassion you made sure were as important to me as they are to you, and I would never have finished without your confidence that I could actually do so. And to my mother-in-law Sue, thank you so much for opening your home and your family to me, for always making sure we were well-fed and well-rested at the most stressful points of graduate school, for loving Charlie unconditionally even at his most obnoxious, and for being an amazing second mom.

Finally, to Yang-Yang — we've seen each other through a campaign, fieldwork, cross-country road trips, sickness and surgeries, beach lounging, at least five different apartments (depending on how you count), a dog-son that at first held us captive for months on end, the job market (as I write this!) and an unbelievable amount of change since Festivus 2014. Throughout everything, you have been the greatest source of stability, humor, wisdom, and love I could ever ask for, and I am so grateful for the strength and confidence you give me every day. I can't wait to see where our life together takes us.

November 16, 2018

Plainsboro, NJ

For my family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Large-scale administrative data and new computational resources have fundamentally changed both the study and practice of electoral politics. For scholars, these new resources have opened up new avenues of study for understanding public opinion and representation in American politics, the dynamics of international organization network formation in international relations, the success or failure of different autocratic censorship strategies in comparative politics, and countless other questions that have furthered our understanding of the political decisions of individuals, elites, and institutions. For political practitioners, these resources have led to an explosion of new voter targeting and communication efforts, with campaigns and party organizations aggressively investing in new data sources and analytical strategies to learn as much as possible about the American voting public. As these new methods and data continue to proliferate and change political science and political practice, validation efforts to ensure their validity and accuracy have become increasingly important. This dissertation develops new methods that lie at the intersection of academic political science and commerical political targeting, and it strives to take this issue of methodological validation and data integrity seriously.

In Chapter 2, "Model Selection and Model Comparison for Predicting Heterogeneous Treatment Effects," I develop a set of model-building heuristics and diagnostics for statistical models of heterogeneous treatment effects (HTE), which are used by scholars to test theories about differential responses to interventions and by practitioners to target political communication to the most responsive voters. Unlike standard predictive models, HTE models are difficult to validate and tune because the outcome of interest is the individual-level treatment effect, which is not observed by the researcher. As a result, standard model-building techniques such as out-of-sample validation on mean-squared error or the ROC curve cannot be directly applied. I adapt an HTE model visualization technique called the uplift curve from the marketing science literature as a metric for model comparison, and I validate it as an informative diagnostic test for model selection, criticism, and tuning. I also introduce a graphical stack-ranking validation plot that can be used to visualize areas of mis-fit along the estimated prediction space. Finally, I apply the new diagnostic tests to build and analyze heterogeneous treatment effects in two canonical studies of social pressure get-out-the-vote mailers, and I find significant predictive overlap between the two experiments. The techniques introduced in this chapter are implemented in the open-source R package, hetlearner.

Chapter 3, "Validating Ensembles of Simulated Redistricting Plans," introduces new validation techniques to ensure the accuracy and representativeness of ensembles of simulated redistricting plans. New computational resources have simultaneously allowed partisan officials to more accurately ensure certain electoral outcomes by redrawing congressional districts, while also giving scholars new tools to study the consequences of this political manipulation. One of these tools, redistricting simulation, leverages these computational resources to draw large numbers of counterfactual redistricting plans that satistfy certain legal and political constraints, and they have become an important class of evidence when redistricting plans are challenged in

court. However, it is unclear whether these simulated redistricting plans are a truly representative sample from the underlying distribution of valid redistricting plans. I propose validating redistricting simulation methods on small test maps sampled from actual states where all possible valid redistricting plans can be accurately enumerated, and I use newly developed methods from the computer science literature to scale up the size and scale of the enumeration datasets. I apply these new validation tests to two prominent classes of redistricting simulators, and give some practical suggestions for how these tests can be integrated into the legal and scholarly analysis of redistricting. These new validation tests are implemented in the open-source R package, redist.

Chapter 4, "fastLink: R Package for Fast Probabilistic Record Linkage," is a practical guide to conducting probabilistic record linkage using the open-source R package fastLink. Political scientists and practitioners frequently merge data sets to answer new questions and more effectively measure traits of the voting public, but these data often do not have a unique identifier such as social security number that can be used for merging. Instead, researchers and practitioners either use ad-hoc exact matching on fields such as first name, last name, address, and birth date that can have high error rates under common scenarios, or they rely on proprietary methods that are expensive and not transparent. fastLink is a fast implementation of the canonical Fellegi-Sunter probabilistic record linkage model that can readily handle misspellings and administrative errors in record linkage, while also scaling to much larger data sets. I briefly review fastLink and the computational improvements it implements, and I then walk through several applied data examples using the software that illustrate its preprocessing, merging, and inspection functionalities. Finally, I apply the proposed workflow and software to a validation exercise using data on local-level politicians in Rio de Janeiro, Brazil, where I use fastLink to analyze rates of party

switching across election cycles. This chapter is adapted in part from coauthored work with Ted Enamorado and Kosuke Imai in Enamorado, Fifield and Imai (2017).

Taken together, these three essays advance the ability of scholars and practitioners to take advantage of exciting new data and computational resources, while also improving the transparency and accuracy of analysis and scholarship.

Chapter 2

Model Selection and Model Comparison for Predicting Heterogeneous Treatment Effects

2.1 Introduction

Heterogeneous treatment effect (HTE) models are powerful tools for effectively targeting new interventions towards the right populations. In political campaigns, only some registered voters will be more likely to turn out to vote after speaking to a campaign canvasser, while only a subset of social media users will be likely to spend more time on platform after being reminded of their online interactions with a close friend. By combining randomized experiments with predictive modeling, analysts can use HTE models to not only target treatments to the most responsive subset of the population, but also avoid delivering new features and interventions to individuals predicted to respond negatively to a change in their environment.

However, unlike standard predictive models where the predictive quantity of interest is a measured and observed outcome, the analyst never observes the individual-level treatment effect for any observation when building HTE models. This makes

standard model selection and model comparison techniques, such as mean squared predictive error or the Area-under-Curve (AUC) metric, impossible to use. As a consequence, there is little agreement on what a model-building and model-selection workflow should look like for HTE models.

This paper aims to provide a framework for model building and model selection for HTE models to aid practitioners in choosing the most appropriate estimator and algorithm for their application. It introduces a set of heuristic graphical diagnostics that can be used to assess model fit for HTE models despite not observing the individual-level treatment effect, as well as an area-under-curve diagnostic that can be used as a model comparison metric. These diagnostics can be applied regardless of the estimating algorithm used, and can be used either to compare across estimating algorithms, or to compare model specifications conditional on the estimating algorithm chosen. I also propose a simple workflow that uses these graphical diagnostics to build and select heterogeneous treatment effect models that accurately predict treatment effects out-of-sample.

Lastly, I apply these estimators to an influential literature on social pressure treatments for encouraging voter turnout, to see if treatment effects predicted on one experimental sample generalize to new experiments. Starting with Gerber, Green and Larimer (2008), an extensive literature has studied treatments designed to place public pressure on the voter to turn out to vote, often using the threat of postcards mailed post-election that make public the voting histories of the voter and their neighbors. I first use HTE models to predict treatment heterogeneity in the original social pressure experiment, which was conducted in a low-salience election in Michigan in 2006. Since then, numerous social pressure Get-Out-the-Vote (GOTV) mailers have been sent and tested across the country and across electoral contexts. I find that in similarly low-salience contexts, the same models predict the patterns of heterogeneity fairly well, and that the estimated treatment effects can help discover the underlying

source of shared heterogeneity. This paper then concludes with a set of recommendations for building and deploying HTE models, either for prediction in industry or for academics characterizing patterns of heterogeneity.

2.2 Heterogeneous Treatment Effect Models in Political Science and Political Campaigns

On political campaigns, HTE models play a specialized role in helping direct mail, digital, in-person, and television persuasion and GOTV efforts. For example, President Obama's 2012 re-election campaign deployed a massive persuasion experiment of over 500,000 voters (Hersh, 2015, 151-153) to gather data for HTE model-building, which was then used to target vote persuasion mail and campaign contact. Although the size and scale of that particular experiment make these models out-of-reach for most campaigns, national political organizations such as the DNC/RNC, DGA/RGA, unions, and advocacy groups can and do made use of HTE models to direct their contact with voters and the broader public.

More generally, this research is directly inspired by the use of Experimentally Informed Programs (EIPs) in political campaigns. As discussed by Kalla and Broockman (2018), an EIP is a small-scale experiment run to identify (1) whether a treatment moves the electorate in the aggregate and (2) whether there is heterogeneity in responsiveness to the treatment that can be used to better target its delivery to the full electorate. In political campaigns, the full EIP modeling flow is:

1. Define treatment (or treatments) of interest for testing
2. Sample small portion of the target voter universe from the voter file
3. Randomly assign treatment to the sample, and attempt to deliver treatment
4. Attempt to survey voters in the sample after receipt of the treatment

5. Build HTE model on survey data linked back up to covariates in voter file

6. Score entire voter file on the model

7. Use model to inform allocation of the treatment to most responsive voters, while suppressing treatment for voters modeled to show backlash effects

In this project, I am particularly concerned with items 5, 6, and 7, although optimal survey design and power analysis for HTEs are both important open questions. In addition, since the final product of the EIP is to score the entire voter file using the estimated model, it is necessary that the score and the observed patterns of heterogeneity have strong external validity.

2.2.1 Review of Heterogeneous Treatment Effect Models

HTE models attempt to estimate the *conditional average treatment effect* (CATE) of a treatment T_i, which here can take the value of either 0 (voter does not receive the treatment) or 1 (voter receives the treatment). The CATE is formally defined as

$$\tau_i = \mathbb{E}(Y_i(T_i = 1) - Y_i(T_i = 0) \mid X_i = x_i)$$

or the expected difference between counterfactual outcomes $Y_i(T_i)$ under treatment and control, conditional on covariates x_i. Importantly, we never observe both counterfactual outcomes for any given unit (Holland, 1986), meaning that the true individual-level treatment effect is never observed for any unit. This makes building a model to predict CATE uniquely difficult. In a standard predictive model, the analyst can use train-test splits to compare out-of-sample predictions of a predicted outcome, $\hat{Y}_i$, against the true outcome Y_i, in order to select the model that maximizes out-of-sample predictive performance. This is not possible in HTE models, given that the predictive target is the difference between counterfactual outcomes for a given unit but only a single counterfactual outcome is observed for each unit.

Although the researcher cannot directly observe τ_i for any individual unit, they can still build statistical models to try and measure it. These often take the form of training some sort of statistical model $f_\tau(\cdot)$ where

$$\begin{aligned} Y_i &= \underbrace{f_\tau(X_i, T_i)}_{\text{Model prediction}} + \underbrace{\epsilon_i}_{\text{Model error}} \\ \widehat{Y_i} &= f_\tau(X_i, T_i) \end{aligned}$$

The statistical model $f_\tau(\cdot)$ can either be a single model or a series of models, which I discuss in further detail in Section 2.2.2. We can then generate an estimate of the CATE, $\widehat{\tau_i}$, as

$$\begin{aligned} \widehat{\tau_i} &= \widehat{Y_{i|T_i=1}} - \widehat{Y_{i|T_i=0}} \\ &= f_\tau(X_i = x_i, T_i = 1) - f_\tau(X_i = x_i, T_i = 0) \end{aligned}$$

which is simply the difference in the predicted outcomes for unit i when setting treatment status to 1 versus setting treatment status to 0, conditional on observed covariates x_i.

2.2.2 Status Quo Methods for Building and Deploying Heterogeneous Treatment Effect Models

To date, most research into HTE's has focused on deriving algorithms for estimating $\widehat{\tau_i}$ and $f_\tau(\cdot)$. A non-exhaustive list of studies introducing new CATE estimators can be found in Table 2.1, which roughly divides proposed estimators into four categories. Tree models rely on recursive partitioning methods to find subgroups with maximally heterogeneous treatment effect estimates, while sparse models use either Bayesian or non-Bayesian methods coupled with deep interactions between treatment indicators and pre-treatment covariates along with variable selection techniques.

Ensemble methods follow the *super learner* framework introduced in van der Laan, Polley and Hubbard (2007) for prediction problems, where the researcher constructs many different models and then weights their contribution to the final prediction by their out-of-sample predictive error relative to the observed outcome. Instead of applying these weights to multiple predictions of the outcome, ensemble methods apply the weights to multiple predictions of the CATE from different models. Lastly, Bayesian models for HTE's is somewhat of a catch-all — Lam (2013) derives a Gibbs sampler that incorporates imputation of the missing counterfactual outcomes into estimation, while Shiraito (2016) places HTE estimation in a non-parametric Bayesian framework by utilizing Dirichlet process priors and making latent group membership a function of pre-treatment covariates.

Class of Model	**Relevant Estimators and Algorithms**
Tree Models	Zeileis, Hothorn and Hornik (2008); Hill (2011); Imai and Strauss (2011); Kern and Green (2012); Athey and Imbens (2016); Athey, Tibshirani and Wager (2017); Athey and Wager (Forthcoming); Kunzel et al. (2018)
Sparse Models	Imai and Ratkovic (2013); Ratkovic and Tingley (2017, 2018)
Ensembles	Grimmer, Messing and Westwood (2017); Samii, Paler and Daly (2017)
Bayesian Models	Lam (2013); Shiraito (2016)

Table 2.1: Prior literature focusing on deriving estimators for conditional average treatment effect models.

Kunzel et al. (2018) take a step back from the estimator-deriving literature to summarize existing HTE models in terms of four related *meta-learners*, whose categorization I follow for the remainder of the paper. The four meta-learners can be applied to any estimating function $f(\cdot)$, where $f(\cdot)$ may be a linear model, GLM's, regularized linear models such as LASSO or Elastic Net, or more non-parametric models such as BART, random forests, or neural networks.[1] The first learner, which they term the *S-learner*, first estimates a single model $f_\tau^S(X, T)$ predicting the observed Y_i

[1]Formal algorithms for each learner can be found in Appendix Section A.1.

where the treatment indicator is included as a standard covariate. Then, predictions are generated for each observations setting T to 1 and to 0 for each observations, so that the final estimate of the individual-level treatment effect is

$$\widehat{\tau_i^S} = \widehat{f_\tau^s(X_i, T_i = 1)} - \widehat{f_\tau^s(X_i, T_i = 0)}$$

The second learner, the *T-learner*, estimates two separate models by modeling the outcome as a function of covariates separately for treatment ($f^T_{\tau|T_i=1}(X_i)$) and control ($f^T_{\tau|T_i=0}(X_i)$) groups. This modeling strategy follows the logic of work such as Bansak (2018), which advocates sample-splitting by treatment group rather than pre-treatment moderating variables in order to estimate unbiased causal moderation effects. The estimate of the CATE is then generated as

$$\widehat{\tau_i^T} = \widehat{f^T_{\tau|T_i=1}(X_i)} - \widehat{f^T_{\tau|T_i=0}(X_i)}$$

The third learner, called the *F-learner*, relies on a transformation of the outcome variable examined in Athey and Imbens (2016) (among others) where $Y_i^* = Y_i \times \frac{T_i - e(X_i)}{e(X_i)(1-e(X_i))}$, and where $e(X_i)$ is an estimate of the propensity score. This transformed outcome can be easily shown to equal τ_i in expectation[2] — however, it is known to be an inefficient estimate of the individual-level treatment effect as it does not use the treatment group information beyond the transformation of the outcome variable. The F-learner algorithm is straightforward — a model $f_\tau^F(\cdot)$ is trained against the transformed outcome Y_i^*, and then the resulting estimate of the CATE is

$$\widehat{\tau_i^F} = \widehat{f_\tau^F(X_i)}$$

[2]This is shown in Appendix Section A.1.1

Last, Kunzel et al. (2018) introduce the *X-learner*, which uses two separate stages of estimation. The first stage follows the T-learner by predicting the outcome as a function of covariates for the treatment and control groups separately. Then, outcomes are predicted for the treated observations using the control group model and for the control observations using the treatment group model, and residuals η_i are calculated. Finally, the residuals are predicted separately for each treatment group ($f^X_{\tau|T_i=1}(X_i)$ and $f^X_{\tau|T_i=0}(X_i)$) as a function of covariates. The final prediction is then

$$\widehat{\tau_i^X} = e(X_i)\widehat{f^X_{\tau|T_i=1}(X_i)} + (1 - e(X_i))\widehat{f^X_{\tau|T_i=0}(X_i)}$$

where $e(X_i)$ is an estimate of the propensity score.

Despite the growing number of estimation methods for HTEs in the existing literature, little attention has been paid to the problem of model comparison and model selection for them. Unlike standard predictive models, there is no standard metric for evaluating the quality of out-of-sample predictions from competing HTE estimators and model specifications, given that the predictive quantity of interest is unobserved. Most papers on HTE models instead focus on simulations to show the performance of the proposed method — there, counterfactual outcomes can be carefully generated in line with an exact specification of the data generating process, allowing researchers to observe the individual-level treatment effect and thereby benchmark using standard model fit metrics such as mean-squared error and classification accuracy measures. And while simulations are useful for understanding the performance of a proposed method in controlled conditions, these comparisons provide little guidance to an analyst looking to benchmark performance of multiple estimators on a train-test split of experimental data.

Furthermore, the few papers in the HTE literature that do consider out-of-sample performance as a means to tune the model or perform model selection do so in terms

of out-of-sample performance in predicting Y_i, rather than τ_i. For example, Grimmer, Messing and Westwood (2017) build an ensemble of HTE models such that $\widehat{\tau_i} = \sum_m w_m \widehat{f_\tau^m(X_i)}$, but the weights w_m are selected based on out-of-sample predictive performance on Y_i, rather than the individual-level treatment effect. The work of Athey and coauthors (e.g. Athey and Imbens, 2016; Athey, Tibshirani and Wager, 2017; Athey and Wager, Forthcoming) takes the question of out-of-sample performance more seriously, by formulating a method for explicitly evaluating out-of-bag performance in random forest models on the estimated treatment effects rather than the observed outcome. However, even here, their focus is more on choosing optimal splits given the selected model rather than comparing across sets of candidate models. To fill this gap in the literature, in the following section I introduce a set of graphical diagnostics for understanding HTE model fit and mis-fit, as well as a new model selection criterion to help researchers and analysts adjudicate between candidate sets of HTE models for prediction.

2.3 Diagnostics for HTE Model Selection

In this section, I introduce a new metric for HTE models that can be used for model comparison and selection, along with a graphical diagnostic for examining model fit across all ranges of the estimated treatment effect. Both diagnostics operate by examining the observed Local Average Treatment Effect (LATE) among subsections of the data defined by their estimated treatment effects. While the graphical diagnostic, the "stack-ranking" plot, simply checks that the observed LATE aligns with the average estimated treatment effect across buckets of the estimated score, the proposed model-selection metric, which I call the *Treatment Area-under-Curve* (T-AUC) diagnostic, evaluates the "lift" achieved by using the score versus random targeting at different portions of the estimated treatment response curve. The researcher can then select model specifications that maximize this lift, thereby concentrating the

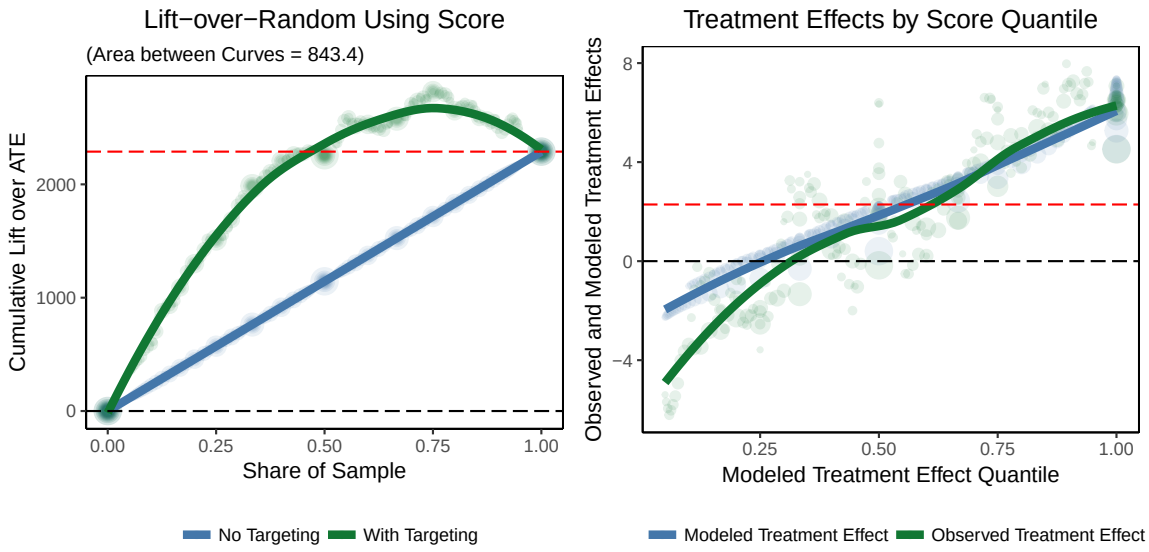

Figure 2.1: Two graphical diagnostics for improving model building of HTE models. The left plot is the Treatment Area-under-Curve (TAUC) diagnostic, which measures the "lift" over random of targeting by the score at different portions of the estimated treatment response scale. The area between the green line (observed lift through targeting by score, LOESS fit across bins) and the blue line (observed lift if targeting randomly, LOESS fit across bins) can be used as a model selection metric by selecting the model specification that maximizes the targeting lift indicated by the area between the curves. The right plot is the stack-ranking plot, where observations are binned according to the estimated treatment response score. The blue line is a LOESS fit across the bins of the observed treatment effect within each bin, while the green line is a LOESS fit across the bins of the average estimated treatment effect within each bin. Good model fit is indicated by closely overlapping curves.

highest-responding subjects at the top end of the score and the subjects most likely to show backlash at the bottom end of the score.

First, I discuss the TAUC metric, which is the left-hand plot in Figure 2.1. Although introduced originally in Naranjo (2012) and mentioned briefly in Gutierrez and Gerardy (2016)[3], its properties as a model selection technique for HTE models have not been explored. The TAUC metric involves first binning (out-of-sample) observations according to their estimated treatment response from the model and assigning each observation a label $k = \{1, 2, \ldots, K\}$ where $k = 1$ indicates the high-

[3]It was also used as a graphical illustration of model lift in Imai and Strauss (2011)

est quantile of $\widehat{\tau}_i$ and $k = K$ indicates the lowest quantile. Then, for each bin k, I calculate

$$\widehat{\text{Lift}}_k = \underbrace{\left(\frac{1}{|i : T_i = 1, k_i \leq k|} \sum_{i:T_i=1,k_i \leq k} Y_i - \frac{1}{|i : T_i = 0, k_i \leq k|} \sum_{i:T_i=0,k_i \leq k} Y_i \right)}_{\text{Treatment effect in top } k \text{ quantiles}} \times \underbrace{|i : k_i \leq k|}_{\text{\# observations in top } k \text{ quantiles}}$$

Each point along the curve is a measure of the estimated *cumulative lift* achieved if the top k bins of $\widehat{\tau}_i$ had been assigned the treatment, given the average treatment effect among that subset of observations. This curve is compared against the cumulative lift achieved by random targeting, which in bin k is simply equal to the average treatment effect across the whole sample multiplied by the number of observations in bins 0 through k. These quantities are equal when $k = 0$ (cumulative lift of 0; no observations treated) and when $k = K$ (cumulative lift is ATE $\times$ N; estimate of cumulative lift if every observation given treatment). These two curves can then be plotted against each other, and the area between the two curves can be easily estimated as a measure of the model's lift-over-random targeting. I refer to this area-between-curves as the Treatment Area-under-Curve (T-AUC) metric throughout the rest of the paper. In Subsections 2.3.1 and 2.3.2, I show how the T-AUC metric can be used as a model selection and model comparison metric.

We can also test the robustness of the T-AUC metric using straightforward randomization inference techniques. We can randomly shuffle the labels k and re-calculate the T-AUC many times, which gives us a distribution of what T-AUC would be under the null hypothesis that the model is doing no better than random targeting. Figure 2.2 shows the results of this exercise on the same simulated data used to create Figure 2.1, across 250 random shufflings of the labels. As expected, the null distribution of the T-AUC under random targeting is centered around 0, which

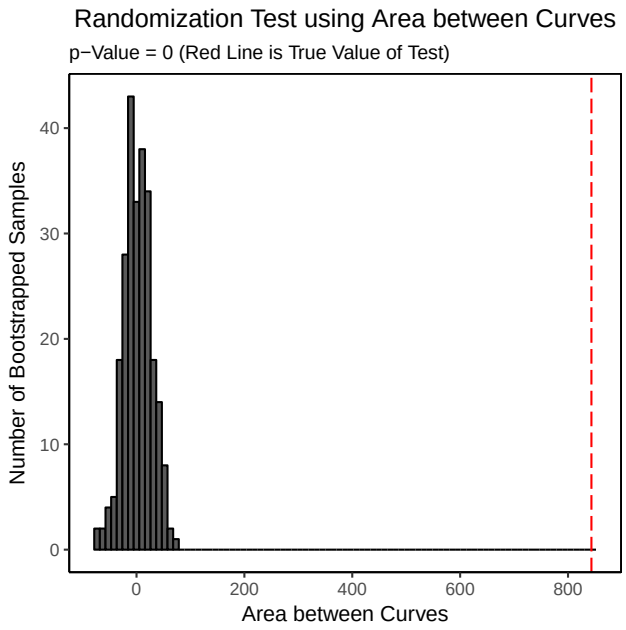

Figure 2.2: A diagnostic for assessing the robustness of the estimated T-AUC curve. In this diagnostic, we re-estimate the T-AUC after shuffling the score ordering of the observations R many times. This gives an empirical distribution of the T-AUC on the data that we would expect under the null hypothesis that the estimated score does no better than random targeting. We can then calculate the proportion of T-AUC estimates that are equal to or greater than the true T-AUC, which can be interpreted as a p-value assessing the probability that we would observe a T-AUC estimate as large as or larger than what we get from targeting by the score compared to the null hypothesis of random targeting.

is what we would expect if the score was sorting no better than random. In contrast, the true T-AUC from the score is well outside the range of the null distribution. We can also calculate a p-value of the probability that we would observe a T-AUC this extreme or more under the null hypothesis that the score were doing no better than random targeting over R random permutations of the score, as

$$p^{\text{TAUC}} = \frac{\sum_R \mathbf{1}\{\text{TAUC}_r \geq \text{TAUC}_{\text{true}}\}}{R}$$

Another metric for assessing HTE model fit is the *stack-ranking* diagnostic, depicted on the right side of Figure 2.1. The stack-ranking diagnostic visualizes whether

the out-of-sample predictions $\widehat{\tau_i}$ align closely with the true values of τ_i — as τ_i is unobservable, we instead compare the mean of $\widehat{\tau_i}$ in subsets of the data against the true average treatment effect in the same subset of the data. In a correctly-specified model, the mean scores and the true ATE in each subset should be equal to each other. Like the T-AUC diagnostic, we first bin (out-of-sample) observations according to their estimated treatment response from the model and assigning each observation a label $k = \{1, 2, \ldots, K\}$ where $k = 1$ indicates the lowest quantile of $\widehat{\tau_i}$ and $k = K$ indicates the highest quantile. Then, for each label k, we calculate:

$$\widehat{\overline{\tau_k}} = \frac{1}{|i : k_i = k|} \sum_{i:k_i=k} \widehat{\tau_i}$$

$$\text{ATE}_k = \frac{1}{|i : T_i = 1, k_i = k|} \sum_{i:T_i=1,k_i=k} Y_i - \frac{1}{|i : T_i = 0, k_i = k|} \sum_{i:T_i=0,k_i=k} Y_i$$

Then, by plotting ATE_k and $\widehat{\overline{\tau_k}}$ against each other, we can diagnose where the out-of-sample predictions from the HTE model are mis-fit against the true value of the ATE in different cuts of the data.

Both the T-AUC and the stack-ranking graphical diagnostic require choosing the parameter K, which is the number of quantiles to cut the data on using $\widehat{\tau_i}$. This is a difficult tradeoff for the applied researcher — choosing K to be very coarse gives more precise subgroup estimates but risks smoothing over real mis-specification in the data, while choosing K to be very fine may reveal mis-fits in the data that are simply noise. To avoid this balancing act by the researcher, I repeat all calculations starting with $K = 2$ and increasing it until $K = 20$, storing all estimates. This returns the table shown in Table 2.2, where for the T-AUC diagnostic Estimate_k is both TAUC_k and $\text{ATE} \times \frac{N}{k}$, while for the stack-ranking diagnostic it is $\widehat{\overline{\tau_k}}$ and ATE_k. I then run a simple weighted LOESS of Estimate_k onto k/K, weighted by N to up-weight the importance of estimates made with larger sample sizes. The final T-AUC metric is the area between the predictions of the curves returned by the LOESS fit.

k	K	Estimate$_k$	N
1	2	...	Sample Size/2
2	2	...	Sample Size/2
1	3	...	Sample Size/3
⋮			
3	3	...	Sample Size/3
⋮			
1	20	...	Sample Size/20
⋮			
20	20	...	Sample Size/20

Table 2.2: Table of estimates for HTE diagnostics, re-estimated over different values of K. When re-estimating diagnostics across different values of K, we can use simple weighted smoothing techniques such as LOESS to estimate these diagnostics without having to balance between values of K that are either too coarse or too fine.

2.3.1 Simulations to Validate the T-AUC Metric

The T-AUC metric, introduced in the previous section, provides both a graphical summary of the efficiency of the model as well as a single numerical summary of its efficiency over random targeting. In this section, I validate the estimated area between the targeting-by-score curve and the targeting-by-random curve as a model selection criterion using simulation evidence. The proposed metric successfully selects the correct model in many settings, although it struggles when the data-generating process in the model for τ_i is complex and both the data-generating process and the covariate set are misspecified. I also compare the T-AUC metric against another metric that selects the model that minimizes the average distance between the estimated score and the transformed outcome used in the F-learner model. Both perform well, although the T-AUC does slightly better under a larger set of data-generating processes.

To assess the model selection performance of the proposed metric, I generate 1000 simulated data sets using Algorithm 5. For each simulated data set, I run each of the four learner algorithms as a linear model and a BART model under the default settings

for both estimators. Given that there are two levels of the data-generating process — one for the outcome model of $Y_i(0)$ and another for the heterogeneity model of τ_i — I also systematically omit a variable relevant to the outcome model, the τ_i model, or a variable relevant to both models simultaneously to examine the performance of the proposed metric under misspecification. Out-of-sample, I then calculate the mean absolute forecasting error of the model predictions (which is unobservable outside of simulations), defined as

$$\text{MAFE} = \frac{\sum_{i=1}^{N} |\tau_i - \widehat{\tau}_i|}{N},$$

as well as the T-AUC. Here, the MAFE represents a type of ground truth, in that it is a measure of model accuracy that we would likely use if we were able to actually observe the out-of-sample τ_i — therefore, if the T-AUC is selecting similar models that we would select if we were able to rely on MAFE, then it is a doing a good job as a proxy measure of model performance. I then store the correlation between the MAFE and the TAUC, where a correlation closer to -1 suggests that the T-AUC is performing as well as the true and unobserved MAFE as a measure of model accuracy, even without observing the true target outcome. I also store whether the specification that minimizes the true MAFE is also one of the top two model specifications that maximizes the T-AUC. The results are shown in Figure 2.3.

In the top row, the simulation results show that the out-of-sample T-AUC and MAFE correlate closely in most scenarios, with most simulations averaging a rank correlation of around -0.70 or lower. Only when the DGP for τ_i is misspecified and complex (blue and yellow lines, right-hand column), and the correlation between the excluded and included variables in the τ_i model very low, does the T-AUC start to struggle as a model selection metric. In other scenarios, including when the outcome DGP is complex and the DGP for τ_i is simple, the T-AUC effectively orders the candidate models in line with the MAFE had it been observed. The bottom row also

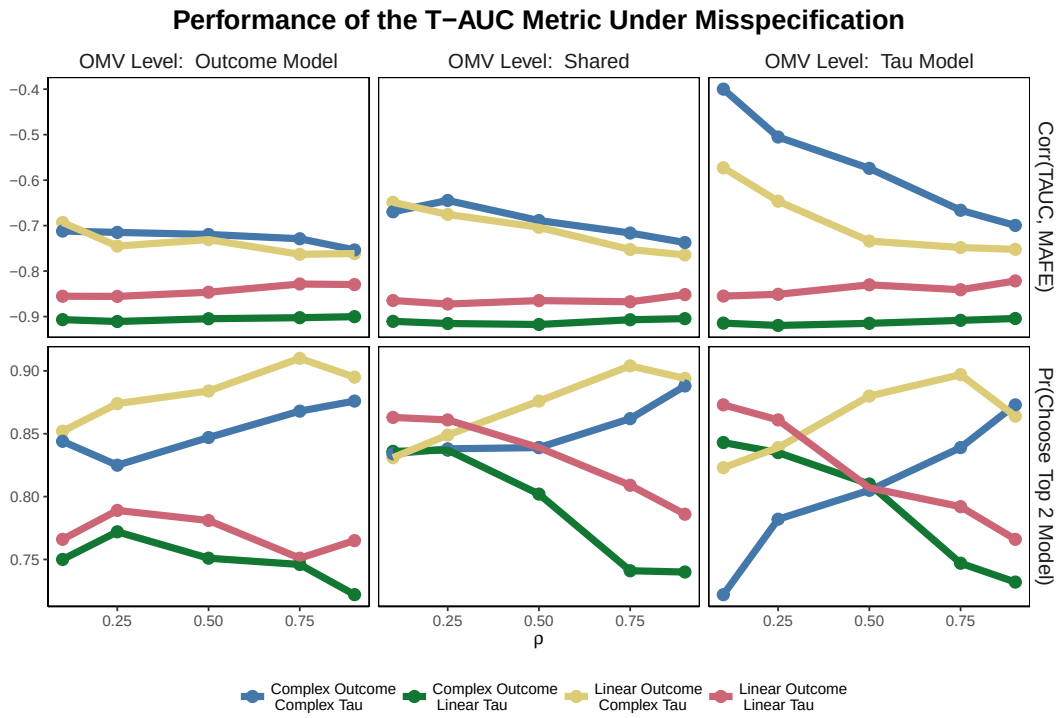

Figure 2.3: Simulations validating the T-AUC as a model selection metric. Each panel represents a different misspecification mechanism (misspecified outcome DGP, both DGPs misspecified, τ_i DGP misspecified) and each line represents a different data-generating process for the simulations. The x-axis indicates how correlated the misspecified and included covariates are, and the y-axis in the top row shows the average out-of-sample correlation across 1000 simulated data sets, where lower correlation indicates better T-AUC performance. In the bottom row, the y-axis shows how frequently across 1000 simulated data sets the model specification that minimizes the true MAFE is also one of the top two best-performing model specifications as measured by T-AUC, where higher values indicate better model performance.

shows that the T-AUC is effectively choosing the best-performing candidate models, as measured by whether the specification that minimizes the MAFE is also one of the top two specifications as measured by T-AUC. In most data-generating scenarios, this occurs over 80% of the time across the 1000 simulated data sets.

We can also constrast the T-AUC with another test statistic for HTE model performance — minimizing the disparity between the modeled $\widehat{\tau}_i$ and the transformed outcome Y_i^* proposed in the F-learner. As a brief review, the F-learner is attractive as a modeling strategy for HTE models because the transformed outcome, Y_i^*, is equal

to the individual-level treatment effect in expectation[4], where

$$Y_i^* = Y_i \times \frac{T_i - e(X_i)}{e(X_i)(1 - e(X_i))}$$

In theory, an attractive estimator of τ_i would then be to regress Y_i^* onto covariates using the estimator of choice — however, as discussed in Athey and Imbens (2016), the fact that this estimator does not use the treatment assignment information beyond the transformation of the outcome causes much of this inefficiency. However, we can try to use Y_i^* as a model selection criterion independent of its use in the F-learner, where the analyst selects the model that minimizes the aggregated distance between $\widehat{\tau}_i$ and Y_i^* as measured using mean absolute error, mean squared error, or some other distance measure. I run the same set of models as the simulations presented in Figure 2.3, but in addition to calculating the T-AUC, I also calculate

$$\text{MAFE}^{Y^*} = \frac{\sum_{i=1}^{N} |\widehat{\tau}_i - Y_i^*|}{N}$$

for each model run. I then take the correlation between MAFE^{Y^*} and the true MAFE across all models run for a given simulation, where higher levels of correlation indicate better performacne of the tranformed outcome metric. I also store how frequently the model specification that minimizes the true MAFE is one of the two specifications that does best as measured by MAFE^{Y^*}.

Figure 2.4 gives the results of the comparison. In most scenarios, the T-AUC metric outperforms the transformed outcome-based metric for model selection, in that it correlated closer with the true MAFE than the transformed outcome metric (top row) and more frequently identified the model that minimized the true MAFE as one of the top-two performing models as measured by T-AUC (bottom row). When the τ_i DGP is complex and the left-out covariates were uncorrelated with included

[4]This is shown in Appendix Section A.1.1

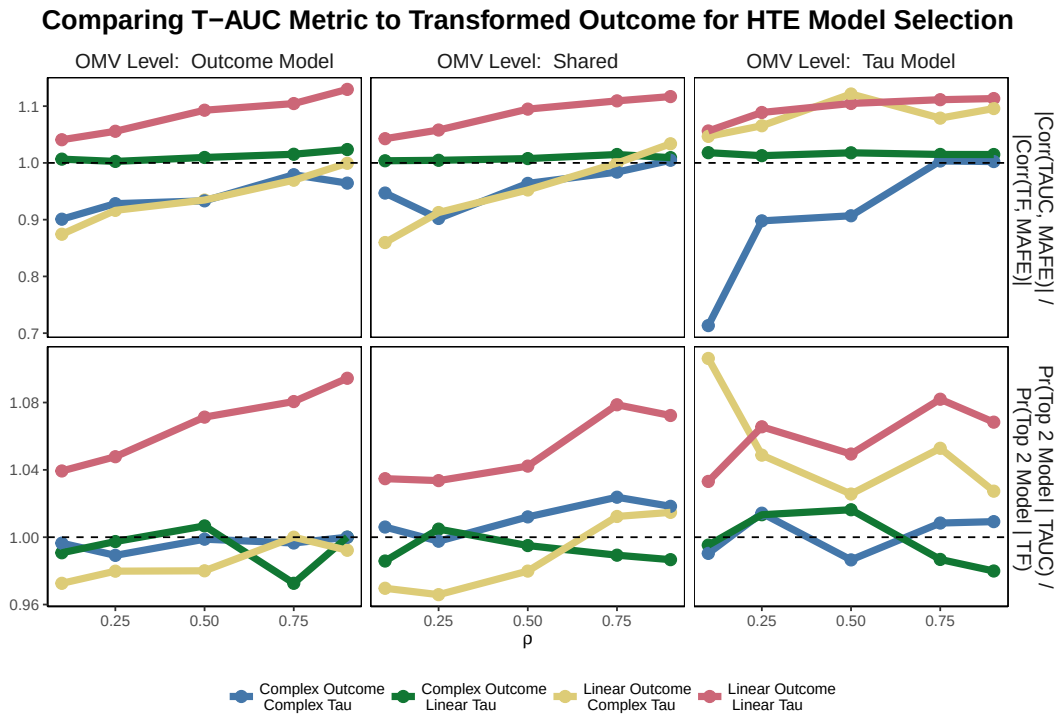

Figure 2.4: Simulations comparing the transformed outcome metric to the T-AUC metric for model selection. These simulations follow the same setup as those presented in Figure 2.3. Here, the y-axis is the ratio between the average correlation of the true MAFE and the T-AUC, and the average correlation between the true MAFE and the MAFE between $\widehat{\tau}_i$ and Y_i^*. Any lines above 1 indicate that the T-AUC metric is outperforming the transformed outcome metric, while lines below one indicate the opposite. In most scenarios, the T-AUC outperforms the metrics based on Y_i^* as a model selection metric.

covariates, however, the transformed outcome approach slightly outperforms T-AUC as a model selection criterion.

2.3.2 Tuning a Random Forest using the T-AUC Metric

Having showed that the T-AUC metric can be a useful metric for model selection, I now provide a practical illustration of how it can be used — tuning a random forest estimator of $\widehat{\tau}_i$. Random forests, which were introduced in Breiman (2001), are powerful predictive models based on recursive partitioning methods, which iteratively select decision rules that maximize the ability to predict an outcome of interest. Unlike standard recursive partitioning models such as CART, which grow only a

single "decision tree", random forests grow many trees using random sub-samples of both the data and of the explanatory variables, which help prevent over-fitting. The final prediction for a given observation is a weighted average of the predictions of all of the trees grown by the random forest.

Despite their good predictive performance, random forests have many tuning parameters that are difficult to choose well if there is no way to assess out-of-sample predictive performance. These parameters include the number of trees to grow to form the final ensemble, the maximum number of "nodes" to grow each tree to (each observation grouped into a node gets the same prediction — fewer nodes guard against overfitting at the cost of additional bias), and the number of covariates to test to make each split. This issue is not just specific to random forests — other tree-based models, such as GBMs and BARTs, have many tuning parameters that are often chosen through out-of-sample validation exercises. For the random forest, these parameters include the number of trees to grow in the ensemble, how deep to grow each tree, the number of rows to sample in each new split of the tree, and the number of columns to sample in each new split of the tree.

Here, I use the T-AUC metric calculated on simulated out-of-sample data to choose the best-performing set of tuning parameters (maximum number of nodes to grow and the number of covariates to sample) in a setting where the relationship between covariates and the true treatment effect is highly non-linear and interactive, thereby requiring non-parametric methods such as random forests or GBM's.[5] I first run the random forest given a certain tuning parameter setting on a 2500-observation subset of the data, and then generate predictions on an out-of-sample subset of 2500 observations. I then calculate both the T-AUC on the out-of-sample subset, as well as the true Mean Absolute Forecasting Error.

[5] I generate this data set using a single iteration of Algorithm 5, where $\rho = .25$, $Y_i(0)$ DGP is complex, and τ_i DGP is complex.

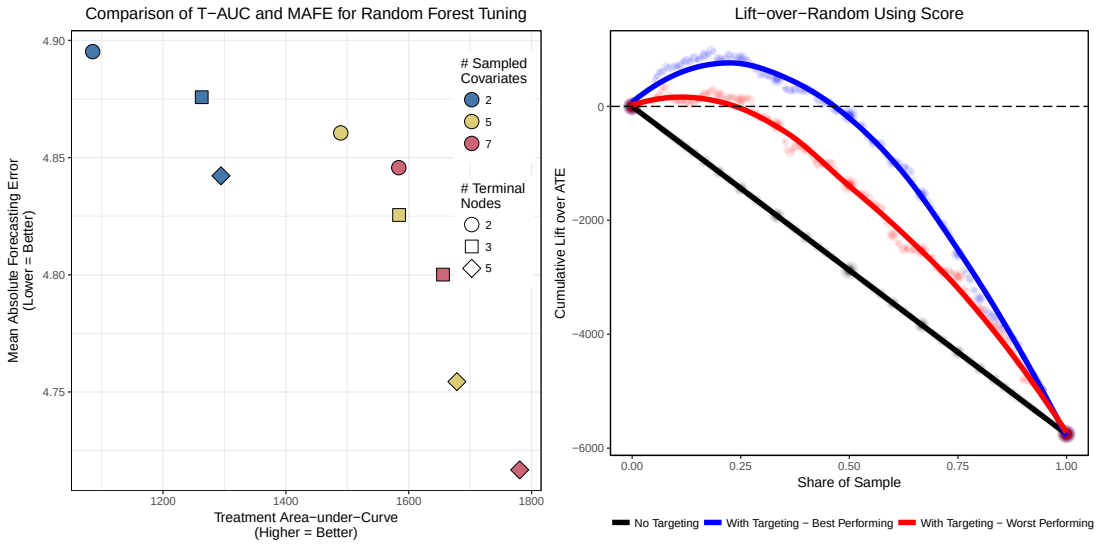

Figure 2.5: Validating the T-AUC as a model selection metric. The left-hand plot shows the relationship between the T-AUC metric calculated out-of-sample against the true out-of-sample Mean Absolute Error. Each point represents the out-of-sample performance of a Random Forest HTE model run as an X-learner, where the colors indicate the number of sampled covariates in each split and the shapes indicate the maximum tree depth grown. The two measures strongly negatively correlate, suggesting that the T-AUC can be effectively used as a model selection metric for HTE models. The right-hand plot shows the T-AUC for the worst-performing model (red) and the best-performing model (blue) relative to random targeting (black). Using the T-AUC to select the hyperparameter settings results in a model that selects more treatment-responsive individuals while avoiding treating negatively-responsive individuals.

Figure 2.5 shows the results of this test. In the left-hand plot, we can see that the T-AUC and the true MAFE correlate extremely closely, at -0.95. Using the T-AUC to select specifications for the random forest would give us the same specification that minimizes the true MAFE, with a maxmimum tree depth of 5 and 7 sampled covariates at each split. That the deepest trees that sample the most covariates at each split performs the best is reflective of the highly interactive and nonlinear DGP in both the outcome stage and the τ stage. The right-hand plot visualizes the consequences of choosing the worst-performing model versus the best-performing model using the T-AUC. The worst-performing model shows a positive cumulative lift over the first 30% of the sample using the out-of-sample score estimates, before

returning negative cumulative treatment effects. In contrast, the best-performing model still shows positive cumulative effects of the treatment after bringing in the top 50% of the sample ordered by score, despite the average treatment effect across the entire sample being negative.

2.4 Uncovering Turnout Persuadability in Social Pressure GOTV Experiments

The previous sections introduced some practical methods for building and tuning heterogeneous treatment effect models for targeting voters and consumers, and validated the proposed metrics in a set of simulations. Here, I apply these diagnostics and metrics to develop a model-building workflow that uncovers shared patterns of treatment responsiveness in a series of social pressure Get-Out-the-Vote experiments. The proposed workflow allows the researcher to compare across candidate models while also visually assessing misspecification using the tools introduced in previous sections.

I demonstrate these tools in the context of a set of experiments studying how effectively Get-out-the-Vote (GOTV) mailers triggering feelings of social pressure can motivate citizens to vote. This literature draws extensively from social psychology, where researchers have found that increasing the prevalence of dominant social norms can make individuals more likely to comply with those norms. Gerber, Green and Larimer (2008) draw from this tradition to design a series of GOTV mailers that gradually increase the amount of social pressure placed on the voter to turn out to vote in a low-salience primary in Michigan taking place in 2006. They randomized voters into five separate groups, including four treatment groups (ordered roughly by level of social pressure exerted):

1. **Control:** No mailer.

2. **Civic duty:** Mailer emphasizes importance of civic duty of voting.

3. **Hawthorne effect:** Mailer tells voters that their voting behavior in upcoming primary is being studied.

4. **Self:** Mailer reminds voter that vote history is public and reports the previous voting history of voters in the household. Suggests that updated mailers will be sent after election.

5. **Neighbors:** Mailer reminds voter that vote history is public and reports the previous voting history of all voters in the neighborhood of the address. Suggests that updated mailers will be sent after election.

Table 2.3 shows the results of the treatments. The left-most column shows the baseline turnout rate as estimated in the control group. Each additional row reports the results from treatments applying additional social pressure. As found in the initial experiment, the most aggressive social-pressure treatments lead to massive increases in turnout, with the Neighbors treatment inducing more than a 25% increase over baseline turnout in the control group. However, the other treatments also lead to substantively large increases in turnout. The standard academic experiment would stop at this point, having identified a substantively and theoretically important effect. However, for a campaign looking to boost turnout by targeting these mailers, a decision-maker may want to explicitly model these treatment effects for one of two reasons — first, they may be budget-constrained, thereby requiring them to estimate which members of the electorate will be most responsive to the mailer. Second, even if the campaign is not budget-constrained, they may be wary of backlash effects and may want to suppress voters expected to react poorly to the treatment.

Here, I use the tools proposed in previous sections to build a best-performing heterogeneous treatment effect model on the Gerber, Green and Larimer data, which I then apply to the data from another social pressure GOTV experiment conducted

	Control	Civic Duty	Hawthorne	Self	Neighbors
% Voting	31.1%	33%	33.7%	36.1%	39.4%
N	184749	36903	37005	37011	36893

Table 2.3: Replication of Table 2 in Gerber, Green and Larimer (2008). While the effects scale and N differ slightly from the original due to a pre-processed replication file, the same substantively large effects are apparent here. All estimated turnout shares are simple means calculated within each treatment group.

two years later by Sinclair, McConnell and Green. While both experiments were run in low-salience election environments (a Michigan primary election in Gerber, Green and Larimer, and a Chicago municipal special election in Sinclair, McConnell and Green), they differ in the year conducted, the timing of the election, the baseline level of turnout, and the level of government for the contested seat. Therefore, this presents a hard test to discover any relevant patterns of treatment effect heterogeneity that could generalize across experiments.

To build the model on Gerber, Green and Larimer (2008), I use the following model-building workflow:

1. Clean and harmonize data to a common set of covariates between experiments:
    - **Treatment indicator:** 0 for no mailer, 1 for either social pressure mailer
    - **Outcome:** 1 if voted in relevant election, 0 if not
    - **Common covariates:** gender, age buckets (18-35, 35-54, 55-69, 70+), vote history dummies
2. Run treatment effect models with 5-fold cross-validation on following parameters:
    - **Estimator:** linear model, lasso-linear model, random forest
    - **Learner:** F, S, X, F

- **Random Forest Tuning Parameters:** Number of trees grown (500, 1000, 1500), number of covariates sampled per split (2, 3, 5), maximum nodes to grow per tree (2, 3, 5)

3. Average out-of-sample TAUC for each parameter set

4. Visually inspect out-of-sample TAUC, its robustness, and the stack-ranking on best-performing model

5. Estimate model using best-performing model specifications on the full sample from Gerber, Green and Larimer (2008)

6. Generate predictions from the model on Sinclair, McConnell and Green (2012)

The visual diagnostics from the best-performing model are shown in Figure 2.6. Using the procedure described above, I selected a random forest X-learner growing 1000 trees, sampling two covariates at each split, and growing each tree to a maximum of five terminal nodes. The T-AUC plot (top left) shows that the model gives targeting performance that is strictly better than random across the entire estimated score space, although it struggles to give much lift at the highest quantiles of the score. Despite the poor performance at the top end of the score, the randomization test of the T-AUC score (bottom left) suggests that the score is still doing much better than fully random targeting.

This finding is echoed in the stack-ranking plot (top right), where we see that the score struggles to distinguish between voters estimated to be in the top half of treatment responsiveness as indicated by the non-monotonic green curve above the 50% quantile. In contrast, voters estimated to be in the bottom half of treatment responsiveness appear to be tracking fairly monotonically with the observed ATE in each cut. The model's failure to capture the full range of treatment responsiveness may be in part to the limited number of available covariates — any heterogeneity

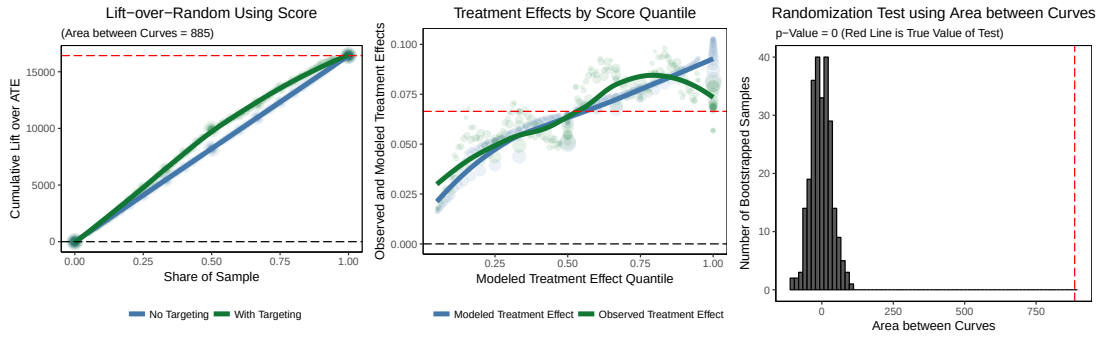

Figure 2.6: Visual diagnostic plots from the best-performing heterogeneous-treatment effect model estimated on Gerber, Green and Larimer (2008). The top-left plot is the T-AUC curve and the estimated out-of-sample area, the top-right plot is the stack-ranking of the model, and the bottom-left plot is the result of the randomization test of the T-AUC curve. All results are out-of-sample from five-fold cross-validation.

by partisanship or race, for example, that is not already captured by vote history, age, or gender will go un-modeled here. Neymanian and Fisherian hypothesis tests such as those proposed in Ding, Feller and Miratrix (2016) and Ding, Feller and Miratrix (Forthcoming) can help detect this additional un-modeled treatment effect heterogeneity.

Next, to test the generalizability of the estimated model, I predict the pattern of treatment responsiveness in Sinclair, McConnell and Green (2012) and examine their diagnostics. The results, presented in Figure 2.7, suggest that the pattern of treatment responsiveness estimated from Gerber, Green and Larimer (2008) travels well to new (and fully out-of-sample) data. This is illustrated most effectively in the stack-ranking plot (top right) — the observed treatment effect increases nearly perfectly monotonically along the estimated treatment responsiveness scale. While there is an intercept shift relative to the estimated score, due to the difference in scale between the baseline treatment effects between the two experiments, the estimated and actual treatment effects within each cut of the score track each other closely. Both the T-AUC and the randomization test of the T-AUC suggest that this finding is fairly robust.

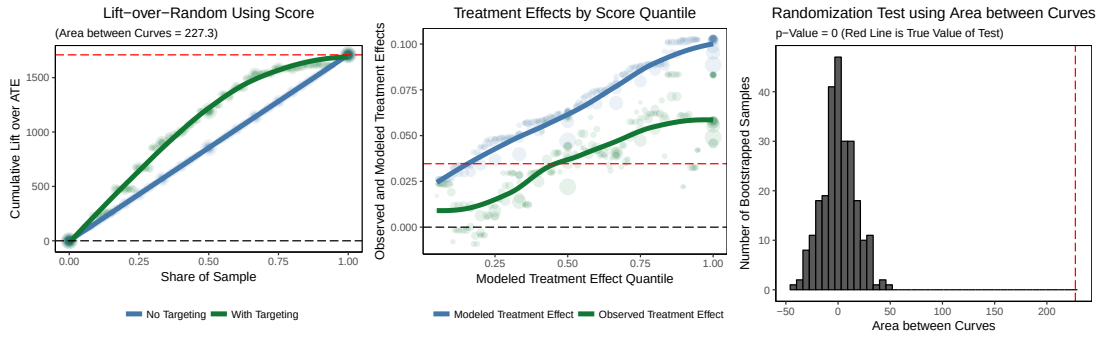

Figure 2.7: Visual diagnostic plots from scoring the data from Sinclair, McConnell and Green (2012) using the best-performing model estimated on Gerber, Green and Larimer (2008). The top-left plot is the T-AUC curve and the estimated out-of-sample area, the top-right plot is the stack-ranking of the model, and the bottom-left plot is the result of the randomization test of the T-AUC curve.

What is the source of this underlying shared treatment propensity? To better understand the shared factor driving treatment response heterogeneity in these separate experiments, I first plot the distribution of estimated treatment effects within factor levels. While this does not fully marginalize over other covariates, it nonetheless presents an initial descriptive glance at what factors are driving the sorting in the model. I examine these distributions within the unique levels of gender and age buckets, as well as quantiles of a turnout propensity score built by predicting turnout in the 2002 general election as a function of covariates among the control group using a random forest.

Figure 2.8 shows the distribution of predicted treatment responsiveness within observed factor levels. There is very little noticeable heterogeneity in treatment response across gender, and while the estimated treatment response increases somewhat across quantiles of the vote propensity score, it remains fairly noisy. However, there is clear heterogeneity by age that is apparent when plotting the distribution of the estimates within age bucket. Treatment response propensity is far lower for younger voters (18-35) than for middle-aged or elderly voters, and this propensity increases nearly twofold when targeting voters aged 70+ versus voters aged 18-35.

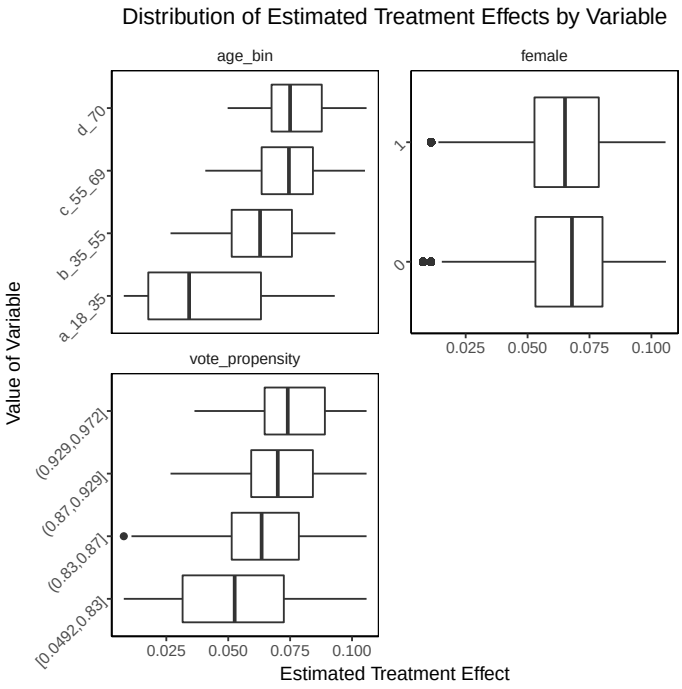

Figure 2.8: Distribution of estimated treatment effects in Gerber, Green and Larimer (2008) using best-performing model. This plot shows distributions of estimated treatment effects within each unique factor level for age buckets and gender, and within a quantiling of a vote propensity score.

This pattern is echoed in the raw data for both Gerber, Green and Larimer and Sinclair, McConnell and Green, and appears to be the primary driver behind why the model from the former extrapolates well to the latter data. Figure 2.9 shows the raw conditional average treatment effects by age buckets separately for each experiment. Both experiments show similar jumps over the baseline treatment effect for older cohorts — when compared to the baseline treatment effect (for voters aged 18-34), older voters show anywhere between a 3-4 percentage point increase in treatment responsiveness. Thus, the score estimated on the data from Gerber, Green and Larimer, which captures this heterogeneity, extrapolates well to Sinclair, McConnell and Green by successfully modeling these differences in treatment responsiveness.

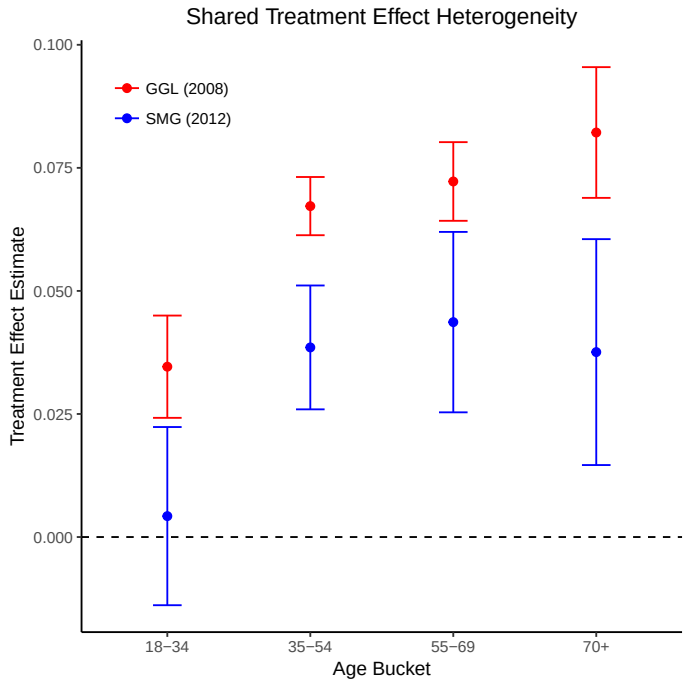

Figure 2.9: Conditional average treatment effects by age bucket for Gerber, Green and Larimer and Sinclair, McConnell and Green. Both experiments show similar patterns of heterogeneity by age, although treatment effects for each age bucket in Sinclair, McConnell and Green are lower than those in Gerber, Green and Larimer. This common heterogeneity helps explain why models built on one of the two experiments extrapolate well to the other experiment.

2.5 Conclusion

While numerous methods have been proposed for estimating and measuring heterogeneous treatment effects, comparatively little work has studied how to conduct model selection in order to choose the best-performing of a set of candidate heterogeneous treatment effect models. This paper introduces a series of graphical diagnostics to aid researchers and analysts constructing these models, as well as a new metric, the Treatment Area-under-Curve, to aid in explicit model comparison even when the outcome of interest is unobservable. I also provide practical advice on model building, analysis, and inspection for applied researchers and analysts seeking to construct HTE models. Lastly, I propose a straightforward model-building workflow and apply

it to two separate social pressure Get-out-the-Vote experiments in order to analyze shared patterns of heterogeneity between the two studies.

While this paper gives additional guidance on how to build robust and well-performing heterogeneous treatment effect models, numerous questions still remain understudied in the HTE modeling literature. Among these is the role of survey design in creating high-quality training data. In political targeting, training data is often created by running surveys linked back up to a voter file with either a survey-embedded experiment or in-person delivered treatment, and recent research suggests panel designs (Kalla and Broockman, 2018) can improve the efficiency of the estimated sample average treatment effect, thus requiring less data for equivalent precision. One question for HTE modeling is the impact this survey design choice has on the external validity of the estimated score, especially in the presence of severe differential non-response bias.

Another open question is how to conduct proper power analysis for an experiment that will be used as training data for HTE models. Few recommendations exist in the literature, especially since the "hypothesis" to be tested and measured is not completely clear in the context of treatment effect modeling. While the T-AUC metric may be a possible component of a successful power analysis, I leave this question for future research.

Chapter 3

Validating Ensembles of Simulated Redistricting Plans

3.1 Introduction

Congressional redistricting, which is the practice of redrawing congressional district lines following the constitutionally mandated decennial Census, is of major political consequence in the United States. Redistricting fundamentally reshapes geographic boundaries that define a central class of representation and governance in the American political system, and changes in those boundaries have major political consequences. As a fundamentally political process, redistricting has also been manipulated to fulfill partisan ends, and recent debates have raised possible reforms to lessen the role of politicians and the influence of political motives in determining the boundaries of these political communities.

Starting in the 1960s, scholars began proposing simulation-based approaches to make the redistricting process more transparent, objective, and unbiased (early proposals include Vickrey, 1961; Weaver and Hess, 1963; Nagel, 1965; Hess et al., 1965). While this research agenda lay dormant for some time, recent advances in computing capability has led to a resurgence in proposals, implementations, and applications of

simulation methods to applied redistricting problems (e.g. Cirincione, Darling and O'Rourke, 2000; McCarty, Poole and Rosenthal, 2009; Altman and McDonald, 2011; Chen and Rodden, 2013; Mattingly and Vaughn, 2014; Liu, Tam Cho and Wang, 2016; Herschlag, Ravier and Mattingly, 2017; Fifield et al., 2018; Chikina, Frieze and Pegden, 2017; Magleby and Mosesson, 2018). Furthermore, simulation methods for redistricting play an increasingly important role in court cases challenging redistricting plans. In 2017, simulation evidence was introduced and accepted by courts hearing redistricting challenges in North Carolina (*Common Cause v. Russo* (2017): Mattingly, 2017) and Pennsylvania (*League of Women Voters v. Wolf et al.* (2017): Pegden, 2017; Chen, 2017; Cho, 2017). Given the pending challenges to maps in Michigan and Virginia as well as the upcoming decennial Census in 2020, simulation methods are likely to become an even more influential source of evidence for legal challenges to redistricting plans at all levels of government.

Simulation methods are particularly useful because enumeration of all possible redistricting plans in a state is computationally infeasible. In New Hampshire, for instance, there are 1.4 x 10^{98} partitions of the state's 327 voting precincts into two congressional districts using the standard Stirling number calculation — a massive number for a relatively simple redistricting problem. While statutory guidelines and requirements such as district contiguity, population parity, compactness, and avoiding splitting communities of interest reduce the number of solutions dramatically, the resulting problem remains out-of-reach of full enumeration methods. Therefore, to compare an implemented redistricting plan against a set of other candidate plans on any number of partisan or representational outcomes, researchers and redistricting reform advocates must resort to simulation methods.

However, despite the widespread use of redistricting simulation methods in court cases, researchers rarely attempt to evaluate their accuracy — the ability of the simulated sample, commonly referred to as an *ensemble*, to approximate the true

distribution of redistricting plans. This has profound implications for the quality of expert testimony in court and for academic scholarship. If researchers cannot provide evidence that the ensembles drawn using their simulation methods are representative of the full space of valid redistricting plans, the comparison between an implemented redistricting plan against a set of simulated plans on some partisan or representational benchmark provides little to no scientific evidence. To address this shortcoming, Fifield et al. (2018) enumerate all possible partitions of 25 precincts from Florida into three contiguous congressional districts (which I refer to as FL25) and compare the ensemble from their algorithm against the true distribution of plans. Other researchers (Magleby and Mosesson, 2018) have used FL25 to evaluate their own algorithms — however, this data represents only a single set of precincts representing a specific political geography, and may not be representative of other redistricting problems. Furthermore, as noted by Magleby and Mosesson (2018), this data set is not particularly balanced — only six partitionings fall within standard levels of population parity ($\pm 1.5\%$), and most fall above 10%.

To overcome this shortcoming, this paper proposes a set of tests based off of enumeration methods that can be used to evaluate the accuracy of redistricting simulation methods. I propose randomly sampling many small maps of a pre-specified size from actual electoral geographies such as state shapefiles, and then using enumeration methods to calculate all possible partitions of those maps into a prespecified number of contiguous districs. Simulation methods can then be used on each subset separately, and then the true and simulationed distributions can be compared against each other in each sub-map using measures of distributional similarity such as Kolmogorov-Smirnov tests, the Kullback-Liebler divergence, or Earth Mover's distance. This proposal essentially repeats the exercise conducted in Fifield et al. (2018) many times, in order to smooth over the idiosyncracies of FL25 and generalize the process to other states and geographies. I note, however, that providing positive evi-

dence that a simulation method does well on these tests is *not* positive evidence that it will perform well on redistricting problems the size of a full state — rather, failing these tests on small redistricting problems such as FL25 is evidence that a simulation method will scale poorly when applied to full states. These methods are implemented in an open-source R software library, redist (Fifield, Tarr and Imai, 2015).

In the next section, I briefly review existing redistricting simulation methods, and how they incorporate standard redistricting constraints such as compactness, population parity, and respecting communities of interest. Section 3.3 discusses how FL25 has been used to evaluate redistricting simulation methods, as well as the shortcomings and idiosyncracies of that particular data set. Section 3.4 introduces the proposed test, and applies it to the open-source redistricting simulators of Chen and Rodden (2013) and Fifield et al. (2018). Finally, Section 3.5 concludes.

3.2 Simulation Methods for Evaluating Redistricting Plans

Since the early 2000s, a number of redistricting simulation methods have been proposed for the purpose of drawing a large sample of redistricting plans respecting a variety of statutory and political constraints. In this section, I briefly review the broad classes of methods introduced to tackle the redistricting simulation problem, along with how they incorporate in basic constraints and whether they provide any theoretical guarantees on the representativeness of the samples they draw. Note that I specifically examine redistricting *simulation* methods in this section — a parallel line of research has developed redistricting *optimization* methods for drawing an approximately or exactly optimal plan given a particular objective function.[1] I therefore exclude them from the discussion here and the analysis to follow.

[1]Papers in this line of research include Garfinkel and Nemhauser (1970); Browdy (1990); Chou and Li (2006); Fryer and Holden (2011), among others.

In the analysis to follow, I will use the following common notation and assumptions. I assume that any state has already been divided into a number of discrete sub-units, frequently voting precincts or census block units. Each state can then be represented as an adjacency graph $G = \{V, E\}$, where $V = \{\{1\}, \{2\}, \ldots, \{m\}\}$ represents each discrete sub-unit as a node in an adjacency graph, and E is the set of edges connecting these nodes, indicating that those two nodes are geographically adjacent to each other. For example, if nodes i and j are adjacent to each other, then an edge between them exists such that $(i, j) \in E$. A redistricting plan is simply a partition of the node set V into n contiguous sub-units $\mathbf{v} = \{V_1, V_2, \ldots, V_n\}$ by removing a subset of the edge pairs $(i, j) \in E$.

Redistricting simulation methods generally fall into one of three categories. *Random-Seed-and-Grow* methods (RSG) draw districts by randomly selecting n seed nodes from the node set V, and randomly adding adjacent precincts to those seeds until the desired number of districts are drawn using all geographic units. Then, nodes are swapped between adjacent partitions until any thresholding requirements on compactness, population parity, or other constraints are satisfied. *Markov chain Monte Carlo* methods (MCMC) start with a valid partition $\mathbf{v}$ and then propose a partition $\mathbf{v}^*$ achieved by cutting some subset of the edges of $\mathbf{v}$ and reassigning a subset of nodes to a new district. That proposal is then accepted according to some probability determined by the characteristics of the map and the nodes cut, and the accepted or rejected map is then used as the starting partition for the next iteration of the algorithm. Finally, *Evolutionary Algorithm* methods (EA) declare an objective function defined by constraints such as compactness, contiguity, population parity, and other requirements and iteratively move towards solutions that reach maxima of the objective function. This procedure is repeated many times to create a distribution of plans. In the following sections, I discuss each of these methods in more detail, as well as their benefits and shortcomings.

3.2.1 Random-Seed-and-Grow Simulation Methods

RSG methods represent the first viable simulation tools for effectively simulating redistricting plans at scale. Initially introduced in Cirincione, Darling and O'Rourke (2000) to analyze the 1990 South Carolina redistricting, RSG was further popularized by the open-source BARD implementation for the R statistical programming language (Altman and McDonald, 2011) and the work of Chen and Rodden (2013)[2]. In the latter article, the authors use an RSG algorithm to simulate many redistricting plans for the state of Florida to argue that Democratic voters are inefficiently clustered in urban areas, leading to under-representation in the Florida congressional delegation relative to their population share as a whole. Furthermore, Chen has used the same method as an expert witness in judicial testimony for several court cases challenging redistricting plans (in *Rene Romo v. Rick Scott et al.* (2012), *League of Women Voters et al. v. Detzner et al.* (2015), and *League of Women Voters v. Wolf et al.* (2017), among others). More recently, Magleby and Mosesson (2018) introduce a number of computational improvements from the graph partitioning literature in computer science to improve the computational efficiency of the RSG algorithm.

The basic RSG algorithm for partitioning a graph $G = \{V, E\}$ into n precincts is as follows. First, n "seed" nodes are sampled from the node set V, where each seed node forms the basis for a separate district. Next, nodes adjacent to the district are gathered, and a single one is added to its adjacent seed at random. This grows the size of the district by 1. This process is repeated until each node is assigned to a district, which Magleby and Mosesson (2018) refer to as "multinodes." Finally, in order to respect population, compactness, or other thresholds, nodes resting on the boundary of two districts are iteratively and randomly swapped until the resulting

[2] An additional R implementation can be found in the redist.rsg() function in redist (Fifield, Tarr and Imai, 2015).

districts fall within the specified thresholds. This process is repeated until the desired number of samples is obtained.

RSG algorithms demonstrate several desireable properties. First, they are easy to understand — for the purposes of evidentiary value in court, the intuition behind the algorithms is easy to grasp. When the highest judge in the American judicial system is prone to referring to quantitative measures of gerrymandering as "sociological gobbledygook" (Chief Justice Roberts, oral arguments in *Gill v. Whitford*, October 3rd, 2017), the importance of effectively communicating redistricting simulation methodologies in court cannot be overstated. Second, RSG methods return a fully independent sample in each iteration. Other methods, particularly MCMC methods, take the solution from the previous iteration as the starting point for the next iteration, which lead to highly dependent samples that all look similar. RSG methods have no such issue. Third, they can easily incorporate additional constraints as thresholds. After the multinodes have been grown, RSG methods swap precincts into adjacent districts and evaluate whether a given set of thresholds have been hit. This set of thresholds can be as extensive and arbitrary as the researcher demands.

However, as noted by Fifield et al. (2018), the RSG algorithm is a heuristic algorithm with no theoretical guarantees. That is, the literature on RSG methods can provide no guarantees that the resulting sample from an RSG algorithm is a truly random set of draws from the distribution of valid redistricting plans, given a set of constraints. In a set of validation exercises using FL25 (Fifield et al., 2018, Figure 3), the authors show that RSG methods fail to return a representative sample of redistricting plans even in a simple redistricting problem where all possible solutions are known ex ante. Furthermore, these methods typically rely on rejection sampling methods, where after a certain number of iterations a draw is discarded if it has not yet satisfied all of the thresholds. This can be inefficient and cause the algorithm to

take a very long time to return the desired number of solutions if the thresholds are numerous and difficult to hit, as is often the case in applied redistricting problems.

While much of the early literature on RSG methods has not attempted to validate their simulation methods, Magleby and Mosesson (2018) advances the literature by conducting two separate validation exercises. First, the authors calculate the mean-median statistic (Wang, 2016) for every redistricting plan in FL25, as well as for the plans returned by their simulation method. They then compare the mean, median, standard deviation, and kurtosis for the true and simulated distributions of the statistic, and formally test them with the Kolmogorov-Smirnov test to show that the two distributions are substantively similar when population-parity thresholding at both $\pm 10\%$ and $\pm 1.5\%$. For the latter constraint, the authors correctly note that FL25 is paritcularly unbalanced — very few maps satisfy this tight population constraint, which is more reflective of true statutory requirements for implemented redistricting plans. Second, the authors apply their method to a 100 x 100 grid where each node has equal population, while imposing that both districts have equal population. They then show that horizontal splits in the grid are as likely as vertical splits, which should be expected if the method is sampling in an unbiased manner.

3.2.2 Markov chain Monte Carlo Simulation Methods

More recently, a number of scholars have used Markov chain Monte Carlo techniques to simulate redistricting plans. The basic framework, which was developed independently by Fifield et al. (2018) and Mattingly and Vaughn (2014), has been used in expert testimony in redistricting cases in North Carolina (*Common Cause v. Russo* (2017): Mattingly, 2017) and Pennsylvania (*League of Women Voters v. Wolf et al.* (2017): Pegden, 2017). The basic MCMC algorithm proceeds as follows: first, the algorithm starts at an existing partition $\mathbf{v}_0$ of the graph G. The algorithm then randomly cuts the remaining edges E according to a pre-specified probability — if

the probability is set to one, G is completely shattered. A swap between districts is then proposed by randomly selecting a set of connected node components on the boundary of two districts. Finally, that swap is accepted or rejected according to a specific probability, leading to a new partition $\mathbf{v}_1$. The entire process is then repeated starting from $\mathbf{v}_1$.

A major benefit of MCMC methods are its theoretical properties. The acceptance probability guarantees that the sample of plans will eventually converge to be a uniform sample from the target distribution defined by the user.[3] When no constraints are applied, this is a simple uniform sample from the set of contiguous partitions of G into n partitions. When constraints are applied, the target distribution becomes

$$g_\beta(\mathbf{v}) = \frac{1}{Z(\beta)} \exp\left(-\beta \sum_{V_k \in \mathbf{v}} \sum_{\ell=1}^{L} w_\ell \psi_\ell(V_k)\right)$$

where $\beta \in [0, 1]$ is a temperature parameter governing the strength of the constraint, and $Z(\beta)$ is a normalizing constant. $\psi_\ell(V_k)$ is the value of the ℓ'th constraint evaluated on the k'th partition — for example, how far district k is from population parity. Finally, w_ℓ is a user-specified weight assigned to each constraint — for example, if the user wants to upweight the strength of the population parity constraint ($\psi_{\text{pop}}(\cdot)$) over the compactness constraint ($\psi_{\text{comp}}(\cdot)$), they would increase w_{pop} and hold constant or decrease w_{comp}. Therefore, MCMC algorithms can be scaled to accomodate an arbitrary number of constraints, and if run for long enough, guarantee a representative sample of plans from the target distribution.

Despite these theoretical guarantees, it is impossible to test whether an MCMC algorithm has actually converged to the target distribution in practice. While researchers can rely on graphical checks such as traceplots, autocorrelation plots, and statistical tests such as the Gelman-Rubin diagnostic (Gelman and Rubin, 1992),

[3] Fifield et al. (2018) provides more detail on this in Section 2.2.

these tests and diagnostics are merely suggestive. Furthermore, MCMC methods may traverse the entire space of redistricting plans very slowly — unlike RSG algorithms, MCMC methods take the previously accepted plan as a starting point for the next iteration of simulations, and make iterated small changes. Therefore, the entire Markov chain may demonstrate severe autocorrelation, which is exacerbated when constraints are applied since it will become easier to get stuck in local modes. Although tools such as simulated tempering (Marinari and Parisi, 1992; Geyer and Thompson, 1995) and parallel tempering (Geyer, 1992) can reduce autocorrelation and allow the algorithm to transfer more easily between local modes, MCMC methods still struggle to scale to large redistricting problems.

MCMC algorithms have been subjected to a number of validation exercises, which have provided mixed evidence that they are able to scale to global problems. As mentioned previously, Fifield et al. (2018) constructed the FL25 data set in order to validate the performance of their algorithm on a small-scale problem, where it performed well under population constraints of various strength. The authors then subjected the algorithm to two state-level problems — global simulations of the New Hampshire map, where standard MCMC diagnostics suggested that the Markov chain had converged, and a more limited set of "local" simulations exploring the space of congressional district plans in Pennsylvania that look similar to the implemented plans. When applying their method to a global search of Pennsylvania, the Markov chain appeared not to have converged even after an extended run. In addition, Herschlag et al. (2018), who conducted a global search of North Carolina's redistricting plans, validate their MCMC algorithm by starting it from a number of different seed plans for North Carolina. They show that the resulting distributions are fairly similar, which they provide as evidence that their algorithm accurately samples from the target distribution.

3.2.3 Evolutionary Algorithm Methods

A new class of redistricting simulation methods based on evolutionary algorithms (EA) (Liu, Tam Cho and Wang, 2016; Tam Cho and Liu, 2016) use massive computational resources to efficiently draw large numbers of redistricting plans that satisfy a set of constraints. Evolutionary algorithms start with a definition of constraints that the algorithm must respect — for the redistricting problem, these often include population parity, contiguity, assignment of each unit to at most one district, and avoiding splitting communities of interest (among others). Given the constraints, an *objective* to maximize is then defined — this distinguishes EA methods from RSG and MCMC algorithms, in that each iteration of the algorithm technically searches for an optima on the defined metric subject to the predefined constraints. Finally, the EA algorithm repeatedly searches for solutions by first splitting up a single state graph G into a number of contiguous sub-maps that can be operated on in parallel. EA optimizes within each of these sub-graphs while using message-passing interface (MPI) operators to exchange solutions across maps and track solutions on the level of the full map. The stochastic nature of the algorithm ensures that it will not converge to a single solution — rather, running EA repeatedly on the same graph will return a distribution of solutions to the redistricting problem for a given state.

Given the huge computational resources they can harness[4], EA algorithms are exciting as a potential alternative method of drawing huge ensembles of redistricting plans subject to specified constraints. However, like RSG methods, EA methods are *heuristic* algorithms with no theoretical guarantees and without a defined target distribution of interest. Furthermore, to the best of our knowledge, no validation exercises of EA algorithms along the lines of Fifield et al. (2018) or Magleby and Mosesson (2018) exist. This lack of validation is further exacerbated by the huge

[4]Liu, Tam Cho and Wang (2016) run their algorithm on the BlueWaters supercomputer infrastructure and have scaled their algorithm up to run on 131,000 processors.

computational resources needed to run these algorithms — Liu, Tam Cho and Wang (2016) have optimized their method for the BlueWaters supercomputing resource at the University of Illinois at Urbana-Champaign, and furthermore no open-source implementation of their PEAR EA algorithm exists for adaptation and testing.

In sum, a wide variety of solutions for simulating large numbers of redistricting plans exist, but they differ in their theoretical guarantees, the set of validation exercises they have been subjected to, their scalability, and whether they have been released as open-source software for broader study and external validation. In the next section, I examine the primary validation data set for redistricting algorithms, FL25, discuss its shortcomings as a general validation solution, and then propose a new method for validating redistricting simulation algorithms.

3.3 Validation Exercises using FL25

Section 3.2.2 briefly discusses the FL25 data set and its use in validation exercises for redistricting simulators. This data set, which was introduced in Fifield et al. (2018), enumerates every valid partitioning of a 25-precinct subset of Florida's voting precincts into three congressional districts (a total of 117,688 possible plans that respect district contiguity). This data set has been used for validation exercises in a number of other papers and studies — as mentioned previously, Magleby and Mosesson (2018) compare simulated plans from their algorithm against the true distribution of FL25, and Cho (2017), in an expert report filed in *League of Women Voters v. Wolf et al.* (2017), uses FL25 to counter claims that RSG algorithms can sample uniformly from the underlying distribution of redistricting plans. Figure 3.1 shows the FL25 subset of Florida, as well as the distribution of precinct populations within the subset.

While FL25 improves on the current state of validation methods for redistricting simulation, it is still a single subset of electoral geography with its own idiosyncracies that do not necessarily generalize to other maps. As Magleby and Mosesson (2018)

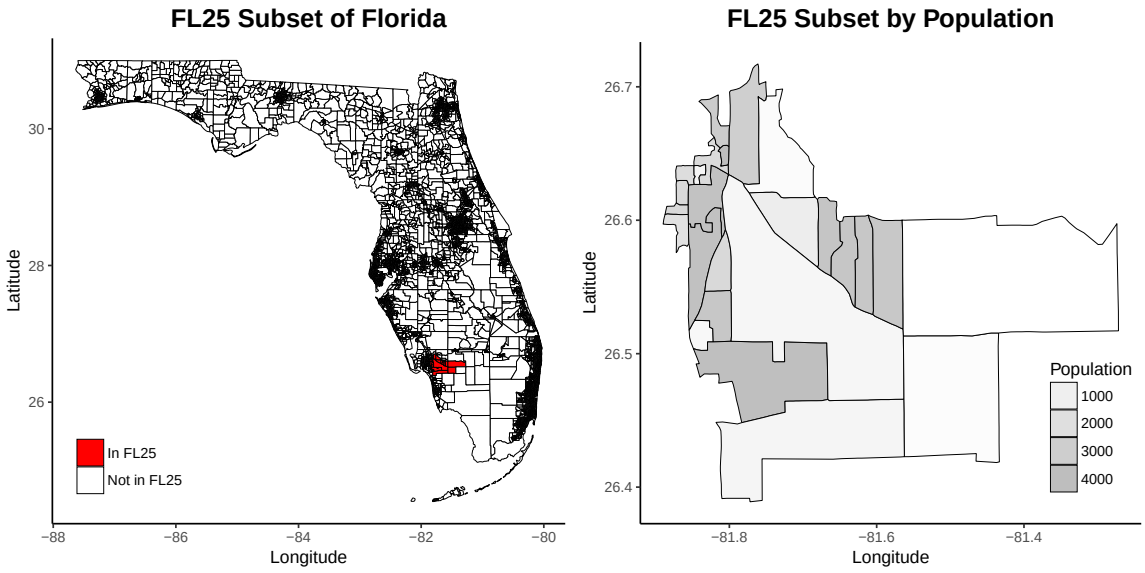

Figure 3.1: Maps of the FL25 subset. The left plot situates the FL25 subset in the larger map of Florida, while the right-hand map shows the precincts in FL25 shaded by precinct population.

note, FL25 is highly unbalanced — only 8 out of the 117,688 enumerated plans achieve standard levels of population parity ($\leq 1\%$). While it will be more difficult to obtain a balanced graph when the geographic units are more coarsened, it nonetheless raises the question of how representative FL25 is compared to other subsets of electoral geography in Florida.

I examine the representativeness of FL25 in Figure 3.2. To do so, I took the same baseline map of Florida precincts (6,688 precincts in total) and randomly sampled 300 separate sets of 25 adjacent precincts. For each set, I then enumerated every partition of each set into three contiguous sub-partitions. To enumerate the partitions quickly, I rely on the `enumpart` library described in Kawahara et al. (2017), who introduce a novel algorithm using zero-suppressed decision diagrams that improves the computational performance of enumeration algorithms for graphs. For each map, I then count the number of plans whose level of population parity is within 1% (distribution plotted on the left) and I estimate the standard deviation of precinct

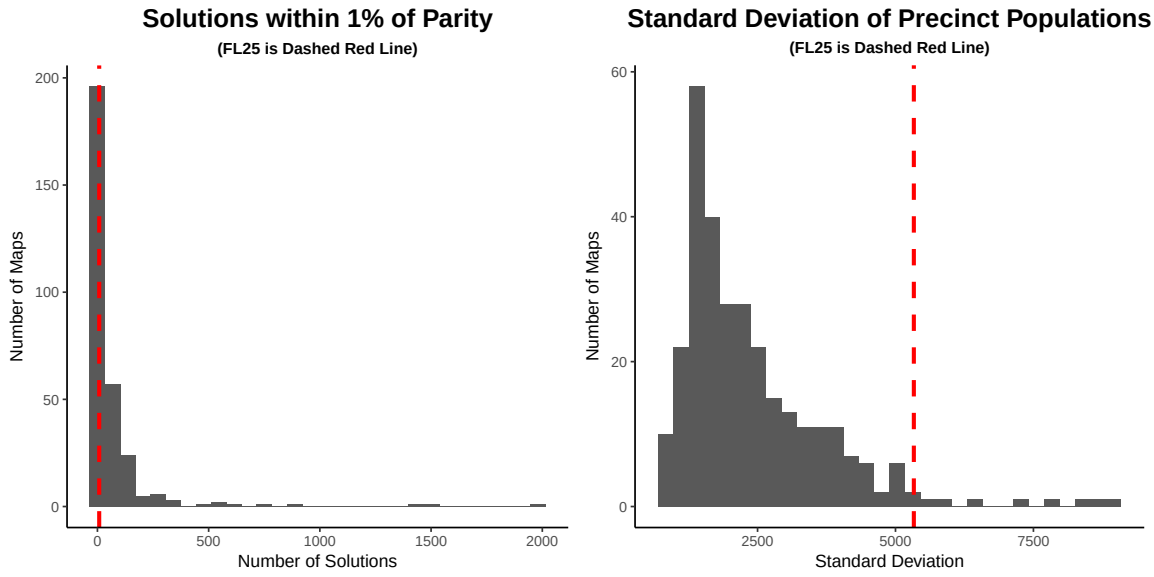

Figure 3.2: Plot of the distribution of the number of enumerated plans within 1% of population parity (left plot) and of the standard deviation of precinct populations in the 25-precinct test maps (right plot). Values for FL25 are plotted as dashed red lines. Distributions are the result of fully enumerating all three-district partitions of 300 independent 25-precinct maps, each drawn from Florida.

population (distribution plotted on the right). In both plots, the value for the FL25 subset is plotted as a dashed red line.

Both plots show that FL25 is somewhat unrepresentative compared to a random sample of similarly-sized subsets from the same Florida graph. More importantly, there is substantial variation in the traits of the solution space depending on the layout of the starting map. Focusing on population alone, I find that enumerated plans vary substantially in their balance — the degree to which the districtings for each enumerated map fall within standard levels of population parity (left plot). FL25 has on-average lower balance, such that 64% of maps have more enumerated plans within 1% of population parity. More striking, however, is the degree of overdispersion in the distribution — while the mean number of plans within population parity for any given map is 69.2, the variance is 36,317. FL25 appears to be even more of an outlier when looking at the standard deviation in precinct populations (right plot)

— out of the sampled maps, only 2.67% have a higher standard deviation of precinct populations than FL25.

While the validation exercises using FL25 or any other fully enumerated maps are an improvement over the status quo of no validation, validation on a single map may still reflect idiosyncracies of that particular set of geographies. In the following section, I show how to use improved enumeration algorithms to more rigorously test redistricting simulation algorithms.

3.4 General Tests for Evaluating Redistricting Simulation Methods

Here, I introduce a new method for validating ensembles of simulated redistricting plans that avoids relying on a single enumerated map that may be an unrepresentative subset of the state at large. The new validation method generates many fully enumerated maps and runs the simulation algorithm separately on each map, creating different ensembles on a wider variety of geographies. Statistical tests can compare the similarity of the simulated distribution to the enumerated, target distribution, which allows analysts to assess how close the simulated ensembles come to recovering the target distributions across a wide variety of geographies.

At its core, the new test repeats the validation exercise conducted in Figure 3 of Fifield et al. (2018) many times. First, a small test map is created, and every partition of that map into n contiguous sub-units is enumerated. Second, the redistricting algorithm is run for a set number of iterations on the test map. Third, a summary statistic of interest, such as the efficiency gap (Stephanopoulos and McGhee, 2015) or partisan symmetry (King and Browning, 1987) is calculated for every plan in the simulated ensemble and in the full enumerated set of plans. Finally, the two distributions are compared using a measure of similarity between two distributions, such as the Kolmogorov-Smirnov test, Kullback-Leibler distance, or Earth Mover's

distance. Once this procedure has been run k times, the distribution of the chosen similarity measure can be plotted as a means of assessing accuracy. I outline the validation procedure in full in Appendix Section B.1.

The computational bottleneck in this procedure is in Step 1, where every partition of a given sub-map into n connected components is enumerated. Existing enumeration methods, such as the redist.enumerate() function in the R package redistwhich relies on spanning tree representations of the underlying map, struggle to enumerate even small test sets quickly, although they improve on naïve approaches that enumerate every possible partition of the map and then check each one for contiguity. Here, I rely on the enumpart library introduced by Kawahara et al. (2017) to conduct the enumeration step quickly. enumpart combines efficient data structures called *zero-suppressed decision diagrams* (ZDD) with frontier-based search methods to quickly enumerate all partitions of a map into n contiguous components. enumpart also incorporates constraints such as population parity through weights applied to each node.

Figure 3.3 compares the runtime of the spanning tree enumeration algorithm implemented in redist against enumpart. The x-axis gives the number of geographic units being partitioned, while the y-axis gives the natural log of the runtime of the algorithm in seconds. Each boxplot gives the distribution of runtime across 50 trials. For both the two-district and three-district partitioning task, enumpart scales much more efficiently than redist.enumerate(). Furthermore, while enumpart can complete partitioning tasks of maps into more than three districts, redist.enumerate() struggles to enumerate solutions to these problems quickly. In general, I am agnostic as to the enumeration method used, other than recommending that the enumeration method that allows for the most naturalistic test maps possible be used. As more efficient enumeration methods are derived, allowing researchers to create test maps with more geographic units, I recommend researchers use those for validation exercises.

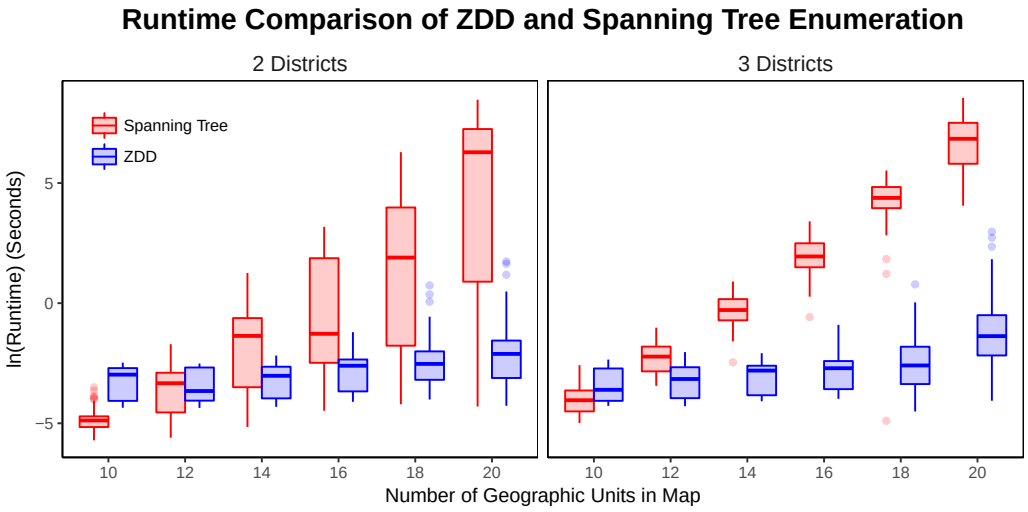

Figure 3.3: Runtime comparison of the spanning tree enumeration procedure, as implemented in redist.enumerate() (Fifield, Tarr and Imai, 2015), and the ZDD enumeration procedure, as implemented in the enumpart library (Kawahara et al., 2017). Each boxplot represents the method's runtime across 50 trials. As the number of geographic units of the underlying map increases, the spanning tree procedure scales increasingly poorly relative to the ZDD method.

Next, I illustrate the proposed validation procedure. The underlying map for this exercise is Pennsylvania's 9,059 voting precincts, which are divided into 18 congressional districts. I first sample 300 independent subsets of the map of 20 precincts each. Using enumpart, I enumerate for each small map every partition of that map into two contiguous congressional districts. I then run both the random-seed-and-grow algorithm, as implemented in the redist.rsg() function in redist, and an MCMC algorithm as implemented in redist.mcmc(). Next, I calculate the Republican dissimilarity index (Massey and Denton, 1988) as a substantive quantity of interest for each enumerated and simulated redistricting plan for each map.[5] Last, I compare

[5]The dissimilarity index (Massey and Denton, 1988), which is a common measure of segregation, is defined as

$$D = \sum_{i=1}^{n} \frac{t_i |p_i - P|}{2TP(1-P)}$$

where i indexes n areal units such as precincts, p_i is the share of areal unit i that identifies as a particular group, T is the total population across all areal units, and P is the share of a group across all areal units. Substantively, it can be interpreted as the share of a particular group that would have to be moved in order to be evenly distributed across all n areal units.

the distribution of the dissimilarity index in the set of simulated plans against the distribution of the dissimilarity index in the fully enumerated set of plans using the Kolmogorov-Smirnov (KS) test, which is a nonparametric test of the null hypothesis that two samples are drawn from the same distribution.

For the MCMC algorithms run using simulated tempering, I tune the algorithms using the following procedure. First, I enumerate a set of weights for w_{pop} as described in Section 3.2.2 that upweight draws closer to population parity, as well as a range of transition weights that favor draws which constrain more aggressively. I then run each parameter combination for 50,000 iterations. Finally, among the set of parameter combinations where the Metropolis-Hastings acceptance probability is between 20% and 40%, I select the parameter combination where the temperatures appear to be the closest to uniformly distributed. This is similar to tuning procedures in full-state substantive applications, where researchers have to look at proxy measures such as the Metropolis-Hastings acceptance probability, trace plots, and Gelman-Rubin Rhat measures to assess a well-performing markov chain. Note that none of the decisions involve selecting a parameter combination based on some measure of distributional similarity when compared to the true distribution, which is not possible in applications on the scale of a full state.

Figure 3.4 plots the results of this test. Each panel is a QQ-plot that compares the distribution of p-values from each KS test against the uniform distribution under different levels of required population parity (from left to right: no constraint, 20%, 10%). Under the null hypothesis that each simulated ensemble is a random sample from the target distribution, the p-value from the KS test should be uniformly distributed between 0 and 1, which in the QQ-plot looks as though the points fall close to the 45-degree line. Across all three levels of population parity, the MCMC algorithms outperform the RSG algorithms by a wide margin. For the RSG algorithms, the KS p-values are tightly clustered at 0, which is a rejection of the null hypothesis that

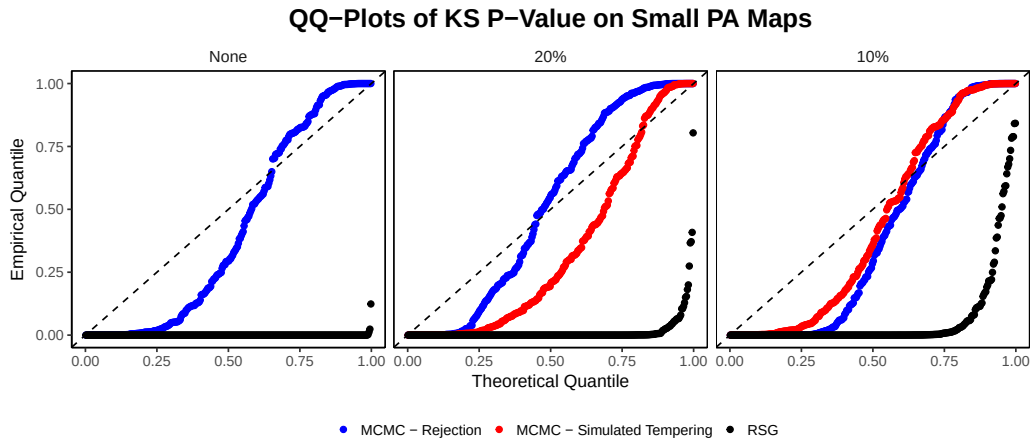

Figure 3.4: Results from the proposed validation procedure at three different population parity levels. The left-most panel plots the results of the validation when no population parity level is applied, the center panel plots the results when a population parity measure of 20% is applied, and the right panel applies a population parity threshold of 10%. In all cases, the distribution of the KS p-value is much closer to the uniform distribution for MCMC algorithms than for the RSG algorithm.

the simulated distribution is drawn randomly from the target distribution. For both MCMC algorithms, the KS p-values are not as clustered at 0 or 1, but they are are also not distributed fully uniformly, indicating some degree of bias in the simulated set of plans. However, they show substantial improvements over RSG algorithms at every level of population parity.

One downside of using KS tests for the validation metric is their sensitivity — even small deviations from the target distribution in the sample that would even out with more samples can lead to a rejection of the null hypothesis. This is because the KS test, which is defined as

$$D_n = \sup_x |F_n(x) - F(x)|,$$

searches for the point of the greatest difference between the cumulative distribution function of the sample ($F_n(x)$) and the target cumulative distribution function ($F(x)$) and tests for a significant difference at that point. Researchers can use different

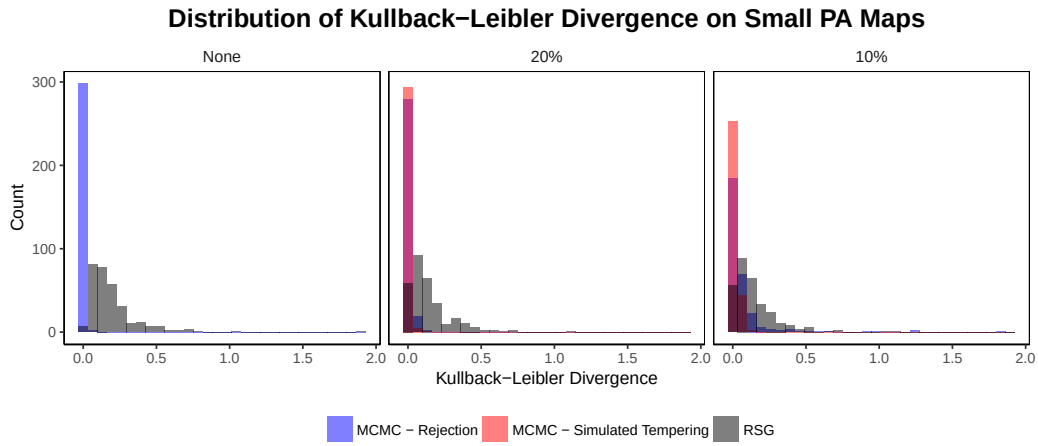

Figure 3.5: Results from the proposed validation procedure at three different population parity levels, using the Kullback-Leibler divergence measure. The left-most panel plots the results of the validation when no population parity level is applied, the center panel plots the results when a population parity measure of 20% is applied, and the right panel applies a population parity threshold of 10%. In all cases, the distribution of the Kullback-Leibler divergence is much closer to 0 for the MCMC algorithms than for the RSG algorithms.

measures of distribution similarity in order to avoid this sensitivity. One possible measure is the Kullback-Leibler divergence, which is defined as

$$D_{KL}(S\|T) = \int_{-\infty}^{\infty} s(x)\log\frac{s(x)}{t(x)}dx$$

where S indicates the distribution of the sample and $s(x)$ the density of S, while T indicates the distribution of the target and $t(x)$ its density. When S and T are the same, $D_{KL}(S\|T)$ is equal to 0, and greater values indicates a larger difference between two distributions.

In Figure 3.5, I conduct the procedure proposed above, but instead of conducting a KS test comparing the Republican dissimilarity indices from the ensemble against the target distribution, I compare them using Kullback-Leibler divergences. Again, under all levels of population parity, the MCMC algorithms outperform the RSG algorithm, which is indicated by the tight clustering of the Kullback-Leibler divergences for

the MCMC algorithms around 0. In contrast, the RSG algorithm exhibits greater divergence from the target distribution under repeated trials, as indicated by the wider range of the distribution of the divergences. Other measures of distributional similarity, such as Earth Mover's distance, can be used to test the robustness of any results found using a single measure of similarity. I also recommend using multiple continuous summary measures to assess robustness — for instance, partisan symmetry (King and Browning, 1987), the efficiency gap (Stephanopoulos and McGhee, 2015), or electoral competitiveness (Tam Cho and Liu, 2016) can all be quickly calculated and tested in the same way once the ensembles have been run.

3.5 Conclusion

I introduce a simple validation test for simulated redistricting ensembles to assess whether a sample of redistricting plans is an accurate representation of the true distribution of redistricting plans on different sets of geographies. The method relies on recent advances in enumeration methods to quickly find all contiguous partitions of a given map into n connected components, overcoming previous computational bottlenecks that limited researchers to small test maps. In addition, I caution against validation exercises that rely on enumeration results from a single map by showing how an open-source validation data set, FL25, deviates relative to a random subset of equally-sized maps drawn from the same map of Florida. While none of these tests can provide *positive* evidence that a given simulation method can uniformly sample redistricting plans for a full state, researchers should be skeptical that simulation methods which fail these small-map tests can scale to full-state redistricting problems.

I note that in general, larger validation data sets will always be better for the purpose of naturalistic comparison and validation of simulated redistricting plan ensembles. That is, as enumeration algorithms improve and allow researchers to enumerate every partition of increasingly larger sets of geographic units into more connected

components, I urge researchers to use those new methods for the validation step of this test. Stated somewhat differently, the smaller the validation test sets that can be used for validation, the less weight we should place on evidence derived from ensembles of simulated plans. Furthermore, as these methods continue to scale to larger redistricting problems, they can also be used to test more substantive questions about redistricting. For example, while the Stirling number representation of the number of possible partitions of a map into n congressional districts suggests an intractable number of solutions for enumeration, common legal and political constraints such as population parity, compactness, preservation of communities of interest, and status quo bias will reduce the solution space by many orders of magnitude. Improved enumeration methods can help us understand how much that solution space decreases with additional statutory and political constraints, while also helping to map out the multimodality of the true solution space.

Lastly, I urge researchers, practitioners, and activists to continue pushing for open and transparent evaluations of redistricting simulation methods. As the 2020 Census passes, lawsuits challenging proposed redistricting plans will inevitably be brought to court, and simulation evidence will be used to challenge and defend those plans. It is necessary that the simulation methods used to create that evidence be rigorously evaluated, and this paper introduces what will hopefully be one of many complementary validation tests used to ensure that this evidence is of the highest possible quality.

Chapter 4

fastLink: R Package for Fast Probabilistic Record Linkage[1]

4.1 Introduction

Modern social science research often relies on innovative combinations of survey, administrative, and textual data sources in order to advance new claims and knowledge about the world. Merging can be a trivial task if there exists a unique identifier that unambiguously identifies records, in which case the `merge()` function in R can seamlessly and quickly conduct a 1-to-1, 1-to-many, or many-to-1 merge on the unique identifier. Unfortunately, in practice, this unique identifier is rarely available. Applied researchers have previously relied on deterministic algorithms, which suffer from an inability to flexibly handle missing data and measurement error, or proprietary merging algorithms, which prevent the merging process from being incorporated transparently into the replication process. Furthermore, none of the deterministic or proprietary algorithms allow the researcher to quantify the uncertainty inherent in the merging process.

[1] Portions of this chapter are adapted from joint work with Ted Enamorado and Kosuke Imai in Enamorado, Fifield and Imai (2017).

In this paper, we introduce a new R package, fastLink (Enamorado, Fifield and Imai, 2017) for conducting data merges. The fastLink package includes: (1) a fast implementation of the canonical Fellegi-Sunter (Fellegi and Sunter, 1969) probabilistic record linkage (PRL) model (2) a series of utilities for flexibly clustering, preparing, adjusting, and summarizing data merges (3) extensions that relax the core assumptions of the Fellegi-Sunter model, and methods to incorporate auxiliary information about migration to inform merges of data sets across time and space and (4) easy parallelization and automatic hashing of merging information in order to quickly merge data sets in a memory-efficient manner. We focus here on the demonstration of the functionalities included in the fastLink package. The statistical theory underlying the procedures we demonstrate here can be found in Enamorado, Fifield and Imai (2017).

The fastLink package is freely available for download through the Comprehensive R Archive Network (CRAN) at https://cran.r-project.org/package=fastLink and can be installed using the standard syntax for installing an R package:

```
install.packages("fastLink")
```

where users may be prompted to select a CRAN mirror from which the package will be downloaded. This step needs to be done only once (unless one wishes to update fastLink to a new version).

In the next section, we provide an overview of the fastLink package, including the computational implementation and a summary of our improvements on the Fellegi-Sunter model to incorporate aggregate migration information. We then describe how to conduct a basic data merge (Section 4.3) using fastLink along with methods for analyzing the results of the merge, methods for incorporating auxiliary information to inform the merge and post-processing of merged datasets (Section 4.4), and functionalities included for pre-processing and clustering data in preparation for a merge (Section 4.5). In Section 4.6, we apply fastLink to estimate rates of party switching

among local politicians in Rio de Janeiro, Brazil between 2012 and 2016 to illustrate a principled record linkage workflow. Finally, Section 4.7 concludes.

4.2 Overview of the fastLink Algorithm

4.2.1 The Model and Assumptions

Before describing the functions available in fastLink, we briefly describe the statistical model and assumptions implemented in our software. Following the canonical model of record linkage proposed by Fellegi and Sunter (1969), we model the probability of two records, (record i in dataset $\mathcal{A}$ and record j in dataset $\mathcal{B}$) matching using the following finite mixture model,

$$\gamma_k(i,j) \mid M_{ij} = m \overset{\text{indep.}}{\sim} \text{Discrete}(\boldsymbol{\pi}_{km}) \tag{4.1}$$

$$M_{ij} \overset{\text{i.i.d.}}{\sim} \text{Bernoulli}(\lambda) \tag{4.2}$$

where the latent mixing variable M_{ij} denotes whether i and j are a match. For variable k, the vector $\boldsymbol{\pi}_{km}$ denotes the probability of each agreement level for that variable, conditional on M_{ij}, and λ is the probability of a match across all pairwise comparisons.

The model relies on two independence assumptions, which are reviewed in more detail in Enamorado, Fifield and Imai (2017). First, we assume that the latent mixing variable M_{ij} is independently and identically distributed. Second, we assume conditional independence across the k linkage variables conditional on match/non-match status.

In addition to the two independence assumptions, we follow Sadinle (2014) and assume the data are Missing At Random (MAR) conditional on M_{ij}. This avoids ad hoc procedures to handling missing values in the data, such as recoding all missing

data as disagreements. The MAR assumption also allows us to simply ignore missing data, leading to a tractable form for the likelihood function.

4.2.2 The EM Algorithm

Following Winkler (1988), we apply the expectation and maximization (EM) algorithm, which is an iterative procedure, to estimate the model parameters (Dempster, Laird and Rubin, 1977). Under the modeling assumptions described in Section 4.2.1, the complete-data likelihood function is given by,

$$\mathcal{L}_{com}(\lambda, \boldsymbol{\pi} \mid \boldsymbol{\gamma}, \boldsymbol{\delta}) \propto \prod_{i=1}^{N_{\mathcal{A}}} \prod_{j=1}^{N_{\mathcal{B}}} \prod_{m=0}^{1} \left\{ \lambda^m (1-\lambda)^{1-m} \prod_{k=1}^{K} \left(\prod_{\ell=0}^{L_k-1} \pi_{km\ell}^{\mathbf{1}\{\gamma_k(i,j)=\ell\}} \right)^{1-\delta_k(i,j)} \right\}^{\mathbf{1}\{M_{ij}=m\}}$$

where $N_{\mathcal{A}}$ and $N_{\mathcal{B}}$ indicate the number of records in dataset A and B, respectively, $\delta_k(i,j)$ indicates whether observation i or j is missing information on variable k, and where M_{ij} is unobserved.

Given this complete-data likelihood function, the E-step is given by

$$\begin{aligned} \xi_{ij} &= \Pr(M_{ij} = 1 \mid \boldsymbol{\delta}(i,j), \boldsymbol{\gamma}(i,j)) \\ &= \frac{\lambda \prod_{k=1}^{K} \left(\prod_{\ell=0}^{L_k-1} \pi_{k1\ell}^{\mathbf{1}\{\gamma_k(i,j)=\ell\}} \right)^{1-\delta_k(i,j)}}{\sum_{m=0}^{1} \lambda^m (1-\lambda)^{1-m} \prod_{k=1}^{K} \left(\prod_{\ell=0}^{L_k-1} \pi_{km\ell}^{\mathbf{1}\{\gamma_k(i,j)=\ell\}} \right)^{1-\delta_k(i,j)}} \end{aligned} \tag{4.3}$$

where the posterior probability of being a true match is computed for each pair given the current values of model parameters. Using this posterior match probability, the M-step can be implemented as follows,

$$\lambda = \frac{1}{N_{\mathcal{A}} N_{\mathcal{B}}} \sum_{i=1}^{N_{\mathcal{A}}} \sum_{j=1}^{N_{\mathcal{B}}} \xi_{ij} \tag{4.4}$$

$$\pi_{km\ell} = \frac{\sum_{i=1}^{N_{\mathcal{A}}} \sum_{j=1}^{N_{\mathcal{B}}} \mathbf{1}\{\gamma_k(i,j) = l)\}(1-\delta_k(i,j))\xi_{ij}^m (1-\xi_{ij})^{1-m}}{\sum_{i=1}^{N_{\mathcal{A}}} \sum_{j=1}^{N_{\mathcal{B}}} (1-\delta_k(i,j))\xi_{ij}^m (1-\xi_{ij})^{1-m}} \tag{4.5}$$

Then with a suitable set of starting values, we repeat the E-step and M-step until convergence. When setting the starting values for the model parameters, we impose inequality constraints based on the following two ideas: (1) the set of matches is strictly smaller than the set on non-matches $\lambda \ll 1-\lambda$, and (2) for binary comparisons, we have $\pi_{k10} \ll \pi_{k11}$ and $\pi_{k01} \ll \pi_{k00}$ for each k (Jaro, 1989; Winkler, 1993; Sadinle and Fienberg, 2013). The latter implies that agreement (disagreement) is more likely among matches (non-matches).

4.2.3 Hashing for Efficient Memory Management

While the EM algorithm described above is relatively simple, we find that existing implementations are computationally inefficient (see Enamorado, Fifield and Imai (2017) for more details). To overcome this challenge, we develop a computationally efficient implementation of the EM algorithm. First, for implementing the E-step, notice that the posterior match probability given in equation (4.3) takes the same value for two pairs if their agreement patterns are identical. For the sake of illustration, consider a simple example where two variables are used for merging, i.e., $K = 2$, and binary comparison is made for each variable, i.e., $L_k = 2$. Under this setting, there are a total of nine agreement patterns: $(\texttt{0}, \texttt{0})$, $(\texttt{0}, \texttt{1})$, $(\texttt{1}, \texttt{0})$, $(\texttt{1}, \texttt{1})$, $(\texttt{NA}, \texttt{0})$, $(\texttt{NA}, \texttt{1})$, $(\texttt{0}, \texttt{NA})$, $(\texttt{1}, \texttt{NA})$, and $(\texttt{NA}, \texttt{NA})$ where `1` and `0` represent agreement and disagreement, respectively while `NA` represents a missing value. Then, for instance, the posterior match probability for $(\texttt{0}, \texttt{1})$ is given by $\lambda\pi_{110}\pi_{211}/\{\lambda\pi_{110}\pi_{211} + (1-\lambda)\pi_{100}\pi_{201}\}$ whereas that for $(\texttt{1}, \texttt{NA})$ is equal to $\lambda\pi_{111}/\{\lambda\pi_{111} + (1-\lambda)\pi_{101}\}$. If all comparison values are missing, e.g., $(\texttt{NA}, \texttt{NA})$, then we set the posterior match probability to λ. Thus, the E-step can be implemented by computing the posterior match probability for each of the *realized* agreement patterns. Often, the total number of realized agreement patterns is much smaller than the number of all *possible* agreement patterns.

Second, the M-step defined in equations (4.4) and (4.5) requires the summation of posterior match probabilities across all pairs or their subset. Since this probability is identical within each agreement pattern, all we have to do is to count the total number of pairs that have each agreement pattern. We use the following hash function for efficient counting,

$$\mathbf{H} \;=\; \sum_{k=1}^{K} \mathbf{H}_k \quad \text{where} \quad \mathbf{H}_k \;=\; \begin{bmatrix} h_k^{(1,1)} & h_k^{(1,2)} & \dots & h_k^{(1,N_{\mathcal{B}})} \\ \vdots & \vdots & \ddots & \vdots \\ h_k^{(N_{\mathcal{A}},1)} & h_k^{(N_{\mathcal{A}},2)} & \dots & h_k^{(N_{\mathcal{A}},N_{\mathcal{B}})} \end{bmatrix} \tag{4.6}$$

where $h_k^{(i,j)} \;=\; \mathbf{1}\left\{\gamma_k(i,j) > 0\right\} 2^{\gamma_k(i,j) + \mathbf{1}\{k>1\} \times \sum_{e=1}^{k-1}(L_e - 1)}$. The matrix $\mathbf{H}_k$ maps each pair of records to a corresponding agreement pattern in the kth variable that is represented by a unique hash value based on the powers of 2. These hash values are chosen such that the matrix $\mathbf{H}$ links each pair to the corresponding agreement pattern across K variables.

Since an overwhelming majority of pairs are not true matches, most elements of the $\mathbf{H}_k$ matrix are zero. As a result, the $\mathbf{H}$ matrix also has many zeros. In our implementation, we utilize sparse matrices whose lookup time is $O(T)$ where T is the number of unique agreement patterns observed. In most applications, T is much less than the total number of possible agreement patterns, i.e., $\prod_{k=1}^{K} L_k$. This hashing technique is applicable if the number of variables used for merge is moderate. If many variables are used for the merge, approximate hashing techniques such as min hashing and locally sensitive hashing are necessary.

4.2.4 Reverse Data Structures for Field Comparisons

The critical step in record linkage is to compare pairs of records across the K fields used to link two datasets, which is often regarded as the most expensive step in terms of computational time (Christen, 2012). To do so, for each linkage field k,

we first compare observation i of dataset $\mathcal{A}$ and j from dataset $\mathcal{B}$ via a pre-defined distance metric (e.g., Jaro-Winkler for string-valued fields) and obtain a value which we call $S_k(i,j)$. However, comparisons in the Fellegi-Sunter model are represented in terms of a discrete agreement levels per linkage field, not a continuous measure of agreement as the one implied by the distance metric. In other words, we need a discrete representation of $S_k(i,j)$. Specifically, if we have a total of L_k agreement levels for the kth variable, then,

$$\gamma_k(i,j) \quad = \quad \begin{cases} 0 & \text{if } S_k(i,j) \leq \tau_0 \\ 1 & \text{if } \tau_0 < S_k(i,j) \leq \tau_1 \\ \vdots & \\ L_k - 1 & \text{if } \tau_{L_k-2} < S_k(i,j) \leq \tau_{L_k-1} \end{cases} \tag{4.7}$$

where $\gamma_k(i,j)$ represents the agreement level between the values for variable k for the pair (i,j) and $\boldsymbol{\tau} = \{\tau_0, \tau_1, \ldots, \tau_{L_{k-1}}\}$ the set of predetermined thresholds use to define the agreement levels. For example, to compare names and last names, some authors such as Winkler (1990) argue in favor of using the Jaro-Winkler string distance to produce S_k, where one could use $\boldsymbol{\tau} = \{0.88, 0.94\}$ to construct γ_k for three agreement levels.

Still the problem with constructing γ_k is that the number of comparisons we have to make is often large. In our proposed implementation we exploit the following characteristics of typical record linkage problems in social sciences:

- The number of unique values observed in each linkage field is often less than the number of observations in each dataset. For example, consider a variable such as first name. Naively, one may compare the first name of each observation in dataset $\mathcal{A}$ with that of every observation in $\mathcal{B}$. In practice, however, we can reduce the number of comparisons by considering only unique first name that

appears in each data set. The same trick can be used for all linkage fields by focusing on the comparison of the unique values of each variable.

- For each comparison between two unique first names ($name_{1,\mathcal{A}}$ and $name_{1,\mathcal{B}}$), for example, we only keep the indices of the original datasets and store them using what is often referred as a reverse data structure in the literature (Christen, 2012). In such an arrangement, a pair of names ($name_{1,\mathcal{A}}$, $name_{1,\mathcal{B}}$) becomes a key with two lists, one containing the indices from dataset $\mathcal{A}$ that have a first name equal to $name_{1,\mathcal{A}}$, and another list that does the same for $name_{1,\mathcal{B}}$ in dataset $\mathcal{B}$.

- Comparisons involving a missing value need not be made. Instead, we only need to store the indices of the observations in $\mathcal{A}$ and $\mathcal{B}$ that contain missing information for field k.

- Since the agreement levels are mutually exclusive, we use the lowest agreement level as the base category. Once a set of threshold values has been defined, then a pair of names can only be categorized in one of the L_k agreement levels. The indices for the the pairs of values that can be categorized as disagreements (or nearly disagreements) do not need to be stored. For most variables, disagreement is the category that encompasses the largest number of pairs. Thus, our reverse data structure lists become quite sparse. This sparsity can be exploited by the use of sparse matrix, yielding a substantially memory efficient implementation.

Together, these improvements make fastLink more efficient in runtime and memory management than other existing open-source implementations of the Fellegi-Sunter record linkage model and allow it to scale to larger problems.

4.2.5 Parallelization and Random Sampling

Under the proposed probabilistic modeling approach, a vast majority of the computational burden is due to the enumeration of agreement patterns. In fact, the actual computation time of implementing the E and M steps, once hashing is done, is fast even for large data sets. Therefore, for further computational efficiency, we parallelize the enumeration of agreement patterns. Specifically, we take a divide-and-conquer approach by partitioning the two data sets, $\mathcal{A}$ and $\mathcal{B}$, into equally-sized subsets such that $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_{M_\mathcal{A}}\}$ and $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{M_\mathcal{B}}\}$. Then, using OpenMP, we count the agreement patterns for each partition pair $\{\mathcal{A}_i, \mathcal{B}_j\}$ in parallel using the hash function given in equation (4.6). As explained above, we utilize sparse matrix objects to efficiently store agreement patterns for all pairwise comparisons. Finally, the entire pattern-counting procedure is implemented in C++ for additional performance gains. Taken together, our approach provides simple parallelization of the pattern-counting procedure and efficient memory usage so that our linkage procedure can be applied to arbitrarily large problems.

Another advantage of the probabilistic modeling approach is the use of random sampling. Since the number of parameters, i.e., λ and $\boldsymbol{\pi}$, is relatively small, we can efficiently estimate these parameters using a small random subset. Once we obtain the parameter estimates, then we can compute the posterior match probabilities for every agreement pattern found in the entire data sets in parallel. In this way, we are able to scale the model to massive data sets.

4.3 Conducting Data Merges using fastLink

The fastLink package consists of several main functions as well as various methods for summarizing output from these functions (e.g., `plot()` and `confusion()`). Figure 4.1 illustrates the core structure of the fastLink package.

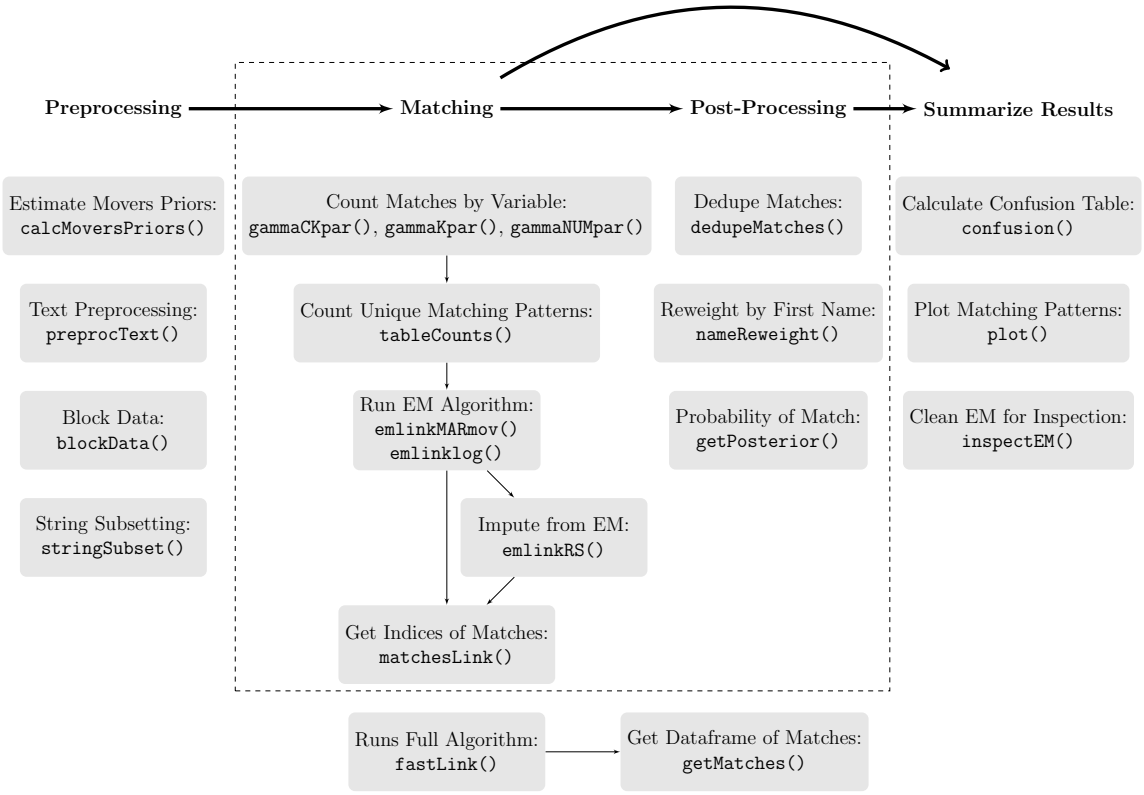

Figure 4.1: Core structure of the fastLink package as of version 0.4.1.

The core of the algorithm can be run using the `fastLink()` wrapper, which we will describe in more detail below. This includes utilities to count unique patterns, run the EM algorithm, and adjust the estimated matching probabilities by deduping and through auxiliary information such as name frequency. Each of these steps can also be run separately through the functions listed in the utilities. fastLink also includes utilities to prepare data for a merge by harmonizing fields and data blocking, as well as utilities for summarizing and inspecting data merges.

4.3.1 A Small-Scale Example

The `fastLink()` function takes two data sets and a simple description of the merging fields, and returns the results of the data merge which can then be passed to the `plot()` and `confusion()` functions for inspection and summarization. We illustrate the use of the function using a small-scale example, which is included in the fastLink package. The two data sets, `dfA` and `dfB`, are samples from the California voter file which are then shuffled within-field to preserve anonyminity. There are 50 true matches across the two data sets, and a brief description of the fields in the two data sets can be found by running

```
?dfA
?dfB
```

Here, we run a simple match using `fastLink()` to merge `dfA` and `dfB`:

```
fl_out <- fastLink(
  dfA = dfA, dfB = dfB,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear")
)
```

The first two arguments in `fastLink()` are where the user specifies the two data sets to link. The only other required argument is 'varnames', where the user specifies the variables shared between the two data sets that should be used to match on.

Without any other arguments, `fastLink()` will compare each specified variable using a simple match/non-match criterion, where any pair on a variable that does not exactly match will be declared a non-match. The output object `fl_out` contains the following objects:

```
names(fl_out)
```

- `matches`: A matrix with two columns and a row for each matched pair above the specified threshold. The first column, `inds.a`, gives the row numbers of the successfully matched observations in datset A, while the second column `inds.b` gives the row numbers of successfully matched observations in dataset B. The user can recover subsetted data frames by specifying `return.df = TRUE` in `fastLink()`, or by using the function `getMatches()`.
- `EM`: Parameter estimates from the Expectation-Maximization algorithm, which are used to calculate the confusion table and visualize the quality of different matching patterns.
- `patterns`: The matching patterns for each match variable, sorted to correspond to the pairs in `matches`. Here, 2 indicates an exact match, 0 indicates a non-match, 1 indicates a partial match, and `NA` indicates a case where one or both of the observations had a missing value.
- `posterior`: The posterior matching probability for each matched pair, sorted to correspond to the pairs in `matches`.
- `nobs.a`, `nobs.b`: The number of observations in dataset A and dataset B, respectively.

As mentioned above, the user can recover a data frame of the successfully matched observations using the `getMatches()` function. The resulting data frame contains the

union of all unique column names between the two merged data frames, as well as the match patterns and the posterior matching probability, as shown below:

```
matched_dfs <- getMatches(dfA = dfA, dfB = dfB,
                          fl.out = fl_out, threshold.match = 0.85)
```

The first two arguments to `getMatches()` are the data frames that we matched in the call to `fastLink()` above, while the third argument (`fl.out`) is the resulting `fastLink` object containing the EM results, the matched indices, the posterior matching probabilities for each pair of observations, and the full set of match patterns for all pairs. Finally, we specify the threshold above which we declare a pair a match using the `threshold.match` argument — this is the ξ_{ij} parameter introduced in Equation 4.3, which is the posterior probability of being in the matched set conditional on the observed data. Here, if the estimated value of ξ_{ij} is above 0.85 for a given pair, we declare the pair a true match and return it as part of the final matched data frame.

We can also summarize the merge using the `confusion()` function.

```
confusion(fl_out, threshold = 0.85)

## $confusion.table
##                      'True' Matches 'True' Non-Matches
## Declared Matches               50.0                0.0
## Declared Non-Matches            0.3              299.7
##
## $addition.info
##                                  results
## Max Number of Obs to be Matched  350.00
## Sensitivity (%)                   99.40
## Specificity (%)                  100.00
## Positive Predicted Value (%)     100.00
## Negative Predicted Value (%)      99.90
## False Positive Rate (%)            0.00
## False Negative Rate (%)            0.60
## Correctly Classified (%)          99.91
## F1 Score (%)                      99.70
```

`confusion()` outputs a series of summary statistics about the quality of the data merge, using the parameters estimated by the EM algorithm. The confusion table estimates the number of true matches, true non-matches, and mis-classified matches and non-matches using the posterior matching probability ξ_{ij}. For example, to estimate the false positive rate and false negative rate using a posterior match threshold of 0.85, the function calculates

$$\text{False Positive Rate: } \Pr(M_{ij} = 0 \mid \xi_{ij} \geq 0.85) = \frac{\sum_{i=1}^{N_\mathcal{A}} \sum_{j=1}^{N_\mathcal{B}} \mathbf{1}\{\xi_{ij} \geq 0.85\}(1 - \xi_{ij})}{\sum_{i=1}^{N_\mathcal{A}} \sum_{j=1}^{N_\mathcal{B}} \mathbf{1}\{\xi_{ij} \geq 0.85\}}$$

$$\text{False Negative Rate: } \Pr(M_{ij} = 1 \mid \xi_{ij} < 0.85) = \frac{\sum_{i=1}^{N_\mathcal{A}} \sum_{j=1}^{N_\mathcal{B}} \xi_{ij} \mathbf{1}\{\xi_{ij} < 0.85\}}{\lambda N_\mathcal{A} N_\mathcal{B}}$$

The function returns a number of other summary metrics about the quality of the match, including sensitivity, specificity, the F1 score, and the percent of observations correctly classified as matches and non-matches.

Finally, we can visualize the matching patterns using the `plot()` function, as follows:

```
plot(fl_out, posterior.range = c(0.85, 1))
```

The function simply takes the output of `fastLink()` and visualizes the matching patterns within the specified posterior matching probability range. The top row shows the matching patterns for the observations with the highest posterior matching probability, and that those observations are exact matches on all six variables we have included in the match. The second row shows the pattern with the second-highest posterior match probability — pairs with this matching pattern exact-match on all variables except for `birthyear`, where one or both observations are missing information there. The visualization also shows partial matches for string-distance and numeric-distance match patterns, which we will cover in more detail in the next section, as well as variables where patterns do not match.

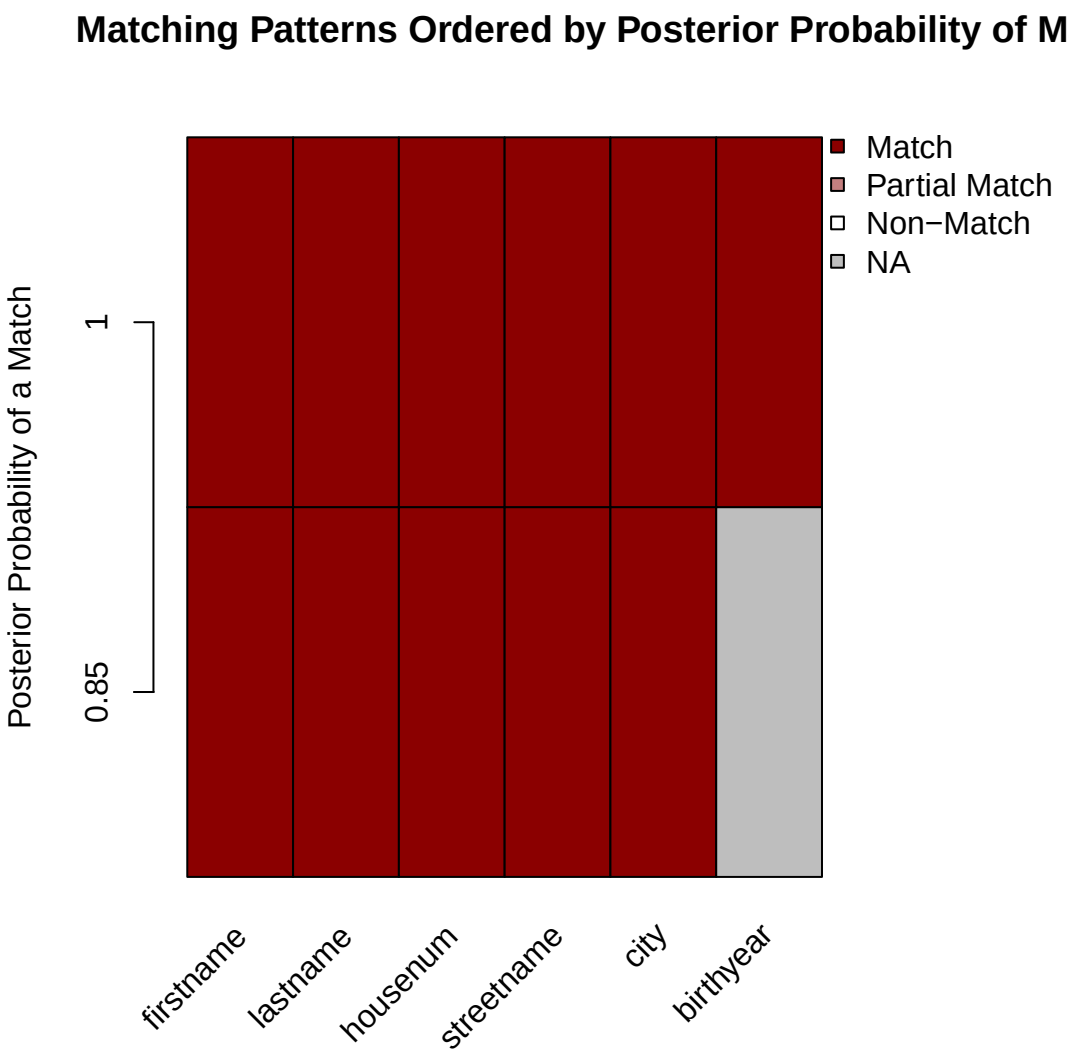

Figure 4.2: Plot of matching patterns using posterior match probabilities from fastLink. The x-axis indicates different covariates used for the match, each row on the y-axis is a different matching pattern ordered by posterior match probability, and the colors indicate different matching categories.

4.3.2 Alternative Methods of Constructing Matching Patterns

In the example above, we matched two data sets on six variables using exact matching on each variable. However, misspellings and administrative errors may cause exact matching to declare a variable pair a non-match even when it comes from the same

observation, and researchers may want to take that uncertainty into account when merging data sets. In `fastLink()`, we allow researchers to capture the fuzziness of the data match using both string-distance matching, which creates a numerical summary of the closeness of two string variables such as age, and numeric matching, which takes differences between numeric variables as a measure of closeness.

`fastLink()` lets users specify one of three separate string-distance measures — the Jaro distance metric (Jaro, 1989), the Levenshtein edit distance (Levenshtein, 1965), and the Jaro-Winkler distance metric (Winkler, 1990). While we refer interested readers to the extensive literatures on these measures for exact definitions and more information, their use in record linkage is fairly consistent. First, some string distance measure $M(s_i, s_j)$ is calculated on two strings s_i and s_j. The measure is converted to a similarity measure such that values closer to 1 are maximally similar, and values closer to 0 are maximally dissimilar. Lastly, a cutoff is chosen such that values of the similarity measure above the cutoff are coded as matches, while values of the similarity measure below the cutoff are coded as non-matches.

We can tell `fastLink()` to use string distance measures for specific variables using the `stringdist.match` argument as follows:

```
fl_out <- fastLink(
  dfA = dfA, dfB = dfB,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear"),
  stringdist.match = c("firstname", "lastname", "streetname"),
  stringdist.method = "jw", cut.a = 0.94
)
```

The argument `stringdist.match` contains a vector of variables that should be matched using a string-distance metric, which is specified in `stringdist.method`. Here, we've specified that `fastLink()` should use the Jaro-Winkler metric (`jw`), although other valid metrics include the Jaro metric (`jaro`) or the Levenshtein distance (`lv`). Finally, `cut.a` specifies that if the string similarity measure between pairs of

observations is above 0.94, the pair should be coded as a match. Default settings are `stringdist.method = "jw"` and `cut.a = 0.94` (following the recommendations of Winkler (1990)).

Specifying the variables to be compared using numeric matching is similar. If, for example, we believe that there may be administrative errors in year of birth or address number, we may want to construct a measure of similarity by simply taking the absolute value of the difference between the observations for each pair. The closer the difference is to 0, the more likely they are to be a match on that variable. We can make that comparison using the `numeric.match` argument in `fastLink()`, as follows:

```
fl_out <- fastLink(
  dfA = dfA, dfB = dfB,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear"),
  stringdist.match = c("firstname", "lastname", "streetname"),
  stringdist.method = "jw", cut.a = 0.94,
  numeric.match = c("birthyear"), cut.a.num = 1
)
```

Of the six variables we specified to match on, three (`firstname`, `lastname`, and `streetname`) are now being compared using Jaro-Winkler string distances as described above, and one (`birthyear`) is being compared with numeric distances. The final two variables, `city` and `housenum`, use exact comparisons (the default). We have also specified the threshold below which a numeric comparison is declared a match using `cut.a.num` — here, if the birth year of observation i and observation j are less than ± 1 year apart or less, it is declared a match.

4.3.3 Incorporating Partial Match Categories

So far, we have introduced string distance and numeric distance comparisons in terms of a single cutoff — if the comparison value is above the cutoff, it is declared a match, while if it is below the cutoff, it is declared a non-match. For both comparison types,

though, we can introduce partial matching categories that can take advantage of more information in the data. Here, a comparison is declared a match if it is above the threshold provided in `cut.a` for a string distance comparison and `cut.a.num` for a numeric distance comparison, and it will be considered a partial match if its value is above the threshold given in `cut.p` (`cut.p.num`) and below the threshold given in `cut.a` (`cut.a.num`) for string.distance comparisons (numeric distance comparisons). If the value of the comparison is below `cut.p` for string distance comparisons or `cut.p.num` for numeric comparisons, it is declared a non-match. We can specify partial match comparisons as follows, using the argument `partial.match`:

```
fl_out <- fastLink(
  dfA = dfA, dfB = dfB,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear"),
  partial.match = c("streetname", "birthyear"),
  stringdist.match = c("firstname", "lastname", "streetname"),
  stringdist.method = "jw", cut.a = 0.94, cut.p = 0.88,
  numeric.match = c("birthyear"), cut.a.num = 1, cut.p.num = 2.5
)
```

As before, we are matching the variables `firstname`, `lastname`, and `streetname` using string-distance matching, `birthyear` using numeric distance matching, and `housenum` and `city` using exact matching. However, we've also specified that the comparisons for `streetname` and `birthyear` include a partial match category, and we've also noted the lower cutoffs for declaring a comparison a partial match using the arguments `cut.p` and `cut.p.num`. For all other variables without a partial match category, the cutoff for match/non-match is still `cut.a` and `cut.a.num`.

4.3.4 Random Sampling to Speed up Large-scale Data Merges

One advantage of the probabilistic modeling framework is that we can use random sampling of observations to reduce the computational burden of estimating the model.

Calculating agreement patterns across two data sets of several million observations each can be computationally difficult even for fastLink, and since the number of parameters being estimated by the model is fairly small (just $\boldsymbol{\pi}$ and λ), we can easily estimate them on a small random subset of the data. Then, the user can split up the larger data linkage task into smaller chunks and apply the parameter estimates from the random sample to the full set of agreement patterns in parallel.

We demonstrate a simple example of this workflow below. We start by creating two small random samples of our test data, `dfA.s` and `dfB.s`, by randomly sampling 30% of of the observations in each. Since we're only estimating the model on this subset and not actually getting matched pairs, we can tell `fastLink()` to only return the EM object by specifying `estimate.only = TRUE`. This not only reduces the computational time for estimating and returning the fastLink object, but it also returns the EM object in a form that can be fed back into `fastLink()` when getting posterior match probabilities for the full sample (`fl_out_rs`).

```
## Take 30% random samples of dfA and dfB
dfA.s <- dfA[sample(1:nrow(dfA), nrow(dfA) * .3),]
dfB.s <- dfB[sample(1:nrow(dfB), nrow(dfB) * .3),]

## Run the algorithm on the random samples
fl_out_rs <- fastLink(
  dfA = dfA.s, dfB = dfB.s,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear"),
  estimate.only = TRUE
)

## Estimate parameters for whole dataset
fl_out_predict <- fastLink(
  dfA = dfA, dfB = dfB,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear"),
  em.obj = fl_out_rs
)
```

Finally, we take the estimated model parameters and feed them back into `fastLink()` to predict posterior match probabilities on the full data set. The EM object is simply fed back into `fastLink()` using the `em.obj` argument, which tells the function to take the estimated match parameters and apply them to the match patterns in the full data set. The user can also split up the full data set and provide the same estimated model parameters to each separate run of `fastLink()` to further reduce the computational burden.

4.3.5 Capturing Dependence between Linkage Fields

A shortcoming of the Fellegi-Sunter model, as described in Section 4.2.1, is the assumption of conditional independence across fields — conditional on the match status M_{ij}, the observed match pattern of variable k is assumed to be independent of the observed match pattern for variable k'[2] For example, conditional on being a true match, the likelihood of being a match on first name should be fully independent of the likelihood of being a match on last name. This is often an infeasible assumption, particularly when dealing with large households. One proposed solution to this limitation of the standard Fellegi-Sunter model (see Winkler, 1989, 1993; Thibaudeau, 1993; Larsen and Rubin, 2001, for full details) is to use weighted log-linear models to capture the full set of dependencies across linkage fields, conditional on match status.

We implement the log-linear model in fastLink through the option `cond.indep` — when set to true (default), `fastLink()` runs the standard Fellegi-Sunter conditionally independent model. When set to false, it substitutes the log-linear modeling strategy in the EM algorithm to model dependencies between linkage fields.

```
fl_out <- fastLink(
  dfA = dfA, dfB = dfB,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear"),
  cond.indep = FALSE
```

[2]This can be formally stated as $\gamma_k(i,j) \perp\!\!\!\perp \gamma_{k'}(i,j) \mid M_{i,j}$.

```
)
```

As of version 0.4.1 of fastLink, the log-linear modeling strategy cannot accomodate priors as described in Section 4.4.1. We save these extensions for future work.

4.3.6 Finding Duplicates in a Single Data Set

fastLink can also be used to de-duplicate a single data frame using the PRL framework. Within-data-frame dedupliation is straightforward in fastLink— the user simply provides the same data frame to both `dfA` and `dfB` arguments, and runs `fastLink()` as they normally would. The `blockData()` function then takes both data frames and the resulting output from `fastLink()`, and returns a final data frame with a new ID column indicating identical observations.

Below is a simple workflow to de-duplicate `dfA`, after adding 10 duplicated observations manually:

```
## Add duplicates
dfA <- rbind(dfA, dfA[sample(1:nrow(dfA), 10, replace = FALSE),])

## Run fastLink
fl_out_dedupe <- fastLink(
  dfA = dfA, dfB = dfA,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear")
)

##
## ====================
## fastLink(): Fast Probabilistic Record Linkage
## ====================
##
## dfA and dfB are identical, assuming deduplication of a single data set.
## Setting return.all to FALSE.
##
## Calculating matches for each variable.
## Getting counts for parameter estimation.
##     Parallelizing calculation using OpenMP. 1 threads out of 8 are used.
## Running the EM algorithm.
```

```
## Getting the indices of estimated matches.
##     Parallelizing calculation using OpenMP. 1 threads out of 8 are used.
## Calculating the posterior for each pair of matched observations.
## Getting the match patterns for each estimated match.

## Run getMatches
dfA_dedupe <- getMatches(dfA = dfA, dfB = dfA, fl.out = fl_out_dedupe)

## Look at the IDs of the duplicates
names(table(dfA_dedupe$dedupe.ids)[table(dfA_dedupe$dedupe.ids) > 1])

##  [1] "501" "502" "503" "504" "505" "506" "507" "508" "509" "510"

## Show duplicated observation
dfA_dedupe[dfA_dedupe$dedupe.ids == 501,]

##     firstname middlename  lastname housenum streetname          city
## 77    jeffrey          j blomquist     2506  anita ave Castro Valley
## 771   jeffrey          j blomquist     2506  anita ave Castro Valley
##     birthyear dedupe.ids
## 77       1951        501
## 771      1951        501
```

Users can then dedupe data sets using the `dedupe.ids` covariate in the returned data frame.

4.4 Incorporating Auxiliary Information and Post-Processing Data Merges

Another advantage of the probabilistic modeling approach is its ability to incorporate auxiliary information into the estimation process. This information can take the form of ex-post adjustments or as Bayesian priors on the relevant parameters, and is easily incorporated into the fastLink estimation step. We detail the two main auxiliary information types — migration rate information and first name frequencies — below, and show how to include them in model estimation. In addition, we show how to post-process data merges by enforcing a one-to-one restriction, such that every obsevation in dataset A is matched to at most one observation in dataset B, and vice versa.

4.4.1 Information on Migration Rates

One important substantive application of record linkage methodologies is to study individuals who move across neighborhoods, states, and counties. While these individuals can be hard to track because of the loss of address information to inform the match, their social behavior, and the behavior of those around them, is of interest to many substantive scholars. To improve the quality of matching data sets for discovering movers, we provide a few tools to calibrate match rates using known auxiliary data on movers' rates.

Two parameters, λ (the probability that a given pairwise comparison is a true match) and $\pi_{\text{adr},1,0}$ (the probability that a true matched pair has different addresses) can be calibrated using available auxiliary data on migration. For example, when matching two voter files from the same state across different years, the prior on λ can be calibrated as:

$$\lambda^{\text{prior}} = \frac{\#\text{ of non-movers} + \#\text{ of in-state movers}}{N_{\mathcal{A}} \times N_{\mathcal{B}}}$$

while the prior on $\pi_{\text{adr},1,0}$ can be calibrated as

$$\pi_{\text{adr},1,0}^{\text{prior}} = \frac{\#\text{ of in-state movers}}{\#\text{ of in-state movers} + \#\text{ of non-movers}}$$

Counts of movers can come from a number of auxilary data sources — in fastLink, we use the IRS Statistics of Income datasets to recover these counts for within-state and across-state movers in the United States. The IRS SOI data is a definitive source on migration in the United States that relies on tax returns to track individual year-

to-year migration.[3] To automatically generate these priors, we can use the function `calcMoversPriors()` as follows:

```
priors_out <- calcMoversPriors(geo.a = "CA", geo.b = "CA",
                               year.start = 2014, year.end = 2015)
priors_out

## $lambda.prior
## [1] 6.925788742782244e-08
##
## $pi.prior
## [1] 0.02598119909353664
```

Here, we're recovering the estimated prior values for λ and $\pi_{\text{adr},1,0}$ for a match of the California voter file in 2014 to the California voter file in 2015.

We can then feed these into `fastLink()` using the `priors.obj` argument. In order to properly specify the prior, we also need to tell `fastLink()` how much to weight the prior estimates relative to the parameter estimates implied by the observed match patterns. To weight the priors, we specify the arguments `w.lambda` and `w.pi` as the weights for the λ^{prior} and $\pi^{\text{prior}}_{\text{adr},1,0}$, respectively. Specifying `w.lambda = .25` tells `fastLink()` that the final estimate for λ should be a weighted average where 25% is the prior estimate, and 75% is the maximum likelihood estimate from the observed match matterns. Lastly, when specifying a prior on the address field, we tell `fastLink()` which field is an address field using the `address.field` argument. An example, with priors calibrated for the test data set, is:

```
## Reasonable prior estimates for this dataset
priors_out <- list(lambda.prior = 50/(nrow(dfA) * nrow(dfB)),
                   pi.prior = 0.02)

fl_out <- fastLink(
  dfA = dfA, dfB = dfB,
```

[3]The IRS Statistics of Income data for both county-to-county migration and state-to-state migration, which dates back to 1990, is available online at https://www.irs.gov/statistics/soi-tax-stats-migration-data.

```
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear"),
  priors.obj = priors_out,
  w.lambda = .25, w.pi = .25,
  address.field = "streetname"
)
```

We place Beta priors on the relevant parameters to maintain conjugacy, while leaving the priors on the remaining parameters improper. Other data sources for auxiliary information on migration can be used in `fastLink()` by feeding them in as a list with the names `lambda.prior` or `pi.prior` as above.

4.4.2 Reweighting Match Probabilities Ex-Post with Name Frequencies

We can also take advantage of information about common and uncommon first names in order to improve match quality. Unlike the migration priors discussed above, it is more difficult to incorporate information about name frequency into estimation without dramatically increasing the computational cost of the estimation — instead, we follow the existing literature (e.g. Winkler, 2000) and make an ex-post adjustment to the estimated parameters using the name frequencies that are observed in the data. Specifically, we correct for the possibility that a pair sharing a common first name, such as John, may be more likely to be a true non-match than a pair sharing an uncommon first name, such as Jocelyn.

To conduct an ex-post adjustment on observed name frequencies, the user simply sets `reweight.names = TRUE` in `fastLink()` and provides the name of the first name field in `firstname.field`, as follows:

```
fl_out <- fastLink(
  dfA = dfA, dfB = dfB,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear"),
```

```
  reweight.names = TRUE, firstname.field = "firstname"
)
```

The unweighted posterior match probabilities are then replaced with the reweighted posterior match probabilities in the `posterior` slot of the returned `fastLink` object.

4.4.3 Enforcing a One-to-One Merge

One issue with the probabilistic modeling framework is that a single observation in one data set can be matched to multiple observations in the second data set. Individuals within a household who share a name, for example, can be a challenge for PRL models, where the posterior probability of matching can still be quite high even if name suffixes ("Jr.", "Sr.", "III") are different. To adjust for this, `fastLink()` offers several methods to enforce a "one-to-one" merge such that each observation in dataset A is matched to at most one observation in dataset B, and vice versa.

The first method implemented, which is the default method used when enforcing a one-to-one merge, is a greedy algorithm that takes the best possible match for each observation in dataset A in dataset B, and then vice versa for dataset B among the possible remaining matches in dataset A. Any remaining ties are broken at random. By default, `fastLink()` runs the greedy de-duping algorithm after recovering all successfully matched indices from both data sets, but it can be called explicitly by running:

```
fl_out <- fastLink(
  dfA = dfA, dfB = dfB,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear"),
  dedupe.matches = TRUE
)
```

fastLink also implements a linear sum assignment solution proposed by Jaro (1989); Winkler (1994), which maximizes the sum of the posterior match probabilities subject

to the one-to-one match constraint. Unlike the greedy de-duping algorithm, this method considers all possible assignments across pairs to find the optimal set that maximizes the sum of the posterior match probabilities, and Winkler (1994) shows that this method is more robust than the greedy algorithm in small-to-medium-sized data sets. However, as the size of the data grows large, the method scales poorly in runtime, and the relative advantages in accuracy are smaller. However, we recommend this method for smaller data merging problems. Users can implement the linear sum assignment de-duping method by setting `linprog.dedupe = TRUE`, as follows:

```
fl_out <- fastLink(
  dfA = dfA, dfB = dfB,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear"),
  dedupe.matches = TRUE, linprog.dedupe = TRUE
)
```

4.5 Preprocessing Data Merges

While much of fastLink focuses on improving the speed and quality of the Fellegi-Sunter model, the success of a data merge is fundamentally dependent on preprocessing and cleaning decisions made by the analyst. Such decisions include ensuring that both data sets abide by similar rules about leading zeros, that they both split strings of names into first/middle/last names in identical ways, that abbreviations such as Rd. are changed to be equivalent to Road, and that accents in names are handled in similar ways. Here, we describe some tools for effectively harmonizing data sets before conducting a merge using fastLink, along with tools for blocking data in preparation for a match.

4.5.1 Cleaning and Harmonizing Strings

Preprocessing string data for a data merge generally proceeds in two steps — first, data is tokenized into similar fields (from "Benjamin Haber Fifield" to "Benjamin",

"Haber", and "Fifield" for names, and from "19 Crawford Road" to "19", "Crawford", and "Road" for addresses), and then harmonizing those fields using some manner of text standardization. While we do not implement the tokenization step in fastLink, we will discuss a Python library called probablepeople that we have used extensively in our own substantive work. We will then briefly introduce a function in fastLink, `preprocText()`, for standardizing and harmonizing text fields that have already been tokenized using United States Postal Service benchmarks for standardization.

Tokenizing String Data

While we do not implement tokenizing ourselves in fastLink, we recommend that users seeking to tokenize data use the Python library probablepeople for personal and corporation names and usaddress for addresses, which we have used in our own substantive applications. Both libraries use pre-trained conditional random field models to split name, entity, and address strings into specific components such as first name, last name, and suffix (for names), corporation name and corporation legal type (for corporations and entities), and street number, street name, town, state, and zip code (for addresses).

Below we show a simple example where we use probablepeople to parse strings of corporation names and personal names into separate components. For a given string, we simply use the `tag()` function in the probablepeople library, which outputs an ordered dictionary where each identified component is assigned to one of several pre-defined tags. For names, these tags include `FirstName`, `Surname`, `Nickname`, `PrefixMarital`, and others.[4] Corporations have their own set of pre-defined tags for the separate components.

```
# Parsing personal and corporation names using probablepeople
import probablepeople as pp
```

[4]See the documentation for probablepeople at https://probablepeople.readthedocs.io/en/latest/ for more detail.

```
name_str = 'Ted Enamorado'
uni_str = 'Princeton University'

name_str_parsed = pp.tag(name_str)
uni_str_parsed = pp.tag(uni_str)

print "Parsed fields for name:"
for key in name_str_parsed[0]:
    print("{}: {}".format(key, name_str_parsed[0][key]))
print "\n"

print "Parsed fields for university:"
for key in uni_str_parsed[0]:
    print("{}: {}".format(key, uni_str_parsed[0][key]))

## Parsed fields for name:
## GivenName: Ted
## Surname: Enamorado
##
##
## Parsed fields for university:
## CorporationName: Princeton
## CorporationNameOrganization: University
```

We can apply the same methodology to address strings in order to parse them into usable components. Again, the `tag()` function will take an input string and output an ordered dictionary with pre-defined tags. Valid tags for addresses include `AddressNumber`, `BuildingName`, `StateName`, `PlaceName`, and `ZipCode`, among other tags.[5]

```
# Parsing addresses using usaddress
import usaddress as usr

adr_string = '001 Fisher Hall, Princeton, NJ 08540'

adr_str_parsed = usr.tag(adr_string)

print "Parsed fields for address:"
```

[5]See the documentation for usaddress at https://usaddress.readthedocs.io/en/latest/ for more detail.

```
for key in adr_str_parsed[0]:
    print("{}: {}".format(key, adr_str_parsed[0][key]))

## Parsed fields for address:
## AddressNumber: 001
## StreetName: Fisher Hall
## PlaceName: Princeton
## StateName: NJ
## ZipCode: 08540
```

Once parsed, these tagged fields can be used as inputs for `fastLink()` as additional fields to match datasets on. However, we remind users that once parsed, the original string should not be fed into `fastLink()` in addition to the parsed strings. Doing so, in addition to leading to longer runtimes, will violate the conditional independence assumption such that the agreement patterns of the parsed fields will be dependent on the agreement patterns of the un-parsed fields.

Harmonizing String Fields

Once strings are parsed into individual components that will be used for a match, we also have to ensure that the same field across datasets handles the same data identically. For example, if street types in dataset A are 'Road', 'Street', and 'Avenue', but in dataset B they are 'Rd.', 'St.', and 'Ave.', these true matches will be coded by fastLink (and any other matching algorithm) as a non-match on that variable. To ensure harmonization across dataset by field, we provide a function for text standardization, preprocText(), that can help ensure this equivalence across datasets.

The function preprocText() takes as argument a vector of text, which it assumes has already been split into fields, and then provides a number of options for conversion. The options applicable to all string types include `tolower` (convert all text to lower-case), `remove_whitespace` (strip leading and trailing whitespace, and convert multiple spaces to single space), and `remove_punctuation` (remove all punctuation

such as apostrophes, semi-colons, colons, periods, question marks, etc.). For example, we can convert text as follows:

```
preprocText(text = " Road.", tolower = TRUE,
            remove_whitespace = TRUE, remove_punctuation = TRUE)

## [1] "road"
```

For names, `preprocText()` also implements Soundex encoding from the `stringdist` package. The Soundex algorithm is used by the U.S. Census Bureau to increase the likelihood of matches across similar-sounding names — for example, while Rupert and Robert are spelled differently and would be coded as non-matches by any string distance measure, Soundex encodings of both names are identical. We refer users to other references (e.g. Knuth, 1973, pgs. 391-92) for a full description of the algorithm, but as an overview, the algorithm keeps the first letter of a string and then follows a series of rules to convert the remaining portion of the string to a three-digit numerical code based on letter combinations that are similar in sound. Below, we show the output from a Soundex conversion of Rupert and Robert:

```
preprocText(text = c("Rupert", "Robert"), soundex = TRUE)

## [1] "R163" "R163"
```

Finally, for addresses, we also implement United States Postal Service (USPS) address standardization of street names to ensure all street types (for example, Road, Street, and Avenue) are consistently recorded across data sets. The USPS converts all street types to standard abbreviations (such as 'Road' to 'Rd.' and 'Street' to 'St.'), and we implement the same conversion in `preprocText()` through the `usps_address` argument:

```
preprocText(text = c("Street", "Boulevard"), usps_address = TRUE)

## [1] "st"   "blvd"
```

Together, these resources can help ensure the uniformity across data sets of string variables before conducting a data merge.

4.5.2 Blocking Data to Improve Merge Quality

fastLink also includes a series of tools to help block data sets before conducting a merge. The goal of blocking is to avoid comparisons between observations that are certain non-matches, in order to increase the amount of overlap between data sets and to reduce computation time resulting from irrelevant comparisons. For example, if an analyst is certain there are no true matches across states when merging survey data to voter files, they can block by state and only compare observations within states. We refer readers interested in a comprehensive review of blocking techniques to Christen (2012) and Steorts et al. (2014) — here, we will cover the methods implemented in fastLink for blocking data before conducting the merge, and for identifying and removing observations with no obvious candidate matches.

In fastLink, the function `blockData()` can block two data sets using a single variable or combinations of variables using several different blocking techniques. The basic functionality is similar to that of `fastLink()`, where the analyst inputs two data sets and a vector of variable names that they want to block on. A simple example follows, where we are blocking the two sample data sets by gender:

```
blockgender_out <- blockData(dfA, dfB, varnames = "gender")

##
## ====================
## blockData(): Blocking Methods for Record Linkage
## ====================
##
## Blocking variables.
```

```
##     Blocking variable gender using exact blocking.
##
## Combining blocked variables for final blocking assignments.

names(blockgender_out)

## [1] "block.1" "block.2"
```

In its simplest usage, `blockData()` takes two data sets and a single variable name for the `varnames` argument, and it returns the indices of the member observations for each block. Data sets can then be subsetted as follows and the match can then be run within each block separately:

```
## Subset dfA into blocks
dfA_block1 <- dfA[blockgender_out$block.1$dfA.inds,]
dfA_block2 <- dfA[blockgender_out$block.2$dfA.inds,]

## Subset dfB into blocks
dfB_block1 <- dfB[blockgender_out$block.1$dfB.inds,]
dfB_block2 <- dfB[blockgender_out$block.2$dfB.inds,]

## Run fastLink on each
fl_out_block1 <- fastLink(
  dfA_block1, dfB_block1,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear")
)
fl_out_block2 <- fastLink(
  dfA_block2, dfB_block2,
  varnames = c("firstname", "lastname", "housenum",
               "streetname", "city", "birthyear")
)
```

Blocking on gender substantially reduces the number of comparisons fastLink has to make. Without blocking, fastLink makes $510 \times 350 = 178{,}500$ comparisons, and the probability that any given comparison is a match is 0.00029. However, when blocking on gender, fastLink makes $269 \times 165 + 241 \times 185 = 88{,}970$ comparisons, while the baseline probability of finding a match goes up to 0.00056 in Block 1, and

0.00058 in Block 2. Stricter blocking strategies, of course, will reduce the number of comparisons even further.

Furthermore, `blockData()` allows analysts to go beyond exact blocking on a single variable. Users can block on multiple variables by feeding a vector of variable names to `varnames`, as follows:

```
## Exact block on gender and city
blockdata_out <- blockData(dfA, dfB, varnames = c("gender", "city"))
```

`blockData()` also implements other methods of blocking other than exact blocking. Analysts commonly use *window blocking* for numeric variables, where a given observation in dataset A will be compared to all observations in dataset B where the value of the blocking variable is within $\pm K$ of the value of the same variable in dataset A. The value of K is the size of the window — for instance, if we wanted to compare observations where birth year is within ± 1 year, the window size is 1. Below, we block `dfA` and `dfB` on gender and birth year, using exact blocking on gender and window blocking with a window size of 1 on birth year:

```
## Exact block on gender, window block (+/- 1 year) on birth year
blockdata_out <- blockData(
  dfA, dfB,
  varnames = c("gender", "birthyear"),
  window.block = "birthyear", window.size = 1
)
```

`blockData()` also allows users to block variables using k-means clustering, so that similar values of string and numeric variables are blocked together. When applying k-means blocking to string variables such as name, the algorithm orders observations so that alphabetically close names are grouped together in a block. In the following example, we block `dfA` and `dfB` on gender and first name, again using exact blocking on gender and k-means blocking on first name while specifying 2 clusters for the k-means algorithm:

```
## Exact block on gender, k-means block on first name with 2 clusters
blockdata_out <- blockData(
  dfA, dfB,
  varnames = c("gender", "firstname"),
  kmeans.block = "firstname", nclusters = 2
)
```

In addition to the blocking functionalities, fastLink also includes methods to help researchers discard observations with no obvious matching candidates in the paired data set using string distance comparisons. `stringSubset()` calculates the string distance between each pair of observations in two data sets and returns the indices where there is at least one candidate match with a string similarity measure above a certain threshold. That is, it reduces the set of candidate matches by throwing out any observation in dataset A where no observation in dataset B has a sufficiently similar first name/last name/street name, and vice versa.

As an example, we will reduce the set of candidate matches by discarding any observation in the two sample datasets where the Jaro-Winkler string similarity measure is below 0.8, as follows:

```
stringsub_out <- stringSubset(dfA$firstname, dfB$firstname,
                              similarity.threshold = .8,
                              stringdist.method = "jw")
names(stringsub_out)

## [1] "dfA.inds" "dfB.inds"
```

In the output object, `dfA.inds` contains the indices of the observations in `dfA` where at least one observation in `dfB` has a sufficiently similar first name, and vice versa for `dfB.inds`. This procedure has also reduced the number of possible comparisons — whereas the standard merge has to make 178,500 comparisons, now `fastLink()` only has to make 145,222 comparisons within the subsetted data frames. `stringSubset()` also allows users to specify Jaro string-distance measures and Levenshtein distances instead of the default Jaro-Winkler distance.

4.6 Application — Party Switching in Brazil

We illustrate a data merging workflow using fastLink by merging two datasets of local politicians in Rio de Janeiro, Brazil to estimate rates of party switching between 2012 and 2016. The data, obtained from the Brazil's Tribunal Superior Eleitoral (TSE), records the names, party affiliations, and background information of all local poiticians in Rio, and as the string fields are manually entered possible misspellings can lead to errors when attempting to exact-match.[6] Therefore, a PRL approach can help control the false negative rate that exact matching can exacerbate. Importantly, the data includes a perfectly recorded unique identifier, which can be used to validate the accuracy of the PRL model — the Cadastro de Pessoas Físicas (CPF), the Brazilian individual taxpayer registry identification number. Therefore, the data is a valuable naturalistic test of PRL models that can be easily validated.

The data merge proceeds in three steps — first, we harmonize the merging fields to be consistent across the two data sets and create the variables necessary for the merge. Next, we create blocks to increase overlap while minimizing the false negative rate, and then run the merge block-by-block. We then examine some basic diagnostics of the merge, and importantly, we validate the merge against the ground truth using the CPF and compare against an exact matching strategy. Finally, we analyze rates of party switching from the fastLink-merged data and compare to results that use the exact-merged data.

Data Cleaning In the first step of the merge, we construct variables that can be used to help identify true matches. As the data is relatively clean to begin with, and there is no information on addresses in the data that can be used to inform the merge beyond province, we focus on constructing useful name and age variables.

[6]The raw TSE data is available from http://www.tse.jus.br/eleicoes/estatisticas/repositorio-de-dados-eleitorais-1/repositorio-de-dados-eleitorais.

The raw name data for the politicians comes as a single string, so we start by tokenizing the names into first, middle, and last names. While probablepeople, which we introduced in Section 4.5.1, is most useful for messy and unstructured name fields, the Brazil data is already fairly clean — therefore, we rely on the parse_names() function in the humaniformat R package to split the names into first, middle, and last name. We will compare both fields using string-distance measures in fastLink, with an included partial match category.

```
## Load data
load("../linkage/data/data2012_candidatesRJ.RData")
load("../linkage/data/data2016_candidatesRJ.RData")

## Parse name variables
data2012_names <- parse_names(data.2012.RJ$candidate_name)
data2016_names <- parse_names(data.2016.RJ$candidate_name)

data.2012.RJ$first_name <- data2012_names$first_name
data.2012.RJ$middle_name <- data2012_names$middle_name
data.2012.RJ$last_name <- data2012_names$last_name

data.2016.RJ$first_name <- data2016_names$first_name
data.2016.RJ$middle_name <- data2016_names$middle_name
data.2016.RJ$last_name <- data2016_names$last_name
```

As entered in the raw data, birth date can only be compared using string distances, which are inappropriate for the underlying numeric data. We parse the birth dates into R's date class, and then calculate each politician's age in years. We will compare the age field across data using numeric matching, where an exact match is any set of ages within ± 1 year and a partial match is within ± 2.5 years.

```
## Parse age variable
data.2012.RJ$age <- floor(
  age_calc(
    as.Date(data.2012.RJ$candidate_dob, format = "%d/%m/%Y"),
    units = "years"
  )
```

```
)
data.2016.RJ$age <- floor(
  age_calc(
    as.Date(data.2016.RJ$candidate_dob, format = "%d/%m/%Y"),
    units = "years"
  )
)
```

Blocking the Data Having harmonized the fields across the 2012 and 2016 data sets, we then block the data to increase the amount of overlap between the data being matched. Correctly chosen blocking variables attempt to minimize the false negative rate across blocks — that is, since observations grouped into different blocks will not be compared, there should be as few true positives as possible assigned to different blocks. More formally, if b_i indicates the block of observation i, we want to ensure that

$$\Pr(M_{ij} = 1 \mid b_i \neq b_j) = 0$$

We choose the politician's home municipality for this reason. First, there are 92 municipalities in Rio, which will substantially reduce the number of comparisons to make. We also know *a priori* that municipality is perfectly recorded in the data, that the municipality lines did not change between 2012 and 2016, and that very few local politicians moved across municipal lines between 2012 and 2016, which gives us high confidence that the true match rate across blocks is nearly 0. We create the blocks as follows:

```
## Block datasets by municipality
block_out <- blockData(
  dfA = data.2012.RJ, dfB = data.2016.RJ,
  varnames = "mun_name"
)
```

We can check exactly how much the comparison space is reduced. Unblocked, fastLink has to conduct $21{,}905 \times 22{,}346 = 489{,}489{,}130$ comparisons, which is large but

feasible. After blocking, this comparison space is reduced to 9,630,656 comparisons, or 1.97% of the original comparison task with minimal loss of true matches.

Conducting the Merge Now that both datasets are harmonized with the merging variables and blocked, we can run fastLink within each block to merge the full data set. Within each block, we first subset the data, and then merge the subsetted data frames on first, middle, and last name, age, gender, and candidate marital status. First, middle, and last name is compared using string distances, marital status is compared using exact comparisons, and age is compared using numeric distance. All distance cutoffs use the fastLink defaults to determine whether a comparison on a particular variable is a match, a partial match, or a non-match. We declare any pair with a posterior match probability above 0.85 (the fastLink default) to be a match and store the successful matches, and we also store the fl_out objects for analysis.

```
## Loop over blocks and merge
match_out <- vector(mode = "list", length = length(block_out))
flobj_out <- vector(mode = "list", length = length(block_out))
for(i in 1:length(block_out)){

  ## Subset data
  data.2012.sub <- data.2012.RJ[block_out[[i]]$dfA.inds,]
  data.2016.sub <- data.2016.RJ[block_out[[i]]$dfB.inds,]

  ## Run fastLink
  fl_out <- fastLink(
    dfA = data.2012.sub, dfB = data.2016.sub,
    varnames = c("first_name", "middle_name", "last_name",
                 "age", "candidate_mar_status_desc"),
    stringdist.match = c("first_name", "middle_name", "last_name"),
    numeric.match = "age",
    partial.match = c("first_name", "last_name", "age")
  )

  ## Get matches, store
  match_out[[i]] <- getMatches(
    dfA = data.2012.sub, dfB = data.2016.sub, fl.out = fl_out,
    threshold.match = 0.85, combine.dfs = FALSE
```

```
  )
  flobj_out[[i]] <- fl_out

}

## Combine data frames
match_2012 <- do.call("rbind", lapply(match_out, "[[", "dfA.match"))
match_2016 <- do.call("rbind", lapply(match_out, "[[", "dfB.match"))
```

Having run the match, we can now examine the estimated quality of the matches using the confusion() function. In addition to estimating summary statistics for a single fastLink object, confusion() can also aggregate over multiple (non-overlapping) blocks to generate estimates of the false positive rate, false negative rate, sensitivity, specificity, and other summary statistics.

```
confusion(flobj_out)

## $confusion.table
##                       'True' Matches 'True' Non-Matches
## Declared Matches            6619.64              22.36
## Declared Non-Matches          11.14           15251.86
##
## $addition.info
##                                    results
## Max Number of Obs to be Matched 21905.00
## Sensitivity (%)                    99.83
## Specificity (%)                    99.85
## Positive Predicted Value (%)       99.66
## Negative Predicted Value (%)       99.93
## False Positive Rate (%)             0.15
## False Negative Rate (%)             0.17
## Correctly Classified (%)           99.85
## F1 Score (%)                       99.75
```

The estimates from the model suggest that fastLink() has effectively minimized both the false positive and the false negative rate. Out of a total of 6,642 estimated matches, only 22.36 are estimated to be false positives. Likewise, out of the estimated 15,263 non-matches, only 11.14 are estimated to be false negatives.

	Exact Match		fastLink Match	
	True Match	True Non-Match	True Match	True Non-Match
Declared Match	63.3%	0.0781%	95%	2.75%
Declared Non-Match	36.7%	99.9%	5.01%	97.2%

Table 4.1: Validation of the TSE data merge using the CPF number. This table shows the true classification accuracy of the exact matching strategy (left) and fastLink-matching strategy (right) using the CPF unique identifier. While the exact matching strategy minimizes false positives, it does so at the cost of a large number of false negatives — true matches that were incorrectly classified as non-matches. In contrast, while the fastLink matching strategy does slightly worse on the false positive rate, it substantially out-performs an exact matching strategy on the false negative rate.

Comparison to Ground Truth A benefit of the TSE data is the existence of a ground truth that can be used to calculate actual classification accuracy, using the CPF number of every observation in the data. If we had merged these data on CPF, we would get a completely accurate match — however, here we use it as a validation technique to compare fastLink against the exact matching strategy. To conduct an exact match, we create a unique key for each observation out of first, middle, and last name, age, and candidate marital status (the same variables used for the fastLink merge) and then run an inner join on those variables. Table 4.1 gives the results of the validation exercise.

First, looking at the results for the exact matching strategy, we see that it controls the false positive rate effectively. In order to be declared a true match, a pair of observations must match exactly on all three name fields, plus age, plus marital status — a stringent requirement that leads to nearly none of the declared matches being misclassified. However, this comes at a cost — as we make the threshold for being declared a true match more and more stringent, we increase the false negative rate, where true matches are incorrectly declared non-matches. It is clear that the exact matching strategy falls short here — 36.7% of all true matches are incorrectly declared non-matches, due to minor misspellings and other administrative errors that the exact matching strategy cannot accomodate.

In contrast, the PRL strategy implemented in fastLink does substantially better in avoiding false negatives. Only 5.01% of true matches are incorrectly classified as non-matches, since fastLink and other PRL implementations can efficiently incorporate information on string distances, partial matches and other types of distance measures that exact matching strategies cannot use. This does come at a small cost — while the exact matching strategy reduces the false positive rate nearly to 0, the PRL approach introduces a small number of false positives. However, it is clear that PRL using fastLink balances the false positive rate and the false negative rate much more effectively than an exact matching approach, leading to more accurate analyses using merged data sets.

Analyzing Party-Switching in Rio de Janeiro Using the merged data, we can answer substantive questions about party alignment and politician behavior over time in Rio de Janeiro. Here, we focus on party-switching behavior, where one politician may switch parties for strategic reasons. We start by getting the ground truth of party-switching by merging the 2012 and 2016 data on CPF — out of a total of 6,594 politicians that are in both data sets, 68.44% switch political parties between 2012 and 2016. The fastLink-merged data gets very close to this benchmark, slightly overestimating the true party-switching rate at 70.23%.

In Figure 4.3, we also break down the party-switching rates in Rio by municipality. Party-switching rates derived from the ground truth are on the left, while those from the fastLink match are on the right. We can see that the party-switching rates from fastLink correlate closely with the ground truth. Both uncover similar levels of high party-switching behavior across Rio, but also similar geographic distributions of party-switching. The highest rates of party-switching are in the periphery of Rio, while party-switching is highest in more geographically central municipalities.

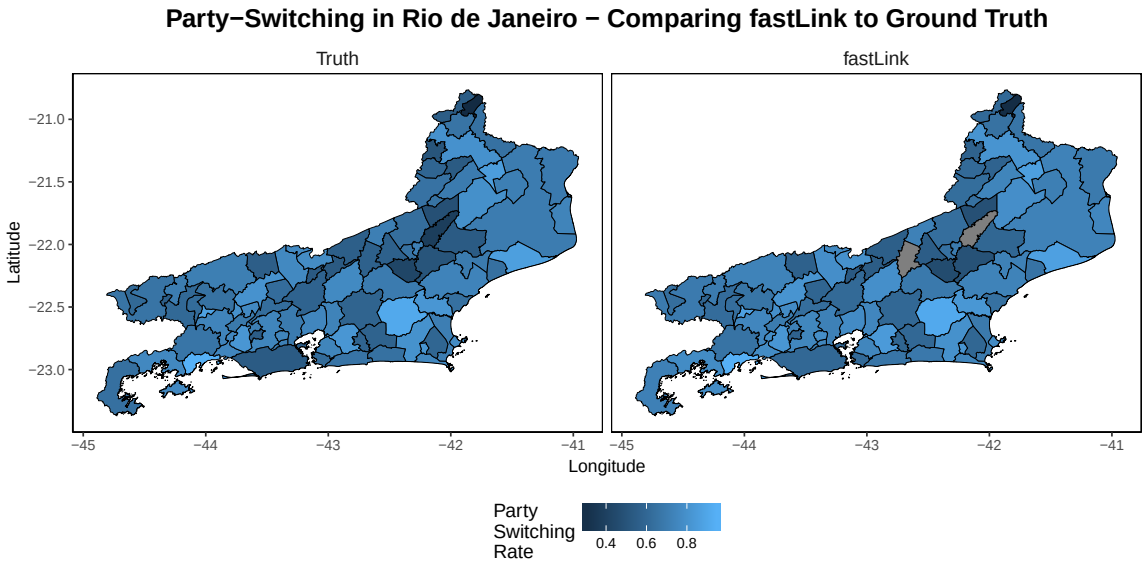

Figure 4.3: Party-switching by municipality in Rio de Janeiro. The left-hand plot shows the true rates of party switching across Rio de Janeiro between 2012 and 2016, while the right-hand plot shows the party-switching rate estimated after matching the 2012 and 2016 TSE data using fastLink.

4.7 Conclusion

In this paper, we describe the functionalities of the fastLink package, which implements an improved algorithm for merging data under the Fellegi-Sunter probabilistic record linkage model. The package implements a general version of the algorithm, along with a number of improvements such as easy handling of missing data, incorporation of auxiliary information, numeric and string-distance measures for calculating match patterns, and parallelization to improve the computational efficiency of the algorithm. In addition, numerous functionalities for standardizing and pre-processing data, including blocking and string-distance thresholding techniques, are implemented. As the literature on record linkage continues to grow, driven in large part by the number of social scientists utilizing cutting-edge data sets in their research, we hope that fastLink can serve as a platform for the continued implementation and improvement of these new methods.

Bibliography

Altman, Micah and Michael P. McDonald. 2011. "BARD: Better Automated Redistricting." *Journal of Statistical Software* 42:1–28.

Athey, Susan and Guido Imbens. 2016. "Recursive Partitioning for Heterogeneous Causal Effects." *Proceedings of the National Academy of Sciences* 113:7353–7360.

Athey, Susan, Julie Tibshirani and Stefan Wager. 2017. "Generalized Random Forests." Working Paper.

Athey, Susan and Stefan Wager. Forthcoming. "Estimation and Inference of Heterogeneous Treatment Effects using Random Forests." *Journal of the American Statistical Association*.

Bansak, Kirk. 2018. "A Generalized Framework for the Estimation of Causal Moderation Effects with Randomized Treatments and Non-Randomized Moderators." Working Paper.

Breiman, Leo. 2001. "Random Forests." *Machine Learning* 45:5–32.

Browdy, Michelle H. 1990. "Simulated Annealing: An Improved Computer Model for Political Redistricting." *Yale Law & Policy Review* 8:163–179.

Chen, Jowei. 2017. Expert Report of Jowei Chen, Ph.D. Technical Report. Department of Political Science, University of Michigan.

Chen, Jowei and Jonathan Rodden. 2013. "Unintentional Gerrymandering: Political Geography and Electoral Bias in Legislatures." *Quarterly Journal of Political Science* 8:239–269.

Chikina, Maria, Alan Frieze and Wesley Pegden. 2017. "Assessing significance in a Markov chain without mixing." *Proceedings of the National Academy of Sciences of the United States of America* 114:2860–2864.

Cho, Wendy Tam. 2017. Expert Report of Wendy K. Tam Cho, RE: League of Women Voters v. Wolf et al. Technical Report. Department of Political Science, University of Illinois-Urbana Champaign.

Chou, Chung-I and S. P. Li. 2006. "Taming the Gerrymander — Statistical physics approach to Political Districting Problem." *Physica A: Statistical Mechanics and its Applications* 369:799–808.

Christen, Peter. 2012. *Data Matching. Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Springer.

Cirincione, Carmen, Thomas A. Darling and Timothy G. O'Rourke. 2000. "Assessing South Carolina's 1990s congressional districting." *Political Geography* 19:189–211.

Dempster, Arthur P., Nan M. Laird and Donald B. Rubin. 1977. "Maximum Likelihood from Incomplete Data Via the EM Algorithm (with Discussion)." *Journal of the Royal Statistical Society, Series B, Methodological* 39:1–37.

Ding, Peng, Avi Feller and Luke Miratrix. 2016. "Randomization Inference for Treatment Effect Variation." *Journal of the Royal Statistical Society, Series B* 78:655–671.

Ding, Peng, Avi Feller and Luke Miratrix. Forthcoming. "Decomposing Treatment Effect Variation." *Journal of the American Statistical Association.*

Enamorado, Ted, Benjamin Fifield and Kosuke Imai. 2017. Using a Probabilistic Model to Assist Merging of Large-scale Administrative Records. Technical Report. Department of Politics, Princeton University.

Fellegi, I. P. and A. B. Sunter. 1969. "A Theory for Record Linkage." *Journal of the American Statistical Association.* 64:1183–1210.

Fifield, Benjamin, Alexander Tarr and Kosuke Imai. 2015. "redist: Markov Chain Monte Carlo Methods for Redistricting Simulation." available at the Comprehensive R Archive Network (CRAN). https://CRAN.R-project.org/package=redist.

Fifield, Benjamin, Michael Higgins, Kosuke Imai and Alexander Tarr. 2018. A New Automated Redistricting Simulator Using Markov chain Monte Carlo. Technical Report. Department of Politics, Princeton University.

Fryer, Roland G. Jr. and Richard Holden. 2011. "Measuring the Compactness of Political Districting Plans." *Journal of Law and Economics* 54:493–535.

Garfinkel, R. S. and G. L. Nemhauser. 1970. "Political Districting by Implicit Enumeration Techniques." *Management Science* 16:B495–B508.

Gelman, Andrew and Donald B. Rubin. 1992. "Inference from Iterative Simulations Using Multiple Sequences (with Discussion)." *Statistical Science* 7:457–472.

Gerber, Alan, Donald Green and Christopher Larimer. 2008. "Social Pressure and Voter Turnout: Evidence from a Large-Scale Field Experiment." *American Political Science Review* 102:33–48.

Geyer, Charles J. 1992. "Practical Markov Chain Monte Carlo." *Statistical Science* 7:473–511.

Geyer, Charles J. and Elizabeth A. Thompson. 1995. "Annealing Markov Chain Monte Carlo with Applications to Ancestral Inference." *Journal of the American Statistical Association* 90:909–920.

Grimmer, Justin, Solomon Messing and Sean Westwood. 2017. "Estimating Heterogeneous Treatment Effects and the Effects of Heterogeneous Treatments with Ensemble Methods." *Political Analysis* 25:413–434.

Gutierrez, Pierre and Jean-Yves Gerardy. 2016. Causal Inference and Uplift Modeling: A Review of the Literature. In *JMLR: Workshop and Conference Proceedings*. Vol. 67 pp. 1–16.

Herschlag, Gregory, Han Sung Kang, Justin Luo, Christy Vaughn Graves, Sachet Bangia, Robert Ravier and Jonathan Mattingly. 2018. Quantifying Redistricting in North Carolina. Technical Report. Department of Mathematics, Duke University.

Herschlag, Gregory, Robert Ravier and Jonathan C. Mattingly. 2017. Evaluating Partisan Gerrymandering in Wisconsin. Technical Report. Department of Mathematics, Duke University.

Hersh, Eitan. 2015. *Hacking the Electorate: How Campaigns Perceive Voters*. Cambridge University Press.

Hess, S. W., J. B. Weaver, H. J. Siegfeldt, J. N. Whelan and P. A. Zitlau. 1965. "Nonpartisan Political Redistrictingn by Computer." *Operations Research* 13:998–1006.

Hill, Jennifer. 2011. "Bayesian Nonparametric Modeling for Causal Inference." *Journal of Computational and Graphical Statistics* 20:217240.

Holland, Paul. 1986. "Statistics and Causal Inference." *Journal of the American Statistical Association* 81:945–960.

Imai, Kosuke and Aaron Strauss. 2011. "Estimation of Heterogeneous Treatment Effects from Randomized Experiments, with Application to the Optimal Planning of the Get-Out-the-Vote Campaign." *Political Analysis* 19:1–19.

Imai, Kosuke and Marc Ratkovic. 2013. "Estimating Treatment Effect Heterogeneity in Randomized Program Evaluation." *Annals of Applied Statistics* 7:443–470.

Jaro, Matthew. 1989. "Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida." *Journal of the American Statistical Association.* 84:414–420.

Kalla, Joshua and David Broockman. 2018. "The Minimal Persuasive Effects of Campaign Contact in General Elections: Evidence from 49 Field Experiments." *American Political Science Review* 112:148–166.

Kawahara, Jun, Takashi Horiyama, Keisuke Hotta and Shin-ichi Minato. 2017. Generating All Patterns of Graph Partitions Within a Disparity Bound. In *WALCOM 2017: International Workshop on Algorithms and Computation*. WALCOM pp. 119–131.

Kern, Holger and Donald Green. 2012. "Modeling Heterogeneous Treatment Effects in Survey Experiments with Bayesian Additive Regression Trees." *Public Opinion Quarterly* 76:491511.

King, Gary and Robert X. Browning. 1987. "Democratic Representation and Partisan Bias in Congressional Elections." *American Political Science Review* 81:1251–1276.

Knuth, Donald. 1973. *The Art of Computer Programming: Volume 3, Sorting and Searching*. Addison-Wesley.

Kunzel, Soeren, Jasjeet Sekhon, Peter Bickel and Bin Yu. 2018. "Meta-learners for Estimating Heterogeneous Treatment Effects using Machine Learning." Working Paper.

Lam, Patrick. 2013. "Estimating Individual Causal Effects." Dissertation.

Larsen, Michael D. and Donald B. Rubin. 2001. "Iterative Automated Record Linkage Using Mixture Models." *Journal of the American Statistical Association* 96:32–41.

Levenshtein, V.I. 1965. "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals." *Doklady Akademii Nauk SSSR* 163:845–848.

Liu, Yan Y., Wendy K. Tam Cho and Shaowen Wang. 2016. "PEAR: a massively parallel evolutionary computation approach for political redistricting optimization and analysis." *Swarm and Evolutionary Computation* 30:78–92.

Magleby, Daniel and Daniel Mosesson. 2018. "A New Approach for Developing Neutral Redistricting Plans." *Political Analysis* 26:147–167.

Marinari, E. and Giorgio Parisi. 1992. "Simulated Tempering: A New Monte Carlo Scheme." *Europhysics Letters* 19:451–458.

Massey, Douglas and Nancy Denton. 1988. "The Dimensions of Racial Segregation." *Social Forces* 67:281–315.

Mattingly, Jonathan. 2017. Report on Redistricting: Drawing the Line. Technical Report. Department of Mathematics, Duke University.

Mattingly, Jonathan C. and Christy Vaughn. 2014. Redistricting and the Will of the People. Technical Report. Department of Mathematics, Duke University.

McCarty, Nolan, Keith T. Poole and Howard Rosenthal. 2009. "Does Gerrymandering Cause Polarization?" *Amerian Journal of Political Science* 53:666–680.

Nagel, Stuart S. 1965. "Simplified Bipartisan Computer Redistricting." *Stanford Law Journal* 17:863–899.

Naranjo, Oscar Mesalles. 2012. "Testing a New Metric for Uplift Models." Dissertation.

Pegden, Wesley. 2017. Pennsylvania's Congressional districting is an Outlier: Expert Report. Technical Report. Department of Mathematics, Carnegie Mellon University.

Ratkovic, Marc and Dustin Tingley. 2017. "Sparse Estimation and Uncertainty with Application to Subgroup Analysis." *Political Analysis* 25:1–40.

Ratkovic, Marc and Dustin Tingley. 2018. "The Method of Direct Estimation for Causal Inference." Working Paper.

Sadinle, Mauricio. 2014. "Detecting Duplicates in a Homicide Registry Using a Bayesian Partitioning Approach." *Annals of Applied Statistics.* 8:2404–2434.

Sadinle, Mauricio and Stephen Fienberg. 2013. "A Generalized Fellegi-Sunter Framework for Multiple Record Linkage With Application to Homicide Record Systems." *Journal of the American Statistical Association.* 108:385–397.

Samii, Cyrus, Laura Paler and Sarah Daly. 2017. "Retrospective Causal Inference with Machine Learning Ensembles: An Application to Anti-Recidivism Policies in Colombia." *Political Analysis* 24:434–456.

Shiraito, Yuki. 2016. "Uncovering Heterogeneous Treatment Effects." Working Paper.

Sinclair, Betsy, Margaret McConnell and Donald Green. 2012. "Detecting Spillover Effects: Design and Analysis of Multilevel Experiments." *American Journal of Political Science* 56:1055–1069.

Steorts, Rebecca C., Samuel L. Ventura, Mauricio Sadinle and Stephen E. Fienberg. 2014. A Comparison of Blocking Methods for Record Linkage. In *Lecture Notes in Computer Science*. Vol. 8744 Privacy in Statistical Databases pp. 253–268.

Stephanopoulos, Nicholas and Eric McGhee. 2015. "Partisan Gerrymandering and the Efficiency Gap." *University of Chicago Law Review* 82:831–900.

Tam Cho, Wendy and Yan Liu. 2016. "Toward a Talismanic Redistricting Tool: A Computational Method for Identifying Extreme Redistricting Plans." *Election Law Journal* 15:351–366.

Thibaudeau, Yves. 1993. "The Discrimination Power of Dependency Structures in Record Linkage." *Survey Methodology.*

van der Laan, Mark, Eric Polley and Alan Hubbard. 2007. "Super Learner." *Statistical Applications in Genetics and Molecular Biology* 81:1–21.

Vickrey, William. 1961. "On the Prevention of Gerrymandering." *Political Science Quarterly* 76:105–110.

Wang, Sam S.-H. 2016. "Three Tests for Practical Evaluation of Partisan Gerrymandering." *Stanford Law Review* 68:1263–1321.

Weaver, James B. and Sidney W. Hess. 1963. "A Procedure for Nonpartisan Districting: Development of Computer Techniques." *Yale Law Journal* 73:288–308.

Winkler, William E. 1988. Using the EM Algorithm for Weight Computation in the FellegiSunter Model of Record Linkage. In *Proceedings of the Section on Survey Research Methods, American Statistical Association*. pp. 667–671.

Winkler, William E. 1989. Near Automatic Weight Computation in the Fellegi-Sunter Model of Record Linkage. Technical Report. Proceedings of the Census Bureau Annual Research Conference.
URL: *https://www.researchgate.net/publication/243778219_Near_Automatic_Weight_Computation_in_the_Fellegi-Sunter_Model_of_Record_Linkage*

Winkler, William E. 1990. "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage." Proceedings of the Section on Survey Research Methods. American Statistical Association.
URL: *https://www.iser.essex.ac.uk/research/publications/501361*

Winkler, William E. 1993. "Improved Decision Rules in the Fellegi-Sunter Model of Record Linkage." In Proceedings of Survey Research Methods Section, American Statistical Association.
URL: *http://ww2.amstat.org/sections/srms/Proceedings/papers/1993_042.pdf*

Winkler, William E. 1994. Advanced Methods for Record Linkage. Technical Report. Proceedings of the Section on Survey Research Methods, American Statistical Association.

Winkler, William E. 2000. Using the EM Algorithm for Weight Computation in the Felligi-Sunter Model of Record Linkage. Technical Report No. RR2000/05. Statistical Research Division, Methodology and Standards Directorate, U.S. Bureau of the Census.

Zeileis, Achim, Torsten Hothorn and Kurt Hornik. 2008. "Model-based Recursive Partitioning." *Journal of Computational and Graphical Statistics* 17:492514.

Appendix A

Appendix for "Model Selection and Model Comparison for Predicting Heterogeneous Treatment Effects"

A.1 Kunzel et al. (2018) Meta-Learners

Algorithm 1 S-Learner

1. Estimate $Y_i = f(X_i, T_i) + \epsilon_i$
2. Generate predictions $\widehat{Y_i^T} = \widehat{f(X_i, T_i = 1)}$ and $\widehat{Y_i^C} = \widehat{f(X_i, T_i = 0)}$ for all observations
3. Generate predictions of the individual-level treatment effect as $\widehat{\tau_i} = \widehat{Y_i^T} - \widehat{Y_i^C}$

Algorithm 2 T-Learner

1. Estimate $Y_i = f_T(X_i) + \epsilon_i^T$ for treated observations, and $Y_i = f_C(X_i) + \epsilon_i^C$ for control observations
2. Generate predictions $\widehat{Y_i^T} = \widehat{f_T(X_i)}$ and $\widehat{Y_i^C} = \widehat{f_C(X_i)}$
3. Generate predictions of the individual-level treatment effect as $\widehat{\tau_i} = \widehat{Y_i^T} - \widehat{Y_i^C}$

Algorithm 3 F-Learner

1. Generate $Y_i^* = Y_i \times \frac{T_i - e(X_i)}{e(X_i)(1-e(X_i))}$, where $e(X_i)$ is an estimate of the propensity score or simply the share of treated observations in a randomized experiment
2. Estimate $Y_i^* = f(X_i) + \epsilon_i$
3. Generate predictions of the individual-level treatment effect as $\widehat{\tau_i} = \widehat{f(X_i)}$

Algorithm 4 X-Learner

1. Estimate $Y_i = f_T^Y(X_i) + \epsilon_i^T$ for treated observations, and $Y_i = f_C^Y(X_i) + \epsilon_i^C$ for control observations
2. Generate residuals for treated observations $\eta_i^T = Y_i - \widehat{f_C^Y(X_i)}$ and for control observations $\eta_i^C = \widehat{f_T^Y(X_i)} - Y_i$
3. Estimate $\eta_i^T = f_T^\eta(X_i) + \gamma_i^T$ for treated observations, and $\eta_i^C = f_C^\eta(X_i) + \gamma_i^C$ for control observations
4. Generate predictions $\widehat{\eta_i^T} = \widehat{f_T^\eta(X_i)}$ and $\widehat{\eta_i^C} = \widehat{f_C^\eta(X_i)}$ for all observations
5. Generate predictions of the individual-level treatment effect as $\widehat{\tau_i} = e(X_i)\widehat{\eta_i^C} + (1 - e(X_i))\widehat{\eta_i^T}$, where $e(X_i)$ is an estimate of the propensity score or simply the share of treated observations in a randomized experiment

A.1.1 Reasoning behind Y_i^*

Section 2.2.2 reviews a number of generic learner types for estimating heterogeneous treatment effects, including the F-learner. In the F-learner, the observed outcome Y_i is transformed into $Y_i^* = Y_i \times \frac{T_i - e(X_i)}{e(X_i)(1-e(X_i))}$, where T_i indicates treatment status and $e(X_i)$ is an estimate of the propensity score. Below, I show the logic behind this transformation and how it equals our target quantity of interest in expectation. This derivation borrows heavily from Athey and Imbens (2016).

$$
\begin{aligned}
\mathbb{E}(Y_i^* \mid X_i) &= \mathbb{E}\left(Y_i \times \frac{T_i - e(X_i)}{e(X_i)(1-e(X_i))} \mid X_i\right) \\
&= \mathbb{E}\left(T_i Y_i \times \frac{T_i - e(X_i)}{e(X_i)(1-e(X_i))} + (1-T_i) Y_i \times \frac{T_i - e(X_i)}{e(X_i)(1-e(X_i))} \mid X_i\right) \\
&= \mathbb{E}\left(T_i Y_i(1) \times \frac{T_i - e(X_i)}{e(X_i)(1-e(X_i))} \mid X_i\right) + \\
&\quad \mathbb{E}\left((1-T_i) Y_i(0) \times \frac{T_i - e(X_i)}{e(X_i)(1-e(X_i))} \mid X_i\right) \\
&= \mathbb{E}\left(\frac{T_i Y_i(1)}{e(X_i)} \mid X_i\right) - \mathbb{E}\left(\frac{(1-T_i) Y_i(0)}{1-e(X_i)} \mid X_i\right) \\
&= \frac{\mathbb{E}(T_i \mid X_i)\mathbb{E}(Y_i(1) \mid X_i)}{e(X_i)} - \frac{\mathbb{E}(1-T_i \mid X_i)\mathbb{E}(Y_i(0) \mid X_i)}{1-e(X_i)} \\
&= \mathbb{E}(Y_i(1) \mid X_i) - \mathbb{E}(Y_i(0) \mid X_i)
\end{aligned}
$$

A.1.2 Simulation Setup

Algorithm 5 Data Generating Process for Simulated Data Exercises.
Input: covariate correlation level ρ, $Y_i(0)$ DGP $\in \{\text{linear}, \text{complex}\}$, τ_i DGP $\in \{\text{linear}, \text{complex}\}$

for 1000 iterations **do**
  Draw correlation matrix $\Sigma \sim \text{Vine}(\rho)$
  Draw covariates $\mathbf{X}_i \sim \mathcal{MVN}(0, \Sigma)$

  $\beta_k \sim \mathcal{U}(-5, 5)$
  $\epsilon_i \sim \mathcal{N}(0, 1)$
  if $Y_i(0)$ DGP = complex **then**
    $Y_i(0) = \beta_1 \text{sin}(\pi X_{i5}/X_{i2}) + \beta_2 \text{log}(|X_{i1} - X_{i2}X_{i6}|) + \epsilon_i$
  else
    $Y_i(0) = \beta_1 X_{i1} + \beta_2 X_{i2} + \beta_3 X_{i5} + \beta_4 X_{i6} + \epsilon_i$
  end if

  $\gamma_k \sim \mathcal{U}(-5, 5)$
  $\eta_i \sim \mathcal{N}(0, 1)$
  if τ_i DGP = complex **then**
    $\tau_i = \gamma_1 \text{sin}(\pi X_{i5}/X_{i4}) + \gamma_2 \text{log}(|X_{i3} - X_{i4}X_{i6}|) + \eta_i$
  else
    $\tau_i = \gamma_1 X_{i3} + \gamma_2 X_{i4} + \gamma_3 X_{i5} + \gamma_4 X_{i6} + \eta_i$
  end if

  Draw treatment indicator $T_i \sim \text{Bern}(.5)$
  Calculate potential outcomes under treatment as $Y_i(1) = Y_i(0) + \tau_i$
  Calculate observed outcomes as $Y_i^{\text{obs}} = Y_i(0) + T_i \times \tau_i$
end for

Appendix B

Appendix for "Validating Ensembles of Simulated Redistricting Plans"

B.1 The Proposed Enumeration Algorithm

Algorithm 6 Proposed Validation Scheme for Redistricting Simulators.
Input: State graph G, number of precincts to sample n, number of districts to partition p, number of iterations to run sampler k, number of iterations to repeat validation exercise t

for t iterations **do**

1. Sample small map $G_{\text{sub}} \in G$ with n precincts.

2. Enumerate every partition of G_{sub} into p contiguous districts.

3. Calculate statistic of interest for each enumerated plan S_{targ}.

4. Run redistricting simulation method on G_{sub} to draw k maps partitioned into p districts.

5. Calculate statistic of interest for each simulated plan S_{sim}.

6. Compare S_{targ} to S_{sim} using chosen measure, and store that measure.

end for