Entity Resolution with Attention Based Convolutional Neural Network

Kailin Tang
tangkl@umich.edu

Siyuan Xie
xiesiyu@umich.edu

ABSTRACT

Entity resolution is a classic problem in the data integration field which finds matching entities from two different data sources. In recent years, deep learning has gained great popularity in pattern recognition fields like natural language processing (NLP) and computer vision (CV). By following this trend, researchers also bring deep learning to the entity resolution problem. The DeepMatcher [26] explores a systematic design space involving different RNN and Attention- based architectures in entity resolution and shows that deep learning is effective in this field. However, we find that there are still something worthwhile exploring in such a design space, for example, CNN models for tuple representation. CNN and RNN models are both powerful structures in NLP domains, while CNN is more beneficial to discover local relationships and easier to implement with attention mechanism. We complement the design space of DeepMatcher by adding naïve CNN and attention-based CNN module as tuple attribute summarizer and perform several experiments to evaluate the performance of CNN related architectures in entity resolutions. The experimental results indicate that the attention-based CNN models can achieve state-of-art performance for entity resolution, which shows that CNN models can also be applied in entity resolution field as alternatives. Next, in order to reduce the training time for CNN related models, we show that dimension truncation for tuple representation is effective for reducing the training time without sacrificing too much performance, which makes the attention-based CNN model for entity resolution more practical.

KEYWORDS

Entity Resolution, Deep Learning, Convolutional Neural Networks, Attention Mechanism

1 INTRODUCTION

Entity Resolution (ER) (a.k.a. record linkage) is a classical problem in data integration field that focuses on finding records in a dataset that refer to same entity across different data sources. Due to the heterogeneity feature of data formats and errors happened during operation, the duplicated, incomplete and erroneous data often appear in database systems. In order to manage this data properly, it is necessary to clean it.

In the past few years, deep learning (DL) has become a major direction in machine learning [6, 9, 25, 32]. DL yields state-of-art results for tasks over data with some hidden structure, e.g., text, image, and speech and have been successfully applied to the pattern recognition problems like natural language process and computer vision and outperform other traditional models on such data, using labeled examples, DL can automatically construct important features, thereby obviating the need for manual feature engineering. This has transformed fields such as image and speech processing, medical diagnosis, autonomous driving, robotics, NLP, and many others [6, 32]. Recently, DL has also gained the attention of the database research community [5, 9].

Some researches also introduced deep learning to the field of entity resolution and make some progress [7, 19, 26]. The authors in [19] proposed distributed tuple representation using RNN models to better represent the entities. The authors in [26] propose DeepMatcher, a design space of deep learning structures in entity resolution and make comprehensive experiments on the deep learning design space to evaluate the limits and benefits of deep learning methods in entity resolution. Our work is based on DeepMatcher and we wants to extend the design space of DeepMatcher and evaluate the performance of different models under diverse scenarios.

Convolutional Neural Networks (CNNs) (LeCun et al., 1998) are widely used to model sentences[16, 31] (Kalchbrenner et al., 2014; Kim, 2014) and sentence pairs (Socher et al., 2011; Yin and Schu ¨tze, 2015a), especially in classification tasks. CNNs are supposed to be good at extracting robust and abstract features of input. The CNN has been shown to be effective in sentence representation and sentence matching, so we expect them also to be effective for entity matching.

Some prior work proposes simple mechanisms that can be interpreted as controlling varying attention; e.g., Yih et al. (2013) employ word alignment to match related parts of the two sentences. Yin proposed ABCNN, which is to implement attention mechanism into CNN models for sentence matching. In contrast, Yin's attention scheme based on CNNs models relatedness between two parts fully automatically. Moreover, attention at multiple levels of granularity, not only at word level, is achieved as Yin stack multiple convolution layers that increase abstraction. The attention-based CNN is shown to be powerful in discovering the semantic similarities between sentences.

In this paper, we want to follow the trend of deep learning and penetrate deeper into the entity resolution field using cutting edge deep learning techniques. We integrate the attention-based CNN model with DeepMatcher and achieved state-of-art performance across several datasets.

2 RELATED WORK

2.1 Entity Matching

EM has received much attention [1, 4, 12, 20, 33]. Most EM approaches in the database literature match structured records[12], whereas most related works in NLP and similarity learning match entity mentions that are text spans. Work on the matching step of EM typically uses rules, learning, or crowdsourcing. Rule-based solutions[23, 27] are interpretable but require the heavy involvement of a domain expert. To address this problem, some work has focused on learning matching functions [2, 8, 24]. Finally, a different line of work develops methods to leverage human experts [3, 15, 17] to guide EM and help refine the learned matching functions. The blocking step of EM has also received significant attention [4, 11]. Our solutions in this paper assume as input the output of a blocking procedure, and thus can work with any blocking procedure that has been proposed.

2.2 Entity resolution in Deep learning

DeepER[19], a recent pioneering work , focuses on designing DL solutions to EM. That work proposes two DL models that build upon neural network (NN) architectures used extensively in the NLP literature. In addition, DeepER discusses blocking and how distributed representations can be used to design efficient blocking mechanisms. The authors in [26] propose DeepMatcher, a design space of deep learning structures in entity resolution and make comprehensive experiments on the deep learning design space to evaluate the limits and benefits of deep learning methods in entity resolution.They did a great job by experimenting a series of different RNN structures mainly including RNN+LSTM and attention-based RNN. Both DeepER and DeepMatcher formulate EM as a pairwise binary classification task using logistic loss. But there are other ways to formulate the same problem. For example, triplet learning[13] first learns good embeddings for objects using triplet loss, and then learns an NN classifier. The work in [22] attacks the problem by learning good entity embeddings in vector space using contrastive loss. A distance threshold or an NN classifier is used for classification. Yet another potential approach (used in the QuestionAnswering (QA) domain) poses matching as a nearest neighbor search problem [10] and can be adapted for EM. Finally, Matching Networks [21][80], an approach to perform image classification using labels of related images, may also be adapted for EM.

2.3 Attention-Based Deep Learning

Even though there is little if any work on attention mechanisms in CNNs for NLP, attention-based CNNs have been used in computer vision for visual question answering (Chen et al., 2015), image classification (Xiao et al., 2015), caption generation (Xu et al., 2015), image segmentation (Hong et al., 2016) and object localization (Cao et al., 2015). Mnih et al. (2014) apply attention in recurrent neural networks (RNNs) to extract information from an image or video by adaptively selecting a sequence of regions or locations and only processing the selected regions at high resolution. Gregor et al. (2015) combine a spatial attention mechanism with RNNs for image generation. Ba et al. (2015) investigate attention-based RNNs for recognizing multiple objects in images. Chorowski et al. (2014) and Chorowski et al. (2015) use attention in RNNs for speech recognition.Attention-based DL systems have been applied to NLP after their success in computer vision and speech recognition. They mainly rely on RNNs and end-to-end encoderdecoders for tasks such as machine translation (Bahdanau et al., 2015; Luong et al., 2015) and text reconstruction (Li et al., 2015; Rush et al., 2015).

2.4 Deep Learning on sentence pair modeling

Kim Y[31] adopt a simple CNN model to represent sentence with different feature maps by using different kernel windows. Kalchbrenner[16] improved Kim's work in using CNN for sentence representation by introducing wide convolution and dynamic k-max pooling.Yu et al. (2014) present a bigram CNN to model question and answer candidates. Yang et al. (2015) extend this method and get state-of-the-art performance on the WikiQA dataset. Feng et al. (2015) test various setups of a bi-CNN architecture on an insurance domain QA dataset. Tan et al. (2016) explore bidirectional LSTMs on the same dataset.Blacoe and Lapata (2012) form sentence representations by summing up word embeddings. Socher et al. (2011) use recursive autoencoders (RAEs) to model representations of local phrases in sentences, then pool similarity values of phrases from the two sentences as features for binary classification. Yin and Schutze (2015a) similarly replace an RAE with a CNN. In all three papers, the representation of one sentence is not influenced by the other – in contrast to our attention-based model. Bowman et al. (2015b) use recursive neural networks to encode entailment on SICK (Marelli et al., 2014b). Rocktaschel et al. (2016) present an ¨ attention-based LSTM for the Stanford natural language inference corpus (Bowman et al., 2015a). Some prior work aims to solve a general sentence matching problem. Hu et al. (2014) present two CNN architectures[14], ARC-I and ARC-II, for sentence matching. ARC-I focuses on sentence representation learning while ARC-II focuses on matching features on phrase level. Yin and Schutze (2015b) propose the MultiGranCNN architecture to model general sentence matching based on phrase matching on multiple levels of granularity and get promising results. Wan et al. (2015) try to match two sentences in AS and SC by multiple sentence representations, each coming from the local representations of two LSTMs. Yin proposed BCNN[29] in sentence matching by adopting Kal's method for sentence representation and extracting multi-granularity sentence relationship features across the network. Yin further improved BCNN by exploring the ways to implement attention mechanism in CNN and proposed ABCNN[28].

3 PROBLEM DEFINITION AND DATASETS

We are going to solve entity resolution problem by finding the matching tuples in a set of entities, here is the definition: Given an entity set T with n tuples and m attributes, for all pairs within T say $(t, t^{'})$ where $t! = t^{'}$, determine whether the two tuples in the pair is matching[19]. This problem can be treated as a binary classification problem in machine learning. Given two entities, the binary classifier needs to answer whether they are matching.
To make sure we can test and evaluate our method, we find some datasets for entity matching, there are totally 6 datasets selected which focus on two categories: products and citations. Despite of different source, the size of the datasets also varies from thousands to few millions, the detailed description of each dataset is given Table 1

Dataset	Domain	Pairs	Positive	Attribute
iTunes-Amazon	music	539	132	8
Abt-Buy	product	9575	1028	3
Walmart-Amazon	electronics	10242	962	5
Amazon-Google	software	11460	1167	5
DBLP-ACM	citation	12363	2220	4
DBLP-Scholar	citation	28707	5347	4

Table 1: Datasets

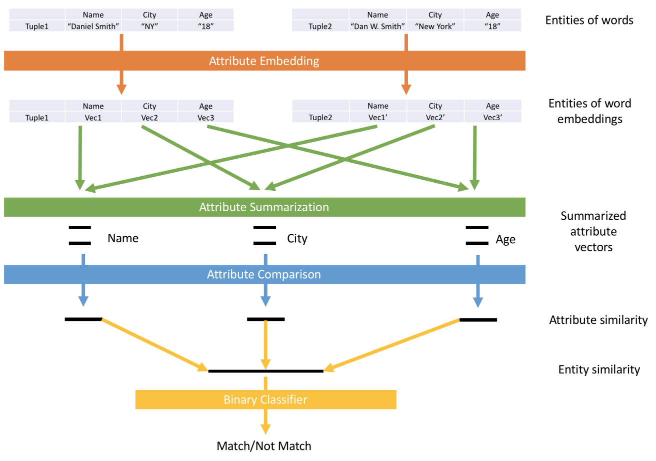


Figure 1: The design space of Entity Resolution in DeepMatcher. The Attribute Embedding transforms string to numerical vector that similar attributes are close to each other in values. The Attribute Summarization generates summarized vectors from embedding vectors. The Attribute Comparison compares summarized vector of two attributes. The compared vector of each attribute will be concatenated into one compared vector of tuples

4 PROPOSED APPROACHES

4.1 DeepMatcher Design Space

The authors in [26] proposed DeepMatcher, where they conduct a systematic experiment of RNN choices under their design space. We utilize their preexisting system and add our CNN and attention-based CNN model in the system for evaluating and completing the system.

Figure 1 shows the architecture template of DeepMatcher for DL solutions for EM. It uses the categorization of DL models for matching related tasks and is built around the same categorization dimensions: (1) the language representation, (2) the summarization technique, and (3) the comparison method used to analyze an input pair of sequences. The template consists of three main modules each of which is associated with one of these dimensions. Before discussing the modules, we discuss assumptions regarding the input. DeepMatcher assume that each input point corresponds to a pair of entity mentions $(e1, e2)$, which follow the same schema with attributes $A_1, ..., A_N$. Textual data can be represented using a schema with a single attribute. It further assumes that the value of attribute A_j for each entity mention e corresponds to a sequence of words $w_{e,j}$. It allows the length of these sequences to be different across different entity mentions. Given this setup, each input point corresponds to a vector of N entries (one for each attribute $A_j \epsilon A_1, ..., A_N$) where each entry j corresponds to a pair of word sequences $w_{e_1,j}$ and $w_{e_2,j}$.

The Attribute Embedding Module: For all attributes $A_i \epsilon A_1 A_N$, this module takes sequences of words $w_{e_1,j}$ and $w_{e_2,j}$ and converts them to two sequences of word embedding vectors $u_{e_1,j}$ and $u_{e_2,j}$ whose elements correspond to d-dimensional embeddings of the corresponding words. More precisely, if for e_1, word sequence $w_{e_1,j}$ contains m elements then we have $u_{e_1,j} \epsilon R_{d \times m}$. The same holds for e_2. The overall output of the attribute embedding module of our template is a pair of embeddings $u_{e_1,j}$ and $u_{e_2,j}$ for the values of attribute A_j for entity mentions e_1 and e_2. We denote the final output of this module as $(u_{e_1,j}, u_{e_2,j})_{j=1}^{N}$.

The Attribute Similarity Representation Module: The goal of this module is to automatically learn a representation that captures the similarity of two entity mentions given as input. This module takes as input the attribute value embeddings $(u_{e_1,j}, u_{e_2,j})_{j=1}^{N}$ and encodes this input to a representation that captures the attribute value similarities of e_1 and e_2. For each attribute A_j and pair of attribute embeddings $(u_{e_1,j}, u_{e_2,j})$ the operations performed by this module are split into two major parts: (1) Attribute summarization(green in Figure 1). This module takes as input the two sequences $(u_{e_1,j}, u_{e_2,j})$ and applies an operation H that summarizes the information in the input sequences. More precisely, let sequences $u_{e_1,j}$ and $u_{e_2,j}$ contain m and k elements respectively. A_n h dimensional summarization model H takes as input sequences $u_{e_1,j} \epsilon R_{d \times m} and u_{e_2,j} \epsilon R_{d \times k}$ and outputs two summary vectors $s_{e_1,j} \epsilon R^h and s_{e_2,j} \epsilon R^h$. The role of attribute summarization is to aggregate information across all tokens in the attribute value sequence of an entity mention. This summarization process may consider the pair of sequences $(u_{e_1,j}, u_{e_2,j})$ jointly to perform more sophisticated operations such as soft alignment [18]. (2) Attribute comparison(blue in Figure 1). This part takes as input the summary vectors $s_{e_1,j} \epsilon R^h and s_{e_2,j} \epsilon R^h$ and applies a comparison function D over those summaries to obtain the final similarity representation of the attribute values for e_1 and e_2. We denote that similarity representation by $s_j \epsilon R^l with s_j = D(s_{e_1,j}, s_{e_2,j})$.

4.2 Word Embedding

Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers [30]. It's very important as it converts texture information to numerical features, which are feasible to be analyzed and modeled by machine learning methods. One common way is Word2vec [18], which is the most popular way in recent NLP domains for word embedding. Word2vec is a group of related models that are shallow, two-layer neural networks which are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word2vec performs well in the measure the similarity between two groups of words and the two most important characteristics of Word2vec are the preservation of semantic and syntactic relationships.

4.3 CNN in sentence representation

CNN is the state-of-art and trend in dealing with computer vision problems in recent years, while it also becomes more and more popular in NLP domains for its characteristics to excavate features from local components.

Deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers. Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. Each convolutional neuron processes data only for its receptive field. Pooling layers combine the outputs of neuron clusters at one layer into a single neuron in the next layer. Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network.

Kim first adopt CNN in sentence representation by implementing a typical CNN structure. The basic idea is to utilize different kernel sizes of convolution layers to exploit the local features upon the embedding of the sentences. Kalchbrenner made some improvements on the sentence representation by introducing wide-convolution and dynamic k-max pooling. More improvements can be the combination of features from different granularities by concating the features in different stages in the deep learning structure.

We basically implement Kim's method for the simple CNN structures in entity resolution shown as Figure 2. The convolution layer generate different feature maps by using kernels with different sizes(e.g, size 2,3,4, which responds to red, blue and yellow in Figrue 2) along the sentence embedding matrix. The matrix has a fixed sentence length, with extra space filling zeros. We generate features using max-pooling from feature maps as the sentence representation. The sentence representation of two sentence will then be attributed compared and make a classification.

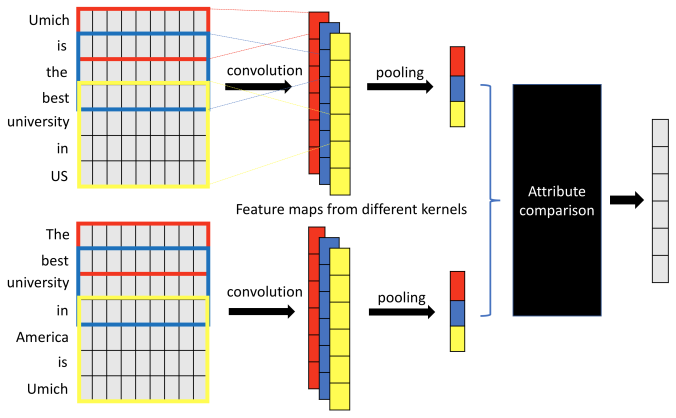


Figure 2: The naive CNN structure for Entity Resolution. The red, blue, yellow window represent kernel with size 2,3 and 4, respectively in the convolution layer.

4.4 Attention

The idea of attention mechanism is introduced to deep learning and has gained a great popularity in NLP and CV domains. The basic idea is to simulate the attention mechanism of humans, where humans will also focus on some local impressive features when observing something instead of simply traversing all the information. Prior work on attention in deep learning (DL) mostly addresses long short-term memory networks (LSTMs). LSTMs achieve attention usually in a word-to-word scheme, and word representations mostly encode the whole context within the sentence. It is not clear whether this is the best strategy. This observation was also investigated by Yao et al. (2013b) where an information retrieval system retrieves sentences with tokens labeled as DATE by named entity recognition or as CD by POS tagging if there is a "when" question. However, labels or POS tags require extra tools. CNNs benefit from incorporating attention into representations of local phrases detected by filters; in contrast, LSTMs encode the whole context to form attention-based word representations - a strategy that is more complex than the CNN strategy and (as our experiments suggest) performs less well for some tasks.

4.5 ABCNN

We adopt the attention-based model proposed by Yin, ABCNN. The architecture have three forms, ABCNN-1, ABCNN-2 and ABCNN-3, where the former 2 are different choices of implementing attention-based mechanism in CNN and the latter is the hybrid model of those two.

ABCNN-1. The ABCNN-1 (left in Figure 3) employs an attention feature matrix A to influence convolution. Attention features are intended to weight those units of s_i more highly in convolution that are relevant to a unit of $s_{1-i}(i\epsilon 0,1)$; we use the term "unit" here to

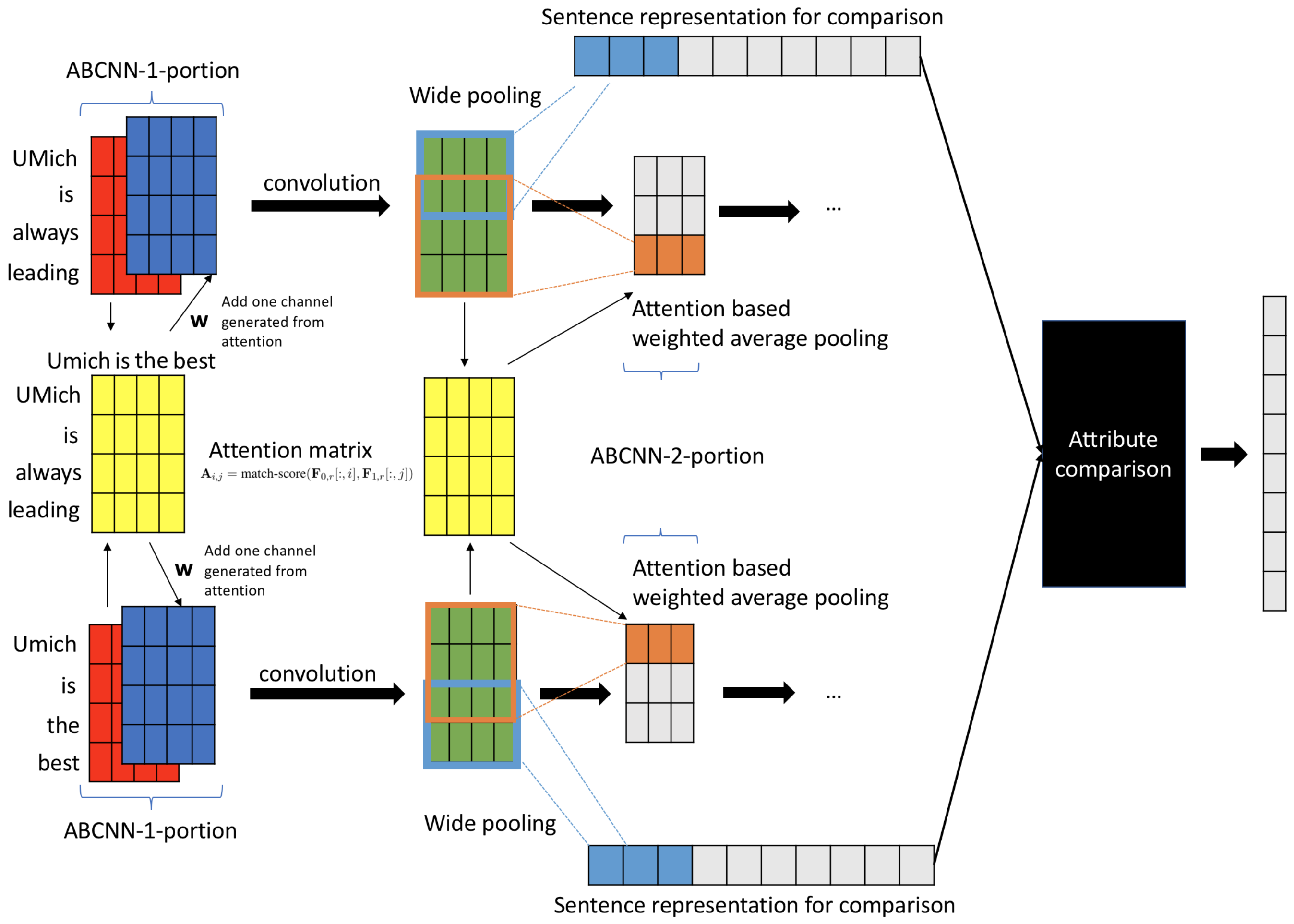


Figure 3: The ABCNN structure. The red matrix represent original sentence matrix before convolution, yellow matrix represent attention matrix, blue matrix represent additional channel transformed from attention matrix, green matrix represent matrix after convolution. The sentence representation comes from average pooling of matrix after convolution. The structure considers correlation between two sentences by integrating attention matrix in the CNN structure

refer to words on the lowest level and to phrases on higher levels of the network.Each column is the representation of a unit, a word on the lowest level and a phrase on higher levels. We first describe the attention feature matrix A informally (between layer "Conv input", yellow in Figure 3). A is generated by matching units of the left representation feature map with units of the right representation feature map such that the attention values of row i in A denote the attention distribution of the $i-th$ unit of s_0 with respect to s_1, and the attention values of column j in A denote the attention distribution of the $j-th$ unit of s_1 with respect to s_0. A can be viewed as a new feature map of s_0 (resp. s1) in row (resp. column) direction because each row (resp. column) is a new feature vector of a unit in s_0 (resp. s_1). Thus, it makes sense to combine this new feature map with the representation feature maps and use both as input to the convolution operation. We achieve this by transforming A into the two blue matrices in Figure 3 that have the same format as the representation feature maps. As a result, the new input of convolution has two feature maps for each sentence (shown in red and blue). Our motivation is that the attention feature map will guide the convolution to learn "counterpart-biased" sentence representations. More formally, let $F_{i,r} \epsilon R_{d \times s}$ be the representation feature map of sentence $i(i \epsilon 0, 1)$. Then we define the attention matrix $A \epsilon R_{s \times s}$ as follows:

$$A_{i,j} = match - score(F_{0,r}[:,i], F_{1,r}[:,j])$$

(1) The function match-score can be defined in a variety of ways. We found that $1/(1+|x-y|)$ works well where $|\cdot|$ is Euclidean distance. Given attention matrix A, we generate the attention feature map $F_{i,a}$ for s_i as follows:

$$F_{0,a} = W_0 \cdot A^T, F_{1,a} = W_1 A$$

The weight matrices $W_0 \epsilon R_{d \times s}$, $W_1 \epsilon R_{d \times s}$ are parameters of the model to be learned in training. We stack the representation feature map $F_{i,r}$ and the attention feature map F_i,a as an order 3 tensor and feed it into convolution to generate a higherlevel representation feature map for $s_i(i \epsilon 0, 1)$. In Figure 3, s_0 and s_1 has 4 units. The output of

convolution (shown in green, filter width w = 3) is a higher-level representation feature map with 4 columns for s_0 and 4 columns for s_1.
ABCNN-2. The ABCNN-1 computes attention weights directly on the input representation with the aim of improving the features computed by convolution. The ABCNN-2 (right in Figure 3)) instead computes attention weights on the output of convolution with the aim of reweighting this convolution output. In the example shown in Figure 3, the feature maps output by convolution for s_0 and s_1 (layer marked "Convolution" in Figure 3) have 4 columns; each column is the representation of a unit. The attention matrix A compares all units in s_0 with all units of s_1. We sum all attention values for a unit to derive a single attention weight for that unit. This corresponds to summing all values in a row of A for s_0 ("col-wise sum", resulting in the column vector of size 4 shown) and summing all values in a column for s1 ("row-wise sum", resulting in the row vector of size 4 shown). More formally, let $A \epsilon R_{s \times s}$ be the attention matrix, $a_{0,j} = \sum A[j,:]$ the attention weight of unit j in s_0, $a_{1,j} = \sum A[:,j]$ the attention weight of unit j in s_1 and $F^c_{i,r} \epsilon R^{d \times (s_i + w - 1)}$ the output of convolution for s_i . Then the j-th column of the new feature map $F^p_{i,r}$ generated by w-ap is derived by:

$$F^p_{i,r}[:,j] = \sum_{k=j:j+w} a_{i,k} F^c_{i,r}[:,k], j = 1 \ldots s_i$$

Note that $F^p_{i,r} \epsilon R_{d \times s_i}$, i.e., ABCNN-2 pooling generates an output feature map of the same size as the input feature map of convolution. This allows us to stack multiple convolution-pooling blocks to extract features of increasing abstraction.

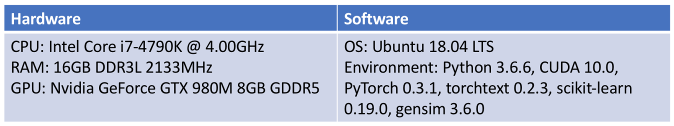
There are three main differences between the ABCNN-1 and the ABCNN-2. (i) Attention in the ABCNN-1 impacts convolution indirectly while attention in the ABCNN-2 influences pooling through direct attention weighting. (ii) The ABCNN-1 requires the two matrices W_i to convert the attention matrix into attention feature maps; and the input to convolution has two times as many feature maps. Thus, the ABCNN-1 has more parameters than the ABCNN-2 and is more vulnerable to overfitting. (iii) As pooling is performed after convolution, pooling handles larger-granularity units than convolution; e.g., if the input to convolution has word level granularity, then the input to pooling has phrase level granularity, the phrase size being equal to filter size w. Thus, the ABCNN-1 and the ABCNN-2 implement attention mechanisms for linguistic units of different granularity. The complementarity of the ABCNN-1 and the ABCNN-2 motivates us to propose the ABCNN-3, a third architecture that combines elements of the two. ABCNN-3 (Figure 3) combines the ABCNN-1 and the ABCNN-2 by stacking them; it combines the strengths of the ABCNN-1 and -2 by allowing the attention mechanism to operate (i) both on the convolution and on the pooling parts of a convolution pooling block and (ii) both on the input granularity and on the more abstract output granularity.

5 EXPERIMENT

5.1 Experiment Setup

In this project, we evaluate the performance of entity resolution with the following setup. All experiments are done with a Clevo P770ZM-G mobile workstation. For the hardware, this machine has an Intel Core i7 CPU, 16GB RAM, and a Nvidia GTX 980M graphic card. For the software, all experiments are done in an Ubuntu 18.04 LTS operating system with necessary system environments, the detailed specification of the experimental setup is shown in Figure 4.

The experiments are done in all six datasets we mentioned in the previous sections. In order to compare with the state-of-art deep learning-based entity machine algorithms, we used 5 baseline models called Average, SIF, RNN, Attention, and Hybrid. The Average approach proposed in [19] is a simple approach which uses the sampled mean vector to represent the tuple. The SIF approach [26] is a more advanced Average approach which takes the word frequency into account. The RNN approach which is used in both [19] and [26] is a classical approach used in NLP field which uses a bidirectional RNN with LSTM to take the order of words into account when modeling the tuple. Attention-based attributes summarization proposed in [26] use two sequence alignment in two different directions to represent the tuple. Hybrid combined the RNN and Attention approach which take both the sequence context and alignment into account to generate the tuple representation.



Hardware	Software
CPU: Intel Core i7-4790K @ 4.00GHz RAM: 16GB DDR3L 2133MHz GPU: Nvidia GeForce GTX 980M 8GB GDDR5	OS: Ubuntu 18.04 LTS Environment: Python 3.6.6, CUDA 10.0, PyTorch 0.3.1, torchtext 0.2.3, scikit-learn 0.19.0, gensim 3.6.0

Figure 4: Experiment Setup

We used the original implementation of the SIF, RNN, Attention and Hybrid model provided in [26] called DeepMatcher. This open-source library is implemented in Python and provide flexible interfaces for adding new features and modules. All of the models in DeepMatcher are implemented in PyTorch, an open-source machine learning library for Python. For the average method, we simply use a Python lambda function to represent the functionality of attribute summarizer as the mean vector of all word embedding vector for each attribute. For the naïve CNN and attention-based CNN, we designed and implemented two PyTorch module on the top of the DeepMatcher as the summarizer, which is compatible with DeepMatcher so that it can benefit from other existing modules from DeepMatcher like attribute embedding and attribute comparison. We implement the network of the two CNN models we described in the previous section as the attribute summarizer of the entity resolution workflow. For both models, in the attributes comparison, we use the absolute difference method to model the similarity between two tuples. Next, for the fully connected layer for classification, we use a 3-layer residual network with ReLu activation function as the final decision module which is used to decide whether the given tuple pair are referred to the same object.

We designed several experiments to evaluate the performance of the entity matching algorithms. First of all, we compared the F1 score of our CNN based models with the baselines to measure the overall performance of the matching. Second, we measured the training time of all seven models in each training epoch. Next, we measure the performance of two CNN models with truncated input dimensions. Last, we compared the performance of different classification models for entity similarity vectors.

5.2 Experiment Result

5.2.1 F1 Scores of Baselines and CNN Based Models.

First of all, as the evaluation provided in [26], we also evaluate the F1 scores for the models. The Table 2 shows the results in all six datasets. From the result, we can come to the conclusion that the ABCNN based attribute summarization can achieve the state-of-art performance for entity resolution tasks. In iTunes-Amazon, Amazon-Google and Walmart-Amazon datasets, ABCNN outperforms all other baseline models. In DBLP-ACM and DBLP-Scholar datasets, though ABCNN doesn't yield the best result, the difference between the best model and the ABCNN based model are still very small. In the Abt-Buy dataset, ABCNN doesn't behave well, the reason is that the Abt-Buy dataset has many missing entries for some very important attributes like description, these missing entries greatly degrade the performance of the model. From the result, we find that average based methods (Average, SIF) perform the worst, it is because such models consider neither intra-sentence word order nor inter-sentence correlation (attention). For the Hybrid and ABCNN model, both of them take the sequence order and attention into account, so both of them outperforms other models at the most time. ABCNN perform well because it uses a more complex model to capture the hidden pattern under tuples and take attention into account, it is easier and more natural to combine attention mechanism with CNN models rather than RNN models. The Table 3 shows the percentage that ABCNN can outperform compared with baselines and naïve CNN, note that negative percentage means the percentage that CNN is worse than other models.

Another take away from this experiment is that the F1 scores can be achieved for entity resolution are dataset variant, for the dataset like DBLP-ACM and DBLP-Scholar, entities are all well-defined with rich texts, in which almost every algorithm can get a satisfactory result. However, if the data are missing or noisy, deep learning based entity resolution may not perform well.

5.2.2 Training Time of Baselines and CNN Based Models per Epoch in Seconds.

As an important concern for practical entity resolution, we also measure the training time of baseline and CNN related models per epoch shown in Table 4. Unfortunately, the ABCNN got the worst training time, at the worst case in DBLP-Scholar, it is more than 20x slower than Hybrid, the second slowest method. The reason why ABCNN is extremely slow is that the complexity of model itself, ABCNN-3 is hybrid from ABCNN-1 and ABCNN-2, there are more parameters for such a model to estimate thus it takes longer time. However, it is not a big deal for the entity resolution workload, because, for most data integration and cleaning platform, this step is always done offline in batch. Besides, although the model training takes more time, the model inference is done instantaneously, so if we got a good model one time, we can use it all the time in the future.

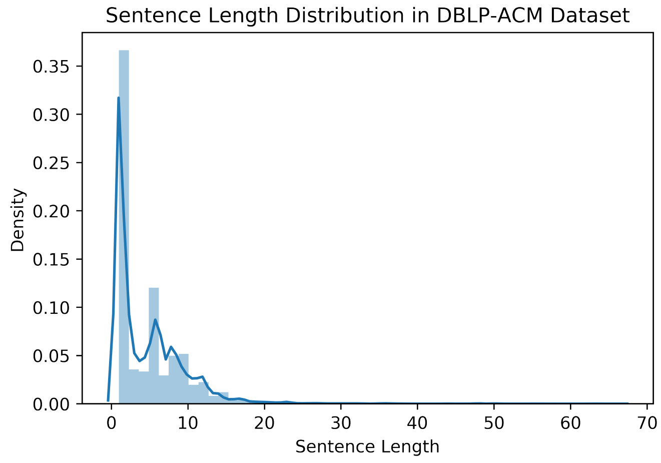


Figure 5: Sentence Length Distribution in DBLP-ACM Dataset

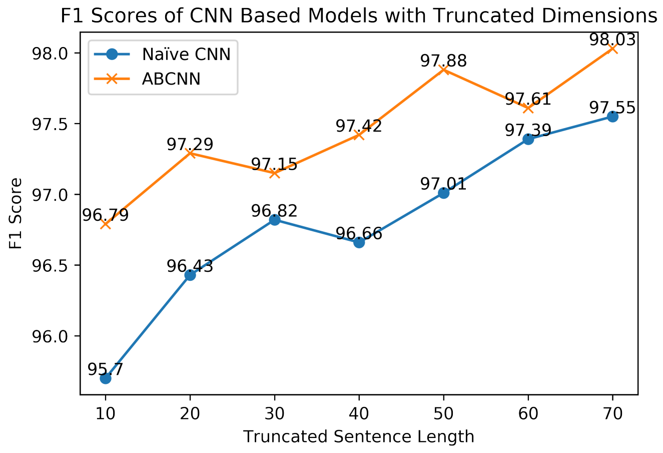


Figure 6: F1 Scores of CNN Based Models with Truncated Dimensions

5.2.3 Performance of CNN Based Models with Truncated Dimensions.

As we shown in the previous part, although the ABCNN can achieve the promising result, the biggest downside of that approach is the training time. It takes even more than 10x of time compared to baseline models. Apart from the complexity of the model, another reason makes the large training time is the full sentence length usage for each attribute. Taking the DBLP-ACM dataset, for example, the maximum sentence length for all entries is 66, as a result, the input of the ABCNN model is a fixed dimension, which is 66 * 300, where 300 is the length for word embedding. If the sentence length in some entries does not reach 66, we just use a zero padding for

	Average[19]	SIF[26]	RNN[19, 26]	Attention[26]	Hybrid[26]	naïve CNN	ABCNN
iTunes-Amazon	82.75	84.74	84.21	80.64	84.37	72.41	90.62
Abt-Buy	32.9	36.06	38.82	45.2	67.45	37.71	43.93
Amazon-Google	54.74	56	61.44	50.81	68.52	58.56	69.32
Walmart-Amazon	46.28	58.85	63.32	49.09	63.48	53.1	66.15
DBLP-ACM	94.41	96.57	97.24	98.21	97.89	97.73	98.1
DBLP-Scholar	87.42	87.82	91.79	91.93	94.2	91.01	92.87

Table 2: F1 Scores of Baselines and CNN Based Models

	Average[19]	SIF[26]	RNN[19, 26]	Attention[26]	Hybrid[26]	naïve CNN
iTunes-Amazon	9.51%	6.94%	7.61%	12.38%	7.41%	25.15%
Abt-Buy	33.53%	21.82%	13.16%	-2.81%	-34.87%	16.49%
Amazon-Google	26.64%	23.79%	12.83%	36.43%	1.17%	18.37%
Walmart-Amazon	42.93%	12.40%	4.47%	34.75%	4.21%	24.58%
DBLP-ACM	3.91%	1.58%	0.88%	-0.11%	0.21%	0.38%
DBLP-Scholar	6.23%	5.75%	1.18%	1.02%	-1.41%	2.04%

Table 3: F1 Scores Percentage of ABCNN that outperforms Baselines and naïve CNN

	Average[19]	SIF[26]	RNN[19, 26]	Attention[26]	Hybrid[26]	naïve CNN	ABCNN
iTunes-Amazon	0.2	0.4	1.7	2.7	5.4	0.9	24
Abt-Buy	0.7	1.1	4.1	7.9	15.5	2.5	196.3
Amazon-Google	0.8	1	2.9	4.3	10.8	2.1	117.4
Walmart-Amazon	0.9	1.3	3.8	8.2	17.8	3.1	177.2
DBLP-ACM	0.5	0.8	4.2	10.1	20.3	2.9	265.9
DBLP-Scholar	0.6	1	6.9	22.1	36.5	6.1	838.6

Table 4: Training Time of Baselines and CNN Based Models

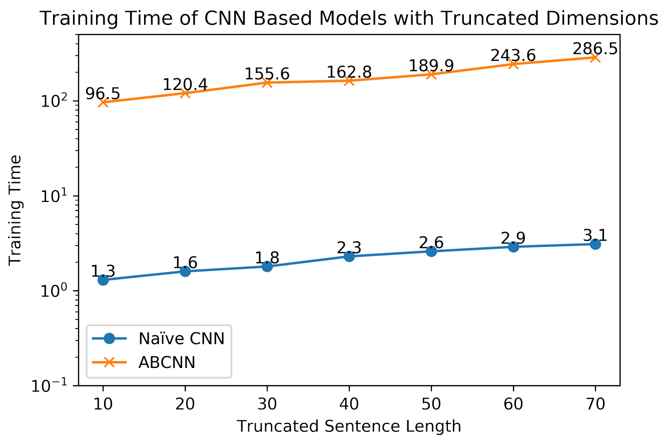


Figure 7: Training Time of CNN Based Models with Truncated Dimensions

remaining positions. This is a potential drawback of CNN model which require fixed input shape. However, as we visualized the sentence length distribution in the DBLP-ACM shown in Figure 5, most entries' length is within 20, based on this, we come up with an idea to truncate the input size and just drop any words larger than that threshold. The Figure 6 and Figure 7 show the performance after dimension truncation.

From those two figures, we can find that the performance of the entity matching doesn't degrade too much even when we greatly shrink the input, it is because most entries in the given datasets are short. We think the dimension truncation for CNN model is a trade-off for entity matching, if you have many data with short length or limited computational resources, it is fine to truncate the dimension and greatly reduce your training time. This truncation makes the ABCNN based entity resolution become practical in real life.

5.2.4 Classification Models for Entity Similarity.

As we get an entity similarity vector from the attribute comparator, we need to use a binary classifier to judge whether the given pair is matching or not. In our implementation of naïve CNN and ABCNN, we adopt a 3-layer fully connected residual network with ReLU activation function to do the classification job. It is natural to use a fully connected layer at the end of CNN for most work, but we still evaluate the performance of different classifier for entity similarity vectors. We generate the AUC curve for seven different classifiers with averaged entity similarity vector and the result is shown in Figure 8. From the result, one can easily find that DNN outperforms

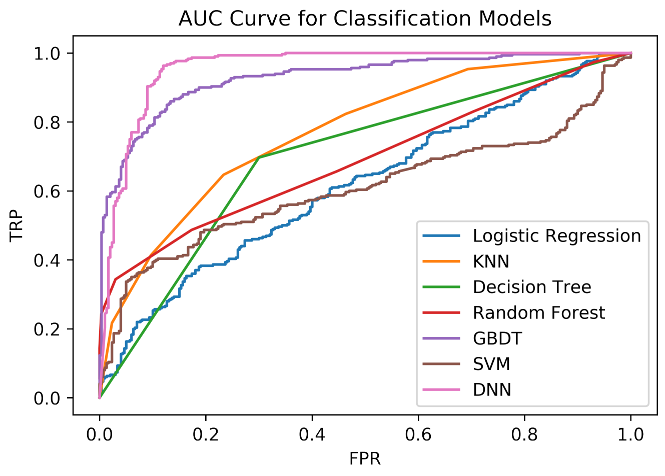


Figure 8: AUC Curve for Classification Models

all other classification models because DNN is able to capture more correlations within the similarity vector.

6 CONCLUSIONS

In this project, we further explore the design space of entity resolution with deep learning. We proposed two novel CNN based attribute summarizers for entity resolution framework. For naïve CNN, we use simple kernels of different size to capture the spatial proximity of words in table entry and generate attribute representation. For attention-based CNN, we consider both intra-sentence word order and inter-sentence correlation (attention) and hybrid them together using a CNN model for attribute summarization. From the experiments, we show that the ABCNN can achieve the state-of-art performance compared to other deep learning based entity resolution algorithms. We also show that the training time of ABCNN can be greatly reduced by performing dimension truncation at the input layer which makes the ABCNN become practical.

ACKNOWLEDGMENTS

Thanks to Prof. Barzan Mozafari for providing this wonderful course, through which we learned a lot of advanced state-of-art database management systems techniques. Also, thanks to the great advice from the professor with respect to our research project. We would like to thank the AnHai's Group of the University of Wisconsin-Madison who provided the datasets for evaluation and also the open-sourced version of DeepMatcher library for baseline model implementation that greatly facilitates our work.

REFERENCES

[1] Panagiotis G. Ipeirotis Ahmed K. Elmagarmid and Vassilios S. Verykios. 2007. *Duplicate Record Detection: A Survey.* TKDE 19, 1 (Jan. 2007), 1-16.
[2] Mikhail Bilenko and Raymond J. Mooney. 2003. *Adaptive Duplicate Detection Using Learnable String Similarity Measures.* KDD.
[3] AnHai Doan et al Chaitanya Gokhale, Sanjib Das. 2014. *Corleone: Hands-off Crowdsourcing for Entity Matching.* SIGMOD.
[4] Peter Christen. 2012. *Data Matching.* Springer.
[5] Jens Dittrich. 2017. *Deep Learning (m)eats Databases.* VLDB Keynote.
[6] Ian Goodfellow et al. 2016. *Deep Learning.* MIT press.
[7] Matthew Francis-Landau et al. 2016. Capturing semantic similarity for entity linking with convolutional neural networks. CoRR abs/1604.00734 (2016).
[8] Parag Singla et al. 2006. *Entity Resolution with Markov Logic.* ICDM.
[9] Wei Wang et al. 2016. *Database Meets Deep Learning: Challenges and Opportunities.* ACM SIGMOD Record 45, 2 (2016), 17-22.
[10] Wenpeng Yin et al. 2016. *Simple Question Answering by Attentive Convolutional Neural Network.* COLING.
[11] Avigdor Gal et al George Papadakis, Jonathan Svirsky. 2016. *Comparative Analysis of Approximate Blocking Techniques for Entity Resolution.* VLDB.
[12] Lise Getoor and Ashwin Machanavajjhala. 2012. *Entity Resolution: Theory, Practice Open Challenges.* VLDB.
[13] Elad Hoffer and Nir Ailon. 2015. *Deep metric learning using triplet network.* In International Workshop on Similarity-Based Pattern Recognition. Springer.
[14] Li H et al Hu B, Lu Z. 2014. Convolutional neural network architectures for matching natural language sentences. Advances in Neural Information Processing Systems. 2014: 2042-2050.
[15] Michael J. Franklin Jiannan Wang, Tim Kraska and Jianhua Feng. 2012. *CrowdER: Crowdsourcing Entity Resolution.* VLDB.
[16] Blunsom P Kalchbrenner N, Grefenstette E. 2014. A convolutional neural network for modelling sentences. arXiv preprint arXiv:1404.2188, 2014.
[17] Ihab F. Ilyas et al Michael Stonebraker, Daniel Bruckner. 2013. *Data Curation at Scale: The Data Tamer System.* CIDR.
[18] Chen K Corrado G. Dean J Mikolov, T. 2010. Efficient Estimation of Word Representations in Vector Space. Athttp://arxiv.org/abs/1301.3781.
[19] Shafiq Joty Mourad Ouzzani Muhammad Ebraheem, Saravanan Thirumuruganathan and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. PVLDB, 11 (11): 1454-1467, 2018.
[20] Felix Naumann and Melanie Herschel. 2010. *An Introduction to Duplicate Detection.* Morgan and Claypool Publishers.
[21] Tim Lillicrap Daan Wierstra et al Oriol Vinyals, Charles Blundell. 2016. *Matching networks for one shot learning.* ACL.
[22] Maarten Versteegh Paul Neculoiu and Mihai Rotaru. 2016. *Learning text similarity with siamese recurrent networks.* ACL.
[23] Ahmed Elmagarmid et al Rohit Singh, Vamsi Meduri. 2017. *Generating Concise Entity Matching Rules.* SIGMOD.
[24] P. S. G. C. C. Gokhale S. Das, A. Doan and P. Konda. 2017. The magellan data repository. https://sites.google.com/site/anhaidgroup/useful-stuff/data.
[25] Jürgen Schmidhuber. 2015. *Deep learning in neural networks: An overview.* Neural networks 61 (2015), 85-117.
[26] Theodoros Rekatsinas Sidharth Mudgal, Han Li. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *SIGMOD 18: 2018 International Conference on Management of Data, June 10-15, 2018.* ACM, New York, 16 pages.
[27] Jianzhong Li Wenfei Fan, Xibei Jia and Shuai Ma. 2009. *Reasoning About Record Matching Rules.* VLDB.
[28] Bing Xiang Bowen Zhou Wenpeng Yin, Hinrich Schutze. 2016. ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs. arXiv:1512.05193v4.
[29] Hinrich Schütze Wenpeng Yin. 2015. Convolutional Neural Network for Paraphrase Identification. The 2015 Conference of the North American Chapter of the Association for Computational Linguistics.
[30] Wikipedia. 2018. *Word embedding.* https://en.wikipedia.org/wiki/Word_embedding.
[31] Kim Y. 2014. Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882, 2014.
[32] Yoshua Bengio Yann LeCun and Geoffrey Hinton. 2015. *Deep Learning.* Nature 521, 7553 (2015), 436-444.
[33] Christian Borgs Rainer Schnell Ziad Sehili, Lars Kolb and Erhard Rahm. 2015. *Privacy Preserving Record Linkage with PPJoin.* BTW.