Autonomous University of Barcelona

Master Thesis

Record Linkage in Historical Records of Barcelona

Bhaskar Gautam

Supervised by

Dr. Oriol Ramos Terrades

October 16, 2019

Abstract

Record Linkage is the task of finding records in a data set that refer to the same entity across different data sources. Traditional methods (e.g. attribute comparison-based methods, clustering based methods, and active learning methods) are usually trained in supervised manner by using labelled data as prior knowledge. Since it is not trivial to label data for training, researchers have recently turned to unsupervised methods, and have thus developed similarity-based methods, probabilistic methods, hierarchical graph model-based methods, knowledge graph embedding methods etc. Utilizing the graphical structure of the data can help improve the record linkage performance. In this report, we experiment with various record linkage methods on three different datasets Census dataset, FEBRL dataset and CORA dataset. We also propose two novel methods (RLTransE and VEER) for record linkage based on the knowledge graph embedding.

Keywords: record linkage, entity alignment, author disambiguation, knowledge graph embedding, machine learning, historical census data

Contents

Chapter 1

Introduction

Record Linkage is the task of finding records in a data set that refer to the same entity across different data sources. Often similar data is collected in different format and for different purposes. Combining records from different data sources can help create a larger understanding of the subject and cross-validate records. Other common terms used for the task includes deduplication (in database community), entity alignment (in knowledge graph community) and object identification (in artificial intelligence community). Deduplication aims at eliminating repeated data or multiple copies and thus compressing the database. Entity Alignment aims to predict a link or relation between two entities from the same or different knowledge graphs. Object identification aims at recognizing various patterns (e.g. face detection or car detection) from given training data.

1.1 Motivation

Historical census data capture valuable information of individuals and households in a region or a country. They play an important role in analysing the social, economic, and demographic aspects of a population [1]. Difficulties of historical census data linkage come from several aspects. These include poor data quality caused by the census data collection and digitisation process, and large amount of similar values in names, ages and addresses. More importantly, the condition of individuals in a household may change significantly

between two censuses. For example, people are born and die, get married, change occupation, or moved home. These problems are made more challenging in early historical census, i.e. those collected in the 19th or early 20th century, where only limited information about individuals were available. As a result, linking individuals is not reliable, and many false or duplicate matches are often generated. This is also a common problem in other record linkage applications, such as author disambiguation [2].

Early development in computing devices like Hollerith Tabulator [3] was a result of increasing size of census data in late 17th century and lack of resources to tabulate and analyse this information. Since then, many government research departments have extensively used their local census data to project population and plan for provision in local government. Now, various countries have made their historical census data available in the public domain. Universities and businesses have been able to uncover deeper trends and extend their existing databases. Recently, researchers are exploring ways to improve the survey methods and increase residents participation. We can expect more accurate and detailed censuses in near future.

United Nations have recommended that every country should conduct a national census every 10 years [4]. The 2020 World Population and Housing Census Program have been approved by United Nations in 2015, which aims to measure the progress of sustainable development.

1.2 Related Work

The task of record linkage has been explored by a number of communities, including statistics, databases, and artificial intelligence. Each community has formulated the problem differently, and different techniques have been proposed [5]. The database community have proposed various string edit-distance based methods for record matching like a general-purpose scheme in 1997 [6] and a knowledge-intensive approach in 2001 [7].

In statistics, a long line of research has been conducted in probabilistic record linkage, largely based on the seminal paper by Fellegi and Sunter in 1969. [8] formulates entity matching as a classification problem, where the basic goal is to classify entity pairs as matching or non-matching. Fellegi and Sunter propose using largely unsupervised methods for this task, based on a

feature-based representation of pairs which is manually designed and to some extent problem-specific. These proposals have been, by and large, adopted by subsequent researchers, often with elaborations of the underlying statistical model. Jaro-Winkler distance, a popular metric for string comparison, has been used for record linkage purposes in 1995 [9] and 1999 [10].

The AI community has focused on applying supervised learning to the record-linkage task for learning the parameters of string-edit distance metrics [11] and combining the results of different distance functions [12]. They learn models from labelled data to conduct record linkage. The major problem of supervised methods is that it is arduous and time-consuming to label data. Unsupervised methods include similarity-based ones [13], probabilistic ones [14], hierarchical graph model-based ones [5] and self-learning and embedding based entity alignment [15].

In terms of Knowledge graph, which allows to organize the knowledge in a semantic manner, record linkage is also known as entity alignment. Entity alignment aims to align entities between different entity groups or even different knowledge graph [15]. A few studies have employed knowledge graph embedding to conduct entity alignment, which do not need to define similarity measures or make assumptions in advance. Actually, as an important part of a knowledge graph, entities contain rich semantic information, which can be utilized to conduct or promote the performance of entity alignment. [16] proposed a method that jointly learns the embedding of multiple knowledge graphs in a uniform vector space to align entities.

In this report, we experiment performance of various record linkage algorithms on three different datasets. Next chapter details the datasets selected for experimentation. Chapter 3 gives theory of state-of-art methods currently available for record linkage based on Fellegi and Sunter Framework. Chapter 4 explains the various knowledge graph embedding methods and how they can be utilized for entity alignment or record linkage. Chapter 5 provides a few novel techniques to improve the performance of record linkage. The report is concluded in chapter 6 and future research possibilities are explored.

Chapter 2

Dataset

We selected following three labelled datasets to experiment and evaluate various record linkage techniques.

2.1 Historical Census Data:

We used the census records for *Sant Feliu* town of Barcelona collected in 16 different census from 1828 to 1940. Figure 2.1 depicts the population of the town during each census. We note that the population of the town almost doubled in two decades (1920 to 1940). The dataset contains 59084 records of individuals with 30 attributes. Available attributes include individual's full name, year of birth, civil status, occupation and relationship with head of the family. We attempt to link records pertaining to a single individual across census years.

Figure 2.2 represents the changes occurring in the family structure of *Abel Ponc*. When he is born in 1915, his parents live with his grandmother. Later in 1918, his younger sister is born and his father becomes the head of the family. By 1924, Abel's grandmother is probably dead and a stranger *Joan Roca* moves in with them. In 1936, Abel's aunt has also moved in with his family. In 1940 Abel marries *Maria* and moves out of the family. Here we note that the attributes related an individual changes with time and so does the family structure. Often there is error in names of individual as in case of

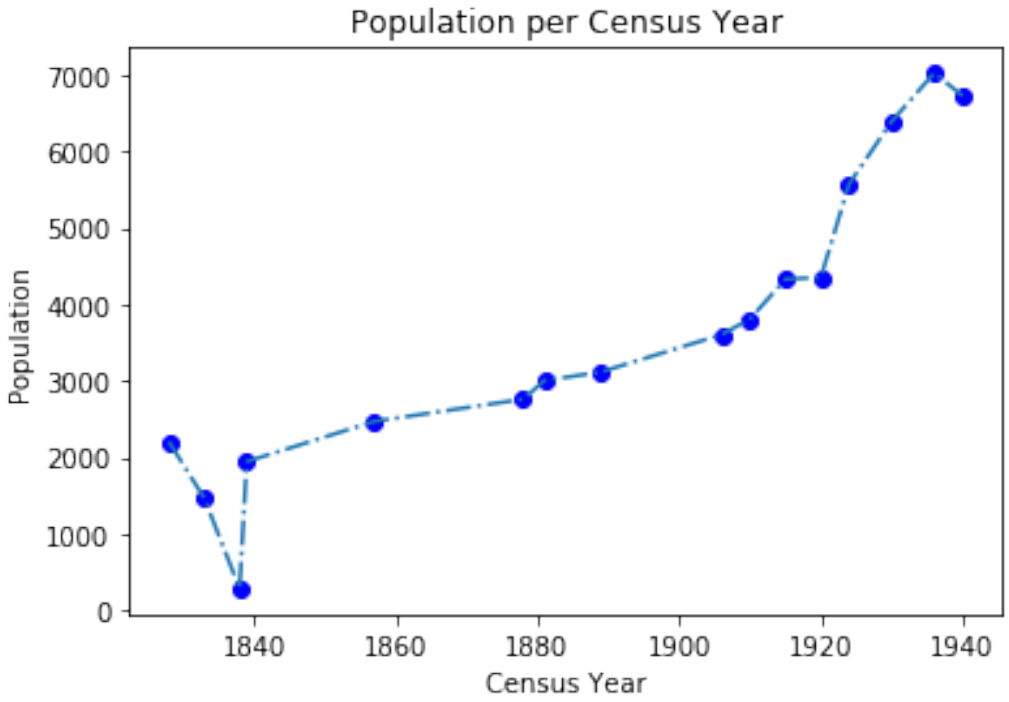

Figure 2.1: Population of Sant Feliu Town

Abel's sister for census year 1924. This makes the process of linking records hard and prone to errors.

2.2 FEBRL

Freely Extensible Bio-medical Record Linkage (FEBRL[1]) [17] is a synthetic dataset about bio-medical records of patients. Each record provides information like full name, address, postal code and date of birth. It contains 10000 records split across two datasets of 5000 records each. For each record, there exists a duplicate record in the other dataset. We attempt to link these duplicate records across these two datasets.

[1]https://github.com/J535D165/recordlinkage/tree/master/recordlinkage/datasets/febrl

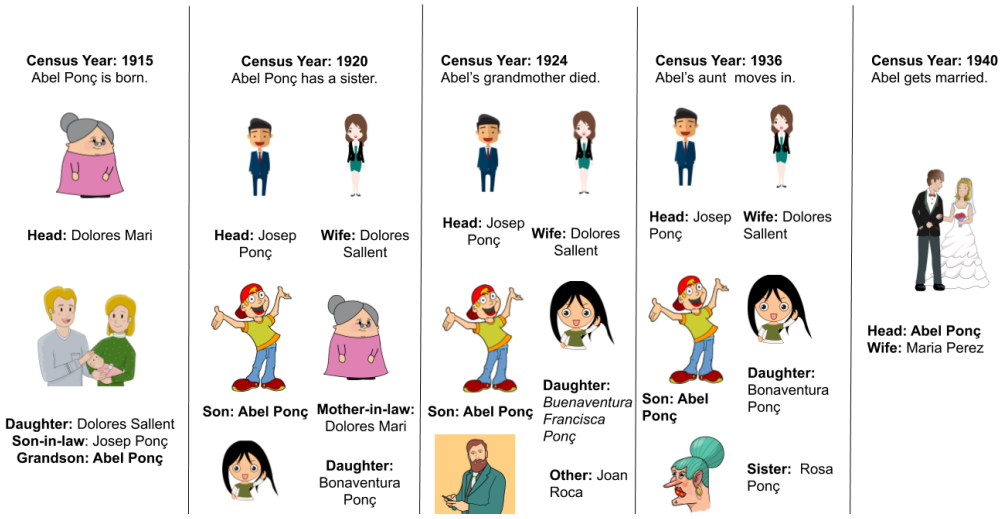

Figure 2.2: Changes in a Family with time

2.3 CORA

CORA[2] [18] dataset contains bibliographical information about scientific papers in XML format. It contains 1879 records of scientific citations with details like title, author(s), publisher, journal and date. The citations refer to 191 unique papers. We attempt to link all citations referring to a same scientific paper.

[2]https://hpi.de/naumann/projects/data-quality-and-cleansing/dude-duplicate-detection.html#c115302

Chapter 3

Record Linkage Methods

In this chapter we explore the state of the art methods suggested for the record linkage task. Data pre-processing algorithms, like cleaning and standardizing, can increase the accuracy of record linkage methods [19]. Phonetic encoding algorithms like Soundex [20] and its variants have been used across many applications since its invention in 1918. Hidden Markov Model has also been used to standardize name and address before proceeding with record linkage [21].

3.1 Fellegi and Sunter Framework

Fellegi and Sunter proposed a framework [8] in 1969 for probabilistic record linkage. The framework considers two datasets A and B containing records related to common domain. For each pair (a, b), where a belongs to A and b belongs to B, we estimate the probability of true match denoted by $P(m = 1|(a, b))$. If the probability is above a certain threshold the pair is considered as linked.

Figure 3.1 depicts the general record linkage work-flow based on [19] for the census dataset where the records related to the census year 1940 are considered as dataset A and records related to the census year 1936 are considered as dataset B. We attempt to link records pertaining to the same individual.

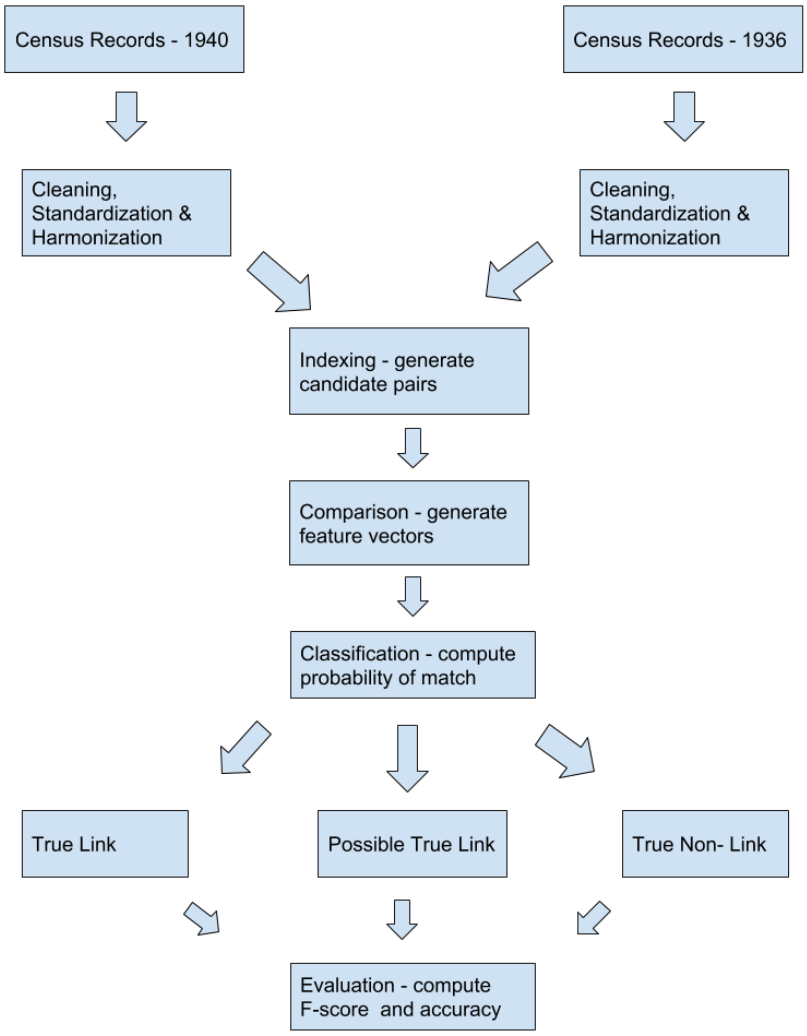

Figure 3.1: Record Linkage Workflow

3.2 Data Preprocessing

First step of data preprocessing is to clean the data i.e. remove extra space, newline and other unwanted tokens. Since, the source of records is same for census dataset, we use same data preprocessing methods both datasets A and B. Often the source, language or schema of the two datasets are different and they may require specific methods to normalize and standardize the data.

Census Dataset contains the harmonized values for the corresponding first names and surnames. Also, the occupation field has been standardized using HISCO [1] (Historical International Standard of Classification of Occupations).

3.3 Indexing

Indexing process aims at generating pairs of records, called candidate pairs, which are considered for linking process. Performance of record linkage is proportional to the number of candidate pairs. Maximum number of candidate pairs can be $M \times N$, where M is the number of records in first dataset and N is the number of records in second dataset. Since M and N are generally large, it is vital to reduce the number of candidate pairs. Blocking and Sorted Neighborhood are such algorithms which filters the candidate pairs based on similar attributes [19]. Indexing process results in a set of candidate pairs (a, b), where a belongs to dataset A and b belongs to dataset B. Size of the set of candidate pairs, denoted by S, has the upper bound of $M \times N$.

3.4 Feature Generation

Instead of relying on the exact string matches, various String edit distance algorithms like Levenshtein, Jaro, Jaro-Winkler, hamming distance and Q-gram are generally used to generate the features. For numeric or date-time attributes, partial similarity is generated using a decay function (like step, linear, gaussian or square functions) centered at a fixed origin [19]. Each attribute of record a and b are compared to generate a latent variable L, which denotes the probability of a true match.

For each candidate pair, K latent variables are generated by comparing corresponding attributes of the two records. In some cases a simple rule-based model to classify the candidate pair as matching or non-matching based on K latent variables can provide accurate results. For other cases, machine learning algorithms can be used to model a binary classifier. The K latent variables can be treated as a feature vector for each candidate pair (a, b).

[1] https://iisg.amsterdam/en/data/data-websites/history-of-work

3.5 Classification

The Fellegi and Sunter framework provides an optimal linkage rule using Nayman-Pearson Lemma [22]. The lemma states that a hypothesis test with two simple hypotheses that rejects small values of the likelihood ratio is the most powerful test of all tests with a given significance level of α. Fellegi and Sunter framework uses two such hypothesis:

$$H_0 : m = 0, \text{ denoting true non-link}$$
$$H_1 : m = 1, \text{ denoting true link}$$

To test the above hypothesis, first the likelihood ratio is computed as for each feature vector:

$$y = \frac{P(L_1, L_2, \ldots, L_K | m = 1)}{P(L_1, L_2, \ldots, L_K | m = 0)} \tag{3.1}$$

The feature vectors are sorted in decreasing order of likelihood ratio and classification decision is made using following function:

$$d(L_1, L_2, \ldots, L_K) = \begin{cases} \text{true link, if } T_m u \leq y \\ \text{possible link, if } T_\lambda \leq y < T_\mu \\ \text{true non-link, if } T_\lambda \geq y \end{cases} \tag{3.2}$$

where $T\mu$ and $T\lambda$ are threshold levels such that $T_\mu > T_\lambda$. Fellegi and Sunter proved that for given error levels (μ, λ) between $(0, 1)$, the above given function d is the best linkage rule possible. (μ, λ) denotes the probabilities of miss-classification as below:

$$\mu = E[\text{true link} | m = 0]$$
$$\lambda = E[\text{true non-link} | m = 1]$$

After feature generation machine learning algorithms like logistic regression, support vector machine or neural networks can be used to train the classification model. Broadly these methods can be classified as supervised and unsupervised methods based on requirement of labelled training data.

3.5.1 Supervised Methods

1. **Linear Regression Classifier**: The objective function predicts the probability of match given the candidate pair (a, b) denoted by $P(m = 1|(a, b))$, which is computed as below,

$$P(m = 1|(a, b)) = w_0 + \sum_{i=1}^{K} w_i L_i \tag{3.3}$$

   where, $L_1, L_2, \ldots, L_K$ are the K latent variables generated by comparing the records from the candidate pairs, w_0 is the bias and $w_1, \ldots, w_K$ are weights which are optimized by learning.

2. **Logistic Regression Classifier**: This type of classifier is also called a deterministic classifier. The probability is computed by applying a sigmoid function to the linear regression model as below:

$$P(m = 1|(a, b)) = \frac{1}{1 + e^{(w_0 + \sum_{i=1}^{K} w_i L_i)}} \tag{3.4}$$

   where e represents the exponential function.

3. **Naive-Bayes Classifier**: This is a probabilistic classifier based on Bayes theorem which assumes independence between the various features. The probability of match can be decomposed as below:

$$P(m = 1|(a, b)) = \frac{P(m = 1)P((a, b)|m = 1)}{P(a, b)} \tag{3.5}$$

   $P(m = 1)$ and $P(a, b)$ are computed using the available training data and $P((a, b)|m = 1)$ is computed using multivariate Bernoulli event model.

$$P((a, b)|m = 1) = \prod_{i=1}^{K} B_i \tag{3.6}$$

$$B_i = P(i|m=1)^{x_i}.(1 - P(i|m=1))^{1-x_i} \tag{3.7}$$

where x_i is the binarized value of the K Latent Variable defined above as L_i.

4. **Support Vector Machine**: The probability is computed by first applying a kernal to project the data to another space. Popular kernals are linear, polynomial, guassian function and radial basis function. The method finds a hyperplane such that there is a minimum margin of d between positive and negative samples. In case of linear SVM, we consider following hypothesis H_2 and H_3 for positive samples and negative samples respectively.

$$H_2 : \sum_{i=1}^{K} w_i L_i + w_0 \geq d \tag{3.8}$$

$$H_3 : \sum_{i=1}^{K} w_i L_i + w_0 \leq -d \tag{3.9}$$

3.5.2 Unsupervised Methods

1. **Expectation Condition Maximization Classifier** : This unsupervised algorithm is based on Fellegi and Sunter framework [8]. In 1993, Meng and Rubin proposed a variant to Expectation Maximization algorithm (EM method) called Expectation Conditional Maximization (ECM method [23]). The algorithm optimizes the model parameters θ by maximizing the log-likelihood of the complete data. Basic assumption made is that the components of the feature vector are conditionally independent. The first step (called E step) is to compute the log-likelihood of generating the observed features from the model parameters (θ). The second step (called CM step) is to conditionally maximize θ given the log-likelihood. The algorithms iterates over E-step and CM-step until θ are stable.

   Considering the binary assumption that the latent variables $L_i \in 0, 1$ for $i \in [0, 1, \ldots, K]$, Fellegi and Sunter framework allows to compute the weights w_i as below:

$$w(L) = log\left(\frac{P((L_1, L_2, \ldots, L_K)|m=1)}{P((L_1, L_2, \ldots, L_K)|m=0)}\right) \tag{3.10}$$

2. **K-Means Clustering**: The feature vectors can be partitioned in two clusters, match and non-match, using K-Means algorithm. The algorithm estimates the mean centres for both clusters. A feature vector is classified as matching if its distance from m_1, the mean centre of matching cluster, is less than the distance from m_0, the mean centre of non-matching cluster. The algorithm initializes (m_0, m_1) randomly and then iterates over following two steps:

   (a) *Assignment step*: Each feature vector f_j for j $\in$ [1, 2, ..., S], is assigned a class i.e matching (C_1) or non-matching (C_0), based on distance from mean center as below:

$$C_j = \begin{cases} 1, \text{ if } |f_j - m_0|^2 & \leq |f_j - m_1|^2 \\ 0, \text{ if } |f_j - m_0|^2 & < |f_j - m_1|^2 \end{cases} \tag{3.11}$$

   (b) *Update step*: After assigning a class for each feature, both mean centres are updated as below:

$$m_0 = \frac{\sum_{f_j \in C_0} f_j}{|C_0|} \tag{3.12}$$

$$m_1 = \frac{\sum_{f_j \in C_1} f_j}{|C_1|} \tag{3.13}$$

After applying a classification method, we get the probability of match $(P(m = 1|(a, b)))$ for each candidate pair. Next, we label them either as a link or a non-link. A threshold (θ) is chosen to label pairs as links if $P(m = 1|(a, b)) > \theta$, other the pair is labeled as non-link.

3.6 Evaluation

After classification step, we evaluate the performance of the record linkage task by dividing the candidate pairs in one of the following four sets:

1. **True Positive** (TP): True Links classified as links
2. **False Positive** (FP): True non-links classified as links

3. **True Negative** (TN): True non-links classified as non-links

4. **False Negative** (FN): True links classified as non-links

We denote number of candidate pairs in each set as N_{TP}, N_{FP}, N_{TN} and N_{FN}. We use following four quality measure:

$$\text{Accuracy} = \frac{N_{TP} + N_{TN}}{N_{TP} + N_{FP} + N_{TN} + N_{FN}} \tag{3.14}$$

$$\text{Precision} = \frac{N_{TP}}{N_{TP} + N_{FP}} \tag{3.15}$$

$$\text{Recall} = \frac{N_{TP}}{N_{TP} + N_{FN}} \tag{3.16}$$

$$\text{F-Score} = \frac{2.precision.recall}{precision + recall} \tag{3.17}$$

Each of these quality measures indicate good classification when they are close to 1. TN set generally has most candidate pairs (¿ 90%). Accuracy is not a good measure since it is dominated by N_{TN}. Precision and recall does not consider N_{TN}, thus offering better measure than accuracy. F-score is the harmonic mean between precision and recall, thus providing most significant quality measure. We learn our hyper-parameters by maximizing the F-score.

3.7 Experiments

We used python record linkage toolkit [2] for testing performance with the algorithms described above. For supervised learning, we chose Logistic Regression Model and for unsupervised learning we chose K-Means Clustering and ECM Classifier.

[2] `https://recordlinkage.readthedocs.io/en/latest/`

3.7.1 Setup

First we split the data in training, testing and validation sets and apply data preprocessing techniques. After data preprocessing, we divide the data in two datasets (A and B). Using indexing we generate the candidate pairs for linking. To train the classification model we first define a comparison function to compare two records of a dataset. We use either exact string match or Jaro-Winkler distance to compare the attributes. A single binary feature is generated by considering the Jaro-Winkler distance above the threshold of 0.85 as a match. Thus we get a latent variable for each attribute compared. We also filter true pairs from candidate pairs using the available label. True pairs are used for performance evaluation and also for training supervised methods.

1. **Census dataset:** Details of number of records in train, test and validation sets is given in Table-3.1. For training purpose, we labeled data from census years 1930 & 1889 as dataset A (total 9510 records) and data from census years 1936 & 1906 was labeled as dataset B (total 10629 records). Both dataset A and B, contains 8228 common records which are linked and are used as ground truth for training supervised methods.

   An index is created for all possible record pairs from dataset A and B. The index size is pruned by restricting the pairs with same surname. This reduces the resultant number of candidate pairs to 100 thousand from 47 million. We lose 1480 (17%) true links by pruning candidate pairs but this helps reduce the training time significantly. Precisely, we have 113568 candidate pairs with 6748 true pairs.

   For testing, we used the census years of 1910 & 1924 as dataset A (total 9384 records) and labelled records from census years 1915 & 1930 (total 10722 records) as dataset B. For generating candidate pairs we again used the blocking indexing on surname column. This results in a total of 240954 candidate pairs out of which only 9049 are the true links.

   We also keep records for validation purposes. The records pertaining to census years 1906 & 1936 (total 10629 records) is labeled as dataset A and records from census years 1910 & 1940 (total 10536 records) is labeled as dataset B. We extract 240954 candidate pairs by using surname blocking. We have 8412 true pairs for validation.

   We consider seven latent variables for matching normalised first name, normalised surname 1, normalised surname 2, year of birth, normalised

	Train Count	Test Count	Validation Count
Data set A	9510	9384	10629
Data set B	10629	10722	10536
Candidate Pairs	238854	240954	244696
True Pairs	6748	9049	8412

Table 3.1: Census Data Details

relationship with family's head, normalised occupation and normalised civil status. We used exact match for year of birth, occupation and civil status. Rest were compared using Jaro-Winkler with a threshold of 0.85.

2. **FEBRL dataset:** FEBRL dataset contains 5000 original records and 5000 duplicate records, split across 2 datasets with 5000 records each. We used 3000 records for training, 1500 records for testing and remaining 500 for validation, details provided in Table-3.2. Each record provides information about individual's name, surname, date of birth and address with postal code.

   The index is created by restricting candidate pairs to have same given name which results in total of 29066 pairs for training, 959 pairs for validation and 7674 pairs for testing. We lose about 33% of the true links due to blocking by given name. We have 1952 true links available to learn from during training phase and 995 true links in test set. We also have 340 true pairs as part of validation data.

	Train Count	Test Count	Validation Count
Data set A	3000	1500	500
Data set B	3000	1500	500
Candidate Pairs	29066	7674	959
True Pairs	1952	995	340

Table 3.2: FEBRL Data Details

   We consider only four latent variables given name, surname, date of birth and state of residence. Only surname is matched using Jaro-Winkler, for rest we use exact match.

3. **CORA dataset:** CORA provides total of 1879 records with details like title, author, publisher, journal, date, etc. We used 940 records related to 157 unique papers for training purpose (detailed in Table-3.3). One of the duplicate records is assigned to dataset A and other

to dataset B. The dataset contains 18 attributes in total, but most attributes are not defined for all records. This sparse dataset possess challenge in designing a machine learning model.

For training, we have 470 records in dataset A and another 470 records in dataset B.We have to rely on full indexing to generate the candidate pairs since no attribute is consistent in this dataset. This results in a total of 220900 candidate pairs for training of which 7961 are true links.

We collect 314 records in dataset A and 313 records in dataset B for testing purposes. The full indexing results in 98282 candidate pairs of which 3601 are true pairs. For validation, we keep 156 records dataset A and another 156 in dataset B. We have 24336 candidate pairs and 883 are the true pairs in validation set.

	Train Count	**Test Count**	**Validation Count**
Data set A	470	314	156
Data set B	470	313	156
Candidate Pairs	220900	98282	24336
True Pairs	7961	3601	883

Table 3.3: Cora Data Details

We considered eight latent variables title, author, journal, publisher, date, pages, volume and address. All fields are matched using Jaro-Winkler distance using the threshold of 0.85.

As part of feature generation, we compared all possible candidate pairs using above defined latent variables. The generated features and given list of linked pairs is used to train the classification model. Given K latent variables for a record pair, the model can predict whether the record pair is a true link or not.

We use same features for training Logistic Regression, ECM Classifier and K-Means Clustering models. The only difference is that we learn the Logistic Regression model in a supervised manner and others in unsupervised manner. For K-Means we used two clusters, one for matching pairs and another for non-matching pairs. Centers for these two clusters is estimated from the training features and a new record pair is classified according to its distance with the cluster centers.

3.7.2 Results

Dataset	Method	F-Score	Accuracy	Precision	Recall
Census	ECM	0.98	0.99	0.98	0.99
Census	Logistic	0.98	0.99	0.98	0.98
Census	K-Means	0.23	0.78	0.13	0.89
FEBRL	ECM	0.98	0.99	0.98	0.99
FEBRL	Logistic	0.99	0.99	0.98	0.99
FEBRL	K-Means	0.98	0.99	0.98	0.98
CORA	ECM	0.84	0.99	0.77	0.93
CORA	Logistic	0.70	0.98	0.85	0.59
CORA	K-Means	0.47	0.94	0.42	0.53

Table 3.4: Record Linkage Results

Table 3.4 provides the results for record linkage. For census dataset ECM Classifier and Logistic Regression provides the best F-Scores of 0.98. K-Means suffers from low precision and thus performs poorly. For FEBRL dataset ECM Classifier, K-Means and Logistic Regression provides the best F-scores of 0.98. For CORA dataset Logistic Regression provides the best F-scores of 0.98. K-Means and ECM Classifier has low precision and thus has a F-scores of around 0.75.

We also experimented with a simple classifier by computing the sum of the latent variables which provided an F-Score of 0.85. We considered at least six of the seven latent variables need to be set to be considered as matching pair.

We noted that training a binary classifier as per the Fellegi and Sunter framework can provide efficient record linking results. Though the major concern is False Positives, since incorrect linking can have a chain effect and have negative impact on the concerned research. Since the proportion of true pairs is very low (3% census, 7% febrl, 4% cora) compared to candidate pairs, it is hard to avoid false positives without increasing false negatives.

Figure-3.2 shows the normalized weights after training ECM and Logistic Regression models. We note that first name, surname and year of birth are most informative attributes. Both models learns the change or error in these attributes. Since civil status, relation and occupation often changes over multiple census years, the model assigns them very low weights. Both

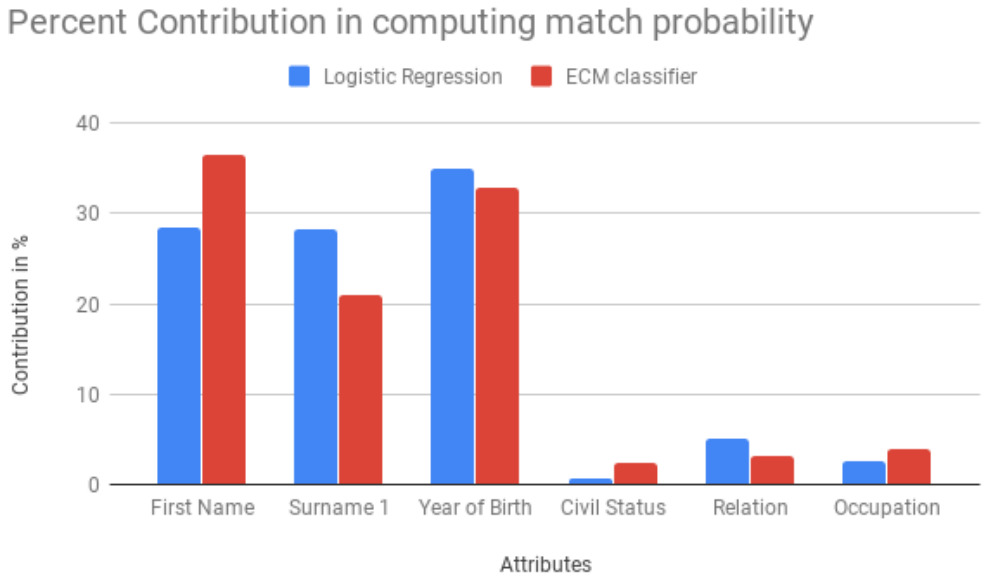

Figure 3.2: Percent Contribution in computing match probability

models fails to learn the change or evolution in these attributes.

Next, we analyze the errors for Census dataset. We noticed 247 and 314 False Negatives for ECM and logistic regression methods. Following are the different reasons for the incorrect classification:

1. **Full names are different:** This is the most common case where either the first name or the surname is different for the record pair. About 95% of the false negatives have significant difference in full names (234 for ECM and 301 for Logistic regression).

2. **Years of birth are different:** For ECM method we have about 30% (75 record pairs) such cases where the record pair have different year of birth mentioned. For Logistic Regression we have about 23% (72 record pairs) such cases.

3. **Missing Attributes:** The process of reading the raw data from a image is error prone. If the process fails to read any specific attribute, it is marked as 'illegible'. We have 12 such pairs for ECM and 15 in case of logistic regression.

4. **Inconsistent gender:** We noticed that certain false negative record pairs have different gender associated to them. For example there are 13 such pairs where the relation with head is *fill* (son) in one record and *filla* (daughter) in other.

5. **Marriage:** We noticed that often the civil status of the person changes with time and along with it changes the relationship with family's head. We found five cases of women who got married and became wives (esposa). Also, there is a single false positive for a male who got married and became head of the family.

We also had 87 False Positives for ECM and 51 for Logistic Regression. Following are the common reasons for False Positives.

1. **Same Full Name:** We have 9 such cases for both methods where the full names of both records is same but records belong to different individuals.

2. **Missing Year of Birth:** We noticed 16 false positives for ECM where year of birth is missing for one of the two records. We only have 3 such cases for logistic regression.

3. **Same Civil Status and Relation:** We have 53 cases for ECM and 22 for logistic regression where both civil status and relation with head of family are same.

We have published the record linkage tests along with FEBRL and CORA dataset as public git repository [3].

[3] `https://github.com/bhaskargautam/record-linkage`

Chapter 4

Entity Alignment Methods

In this chapter we explore the state of the art methods for entity alignment using knowledge graph embedding. In computer science a graph is represented as a list of vertices (V) and edges (E). An edge connect two vertices (v_1, v_2). A directed graph has a fixed direction for each edge and thus precedence of vertices is important. A labelled graph has a label associated for each vertex and edge. A Knowledge Graph is a directed and labelled graph representing concepts in a specific domain.

4.1 Knowledge Graph Representation

The Knowledge graph (KG) represents a collection of interlinked descriptions of entities real-world objects, events, situations or abstract concepts in a formal structure. Popular knowledge graphs include Freebase, DBpedia, YAGO, Satori, etc. which have billions of triples and millions of entities. These large knowledge graphs are still incomplete and some links are missing. Many researchers have developed various link prediction methods using knowledge graph embedding [24] [25] [26]. Knowledge Graph is also called linked data in terms of semantic web.

We represent a knowledge graph as $G = (E, R, T)$, where E is set of all entities, R is the set of all relations and T is the set of all triples. A single triple is denoted as (h, t, r), where h is the head element from set E, r is the

relation element from set R and t is the tail element from the set E. Entity Alignment as a process attempts to predict new links (link discovery) of form $(e1, e2, \text{same\_as})$, where e_1 is the entity from dataset A, e_2 is from dataset B and same_as is considered a new relation to denote that e_1 and e_2 are linked or aligned.

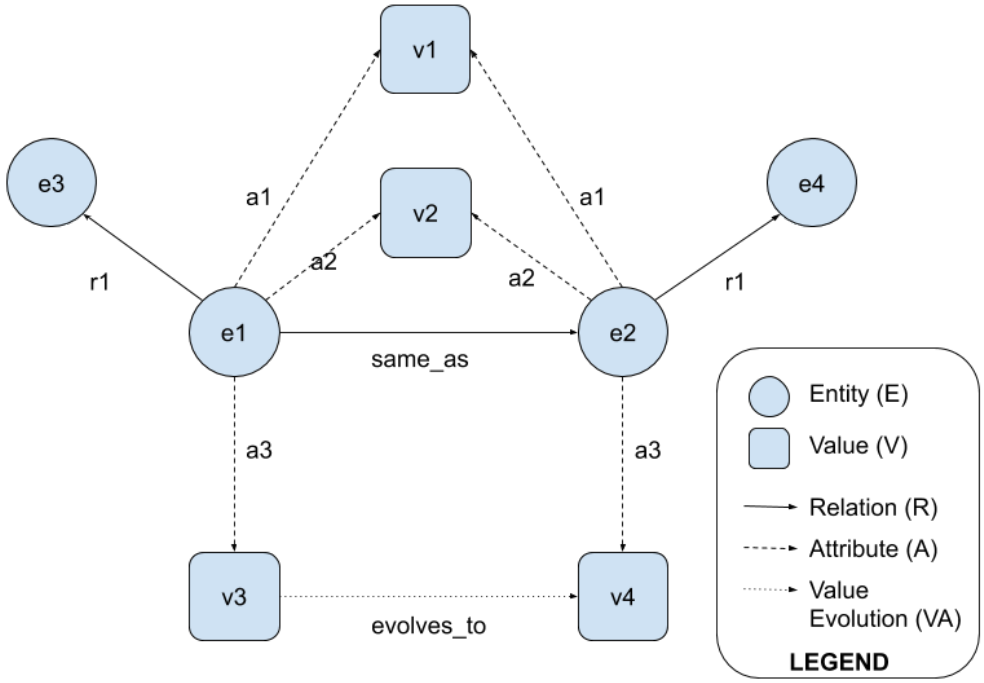

Figure 4.1: Knowledge Graph Representation

Figure 4.1 represents the various terminologies used for knowledge graph representation. Few methods partition the entity set (E) to extract value set (V). The entities having one-to-many or many-to-many relations e.g. 'male' or 'female', are considered as values. Similarly, the relation set (R) is also partitioned to extract attribute set (A). The relations linking entities to values are considered as attributes. The aim of entity alignment task is to predict the relation 'same_as' between the two entities (e_1, e_2). Further, we introduce the 'evolves_to' relation through which we attempt to learn the valid / invalid value evolution in next chapter.

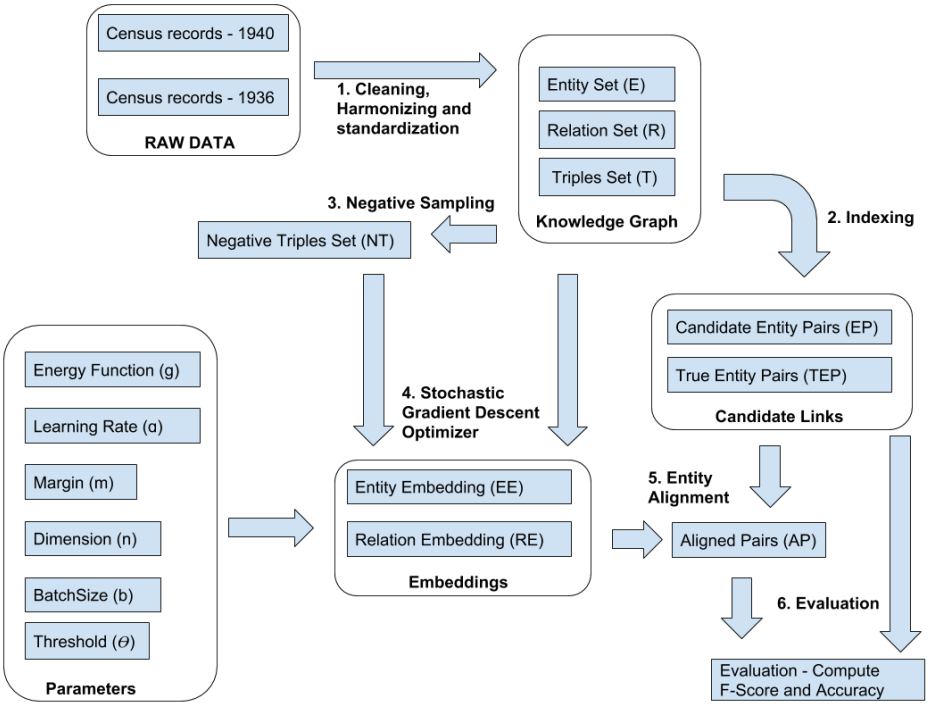

Figure 4.2: Entity Alignment using Graph Embedding

4.2 Entity Alignment Work-flow

Entity alignment uses a similar workflow as record linkage and attempts to link / align entities in a knowledge graph. Figure 4.2 depicts the work-flow where first the embedding vectors are learnt for entities and then features are generated by comparing embedding vectors.

1. **Knowledge Graph Construction :** We use same cleaning and standardization methods as discussed in previous chapter. After cleaning and standardization of raw data, we first model the training data as a knowledge graph. We create a new entity for each record and connect it with the various attributes given for the record. Unique nouns like 1994, Abdulia, Miami, etc. are also included in the entity set. We use relations like 'first_name', 'surname', 'occupation', etc. For census dataset we also create a node for each household and link it with all individuals of that household using the 'relation with head of house-

hold'.

2. **Indexing :** The entities are split in two datasets A and B and indexing process generates the candidate entity pairs (EP) of form $(e1, e2)$, where $e1 \in A$ and $e2 \in B$. We utilize blocking indexing process to control the size of candidate pairs. We also filter out the subset of candidate pairs called true entity pairs (TEP) which forms the ground truth and is used for evaluating the results.

3. **Negative Sampling :** The third step requires to generate a set of negative triples from the given knowledge graph. The negative samples are generated by replacing one of the three components of a triple with another valid value e.g. the head entity h is replaced with some other random entity h_n, such that $(h_n, t, r) \notin$ T. We randomly select the component for replacement with equal probabilities. For each valid triples in T, we generate N_r negative samples by replacing relation and N_e negative samples by replacing either head or tail. $(N_r + N_e)$ represents the negative sampling rate. Thus, size of negative sample set (NT) is $((N_r + N_e) \times |T|)$, where $|T|$ is the size of positive triples set (T). Based on requirement the ratio of positive to negative triples can be controlled by changing N_r and N_e.

4. **Learning Graph Embedding :** Instead of generating feature vectors by comparing the entities directly, we first learn the n-dimensional embedding for each entity and relation, denoted with the same letter in bold face as **(h, t, r)**. The n-dimensional embedding $\mathbf{X}$ are initiated using Xavier's initializer [27], which sets the embedding uniformly between $(-\frac{\sqrt{6}}{\sqrt{n}}, \frac{\sqrt{6}}{\sqrt{n}})$. Thus, embedding are represented by two matrices Entity Embedding (EE) having shape of $(|E| \times n)$ and Relation Embedding (RE) having the shape of $(|R| \times n)$.

   We use Stochastic Gradient Decent Optimizer to train both entity and relation embedding. Both positive triples and negative triples are fed into the optimizer in the batches of batch size (b). We compute loss and gradient using the energy function (g) and then update the embedding accordingly using the learning rate (α). Margin (m) or bias is used to prevent the over-fitting on training data.

   The Stochastic Gradient Decent algorithm is used to optimize the embedding such that the knowledge graph holds i.e. $P((h, r, t)|\mathbf{X})$ should be more for a positive triple than another negative triple. We optimize the probability $P(h|r, t, \mathbf{X})$, $P(t|h, r, \mathbf{X})$ and $P(r|h, t, \mathbf{X})$ instead of $P((h, r, t)|\mathbf{X})$.

Bordes suggested in 2013 to use soft-max function to predict probability of head entity given relation, tail and embedding, ($P(h|r,t,\mathbf{X})$) as below:

$$P(h|r,t,\boldsymbol{X}) = \frac{e^{g(\boldsymbol{h},\boldsymbol{t},\boldsymbol{r})}}{\sum_{h_n \in (E-h)} e^{g(\boldsymbol{h_n},\boldsymbol{t},\boldsymbol{r})}} \tag{4.1}$$

where, g is the energy function which indicates correlation between relation r and entity pair (h,t). Various energy functions which can be used for generating graph embedding are discussed in subsequent sections. Based on the structure the knowledge graph we can choose different graph embedding methods which are described in next section.

5. **Entity Alignment :** The cosine distance between the embedding of entities $(e1, e2)$ is used to align two entities. If the cosine distance is less than the defined threshold (θ) the pair is added in aligned pairs (AP) set. Also, for each entity we rank the closest entities in the graph for rank analysis. In real-world scenario, the user expects to examine top-K (say top-3 or top-10) before actually linking two entities. This process helps to reduce incorrect alignment and is often used in information retrieval systems.

6. **Evaluation :** Lastly, we compare the aligned pairs (AP) with true entity pairs (TEP) to evaluate the performance of the model. We compute F-Score, accuracy, precision and recall as discussed in previous chapter.

   For information retrieval analysis, we create Q queries as the set of first entities in each entity pair *(a, b)*. For each unique query, we rank the list of entities closet to entity a based on cosine distance between their embedding. We use following Information Retrieval Metrics to compare performance:

   (a) **Mean Precision@K** (MP@K): For each query, we rank all the entities based on cosine distance from entity *a*. Then, we label each ranked entity as relevant or not by verifying in true entity pairs set. Precision@K is defined as the ratio between number of relevant records in top-K results and number of total results under consideration i.e. K. We consider the mean of precision@K for each unique entity *a*. We evaluate models for two K values - 1 and 10.

(b) **Mean Reciprocal Rank** (MRR): This metric is the multiplicative inverse of rank of the first relevant entities. Mean value of reciprocal ranks for each query is computed as:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{4.2}$$

(c) **Mean Average Precision:** (MAP): Average Precision for a particular query (Q) is computed by averaging precision for each relevant entity in the ranked results. The mean of average precision across multiple queries is termed a mean average precision. In case, only one relevant entity exists for all queries the Mean average precision is same as Mean Reciprocal Rank.

All three metrics have the value in the range of 0 to 1. The model performs better when these metrics are close to 1. MAP and MRR are more informative measures than MP@K. We optimize our hyperparameters by maximizing MRR.

4.3 Semi-Supervised Learning

Generally the labelled data is not available or is scarce, since the labelling require human efforts and is costly. To counter this problem, many semi-supervised tricks have been suggested like self-learning, active learning and few shot learning.

1. **Self-Learning:** The model is trained on the given small labelled data to classify the unlabelled data. The most probable matches are considered as true matches and added to the labelled data (train data) itself. The model is updated based on these new samples and process continues until no new matches are found.

2. **Active Learning:** The problem with self-learning is that the false positives also gets added to the training data, thus forcing model to learn incorrect patterns. To avoid this the probable matches are reviewed manually (or checked with ground truths) and only the true positives are added to the training data. This helps the model to correct any erroneous pattern learned from available small training data.

3. **Few Shot Learning:** In case we have small number of labelled inputs per class (say K), we try to learn the classification model by comparing the test input with given labelled samples. This method is useful when number of classes is large and samples per class is low. This method is also called K-shot learning or One-shot learning (if $K = 1$).

These methods can be used to improve the performance of the classification model when training data is scarce. Active Learning still requires human efforts but lesser than labelling the raw data, since the number of candidate pairs to be verified is filtered by the classification model.

4.4 Knowledge Graph Embedding Methods

We considered three knowledge graph structures - ER (Entity-Relation), EAR (Entity-Attribute-Relation) and $ER - ER$ (Entity-Relation−Entity-Relation), which are described below along with popular embedding methods. Each method tries to project the knowledge graph to a n-dimensional vector space for entity alignment purpose.

4.4.1 ER (Entity-Relation) Graph

Entity-Relation (ER) Graph models the knowledge graph as three sets - Entity (E), Relation (R) and Triples (T). ER graph follows a simple graph structure, where all edges are represented a triple of form (h, t, r) where h and t are the head and tail entities and r is the relationship between them. We model a single knowledge graph for both dataset A and B i.e. we project the entities from both datasets to the same vector space. The knowledge graph construction steps for the three datasets are given below.

1. **Census Dataset:** Figure-4.3 depicts a sample ER graph for Census Dataset. We create two new entities for each record, one corresponding to individual identifier in the given census year and other corresponding to the household identifier in the given census year. We connect the individual to with relationship with the head of the household e.g. (19241481, 19240287, esposa), where 19241481 is the entity identifying

an individual, 19240287 is another entity identifying a household from census year 1924 and esposa (wife in Spanish) is the relation between the individual and the head of the household. Next, we create entities for each of the attributes associated with individual i.e. name, occupation, etc. and link them to individual's entity. This results in in total of 56 possible relationships which form the relation set R. We add one more relationship 'same_as' in relation set to link aligned entity pairs.

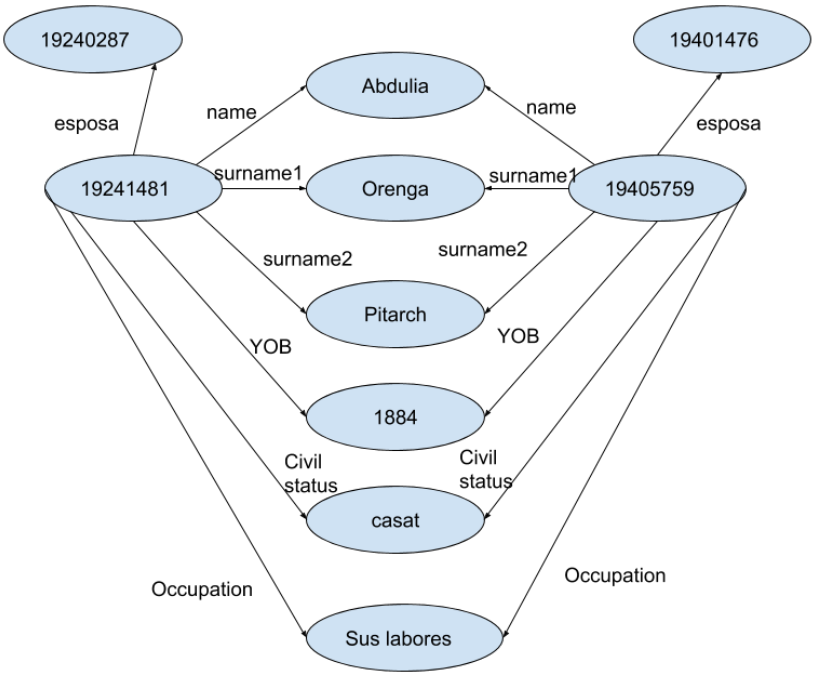

Figure 4.3: Knowledge Graph for Census dataset

2. **FEBRL :** Similarly for FEBRL dataset, we create a new entity for each record and creates a new triple as (rec_1070, Miami, addr_2), where *rec*_1070 is the entity identifying a person living in Miami city. Figure-4.4 depicts a sample ER graph for FEBRL. We also create a NULL entity for blank or missing relationships e.g. (rec_1070_dup_0, NULL, state), where *rec*_1070_*dup*_0 is the entity identifying a person whose residing state is unknown. We only use 6 possible relationships - name, surname, state, date of birth, postcode and same_as. We add five triples for each record, one for each relation except same_as. Thus we have total of 30000 triples for 6000 records in training set (3000 in

each in dataset A and B).

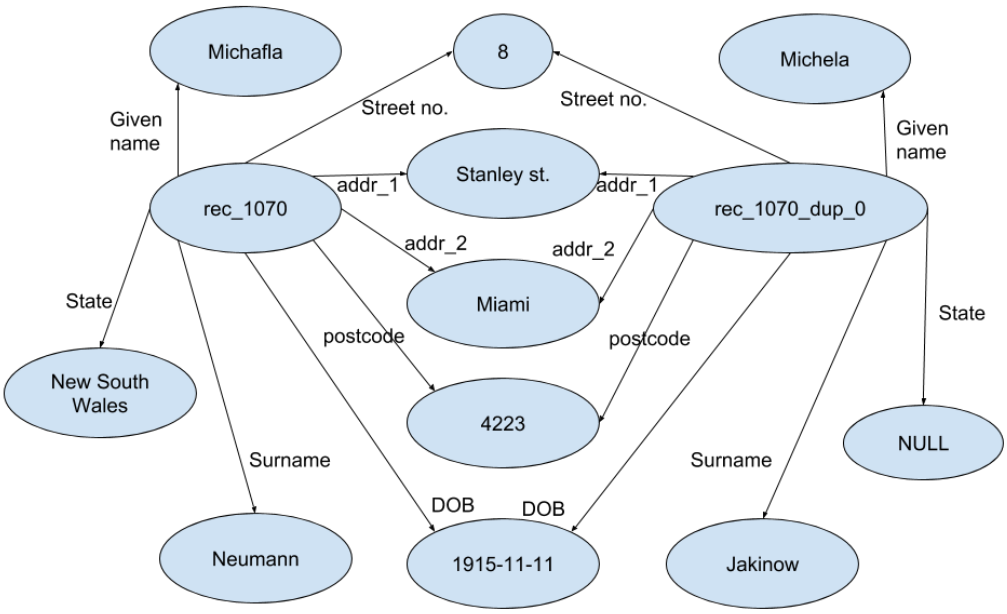

Figure 4.4: Knowledge Graph for FEBRL dataset

3. **CORA :** Figure 4.5 depicts the sample sub-graph for Cora dataset, in which each eclipse represents an entity or node and the edges connecting the nodes represents the relations. We first create a new entity for each record and assign it a unique id e.g. Cora1, Cora2, etc. Next, we create entities for other information related to record e.g. 1994, 893, etc. and lastly we create a triple to store the relationship e.g. (Cora1, 1994, Year), where Cora1 is the entity to identify an journal published in 1994.

   We also create a new entity for each author individually and connect them to entities identifying the record by the relationship of Author. Thus, Author forms a one-to-many relationship with identifying entities. We define 15 relations like author, year, title, etc to link entities in the knowledge graph.

Table-4.1 provides the length of entity, relation and triples set for the

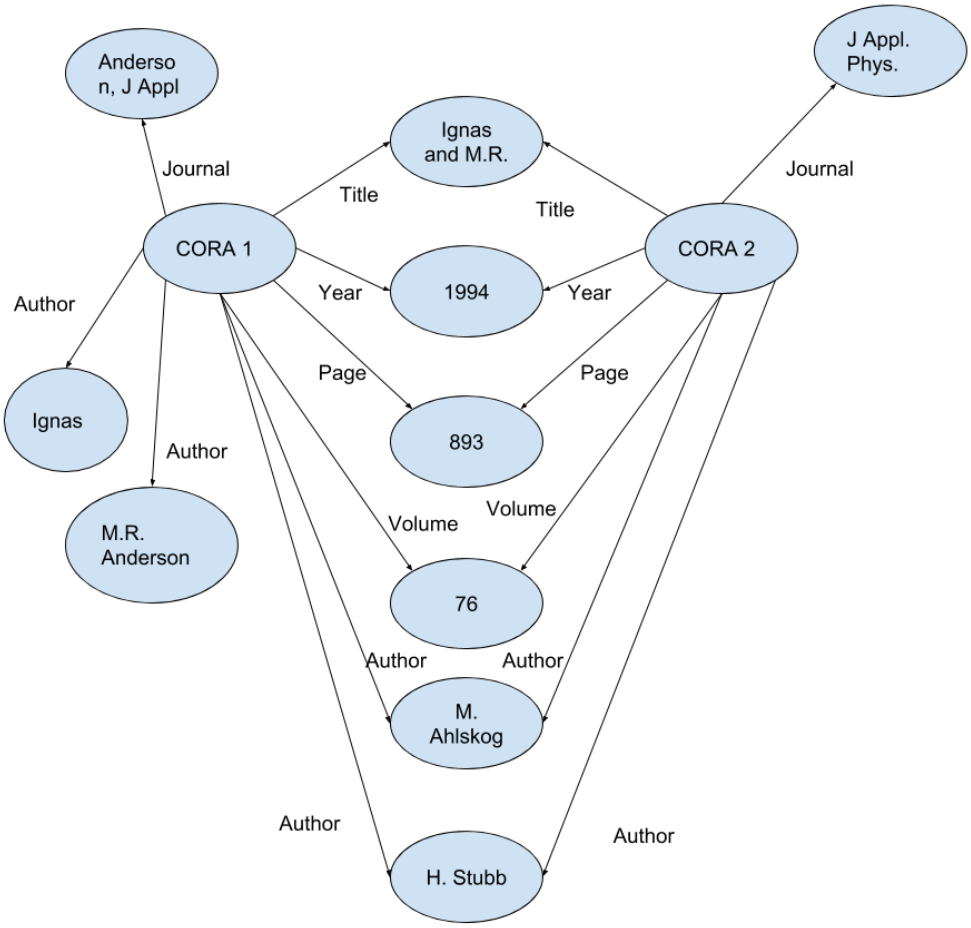

Figure 4.5: Knowledge Graph for CORA dataset

three datasets after modeling their training data as an ER graph.

After generating the knowledge graph we create candidate entity pairs using indexing step and generate negative triplets by negative sampling step. Next, we generate embedding for the ER graph by projecting it to n-dimensional vector space. Following are the various graph embedding methods which can be used for ER graph.

1. **TransE:** [24] is a translation based method to generate knowledge graph embedding. It learns the embedding of entities and relations to support further applications, such as link prediction, triples classi-

	Entity Count	Relation Count	Triple Count
Census	20375	57	96250
FEBRL	13497	6	30000
Cora	2704	16	7550

Table 4.1: ER Graph Details

fication, etc. TransE interprets relations as translations operating on low-dimensional embedding of the entities. If a triple (h, t, r) holds, then the embedding of the tail entity t ($\boldsymbol{t}$) should be close to the sum of embedding of head entity h ($\boldsymbol{h}$) and embedding of relationship r ($\boldsymbol{r}$). The energy function for TransE is given as:

$$g(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{r}) = ||\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}||_{L1/L2} \tag{4.3}$$

The function returns a low score if (h, t, r) holds and returns a high score if (h, t, r) does not hold. The model has poor accuracy while modelling one-to-many and many-to-many relations. It uses margin-based loss function for training stochastic gradient descent optimizer, which is defined as:

$$Loss = \sum_{(h,t,r)\in T} max(0, M + g(h, t, r) - g(h', t, r)) \tag{4.4}$$

where M is margin and h' represents a negative head.

2. **TransH:** [25] is another translation based method which enables an entity to have different embedding based on the relation it is involved in. An entity is first projected on to a hyper-plane represented by a normal vector w_r, where r is the given relation. The energy function is given by:

$$g(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{r}) = ||\boldsymbol{h} - w_r^T \boldsymbol{h} w_r + \boldsymbol{r} - (\boldsymbol{t} - w_r^T \boldsymbol{t} w_r)||_{L1/L2} \tag{4.5}$$

The model projects the entity according to the relation it is involved in. This allows the model to perform well even with one-to-many and many-to-many relations.

3. **TransR:** [26] models entities and relation in different spaces and uses a projection matrix (M_r) to project entities from entity space to relation space. The energy function is given by:

$$g(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{r}) = ||\boldsymbol{h} M_r + \boldsymbol{r} - \boldsymbol{t} M_r||_{L1/L2} \tag{4.6}$$

The model to perform well with one-to-many and many-to-many relations since it uses a different projection matrix for each relation.

4. **RESCAL:** [28] is a compositional model which generates relational embedding using tensor factorization. It has a basic instinct that each dimension in the embedding represents a single feature i.e. if two entities have same feature value, both should have a particular bit set in embedding. Thus, a single dimension of embedding vectors only contains the information about a single feature. The energy function is defined as the tensor product between two entities (h, t):

$$g(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{r}) = \boldsymbol{r}^T(\boldsymbol{h} \otimes \boldsymbol{t}) = \boldsymbol{r}^T \left(\sum_{i=1}^{n} \boldsymbol{h_i t_i} \right) \tag{4.7}$$

Where n represents the dimensions of entity space and $\bigotimes$ represents the tensor product.

5. **HOLE:** Holographic Embedding (HOLE) [29] combines the expressive power of the tensor product with the efficiency and simplicity of TransE by using the circular correlation of vectors to represent pairs of entities. The circular correlation operator is based on Holographic Reduced Representations [30]. The energy function for a triplet can be represented as:

$$g(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{r}) = \boldsymbol{r}^T(\boldsymbol{h} \odot \boldsymbol{t}) \tag{4.8}$$

Where $\odot$ represents a composition operator which is defined as a circular correlation operation for the k^{th} bit as below:

$$[\boldsymbol{h} \odot \boldsymbol{t}]_k = \sum_{i=1}^{n} \boldsymbol{h_i t_{(k+i)\%n}} \tag{4.9}$$

This is the compressed form of tensor product. The intuition here is that a single feature is represented by a partition of embedding vector.

6. **PTransE:** Path-based TransE [31] is an extension of TransE model which considers multi-step relation paths along with direct links. The energy function is defined as:

$$g(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{r}) = ||\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}||_{L1/L2} + g(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{P}) \tag{4.10}$$

where, $P(h, t)$ is the set of multi-relation paths from h to t. $g(h, t, P)$ models the inference correlations between relations in multi-step paths (P) and is defined as:

$$g(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{P}) = \frac{\sum_{p \in P(h,t)} R(p|h,t).g(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{p})}{\sum_{p \in P(h,t)} R(p|h,t)} \tag{4.11}$$

where, $R(p|h,t)$ is the relation path reliability and is computed by following the path p. Consider the path p as $h \xrightarrow{r_1} e_1 \xrightarrow{r_2} e_2 \ldots \xrightarrow{r_l} t$. We initialize $R(h) = 1$ and compute $R(e_i)$ as follows:

$$R(e_i) = \sum_{v \in N^-(e_i)} \frac{R(v)}{|N^-(e_i)|} \tag{4.12}$$

where $N^-(e_i)$ is the set of direct predecessors of the entity e_i. By recursively performing resource allocation, finally we compute $R(t)$ which gives us the reliability of the path p i.e. $R(p|h,t) = R(t)$. Thus, P-TransE regularizes the energy for multi-step paths by reducing the reliability factor in proportion of the number of steps in the path.

In order to compute $g(h,t,p)$, we can represent p as addition of relations $(\boldsymbol{r_1} + \boldsymbol{r_2} + \ldots + \boldsymbol{r_l})$ or multiplication of relations $(\boldsymbol{r_1}.\boldsymbol{r_2}.\ldots.\boldsymbol{r_l})$. [31] also experimented with Recurrent Neural Networks (RNNs) to represent the path embedding. Thus, energy function for a path can be given as:

$$g(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{p}) = ||\boldsymbol{h} + \boldsymbol{p} - \boldsymbol{t}||_{L1/L2} = ||\boldsymbol{p} - \boldsymbol{r}||_{L1/L2} \tag{4.13}$$

It is also important to limit the number of paths selected for each entity pair, since it grows exponentially with step size. It is recommended to work only with 2-step or 3-step paths.

7. **BootEA:** Bootstrapping Entity Alignment [32] aims to update entity embedding using bootstrap process which adds likely alignments to the knowledge graph. This is a semi-supervised method which updates the embedding in iteration. After each iteration, most probable alignments are added back to knowledge graph and then embedding vectors are updated. It defines the limit based loss function as:

$$Loss = \sum_{t \in T^+} [g(t) - \gamma_1] + \mu. \sum_{t' \in T^-} [\gamma_2 - g(t')] \tag{4.14}$$

where T^+ is the set of positive triples and T^- is the set of negative triples. γ_1, γ_2 and μ are hyper-parameters. $g(t)$ is energy function for triple t, as defined by TransE as $g(t) = ||h + r - t||$. BootEA utilizes

ϵ-truncated negative sampling i.e. we replace an entity by its neighbour within ϵ distance for sampling negative triples.

[32] also suggests to use parameter swapping for aligned pairs for entities i.e. we add new triples to T^+ by replacing entity with its aligned pair.

After generating the graph embedding, we evaluate the performance of embedding method by computing the cosine distances between the embedding of two entities. The true links are expected to have a small cosine distance, less than the configured threshold (θ).

4.4.2 EAR (Entity-Attribute-Relation) Graph

Entity, Attribute and Relation (EAR) graph differentiates between a relation and attribute as the fields having too few possible values (like gender) are considered as attributes. Similarly, the entities set does not contain such values which are shared between large number of other entities (like male and female), such values are collected in a separate list called attribute values. For EAR model, we have two sets of triples - relational triples and attributional triples. For a relational triple (h, t, r), h and t belongs to entity set and r belongs to relations set. For a attributional triple (h, v, a), h belongs to the entity set, a belongs to attribute set and v belongs to attribute value set. Thus, we represent EAR knowledge graph as six sets - Entity (E), Attribute (A), Relation (R), Value (V), Attributional Triples (AT) and Relational Triples (RT).

We model our knowledge graph for census dataset as shown in Figure 4.6. The entities which does not refer to an individual or household are considered as value. We consider the relationship of an individual with the head of household as the only relation while rest are considered as attributes. Other relations like name, occupation, etc are many-to-one relations, thus considered as attributes.

For FEBRL and Cora datasets, we consider all relations as attributes and only use a single relation to define same_as relation. Thus the entity set for each only contains the ones which identify a record e.g. Cora1, rec-1-org, etc. Remaining entities like 1994, 893, Miami, etc are considered as values corresponding the attributes.

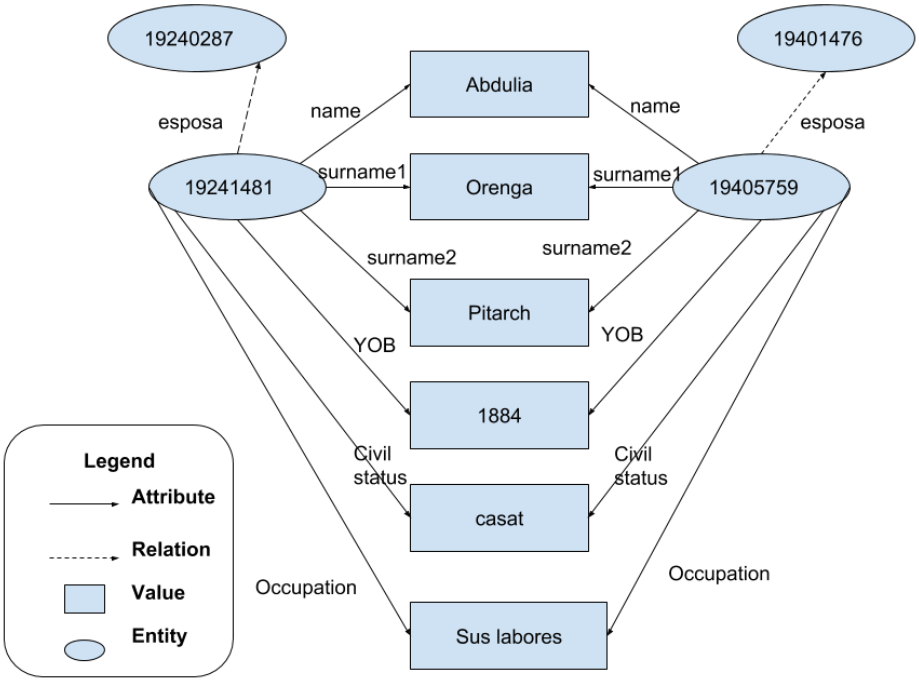

Figure 4.6: EAR model for Census dataset

Table-4.2 provides the length of entity, relation, attribute and triples set for the three datasets after modeling their training data as an EAR graph. This model aims to remove the one-to-many and many-to-many relationships between entities by labeling such relationships as attributes. We project the entity and relation in a relational vector space, while attributes and values are projected to attributional vector space. We use a projection matrix to transform an entity to attribute vector space from relation's vector space.

	Entity	**Attr**	**Rel**	**Val**	**Attr. Triple**	**Rel. Triple**
Census	16883	6	51	3492	82500	13750
FEBRL	6000	5	1	7497	30000	0
Cora	940	15	1	1764	7550	0

Table 4.2: EAR Graph Details

Following are two graph embedding methods which can be used for EAR graph:

1. **KR-EAR:** [33] noted that translation based methods have trouble modelling one-to-many and many-to-many relations. They suggested to use EAR graph which can be represented as $G_{EAR} = (E, R, A, V, AT, RT)$. KR-EAR has two core components:

   (a) **Relational Triple Encoder (RTE)**: This aims to embed entities and relations to capture correlation between them. Given a true relational triple $(h, t, r) \in RT$, we predict $P(h|r, t, m = 1)$ by applying approximation to the softmax function as below:

   $$P(h|r, t, m = 1) = \prod_{(h,t,r)\in RT} \left[\sigma(g(h, t, r)) \prod_{i=1}^{c_1} \mathbb{E}_{h_n \in (E-h)} \sigma(g(h_n, t, r)) \right] \tag{4.15}$$

   Where g is a energy function and any of the translation based methods like TransE or TransR can be used. σ represents the sigmoid function and c_1 is the constant that determine number of negative samples to be used. $\mathbb{E}$ represents the event of sampling negative triples.

   (b) **Attributional Triples Encoder (ATE)**: This aims to encode the correlation between entities and attributes. Given a attributional triple $(h, v, a) \in AT$, we predict the $P(v|h, a, m = 1)$ using the softmax approximation as:

   $$P(v|h, a, m = 1) = \prod_{(h,v,a)\in AT} \left[\sigma(g(h, v, a)) \prod_{i=1}^{c_2} \mathbb{E}_{h_n \in (E-h)} \sigma(g(h_n, v, a)) \right] \tag{4.16}$$

   Where c_2 is the constant determining number of negative values to be used. The energy function g here is defined as below:

   $$g(\boldsymbol{h}, \boldsymbol{v}, \boldsymbol{a}) = ||\tanh(\boldsymbol{h}W_a + \boldsymbol{a}) - \boldsymbol{v}||_{L1/L2} \tag{4.17}$$

   Where W_a is the projection matrix which projects an entity to attribute space.

2. **SEEA:** Self-Learning and Embedding based entity alignment [15] is inspired by KR-EAR and uses EAR representation. This method assumes only single relation denoting the aligned entities. For RTE, the

energy function used is given by:

$$g(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{r}) = ||\boldsymbol{h} - \boldsymbol{t}||_{L1/L2} \tag{4.18}$$

SEEA utilizes the same ATE as defined by KR-EAR. The RTE and ATE steps are executed to generate embedding an then the algorithm utilizes the self-learning mechanism to optimize the embedding.

After generating embedding for the entities, we select the top β entity pairs which are most similar as per energy function (g) defined above. A pair $(e1, e2)$ is considered as aligned only if both entities are mutually closest to each other i.e. $e2$ is closet entity to $e1$ and vice versa. We add these top β entity pairs to set RT and regenerate the embedding. The method updates the embedding using the results of entity alignment of previous iteration in self-learning manner. The algorithm is executed in iterations until no new aligned pairs are detected.

4.4.3 ER-ER (Entity-Relation,Entity-Relation) Graph

ER-ER graph models the data as two separate ER graphs. The entities of each knowledge graph is projected to the different vector space. This model is useful when the initial data sources are in different languages and/or contains different relationships.

In this model, we create two separate knowledge graph using ER model for each. The first knowledge graph contains the records related to dataset A from both train and test data. While the second knowledge graph pertains to the records of dataset B.

We project the embedding for each ER graph in a different entity-relation space, unlike TransE or TransH where we use only a single entity-relation space. We also utilize the prior aligned entities known from the training set by appending them to triples set using same_as relation. The prior pairs i.e. the aligned entities from train data, are used to learn the translation from vector space of first graph to the vector space of second graph. Table 4.3 provides the detailed size of knowledge graphs constructed using ER-ER graph model.

This model is suitable when the two sources of data are in different language (cross-lingual linkage) or utilizes different ontology. Prior Pairs or

Initial inter-lingual links are used to update the embedding and reduce cosine distance of two linked entities.

	Ent. A	**Rel. A**	**Tri. A**	**Ent. B**	**Rel. B**	**Tri. B**
Census	19657	44	91833	18688	59	88186
FEBRL	12477	6	22500	13948	6	22500
Cora	2431	16	6327	2437	16	6248

Table 4.3: ER-ER Graph Details

Following are the methods which can be used for generating graph embedding for ER-ER graph type.

1. **MTransE:** Multilingual Knowledge Graph Embedding for crosslingual knowledge alignment [34] was suggested in 2017 to link two knowledge graphs containing similar information but in different languages. This is a semi-supervised method where some true links between knowledge graphs are used to optimize embedding. Inter-lingual links (ILLs) set contains the pairs of same entities while triple-wise alignment (TWAs) set contains pairs of same triplets. MTransE defines following two components for training:

   (a) **Knowledge Model:** TransE is used to generate embedding for each knowledge graph separately. We consider two knowledge graphs $(G_(L_1))$ and $(G_(L_1))$, defined in two different languages L_1 and L_2. The loss function is defined as sum of TransE loss for both knowledge graph.

   $$S_K = \sum_{L \in (L_1, L_2)} \sum_{(h,t,r) \in G_L} ||h + r - t|| \tag{4.19}$$

   where (h, r, t) is a triplet in the knowledge graph (G_L).

   (b) **Alignment Model :** The objective of alignment model is to construct translations between L_1 and L_2. [34] suggested five different variants for this but most effective one was based on linear transformations. We maintain an entity transformation matrix (M_e) of size $n \times n$, where n is the dimension of the embedding space. For a given aligned entity pair (e, e') in ILLs set, we define the loss function as:

   $$S_A = \sum_{(e,e') \in ILL)} ||M_e e - e'|| \tag{4.20}$$

The objective function for MTransE to minimize is:

$$S = S_K + \alpha.S_A \tag{4.21}$$

where α is a positive hyper-parameter.

MTransE doesn't utilize negative sampling and only learn the embedding using the positive samples. It suggests to use kNN (k-Nearest Neighbours [35]) algorithm to search for entity alignments across the two knowledge graphs.

2. **ITransE:** Iterative TransE [36] aims to align entities across two heterogeneous knowledge graphs. It utilizes the pre-aligned entity pairs (ILL) to predict new aligned pairs. It has following three components of the loss function:

   (a) **Knowledge Embedding:** This component utilizes the margin-based energy functions as defined by TransE or PTransE. Each Knowledge graphs is projected to a separate embedding space. The loss function for a knowledge graph is defined as:

   $$K = \sum_{(h,r,t)} max(0, \gamma + g(h,r,t) - g(h',r',t') \tag{4.22}$$

   where (h', r', t') represents the negative sample for the triple and γ is the constant margin.

   (b) **Joint Embedding:** This component transforms the entity embedding from embedding space of first knowledge graph to the embedding space of the second knowledge graph. The aligned entity pairs are used to compute the loss for this component. This can be achieved by either keeping a translation vector ($r_{e_1 \to e_2}$) or a linear transformation matrix ($M_{e_1 \to e_2}$). The loss function can be defined as:

   $$J = \sum_{(e_1,e_2) \in ILL} \alpha ||M_{e_1 \to e_2} \boldsymbol{e_1} - \boldsymbol{e_2}|| \tag{4.23}$$

   Another model called parameter sharing, is also suggested which aims at keeping same embedding for aligned entities. Loss for parameter sharing model is defined as $J = 0$.

   (c) **Iterative Alignment:** This component aims to find new aligned entity pairs based on the distance between the embedding of the

two entities. The entity pair can be aligned using hard alignment (update embedding for both entities as their mean) or using soft alignment (maintain reliability scores for each entity pair). In case of hard alignment, the loss is defined as $I = 0$ and for soft alignment it is defined as:

$$I = \sum_{(e_1,e_2)\in NAE} R(e_1, e_2)(H(e_1, e_2) + H(e_2, e1)) \tag{4.24}$$

where NAE is the set of newly aligned entity pairs, $R(e_1, e_2)$ gives the reliability score for aligning e_1 and e_2. R can be defined as:

$$R(e_1, e_2) = \sigma(k(\theta - ||\boldsymbol{e_1} - \boldsymbol{e_2}||)) \tag{4.25}$$

where k is the hyper-parameter and θ is the threshold. $H(e_1, e_2)$ computes the loss on triples on replacing entities and is defined as:

$$H(e_1, e_2) = \sum_{(e_1,r,t)} g(e_2, r, t) + \sum_{(h,r,e_2)} g(h, r, e_1) \tag{4.26}$$

where g is the energy function as per TransE or PTransE.

The objective function for ITransE to minimize is:

$$Loss = K + \alpha J + \beta I \tag{4.27}$$

where α and β are positive hyper-parameters.

3. **JAPE:** Joint Attribute-Preserving Embedding (JAPE) [37] is a cross-lingual entity alignment method which jointly embeds two knowledge graphs in a unified vector space. The method utilizes EAR structures for each knowledge graph. It has following two components:

   (a) **Structure Embedding (SE):** This component of JAPE uses TransE model to generate embedding for both knowledge graphs in a unified vector space. Consider set T, consisting of all the triples from both knowledge graphs. For each triple $tr \in T$, we sample negative samples by randomly replacing head or tail entity denoted by tr'. Thus, the loss for structure embedding can be given by:

   $$O_{SE} = \sum_{tr\in T} \sum_{tr'\in T'_{tr}} (g(tr) - \alpha g(tr')) \tag{4.28}$$

   where α is a hyper-parameter and $g(tr)$ is the energy function as defined by TransE i.e. $g(tr) = g(h, t, r) = ||\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}||_{L_1/L_2}$.

(b) **Attribute Embedding (AE):** [37] suggests to only use the abstract data type for attribute values i.e. Integer, Double, Datetime and String. This component generates embedding for attributes such that the correlated attributes are close to each other. First, the similarity matrices are generated using the attribute embedding. The normalized average of attribute vectors is used to represent an entity i.e. $\boldsymbol{e} = \frac{\sum_{a \in A_e} \boldsymbol{a}}{|A_e|}$, where A_e is the set of attributes linked to the entity. Thus we create two matrices $\boldsymbol{E_1}$ for entities in knowledge graph 1 and $\boldsymbol{E_2}$ for entities in knowledge graph 2. The cosine distance between the entities is used to compute the inner similarity and cross-KG similarity. We get three similarity matrices i.e. $\boldsymbol{S_1} = \boldsymbol{E_1}\boldsymbol{E_1}^T$, $\boldsymbol{S_2} = \boldsymbol{E_2}\boldsymbol{E_2}^T$ and $\boldsymbol{S_{12}} = \boldsymbol{E_1}\boldsymbol{E_2}^T$. Using these similarity matrices, the objective function to be minimized is defined as:

$$\begin{aligned} O_{AE} = ||\boldsymbol{E_1} - \boldsymbol{S_{12}}\boldsymbol{E_2}|| + \\ \beta(||\boldsymbol{E_1} - \boldsymbol{S_1}\boldsymbol{E_1}|| + ||\boldsymbol{E_2} - \boldsymbol{S_2}\boldsymbol{E_2}||) \end{aligned} \tag{4.29}$$

where β is the hyper-parameter to balance cross-KG and inner similarities.

The objective function for JAPE to minimize is:

$$Loss = O_{SE} + \alpha O_{AE} \tag{4.30}$$

where α is a positive hyper-parameter.

4.4.4 More Graph Embedding methods

Apart from Knowledge graph embedding methods, there has been wide range of research in generating embedding for any generic graph. [38] presents methods to generate graph kernels for SVM classification model. The initial vectors are generated for each node using the Eigen vectors of the adjacency matrix. A graph is then represented as a bag of vectors representing its nodes and then two graphs are compared using Earth Mover's Distance [39]. This method performs well on structured small graphs (like enzymes) while it is not practical to build adjacency matrices and compute Eigen vectors for a large knowledge graph having millions of entities.

All the embedding methods defined above represents the entities to the low-dimensional euclidean space while [40] represents the embedding in a

hyperbolic space i.e Poincare ball model. This allows to embed the entities in lesser number of dimensions as required by other translation based methods.

Deep-Walk [41] and Node2Vec [42] methods learns the representation of a graph by generating random walks for each vertex or node. Both search algorithms Breadth First Search (BFS) and Depth First Search (DFS) are used to find neighbouring nodes. Skip-Gram model [43] is used to learn the representations from the random walks based on node co-occurrence.

[44], [45], [46] have also experimented with Graph Neural Networks and Graph Convolutions Networks. Aim here is classify a node based on given adjacency matrix for input sub-graph. These methods are used for community detection or image classification.

T-TransE [47] and HyTE [48] are two notable methods to generate embedding for a time-aware knowledge graphs in which relations or attributes about an individual may change with time. An extra time-dimension is added to each triplet to denote its temporal scope or valid time period. T-TransE uses a projection matrix to transform a relation vector to a different temporal scope. HyTE uses a different hyper-plane to embed the triplets for each temporal scope.

4.5 Experiments

We implemented various translation based algorithms for generating knowledge graph embedding. We used tensorflow [1] library to generate and optimize embedding for the three knowledge graph structures defined above. We referred an open-source collection of graph embedding methods called OpenKE [2], to implement following five methods for entity alignment:

1. TransE for ER graph model
2. TransH for ER graph model
3. KR-EAR for EAR graph model

[1] `https://www.tensorflow.org/install`
[2] `https://github.com/thunlp/OpenKE`

4. SEEA for EAR graph model

5. MTransE for ER-ER graph model

For all methods we used eight hyper-parameters - dimension of embedding vectors (n), batch size for training Stochastic Gradient Descent Optimizer (b), learning rate (L_r), margin (m), scale of regularization (λ), maximum number of epochs (M_{epoch}) and negative sampling rates (N_r, N_e). Additionally, SEEA had another hyper-parameter β and MTransE had α.

First, a Knowledge graph is generated for training data in each dataset. For experimenting with TransE and TransH, three sets are generated to represent the knowledge graph entity set, relation set and triples set.

Each record is given a unique id and added to entity set. All attribute labels become part of relation set. The unique values are also added entity set itself. The triples set contain triples of form (e1, e2, r), e1 and e2 belongs to entity set and r belongs to the relation set.

The knowledge graph embedding are represented by two sets entity embedding and relation embedding. Entity Embedding is a list of n-dimensional vectors having same size as entity set. Similarly, relation embedding is also a list of n-dimensional vectors having same size as relation set. For each pair we compute the matching score as the cosine distance between the embedding vectors for the both entities. The pairs with least matching score are selected as linked pairs.

For experimentation with KR-EAR and SEEA method, we further maintain three more sets to represent the knowledge graph attribute set, value set and attributional triples set. For SEEA, we repeat the record linkage process as above in iterations. In each epoch top-β matching pairs are added to the triples set as linked pairs. Using these new links the knowledge graph embedding are re-generated for the next iteration. Stopping criteria is considered as when the no more new linked pairs are found.

For MTransE, we maintain two ER graphs as six sets of entities, relations and triples. We also require to learn the projection matrix as well along with embedding vectors.

4.5.1 Statistical Analysis of Binary Classification

	F-Score	**Accuracy**	**Precision**	**Recall**
ECM classifier	0.98	0.99	0.98	0.99
Logistic Regression	0.98	0.99	0.98	0.98
TransE (ER)	0.21	0.87	0.19	0.25
TransH (ER)	0.17	0.81	0.12	0.26
KR-EAR (EAR)	0.18	0.79	0.13	0.30
SEEA (EAR)	0.08	0.91	0.07	0.10
MTransE (ER-ER)	0.09	0.31	0.05	0.93

Table 4.4: Results For Census Dataset

Table 4.4 provides the results for the census dataset. TransE has the highest F-Score of 0.21 only. TransH and KR-EAR has F-Scores of 0.17 and 0.18. MTransE and SEEA methods does not perform well the census dataset and require further fixation.

	F-Score	**Accuracy**	**Precision**	**Recall**
ECM classifier	0.98	0.99	0.98	0.99
Logistic Regression	0.99	0.99	0.98	0.99
TransE (ER)	0.58	0.95	0.70	0.49
TransH (ER)	0.60	0.95	0.72	0.52
KR-EAR (EAR)	0.13	0.42	0.07	0.65
SEEA (EAR)	0.49	0.91	0.44	0.57
MTransE (ER-ER)	0.09	0.06	0.05	0.99

Table 4.5: Results For FEBRL Dataset

Table 5.3 provides the results for the FEBRL dataset. TransH has the best F-Score of 0.6, while TransE has a F-Score of 0.58. Applying self-learning to KR-EAR i.e. using SEEA method improves the performance significantly. MTransE has the least F-Score and requires further fixation.

Table 4.6 provides the results for the CORA dataset. MTransE has the best F-Score of 0.71. SEEA was able to improve the results slightly from KR-EAR. TransE had best accuracy of 0.99.

The embedding methods provide highly precise results at low recall. In order to achieve high recall we had to increase the threshold applied to the cosine distance between two entities. A high threshold increases number of

	F-Score	**Accuracy**	**Precision**	**Recall**
ECM classifier	0.84	0.99	0.77	0.93
Logistic Regression	0.70	0.98	0.85	0.59
TransE (ER)	0.54	0.99	0.62	0.48
TransH (ER)	0.44	0.95	0.39	0.51
KR-EAR (EAR)	0.32	0.88	0.20	0.81
SEEA (EAR)	0.33	0.97	0.79	0.21
MTransE (ER-ER)	0.71	0.98	0.62	0.83

Table 4.6: Results For CORA Dataset

false positives and thus reduces the precision. At low threshold the recall is low but precision is high.

We noted that F-Score does not allow to compare the performance of embedding methods since our candidate space is large and slight change in threshold can have a significant impact on the F-Score.

4.5.2 Information Retrieval Analysis

After generating embedding for the knowledge graph, we generate the ranked list of closest entities for each entity. To avoid computing cosine distance between each entity pair, we only consider the candidate entity pairs. Here, the candidate pairs are considered as information retrieval queries. We consider the entity from dataset A as the query input and generates a ranked list of entities from dataset B which are closest to the input entity (in terms of cosine distance). The ranks of relevant entities (based on the given ground truth or true pairs) is recorded for computing mean reciprocal rank and mean average precision. We used $K = 1, 10$ to compute mean precision@K for the five embedding methods defined above.

Table 4.7 provides results for Census Dataset. TransE has the best MRR of 0.51. SEEA has lower MRR than KR-EAR i.e. it is learning incorrect patterns through self-learning.

Table 4.8 provides results for FEBRL dataset. TransH has the best MRR of 0.92.

	MP@1	**MP@10**	**MRR**	**MAP**
ECM classifier	0.99	0.1	0.99	0.99
Logistic Regression	0.99	0.1	0.99	0.99
TransE (ER)	0.35	0.09	0.51	0.51
TransH (ER)	0.27	0.09	0.44	0.42
KR-EAR (EAR)	0.29	0.08	0.46	0.46
SEEA (EAR)	0.20	0.09	0.36	0.34
MTransE (ER-ER)	0.28	0.08	0.45	0.45

Table 4.7: Information Retrieval Results for Census

	MP@1	**MP@10**	**MRR**	**MAP**
ECM classifier	0.99	0.10	0.99	0.99
Logistic Regression	0.99	0.10	0.99	0.99
TransE (ER)	0.69	0.10	0.79	0.79
TransH (ER)	0.89	0.10	0.92	0.92
KR-EAR (EAR)	0.37	0.09	0.54	0.54
SEEA (EAR)	0.36	0.09	0.52	0.52
MTransE (ER-ER)	0.28	0.08	0.45	0.45

Table 4.8: Information Retrieval Results for FEBRL

Table 4.9 provides results of entity alignment for Cora Dataset. Both TransE and TransH has the best MRR of 0.94.

We searched for optimal results by using hyper-parameters as dimension of embedding vectors(n) from (64, 128, 256), batch size(b) from (32, 128, 256, 1024), learning rate(L_r) from (0.1, 0.01), margin (m) from (1, 10), scale of regularization (λ) from (0.1, 0.01), maximum number of epochs (M_{epoch}) from (1000, 5000) and negative sampling rates (N_r, N_e) from ((1, 7), (4, 12)). Also for SEEA we choose β from (32, 64) and for MTransE we choose α from (5, 10).

Figure 4.7 depicts the distribution of cosine distance between the candidate entity pairs. We segregate the entity pairs in small buckets having width of 0.05. The first bar represents the entity pairs whose cosine distance is between [0.35, 0.40). We count the number of matching pairs (positive count) and number of non-matching pairs (negative count) for each bucket. We notice most matching pairs have a low cosine distance while most non-matching pairs have a high cosine distance.

	MP@1	MP@10	MRR	MAP
ECM classifier	0.96	0.56	0.97	0.89
Logistic Regression	0.94	0.79	0.95	0.87
TransE (ER)	0.92	0.52	0.94	0.61
TransH (ER)	0.93	0.53	0.94	0.62
KR-EAR (EAR)	0.88	0.51	0.90	0.58
SEEA (EAR)	0.76	0.51	0.81	0.59
MTransE (ER-ER)	0.23	0.24	0.36	0.26

Table 4.9: Information Retrieval Results for Cora

We noted that embedding based methods can used for linking two entities in a knowledge graph. Though they perform poorly than ECM and Logistic Regression but are capable of linking entities from different languages or domains. We do not compare actual strings represented by entities, thus we can have two similar strings (like 'josep' and 'joseph') very far apart in the vector space. On the other hand two different string (like 'josep' and 'NA') can be very close to each other. This results in high number of false negatives and false positives.

We look for errors in classification by analyzing 50 false positives and 50 false negatives for TransE method over census dataset.

1. **Full names are different:** We noted that all 50 false positives have different full names, still there are considered as linked. On the other hand 44 of the false negatives have same full name but still they have been classified incorrectly.

2. **Years of birth are different:** We noted that all 50 false negatives have different year of births while 26 false positives also have different year of births. Also, 42 of the false positives have either of the record pair to have year of birth set as null. Thus the embedding fails to learn the change in year of birth.

3. **Same Civil Status:** We noted that 40 of the false negatives have same civil status yet are classified incorrectly. Also, 27 of the false positives had different civil status yet classified as links.

4. **Same Relation:** We noted that 30 false negatives had the same relationship with the head of family but are classified as non-links while only 13 false positives have same relationship.

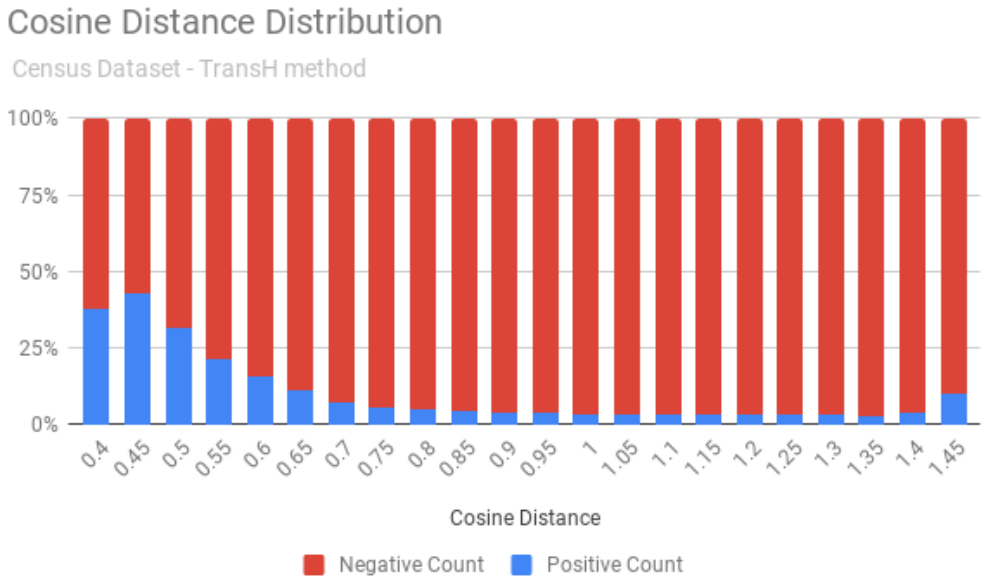

Figure 4.7: Cosine Distance Distribution

5. **Same Occupation:** We noted that 28 false negatives had the same occupation code but are classified as non-links while only 11 false positives have same occupation.

We note that entity alignment methods fails to acknowledge that records having similar attributes can be linked. Even a small error or misrepresentation in entity name can result in incorrect classification, since the embedding vector of similar values can be very distant. In order to avoid such errors, it is suggested that we merge entities which have small edit distance.

Chapter 5

Proposed Methods

We designed a few custom methods inspired from the methods described in previous chapters. We attempted to create novel methods using both record linkage and entity alignment principles.

We noticed that state of the art record linkage methods were not able to learn the valid / invalid evolution of the values. We expect a individual's civil status can change to married from single in subsequent census. Similarly, relationship with head of family and occupation are also expected to evolve with time. Though these methods were able to learn the noise / errors in data to produce satisfactory results.

On the other hand the state of the art methods for entity alignment failed to link records due to both noise in data and evolution in certain attributes. The cosine distance between the two entities does not provide enough information to link records correctly. Instead, we should explore the neighbourhood of each entity and collect more latent variables along with cosine distance between the two entities. This can help us reduce the number of false positives for entity alignment methods.

5.1 Extending Entity Alignment Methods

First, we explore a few methods which extend state of the art entity alignment methods towards record linkage.

5.1.1 Household Alignment using embedding

Household TransE (HTransE): For historical census records, it is important to link two households or families across census years in order to study the change in demographics. [1] suggested to construct bipartite graphs between two candidate households. Instead of classifying candidate pairs based on record similarity, we utilize the similarity among their neighbours as well. This helps us improve the precision of the model by reducing false positives.

First we define a cost matrix (C) for a pair of candidate household pairs (H_A, H_B). A single entry of cost matrix (c_{ij}) is computed as the cosine distance between the i^{th} individual in H_A and j^{th} individual in H_B. This cost matrix is minimized using the Hungarian Algorithm[49] to compute the cost of linking two household. The household pairs having cost less than the hyper-parameter (θ) is considered as the linked households.

This method can be applied for household alignment after generating graph embedding by any method. For census, we had an identifier for each household which is used to club the family members together. For Cora, we attempted to match the author groups for each scientific paper. For FEBRL, we create a group of individuals having same postcode. We estimate the probability of match for a household pair (or pair of record groups) by computing the cost of linking (CL) as:

$$CL(H_A, H_B) = \frac{\sum_{i \in H_A, j \in H_B} c_{ij}.x_{ij}}{|H_A|.|H_B|} \tag{5.1}$$

where c_{ij} represents the cost of linking i and j individual records and x_{ij} is a Boolean flag representing whether records (i, j) are linked or not, after applying Hungarian Algorithm[49] to the cost matrix (C). This problem of assigning links is also called linear sum assignment, perfect matching or complete matching in various contexts.

5.1.2 Logistic Regression over embedding

LogTransE/H: We extended the simple TransE and TransH methods by plugging a logistic regression model over the learned embedding. First, we trained embedding for a ER-ER knowledge graph by applying TransE or TransH to each isolated ER graph. Afterwards, instead of using the cosine distance as probability of match, we additionally trained a logistic model to learn the weights for each dimension. We generate our feature vectors by computing absolute distance for each dimension between two embedding vectors and then applying Gaussian function to estimate the similarity. Thus the feature vector has the same length as the dimension used for generating embedding. Thus for given embedding vectors $(\boldsymbol{a}, \boldsymbol{b})$, we estimate the probability of match as:

$$P(m = 1|(\boldsymbol{a}, \boldsymbol{b})) = \frac{1}{1 + e^{(\sum_{i \in [0,d)} g(\boldsymbol{a}[i], \boldsymbol{b}[i]))}} \tag{5.2}$$

$$\text{where, } g(a[i], b[i]) = w_0 + w_i.G(|\boldsymbol{a}[i] - \boldsymbol{b}[i]| \tag{5.3}$$

where G represents a decay function (e.g. Gaussian, Exponential or Linear), w_i represents the weights learned using Logistic Regression model, d is dimension of the embedding vectors and w_0 is the bias.

5.1.3 Learning evolution of entity values

Evolution MTransE (E-MTransE): We extended MTransE model by defining a Evolution loss between the values of a knowledge graph. For a given ER-ER graph, we apply joint learning to align entities. We expect the values to be either consistent across years or have a fixed rule by which it evolves across years e.g. (solter, casat, civil-status). We define a evolution vector (E_r) for each relation (r) and compute the evolution loss as:

$$S_E = \sum_{(e,e') \in ILL} ||\boldsymbol{e}\boldsymbol{M_e} + \boldsymbol{E_r} - \boldsymbol{e'}|| \tag{5.4}$$

where M_e is the projection matrix as defined for the alignment loss and (e, e') are the entity pairs from Inter-Lingual Links (ILL).

Thus for E-MTransE, we optimize the summation of following three losses:

1. Knowledge Model Loss (S_K): TransE is used to generate embedding for each ER knowledge graph.
2. Alignment Model Loss (S_A): Entities are projected from embedding space of first knowledge graph to the embedding space of second knowledge graph.
3. Evolution Model Loss (S_E): Translation based loss between two different values (one from each knowledge graph) associated by same relationship.

The objective function for E-MTransE is to minimize:

$$S = S_K + \alpha S_A + \beta S_E \tag{5.5}$$

where α and β are positive hyper-parameters.

5.1.4 Record Linkage Translation Embedding

RL-TransE Record-Linkage TransE utilizes the concepts from Fellegi and Sunter Framework to compute probability of match from graph embeddings. Instead of considering the just the cosine distance between embedding vectors of two entities, we also consider the distance between the embedding of its attributes. Consider a candidate entity pair (a, b), we denote the cosine distance between them as, $c(\boldsymbol{a}, \boldsymbol{b})$. The total distance($D$) between a and b is given by:

$$D(a, b) = c(\boldsymbol{a}, \boldsymbol{b}) + \sum_{i \in [1,K]} c(\boldsymbol{a_i}, \boldsymbol{b_i}) \tag{5.6}$$

where K is the number of attributes linked to each entity and (a_i, b_i) represents the attribute values. This method can be applied to align entities after generating embedding using any algorithm.

5.2 Record Linkage on Evolution Embedding

We define a custom knowledge graph structure to experiment with our custom Evolution based methods. We named this graph structure **VEG - Value Evolution Graph**. The graph G is represented as (V, R, T), where V is the set of all unique values like 1994, solter, esposa, etc, R refers to the set of relations like author, name, occupation, etc and T is the set of triples. Here, we don't create entities of each record, rather we only consider the evolution of values with time. Given a true record pair (A, B), we create a triplet $(A[r], B[r], r)$ for each relation r e.g. (solter, casat, civil-status) which represents that the individual's civil-status might change to casat (married) from solter (single) in subsequent census years. Figure 5.1 represents the VEG for Census dataset built from the training data. This method allows us to learn the various transformation rules from the training data without any domain knowledge. We can learn the representations of the values and evolution vectors using TransE, TransH or other similar methods.

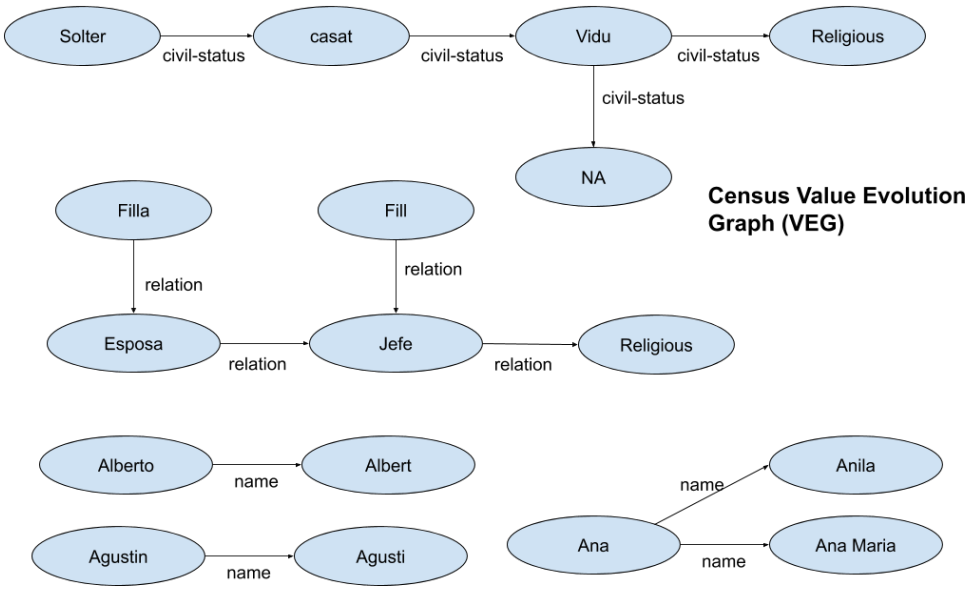

Figure 5.1: VEG model for Census dataset

For this graph structure, we first build a relation to value map. For each relation (like name, author, occupation, etc), we collect all the possible values. Then we create evolutionary triples from the training data by consid-

ering a linked record pair (A, B) and corresponding value pairs $(A[r], B[r])$ related to (A, B) by a relation r. Thus we get triplets of form $(A[r], B[r], r)$ which represents evolution of value $A[r]$ to $B[r]$ in subsequent census years. Table 5.1 details the count of Values, Relations and Triples sets for training, testing and validation purpose.

Data set	Census	FEBRL	CORA
$\#V$	4,289	7,618	1,748
$\#R$	8	5	8
#Train Triples	10,643	1,112	26,123
#Validation Triples	12,858	199	2,895
#Test Triples	11235	622	11,923

Table 5.1: Statistics of the data sets for the VEG representation.

We create n-dimensional vectors for each value and relation. To classify a record pair (C, D) as linked or not, we compute the sum of distance between corresponding values of the two records. If the values are same the distance is considered as 0, otherwise it is computed as $g(v_1, v_2, r) = ||\boldsymbol{v_1} + \boldsymbol{E_r} - \boldsymbol{v_2}||$, where v_1 is value related to a and v_2 is value related to b and E_r is the evolution vector for the relation r.

We train a stochastic gradient descent optimizer to minimize the distance between valid evolutionary value pairs and maximize the distance between invalid value evolution. We define the hinge loss as:

$$Loss = \sum_{(v_1,v_2,r)\in T} \max(0, \gamma + g(v_1, v_2, r) - g(v_1', v_2, r)) \tag{5.7}$$

where γ is the margin and v_1' represents the negative sample by replacing v_1.

After learning embedding vectors for all values, we use **RL-TransE** to compute the distance between two entities. We consider the sum of distances between associated values between record pairs as the probability of match. RL-TransE utilizes the concepts from Fellegi and Sunter Framework to compute probability of match from graph embeddings. Consider a candidate entity pair (a, b), we denote the cosine distance between them as, $c(\boldsymbol{a}, \boldsymbol{b})$. The total distance (D) between a and b is given by:

$$D(a,b) = \sum_{i \in [1,K]} \begin{cases} \text{if } a_i == b_i, & 0 \\ \text{else if } a_i \in V \wedge b_i \in V, & c(\boldsymbol{a_i} + \boldsymbol{E_r}, \boldsymbol{b_i}) \\ \text{else}, & 1 \end{cases} \tag{5.8}$$

where a_i represents the i^{th} values related to entity a and K is the total number of values linked to each entity.

While testing, if we encounter an unknown value for which the embedding vector does not exist, we explicitly consider it as a mismatch. We assign $c(a_i, b_i) = 1$, if either of a_i or b_i is an unknown value.

5.3 Value Evolution Embedding for a Record

VEER generates evolutionary embedding vectors for values associated for each record such that a given record pair can be classified as same or different. VEER is a supervised record linkage method for dynamic data, capable to learning possible evolution in values or literals.

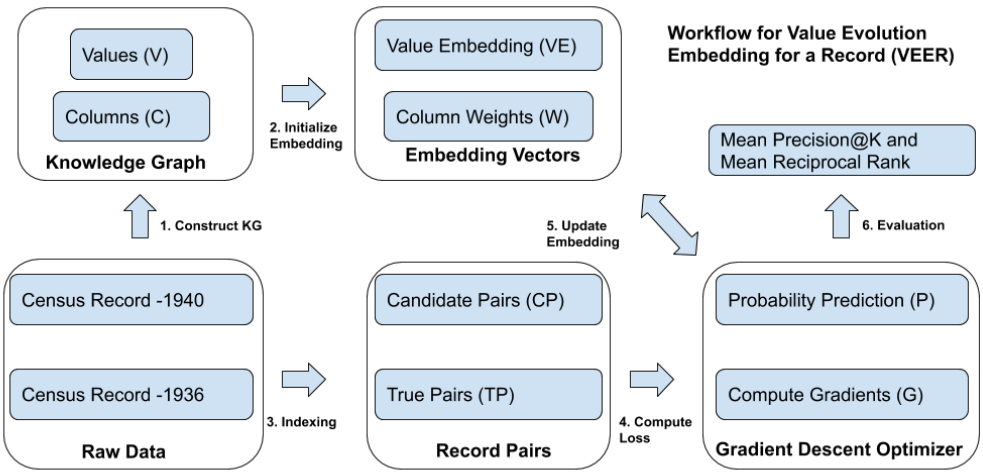

Figure 5.2: VEER Workflow

Figure 5.2 depicts the workflow for VEER method. Initially, we collect all the possible values mentioned the dataset and assign an embedding vector

for each value. We expect embedding vectors should converge for the group of values where evolution among each other is possible. We learn to classify classify a value evolution as invalid or valid based on the candidate pairs provided in the training dataset.

We compute the energy(g) of a record pair (a, b) instead of computing energy of a triplet. We use a different weight (w) for each column (c) and define a energy function as:

$$g(a,b) = \sigma(\frac{\sum_{c \in C} w[c] * ||\boldsymbol{a}[\boldsymbol{c}] - \boldsymbol{b}[\boldsymbol{c}]||}{|C|}) \tag{5.9}$$

where C is the list of column names or attributes associated with each record. To learn the embedding vectors for each value and weights for each column, we minimize following hinge loss:

$$Loss = \max(0, \text{margin} + \sum g(a,b) \cdot t(a,b)) \tag{5.10}$$

where $t(a, b)$ is $+1$ for true record pairs and -1 for incorrect record pairs.

To avoid unknown values, we first collect all values mentioned in the whole dataset including test partition. Thus, even for unknown values we have the default embedding vector.

5.4 Experiments

We implemented the methods defined above using tensorflow library and evaluated their performance compared with methods defined in previous chapters. We present results for following five configurations:

1. HTransE for ER graph model
2. LogTransH for ER graph model
3. E-MTransE for ER-ER graph model
4. RL-TransE for VEG graph model

5. VEER for VEG graph model

We had eight hyper-parameters - dimension of embedding vectors (n), batch size for training (b), learning rate (L_r), margin (m), scale of regularization (λ), maximum number of epochs (M_{epoch}) and negative sampling rates (N_r, N_e).

For RLTransE, we searched for optimal hyper-parameters as n from (32, 64, 128), b from (32, 64, 128), L_r from (0.1), m from (1, 0.1), λ from (0.1), M_epoch from (1000, 5000) and (N_r, N_e) from ((7,1)). For VEER, we searched for optimal hyper-parameters as n from (16, 32, 64), b from (32, 64), L_r from (0.1), m from (1, 0.1), λ from (0.1) and M_epoch from (50, 100, 500). Details of optimal hyper-parameters found after grid search is provided in Appendix A.

For our proposed methods, we model our training dataset as a knowledge graph representation as required by the algorithm. For HTransE and LogTransH, we utilize the ER knowledge graph and for E-MTransE we utilize ER-ER knowledge graph which have been constructed earlier. For RLTransE and VEER we generate a VEG from the training data of each dataset. We optimize embedding vectors by minimizing corresponding loss function using Stochastic Gradient Descent Optimizer. Finally we compute the performance metrics using both statistical analysis of binary classification (e.g. F-Score) and information retrieval analysis (e.g. Mean Precision@1).

5.4.1 Statistical Analysis of Binary Classification

Table 5.2 provides the results for the census dataset. HTransE fails to improve the performance as we notice that there are significant changes in the household structures in subsequent census years. LogTransH also does not provide much improvement since it uses the same embedding vectors generated by TransH method. E-MTransE increases the complexity of MTransE and thus requires more training data or epochs to improve the performance. RLTransE and VEER provides close to state of the art results. With these methods we are able to learn the evolution embedding for the labels used in the dataset.

We found the optimal hyper-parameters for VEER as $\lambda = 0.1$, $L_r = 0.1$,

	F-Score	**Accuracy**	**Precision**	**Recall**
ECM classifier	0.98	0.99	0.98	0.99
Logistic Regression	0.98	0.99	0.98	0.98
TransE (ER)	0.21	0.87	0.19	0.25
KR-EAR (EAR)	0.18	0.79	0.13	0.30
HTransE (ER)	0.01	0.96	0.22	0.01
LogTransH (ER)	0.15	0.72	0.11	0.23
E-MTransE (ERER)	0.08	0.53	0.05	0.58
RLTransE (VEG)	0.95	0.99	0.97	0.92
VEER (VEG)	0.95	0.99	0.98	0.92

Table 5.2: Results For Census Dataset

$n = 16$, $M_{epochs} = 100$, $m = 0.1$ and $b = 32$ with the F-Score of 0.95. Also for RLTransE the optimal hyper-parameters were found as $\lambda = 0.1$, $L_r = 0.1$, $n = 64$, $M_{epochs} = 5000$, $m = 1$, $N_r = 1$, $N_e = 7$ and $b = 64$ with the F-Score of 0.95.

	F-Score	**Accuracy**	**Precision**	**Recall**
ECM classifier	0.98	0.99	0.98	0.99
Logistic Regression	0.99	0.99	0.98	0.99
TransE (ER)	0.58	0.95	0.70	0.49
KR-EAR (EAR)	0.13	0.42	0.07	0.65
HTransE (ER)	0.01	0.93	0.99	0.11
LogTransH (ER)	0.09	0.54	0.05	0.46
E-MTransE (ERER)	0.08	0.45	0.04	0.48
RLTransE (VEG)	0.98	0.99	0.98	0.97
VEER (VEG)	0.96	0.99	0.97	0.95

Table 5.3: Results For FEBRL Dataset

Table 5.3 provides the results for the FEBRL dataset. RLTransE and VEER provides the best F-scores of 0.96. HTransE fails since large number of people live within a postcode and also error in postcode is possible. LogTransH and E-MTransE fails to learn the embedding, possibly due to their complexity. RLTransE and VEER are capable of learning evolution in the various values.

We found the optimal hyper-parameters for VEER as $\lambda = 0.1$, $L_r = 0.1$, $n = 16$, $M_{epochs} = 500$, $m = 0.1$ and $b = 64$ with the F-Score of 0.96. Also for RLTransE the optimal hyper-parameters were found as $\lambda = 0.1$, $L_r = 0.1$,

	F-Score	Accuracy	Precision	Recall
ECM classifier	0.84	0.99	0.77	0.93
Logistic Regression	0.70	0.98	0.85	0.59
TransE (ER)	0.54	0.99	0.62	0.48
KR-EAR (EAR)	0.32	0.88	0.20	0.81
HTransE (ER)	0.43	0.95	0.34	0.57
LogTransH (ER)	0.05	0.53	0.03	0.52
E-MTransE (ERER)	0.34	0.96	0.28	0.41
RLTransE (VEG)	0.57	0.97	0.70	0.48
VEER (VEG)	0.85	0.99	0.89	0.81

Table 5.4: Results For CORA Dataset

$n = 128$, $M_{epochs} = 5000$, $m = 0.1$, $N_r = 1$, $N_e = 7$ and $b = 32$ with the F-Score of 0.98.

Table 5.4 provides the results for the CORA dataset. VEER provides the best F-scores of 0.72. HTransE is able to align the author groups successfully. LogTransH fails to learn the embedding vectors correctly. E-MTransE and RLTransE provides the similar results of F-Score 0.33. VEER is able to learn the possible evolution in values.

We found the optimal hyper-parameters for VEER as $\lambda = 0.1$, $L_r = 0.1$, $n = 16$, $M_{epochs} = 30$, $m = 0.1$ and $b = 512$ with F-Score of 0.72. For RLTransE, the optimal hyper-parameters were $\lambda = 0.1$, $L_r = 0.1$, $n = 32$, $M_{epochs} = 1000$, $m = 1$, $N_r = 1$, $N_e = 7$ and $b = 32$ with F-Score of 0.57.

5.4.2 Information Retrieval Analysis

After generating the embedding vectors, we evaluate the performance of the entity alignment task by computing the cosine distance between the candidate entity pairs. We create Q queries as the set of first entities in each entity pair *(a, b)*. For each unique query, we generate the list of entities closet to entity a based on cosine distance between their embedding and compute four metrics - Mean Precision@1, Mean Precision@10, Mean Reciprocal Rank (MRR) and Mean Average Precision (MAP).

Table 5.5 provides the results for the Census dataset. VEER and RL-

	MP@1	MP@10	MRR	MAP
ECM classifier	0.99	0.10	0.99	0.99
Logistic Regression	0.99	0.10	0.99	0.99
TransE (ER)	0.35	0.09	0.51	0.51
KR-EAR (EAR)	0.29	0.08	0.46	0.46
HTransE (ER)	0.01	0.01	0.01	0.36
LogTransH (ER)	0.25	0.11	0.41	0.4
E-MTransE (ERER)	0.28	0.08	0.45	0.45
RLTransE (VEG)	0.99	0.15	0.99	0.99
VEER (VEG)	0.99	0.15	0.99	0.99

Table 5.5: Information Retrieval Results for Census

TransE provides the best MRR of 0.99. HTransE fails to learn the household structures. LogTransH and E-MTransE fails to learn the embedding vectors correctly. VEER and RLTransE are able to learn the possible evolution in values.

	MP@1	MP@10	MRR	MAP
ECM classifier	0.99	0.10	0.99	0.99
Logistic Regression	0.99	0.10	0.99	0.99
TransE (ER)	0.69	0.10	0.79	0.79
KR-EAR (EAR)	0.37	0.09	0.54	0.54
HTransE (ER)	0.01	0.01	0.01	0.42
LogTransH (ER)	0.48	0.10	0.65	0.65
E-MTransE (ERER)	0.25	0.08	0.43	0.43
RLTransE (VEG)	0.99	0.10	0.99	0.99
VEER (VEG)	0.98	0.10	0.98	0.98

Table 5.6: Information Retrieval Results for FEBRL

Table 5.6 provides the results for the FEBRL dataset. RLTransE provides the best MRR of 0.99. HTransE fails since large number of people live within a postcode and also error in postcode is possible. LogTransH and E-MTransE fails to learn the embedding vectors correctly. VEER and RLTransE are able to learn the possible evolution in values.

Table 5.7 provides the results for the CORA dataset. VEER provides the best MRR of 0.85. HTransE is able to align the author groups successfully. LogTransH and E-MTransE fails to learn the embedding vectors correctly.RLTransE and VEER are able to learn the possible evolution in

	MP@1	MP@10	MRR	MAP
ECM classifier	0.96	0.56	0.97	0.89
Logistic Regression	0.94	0.79	0.95	0.87
TransE (ER)	0.92	0.52	0.94	0.61
KR-EAR (EAR)	0.88	0.51	0.90	0.58
HTransE (ER)	0.85	0.43	0.88	0.51
LogTransH (ER)	0.05	0.06	0.13	0.07
E-MTransE (ERER)	0.12	0.13	0.20	0.15
RLTransE (VEG)	0.80	0.43	0.84	0.64
VEER (VEG)	0.91	0.54	0.92	0.86

Table 5.7: Information Retrieval Results for Cora

values.

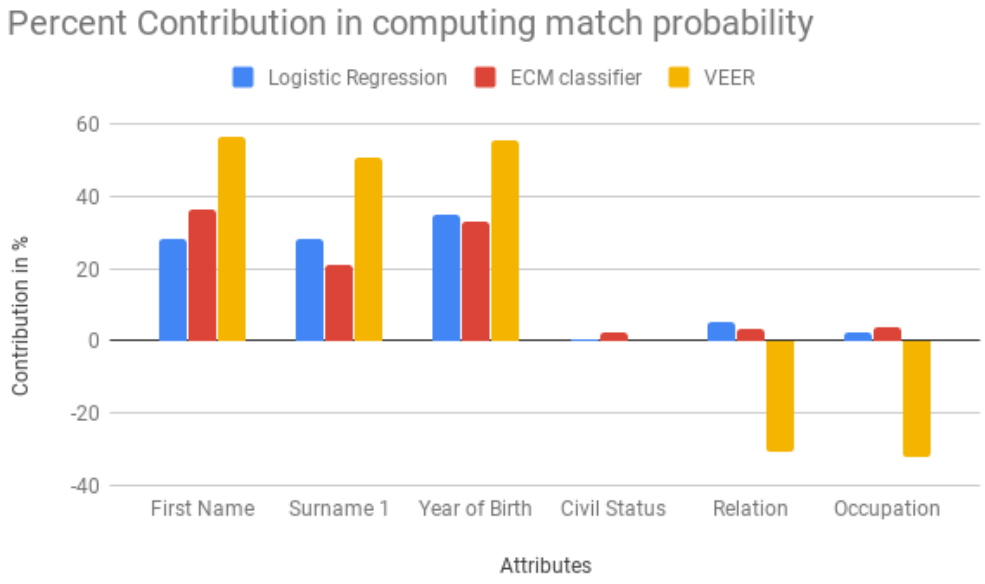

Figure 5.3: Percent Distribution

Figure 5.3 depicts the comparison between the weights learned by different methods. We notice that VEER is capable of penalizing the probability if a valid evolution is found.

Next, we analyze the errors in binary classification of Census data. We had 358 false positives for RLTransE and 158 false positives for VEER. Fol-

lowing are the various reasons for incorrect classification:

1. **Same Full Names:** We had 31 false positives with same full name for RLTransE and only 15 for VEER. Over 90% of false positives have different full names across census years.

2. **Same Years of Birth:** We had 202 false positives with same years of birth for RLTransE and 147 for VEER. For VEER only 11 false positives had different years of birth.

3. **Same Civil Status:** We had 269 false positives with same civil status for RLTransE and 71 for VEER. We had about 56% false positives for VEER due to linking records with different civil status.

4. **Same Occupation:** We had 340 false positives with same occupation for RLTransE and only 15 for VEER. Over 90% false positives for VEER have different occupations.

5. **Same Relationship with head:** We had 205 false positives with same relationship for RLTransE and 18 for VEER. About 90% false positives for VEER have different realtionship across years.

We had 695 false negatives for RLTransE and 748 false negatives for VEER.Following are the various reasons for incorrect classification:

1. **Full names are different:** We had 275 false negatives with different full names for RLTransE and 347 for VEER. Over 50% false negatives have same full names.

2. **Years of birth are different:** We had 17 false negatives with different for RLTransE and only 14 for VEER. Over 90% false negatives have same years of births.

3. **Different Civil Status:** We had 183 false negatives with different civil status for RLTransE and 97 for VEER. Over 70% false negatives have the same civil status.

4. **Different Occupation:** We had 658 false negatives with different occupations for RLTranE and only 184 for VEER. About 75% false negatives for VEER had same occupation across years.

5. **Different Relation:** We had 493 false negatives with different realtion for RLTransE and only 146 for VEER. About 80% false negatives for VEER have same relationship with head.

We notice that RLTransE and VEER are able to provide results which are close to the results of state of the art methods and are able to learn valid value evolution for attributes.

Chapter 6

WERL and MERL

6.1 Weighted Embedding based Record Linkage

WERL extends the graph embedding methods using evolution principle. Instead of computing the cosine distance between the embedding vectors of two entities, we compute the weighted similarity between the values linked with each entity. First, we generate knowledge graph embedding using any methods described in previous chapter. The estimate the energy of the record pair (a, b) as:

$$g(a,b) = \sigma(\sum_{(a,b)\in S}\sum_{c\in C} w[c] \times B(a[c], b[c]) \times ||\boldsymbol{a[c]} + \boldsymbol{E_c} - \boldsymbol{b[c]}||) \tag{6.1}$$

where S is the set of candidate pairs, C is the set of columns, E_c is the evolution vector and $a[c]$ represents the value associated with a. $B(a[c], b[c])$ represents a binary flag which is set only if both values are different. We train weights(w) for each attribute by minimizing following loss definition:

$$Loss = \max(0, \text{margin} + \sum g(a,b) \cdot t(a,b)) \tag{6.2}$$

where $t(a, b)$ is $+1$ for true record pairs and -1 for incorrect record pairs.

E_r is only defined for RL-TransE, so for other methods we initialize E_r as a n-dimensional zero vector for each relation.

We also implemented another variant of WERL by fixing the weights as 1 and named this variant as **MERL (Mean embedding based Record Linkage)**.

6.2 Results

Table 6.1 provides the F-Score for WERL method over three datasets. We used four knowledge graph embedding methods to generate the initial embedding vectors - TransE and TransH for ER graph, RLTransE and VEER for VEG graph. We note that the best F-score for census is given by VEER i.e. 0.81. For Cora and Febrl, TransE had the best F-Score of 0.75 and 0.33.

	TransE	**TransH**	**RLTransE**	**VEER**
Census	0.78	0.77	0.62	0.81
FEBRL	0.75	0.64	0.54	0.54
Cora	0.33	0.33	0.07	0.32

Table 6.1: F-Score for WERL method

Table 6.2 provides F-Score for MERL method over three datasets. We note that the best F-score for census is given by TransE i.e. 0.66. For Febrl, TransE had the best F-Score of 0.78 and for cora the best F-score of 0.33 is given by VEER.

	TransE	**TransH**	**RLTransE**	**VEER**
Census	0.69	0.15	0.41	0.30
FEBRL	0.78	0.19	0.61	0.22
Cora	0.24	0.11	0.07	0.33

Table 6.2: F-Score for MERL method

We note the using weights can improve the performance of linkage in most cases.

Chapter 7

Conclusion

The problem of record linkage can be extremely challenging when the dataset size is huge. Number of candidate pairs increases exponentially with the increase of number of records. Also, records from different datasets can have different attributes attached to it which makes it difficult to design a model and select features.

To avoid any false positives, active learning can be used. The linked pairs can be forced to be reviewed manually before considering it as the true link. The manual feedback can be used to update the model and improve performance overtime.

Negative Sampling is an expensive step based on the size of knowledge graph. The performance of the embedding based methods is highly dependent on quality of negative samples. If a true link is fed in the model as a negative sample, the entity alignment can result in low precision.

Embedding based methods can also be improved by experimenting with various Knowledge Graph structures, also called ontology. In case we have a simple graph with mostly one-to-one relations, TransE can perform well but if we have many-to-many and one-to-many relations in abundance, we should experiment with other embedding based methods and/or re-examine the ontology in use.

We should also avoid keeping duplicate or similar entities. We should aim to reduce the size of entity set by merging multiple entities with ideally

represent the same concept e.g. Spain and Espaa should not be stored as two entities since they represent the same country. This allows the embedding methods to encode the structure well.

We have successfully developed two knowledge-graph based methods for record linkage or entity alignment: RL-TransE and VEER. These methods are capable of learning the evolution in values / labels across time or language.

7.1 Additional Tests

Additionally, we tested our methods on one more census dataset from *Santa Coloma de Cervello* and *Castellvi de Rosanes* towns of Barcelona. The dataset had 8631 records with similar attributes as were provided for *Sant Feliu* town. We trained our models on training partition of *Sant Feliu* town and then tested on this new dataset.

We partition the dataset as Dataset A and B with 4815 and 3816 records. We considered records from census years 1866, 1924, 1936 and 1945 as dataset A. For dataset B, we selected the records from census years 1901, 1930, 1940 and 1950. We apply blocking indexing on the second surname field to yield 83264 candidate pairs, out of which only 6407 are true links.

We applied ECM Classifier, Logistic Regression, RLTransE and VEER methods on this dataset. The results are provided in table 7.1. RLTransE provides the best F-Score of 0.72. ECM Classifier and Logistic Regression provides the best MRR of 0.99. We used the same optimal hyper-parameters as computed earlier.

7.2 Future Work

In this article, we have focused on linking two census records pertaining to a single individual across multiple years. Instead we can also attempt linking two census records pertaining to a single household. Then single record can contain information about the whole family and the family structure can be

	MP@1	**MP@10**	**MRR**	**MAP**
ECM	0.98	0.19	0.99	0.99
Logistic	0.98	0.19	0.99	0.99
RLTransE	0.97	0.19	0.98	0.97
VEER	0.92	0.18	0.95	0.92
	Accuracy	**Precision**	**Recall**	**F-Score**
ECM	0.96	0.94	0.52	0.67
Logistic	0.96	0.97	0.52	0.67
RLTransE	0.96	0.76	0.69	0.72
VEER	0.95	0.74	0.61	0.67

Table 7.1: Additional Results for Santa Coloma and Castellvi towns

utilized for linking purposes.

We shall develop **Chain Translation Embedding (CTransE)** method to create a chain of individuals in a household based on their generation gap with each other. Thus, we attempt to converge the embedding vectors of the siblings (same generation gap). To find parents of an individual we apply translation vector once and to find grand-parents we can apply the translation vector twice.

In future, we shall attempt to link records from two separate datasets i.e. Census Dataset and Marriage Dataset. Our fellow researchers have digitalized these two historical datasets and we intent to create a complete family tree along with finding key events in familys history i.e. birth, death, marriage, relocation, change in occupations and more. We aim to publish our novel algorithms based on knowledge graph embedding in reputed journals.

Acknowledgment

I would like to thank Dr. Oriol Ramos Terrades from Computer Vision Center (CVC) for supervising the research work and providing relevant resources. I would also like to thank Centre for Demographic Studies (CED) for providing access to the census dataset. Finally, I thank fellow researchers in Document Analysis Group (DAG) for all the support. This work is partially funded under Spanish grant RTI2018-095645-B-C21, Xarxes project, RecerCaixa, Obra Social la Caixa and ACUP. The Titan V used for this research was donated by the NVIDIA Corporation.

Bibliography

[1] Z. Fu, P. Christen, and J. Zhou, "Lncs, a graph matching method for historical census household linkage," in *Advances in Knowledge Discovery and Data Mining.*, vol. 8843, 2014.

[2] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Transactions on Knowledge & Data Engineering*, vol. 19, pp. 1–16, 01 2007. [Online]. Available: doi.ieeecomputersociety.org/10.1109/TKDE.2007.9

[3] F. Kistermann, "Hollerith punched card system development (1905-1913)," *Annals of the History of Computing, IEEE*, vol. 27, pp. 56 – 66, 02 2005.

[4] U. Nations, *Principles and Recommendations for Population and Housing Censuses, Revision 3.* United Nations, 2017. [Online]. Available: https://www.un-ilibrary.org/content/publication/bb3ea73e-en

[5] P. Ravikumar and W. W. Cohen, "A hierarchical graphical model for record linkage," in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, ser. UAI '04. Arlington, Virginia, United States: AUAI Press, 2004, pp. 454–461. [Online]. Available: http://dl.acm.org/citation.cfm?id=1036843.1036898

[6] A. Monge and C. Elkan, "An efficient domain-independent algorithm for detecting approximately duplicate database records," 1997.

[7] V. Raman and J. Hellerstein, "Potters wheel: An interactive data cleaning system," *VLDB Journal*, vol. 2, no. Special Issue 3-4, pp. 381–390, 2001.

[8] I. P. Fellegi and A. B. Sunter, "A theory for record linkage," *Journal of the American Statistical Association*, vol. 64, no. 328, pp. 1183–1210, 1969. [Online]. Available: https://amstat.tandfonline.com/doi/abs/10.1080/01621459.1969.10501049

[9] M. A. Jaro, "Probabilistic linkage of large public health data files," *Statistics in Medicine*, vol. 14, pp. 491 – 498, 03 1995.

[10] W. Winkler, "The state of record linkage and current research problems," *Statist. Med.*, vol. 14, 10 1999.

[11] E. S. Ristad and P. N. Yianilos, "Learning string-edit distance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 5, pp. 522–532, May 1998. [Online]. Available: http://dx.doi.org/10.1109/34.682181

[12] S. Tejada, C. A. Knoblock, and S. Minton, "Learning object identification rules for information integration," *Inf. Syst.*, vol. 26, no. 8, pp. 607–633, Dec. 2001. [Online]. Available: http://dx.doi.org/10.1016/S0306-4379(01)00042-4

[13] A. Nikolov, M. d'Aquin, and E. Motta, "Unsupervised learning of link discovery configuration," in *Proceedings of the 9th International Conference on The Semantic Web: Research and Applications*, ser. ESWC'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 119–133. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30284-8_15

[14] F. M. Suchanek, S. Abiteboul, and P. Senellart, "PARIS: probabilistic alignment of relations, instances, and schema," *CoRR*, vol. abs/1111.7164, 2011. [Online]. Available: http://arxiv.org/abs/1111.7164

[15] S. Guan, X. Jin, Y. Jia, Y. Wang, H. Shen, and X. Cheng, "Self-learning and embedding based entity alignment," in *2017 IEEE International Conference on Big Knowledge (ICBK)*, Aug 2017, pp. 33–40.

[16] Y. Hao, Y. Zhang, S. He, K. Liu, and J. Zhao, "A joint embedding method for entity alignment of knowledge bases," in *Knowledge Graph and Semantic Computing: Semantic, Knowledge, and Linked Big Data*, H. Chen, H. Ji, L. Sun, H. Wang, T. Qian, and T. Ruan, Eds. Singapore: Springer Singapore, 2016, pp. 3–14.

[17] U. Draisbach and F. Naumann, "Dude: The duplicate detection toolkit," *ACM - VLDB*, 2010.

[18] P. Christen, "Febrl - an open source data cleaning, deduplication and record linkage system with a graphical user interface," in *KDD*, 08 2008, pp. 1065–1068.

[19] ——, *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Berlin: Springer, 2012.

[20] R. C. Robert, "Index," United States of America Patent US1 261 167A, 04, 1918.

[21] T. Churches, P. Christen, K. Lim, and J. Xi Zhu, "Preparation of name and address data for record linkage using hidden markov models tim churches," *BMC Med Inform Decis Mak*, vol. 16, 07 2004.

[22] J. Neyman and E. S. Pearson, "On the Problem of the Most Efficient Tests of Statistical Hypotheses," *Philosophical Transactions of the Royal Society of London Series A*, vol. 231, pp. 289–337, 1933.

[23] X.-L. Meng and D. B. Rubin, "Maximum likelihood estimation via the ecm algorithm: A general framework," *Biometrika*, vol. 80, no. 2, pp. 267–278, 1993. [Online]. Available: http://www.jstor.org/stable/2337198

[24] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2787–2795. [Online]. Available: http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf

[25] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes." in *AAAI*, 2014, pp. 1112–1119.

[26] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. AAAI Press, 2015, pp. 2181–2187. [Online]. Available: http://dl.acm.org/citation.cfm?id=2886521.2886624

[27] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, 01 2010.

[28] M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11. USA: Omnipress, 2011, pp. 809–816. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104482.3104584

[29] M. Nickel, L. Rosasco, and T. Poggio, "Holographic embeddings of knowledge graphs," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, pp. 1955–1961. [Online]. Available: http://dl.acm.org/citation.cfm?id=3016100.3016172

[30] T. A. Plate, "Holographic reduced representations," *Trans. Neur. Netw.*, vol. 6, no. 3, pp. 623–641, May 1995. [Online]. Available: http://dx.doi.org/10.1109/72.377968

[31] Y. Lin, Z. Liu, H. Luan, M. Sun, S. Rao, and S. Liu, "Modeling relation paths for representation learning of knowledge bases," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2015, pp. 705–714. [Online]. Available: http://aclweb.org/anthology/D15-1082

[32] Z. Sun, W. Hu, Q. Zhang, and Y. Qu, "Bootstrapping entity alignment with knowledge graph embedding," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 4396–4402. [Online]. Available: https://doi.org/10.24963/ijcai.2018/611

[33] Y. Lin, Z. Liu, and M. Sun, "Knowledge representation learning with entities, attributes and relations," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, ser. IJCAI'16. AAAI Press, 2016, pp. 2866–2872. [Online]. Available: http://dl.acm.org/citation.cfm?id=3060832.3061022

[34] M. Chen, Y. Tian, M. Yang, and C. Zaniolo, "Multilingual knowledge graph embeddings for cross-lingual knowledge alignment," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[35] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[36] H. Zhu, R. Xie, Z. Liu, and M. Sun, "Iterative entity alignment via joint knowledge embeddings," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 4258–4264. [Online]. Available: https://doi.org/10.24963/ijcai.2017/595

[37] Z. Sun, W. Hu, and C. Li, "Cross-lingual entity alignment via joint attribute-preserving embedding," in *The Semantic Web – ISWC 2017*, C. d'Amato, M. Fernandez, V. Tamma, F. Lecue, P. Cudré-Mauroux, J. Sequeda, C. Lange, and J. Heflin, Eds. Cham: Springer International Publishing, 2017, pp. 628–644.

[38] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Matching node embeddings for graph similarity," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, pp. 2429–2435. [Online]. Available: http://dl.acm.org/citation.cfm?id=3298483.3298589

[39] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International Journal of Computer Vision*, vol. 40, no. 2, pp. 99–121, Nov 2000. [Online]. Available: https://doi.org/10.1023/A:1026543900054

[40] M. Nickel and D. Kiela, "Poincare embeddings for learning hierarchical representations," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 6338–6347. [Online]. Available: http://papers.nips.cc/paper/7213-poincare-embeddings-for-learning-hierarchical-representations.pdf

[41] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14. New York, NY, USA: ACM, 2014, pp. 701–710. [Online]. Available: http://doi.acm.org/10.1145/2623330.2623732

[42] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[43] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the 26th International Conference on*

Neural Information Processing Systems - Volume 2, ser. NIPS'13. USA: Curran Associates Inc., 2013, pp. 3111–3119. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999792.2999959

[44] V. Garcia and J. Bruna, "Few-shot learning with graph neural networks," in *ICLR*, 2018.

[45] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017, pp. 1–10.

[46] Z. Chen, L. Li, and J. Bruna, "Supervised community detection with line graph neural networks," *International Conference on Learning Representations (ICLR)*, 2018.

[47] T. Jiang, T. Liu, T. Ge, L. Sha, S. Li, B. Chang, and Z. Sui, "Encoding temporal information for time-aware link prediction," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2350–2354. [Online]. Available: https://www.aclweb.org/anthology/D16-1260

[48] S. S. Dasgupta, S. N. Ray, and P. Talukdar, "Hyte: Hyperplane-based temporally aware knowledge graph embedding," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 2001–2011. [Online]. Available: https://www.aclweb.org/anthology/D18-1225

[49] H. W. Kuhn and B. Yaw, "The hungarian method for the assignment problem," *Naval Res. Logist. Quart*, pp. 83–97, 1955.

Appendix A

Optimal Hyper-Parameters

Method	Dataset	F-Score	n	b	L_r	m	λ	M_{epochs}	N_e	N_r	α	β	θ
E-MTransE	Census	0.08	64	128	0.1	1	0.1	-	7	1	5	5	0.95
E-MTransE	Cora	0.34	64	128	0.1	1	0.1	-	7	1	5	5	0.25
E-MTransE	Febrl	0.08	64	128	0.1	1	0.1	-	7	1	5	5	0.95
KR_EAR	Census	0.18	80	100	0.1	1	0.1	500	10	1	-	-	0.65
KR_EAR	Cora	0.32	128	1024	0.1	10	0.1	100	7	1	-	-	0.1
KR_EAR	Febrl	0.13	128	1024	0.5	10	0.1	1000	7	1	-	-	0.1
LogTransH	Census	0.09	128	64	0.1	1	0.1	1000	7	1	-	-	-
LogTransH	Cora	0.08	128	64	0.1	1	0.1	1000	7	1	-	-	-
LogTransH	Febrl	0.09	32	128	0.1	0.1	0.1	1000	7	1	-	-	-
MTransE	Census	0.21	128	512	0.1	2	0.1	-	4	1	10	-	0.85
MTransE	Cora	0.71	128	1024	0.1	1	0.1	-	7	1	5	-	0.4
MTransE	Febrl	0.09	128	512	0.1	1	0.1	-	7	1	5	-	0.9
RLTransE	Census	0.95	64	64	0.1	1	0.1	5000	7	1	-	-	1.56
RLTransE	Cora	0.57	32	32	0.1	1	0.1	1000	7	1	-	-	2.48
RLTransE	Febrl	0.98	128	32	0.1	0.1	0.1	5000	7	1	-	-	2.18
SEEA	Census	0.15	128	256	0.1	10	0.1	-	7	1	-	256	-
SEEA	Cora	0.33	128	256	0.1	10	0.1	-	7	1	-	256	-
SEEA	Febrl	0.49	128	256	0.1	10	0.1	-	7	1	-	256	-
HTransE	Census	0.01	80	100	0.1	1	0.1	100	8	2	-	-	0.95
HTransE	Cora	0.43	32	128	0.1	1	0.1	100.0	8	2	-	-	0.35
HTransE	Febrl	0.01	80	100	0.1	1	0.1	50	8	2	-	-	0.85
TransE	Census	0.21	80	100	0.1	1	0.1	50	8	2	-	-	0.75
TransE	Cora	0.54	80	100	0.1	1	0.1	50	8	2	-	-	0.3
TransE	Febrl	0.58	80	100	0.1	1	0.1	500	8	2	-	-	0.1
TransH	Census	0.17	80	100	0.1	1	0.1	500	1	0	-	-	0.85
TransH	Cora	0.44	64	128	0.1	1	0.1	1000	7	1	-	-	0.2
TransH	Febrl	0.60	64	128	0.1	1	0.1	1000	7	1	-	-	0.1
VEER	Census	0.95	16	81 32	0.1	0.1	0.1	100	-	-	-	-	0.5
VEER	Cora	0.85	16	64	0.1	0.1	0.1	500	-	-	-	-	0.5
VEER	Febrl	0.96	64	32	0.1	1	0.1	50	-	-	-	-	0.55