Name Matching Outline

Michael Cahana, Yixin Sun

April 2019

Contents

1 Introduction

The basic motivation for this project is that we need to use company names in various other research projects relating to oil gas, and generally speaking company names don't come to us in great shape. They aren't normalized across states, and they often contain typos or slight alterations or both, such that two operator names that refer to the same entity are often non-identical. This document outlines the name matching process established in the name_matching repository. Note that from now on we will use "firm" to refer to either the operator or lessee we are performing cleaning and matching on.

2 Data

The following datasets are currently in use in our matching process:

- DI Landtrac Leases: containing a 1:1 mapping of lessee names to lessee addresses
- DI Flatfiles
  - NPH_OPER_ADDR: maps operator ids to operator addresses. Note that an operator usually has more than one unique operator address.
  - pden_desc: maps operator ids to operator names

3 Procedure Outline

3.1 First round

For each dataset $d \in \{leases, flatfiles\}$, in round $t = 1$:

1. Clean firm names
   (a) Normalize punctuation, spacing, and casing
   (b) Drop common words
2. Match unique firm names using three separate string comparison methods
   (a) Shared word - match one firm name to another if they share a word that isn't a common word
   (b) Cosine similarity - process firm names using Bag of words, apply a tf-idf (term frequency–inverse document frequency) weighting to each vector of words, and match firm word vectors using cosine similarity scores, with each potential match being declared a likely match if its score is or exceeds 0.4
   (c) Jaro distance - compute the Jaro-Winkler distance between every firm name and every other firm name, declaring likely matches to be those name/match pairs with J-W scores at or below 0.15
3. Clean firm addresses
   (a) Geocode standard addresses using Google Maps and the googleway R library. Always save coded addresses into a backup file such that addresses aren't re-geocoded with every iteration of name matching (geo-coding requires a Google API key and we'll be charged if we exceed 50,000 searches/month)

(b) Clean P.O boxes by normalizing typos/irregularities and processing all P.O. boxes in a standardized format that removes non-essential information

4. Match firm addresses, keeping only perfect matches
5. Pre-screen firm name matches using firm address matches
   (a) If two firms are declared to be a likely match via string comparison methods, and they also have a perfect address match, then they are marked as correct
6. Pre-screen firm name matches using human names
   (a) Using existing human names datasets, classify firm names as either humans or companies.
   (b) Within human name pairs, check if the last names match up
   (c) For pairs with matching last names, calculate the Jaro distance and cosine similarity of the first names, and whether or not the initials match up. If two human first names have Jaro distance and cosine similarity over .6 and initials do not match up, we can declare that these are not a match
7. Pre-screen firm name matches using a random forest model
   (a) Classify a random subset of 1,000 potential matches to be fed into a random forest algorithm. The algorithm takes in all the string distance metrics generated in prior steps, and generates 500 regression trees from which to average out a prediction. Since the random forest makes use of regression trees and the target variable here is binary, it outputs scores from 0 to 1, where scores closer to 1 imply a greater likelihood of correct match
   (b) Code all matches with random forest predictions $<= 0.2$ as incorrect, unless they've already been verified as correct by an address match (a rare edge case). This threshold is chosen due to the distribution of correct/incorrect matches across prediction scores in our sample data - 80% of all potential matches in the sample are at or below the 0.2 threshold, and of those matches, over 95% are incorrect
8. Review remaining firm name matches manually, reviewing only the matches we deem to be "important". What constitutes an "important" match is an open question, and our answer is going to inevitably be somewhat arbitrary. Our current suggestion is some combination of the following three requirements:
   (a) Pairs with a maximum $n >=$ the 90th percentile of the n distribution. For *leases*, for example, the 90th percentile is 87 leases
   (b) Pairs with a minimum $n >=$ the 70th percentile of the n distribution. For *leases*, for example, the 70th percentile is 10 leases
   (c) Pairs with a $\frac{minimum(n)}{maximum(n)} >= 0.10$
9. Combine together matches from all datasets $d \in \{leases, flatfiles\}$ and generate new firm groups using graph theory
10. Determine whether any firm group names match one another using string comparison methods and subsequent human review

11. Incorporate reviewed group name matches back into the original firm group graphs to ensure that duplicate clusters of firms (clusters referring to the same entity) are joined together as one

3.2 Later rounds

For each dataset $d \in \{leases, flatfiles, ...\}$, in round $t > 1$:

1. Clean firm names
2. Match unique firm names using three separate string comparison methods
3. Clean firm addresses
4. Match firm addresses, keeping only perfect matches
5. Pre-screen firm name matches using firm address matches
6. Pre-screen firm name matches using human names
7. Pre-screen firm name matches using a random forest model
8. Pre-screen firm name matches using firm groups from round $t - 1$
9. Review remaining firm name matches manually, reviewing only the matches we deem to be "important"
10. Combine together matches from all datasets $d \in \{leases, flatfiles\}$ and generate new firm groups using graph theory
11. Determine whether any firm group names match one another using string comparison methods and subsequent human review
12. Incorporate reviewed group name matches back into the original firm group graphs to ensure that duplicate clusters of firms (clusters referring to the same entity) are joined together as one

4 Firm Name Cleaning

DI has already done some of cleaning for lessee and operator names:

- landtrac leases: "alias grantee" is the standardized version of "grantee".
- pden_desc: "common oper name" is the standardized version of "reported oper name".

We use DI aliases as the firm name for subsequent cleaning steps, which basically entail normalizing punctuation, spacing, and casing, and dropping words deemed to be common (such as "PROD", "INC", "COMPANY", "INVESTMENTS", etc.).

5 Firm Name Matching

Given a cleaned set of firm names, we apply three separate string matching algorithms to determine likely matches between names. All three algorithms loop through each firm name, and compare it to all other firm names, declaring two names to be a match if they satisfy a certain criterion. The three algorithms are as follows:

5.0.1 Shared Word

Transform firm names into bags of words, and declare one name to match another if their bags share at least one word. For example, "JAMES L MARSHALL" would be matched to "MARSHALL RICHARD R" due to the shared "MARSHALL". This algorithm is our most conservative, and will likely catch a lot of false matches. But it is also likely to catch most true matches. When determined matches are outputted a *shared_words* score is included, which specifies the number of words shared between a name and its match.

5.0.2 Jaro Distance

Calculate the Jaro-Winkler distance from every name to every other name using the stringdist package. Declare a match pair to be two names with a distance less than or equal to a pre-determined threshold of 0.15. This algorithm is useful for detecting typos. For example, it would determine that "SANDDRIDGE ENERGY INC." and "SANDRIDGE EXPLORATION AND PRODUCTION, LLC" to be a match. When determined matches are outputted a *jw_distance* score is included, the score being simply the Jaro-Winkler distance between a name and its match.

5.0.3 Cosine Similarity

Transform firm names into bag of word vectors using the text2vec package, and use the tf-idf method to weight words by relevance, down-weighting common words that haven't already been removed by the cleaning step. Then compute the cosine similarity between every pair of vectors. Declare a match pair to be two vectors with a cosine similarity greater than or equal to a pre-determined threshold of 0.4. This algorithm captures the same types of matches as the shared word algorithm, except in a more elegant fashion. When determined matches are outputted a *cosine_similarity* score is included as well.

With three sets of potential matches (one set for each method) in hand, the sets are combined together and duplicate matches are removed such that a single list of potential matches, along with relevant scores, is outputted.

6 Address Cleaning & Matching

In parallel to the string cleaning/matching, we also match firms through addresses. We first clean the list of the unique address names for punctuation and spacing. We ensure that all words are only one space apart, remove commas and periods, make all addresses uppercase, normalize all variations of PO BOX, keep only the first five letters of a zip code, etc.

6.1 P.O. boxes

Addresses flagged as P.O. boxes then undergo some further normalization before matches are determined, such that each P.O. box is coded as: [number] [city] [state] [zip code]. Matches are only those P.O. box addresses that perfectly mirror one another.

Note that we generate a backup dataset of P.O. box normalizations such that we don't have to re-clean a P.O. box that has already been cleaned in a prior round.

6.2 Geocoding

We standardize non-P.O. addresses by exploiting Google's name cleaning algorithms. Google provides a geocoding service which cleans up addresses rather well, and we have a Google API key that affords us 50,000 free address searches/month. Geocoded addresses are said to be a match if and only if they are a perfect match.

Note that like P.O. boxes, we generate a backup dataset of geocoded addresses such that we don't have to re-geocode an address that has already been cleaned in a prior round.

7 Pre-screening

7.1 Verifying firm name matches using addresses

If two firms are declared to be a likely match via string comparison methods, and they also have a perfect address match, then they are marked as correct and don't require further human review. We do not declare firms with address matches but no name matches to be correct, since some firms may share an address that is a common business registry, or a common downtown high-rise containing hundreds of unrelated offices, such that an address match alone isn't a good tell of a true positive match.

Note that if round $t > 1$, and we find that matches that are now verified using addresses were coded in a prior round by a human as incorrect, then we output these matches to the file "generated_data/notifications/previous_non_pairs.csv" for the user to review if desired.

7.2 Verifying firm name matches using human names

Using datasets on common male first names, female first names, and surnames, we classify firm names as either being humans or companies (i.e. "CLEO THOMPSON" is a human, but "CHESAPEAKE" is not, and we know "CLEO THOMPSON" is a human because his name exists in our first name/surname data). We then limit ourselves to the subset of firm names deemed to be human names, and order those names such that first names are followed by surnames and preceded by initials (if any exist).

We then compare each human name to every other human name, checking if the humans' last names match up. For human pairs with matching last names, we calculate the Jaro-Winkler distance and cosine similarity between both humans' first names, and also determine whether or not the humans' initials match up. If two human first names have a Jaro-Winkler distance and cosine similarity (note that the cosine similarity metric we use here is flipped, so name pairs with scores closer to 1 are less similar) over 0.6, and initials that do not match up, we declare the pairs to *not* be a match. This conditional statement was chosen by evaluating a decision tree on a classified subset of potential human name matches, and

selecting a branch of that tree in which testing data was perfectly split into incorrect matches only. The branch we selected split data according to the three requirements specified above.

7.3 Verifying firm name matches using a random forest model

We've classified a random subset of 1,000 potential matches to be fed into a random forest algorithm. These 1,000 potential matches were classified by Michael Cahana. The algorithm takes in all the string distance metrics generated in prior steps, and generates 500 regression trees from which to average out a prediction. The algorithm makes use of 2-fold cross validation to mitigate over-fitting. Since the random forest makes use of regression trees and the target variable here is binary, it outputs scores from 0 to 1, with scores closer to 1 implying a greater likelihood of correct match.

We apply this random forest model onto our entire dataset of potential matches. We code all matches with random forest predictions <= 0.2 as incorrect, unless they've already been verified as correct by an address match (a rare edge case). This threshold is chosen due to the distribution of correct/incorrect matches across prediction scores in our sample data - 80% of all potential matches in the sample are at or below the 0.2 threshold, and of those matches, over 95% are incorrect. Note that this method is not perfect at classifying incorrect matches, and we apply it assuming we can live with some false negative classifications. If that assumption changes, and we decide that we cannot incorrectly classify a match as being incorrect, then we should abandon this pre-screening method.

After verifying many firm names to be incorrect using random forests, the amount of matches humans need to review is cut down significantly.

7.4 Verifying firm name matches using firm groups from round $t-1$

If round $t > 1$, then we utilize previously determined clusters of group matches to verify firm name matches that have already been marked as correct in prior rounds. Here we take clusters of group matches and expand them out to be complete subgraphs, such that a cluster with nodes {A,B,C} and edges A-B, B-C is expanded to also contain the edge A-C.

The reason we expand clusters to be complete is so we can not only verify edges that were explicitly coded as correct in prior rounds, but also edges that were implied to be correct in prior rounds since they belong to the same cluster, but were never explicitly reviewed by a human.

Note that if we find matches that are indeed implied to be correct via cluster completeness, but weren't explicitly verified in previous rounds, then we output these matches to the file "generated_data/notifications/inferred_matches.csv" for the user to review if desired.

8 Human Review

At this stage, most likely some matches in dataset d will have been verified by pre-screening, but not enough matches such that 95% of observations represented by name/match pairs are covered. Moreover, not all matches in which both sides of the match pair exceed 100 observations will have been reviewed (note that some of these matches will actually be below the 95% threshold because if both sides of the match appear above the threshold, below the threshold their counts (n.x, n.y) will be zeroed out to avoid double counting). This is where human review comes in.

Matches will be separated by dataset and saved to the "reviewed_data" folder, in ascending order of cumulative percentage coverage (note that observation counts are not double-counted, so occasionally an observation count in a match pair might be 0 since it is repeated). A human must open all of these match datasets and code each row that doesn't already have a keep score assigned (keep = 1 if match, 0 if not), until they reach the row that achieves 95% coverage. If Googling is required to reach a conclusion on the keep score, the reviewer should make note of that in the row as well.

9 Grouping (Graph Theory Magic)

We want to group together matches under a shared name. For example, if the match datasets in "reviewed_data" tell us that "CHESAPEAKE" is matched to "CHESAPEAKE MARCELLUS", and "CHESAPEAKE MARCELLUS" is matched to "CHESAPEAKE UTICA", we want to map all three names to one common name (presumably "CHESAPEAKE"). We do this using the R igraph library.

If we were to construct a graph with the nodes "CHESAPEAKE MARCELLUS", "CHESAPEAKE UTICA", and "CHESAPEAKE", and edges between "CHESAPEAKE"/"CHESAPEAKE MARCELLUS" and "CHESAPEAKE MARCELLUS"/"CHESAPEAKE UTICA", all three nodes are said to form a connected component (what igraph often refers to as a cluster).

This is essentially the graph theory magic we do, for all matches in "reviewed_data". Basically, we read in and combine all matches, then use igraph to form a massive graph of nodes and edges, with nodes being firm names and edges being name/match pairs. Our code then determines connected components and assigns every name within a connected component the name of the firm that is first in alphabetical order within that component.

The resulting dataset, titled "generated_data/grouped_matches/all_groups.csv" will have a column for a firm name, and a column for the firm's assigned group name.

10 Grouping Groups

10.1 Determining group name matches

Here's an edge case we'd like to account for: two distinct datasets (say leases and flatfiles) determine matches that form two distinct clusters, however both clusters refer to the same entity. Consider this example:

Within leases we find matches between "CARIZZO" and "CARIZZO (PERMIAN)", and within flatfiles we find matches between "CARZZO" and "CARIZZO (UTICA)". When we plug these matches into our graph theory process outlined in section 9, we'll have two clusters, one containing "CARIZZO" and "CARIZZO (PERMIAN)", and the other containing "CARZZO" and "CARIZZO (UTICA)". We won't find an edge connecting these two clusters because we don't consider name matches across datasets (doing so would blow up the number of string comparisons we need to make). Yet a human can easily tell that these two clusters belong together.

To account for this edge case, after we generate group matches we also use the same string comparison methods outlined in section 5 to generate a list of potential group name matches. In the example above, the resulting list would contain one row suggesting "CARIZZO" and "CARIZZO (UTICA)" to be a match.

A human must review this list (saved at "reviewed_data/group_name_matches.csv") to effectively determine which clusters belong together. Note that we elected this nested review process instead of considering name matches across datasets because this process requires less review of the human.

10.2 Incorporating group name matches

Once group name matches have been reviewed, they are applied onto our grouped data ("generated_data/grouped_matches/all_groups.csv") such that duplicate clusters of firms come to share the same group name. The resulting output is then saved to the file "generated_data/grouped_matches/grouped_groups.csv".

This is the dataset that the entire name matching process works towards. This dataset can be applied onto raw data (such as leases or flatfiles) to convert names that are effectively duplicates into names that are actually duplicates, and can be grouped as such. Note that this dataset will only contain group names for names that belong to groups of more than one member. That is, it doesn't contain all unique names in our raw datasets, it only contains unique names for firms that previously held multiple aliases.

11 Subsequent Iterations

If/when datasetets are updated or other datasets come to be included, we can re-run the procedure outlined above (for round $t > 1$). A makefile in the name_matching repo specifies the procedure order, so all that's left for a human to do in terms of code changes is mirror the processes outlined in the makefile for older datasets, such that newer datasets are also cleaned, matched, etc. This should be pretty painless, it just involves some slight re-writing to ensure data is being read from the right place, and that the proper columns containing names and addresses are extracted.

Note that our code makes sure to never overwrite any human-reviewed match data, rather only append new matches that need first-time review. So in every subsequent round, a human will never have to re-review matches that they reviewed in round $t - 1$.