Master Data Management

Xu Li
Department of Physics
New York University
New York City, NY 10012
xl1393@nyu.edu

Zihao Wang
Center for Data Science
New York University
New York City, NY 10012
zw1074@nyu.edu

Abstract

As health care systems evolve through mergers, acquisitions, and partnerships, there is a large problem identifying and recognizing duplicate and erroneous information on entities such as doctors, practices, and clinics when the data from various sources is combined. As the number and frequency of these mergers increases, there is a growing need to establish a single "source of truth," using data from key business domains. In this project, the key business domains are Name, Address and Taxonomies. In a short, this project aim to reconcile inconsistencies and eliminate redundancies of a given dataset with proficiency and accuracy.

1 Data Description

The data[4] is given in a `.csv` file with 478695 entries, each entry has 4 columns:

1. ID (Integer) - Index from 0 to n-1
2. Name (String) - Provider name
3. Address (String) - Provider address
4. Taxonomies (String) - Comma separated list of healthcare taxonomies

For instance, one entry looks like:

```
478646,"DR. GEORGE ALNA KRZYMOWSKI, M.D.","741 LAKEWOOD LN, MARQUETTE, MI
40451-9519, US",'Anatomic Pathology & Clinical Pathology'
```

Duplicate records are one such redundancy, and they tend to occur because of one or more of the following:

1. **Data Entry Errors**: Data entry errors causing duplicate records for the same person. For example, a certain physician, Josef Weininger, may have been recorded erroneously as Joseph Weininger.
2. **Absent Data Fields**: Records sourced from multiple systems can differ due to there being inconsistencies in the systems data fields. For example, "Dr. Josef Weininger" in one system may be registered as "Josef M. Weininger, MD" in another.
3. **Synonyms, Abbreviations, and Contractions**: Such issues arise when data models vary between different systems. Examples include Suite - Ste., DC - District of Columbia, NYC, New York City, and 5 digit zip codes vs. 9 digit zip codes.
4. **Equivalent Medical Terms**: Dr. Weininger may be called a "Skin Doctor" in one practice and a "Dermatologist" in the other practice. Both refer to the same specialization and are likely the same doctor.

Final project on DS-GA 1003 Machine Learning (NYU 2016), New York University, USA.

5. **Related Terms**: Dr. Weininger may be called a nephrologist (i.e., a kidney specialist) in one practice and a urologist (i.e., a urinary specialist) in another practice. This makes sense, given the proximity of the kidneys to urology. However, it is less likely that Dr. Weininger the nephrologist (i.e., kidney doctor) is the same as Dr.Weininger the neurologist (i.e., brain specialist).
6. **Name Changes**: Some providers may change their last names when they get married or divorced. When different data systems are merged, if the same name changes have not been made in both systems, then multiple records in the merged system may refer to the same provider.

Also there are several observations that can be made on the data:

1. In both Name and Address fields, the strings are often slightly permuted within the word, The usual case is that two letters are swapped, or one letter is moved somewhere else(**N1**).
2. There are lots of institutions/shops/hospitals/companies within the data. Basic model that doesnt if the record represents person, generates lots of false positives for the non-person record.
3. Some of the titles can have dots or dashes within them. It is assumed that those dots and dashes dont bring any value, so it is safe to ignore all of characters that are not alphanumeric. Also, in the Name field, anything that is not a letter will be ignored.
4. In Taxonomies, sometimes one taxonomy is just a combination of two others that sound similar(**N4**).
5. Sometimes middle name (or even first name) is shortened to a single letter.

2 Preprocessing

The main goal of this question is how to get as small training data as possible. Therefore, we first build up a similarity function that takes two entries as input and produce the similarity score (soft function) or a boolean variable (hard function). Then we use this to find some interesting pairs.

2.1 Parsing the Record

We deal with the record by using different method on different features:

- **Name**: Even though there's no formal format and data seems pretty messy, titles are always at the beginning or at the end. When they are at the end, they are almost always preceded by comma. Thus, words in Name are split into 3 categories: prefixes (personal titles), middle (real names) and suffixes (job/specialization titles). Set of possible prefixes is fixed to `{"DR","MR","MRS","MISS","MS","PROF"}` set, which is analyzed data by hand and it's clear that only those prefixes are used. Suffixes are just words preceded by the comma. Real names are everything that was left. Since sometimes last names have several parts interconnected with dashes, such last names are split into separate words. It is also assumed that the "Real name" contains first name and last name.
- **Address**: For majority records, the Address field has a following structure: Street `[, optional block/suite]`, City, State code + Zip code, Country code. (N1) can occur in every part of the address except for the state code and country code. It also happens with digits. In addition, sometimes there are some additional fields. Also, each field can contain additional commas which usually means that my parsing will partly invalid. When zip code contains less than 5 digits, one can assume that it's missing leading zeros, which will be added.
- **Taxonomies**: It's worth noting that (N1) doesn't occur here. On the other hand, it's quite common for the same taxonomy to repeat within the single record. Two information is extracted from the taxonomies: list of taxonomies and list of words used in taxonomies (because of N4).

2.2 Hashing

Because N1 happens so often and yet the set of letters within each word always stays the same, it does not really complicate too much. The net effect of this anomaly is that words shall be compared by the set of letters they contain. In order to compare set of characters, it's enough to sort the characters within the strings. Also, in many places, hashes are used instead of strings, to speed up the process of comparing. A simple 32-bit rolling hash is used, which is very quick to compute and not random. Hash collisions are actually not considered here, since this will have very little effect on the final outcome.

2.3 Finding Interesting Pair(IP)

One of the biggest challenge in this project is to find interesting pairs, meanly, two entries that likely to be redundant. There are 500K entries in one dataset, so there are roughly 10^{11} pairs. It is impossible to go over all pairs. So finding IPs accurately and efficiently without go over all possible pairs is critical.

Below is the graph to find the interesting pair (Figure 1). As the graph shows, we first parse the raw data into a list of dictionary (or features). Then we sort the list based on one of the features (like 'fullname' or 'state'). For each element of this sorted list, we check the below elements if they fit our similarity function. The benefit of this is that we do not need to check all pairs because once the pair does not meet the requirement (i.e. similarity function return 0 for not likely redundant.), we will stop checking for the below element will also not meet the requirement. Another benefit of this is that it can be run parallel easily. In this problem, we first use this method twice (one is sorting by 'state' and one is sorting by 'name'.). And then combine the result of this two methods.

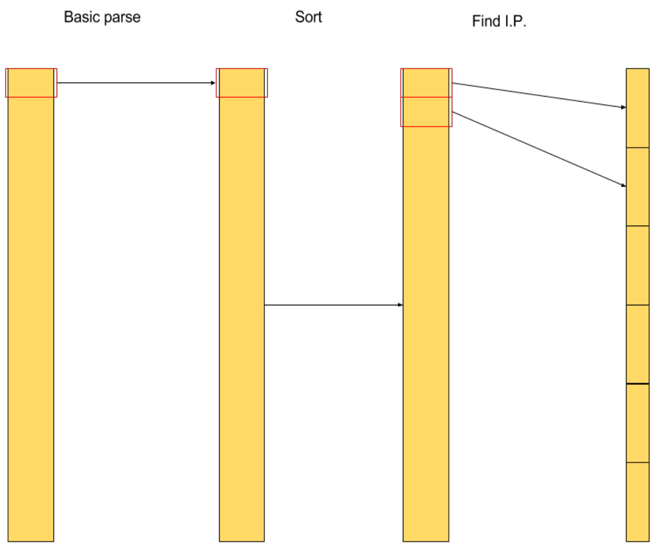

Figure 1: Graph to Find the Interesting Pair

2.4 Local Sensitive Hashing

There is another efficient in reducing the dimensionality called "Locality-sensitive hashing".[1, 2] The key of this method is using random hyperplane to split the feature space into different buckets. For example, we can randomly select lines by generating the weights of the lines w_i, then we hash the feature vector by

$$H(x) = \sum_{i=0}^{n} 2^i 1\{w_i^T x > 0\}$$

However, in this question, it is not well. The reason is that in the ground truth data, we may have some pairs that they have different name but the similar street. But in the LSH method, they will be hashed to different bucket. Also, because we do not know the distribution of the feature vector. Many features vector will be hashed to the same bucket.

3 Feature Extraction

Here are the features constructed from dataset. In ideal situation, only non-redundant features are used. In case several features provide the same information/clues, it's usually best to use the one that provides the most gain. Keep in mind that the intuition behind most of features is just a wild guess.

F0. 0: reserved for future use.

F1. a.country == b.country : same Country?

F2. a.name == b.name : same Name?

F3. a.street == b.street : same Street?

F4. hashString(a.firstName) == hashString(b.firstName) : same First Name?

F5. hashString(a.lastName) == hashString(b.lastName) : same Last Name?

F6. min(a.names.Size, b.names.Size) : minimum number of words in Name

F7. max(a.names.S, b.names.S) : maximum number of words in Name

F8. maxlen / namelen : proportion of characters in shared words

F9. maxsim / min(a.names.Size, b.names.Size) : proportion of shared words

F10. namesSimilarity(a.names, b.names, a.names.hash, b.hnames.hash)/min(a.names.Size, b.names.Size) : namesSimilarity computes something similar to LCS (`https://en.wikipedia.org/wiki/Longest_common_subsequence_problem`), the small difference is that it gives small bonus when only first letter is matching (due to N5)

F11. namesSimilarity(a.realnames,b.realnames,a.realnames.hash,b.realnames.hash)/max(a.realnames.Size, b.realnames.Size) : very similar to F08, but doesn't use titles (suffixes and prefixes).

F12. countCommon(a.suffixes, b.suffixes) / max(a.suffixes.S, b.suffixes.S) : proportion of shared suffixes

F13. countCommon(a.prefixes, b.prefixes) / max(a.prefixes.S, b.prefixes.S) : proportion of shared prefixes

F14. countCommonCharacters(a.firstName,b.firstName): countCommonCharacters counts the common characters in two strings divided by the length of the shorter string, the reason is that sometimes the same name can be written in a different way.

F15. countCommonCharacters(a.lastName, b.lastName) : similar to F14 but with the Last Name.

F16. calcLS(a.firstName, b.firstName) : calcLS returns LCS in two strings divided by the length of the shorter one, it performs a similar function to countCommonCharacters : F16 & F17, are similar to F14 & F15.

F17. calcLS(a.lastName, b.lastName)

F18. countCommonCharacters(a.skills, b.skills) : counts common characters in skills

F19. min(a.realnames.Size, b.realnames.Size) : similar to F04 & F05, but doesn't use prefixes & suffixes

F20. max(a.realnames.S, b.realnames.S)

F21. countCommonCharacters(cleanStringAlnum(a.address),cleanStringAlnum(b.address)) : common characters in Address

F22. countCommonCharacters(cleanStringAlpha(a.name), cleanStringAlpha(b.name)) : common characters in Name

F23. stateSimilarity(a.state, b.state) : stateSimilarity just counts the length of shared prefix in two strings : this essentially detects if two records are from the same state and if they're somewhat close together. (We assumed that if they share few first digits, then they are probably somewhere closer)

F24. countCommon(a.hskillsWords,b.hskillsWords)/max(1,min(a.hskillsWords.Size, b.hskillsWords.Size)) : amount of shared words in Taxonomies.

3.1 T-SNE

To test the performance of this feature extractor, we use T-SNE technique[5]. T-SNE is a technique that can covert the graph to a probability distribution over vertices

$$p_{ij} = \frac{\exp(e_{ij})}{\sum_{k \neq l} \exp(e_{kl})}$$

and strong connected nodes will have large probabilities p_{ij}. Note that the probability p_{ij} is in the high dimensional space. Then T-SNE try to use low dimensional space probability q_{ij} to get close to the p_{ij}. So that

$$q_{ij} = \frac{(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2)^{-1}}{\sum_j \sum_{l \neq k} (1 + \|\boldsymbol{y}_k - \boldsymbol{y}_l\|^2)^{-1}}$$

Finally, it trains the coefficient by using Kullback-Leibler divergence:

$$KL(P\|Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Here is the graph of the 2D embedding: We can see that by using this feature extractor, different

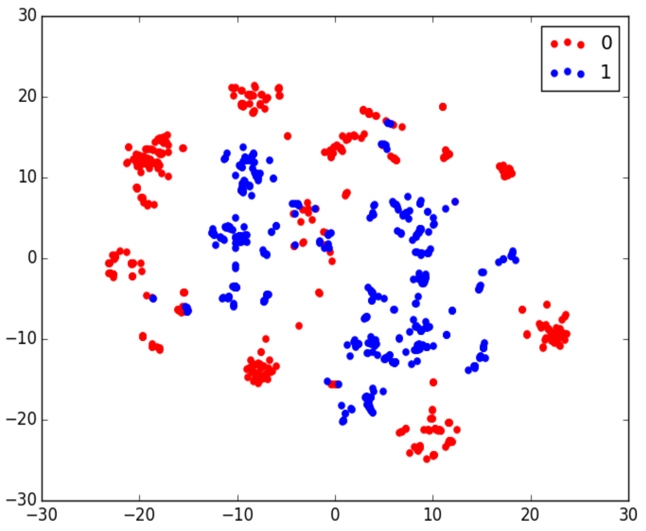

Figure 2: T-SNE of 2D embedding

class pairs can be clearly split which means that we can have great result on this dataset. This is also proved by using the different models.

4 Model Training

From the T-SNE graph, we can clearly see that, after the feature extraction process we can find that correct one and incorrect one has significant difference, which means that we can very great feature extractor.

In fact we have about 3700K pairs after the IP process, and we have 110K truth pairs in it (The whole amount of the ground truth data is 120K, but because we eliminate the data which is outside of US, it makes sense.) To make a balance between the two classes, we decide to randomly select the same amount of the 0 class as the 1 class. So finally we have about 250K data for training and validation. We are going to train the following model:[3]

4.1 Random Forest

Random forest is an ensemble method that combine different trees into a decision function. Because it uses different trees, it can effectively avoid overfitting. So the algorithm is that:

```
**Input**: $X, Y, B, m$
**Output**: $\hat{f}$
**for** $b = 1, \dots, B$ **do**
    Sample, with replacement, $n$ training examples from $X, Y$, call these $X_b, Y_b$;
    Sample $m$ features from the feature space (i.e. $x$ columns), call these $M_b$;
    Train a decision or regression tree $\hat{f}_b$ on $X_b$, $Y_b$ by using $M_b$;
**end**
$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b$;
```

Algorithm 1: Algorithm for random forest

In this algorithm we can see that it can be parallel because different tree can be done by each different thread thus it will be efficient for training and predicting.

4.2 Gradient Boosting

Gradient boosting is also an ensemble method that combine trees. But its construction is different from random forest. This model try to use train to get close to the gradient of the decision function $\hat{f}$. Thus its basic algorithm here is

```
**Input**: $X, Y, L(y, F(x)), M$
**Output**: $\hat{f}$
Initialize model with a constant value: $F_0(x) = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$;
**for** $m = 1, \dots, M$ **do**
    Compute so-called pseudo-residuals: $r_{im} = -\partial L(y_i, F(x_i))/\partial F(x_i)$ in $F(x) = F_{m-1}(x)$, for $u = 1, \dots, n$;
    Fit a base learner (here we use tree classifier with depth 3) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$;
    Compute multiplier $\gamma_m$ by solving the following one-dimensional optimization problem: $\gamma_m = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$.;
    Update the model: $F_m(x) = F_{m-1} + \gamma_m h_m(x)$.
**end**
Output $F_M(x)$;
```

Algorithm 2: Algorithm for gradient boosting

In this algorithm, we can clearly see that because it use weak learner. So it will be hard to get overfitting. But the disadvantage for this method is that we can not use parallel technique since it is recursive.

4.3 Logistic Regression

Logistic Regression is an efficient method and it is easy to interpret since it will predict the probability of specific class. The key of this is using logistic function in the last layer:

$$f(x) = \frac{1}{1 + e^{-x}}$$

So that it will map $(-\infty, \infty)$ to $(0, 1)$ which is exactly the interval that a probability is in. So the loss function of this uses R^2 metric. But because it will output probability, we can use MLE (Maximum Likelihood estimation) to get the coefficient.

5 Result

From the below graph (Figure 3), we can see that all the model have very great performance, but the tree-based model always performs well because we do not need to consider the scale of the feature.

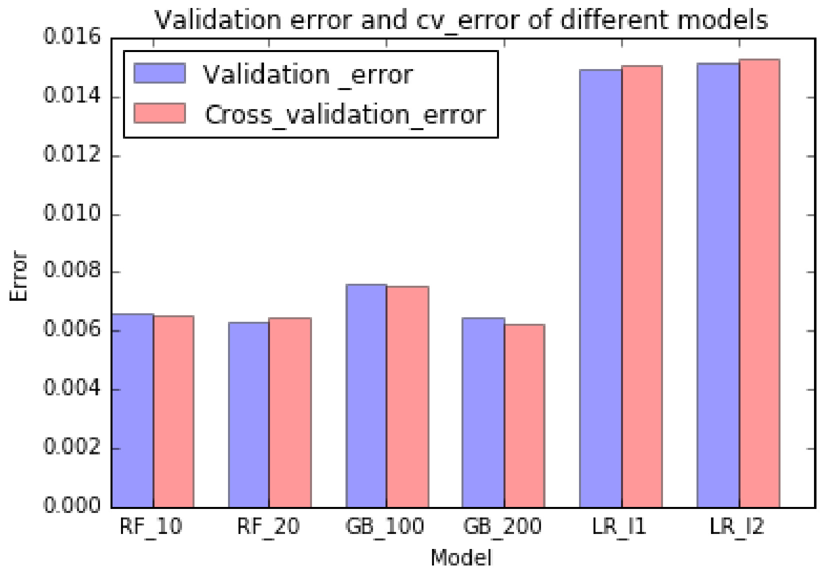

Figure 3: Error for different models

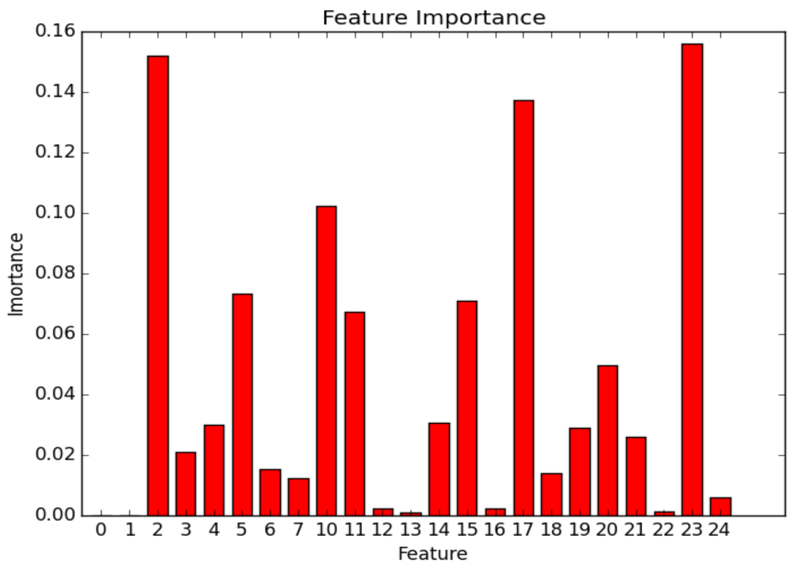

Figure 4: Feature Importance

The parameter here for random forest is the trees number we use (we will create the children node as much as possible). Small number of trees will easily get overfitting but huge number of trees will be more robust.

For gradient boosting, the parameter is the number of the iteration. Huge number of iteration will have get small loss in training set. We do not consider the overfitting for gradient boost if we use huge iteration number since the base function is the linear combination of weak learners.

For logistic regression, we use two way for penalty. The L^1 (also called Lasso regularization) aims to eliminate the number of coefficient while the L^2 (also called Tikhonov regularization) aims to decrease the value of coefficient.

The evaluation method we use is only the ratio of error (0-1 loss) and the cross validation of 0-1 loss. The reason we do not use AUC to evaluate our model is that we have the equal amount data of both 0-class and 1-class.

By using random forest, we also get the importance of different feature (Figure 4). From this figure, we can see that 'F2' and 'F23' have the highest importance here. So that for finding the interesting pairs, we can just need to sort by their name and their state.

Finally, we decide to use the random forest with the parameter $n_estimator = 20$ since it has the best performance.

6 Further improvement

We have gained success in the feature extraction, and we also have a very well model. However the finding interesting pair process is still not perfect for this project, because we generate too much useless pairs (Only 110k of 3700k is the truth). The further improvement of this is how to decrease the useless pairs while keeping the truth having the most of the ground truth.

References

[1] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[2] Diane J Hu, Rob Hall, and Josh Attenberg. Style in the long tail: Discovering unique interests with latent variable models in large scale social e-commerce. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1640–1649. ACM, 2014.

[3] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[4] Topcoder. Problem statement - masterdatamanagement. `https://community.topcoder.com/longcontest/?module=ViewProblemStatement&rd=16529&pm=13949`. Accessed April 4, 2010.

[5] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.