Capstone Project 2 Milestone Report

Current Challenges for Data Enrichment

There are two challenges for current Data Enrichment workflow: (1). How to develop a supervised ML algorithm that is computationally efficient and highly accurate, so third-party data records can be accurately linked with internal data. Without an accurate algorithm, the business value and enterprise analytics built on top of the enriched data is questionable, and therefore cannot provide useful guidance for decision makers. (2). How to design a robust and cost-effective architecture to keep the process running on a fixed schedule, so the business information provided by various data sources are up-to-date and valuable.

In terms of challenge 1, customer data, no matter how detailed, is fundamentally a snapshot in time. Business reputation, service quality and ownership can alter over time. Accurate detection of such information will provide valuable insights for supply chain management and customer services. Given the possibility of all these changes, data enrichment processes need to run on a continuous basis with a quantitative supervised ML algorithm. There are several cloud database services already providing some primitive data enrichment services, e.g. AWS FindMatches, MongoDB Single View. MongoDB Single View is based on query language to filter and match records. While AWS FindMatches has the ML capability to develop a supervised ML model after rounds of human data labeling. However due to the size of the different databases need to be matched for some large business entities (where a single database may contain records up to 10 million), the computational time for such algorithm can be lengthy, and the relatively more expensive AWS Findmatches service is not the most cost-effective solution for such computational intensive task. This paper describes an in-house highly accurate supervised ML model that can be effectively deployed on the more cost-effective virtual machine (AWS EC2 or docker). More specifically, a two stage ML model is used for the first time for third-party data enrichment. After software prototyping and ML algorithm accuracy benchmark, the software application is saved as artifact files in AWS S3 bucket. AWS Code Pipeline is initiated to trigger the ML algorithm with a fixed schedule to generate up-to-date business information at a minimal cost. The final output is QC tested and delivered with AWS glue crawler. The final output of the application is directly served with Redshift Spectrum for further downstream consumption.

For ML algorithm, a two-stage cascade model is adopted to improve the match percentage rate between datasets, while saving unnecessary computational efforts by narrowing down the potential match pairs. The first stage model targets on records with same Zipcode and address number (not the entirely address including both address street and number). This assumption greatly reduced the number of potential match pairs being compared, and almost 85% of match records were generated from this stage. This approach significantly reduces the memory and computation costs associated with other matching algorithm, e.g. fuzzy matching with O(n) running cost. The second stage process including all unmatched records left from the first stage. Comparisons for similar records are performed for these only with same Zipcode(but different address number) at this stage, since the assumption is that records may contain unexpected human errors on address or business might recently move to a new physical location. The number of pairs for comparison significantly increased at this stage. However, this stage is necessary for matching any records that may contain human errors, since it improved the match percentage by another 15%. The XGBoost model is used for both stages of pairing processes. The model parameters and weights are adjusted for the best Roc-Auc accuracy at each stage. Key features extracted from data for XGBoost classifier model are: similarities between Name, address, address number, telephone number and customer type classification. For string type features, Jarowrinkler distance is measured to assess similarities between these short phrases. For numerical features, gaussian distribution based algorithm is used to determine the similarities between these numbers.

The second challenge of capable of serving data in fast turnaround is addressed by fully utilizing the AWS Glue service to directly ingest the JSON documents produced from the ML algorithm as Parquet format, and then using dynamic schema interference of this service to iterate on the self-defined schema without the rigid definition process. This fast turnaround capability is vital for such data enrichment application. Because the downstream data analytics may iterate on the first schema frequently, depends on the analytic topic. Various third-party databases with different focus can be introduced for matching, and each comes with different key and data types. How to adapt the output schema in quick turnaround so that it can satisfy needs of different user case is always challenging. The data post processing and pipeline solution proposed in this can evolve the document-based output according to different business needs, and therefore significantly accelerate the data delivery speed for further analysis. Since this task is relatively less computationally intensive, more mature cloud service can be utilized to accelerate the data sharing process without incurring large cost. When different third-party data are used for data augmentation, the matching algorithm is paralleled to run on individual EC2 machine to accelerate computation. The in-house built algorithm goes through minimal variation for data ingestion from various data sources and can be easily scaled up for parallelization.

Description of the dataset

The dataset used for this project are from Kaggle website: One for Safeagraph, and one for Yelp. Some basic data cleansing is performed on both internal data and external data. First, lowercasing all text data, remove special characters from strings to reduce sparsity of embedded words, and strip empty spaces before and after strings. Extensive noise removal is also applied, including punctuation removal, domain specific keyword removal. Then NLP standardization methods are applied next, including stemming and Lemmatization. For this record comparison task, text information comes as short phrase, often in abbreviation format, so there is no need to remove stopping words. Text normalization is applied to address features, since many address data are noisy, and vary a lot based on peoples' spelling habits. A dictionary type mapping is used for cleansing non-standard spelling.

The unique difference between this task and other NLP taxonomy task is that the similarity scores will be calculated for both text information, and numerical features, such as zip-code, phone, address number. Without a comprehensive assessment with both text and numerical features, the accuracy of record matching will be sacrificed, due to limited information. The cleansing methods for numerical features are: special character removal, extract the first five digit of zip code for standardization, remove country code from phone number and reformat the phone number to normalize data. Remove domain specific information in front of address number to improve consistency.

Another practical observation is that there is very limited benefit to adopt text augmentation and embedding that are prevalent for deep learning NLP approach in this record matching task. This is because text information for records are noun types, and the string length associated with records is short.

After hand-crafted these features for each record, the similarities of different records can be calculated for numerical and string features respectively.

Data Cleansing

Steps of data cleansing of such NLP data are:

1. NLP data type specification: Choose string data type for string similarity calculation, numerical data type for numerical value similarity calculation.
2. Data normalization technique: including remove capital letters, remove unique characters, and remove country code for phone number. Strip the space before and after customer name, city, state and country strings.
3. For zip code, only compare the first 5 digits. Since the detailed last 4 digits associated with possible human errors, so the last 4 digits were removed.

Data Exploration:

The proposed software architecture is shown in the figure 1:

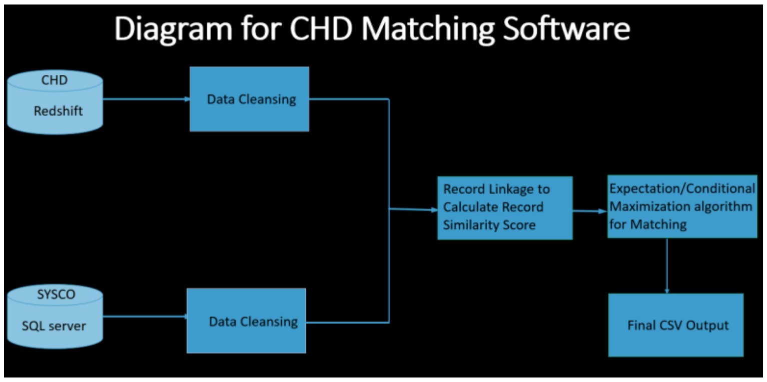


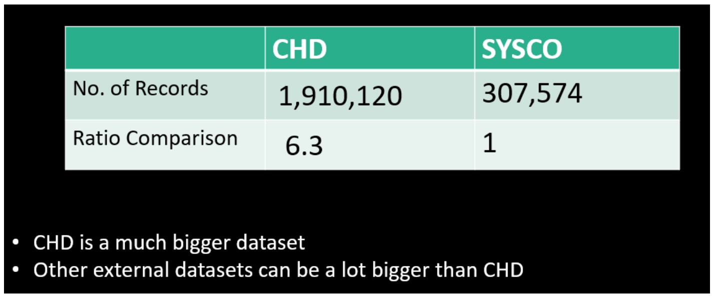
Diagram for CHD Matching Software
CHD
Redshift
Data Cleansing
SYSCO
SQL server
Data Cleansing
Record Linkage to Calculate Record Similarity Score
Expectation/Conditional Maximization algorithm for Matching
Final CSV Output

Figure 1: SVOC Software architecture

After ingested data from two relational databases, the data are not labeled with matched or unmatched. To develop a data science model with supervised ML algorithm and rigorous threshold, the first step is to use Unsupervised learning to generate labeled data. To accelerate this process, a unsupervised ML algorithm is used in this step. For this step, the Expectation/conditional maximization algorithm is used. **Expectation-Maximization algorithm** can be used for the latent variables (variables that are not directly observable and are actually inferred from the values of the other observed variables) too in order to predict their values with the condition that the general form of probability distribution governing those latent variables is known to us. This algorithm is actually at the base of many unsupervised clustering algorithms in the field of machine learning.

The procedures of the EM algorithm are:

1. Given a set of incomplete data, consider a set of starting parameters.
2. Expectation step (E – step): Using the observed available data of the dataset, estimate (guess) the values of the missing data.
3. Maximization step (M – step): Complete data generated after the expectation (E) step is used in order to update the parameters.
4. Repeat step 2 and step 3 until convergence.



	CHD	SYSCO
No. of Records	1,910,120	307,574
Ratio Comparison	6.3	1

- CHD is a much bigger dataset
- Other external datasets can be a lot bigger than CHD

Fig. 2: Initial Data size exploration

Data size comparison showed CHD data is a lot bigger than the internal dataset, and therefore it is important to parallel the data science model and use the memory efficiently.

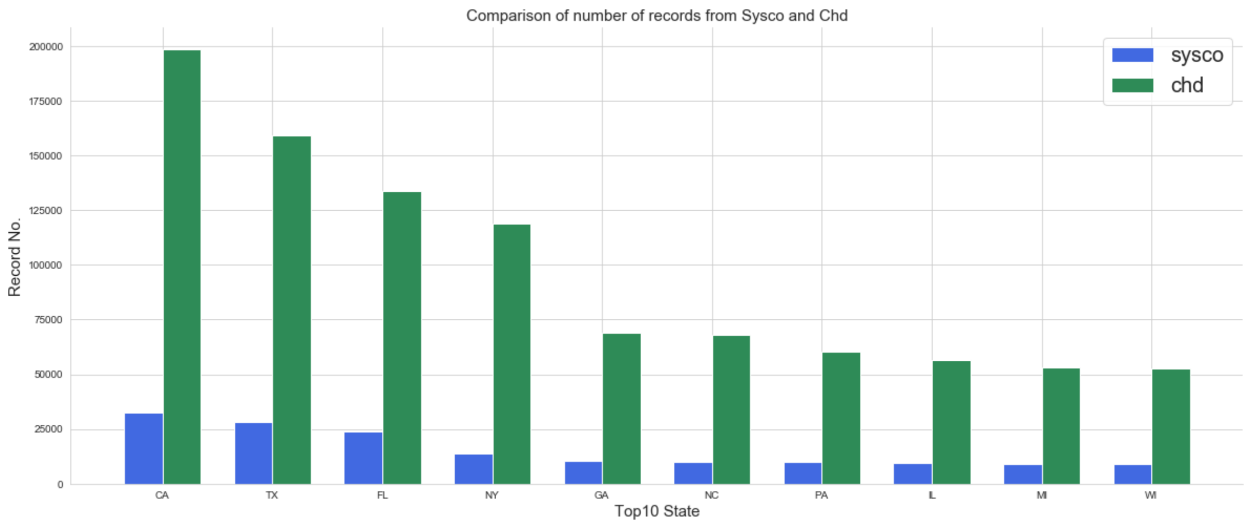


Fig. 3: Initial Data count.

Figure 3 showed that the data distribution in different states. It can be seen that the large cosmopolitan contains 5 times more data compared with less populated states.

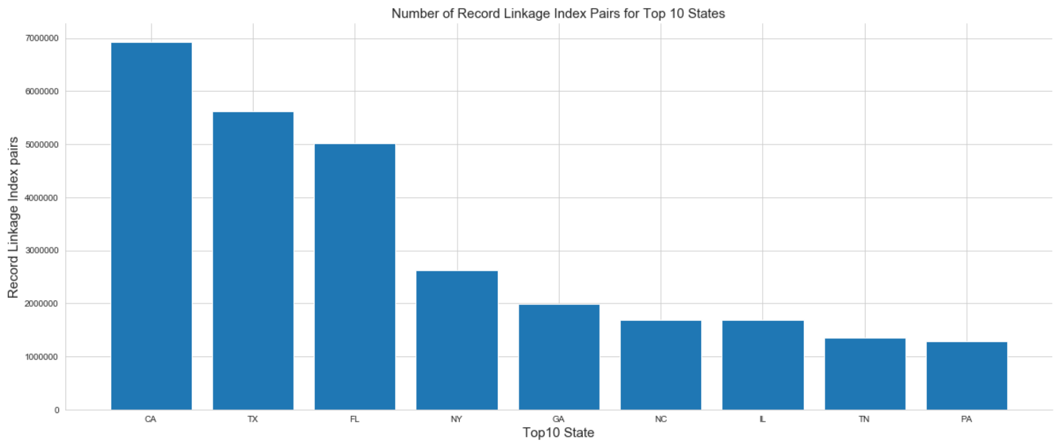


Figure 4: Total Number of Index Pairs generated from RecordLinkage for Calculation

Figure 4 shows total number of index pairs needs to be calculated from all 51 states: 53,541,044, and top 10 states are counted for 60% of the index pairs: 29,561,880. This amount of index pairs are generated by using blocking conditions: Only records with the same city and zip code will be paired for similarity comparison. Even with this blocking condition, there are large number of index pairs need to be calculated. This presents a significant challenge for computational efficiency.

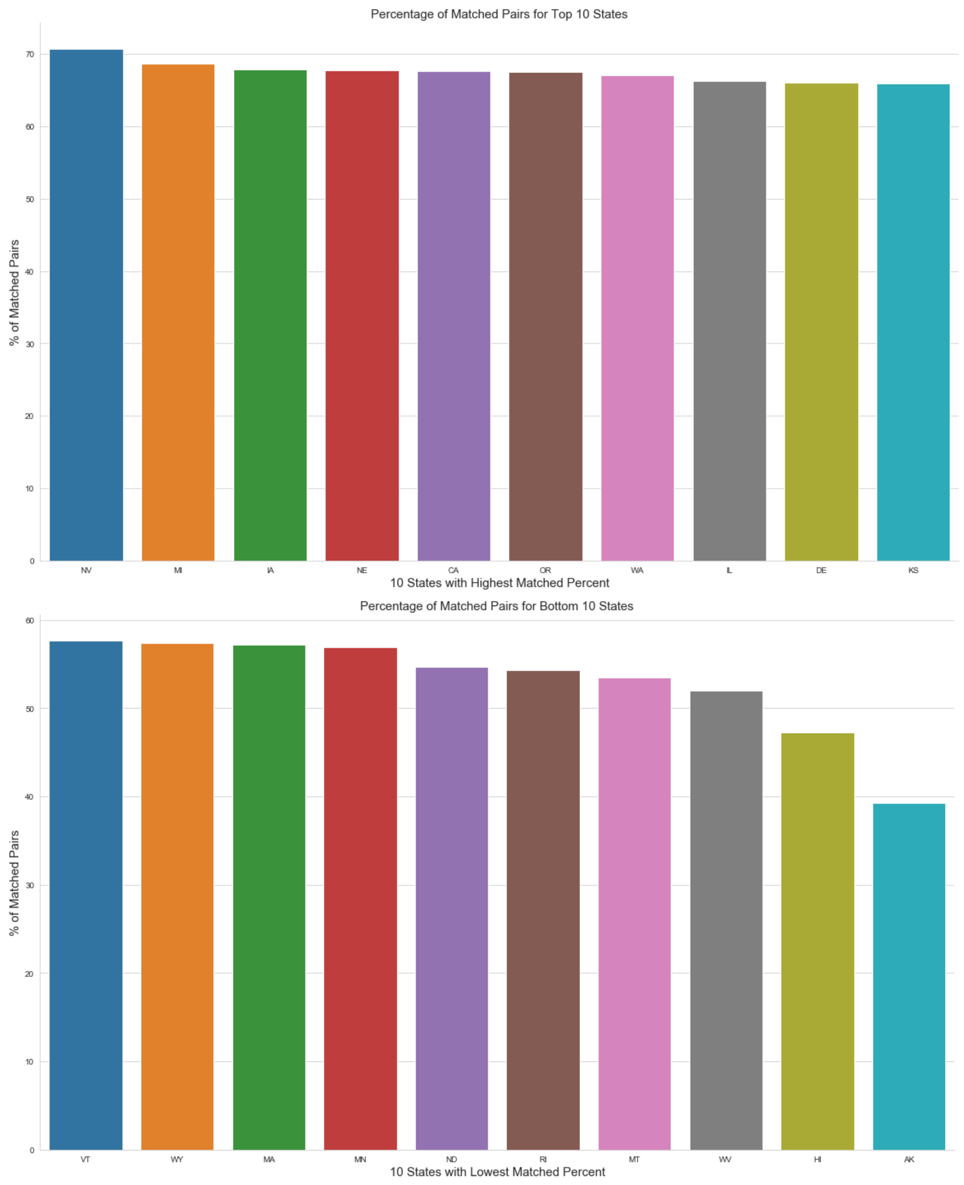


Figure 5: Matched percentage counts for top 10 states and bottom 10 states.

After applying the unsupervised algorithm, the number of matched percentage counts were compared. Filling rate from RecordLinkage is good, top ten states have match rates bigger than 65%. Bottom ten states have match rates bigger than 50%, except AK.

Summary:

Algorithm Analysis Performed by using python version of RecordLinkage.

The results by using unsupervised ML algorithm are impressive, and either library can be used for benchmark purpose due to the unsupervised learning nature.

Initial research was performed to implement the RecordLinkage on Spark platform: Strategic move for future external data consumption.

To improve the performance, the data need to be labeled as true match or false match, and apply supervised learning algorithm to make the algorithm more accurate.