Deepmatcher: Deep Learning for Entity Matching

Belotti Federico

DISCo, Università degli Studi di Milano-Bicocca

July 15, 2019

Overview

1. Introduction
2. Problem setting
3. Regime-matching offers to Ceneje product
4. Regime-matching new Ceneje products in seller's offers
5. Model architecture
6. Results
7. Future developments

Introduction I

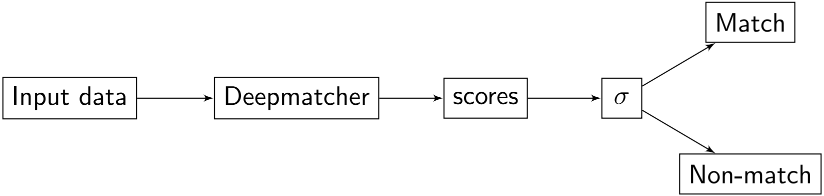
Entity matching (EM) finds data instances referring to the same real-word entity

The most popular type of EM problems are:

- **Clean structured data**: attributes values short and atomic, no missing one
- **Dirty structured data**: some attributes values are missing, and may appear in another attributes cell
- **Textual data**: attributes values correspond to long spans of text (i.e. product description)

Introduction II

Deepmatcher is framework for performing entity matching using deeplearning techniques. It can be summarized as follow:



where σ can be a simple *threshold*, an *arg-max* or a *top-k arg-max* (for human-in-the-loop support)

Problem setting I

We define:

- **Entity**: distinct real world object (i.e. person, organization, ...)
- **Entity mention**: reference to a real-world entity (i.e. record in a dataset, ...)

General problem statement:

- Given two collections D and D' of entity mentions, following the same representation (the same schema with attributes $A_1, ..., A_N$), find all pairs between D and D' that refer to the same real-world entity

Problem setting II

For Ceneje we have:

- **Products**: distinct products in Ceneje catalog
- **Offers**: offers from different sellers that maps to a Ceneje product

Ceneje specific problem statement:

- Given a collection O of seller's offers and a collection P of distinct products, following the same representation (the same schema with attributes $A_1, ..., A_N$), find for every offer in O which Ceneje's product refers to in P

Problem setting III

EM is typically done in two phases:

- **Blocking**: filter out the cross-product $O \times P$ to a candidate set C, containing matching and non-matching tuples
- **Matching**: identify true entity mentions

Deepmatcher works only on the **matching phase**, so:

- Given $e_1 \in O, e_2 \in P$ and the labeled data $T = \{(e_1^i,\ e_2^i,\ label)\} \subseteq C \times \{\text{match}, \text{non-match}\}$, use T as labeled training data to learn a matcher M that classifies pairs of entity mentions in C as match or non-match

Production-matching offers to Ceneje product

At production stage given an offer o, will be created pairs from $o \times P_i$, where i is the category which o belongs to (categorization/blocking phase)

$$\boxed{\begin{array}{c}(o, p_{i_1})\\(o, p_{i_2})\\(o, p_{i_3})\\\dots\\(o, p_{i_n})\end{array}} \longrightarrow \boxed{\text{Deepmatcher}} \longrightarrow \boxed{\text{scores}} \longrightarrow \boxed{\text{arg-max}} \longrightarrow \boxed{\begin{array}{c}p_{j_1}\\p_{j_2}\\\dots\\p_{j_k}\end{array}}$$

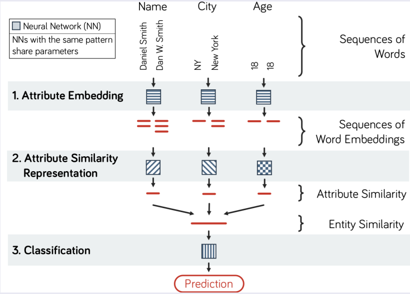
Regime-matching new Ceneje products in seller's offers

At production stage it can happen that some offers refer to no Ceneje product, i.e. are new products

Deepmatcher can be used to cluster those new offers based on the similarity scores, and with the human-in-the-loop support select for every cluster a representative offer to add as new Ceneje product

Model architecture I

Model architecture template



Input

Vector of N entries, one for each attribute $A_i \in \{A_1, \ldots, A_N\}$, where each entry i corresponds to a pair of word sequences $(\mathbf{w}_{e_1,i},\ \mathbf{w}_{e_2,i})$, i.e. attribute value for entity e_1 and e_2 respectively

Model architecture II

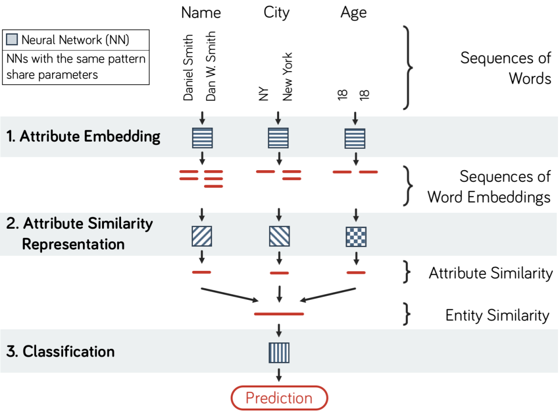
Example Input:

t_1

Name	City	Age
Daniel Smith	NY	18

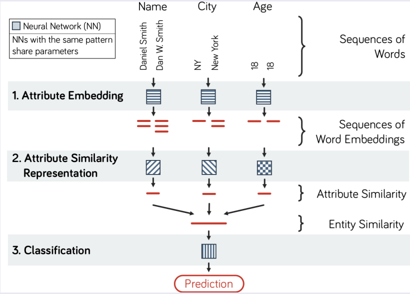
t_2

Name	City	Age
Dan W. Smith	New York	18



Model architecture III

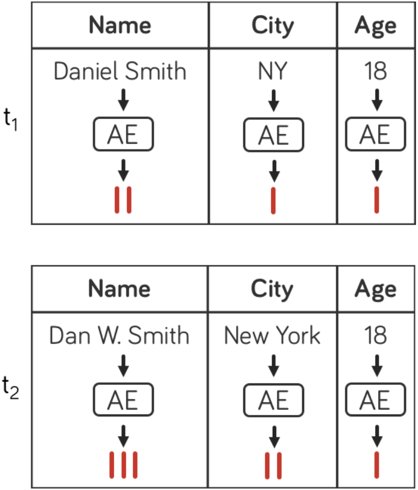
Model architecture template



Attribute Embedding

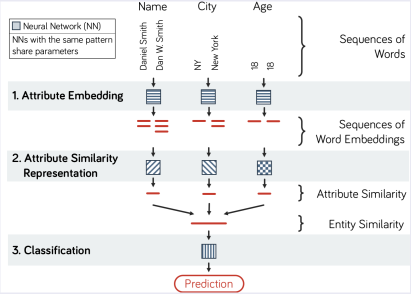
Given a pair $(\mathbf{w}_{e_1,i},\ \mathbf{w}_{e_2,i})$, this module transform them to two sequences of word embeddings vectors $(\mathbf{u}_{e_1,i},\ \mathbf{u}_{e_2,i})$

Model architecture IV



Model architecture V

Model architecture template

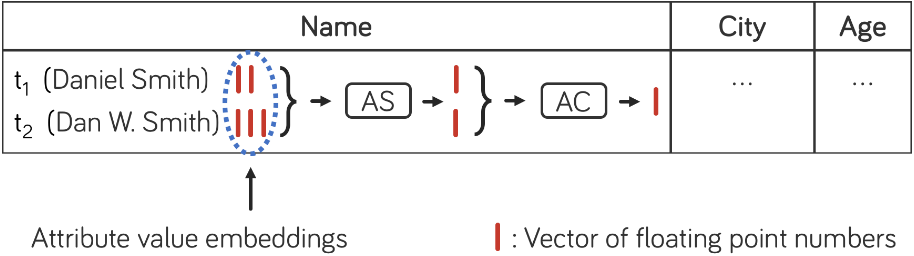


Attribute Similarity Representation

Given the attribute embbedings pair $(\mathbf{u}_{e_1,i},\ \mathbf{u}_{e_2,i})$, encode them to a representation that captures attribute value similarity of e_1 and e_2. Two main operations performed:

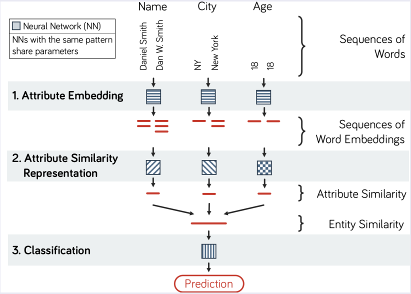
- *Summarization*: aggregate info across all entries in the attribute value embedding and output a summary vectors $(\mathbf{s}_{e_1,i},\ \mathbf{s}_{e_2,i})$
- *Comparison*: apply a comparison function on the summary vectors to obtain a similarity representation s_i, one for each attributes

Model architecture VI



Model architecture VII

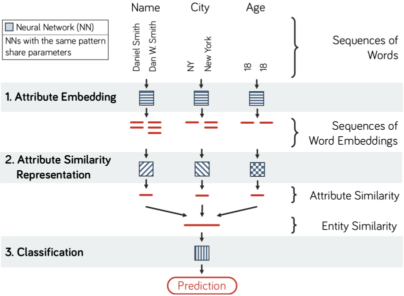
Model architecture template



Classification

Given the similarity representations $(s_1, \ldots, s_N)$, determines if the input entity mentions e_1 and e_2 refer to the same real-world entity

Model architecture VIII



Architecture module		Options	
Attribute embedding		*Granularity:* (1) Word-based (2) Character-based	*Training:* (3) Pre-trained (4) Learned
Attribute similarity representation	(1) Attribute summarization	(1) Heuristic-based (2) RNN-based (3) Attention-based (4) Hybrid	
	(2) Attribute comparison	(1) Fixed distance (cosine, Euclidean) (2) Learnable distance (concatenation, element-wise absolute difference, element-wise multiplication)	
Classifier		NN (multi-layer perceptron)	

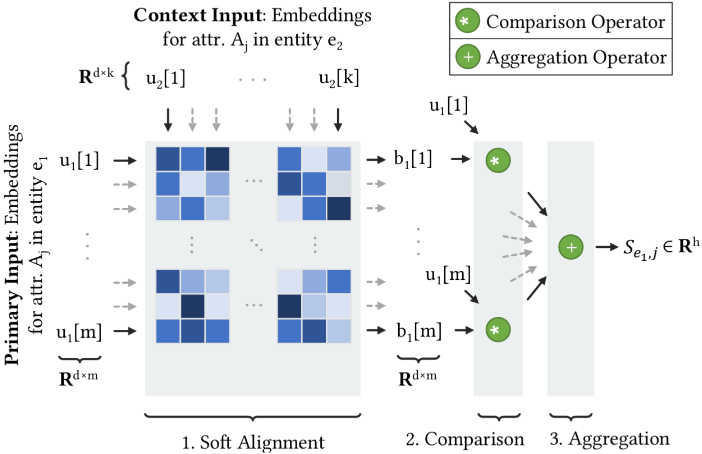
Model architecture IX

1. Smooth Inverse Frequency (SIF): Weighted average $w(s) = a/(a + f(s))$
2. Bidirectional Recurrent Neural Network (RNN)
3. Attention Model: Embedding with context
4. Hybrid: RNN+Attention

	SIF	RNN	Attention	Hybrid
Order awareness	$\times$	$\checkmark$	$\checkmark$	$\checkmark$
Soft Alignment	$\times$	$\times$	$\checkmark$	$\checkmark$
Time	$\checkmark\checkmark$	$\checkmark$	$\times$	$\times$

Model architecture X

Attention Model



Model architecture XI

- For a complete review, please refer to: Deep Learning for Entity Matching: A Design Space Exploration[1]
- Complete Python implementation: https://github.com/anhaidgroup/deepmatcher
- Updated Python implementation: https://github.com/belerico/deepmatcher/tree/torch_1.0.1

[1] Sidharth Mudgal et al. "Deep Learning for Entity Matching: A Design Space Exploration". In: *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. Ed. by Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein. ACM, 2018, pp. 19–34. DOI: 10.1145/3183713.3196926. URL: https://doi.org/10.1145/3183713.3196926.

Datasets I

- **Seller products data**: contains products data from sellers; there is one dataset for every product category. Important attributes taken into accounts in this experiment:
  - *Brand*, e.g. "philips"
  - *Name*, e.g. "tv sprejemniki 65pus650312 philips"
  - *Description*, e.g. "opis izdelka izjemno tanek ledtelevizor 4k uhd smart pixel precise uhd sistemom saphi uivajte jasni loljivosti 4k uhd ..."
- **Ceneje products**: name brand about Ceneje's products; there is a unique instance of every product.
- **Seller products mapping**: mapping from sellers' to Ceneje's products ad matched by Ceneje
- **Ceneje attributes**: attributes for every Ceneje products

Datasets II

From seller products mapping we can obtain, for every category i, the matching offers, so:
Let $M_i = \{product_j | (product_j, product_i) \text{ is a matching offer}, i \neq j\}$
Let $U_i = \bigcup_{i=1}^{N} M_i$ be the set of all matching offers for the category i

Matching and non-matching tuples are created this way:

- **Match**: $\forall m \in M_i$, create pair combination
- **Non match**: $\forall m \in M_i$, sample at random $K(\alpha)$ products from $U_i \setminus M_i$ and create pairs (m, p_{i_1}), (m, p_{i_2}), ..., $(m, p_{i_{K(\alpha)}})$, where $K(\alpha)$ is such that we end up having $\alpha \cdot \binom{|M_i|}{2}$ negative examples

Datasets III

For example, given this matching class M_i

id	Brand seller	Name seller	Desc seller
2368		liebherr sbsesf 7212	elektronika magiceye...
2369		liebherr hladilnik zamrzovalnik sbsesf 7212	side by side kombinacija tehnine...
2370	liebherr	liebherr hladilnik zamrzovalnik liebherr sbsesf 7212	side by side kombinacija

The following pairs will be generated

Left id	Right id	label	Left brand seller	Left name seller	Left desc seller	Right brand seller	Right name seller	Right desc seller
2368	2369	1	...	...	...	...	...	...
2368	2370	1	...	...	...	...	...	...
2369	2370	1	...	...	...	...	...	...
2368	4321	0	...	...	...	...	...	...
2368	1243	0	...	...	...	...	...	...
2369	3218	0	...	...	...	...	...	...
2369	232	0	...	...	...	...	...	...
2370	872	0	...	...	...	...	...	...
2370	241	0	...	...	...	...	...	...

Datasets IV

- 4 different categories: *led TV, monitor, refrigerator and washing machine*
- Non-matching tuples are created such that we have a positive:negative ratio of 1:2 ($\alpha = 2$)
- Data is splitted in training, validation, test and unlabeled as 50%, 20%, 20% and 10% (unlabeled is needed by Deepmatcher to predict scores)
- Data are pairs of matching and non-matching offers

Split	#Pos	#Neg	Total
Train	18,966	38,348	57,314
Validation	8,016	15,865	23,881
Test	7,928	15,954	23,882
Unlabeled	4,892	9,436	14,328

Model architecture for Ceneje

Deepmatcher options for the architecture module:

- **Attribute embedding**: pretrained Slovenian fasttext vectors
- **Attribute summarization**:
  - bidirectional RNN with GRU
  - Self attention
  - Hybrid
- **Attribute comparison**: element-wise absolute difference
- **Classifier**: default Neural Network

Results I

- Naive baseline: attributes values concatenation

Similarity	Precision	Recall	F1	Time (min)
Edit	0.97	0.08	0.15	≈ 9
Jaccard (word level)	0.98	0.09	0.18	≈ 1
Jaccard (2-grams)	0.67	0.19	0.30	≈ 1
Jaccard (3-grams)	0.97	0.10	0.18	≈ 1
Jaccard (5-grams)	0.99	0.09	0.16	≈ 1

Results II

- Weighted attribute similarity: $\frac{1}{N}\sum_{i=1}^{N} w_i \cdot similarity(e^j_{1,i}, e^j_{2,i})$, where N is the number of attributes and $e^j_{1,i}, e^j_{2,i}$ are the *i-th* attribute of the *j-th* pair

Similarity	**Precision**	**Recall**	**F1**	**Time (min)**
Edit	0.82	0.58	0.68	≈ 8
Jaccard (word level)	0.86	0.55	0.67	≈ 0.7
Jaccard (2-grams)	0.83	0.73	0.78	≈ 0.7
Jaccard (3-grams)	0.86	0.56	0.68	≈ 0.7
Jaccard (5-grams)	0.86	0.44	0.58	≈ 0.7

$w_i = 1$ for every attribute
If the attribute value of one pair is null, the computed score is $1/2$

Results III

- Deepmatcher:

Method	Precision	Recall	F1	Time per epoch (min)
RNN	0.98	0.99	0.99	< 6
Attention	0.86	0.98	0.91	≈ 6
Hybrid	0.97	0.99	0.98	≈ 18

Future developments

- Take into account Ceneje attributes and prices
- Try different deepmatcher architecture options
- Make Deepmatcher job more difficult:
    - Create non-matching tuples that are very similar
- Try a logistic/linear regression to learn weights w_i
- Simulate a production environment to get the best matching product from Ceneje catalog for an offer o

The End