Name Vectorization in a Record Linking Context

Joseph P. Price

Brigham Young University

Abstract

We implement name vectorization in the context of record linking. We create graphs of names by summarizing links between census records created by FamilySearch.org. We use the node2vec as implemented in the Stanford SNAP library to get name vectors from the graphs. Name vector distance is comparable to string distance and is much faster to compute.

1. Intro

Perhaps the biggest limitation on record linking is the fact the some of the most important data is contained in fields as strings, like names. We know that the more similar two fields are the more likely the records are to match, but it is unclear which spelling differences are important and which aren't. Moreover, abbreviations, nicknames, and psudonames may be spelled very differently while referring to the same thing. We provide a solution by creating name vectors that are useful in the record linking context.

These name vectors allow us to make faster computations, which ultimately will lead to being able to make more comparisons overall. In addition, this will allow us to use names in a nearest neighbor framework to optimally identify potential matches.

We first create a graph that summarizes how names are mistaken for each other. We then identify an edge weight to use in the name vectorization process that emphasizes the local context of names. Name vectors are extracted from the graph using the node2vec algorithm. Finally, we validate the name vectors in various ways: we find that the Euclidean distance between two name vectors has similar information to the string distance between the same names. Name vector coverage of the population is fairly comprehensive at 98% for first names and 90% for last names. Moreover, using a fill-means method for unvectorized names (e.g. each unvectorized name is assigned the vector [0,0,0,…,0]) yields fairly good results. Finally we note that name vectors form expected clusters around starting letters when visualized in two dimensions and name vector distance is much faster to compute that string distance.

2. Background

3. Data

We use the 5 decennial U.S. censuses from 1900 to 1940. That amounts to 530.1 million entries or an estimated 188.6 million unique individuals. Thanks to the genealogical efforts of FamilySearch.org we also have 218.1 million links between these records (see table). Most of these links were computer generated, but they were high information links utilizing information about the entire family in the census. Thus, they are high quality and serve as good training data. This also means that the links necessarily underrepresent portions of the population as is well documented in the literature. We discuss later the possible impacts of this.

We create a name network by exploiting the fact that not all links in this database match up exactly. For example, someone appears as Adelbert W Stirling in the 1910 census, but as Delbert Stirling in the 1920 census. Thus we conclude that Delbert and Adelbert must be connected names. We summarize these differences to create the name network. Each node in the network represents a name (e.g. “Adelbert”). An edge connects two names that were mistaken for one another and weighted by the number of times that it occured (e.g. “Adelbert” – “Delbert” x1). See Figure for a visualization of the network.

Applying this method to surnames gave us 10 million links between 10 million names. Most of these names were misspellings, appearing only once in our data (See figure). Because of this, our algorithms were unable to produce useful name vectors—the algorithm was unable to optimize across such a low information graph. Thus we decided to focus our vectorization methods on ‘real’ names, or at least on names that were well

connected in our dataset. It is important to note that this means that name vectors cannot tell us the relationship between a name and an arbitrary misspelling.

We had two criterion for name selection of last names. First, we only considered last names that appeared in all 5 censuses. Number of names decreases with census count except at the 5th census, suggesting that there is a group of correctly spelled names that appear consistently in our data. (See table) Secondly, we restricted our focus to names that were connected to each other in the graph. This left us with 421,493 last names to vectorize. Note that this is a siginificantly larger number than is estimated to be the actual number of last names in the U.S. We expect this data to still include misspellings; however we have enough data on the names in question to assign vector values that preserve the relationships between names.

First names were slightly trickier because of middle names. First and middle names can often swap places, middle names can be shortened to a middle initial or be missed altogether. Thus naively matching first name to first name and middle name to middle name isn't sufficient. Instead, we computed a probability weight between all possible name combinations based on string similarity to infer which name matches were correct. This was computed as follows:

$$score = matching\_bigrams(name1, name2) - length(name1)*length(name2)$$

matching_bigrams(a,b) is calculated by counting the number of two letter pairs they have in common (So matching_bigrams('jake','ake') = 2). In addition, we counted matching first letters as an additional point (matching_bigrams('jon','john') = 2). We then subtracted the expected bigrams of two random names with the same lengths. This was mostly done to give higher preference to matching to an initial. Weights below zero

were returned to zero as a negative weight was not informative. We then created the weights by:

weight = score / (score + score_with_other_name)

Some name combinations had no matching bigrams at all. (Like “James” to “Howard K”) In this case, *weight* could not be calculated due to dividing by zero. Hence, *weight* was assigned to be 0.5. Now all name combinations with a weight of zero were that way because the other name combination was strictly more likely. Hence, they were dropped.

Finally all first name links were then summarized into a name network that had 1.3 million names and 9.7 million links. Again, we restricted ourselves to connected names. Due to computation requirements we removed below average links, resulting in 771,251 first names to vectorize with 3 million links. Note that we did not select first names on the number of censuses they appeared on. This was because there were much fewer first names to start with.

4. Method

Creating a scoring method that accurately reflects the likelihood of two names matching is one of the major innovations of this paper. It would be tempting to let the edge weight be simply the frequency those two names were confused for one another (i.e. the original edge weight), but this measure is skewed. Some names are ridiculously more likely to appear than other names (see figure) so that, inevitably, the links to those names dominate most other links. This isn’t what we want: we need the algorithm to learn from the context of its surroundings what name it is, but allowing common names to dominate the graph will force the context to largely be these ultra-common names. Thus it will

learn name attributes in relation to these common names instead of the names that are actually similar.

To correct for this problem we weighted the edges by the commonality of both nodes. Node commonality is simply the number of times that name appeared in our data. We also converted everything into a log scale because most name comparisons are different orders of magnitude. Finally, unlike first names, we had yet to make any string comparisons of the last names themselves. Thus for last names we upweighted the edge weight by the JaroWinkler similarity score. This is simply to give us a second filter to assure link quality. Finally, we added by 1 as appropriate so as to never take the log of 0, divide by 0 or multiply by 0. To summarize, edge weights were given by:

*last name edge weight = ln(link number of occurrences + 1) / (ln(name1 number of occurances + 1) + ln(name2 number of occurances+ 1) + 1) * (Jaro-Winkler(name1, name2) + 1)*

*first name edge weight = ln(link number of occurrences + 1) / (ln(name1 number of occurances + 1) + ln(name2 number of occurances + 1) + 1) * summarized probability weight*

These scores were manually verified to give the highest scores to similar names. However, we recognize that this scoring method is ad hoc and so it would be a useful application to endogenize the scoring process.

Finally, we fed the now properly weighted graphs to a node vectorising algorithm. Though we considered Laplace and GCN we decided to implement Stanford's node2vec algorithm from their SNAP library. This is because their framework was theoretically understandable and it gave us good results.

The node2vec algorithm works as follows: it creates r*N random walks, (N=number of nodes) such that there are r random walks starting from each node. Each random walk is length, l, and the next step is determined by weighting all possible adjacent nodes by their edgeweight and by Return and Inout parameters. These walks are then treated as sentences with each node being a word in a word2vec algorithm. With a given contex size the model has to guess the name given the surrounding names in the sequence, using a neural network. The model then learns the vectors associated with each name that allow it to best guess what the given name is. The output, of course, is these fully trained vectors. Word2vec has shown that the word vectors are useful in other machine learning contexts and we show that our name vectors are useful in the record linking problem.

5. Results

We visualize the name vectors in figures ,,. Figure shows all vectorized names with the vectors compressed into two dimensions by PCA and TSNE. PCA takes the first n principal components of a vector, and we used it to reduce the dimensions from 200 to 25. TSNE is a much more complicated process that retains clusters while not preserving distance. TSNE compressed the final 25 dimensions into 2 to plot. We then labeled the data according to first letter or sex as indicated. It's important to note that last names naturally clustered according to the letter they start with, though some letters are better separated than others. Name clusters according to sex definitely exist as well, though the sexes blur into each other on that front. These visualizations show that characteristics we would expect a name vector to preserve, such as first letter and sex, are indeed preserved.

We compare the efficacy of name vector distance compared to string distance. Name vector distance is the Euclidean distance between the associated name vectors. For string distance we use the JaroWinkler. First we test the relationship between the two in a set of potential matches between the 1910 and 1920 censuses in Allen County, Ohio. We easily reject the null hypothesis that the two are unrelated (t=620) and the single variable regression between the two has an R^2 of 0.41. See figure for the scatterplot.

We next compare the usefulness of the two in a machine learning context. Again using potential matches from Allen County, Ohio we analyze how good each model is at detecting if a match is correct or not and we vary which parameters each model has access to. This was implemented using a Random Forest, though we anticipate that our results are robust to any robust machine learning model. See table for a summary of our results. In the case where we did not give the algorithm string distance or vector distance measures the model performed poorly. When string distance or vector distance measures were included each model performed well, and also about the same as the other. We didn't see significant gains from including both.

These results suggest that name vector distances serve as an adequate substitute to string distance measures. In addition, computing name vector distances takes significantly less computation time (see table). This would allow us to increase the efficiency of computation time and allow us to use wider binning parameters in the linking process. Additionally name vectors allow us to use a nearest neighbor algorithm to determine potential links instead of using coarse binning rules such as the traditional first letter.

One reasonable concern with this method is that we are not vectorising most names, and hence we have the potential to overly miss records with oddly spelled names. Indeed it is true that only 4% of all last names are vectorized. However, those names are the most common ones, so roughly 90% of people have a vectorized last name in the census. The case for first names is even better with 98% of people having their first name vectorized. Thus if we use a system of vectorized first and last names 89% of the population will be covered.

There are two reasonable ways to deal with unvectorized names that we considered. The first method is to not consider unvectorized names at all. This comes at the cost of 11% of the population, but we don't have to make any assumptions about the name. The second method is to fill the missing values with the mean, which in this case assigns the zero vector [0,0,0,…,0] to all unvectorized names (since all components have expectation zero). Table summarizes these results for potential matches in Belmont County, Ohio. The test was originally done on data in Allen County, Ohio, but we found that the county had unusually high coverage for last names, making it unsuitable for this test. Belmont county was of similar size, but coverage was much closer to the national average for those censuses. We found that the fill-mean method marginally outperforms the drop missing method. This suggests that the use of [0,0,0,…,0] as the vector for unvectorized names is appropriate.

A second worry is that name coverage is racially correlated. This would inadvertantely produce racial bias in any model that used name vectors. However, we found that name vectors are inadvertadently not racially biased. Table summarizes our results. It is well documented that the likelihood of finding a suitable link for the record

of a white person is much higher than finding a suitable link for a black person. This is mainly because of widely spread poor census taking techniques employed in African-American areas. Thus, by summarizing all the links between censuses we would expect black names to be underrepresented. And indeed we found it to be the case that for last names that were over 75% black, coverage was significantly lower, even when controlling for the frequency of the name in the data. However, despite the fact that 47% of surnames held by blacks fell into this category, only 3.8% of black people had a last name that was over 75% black. In fact, names that were marginally black were actually much more likely to be vectorized resulting in a random black person being more likely to have a vectorized last name than a random non-black person. Thus, we conclude that last name vectors

6. Conclusion

Name vectors pose significant advantages to traditional linking methods, being a much faster substitute to traditional string distance measures.

6. References

7. Tables:

Table 1. Number of Census-to-Census Links

A. Total Links Available

	1900	1910	1920	1930
1910	24.2			
1920	12.6	30.6		
1930	7.4	17.8	38.7	
1940	4.2	10.0	21.8	50.8
Total	218.1			

B. Links Used in Analysis

	1900	1910	1920	1930
1910				
1920				
1930				
1940				
Total	218.1			

Notes: Each cell provides the number of individuals that have been linked between the census year in the row and the census year in the column (measured in millions). Panel A includes the total number of links that we obtained from FamilySearch and Panel B includes the links that we use in our analysis which are restricted to… XXX.

Links are given in millions. These links not only represent user generated links from the website, but also links that FamilySearch.org computed. Their linking algorithm mostly centered on linking matching families; hence these links are high information and thus fairly high quality.

Table 2. Comparison of Matching Performance

	Precision	Recall
JaroWinkler Only	94.6	92.4
Name Vector Distance Only	95.1	91.5
Both	94.4	92.5
Neither	93.4	87.6

Notes: These estimates are based on a training set from Allen County, Ohio that included 17,493 true matches and 4,333,304 false matches. Precision and Recall are compared between otherwise identical models tasked with identifying true matches from among false matches. Each model was a Random Forest with 200 trees of max depth 15 that had 40 additional features including: birth year, first and last letter, soundex, geographic, and father, mother and self birthplace comparisons in addition to name frequencies.

Table 3. Comparison of Computation Time

	Name Vector Distance	JaroWinkler Distance
Preparation	141.9	63.32
Computation	18.8	316.51
Total	160.7	379.83

Time is in seconds. Note that there are more opportunities to optimize preparation time than computation time.

Table 4. Name Vector Coverage

	First Name	Last Name	First and Last Name
1900	98.7	91.5	90.4
1910	98.7	90.8	89.9
1920	98.6	88.6	87.6
1930	98.8	89.3	88.5
1940	98.9	89.4	88.6
Weighted Average	98.8	89.8	88.9

First name vectors enjoy widespread coverage of the population. Last name vectors also cover most of the population, though they miss a significant 10%. First and Last name coverage is comparable to Last name coverage alone.

Table 5. Comparing Fill-Means Method to Dropping Missing with Last Names

	Precision	Recall
Fill-Means	95.4	91.6
Drop Missing	95.1	90.8

The Fill-Means method was computed by letting any unvectorized name have the vector [0,0,0, ... ,0], as 0 is the expected value of each entry in the vector. The Drop Missing method was computed by simply removing any unvectorized names from the dataset. These results suggest that the methods perform similarly. This test was done on data from Belmont County, Ohio. There were 17,292 true matches and 5,891,305 false matches. Otherwise the same features provided in Table were provided and the JaroWinkler features were not included.

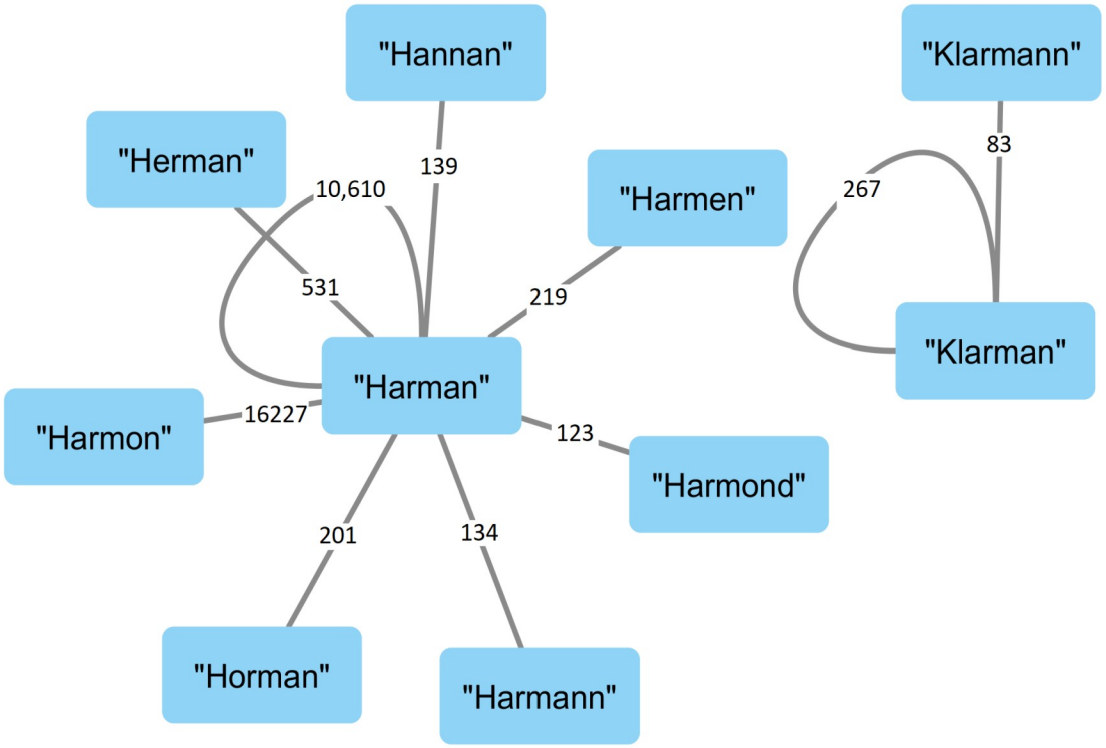
Table 6.

	(1)	(2)	(3)	(4)
Proportion Black	0.0644***	0.659***		
	(0.00109)	(0.000318)		
Name over 75% Black			-0.484**.*	-0.369***
			(0.000607)	(0.000591)
Controls				X
Observations	3,426,662	107,280,380	107,280,380	107,280,380
R-squared	0.001	0.038	0.006	0.062

Standard errors in parentheses
*** p<0.01, ** p<0.05, * p<0.1

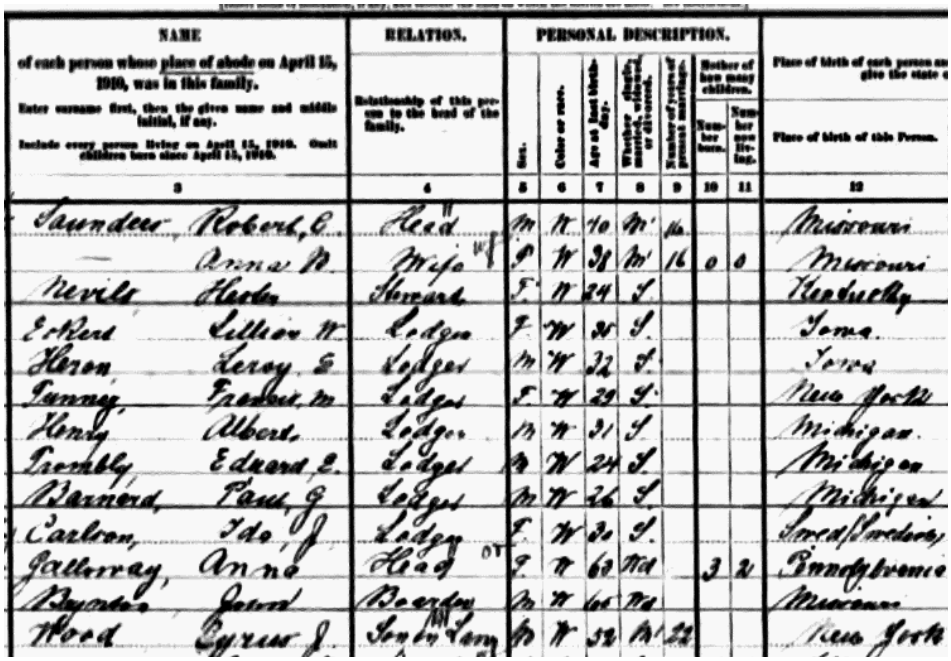
As the proportion of people that hold a surname are black increases, the probablilty that that name is vectorized also increases. This suggests that coverage for black people is in general good. However, the same result does not hold for the top of the distribution—names that are over 75% black are sigfinicantly less likely to be vectorized, even when we control for the name commonality. Names that are over 75% black account for 47% of all surnames held by African-Americans, and about 3.8% of all African-Americans held a name that was over 75% black.

Figure 1. Example of a Small Portion of the Name Network



Notes: This figure provides an example of the data that is provided to the name vectorization algorithm. The surname "Harman" is connected to 7 other surnames in our analysis sample and the number next to each link is the number of times that we see that pair of names in our linked record set. For example, there are 16,227 times where observe a pair of linked records where the individual's surname is "Harman" in one of the records and "Harmon" in the other.

Figure 2. Example image from the 1910 census



NAME of each person whose place of abode on April 15, 1910, was in this family. Enter surname first, then the given name and middle initial, if any. Include every person living on April 15, 1910. Omit children born since April 15, 1910.	RELATION. Relationship of this person to the head of the family.	PERSONAL DESCRIPTION. Sex.	Color or race.	Age at last birthday.	Whether single, married, widowed, or divorced.	Number of years of present marriage.	Mother of how many children. Number born.	Mother of how many children. Number now living.	Place of birth of each person and give the state or country. Place of birth of this Person.
3	4	5	6	7	8	9	10	11	12
Saunders, Robert, C.	Head	M	W	40	M	16			Missouri
— Anna B.	Wife	F	W	38	M	16	0	0	Missouri
Nevils, Hester	Servant	F	W	24	S				Kentucky
Eckert, Lillian M.	Lodger	F	W	25	S				Iowa
Heron, Leroy E.	Lodger	M	W	32	S				Iowa
Tenney, Frederick M.	Lodger	M	W	29	S				New York
Henry, Albert	Lodger	M	W	31	S				Michigan
Trombly, Edward E.	Lodger	M	W	24	S				Michigan
Barnard, Paul, J.	Lodger	M	W	26	S				Michigan
Carlson, Ida, J.	Lodger	F	W	30	S				Sweden/Sweden
Galloway, Anna	Head	F	W	60	Wd		3	2	Pennsylvania
Boynton, Jarvis	Boarder	M	W	66	Wd				Missouri
Wood, Cyrus J.	Son-in-Law	M	W	32	M	22			New York

Notes: This is an example image from the 1910 census that is a subsection that includes the name, relationship, gender, age, and birthplace fields. When family members share the same surname, the surname is only written once, followed by a dash on subsequent rows. Names are listed with the surname first followed by the given name.

Figure 3. Joint Distribution of the Distance Measures among True Matches

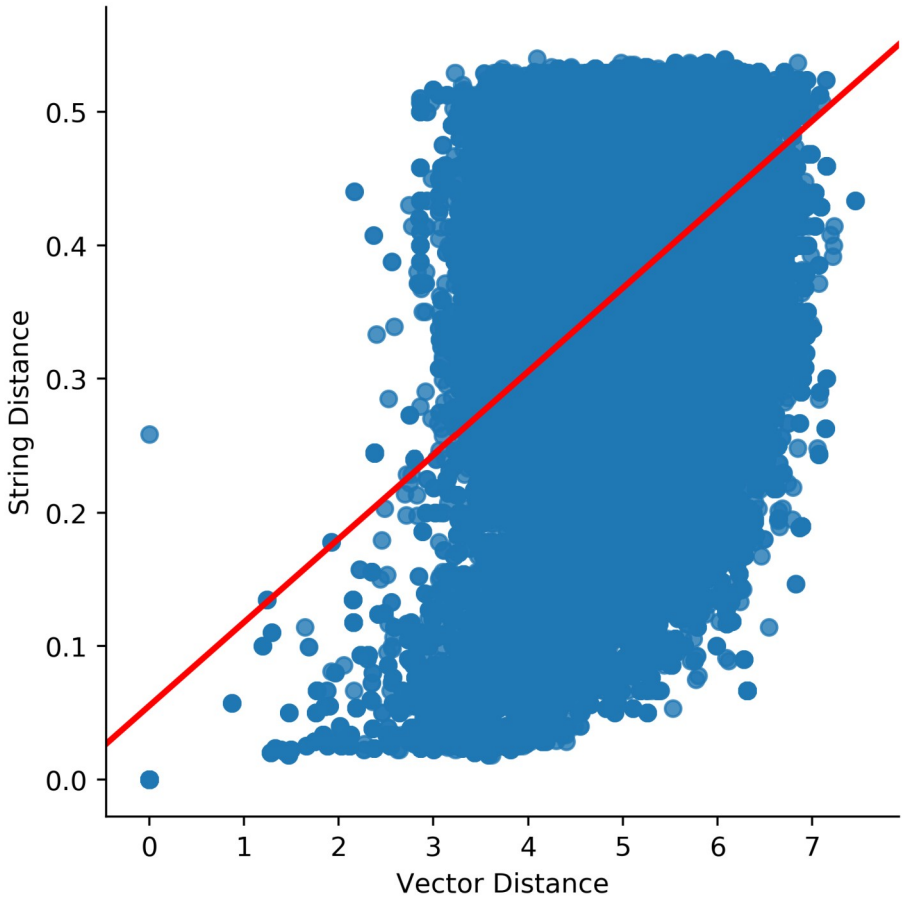
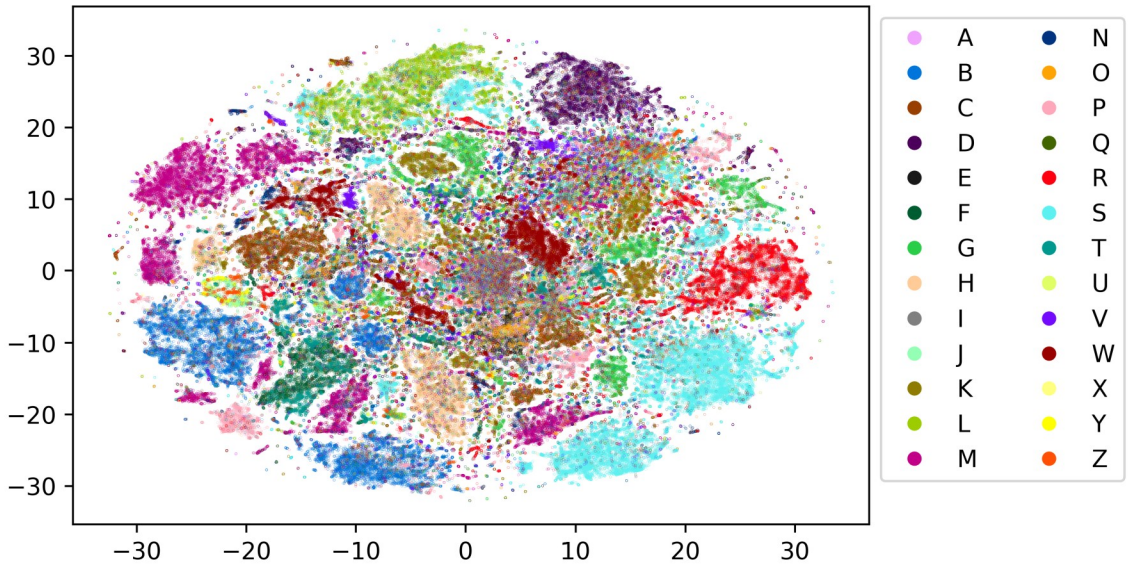
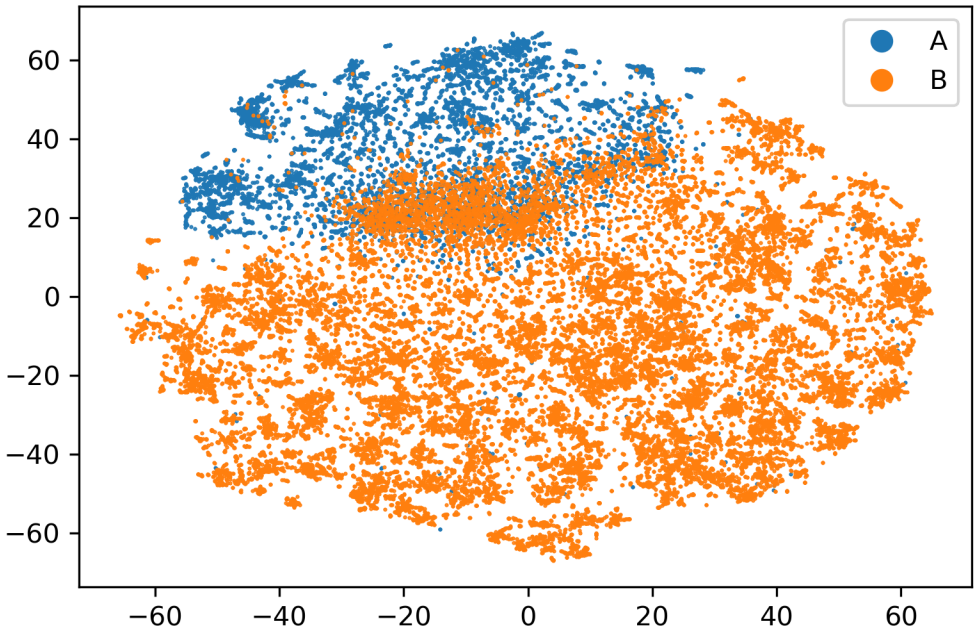


Figure 4. TSNE visualization of last name vectors, labeled by starting letter



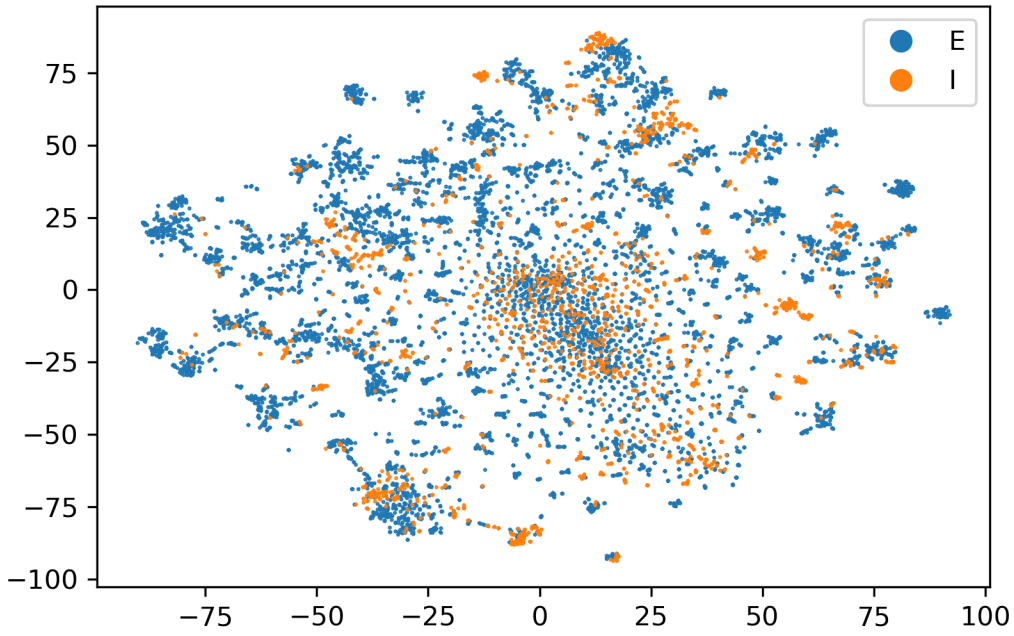
Each point represents a vectorized name. We took the first 25 principle components to reduce the initial 200 dimensions into 25. Then we used the TSNE algorithm from the Scikit-Learn library to further reduce the dimensions to 2. TSNE preserves clusters without preserving distance. Hence, it serves to test if clusters we would expect to see (such as clustering around most first letters) bears out in the data. We see several distinct clusters formed around first letters, though the quantity of letters being compared makes it difficult to discern relationships between individual letters.

2-D TSNE Visualization of Last Name Vectors, Names Starting with A and B



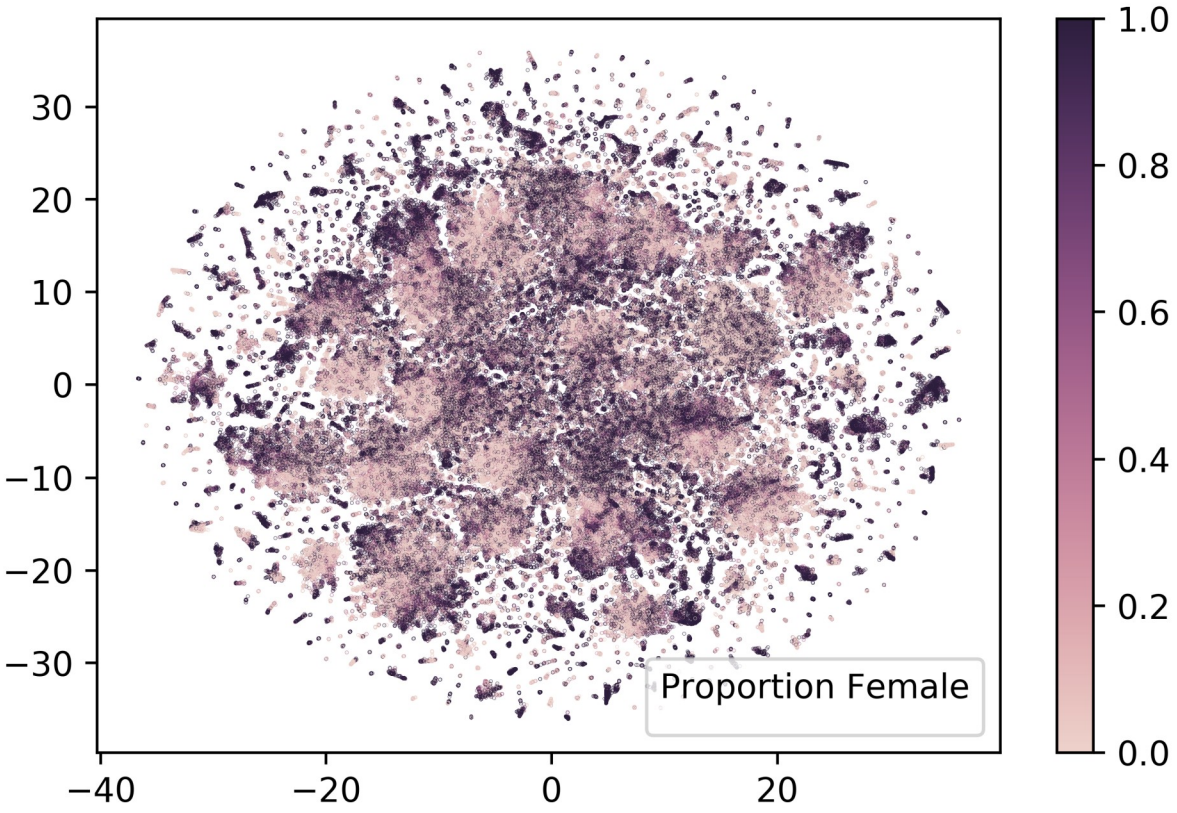
This graph was created in the same way as the one above, except only names starting with A or B were kept. This shows two distinct clusters formed by the two groups.

2-D TSNE Visualization of Last Name Vectors, Names Starting with E and I



This graph was created in the same way as the ones above, except only names starting with E or I were kept. Names that started with vowels were much more likely to be misspelled. Though distinct I and E clusters can be found, the separation between I and E names is much lest distinct.

2-D TSNE Visualization of First Name Vectors, labeled by Sex



Each point represents a vectorized first name. We took the first 25 principle components to reduce the initial 200 dimensions into 25. Then we used the TSNE algorithm from the Scikit-Learn library to further reduce the dimensions to 2. TSNE preserves clusters without preserving distance. Hence, it serves to test if clusters we would expect to see (such as clustering around sex) bear out in the data. This graph shows that there are clusters of distinctly female names and clusters of distinctly male names, though there is significant blending in areas of perhaps gender ambiguous names. Overall, clusters are less crisp than clusters by first letter.