Adaptive Fuzzy String Matching: How to Merge Data Sets with Only One (Messy) Identifying Field

August 5, 2020

Abstract

A single data set is rarely sufficient to address a question of substantive interest. Instead, most applied data analysis combines two or more sources of data, and in doing so encounters the record linkage problem. In practice, very rarely do two data sets contain the same identifiers with which to merge data sets; fields like name, address, and phone number may be entered incorrectly, missing, or in dissimilar formats. While recent work has made great progress in the case where there are many possible fields on which to match, the much harder case of only one identifying field is still largely unsolved. We design and validate an algorithmic solution to this *fuzzy string matching problem* rooted in adaptive learning and show that our tool identifies more matches, with higher precision, than existing solutions. Finally, we illustrate its validity and practical value through applications to matching organizations, places, and individuals.

1 Record Linkage & Data Analysis

Combining data from diverse sources is a critical component of data analysis across computational fields. Linking disease outbreak patients with water supply locations gave John Snow new insights into how cholera spreads (Snow, 1855), forming the foundation for modern public health; merging Congressmembers' campaign donation data with their votes cast addresses the influence of money in politics (Wawro, 2001); and combining student records from multiple schools enables researchers to assess the effects of educational policies many years down the line (Alicandro et al., 2018).

In the best-case scenario, both data sets share in common several unique identifying columns with identical formatting. All common data analysis software includes tools to perform this data merge. However, the best-case scenario rarely appears in practice. More commonly, the two data sets contain different and incomplete sets of identifiers, with different formatting. The first data set might contain a single variable indicating an individual's name, while the second data set might have columns for first name, middle name, last name, and suffix; the first data set might indicate organizations by their full names, while the second data set might only include abbreviations. A correct match identifies that Jenny Smith and Jennifer A. Smith are the same person, or that "JP Morgan", "Jpm and associates", and "JPM" are all the same company as "J.P. Morgan Chase and Co."

In both cases, there are two data sets each with rows representing *entities*: a person in the first example, an organization in the second. When there are multiple variables to identify which entities appear in *both* data sets – name, age, and date of birth, for example – it is called a *record linkage problem* (Fellegi and Sunter, 1969; Jaro, 1989). A large and growing literature reflects the importance of this problem, applying probabilistic and Bayesian methodologies to compute match confidences (Christen, 2005; Enamorado et al., 2019; McVeigh et al., 2019), increasing both the precision (the proportion of identified matches that link the same entities) and recall (the proportion of correct matches found to total matches that exist).

Since this merging step is preliminary to almost all applied data analysis, finding correct matches, and many of them, is consequential. Too few matches and subsequent analyses will have insufficient statistical power; if too many matches are not correct, or if they are systematically incorrect in an important way, any following results may be severely biased. This induces a methodological trade-off: researchers may choose to retain fewer matches of higher quality and bear the consequences of reduced observations, or accept a less stringent standard of match certainty and risk biasing their results.

Fuzzy String Matching

This paper offers improvements on this precision-recall trade-off for an important edge case of the record linkage problem. When the two data sets share only a single imperfect identifier, this is sometimes called the *fuzzy string matching problem* (Hall and Dowling, 1980; Filipov and Varbanov, 2019). Without leveraging similarities across multiple identifying columns, fuzzy string matching problems have fewer tools at their disposal. Those that exist fall into three categories. The most common solutions involve one or more variants of edit distance (Ristad and Yianilos, 1998). Levenshtein distance, for example, calculates the minimum number of insertions, deletions, or replacements needed to convert one string to another: "JP Morgxn" converts to "J.P. Morgan" by inserting 2 periods and replacing the accidental "x" with an "a" for an edit distance of 3. Consequently, Levenshtein distance is very effective for fixing typos, but relatively ineffective at matching "JP Morgan" to "JP Morgan Chase" since that requires 6 new insertions.

A second category of tools examines substrings: the two most common are Jaccard similarity, which measures n-gram overlap, and Jaro-Winkler distance, which examines character index proximity. Jaccard similarity might take the strings "JP Morgxn" and "JP Morgan Chase" and note that both share the 7-gram "JP Morg". Jaro-Winkler distance upweights n-gram matches early in the string, and in those ways, they both avoid the pitfall that Levenshtein distance falls into above. However, they fail where Levenshtein distance does not inasmuch as they are sensitive to typos, since both may miss "PJ Mrogan Chsae" as a misspelled match for "JP Morgan Chase."

A final tool, cosine similarity, is a composition-based measure. By representing a string as a bag-of-letters, cosine similarity creates a letter frequency vector for each of two strings and calculates the cosine of the angle between those vectors. Cosine similarity is thereby robust to switched letters, correctly matching "JP Morgan Chase" with "PJ Mrogan Chsae", but is less successful in matching "JP Morgan" to "JP Morgan Chase" like Levenshtein distance.

While there are dozens are variants of string distance tools, there is little guidance for how applied researchers can choose among them when different methods invariably produce different sets of matches. But critically, while Levenshtein distance is best at identifying typos and Jaccard similarity is best when typos are rare, most applied problems include both types of errors (Table 1). The optimal tool would intelligently use Levenshtein distance to correct types, Jaro-Winkler distance to finish incomplete strings, and cosine distance when letters are swapped, as well as any of the dozens of other tools like Sorensen-Dice distance or the Tversky index as appropriate.

String 1	String 2	Match	Levenshtein	Jaccard	Jaro-Winkler	Cosine
JP Morgan Chase	JP Morgan Chase	Yes	0	0	0	0
JP Morgan Chase	J.P. Morgan	Yes	8	0.36	0.19	0.30
JP Morgan Chase	JPM & Co	Yes	10	0.57	0.34	0.35
JP Morgan Chase	Bank of America	No	15	0.70	0.48	0.37

Table 1: Each row indicates a possible matched pair of String 1 and String 2, and contains the true match status and four different string distance metrics. In the first row, String 1 and String 2 are identical, so all distance scores are 0.

We propose an ensemble learning approach, aggregating these and many more string distance metrics to collaboratively address the fuzzy string matching problem. Drawing inspiration from (Kaufman et al., 2017) as a method for quantifying ”knowing it when you see it”, we design and implement a human-in-the-loop (HITL) algorithm for combining string matching methods into a meta-learner that optimizes both precision and recall, producing better matched data sets for applied research. We show its increased effectiveness over current best practices through a series of diverse applications, and introduce open-source software to implement our procedures. This work represents an important and widely-applicable advance in a ubiquitous problem for data-driven research.

2 An Adaptive Algorithm

The key insight to this algorithm is leveraging one of the foremost principles of human-computer interaction: have computers do what they do well, and let humans do what they do well (Kaufman et al., 2017; Lazar et al., 2017; Norman and Draper, 1986). For our purposes, and drawing inspiration from the literatures on adaptive machine learning and text-as-data (Enamorado, 2018; Miller et al., 2019), we find that while computers can quickly identify large sets of *possible* matches, only humans can quickly identify whether a proposed match is *correct* (Mozer et al., 2018).

Leveraging this, we design an algorithm of three steps. First, a computational model proposes matches. Second, a human in the loop identifies which proposed matches are correct incorrect. Third, the computer refines its model and proposes new matches to repeat the cycle. The training labels indicate whether a pair of strings is a match; the feature set consists of a number of string distance metrics. Note that while this model *does* require a training set, due to Step 2, one training set suffices for all applications. For more details on our implementation, the model's properties, and its drawbacks, see the Online Supplement.

We are agnostic as to which supervised learner our algorithm uses; we experiment with a number of options and select a random forest model as our preferred learner, and as such, our procedure is akin to a boosted trees model (Kaufman et al., 2019). After initializing the algorithm with a training set and producing a baseline model, we repeat the HITL cycle until the model consistently produces few false positive matches. For small data sets, this number of iterations may be 1 or even none; for bigger problems, it may be as many as 10.

Applications & Validation

Across three applications, we illustrate different facets and use cases of our method; in each, we show how our method produces matches robust to a wide variety of typos, abbreviations, mispuctuations, and more. The first application matches *organizations*, the second matches *geographies*, and the third matches *individuals*. The first example involves many HITL iterations, the second example just one, and the third example relies only on the baseline model with no additional researcher input. Each application we validate with comparisons to human labeling, and more details for each are available in the Online Supplement.

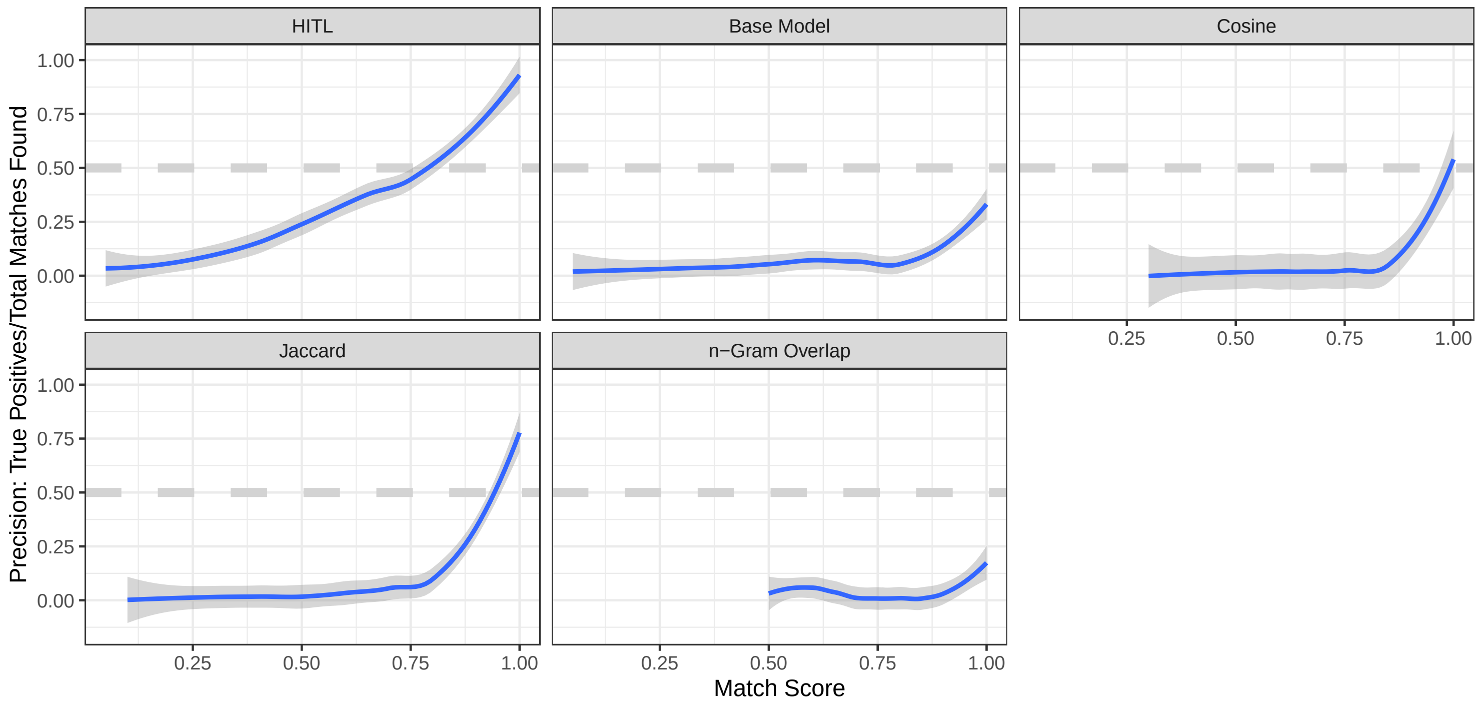

Figure 1: Match precision increases as match score increases for the HITL model, the baseline model, and three constituent measures. At a confidence of 0.95 or greater, the HITL model achieves 96.2% precision.

Matching Campaign Donors to Amicus Cosigners

Our first application is to interest group ideology. We match a data set of campaign donations to interest groups (Bonica, 2014) to a data set of amicus curiae co-signing organizations (Box-Steffensmeier et al., 2013) in the interest of studying the ideological behavior of amicus curiae (Abi Hassan et al. 2020). To validate that our method produces higher precision and recall than alternative methods, we conducted two evaluation studies. The first study addresses recall: we manually identify a number of true matches between the amicus curiae and campaign donation data, and withhold those matches from our training set; we can estimate recall as the proportion of withheld matches that a method identifies (Figure 2a). The second study addresses precision: we sample 200 proposed matches from each method and present them to human coders on Amazon.com's Mechanical Turk. We ask each respondent to examine a proposed match and identify whether that match is correct or incorrect. We triple-code each pair, and estimate precision as the proportion of model-identified matches that the majority of human coders label as correct (Figure 2b).

When the HITL model identifies a match with confidence between 0.95 and 1, it is correct 96.2% of the time; it is correct 88.0% of the time when its confidence is between 0.90 and 0.95. The next-best performing measure, Jaccard similarity, only achieves 84.7% precision at a match score of 0.95 to 1, and an accuracy of 53.0% with a score of 0.9

to 0.95. We compare AUC results for the HITL model, the baseline model, and Jaccard similarity in the Online Supplement.

Matching Misspelled Cities in PPP Data

Our second application turns from organizations to geographies. The Paycheck Protection Program (PPP), instituted as a stimulus for US small businesses in 2020, awarded loans of more than $150,000 to more than 600,000 businesses. Its publicly released loan-allocation data includes the approximate dollar amount, business name, and location of loan recipients. Importantly, the city name variable is often misspelled. First removing any cities that are spelled correctly, we compare every remaining city with the cities in list of almost 30,000 cities that are within the same state. Our procedure identifies matches among both common and uncommon misspellings: “PHILDADELPHILA” and “PHILLADELPHIA” both match Philadelphia, PA; “BERKLELEY” and “BERKLEY” both match Berkeley, CA; “BKLYN” matches to Brooklyn, NY. We successfully match “N LOS VEGAS” and “NO LOS VEGAS” to the city of North Las Vegas, NV; we correct missing or extra spacing, trailing punctuation, and even uncommon abbreviations like “PLSDS” to “PALISADES”. Above a 0.95 confidence, every identified match is a true match as per a manual coding; above 0.75, most of the incorrect matches involve cardinal direction errors (“N. HOLLYWOOD” matching to “W HOLLYWOOD”, for example). See the Online Supplement for more details.

Identifying Incumbent Voting

A final application illustrates this model’s utility *without* the HITL step. The 2016 Cooperative Congressional Election Study (Ansolabehere and Schaner, 2017) asked respondents to identify the Congressional candidate for whom they voted in 2016, and later, using geolocation data, identified each respondent’s incumbent Member of Congress. However, since the standardized version of the incumbent names rarely match respondents’ typed candidate name, determining which respondents cast votes for their incumbent is a fuzzy string matching problem. We apply our base model trained on our interest group data without any HITL steps to this problem, and find that with a confidence threshold of 0.2, it achieves both true-positive and a false-negative rates of less than 1% compared to a manual coding, indicating near-perfect accuracy (see Table B.2, Online Appendix).

3 Discussion

This paper introduces a method and associated software for merging data sets with only a single, error-prone common column. By leveraging the complementary advantages of human coders and computer models, this method is robust to a diverse set of string errors and requires no additional training data in applying it to new domains. We show that it productively applies to matching problems relating to people, places, and organizations, and argue that this robustness and broad applicability arises from its adversarial learning process – having a human in the loop iteratively adds training observations where the model performs poorly.[1] This tool is broadly useful on its own; researchers may also incorporate its confidence scores as a string distance measure in performing probabilistic record linkage.

[1]We show how the model's feature weights vary by application, and discuss potential scenarios where this algorithm performs poorly, in the Online Supplement.

References

Alicandro, G., Frova, L., Sebastiani, G., Boffetta, P., and La Vecchia, C. (2018). Differences in education and premature mortality: a record linkage study of over 35 million italians. *The European Journal of Public Health*, 28(2):231–237.

Ansolabehere, S. and Schaner, B. (2017). Cooperative congressional election study, 2016: Common content, release 2. *Cambridge, MA: Harvard University.*

Barberá, P. (2015). Birds of the same feather tweet together: Bayesian ideal point estimation using twitter data. *Political Analysis*, 23(1):76–91.

Bonica, A. (2014). Mapping the ideological marketplace. *American Journal of Political Science*, 58(2):367–386.

Box-Steffensmeier, J. M., Christenson, D. P., and Hitt, M. P. (2013). Quality over quantity: Amici influence and judicial decision making. *American Political Science Review*, 107(03):446–460.

Christen, P. (2005). Probabilistic data generation for deduplication and data linkage. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 109–116. Springer.

Enamorado, T. (2018). Active learning for probabilistic record linkage. *Available at SSRN 3257638.*

Enamorado, T., Fifield, B., and Imai, K. (2019). Using a probabilistic model to assist merging of large-scale administrative records. *American Political Science Review*, 113(2):353–371.

Fellegi, I. P. and Sunter, A. B. (1969). A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210.

Filipov, L. and Varbanov, Z. (2019). On fuzzy matching of strings. *Serdica Journal of Computing*, 13(1-2):71–80.

Gentzkow, M. and Shapiro, J. M. (2010). What drives media slant? evidence from us daily newspapers. *Econometrica*, 78(1):35–71.

Hall, P. A. and Dowling, G. R. (1980). Approximate string matching. *ACM computing surveys (CSUR)*, 12(4):381–402.

Jaro, M. A. (1989). Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420.

Kaufman, A., King, G., and Komisarchik, M. (2017). How to measure legislative district compactness if you only know it when you see it. *American Journal of Political Science.*

Kaufman, A. R., Kraft, P., and Sen, M. (2019). Improving supreme court forecasting using boosted decision trees. *Political Analysis*, 27(3):381–387.

Lazar, J., Feng, J. H., and Hochheiser, H. (2017). *Research methods in human-computer interaction.* Morgan Kaufmann.

Martin, A. D. and Quinn, K. M. (2002). Dynamic Ideal Point Estimation Via Markov Chain Monte Carlo for the US Supreme Court, 1953—1999. *Political Analysis*, 10(2):134–153.

McVeigh, B. S., Spahn, B. T., and Murray, J. S. (2019). Scaling bayesian probabilistic record linkage with post-hoc blocking: An application to the california great registers. *arXiv preprint arXiv:1905.05337.*

Miller, B., Linder, F., and Mebane, W. R. (2019). Active learning approaches for labeling text: Review and assessment of the performance of active learning approaches. *Political Analysis*, pages 1–20.

Mozer, R., Miratrix, L., Kaufman, A. R., and Anastasopoulos, L. J. (2018). Matching with text data: An experimental evaluation of methods for matching documents and of measuring match quality. *arXiv preprint arXiv:1801.00644*.

Norman, D. A. and Draper, S. W. (1986). *User centered system design: New perspectives on human-computer interaction.* CRC Press.

Ristad, E. S. and Yianilos, P. N. (1998). Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532.

Snow, J. (1855). *On the mode of communication of cholera.* John Churchill.

Wawro, G. (2001). A panel probit analysis of campaign contributions and roll-call votes. *American Journal of Political Science*, pages 563–579.

Appendices

A Human in the Loop Model

Building our Human in the Loop (HITL) model of fuzzy string matching involves three steps: First, a computational model proposes matches. Second, a human in the loop identifies which proposed matches are correct or incorrect. Third, the computer refines its model and proposes new matches to repeat the cycle. This section explains these steps, as well as the initialization step, in greater detail.

Features. In selecting features, standard practice is to include as many as possible so long as no two are perfectly correlated. In practice, since feature generation is the most time-intensive part of this process, we select one of each broad type of feature, noting that within categories of string distances, any two measures are at least moderately correlated. In this paper, our model includes the following features: string overlap, Jaccard similarity, cosine similarity, Levenshtein distance, and longest common substring.

Initialization & Initial Model. Initializing the HITL model requires a training set. We produce one (and only one) training set. For 1000 of our amicus curiae organizations, we tasked Mechanical Turk workers with locating in the FEC list one or more matching organizations. This resulted in 453 positive matches for 199 unique amicus curiae cosigning organizations. We then consider every combination of amicus organizations and bonica organizations in the hand-labeled sample, and all pairs not in the original hand-labeled matches are labeled as negative cases. Our final training set is 71,043 observations. We train a random forest model using the `Sklearn` Python library with cross validation and grid search on nine sets of hyper-parameters.

Human in the loop & Model Updating To perform the HITL iteration, our model produces a list of the top 500 proposed string pairs ordered by match confidence, though users may decide on any number, and manually identify which of those are true positives and which are false-positives. We append the false positive observations to the initial training set of 71,043 observations, and then label every combination of strings produced in the true-positive matches in a manner similar to how we build the training set. Lastly, we retrain the random forest model.

Algorithm Notation

Consider an HITL process with I iterations, a strings in the first set and b strings in the second set, and X string distance metrics. Each HITL iteration will examine the n predictions.

Algorithm 1: Human-in-the-loop algorithm

```
Calculate $X$ for all combinations of $a, b$;
Construct initial training set $Y_{train}, X_{train}$ using observations $train \in X$; Train initial model $M_1$ by
  regressing $Y_{train}$ $X_{train}$ ;
Generate predictions $\hat{Y}_1$ using $M_1, X \setminus X_{train}$;
for $i$ in $I$ do
    Sort predictions $\hat{Y}_i, X_i$, descending;
    Extract the first $n$ predictions from sorted $\hat{Y}_i, X_i$ to $\hat{Y}_{i,n}, X_{i,n}$;
    Manually correct $\hat{Y}_{i,n}$ to $Y_{i,n}$;
    Update training set $Y_{train}, X_{train} = Y_{train}||Y_{i,n}, X_{train}||X_{i,n}$;
    Train new model $M_{i+1}$ by regressing $Y_{train}$ $X_{train}$;
    Generate predictions $\hat{Y}_{i+1}$ using $M_{i+1}, X \setminus X_{train}$;
end
```

Why it works

The HITL model works well by combining insights from two important strands of modeling: adversarial learning, and gradient boosting. Adversarial learning involves providing a model with deliberately difficult cases designed to trick it into making incorrect predictions, then adding training data to help the model in those edge cases. Likewise, boosting models involves training many models in sequence, where each model's training observations are reweighted in proportion to its residuals. This has the effect of producing models that perform well in precisely the cases that previous models perform poorly, then ensembling those models together. As an example of why the HITL component of this model is so important, the naive model before any HITL iterations identifies National Automobile Association and National Autism Association as matches with a confidence of 0.92; after a HITL iteration, that confidence drops to 0.37.

An alternative perspective on why this process works is that it serves as a pre-processing tool for manual matching. The current cutting-edge method for fuzzy string matching is to take a single measure like Jaccard similarity and sort all potential pairs according to that measure. The researcher may then manually examine that list of pairs and discard the ones she determines to be false positives. With unlimited time the researcher may look through the entire list, quickly finding fewer and fewer true positive matches. The HITL model works well because it iteratively produces new lists of top potential matches, including cases where a true match may have a very low Jaccard similarity score. The number of true positives the researcher identifies per pair considered is much higher with the HITL model, so even in the context where the researcher manually vets a large number of potential matches, the HITL model saves time and produces more true positives.

When it fails

Though our HITL algorithm performs exceedingly well in the applications we consider, we note places where it may fail. In particular, it may fails when many strings are highly similar to each other and when the matching problem under consideration is many-to-one.

A Computational Note

The model's most computation-intensive step is feature generation since the model considers whether *each string* in the first data set is a match for *each other string* in the second data set. The complexity of this step is proportional to the number of strings in the first data set multiplied by the number of strings in the second data set, so as the string matching problem increases, the computation time increases very quickly. Relative to the feature generation, the modeling takes little time. Consequently, we recommend in the feature generation step that users leverage the problem's embarrassingly-parallel nature and constrain the possible feature combinations with covariates (as in the city matching example).

B Applications

This section details our applications. In identifying the best method for selecting matches between any two data sets, we consider two key evaluation metrics: precision and recall. Precision captures how many identified matches are true, and is calculated as the number of true positives divided by the number of true positives plus the number of false positives; recall captures the proportion of true matches that the model identifies, and is calculated as the true positives divided by the sum of the true positives and false negatives. Both are important criteria for subsequent analyses: higher precision reduces bias, while higher recall improves statistical power. [2]

Interest Group Ideology

Our first application relates to ideal point estimation. A widely-studied and broadly important subfield in political science, ideal point estimation seeks to quantify and measure the political ideologies of diverse actors ranging from Congressional candidates (Bonica, 2014), Supreme Court judges (Martin and Quinn, 2002), news organizations (Gentzkow and Shapiro, 2010), and Twitter users (Barberá, 2015). Abi Hasan et al (2020) extend this literature to the study of interest group ideology. Using a network of amicus curiae cosigning behavior (Box-Steffensmeier et al., 2013) linked to a small number of existing interest group ideal points from 2014, they develop a novel imputation

[2]Note that we choose to optimize precision at the expense of recall, since a smaller sample size biases us against finding substantive results, but researchers using this tool can adjust the algorithm by loosening certainty thresholds and performing fewer model iterations to prioritize recall instead.

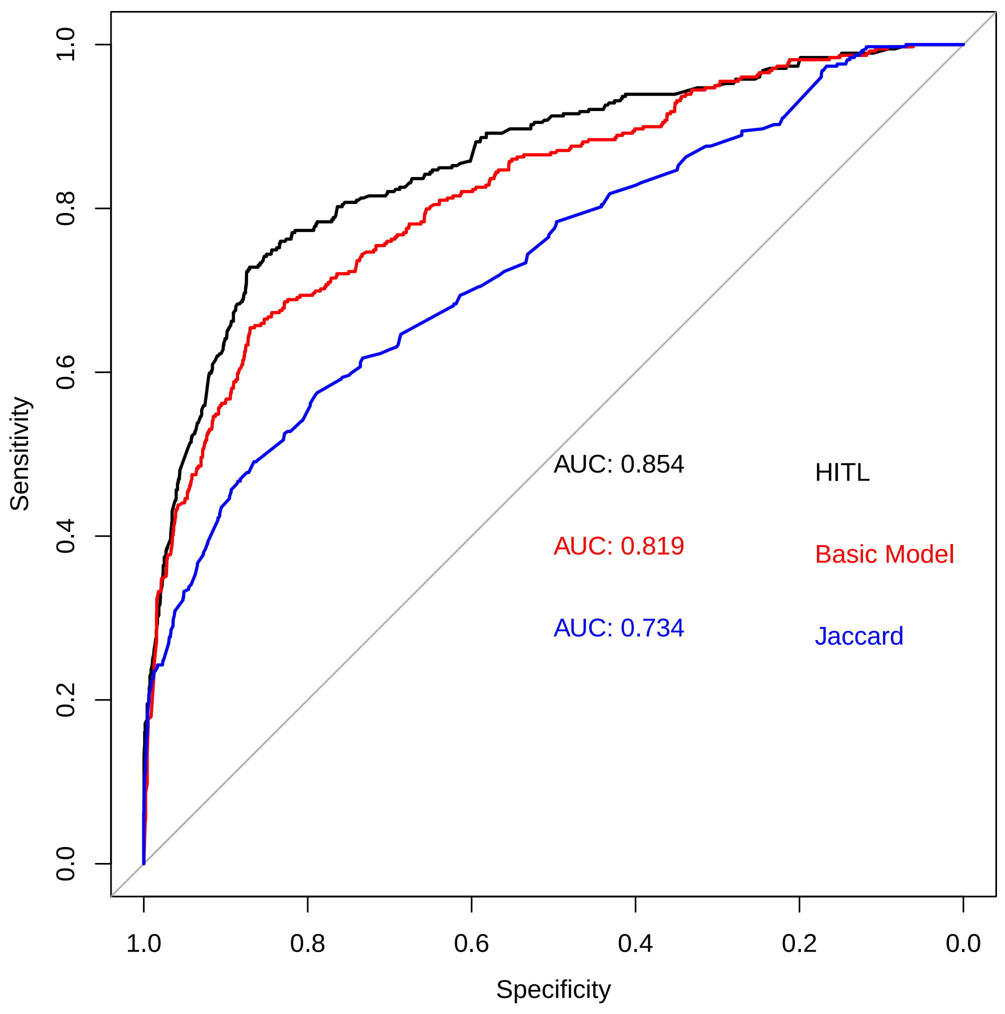

Figure B.1: AUC for the HITL model (black), baseline model (red), and Jaccard similarity, the best-performing single measure (blue).

procedure to estimate ideal points for more than 15,000 interest groups by leveraging network links between groups with pre-existing ideal points and those without. To do so, they first merge two large data sets: (1) 13,939 amicus curiae co-signing organizations, and (2) FEC-derived campaign donation records for 1,332,470 organizations and individuals. The more matches found between these two data sets, the better the network imputation results will be. There are 1,767 organization names that match perfectly, but is is unknown how many other matches exist between these two data sets.

US City	PPP Entry	Confidence
HAGERSTOWN	HAGERTOWN,	0.98
SOUTH SALT LAKE CITY	SALT LAKE CITY	0.95
WEST NORRITON	NORRIESTOWN	0.87
MIAMI BEACH,	N MIAMI BEACH	0.86
MC KINNEY	MCKINEY	0.78
CORPUS CHRISTI	CORPUS CHRISI, TX	0.73
SOUTH WEYMOUTH	S WEYMOUTH	0.71
COMMERCE TOWNSHIP	COMMERCE TWP	0.63
SHELBY CHARTER TOWNSHIP	ROCHESTER HILS	0.62
SOUTH SAN FRANCISCO	S SAN FRAN	0.57
SCHEREVILLE	REELSVILLE	0.51
HAINESPORT	LAWRENCE TOWNSHIP	0.49
LAKEWOOD RANCH	LAKEWWOD RCH	0.43
HARWICH	SOUTH HARWICH	0.40
MASPETH	PT	0.35
FREEHOLD TOWNSHIP	SEWEL	0.23
FRAISER	TRAVERSE CITYF	0.18
PORT CANAVERAL	PNACEA	0.15
CARONA	CHIO	0.13
MOORESTOWN	BEACHHAVEN	0.07
KEW GARDEN	OLIVEBRIDGE	0.00

Table B.1: A random sample of results from the PPP application.

City Name Correction

Our second application involves correcting city names in the PPP database. The base model performs well: of the 506 pairs proposed during the HITL step, 503 are true positives. Table B.1 below presents a random sample of results from our HITL model. After removing exact matches, we look for cities in the PPP database that match known US city names within the state identified in the PPP database: in matching "CHICAGAO" we only look at known US city names in Illinois.

Incumbent Voting

Table B.2 below presents a random sample of results from our HITL model applied to identifying whether a CCES respondent voted for the incumbent. We examine the CCES cumulative file and, removing rows with missingness in either the "voted_rep_chosen" or "rep_current" columns. We are left with 5,459 observations in our analysis. For this application we do not perform any HITL iterations; rather, we use the naive model trained using amicus curiae and FEC data. We see the model is robust to many types of differences: missing middle names, first name shorthand, varying capitalization, varying punctuation, typos ("Grace Napolitano" vs "Grace Napo**k**itano"), honorifics, and nicknames.

Incumbent	R’s Vote	Confidence
Jason T. Smith	Jason Smith	0.97
Janice D. Schakowsky	Jan Schakowsky	0.94
Bobby L. Rush	Bobby Rush	0.90
James McGovern	Jim McGovern	0.84
Jerry McNerney	Jerry Mcnerney	0.81
Grace Napolitano	Grace Napokitano	0.75
Sheila Jackson Lee	Sheila Jackson-Lee	0.69
Anh ’Joseph’ Cao	Joseph Cao	0.52
Earl ”Buddy” Carter	Buddy Carter	0.50
Howard P. ’Buck’ McKeon	Howard McKeon	0.45
John C. Fleming, M.D.	John Fleming	0.37
Robert C. ’Bobby’ Scott	Robert Scott	0.35
Sanford D. Bishop, Jr.	Sanford Bishop	0.28
Henry ”Hank” Johnson, Jr.	Hank Johnson	0.23
Michael Quigley	Mike Quigley	0.21
Tim Mahoney	Tom Rooney	0.16
William Delahunt	William Keating	0.14
Robert Hurt	Robert Wittman	0.08
Kevin Brady	Kent Hargett	0.00

Table B.2: A random sample of results from the CCES application. The mid-table line indicates the cutoff between matches and non-matches.

Model	Cosine	Jaccard	Levenshtein	LCSSTR	Overlap
Amicus Training Set	0.417	0.215	0.224	0.074	0.069
Amicus + HITL	0.227	0.351	0.181	0.087	0.154
PPP City Names HITL	0.291	0.105	0.585	0.003	0.017
CCES Incumbent Names	0.360	0.308	0.189	0.079	0.065

Table B.3: Feature importances for 3 applications as compared to the model produced prior to HITL iterations.

Feature Importances

Tree-based models do not incorporate parametric coefficients like OLS, so we rely on alternative measures to gauge how important individual features are to the model’s overall performance (for a discussion of the varieties of feature importance metrics for tree-based models, see Kaufman et al. (2019)). Importantly, we find that across our various applications, feature importances differ in substantial and systematic ways. The baseline model using only the amicus curiae training set overwhelmingly relies on cosine similarity, but adding the HITL step substantially elevates the importance of Jaccard similarity and, relatively speaking, the Overlap measure. In matching city names, Levenshtein is the most important measure, while LCSSTR is almost entirely unused.

Changes in Accuracy as HITL Iterations Increase

A useful quantity of interest is the change model accuracy from n to $n+1$ HITL iterations. This quantity has important implications for how many iterations a researcher should perform to achieve a desired accuracy. Unfortunately, estimating this quantity is a complex challenge. Each HITL iteration removes true positives and true negatives from the pool of remaining matches, and importantly, often removes the *easiest* pairs to classify. As a result, accuracy may appear to *decrease* from one iteration to the next, despite the model successfully classifying harder cases. We prefer to measure the raw number of new true-positives for each iteration, but this measure too has pathologies: some of our applications have very few total true-positives, so this number may drop off very quickly.