Online Supplement:

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

Aaron R. Kaufman* Aja Klevs†

February 10, 2021

Abstract

A single data set is rarely sufficient to address a question of substantive interest. Instead, most applied data analysis combines data from multiple sources. 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. Combining multiple data sets absent a unique identifier that unambiguously connects entries is called the record linkage problem. 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 remains unsolved: this *fuzzy string matching problem*, both its own problem and a component of standard record linkage problems, is our focus. We design and validate an algorithmic solution called Adaptive Fuzzy String Matching 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.

*Division of Social Sciences, New York University Abu Dhabi, Saadiyat Island, Abu Dhabi, UAE; AaronRKaufman.com; aaronkaufman@nyu.edu

†New York University, Center for Data Science

Appendices

A Human in the Loop Model

Building our AFSM model for 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 (see below), 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, AFSM includes the following features: string overlap, Jaccard similarity, cosine similarity, Levenshtein distance, and longest common substring.

Initialization & Initial Model. Initializing AFSM 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 using a pre-existing Mechanical Turk format where users input substrings and drop-down lists are automatically populated with relevant organizations. For example, if an MTurker is asked to find matches in Bonica's DIME database (Bonica, 2014) of campaign donations for the American Civil Liberties Union, they may type "civil" into the search bar and a drop-down list will present a number of DIME organizations containing that substring. By having three MTurkers search for matches in the DIME data set for each of our 1000 amicus curiae organizations, we mitigate concerns that we are systematically missing sets of true matches. However, we cannot rule out this possibility without having a complete set of true matches to compare to; researchers especially concerned about missing as few matches as possible may choose to add additional coders for each organization.

Our MTurk triple-coding procedure resulted in 453 positive matches for 199 unique amicus curiae cosigning organizations (out of 1000). 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.

We find in practice that this single training set, supplemented by one or more HITL steps, performs impressively for across our range of applications, and recommend that researchers interested in using this tool first test our pre-trained model on their data. If they find after one to two HITL iterations that match quality is poor and

not measurably improving, we recommend following the steps we outline above to produce their own training sets, possibly supplementing ours with their own. We are happy to assist researchers in this procedure, and may use any new training sets to supplement our own in our published software packages in the future.

Human in the loop & Model Updating To perform the HITL iteration, our AFSM model produces a list of the top 500 proposed string pairs ordered by match confidence, though users may decide on any number, and we the researchers manually identify which of those 500 proposed string pairs are true positives and which are false-positives. In selecting the number of proposed string pairs, we make two considerations: the first is mitigating class imbalance, and the second is ensuring that we capture and hand-code the cases for which the model is most uncertain. Class imbalance, whereby our training set has many more true negative than true positive observations, is a danger only insofar as it produces more conservative match probabilities. Typically very few of the 500 most likely matches are extremely high probability positive matches, and most are very low probability positive matches. By sampling the 500 most likely, we can ensure that we include all the true positives to help offset the class imbalance problem of many more true negatives than true positives, while also ensuring that we capture all the model's most uncertain cases as well.

After hand-coding, 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. In our Incumbent Voting application, we find that retraining our model is unnecessary because the 500 proposed string pairs contain nearly every true-positive match in the data set. This is because the set of possible match combinations is small enough to predict all at once. Generally, we recommend that researchers' first HITL iteration contain a relatively small number of possible match pairs, but if the researcher's computational resources are sufficient and their matching task sufficiently similar to the training data, they may consider this shortcut.

In determining how to sample observations for an HITL iteration, consider two sets of names, $N_1 = 1,000$ and $N_2 = 1,000,000$, and a task to identify the entries in N_2 corresponding to the 1,000 entities in N_1. There are two obvious ways to sample potential pairs for conducting an iteration: a subsample of N_1 and all of N_2, or all of N_1 and a subsample of N_2.

If we conduct an HITL iteration consisting of 1 entry from N_1 and all 1,000,000 entries of N_2 (for 1,000,000 potential matches), we are guaranteed to find a match for that entry if it exists, and we are very unlikely encounter a situation where the 500 hand-coded pairs contain no true-negatives, but we learn relatively little information and exacerbate class imbalance because there will be very few true-positives but very many true-negatives.

Instead, we choose to sample all 1,000 observations of N_1 and the first 1,000 observations of N_2 (still 1,000,000 potential matches). While we no longer guarantee finding any true-positives and we do run the risk of falling into the no-true-negatives scenario above, we get superior class balance and find many more true-positives per iteration,

improving the model's per-iteration performance boost. We recommend that researchers examine the results of their first HITL iteration and consider which sampling method may work best for them.

We acknowledge that there are ways the HITL step may go wrong. First, there is the possibility that in the HITL steps, we discard some false-negative matches – that is, some true matches are mistakenly labeled as non-matches. As we show below, any remotely similar pair of strings will receive a match probability in the first 500, the bottom of which include radically different strings:"LAUDEHILL" vs "ROBYN GYURU" and "NANTY GLO" vs "HUNTINGDON VALLEY" both appear in the top 500 matches. Any false-negative that is more dissimilar than those pairs, while desirable to include in the final data set, may be undesirable to include in the HITL iteration as it may erode the model's performance. Secondly, it is possible that non-matches are mistakenly labeled as true matches. For example, even an attentive human may mistakenly match "ACME Bread Co" and "ACME Bead Co." Were they different companies, we may still want to throw out this proposed match instead of adding it to the set of true negatives lest we degrade the model's accuracy since this is generally a type of error we want to reward our model for catching.

Computational Note. AFSM has a complexity of $O(n \times m)$ where n is the length of the first data set and m is the length of the second data set. 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 to compute with multiple threads, and also to constrain the possible feature combinations with covariates (as in the city matching example).

Feature Definitions

String overlap coefficient. The Overlap Coefficient is a string distance metric which considers each string as a set of characters. That is, the order of characters does not matter and each character is only counted once, regardless of how many times it occurs in the string. For instance, both the string "this_is_a_test" and the string "at_the_ties" could be represented as {h,s,i,t,_,a,e}.

The Overlap coefficient is the fraction of characters in the shorter string which are also in the longer string. Let A, B be the set of characters in strings a, b respectively. Then the overlap coefficient for a and b, $overlap(a, b)$ is defined as:

$$overlap(a, b) := \frac{|A \cap B|}{\min(|A|, |B|)}$$

Jaccard index. Jaccard Index is very similar to the Overlap Coefficient, and also considers strings as sets of letters. However we calculate Jaccard Index by finding the number of characters occurring in both strings, and divide by the number of characters in the set of both strings combined. In other words, the Jaccard Index is the fraction of total characters which exist in both strings. Using the same notation as with Overlap Coefficients, The Jaccard Index for strings a and b, $Jaccard(a, b)$ can be written as:

$$Jaccard(a, b) := \frac{|A \cap B|}{|A \cup B|}$$

Cosine similarity. Cosine Similarity represents each string as a vector in n dimension space, where n is the number of distinct characters present in either string. It is a measure of the cosine of the angle between these vector representations. If the cosine similarity is close to 1, then the angle between the words is small and they are close together in latent space. If the cosine similarity is close to 0, then the vector representations of the strings are close to being orthogonal, which means they have very different compositions.

Like Overlap Coefficient and Jaccard Index, Cosine Similarity does not take into account the order of characters, but unlike Overlap and Jaccard it considers multiplicities in characters.

Let $\mathbf{a}, \mathbf{b}$ be the vector representation of strings a and b respectively. Then the cosine similarity between a and b, $cos(a, b)$ is:

$$cos(a, b) := \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||\ ||\mathbf{b}||}$$

where $\cdot$ is the standard inner product and $||\ ||$ is the L2 norm.

Levenshtein distance. To calculate the Levenshtein distance between two strings, we count the number of single-character edits it takes to get from one string to another. Single-character edits are insertions, deletions, and substitutions.

Formally, let's consider strings a, b. Let a_i, b_j be the characters at index i, j for a and b respectively, and let $a[:i], b[:j]$ to be the first i characters of a and the first j characters of b respectively. the Levenshtein distance

between $a[:i]$ and $b[:j]$, $lev_{a,b}(i,j)$ is defined as follows:

$$lev_{a,b}(i,j) := \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0 \\ \min \begin{cases} lev_{a,b}(i-1,j) \\ lev_{a,b}(i,j-1) \\ lev_{a,b}(i-1,j-1) + \mathbf{1}_{a_i=b_j} \end{cases} & \text{otherwise} \end{cases}$$

Where $\mathbf{1}_{a_i=b_j}$ is the indicator function for whether $a_i = b_j$. Note $lev_{a,b}(i,j)$ is defined recursively, and we are only interested in $lev_{a,b}(|a|,|b|)$ where $|a|, |b|$ are the lengths of strings a and b respectively.

Longest common substring. A substring s of a string a is a contiguous sequence of characters within a. We denote that s is a substring of a by $s \subseteq_s n$.The longest common substring for strings a, b is the length of the longest string which is a substring for both a and b. We denote the longest common substring between a and b as $lcssts(a,b)$, and it is formally defined as:

$$lcssts(a,b) := \max(\{|s| : \ s \subseteq_s a \text{ and } s \subseteq_s b\})$$

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 AFSM algorithm performs exceedingly well in the applications we consider, we note places where it may fail. Our tool's success depends largely on the types of string errors that might exist. We note how Levenshtein distance is good at fixing single-letter typos while Jaccard similarity is good at cases where one substring is missing entirely ("JP Morgan" vs "JP Morgan Chase"). If the type of error that is most common in a researcher's data set is not well-addressed by the distance measures in our ensemble, it is likely to fail. This may happen, for example, when working in non-English languages. In this context, we recommend that researchers identify what types of errors are most prominent and locate a string distance measure that may be suitable. Our replication file and training set data make it easy for researchers to add new string distance measures and retrain our model should they encounter this problem.

Another problem that may arise is where two extremely similar strings are not true matches. This may arise for example with "CBS Inc" and "CBSC Inc", and may be common in large databases of individuals where rows may have the same or similar full names despite being different people. This is the most common type of false-positive match. For this reason we believe it is important for the researcher to vet the model's identified matches.

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. [1]

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, Christenson and Hitt, 2013) linked to a small number of existing interest group ideal points from 2014, they develop a novel imputation 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.

Since both the amicus organization list and Bonica's donor list are so large, in order to guarantee that each HITL contains at least some true positive matches, we select as our HITL prediction set a list of 100 amicus organizations and compare them all to every organization in Bonica's set. Even so, this is a formidable computational task. We perform 10 iterations of this HITL step. We present our final accuracy in Figure B.1.

We observe that the Base Model underperforms Jaccard similarity in Figure 1, but exceeds it in the Figure B.1. To explain this, we note that the Base Model does consistently better than Jaccard for lower levels of confidence, though it underperforms it by quite a bit in the higher levels. But to R3's larger point, we note that Figure 1 and Figure B.1 consider entirely different sets of metrics. Figure 1 considers precision, which is the percentage of pairs

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

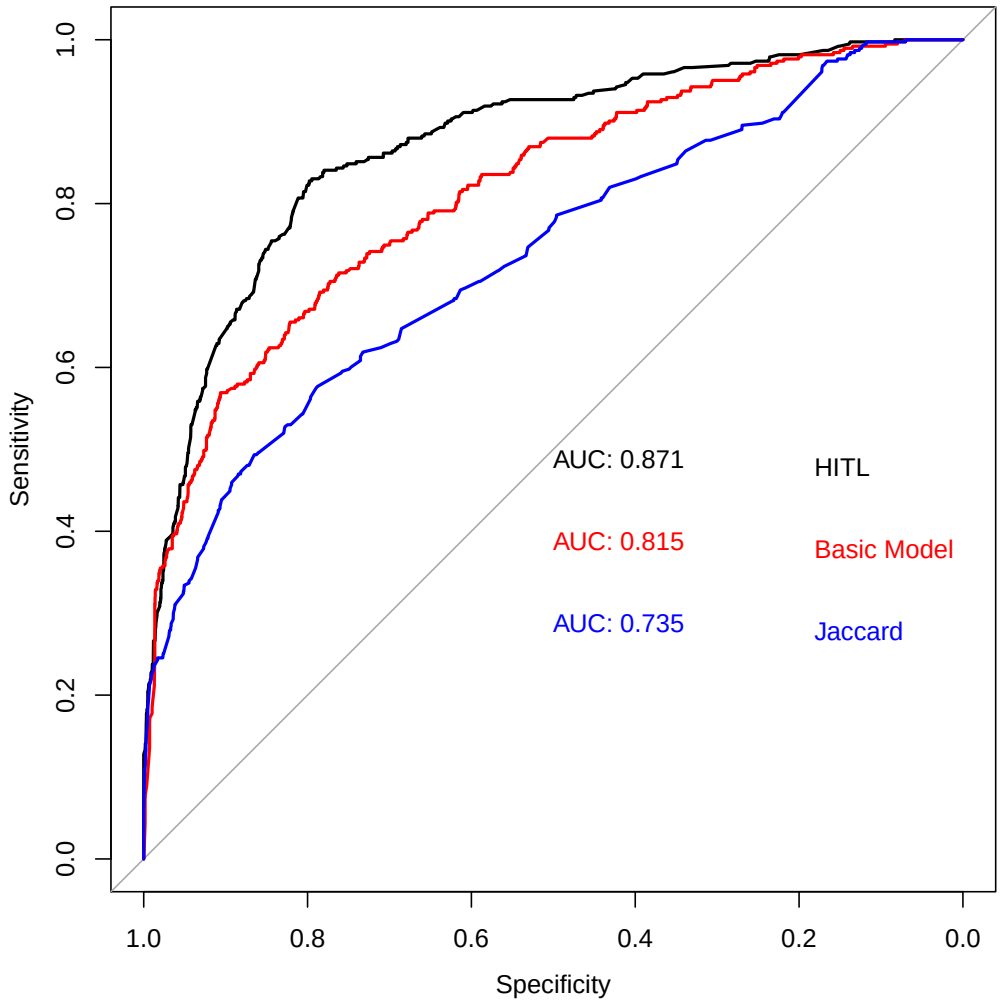

Figure B.1: AUC for the HITL model (black), baseline model (red), and Jaccard similarity, the best-performing single measure (blue).

PPP Entry	US City	AFSM Confidence
MOUNT VERNON	MOUNT VERNON	1.00
LOS ANGELESE	LOS ANGELES	0.90
ROCHESTER NY	ROCHESTER	0.85
SAINT BONIFACIUS	ST. BONIFACIUS	0.77
SHARFTER	SHAFTER	0.73
GOOD YEAR	GOODYEAR	0.67
HOFFMAN EST	HOFFMAN	0.57
ALDIE	LAKESIDE	0.52
MONTALBA	ALMA	0.40
MAUREPAS	AMA	0.36
CREAM RIDGE	ERMA	0.28
AUS	SUN VALLEY	0.26
OZARK	BOWEN	0.15
WOODDALE	SAVANNA	0.11
VESTAL	NISSEQUOGUE	0.01

Table B.1: A random sample of results from the PPP application, ordered by the score from our AFSM model.

labeled as matches that are true-positives (calculated as true-positives / (true-positives + false-positives)). Figure B.1 considers AUC, which considers sensitivity (true-positives/(true-positives + false-negatives)) and specificity (true-negatives/(true-negatives + false-positives)). The reason Jaccard has higher precision but worse AUC is that it assigns very few potential matches to high levels of confidence, whereas the Base Model identifies more matches (both true-positive and false-positive) but gets more of them wrong. Generally, we believe that AUC is a stronger indicator of quality than precision for most research cases.

To see the relative impact of each HITL step, and how our accuracy improves from the base model to the final iteration, please see the subsection "Changes in Model Performance Across HITL Iterations".

City Name Correction

Our second application involves correcting city names in the PPP database as compared to their official names in a list of US cities. We use a single HITL step of 500 city names, and find the base model performs well: of the 506 pairs proposed during the first HITL step, 503 are true positives (note that one city name in the list of US cities may match to multiple cities in the PPP database). 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.

The next best method, Jaccard similarity, has 8 (false-positive) matches above a 0.95 confidence and only 286 total proposed matches above 0.9, as compared to 69 and 253 for AFSM. Above 0.7, most AFSM errors are cardinal direction errors, whereas Jaccard has many more errors above that threshold. We present a random sample of Jaccard

PPP Entry	US City	Jaccard
WOODRIDGE	RIDGEWOOD	1.00
KIGHTDALE	KNIGHTDALE	0.90
WHITE POST	WEST POINT	0.82
OREM,	OREM	0.80
DANIELSVILLE	FRIENDSVILLE	0.71
SAINT REGIS	TWIN BRIDGES	0.64
LORDAN	WOODLAND	0.56
CRANBERRY	BERWYN	0.50
LORAINE	KIDRON	0.44
DALLLAS	LINDALE	0.40
HALLANDLE BCH	VENICE GARDENS	0.29
NEW LEBANON	ST. JOHNSVILLE	0.25
GLENWOOD	BUTLER	0.17
UNIONVILLE	BASS LAKE	0.12
MICO	SPADE	0.00
TABERG	LIC	0.00

Table B.2: A random sample of results from the PPP application ordered by Jaccard similarity.

results below (Table B.2).

Incumbent Voting

Table B.3 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.

The highest-confidence matches according to our feature string distance metrics are typically correct as well. However, they both find fewer matches at a suitable threshold of confidence and have lower precision. The best-performing, Jaccard similarity, identifies only 151 total matches including 1 false positive, while AFSM finds 233 with 1 error, approximately 50% more true matches.

Feature Importance

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, Kraft and Sen (2019)). Importantly, we find that across our various applications, feature importances differ in substantial and systematic ways. The baseline model using

Incumbent	R's Vote	Confidence
Cynthia M. Lummis	Cynthia Lummis	1.00
Ed Royce	Edward Royce	0.92
Jim Sensenbrenner	James Sensenbrenner	0.88
Steve Chabot	Stave Chabot	0.85
Sheila Jackson Lee	Sheila Jackson-Lee	0.81
Jim Clyburn	James Clyburn	0.76
Harold 'Hal' Rogers	Hal Rogers	0.51
Anh 'Joseph' Cao	Joseph Cao	0.49
Sanford Bishop	Sanford D. Bishop Jr.	0.45
Francisco 'Quico' Canseco	Francisco Canseco	0.40
Nicola S. 'Niki' Tsongas	Niki Tsongas	0.36
William 'Lacy' Clay, Jr.	Wm Lacy Clay	0.29
Hank Johnson	Henry "Hank" Johnson, Jr.	0.24
Michael Turner	Mike Turner	0.23
Bob Brady	Robert Brady	0.15
Mike Fitzpatrick	Brian Fitzpatrick	0.12
John Dingell	Debbie Dingell	0.08
Ted Poe	Pat Bryan	0.01

Table B.3: 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
Base Model	0.26	0.23	0.30	0.07	0.15
10 HITL Iterations	0.32	0.25	0.24	0.09	0.11
PPP City Names HITL	0.40	0.31	0.22	0.05	0.02

Table B.4: Feature importances for our two applications as compared to the base model produced prior to HITL iterations.

only the amicus curiae training set primarily relies on Levenshtein distance, but adding the HITL step substantially elevates the importance of Jaccard similarity and cosine similarity. In matching city names, cosine distance is the most important measure, while LCSSTR and Overlap are almost entirely unused. Importantly, we only conduct a single HITL iteration for our PPP application, so all the changes in feature importance are the result of that single iteration. The are no feature importances for the incumbency application because we did not do a HITL iteration for this application; we use the base model by itself.

Changes in Model Performance Across HITL Iterations

A useful quantity of interest is the change in 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. Keeping in mind the problem that each AUC figure is calculated using a different denominator, we present in Figure B.2 below the progression of AUC scores for each of the 10 HITL iterations of our amicus curiae application.

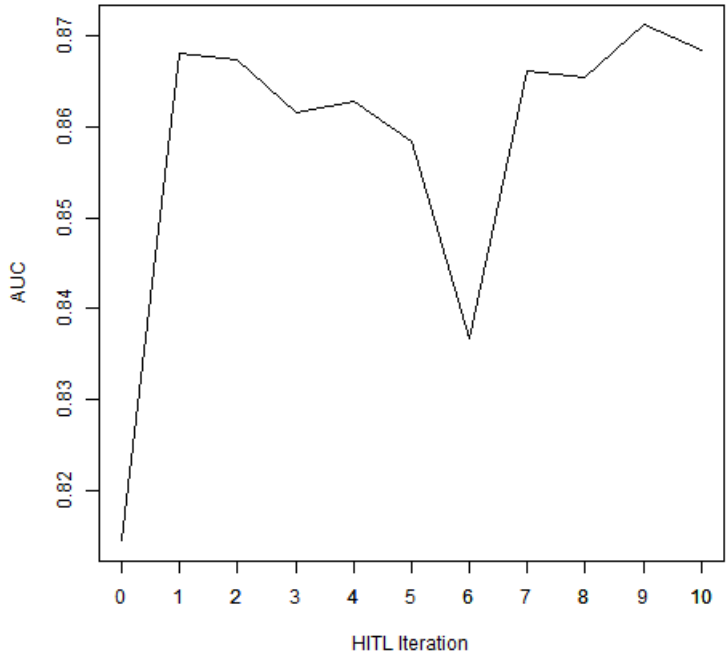

Figure B.2: As the number of HITL iterations increases, AUC increases generally, though not monotonically.

After an initial small decrease in accuracy over the first three iterations, AUC increases from 0.80 to nearly 0.87 over the next four iterations, decreases slightly for the next two, then appears to level off in the final iteration.

These changes in accuracy reflect changes in the ensemble's underlying feature weights as its training set grows to more strongly represent the target set. Table B.5 below illustrates this process. We find that by the 6th iteration, having added fewer than 75 new true positive matches, feature importances change relatively little and look very much like their final state; this is also near our peak AUC from Figure B.2. For researchers implementing this method, we recommend plotting feature importances as they conduct HITL iterations; when they stop changing quickly, you may cease iterations; this stopping rule is an alternative to waiting for the false positive rate to drop below 1% as in either case, further HITL iterations are unlikely to improve performance.

Iteration	Cosine	Jaccard	Levenshtein	LCSSTR	Overlap	True Positives	True Negatives
1	0.26	0.23	0.30	0.07	0.15	19	423
2	0.41	0.13	0.22	0.10	0.14	19	423
3	0.27	0.26	0.20	0.16	0.11	15	295
4	0.32	0.32	0.19	0.07	0.10	11	199
5	0.28	0.24	0.30	0.09	0.09	19	423
6	0.30	0.36	0.18	0.09	0.07	12	220
7	0.31	0.26	0.23	0.09	0.11	30	940
8	0.33	0.21	0.31	0.06	0.09	16	324
9	0.41	0.20	0.13	0.12	0.15	17	355
10	0.32	0.25	0.24	0.09	0.11	20	460

Table B.5: As the number of HITL iterations increases, feature importances vary substantially but generally approach their final state.

C Relationship to the Record Linkage Problem

This section discusses the record linkage problem, its relationship to the fuzzy string matching problem, and how the distinction is often ambiguous.

In a typical instance of a record linkage problem, a researcher may wish to merge two data sets I and J of individuals using fields such as age, name, and sex. The most advanced tools for record linkage assign for any pair of possible matches i, j a probability that those two entries refer to the same entity. This probability $p(i, j)$ is a function of the distance between the age, name, and sex for the two entries: $\{|\text{age}_i - \text{age}_j|, |\text{name}_i - \text{name}_j|, |\text{sex}_i - \text{sex}_j|\}$. While the differences $|\text{age}_i - \text{age}_j|$ and $|\text{sex}_i - \text{sex}_j|$ are easy to calculate, the difference $|\text{name}_i - \text{name}_j|$ is typically computed using a deterministic string distance measure like Jaccard similarity.

TABLE 1. An Illustrative Example of Agreement Patterns.

	Name				Address	
	First	Middle	Last	Date of birth	House	Street
Data set $\mathcal{A}$						
1	James	V	Smith	12-12-1927	780	Devereux St.
2	Robert	NA	Martines	01-15-1942	60	16th St.
Data set $\mathcal{B}$						
1	Michael	F	Martinez	02-03-1956	4	16th St.
2	James	D	Smithson	12-12-1927	780	Dvereuux St.
Agreement patterns						
$\mathcal{A}.1 - \mathcal{B}.1$	Different	Different	Different	Different	Different	Different
$\mathcal{A}.1 - \mathcal{B}.2$	Identical	Different	Similar	Identical	Identical	Similar
$\mathcal{A}.2 - \mathcal{B}.1$	Different	NA	Similar	Different	Different	Identical
$\mathcal{A}.2 - \mathcal{B}.2$	Different	NA	Different	Different	Different	Different

The top panel of the table shows two artificial data sets, $\mathcal{A}$ and $\mathcal{B}$, each of which has two records. The bottom panel shows the agreement patterns for all possible pairs of these records. For example, the second line of the agreement patterns compares the first record of the data set $\mathcal{A}$ with the second record of the data set $\mathcal{B}$. These two records have an identical information for first name, date of birth, and house number; similar information for last name and street name; and different information for middle name. A comparison involving at least one missing value is indicated by NA.

Figure C.1: An example record linkage problem from Enamorado, Fifield, and Imai (2019).

We argue that AFSM can easily plug into any standard record linkage procedure where one or more of the record linking variables are strings. By using an AFSM-generated string distance measure to calculate the distance between a potential pair of records, the record linkage method can generate more informative match probabilities, and researchers may obtain more precise record linkages. This may involve conducting some HITL iterations, or it may only require the base model: both are likely strong improvements over current practice of using a single, arbitrarily chosen distance measure.

For record linkage problems with multiple string columns like for example first name and last name, there may be multiple ways to implement AFSM. The first is to perform AFSM on both sets of strings independently. As in Figure C.1, we would observe a high match probability between "Martinez" and "Martines" but a very low one between "Robert" and "Michael". A second approach is to combine the related strings into a single "Full Name" variable consisting of "Robert Martinez" versus "Michael Martines", and to calculate the AFSM-measured distance between these two strings. There are tradeoffs between these two approaches: while the first approach produces more fine-grained match probabilities, the latter approach is much more computationally efficient. We leave this decision up to the researcher based on their tolerance for false-positive matches and their time constraints.

A final point is that many record linkage problems treat variables as numeric or categorical when they may more fruitfully be analyzed as strings, and would be more suitable for AFSM than for current tools. Consider, for example, two dates of birth: "1981-01-01" and "1918-01-01". Current record linkage procedures would identify the distance between those two entries as 63 years, an enormous difference, but this discrepancy may be due to a simple swapping of two digits, a common error in data entry. Here researchers must leverage their expert understanding of the data generating process: if these dates are from old records, or hand-entered, or hand-transcribed, treating dates of birth as strings and applying AFSM may be a large improvement. If they are newly entered, or computed from other data, or entered from a precise date-selection web form, treating them as numbers may be more precise.

References

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

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

Box-Steffensmeier, Janet M, Dino P Christenson and Matthew P Hitt. 2013. "Quality over Quantity: Amici Influence and Judicial Decision Making." *American Political Science Review* 107(03):446–460.

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

Gentzkow, Matthew and Jesse M Shapiro. 2010. "What Drives Media Slant? Evidence from US Daily Newspapers." *Econometrica* 78(1):35–71.

Kaufman, Aaron Russell, Peter Kraft and Maya Sen. 2019. "Improving Supreme Court Forecasting Using Boosted Decision Trees." *Political Analysis* 27(3):381–387.

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