A Novel Ensemble Method for Named Entity Recognition and Disambiguation Based on Neural Network

Lorenzo Canale[1,2], Pasquale Lisena[1], and Raphaël Troncy[1(✉)]

[1] EURECOM, Sophia Antipolis, France
{canale,lisena,troncy}@eurecom.fr
[2] Politecnico di Torino, Turin, Italy

Abstract. Named entity recognition (NER) and disambiguation (NED) are subtasks of information extraction that aim to recognize named entities mentioned in text, to assign them pre-defined types, and to link them with their matching entities in a knowledge base. Many approaches, often exposed as web APIs, have been proposed to solve these tasks during the last years. These APIs classify entities using different taxonomies and disambiguate them with different knowledge bases. In this paper, we describe Ensemble Nerd, a framework that collects numerous extractors responses, normalizes them and combines them in order to produce a final entity list according to the pattern (surface form, type, link). The presented approach is based on representing the extractors responses as real-value vectors and on using them as input samples for two Deep Learning networks: ENNTR (Ensemble Neural Network for Type Recognition) and ENND (Ensemble Neural Network for Disambiguation). We train these networks using specific gold standards. We show that the models produced outperform each single extractor responses in terms of micro and macro F1 measures computed by the GERBIL framework.

1 Introduction

A crucial task in knowledge extraction from textual document consists in the two complementary tasks of Named Entity Recognition (NER) and Named Entity Disambiguation (NED), achieving the goal of assigning to parts of text (tokens) respectively a type – from a pre-defined taxonomy – and a unique identifier – normally in the form of URI – that points univocally to the referred entity in a given knowledge base. The combination of these two tasks is often abbreviated with the acronym NERD [5,6]. The current state of the art offers an interesting number of NERD extractors. Some of them can be trained by a developer on his own corpus, while other ones are only accessible as black-box services exposed via web APIs offering a limited number of parameters.

In terms of NER, each service provides generally its own taxonomy of named entity types which can be recognised. While they all provide support for three major types (person, organization, location), they largely differ for more fine-grained types which makes hard their comparison and combination. In terms

D. Vrandečić et al. (Eds.): ISWC 2018, LNCS 11136, pp. 91–107, 2018.
https://doi.org/10.1007/978-3-030-00671-6_6

of NED, each extractor can potentially disambiguate entities against specific knowledge bases (KB), but in practice, they mostly rely on popular ones, namely DBpedia, Wikidata, Freebase or YAGO. For this reason, comparing and merging the results of these extractors require some post-processing tasks that typically rely on mappings between those KBs. This task is however simpler than the type alignment, because of the large presence of `owl:sameAs` links between the different KBs.

In this paper, we present **Ensemble Nerd**, a multilingual ensemble method that combines the responses of different NERD extractors. This method relies on a real-value vectorial representation as input samples for two Deep Learning networks, ENNTR (Ensemble Neural Network for Type Recognition) and ENND (Ensemble Neural Network for Disambiguation). The networks provide models for performing type alignment and named entity linking to a knowledge base. This strategy is evaluated against some well-known gold standards, showing that the output of the ensemble outperforms the results of single extractors.

This work aims to answer the following research questions: Can we define an ensemble method that combines the extractors responses in order to create a new more powerful extractor? Is it possible to define an ensemble method that avoids a type alignment step or that computes it automatically, without any human intervention? Which ensemble method should be adopted to exploit all the collected information? Considering that extractors return list of named entities – together with the type and the disambiguation link of each of them –, how this data can be numerically represented? Can we better understand which features contribute more to improve the ensemble output response? How dependant is this feature selection of the corpora, language, entity types and what is the influence of the KB?

The remainder of this paper is organised as follows: Sect. 2 describes some related work. Section 3 details how we represent the extractors responses, while Sect. 4 presents the core of the ensemble method. An evaluation is proposed in Sect. 5, while conclusion and and future work are discussed in Sect. 6.

2 State of the Art

Ensemble methods for the NER and NED tasks have already largely been studied in the literature. The **NERD** framework [5,6] allows to compare and evaluate some of the most popular named entity extractors. It can analyse any textual resource published on the web and to extract the named entities that are detected, typed and disambiguated by various named entity extractor APIs. For overcoming the different type taxonomies, the authors designed the *NERD ontology* which provides a set of mappings between these various classifications and consequently makes possible an evaluation of the quality of each extractor. This task was originally a one time modeling exercise: the authors manually mapped the different taxonomies to the NERD ontology.

NERD-ML, a machine learning approach developed on top of the NERD framework, combines the responses of single extractors applying alternatively three different algorithms: Naive Bayes (NB), k-Nearest Neighbours (k-NN) and Support Vector Machines (SVM) [6,11]. It is a more sophisticated and robust approach that uses machine learning inductive techniques for passing from the output type of single extractors to the right entity type in a normalized types set, i.e. the NERD Ontology [7]. **FOX** [9,10] is a framework that relies on ensemble learning by integrating and merging the results of four NER tools: the **Stanford Named Entity Recognizer** [3], the **Illinois Named Entity Tagger** [4], the **Ottawa Baseline Information Extraction** (Balie) and the **Apache OpenNLP Name Finder**. FOX compares the performance of these tools for a small set of classes namely LOCATION, ORGANIZATION and PERSON. For achieving this goal, the entity types of each NER tools is mapped to these three classes. Given any input text t, FOX processes t with each of the n tools it integrates. The result of each tool T_i is a piece of annotated text t_i, in which either a specific class or zero (not belonging to the label of a named entity) is assigned to each token. The tokens in t are then represented as vectors of length n and are used for getting the final type. The author demonstrates that a Multi-Layer Perceptron (MLP) gets the best results among a pool of 15 different algorithms [9].

3 Feature Engineering for NERD

Ensemble Nerd currently integrates a set of 8 extractors shown in Table 1. An extractor can belong to the set T (extractors that perform NER task) or to the set U (extractors that perform NED task). Currently, *TextRazor* is the only one in both sets: $T \cap U = \{TextRazor\}$. All these extractors relies on Wikidata, Wikipedia or DBpedia for entity disambiguation.

Each extractor produces a list of named entities as response for a specific input text. From this output, we generate 4 different kinds of feature.

1. Surface form features. They are strictly related to the text used to extract named entity. The input text is split into tokens and a word embedding

Table 1. Extractor included in Ensemble Nerd. ✓ indicates that the extractor supports the action (type recognition or named entity disambiguation)

Extractor	Type recognition	NE disambiguation
AlchemyAPI	✓	✗
DandelionAPI	✗	✓
DbSpotlight	✗	✓
TextRazor	✓	✓
Babelfy	✗	✓
MeaningCloud	✓	✗
ADEL	✓	✗
OpenCalais	✓	✗

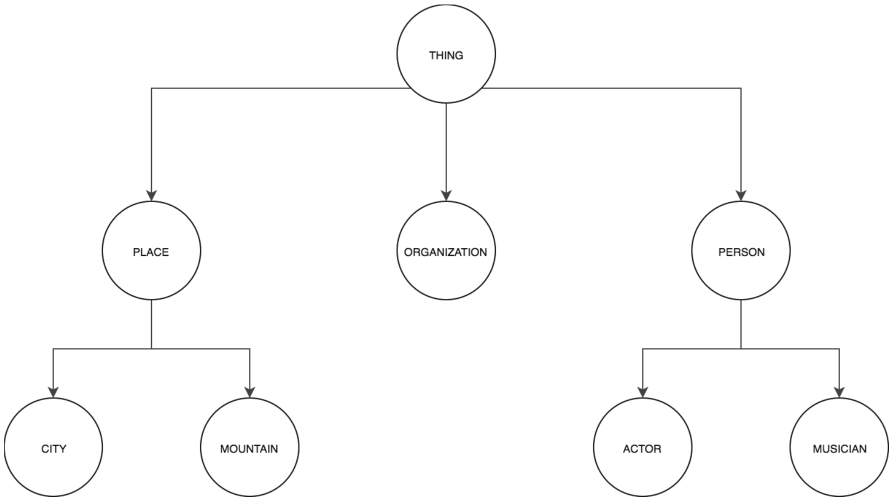

Fig. 1. Example of type taxonomy for a generic extractor.

representation is assigned to each of them. We consider also the stop words, assigning also to them a real-value vectorial representation. The word vectors are computed using *fastText* [1]. We define $\boldsymbol{s^x}$ as the real-valued vector associated to a specific token x:

$$\boldsymbol{s^x} = \left[\boldsymbol{s^x_p}|\boldsymbol{s^x_c}\right], dim(\boldsymbol{s^x}) = 400 \tag{1}$$

where | (pipe) is the concatenation operator and dim is the vector dimension.

$\boldsymbol{s^x_p}$, $dim(\boldsymbol{s^x_p}) = 300$, consists in the token embedding computed using the Wikipedia pre-trained *fastText* models released by the authors. The model changes depending on the language used in the text, since all localised Wikipedia have been used to train language specific models.

$\boldsymbol{s^x_c}$, $dim(\boldsymbol{s^x_c}) = 100$, is the token embedding computed when training *fastText* directly on a particular textual corpus – i.e. the one for which we want to perform the NERD tasks. This means that $\boldsymbol{s^x_c}$ does not vary depending on the language but on the gold standard itself.

2. Type features. Each extractor $e \in T$ has its own type taxonomy o which is a taxonomy of a maximum depth L. In the following, we consider a simple example of an taxonomy o with just a 2 levels hierarchy (Fig. 1):

1. Level 1 includes three types: PLACE, ORGANIZATION and PERSON.
2. Level 2 includes four types: CITY and MOUNTAIN (subtypes of PLACE) and ACTOR and MUSICIAN (subtypes of PERSON).

We name C_i the number of different types inside the level i (e.g. $C_1 = 3$). We infer a one-hot encoding representation for each level as shown in Table 2.

For a generic type τ in the last layer (e.g. ACTOR), the features vector v_τ consists in the concatenation of the one-hot representation of each type founded

Table 2. Representation of types through one-hot encoding.

Level 1		Level 2	
Type	Representation	Type	Representation
PERSON	001	ACTOR	0001
ORGANIZATION	010	MUSICIAN	0010
PLACE	100	CITY	0100
		MOUNTAIN	1000

on the walk from the root to the leaf associate to τ. The features vector for ACTOR is therefore `0010001`, where the first three values `001` derive from PERSON and the last four values `0001` derive from ACTOR. Hence, we can state that $dim(v_\tau) = \sum_i^L C_i$. If the extractor $e \in T$ returns a type that is not the last level in the hierarchy, as PERSON, we fill the missing vector positions with `0`. The features vector v_{PERSON} associated to PERSON is thus `0010000`. This mechanism is extensible to any taxonomy. However the $dim(v_\tau)$ is different for each extractor, depending on the taxonomy that it uses.

This procedure can be extended also to extractors that do not perform NER. A generic extractor e, where $e \in U \wedge e \notin T$, returns a link for each entity. Following the interlinks between KBs, we can always obtain an entity in Wikidata. The type of the entity would be the class of this entity in Wikidata, which is the value of the property *instance of (P31)*[1]. Entities might possess multiple types and for this reason they are represented through K-hot encoding.

For a **typed named entity** w^t with the format `(surface form, type)`, the type feature vector $\boldsymbol{v}_{\boldsymbol{e}}^{\boldsymbol{w}^t}$ is computed for the extractor e where $e \in U \vee e \in T$. $dim(\boldsymbol{v}_{\boldsymbol{e}}^{\boldsymbol{w}^t})$ varies accordingly to the considered extractor. In fact, we get a real-value numerical type representation without a type alignment phase. For this reason, the number of dimensions that forms the type features vector depends on the the number of types in the extractor taxonomy.

3. Entity features. These features represent the similarity between two Wikidata entities w_1 and w_2, as a vector of 5 dimensions. The first four dimensions correspond to **semantic knowledge:**

1. the first dimension $S_{uri}(w_1, w_2)$ indicates if the compared entities share the same URI with a Boolean;
2. the second dimension provides the string similarity between the labels l_{w_1} and l_{w_2} associated to the compared entities:

$$S_{Lev}(w_1, w_2) = max(1 - d_{Lev}(l_{w_1}, l_{w_2})/\beta, 0), \beta = 8$$

where $d_{Lev}(l_{w_1}, l_{w_2})$ is the **Levenshtein distance** between the compared strings and β is a constant equals to the number of maximum differences after which the similarity is saturated to 0.

[1] https://www.wikidata.org/wiki/Property:P31.

3. the third dimension $S_{TfIdf}(w_1, w_2)$ represents the **TF-IDF Cosine Similarity** between the abstracts associated to the compared entities. This dimension represents a textual knowledge as in [12];
4. the fourth dimension $S_{occ}(w_1, w_2)$. value indicates if the compared entities share the same *occupation (P106)*.[2] This property is specific for entities of type PERSON: this Wikidata class has no other subclasses, as opposed to the other types. For this reason this similarity dimension greatly helps in the disambiguation of people with similar names but different professions. $S_{occ}(w_1, w_2)$ is set to 1 when the two entities referred to people that have the same profession, and 0 otherwise (different profession or not a PERSON).

The fifth and last dimension of the vector represents the structural similarity as in [12]. We define a property set P, containing three properties: *subclass of (P279)*[3], *instance of (P31)*[4], and *part of (P361)*[5]. A subgraph G is extracted from Wikidata selecting all the triples in which a property in P appears. We define the distance d_{w_1,w_2} between two generic entities w_1 and w_2 as the shortest path length that links w_1 and w_2 in G. Then, we compute the maximum distance between two nodes in the graph G, defining it as d_{max}. We assess the structural similarity between w_1 and w_2 as:

$$S_{stc}(w_1, w_2) = -\frac{d_{w_1,w_2}}{d_{max}} + 1$$

The total similarity between w_1 and w_2 can be expressed as:

$$\boldsymbol{S(w_1, w_2)} = [S_{uri}(w_1, w_2), S_{Lev}(w_1, w_2), S_{TfIdf}(w_1, w_2), S_{occ}(w_1, w_2), S_{stc}(w_1, w_2)] \tag{2}$$

The choice of representing the similarity between two entities as a real-value vectors rather than using an entity embedding is in line with our goal of representing how the extractors differ in the prediction rather than directly representing an entity. This approach avoids to compute embeddings on the whole Wikidata KB. We rely on interlinks between KBs for guaranteeing that we can always compare Wikidata entities. This causes the risk that no Wikidata entity exists for the source one, i.e. because the information is not present. However, this case is very rare (Table 3) in all the considered benchmarks in the evaluation, thanks to the reliance of all the involved extractors on Wikidata, Wikipedia or DBpedia, which containing similar information. This would become a limit when using different KBs (e.g. thematic ones), not fully interlinkable to Wikidata and for which a loss in information should be taken in account.

4. Score features. Some extractors return scores representing either the confidence or the saliency for each named entity. For each extractor $e \in K$, w^k is

[2] https://www.wikidata.org/wiki/Property:P106.
[3] https://www.wikidata.org/wiki/Property:P279.
[4] https://www.wikidata.org/wiki/Property:P31.
[5] https://www.wikidata.org/wiki/Property:P361.

Table 3. Coverage of matching against Wikipedia of disambiguated entity in the ground truth.

Extractor	Disambiguation KB	WD coverage
Dandelion	Wikipedia	99%
DBSpotlight	DBpedia Fr	98%
TextRazor	Wikidata	100%
Babelfy	DBpedia	100%

a **named entity score** with the format `(surface form, scores)`. We define $\boldsymbol{v}_{\boldsymbol{e}}^{\boldsymbol{w}^k}$ as the features vector representing the scores for w^k and the extractor e. $dim(\boldsymbol{v}_{\boldsymbol{e}}^{\boldsymbol{w}^k})$ depends on the considered extractors, more precisely on the number of scores returned by it.

4 Ensemble NERD: ENNTR and ENND

Our experimental ensemble method relies on two Neural Networks that receive in input the features described in the previous Section. We respectively name them with the acronyms **Ensemble Neural Network for Type Recognition (ENNTR)** and **Ensemble Neural Network for Disambiguation (ENND)**. For both networks, the hyper parameter optimization was done using Grid Search.

These networks architectures come after a series of previous experiments that involved LSTM and BiLSTM, receiving a complete vector including all the features as input sample. A really slow training, the ease of network overfitting to the sample input, and huge difference in dimensionality (and so in impact to the results) between the different features were some of the reasons for which we have abandoned these approaches.

Ensemble Neural Network for Type Recognition (ENNTR). We consider a generic ground truth GT formed by N textual fragments (e.g. sentences), such that we can split each fragment in tokens. X_i is the ordered list of tokens for fragment i. Concatenating the lists X_i, we get a list X, that is the ordered list of tokens for the whole corpus. We call x a generic token in X.

GT associates a type in a taxonomy o_{Gt} to each token x. We identify the neural network target as Y_t. The number of samples in Y_t is equal to the total number of tokens: $dim(Y_t) = dim(X)$. The neural network goal is to assign the right type to each token and its architecture is represented in Fig. 2.

ENNTR has an output layer O formed by $H = card(o_{GT})$ neurons, where $card(o_{GT})$ is the number of different types (or cardinality) in o_{GT}. As a consequence, each value returned by a neuron in the output layer corresponds to the probability that a token x belongs to a specific type. Hence, each target sample $\boldsymbol{y_t}$ is a vector formed by H values, where each value corresponds to a type and a neuron. In Fig. 2, we are assuming that $H = 4$.

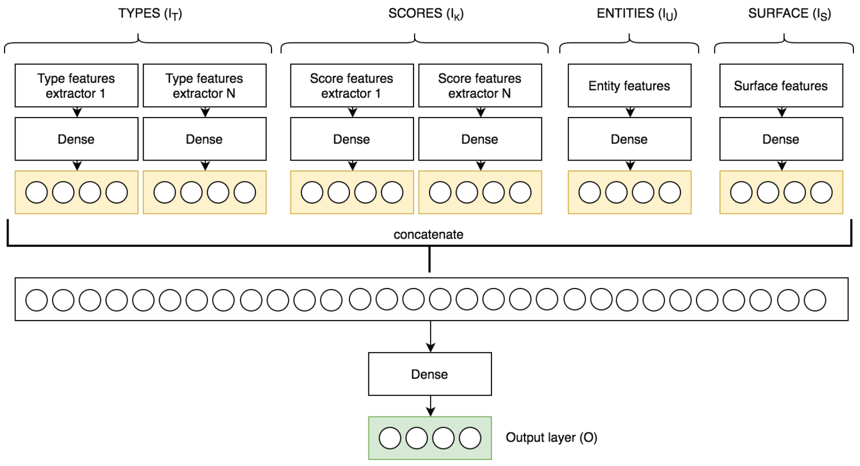

Fig. 2. ENNTR architecture

ENNTR presents many input layers. Using the same notation used in Sect. 3, T is the set of extractors that return type information, K is the set of extractors that return score information, U is the set of extractors that perform disambiguation. Defining I as the set of input layers of ENNTR, we can identify four different types of input layer depending on the kind of features being input.

$$I = I_T \cup I_K \cup I_U \cup I_S$$

$$|I| = |I_T| + |I_K| + |I_U| + |I_S| = |T \cup U| + |K| + 1 + 1$$

All the input layers works at token level, so that the features at entity level defined in Sect. 3 requires a transformation to token-level. The surface form of an entity w (e.g. *Barack Obama*) can be tokenised, producing the list of tokens X_w (e.g. *[Barack, Obama]*). The feature vector of token x is equal to the one of an entity w if x is a token in X_w. Otherwise it is equal to a padding vector $\boldsymbol{d}$, of the same dimension and containing only `0` values.

In particular, I_T receives in input a type features vector $\boldsymbol{t}_e^{\boldsymbol{x}}$, computed like:

$$\boldsymbol{t}_e^{\boldsymbol{x}} = \begin{cases} \boldsymbol{v}_e^{\boldsymbol{w}^t} & if\ x \in X_{w^t} \\ \boldsymbol{d}_t & if\ x \notin X_{w^t} \end{cases} \tag{3}$$

$$\boldsymbol{d}_t = [0, ..., 0], dim(d_k) == dim(\boldsymbol{v}_e^{\boldsymbol{w}^t})$$

Similarly, I_K receives in input a type features vector $\boldsymbol{k}_e^{\boldsymbol{x}}$, computed like:

$$\boldsymbol{k}_e^{\boldsymbol{x}} = \begin{cases} \boldsymbol{v}_e^{\boldsymbol{w}^k} & if\ x \in X_{w^k} \\ \boldsymbol{d}_k & if\ x \notin X_{w^k} \end{cases} \tag{4}$$

$$\boldsymbol{d}_k = [0, ..., 0], dim(d_k) == dim(\boldsymbol{v}_e^{\boldsymbol{w}^k})$$

The Wikidata entity u_e^x for the token x is:

$$u_e^x = \begin{cases} u_e^{w^u} & if\ x \in X_{w^u} \\ NAN & if\ x \notin X_{w^u} \end{cases} \tag{5}$$

The layers I_U receive in input the entity features vector $\boldsymbol{u^x}$, computed for a token x as:

$$\boldsymbol{u^x} = [S(u_1^x, u_1^x), S(u_1^x, u_2^x), ..., S(u_P^x, u_P^x)]$$

Finally, the input layers I_S receive the surface features vector $\boldsymbol{s^x}$ without any further transformation.

Each input layer I_n is fully connected with a layer M_n. M_n, like O, is composed by H neurons, where H is the number of types in the ground truth. The activation of neurons in M_n is linear.

In this first part of the network, each I_n – composed by a different number of neurons depending on the related features vector – is mapped on H neurons in M_n. This avoids that the neural network privileges features vectors with higher dimension – it happens directly concatenating different features vectors. This part of the network can be considered as an **alignment block** since it automatically map the types between the extractors and the ground truth taxonomy. This is pretty similar to the *Inductive Entity Typing Alignment* work described in [7], with the difference that the alignment step is learned by a fully connected layer. Differently from previous works [9,10], the approach does not need any preliminary alignment and recognition, because they are part of the same network.

The last part of the network is the **ensemble block**. M_k layers are concatenated forming a new layer R. $|o_{GT}|$ is the number of types in the ground truth, $|I|$ the number of input layers and $|P|$ the number of neurons in R:

$$|P| = |o_{GT}| \cdot |I|$$

R is fully connected to the output layer O. The activation of the neurons in O is linear. This means that ENNTR finally consists in a linear combinations of features: the key is the way in which the features are generated and entered in the network. The values v_h of the H output neurons in O correspond to the probability that a given type is correct. We take the highest value v_{max} between them and if it is greater than a threshold θ, we set the type related to its neuron as the predicted one. The final output of the ensemble method is a list of predicted type l_p for each token x. In a final step, sequences of token which belong to the same type are merged to a single entity, similarly to [9,10].

Ensemble Neural Network for Disambiguation (ENND). We consider a ground truth GT, similar to the one seen for ENNTR, that this time associates a Wikidata entity identifier (URI) to each token. We identify the target as Y_d.

The ENND architecture is represented in Fig. 3. Differently from related work, the goal of the network would not be to directly predict the right disambiguated entity, but to determine if the predicted entity by an extractor e, where $e \in U$, is correct or not. For this reason, the number of samples in

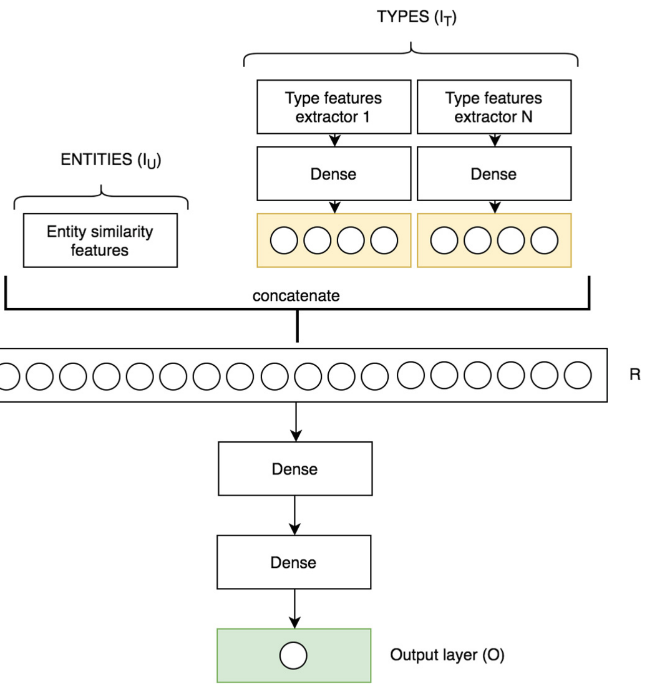

Fig. 3. ENND architecture

target Y_d is not equal to the number of tokens. For each token x, each extractor e returns a predicted entity $\boldsymbol{u_e^x}$: we call C_x the set of predicted entities for the token x, and v_x the correct entity; $|C_x| \leq |U|$ because more extractors could predict the same entity. For each candidate $c_{x,j} \in C_x$, where $0 < j \leq |C_x|$, we generate a target sample $y_d \in Y_d$:

$$y_d = \begin{cases} 1 \; if \; c_{x,j} = v_x \\ 0 \; if \; c_{x,j} \neq v_x \end{cases}$$

The output layer O contains a single neuron that should converge to y_d. The O activation is a sigmoid. Naming I the set of input layers of ENND, two different types of input can be identified depending on the kind of features.

$$I = I_U \cup I_T$$

$$|I| = |I_U| + |I_T| = 1 + |T \cup U|$$

The entity similarity features enter through I_U. We define $c_{x,j}$ as a candidate entity for the token x. For each target sample y_d, we compute a similarity features sample $\boldsymbol{u^{x,j}}$ as:

$$\boldsymbol{u_{x,j}} = [\boldsymbol{S}(c_{x,j}, u_1^x)|\boldsymbol{S}(c_{x,j}, u_2^x)|...|\boldsymbol{S}(c_{x,j}, u_R^x)] \; where \; R = card(U)$$

$$dim(\boldsymbol{u_{x,j}}) = dim(\boldsymbol{S}(\boldsymbol{w_1}, \boldsymbol{w_{2)}})) \cdot card(U)$$

The input layers I_T receive in input the the type feature vector $\boldsymbol{t_e^w}$, computed with the same method used for ENNTR. I_T layers are fully connected to the layers M_n as in ENNTR. M_n is formed by H neurons, where H is an hyper-parameter, set to 4 during our experiment. As for ENNTR, the M_n activation is linear.

After this step, the I_U layer and the M_k layers are concatenated in a new layer R. In this layer, some neurons represent the type information, some other the entity features. This combination aims to exploit the fact that some extractors better disambiguate on certain types. The number of neurons in R is equal to $dim(\boldsymbol{u_{x,j}}) + |T \cup U| \cdot H$.

The last part of the network is composed by two dense layers[6] and the output layer O discussed before. The activation functions of the dense layers cannot be a *softmax* function since the number of candidates – and so is the number of neurons in the output layer – is variable according to each specific token. We so opted for the **Scaled Exponential Linear Units (selu)**:

$$selu(x) = \lambda \begin{cases} x & if\, x > 0 \\ \alpha e^x - \alpha & if\, x \leq 0 \end{cases}$$

The loss function used to train the network is the Mean Square Error, that gives slightly better results and similar training time if compared to MSE.

The neural network goal is to determine the probability that an entity candidate is right. In fact, for each sample, we get an output value that corresponds to this probability. $o_{x,j}$ corresponds to the output value of the input sample associated to the candidate entity j for token x. We select the candidate associated with the highest value $o_{x,max}$ among all output values $\{o_{x,1}, o_{x,2}, ..., o_{x,card(C_x)}\}$. Defining a threshold τ_d, if $o_{x,max} > \tau_d$, we can select as predicted entity for token x the one related to $o_{x,max}$. Otherwise, we consider that the token x is not part of a named entity. This process of **candidate selection** returns the list z_p of predicted Wikidata entities identifiers at token level. In a final step, sequences of tokens which belong to the same Wikidata entity identifiers are merged to a single entity. A_p represents the predicted corpus of annotated fragments.

5 Experiment and Evaluation

We developed an implementation of the two neural networks using Keras.[7] In order to make our approach comparable with the state of the art, our evaluation relies on well-known corpora and metrics, which have been already applied to related work. Moreover, we evaluate our approach on a new gold standard that we provide to the community.

[6] A *dense layer* is a layer fully connected to the previous one.

[7] The source code is available at https://github.com/D2KLab/ensemble-nerd, together with the documentation for accessing the live demo at http://enerd.eurecom.fr.

- **OKE2016:** annotated corpus of English textual resources, created for the 2016 OKE Challenge. The types set contains 4 different tags.[8] This ground truth disambiguates the entities using DBpedia. The ensemble technique we use for scoring is averaging, but not boosting or bagging.
- **AIDA/CoNLL:** English corpus and contains assignments of entities to the mentions of named entities, linked to DBpedia. This dataset does not infer types for NEs and can only be used for evaluating NED.
- **NexGenTV corpus:**[9] dataset composed of 77 annotated fragments of transcripts from politician television debates in French.[10] Each fragment lasts in average 2 min. The corpus is split in 64 training and 13 test samples. The list of types includes 13 different labels.[11] Entities are disambiguated through Wikidata.

Table 4. OKE2016 corpus NER Evaluation

	Token based			Entity based		
	fsc	pre	rec	fsc	pre	rec
adel	0.87	0.88	0.87	0.84	0.85	0.83
alchemy	0.79	0.93	0.68	0.88	0.92	0.86
babelfy	0.66	0.88	0.7	0.74	0.79	0.7
dandelion	0.64	0.89	0.51	0.78	0.83	0.75
dbspotlight	0.59	0.75	0.49	0.6	0.77	0.52
meaning cloud	0.59	0.91	0.44	0.72	0.78	0.69
opencalais	0.56	0.97	0.39	0.69	0.71	0.68
textrazor	0.74	0.86	0.65	0.77	0.81	0.74
ensemble	**0.91**	**0.91**	**0.91**	**0.94**	**0.95**	**0.92**
ensemble ($I = I_T$)	**0.88**	**0.91**	**0.85**	**0.88**	**0.92**	**0.84**
ensemble ($I = I_S$)	**0.50**	**0.53**	**0.47**	**0.50**	**0.52**	**0.48**
ensemble ($I = I_U$)	**0.44**	**0.47**	**0.41**	**0.43**	**0.43**	**0.43**
ensemble ($I = I_K$)	**0.37**	**0.40**	**0.34**	**0.38**	**0.40**	**0.36**

Type Recognition. For each gold standard GT, two different kinds of score are computed. The *token based* scores have been used in [9,10]. From GT, a list of target types l_t with dimension $|X|$ is extracted. We can obtain from ENNTR the list of predicted types l_p. For each type t_{GT} in GT, we compute precision

[8] PERSON, ORGANIZATION, PLACE, ROLE.
[9] http://enerd.eurecom.fr/data/training_data/nexgen_tv_corpus/.
[10] The debates are in the context of the 2017 French presidential election.
[11] PERSON, ORGANIZATION, GEOGRAPHICAL POINT, TIME, TIME INTERVAL, NUMBER, QUANTITY, OCCURRENCE, EVENT, INTELLECTUAL WORK, ROLE, GROUP OF HUMANS and OCCUPATION.

$Precision(l_t, l_p, t_{GT})$, recall $Recall(l_t, l_p, t_{GT})$ and F1 score $F1(l_t, l_p, t_{GT})$. Then, we compute micro averaged measures $Precision_{micro}(l_t, l_p)$, $Recall_{micro}(l_t, l_p)$ and $F1_{micro}(l_t, l_p)$ [8].

The *entity based* scores follow the definition of precision and recall coming from the **MUC-7 test scoring** [2]. Given A_t and A_p as the annotated fragment in GT, the computed measures are $Precision_{brat}(A_t, A_p)$, $Recall_{brat}(A_t, A_p)$ and $F1_{brat}(A_t, A_p)$.

The computed scores for OKE2016 and NexGenTv corpora are reported in Tables 4 and 5. The tables show also the same metrics applied to single extractors, after that their output types have been mapped to the ones of GT through the alignment block of ENNTR. For both token and entity scores, the ensemble method outperforms the single extractors for all metrics.

Table 5. NexGenTv corpus NER evaluation

	Token based			Entity based		
	fsc	pre	rec	fsc	pre	rec
adel	0.68	0.84	0.57	0.75	0.83	0.7
alchemy	0.80	0.83	0.77	0.87	0.97	0.81
babelfy	0.55	0.83	0.41	0.65	0.74	0.59
dandelion	0.26	0.69	0.16	0.51	0.69	0.42
dbspotlight	0.48	0.75	0.34	0.5	0.61	0.45
meaning cloud	0.82	0.88	0.77	0.8	0.87	0.76
opencalais	0.58	0.81	0.45	0.81	0.9	0.76
textrazor	0.81	0.89	0.74	0.75	0.8	0.72
ensemble	**0.94**	**0.97**	**0.91**	**0.92**	**0.98**	**0.87**
ensemble ($I = I_T$)	**0.87**	**0.91**	**0.83**	**0.89**	**0.93**	**0.85**
ensemble ($I = I_S$)	**0.54**	**0.58**	**0.50**	**0.53**	**0.56**	**0.50**
ensemble ($I = I_U$)	**0.47**	**0.49**	**0.45**	**0.46**	**0.47**	**0.45**
ensemble ($I = I_K$)	**0.40**	**0.42**	**0.38**	**0.39**	**0.40**	**0.38**

In order to identify the most impacting features in the obtained results, ENTTR has been sequentially adapted and retrained in order to receive in input only a specific kind of features, i.e. only I_T, I_K, I_U or I_S. The tokens based scores for these new trained networks reveals that the type features I_T are the only ones that, used alone as input, continue to make ENTRR outperforming single extractors, as can be expected given the type recognition goal. The other feature kinds, while having a lower impact, are still improving the final results when combined in the ensemble.

Entity Linking. We evaluate the entity linking for both OKE2016, AIDA/CoNLL and NexGenTv corpora using the GERBIL framework[12] and in particular micro and macro scores for the experiment type "Disambiguate to Knowledge Base" (D2KB). The computed scores are reported in Tables 6 and 7; the ensemble method outperforms again the single extractors that it integrates for all metrics. As for type recognition, we repeated the experiment using only a specific kind of features, in order to show the feature impact. In such case, the most influential features are the entity ones I_U. However, the impact of type features I_T is still crucial because its absence reduce drastically the improvement of the ensemble method with respect to the single extractors.

Tables 8 and 9 compare the NED extractors presented on GERBIL with our ensemble. For OKE2016, PBOH is the only tool which obtains a better score However this extractors reaches very low scores for AIDA/CoNLL, while our

Table 6. GERBIL Micro scores on OKE2016, NexGenTV and AIDA/CoNLL corpus

	OKE2016			NEXGEN			AIDA		
	fsc	pre	rec	fsc	pre	rec	fsc	pre	rec
babelfy	0.54	0.64	0.47	0.51	0.51	0.51	0.66	0.70	0.62
dandelion	0.59	0.77	0.48	0.34	0.50	0.26	0.45	0.66	0.34
dbspotlight	0.39	0.53	0.30	0.38	0.29	0.54	0.47	0.65	0.36
textrazor	0.53	0.78	0.40	0.61	0.55	0.69	0.62	0.57	0.53
ensemble	**0.66**	**0.88**	**0.52**	**0.69**	**0.70**	**0.64**	**0.68**	**0.79**	**0.60**
ensemble ($I = I_U$)	**0.59**	**0.80**	**0.47**	**0.59**	**0.60**	**0.58**	**0.55**	**0.60**	**0.50**
ensemble ($I = I_T$)	**0.41**	**0.45**	**0.38**	**0.42**	**0.47**	**0.38**	**0.48**	**0.52**	**0.45**

Table 7. GERBIL Macro scores on OKE2016, NexGenTV and AIDA/CoNLL corpus

	OKE2016			NEXGEN			AIDA		
	fsc	pre	rec	fsc	pre	rec	fsc	pre	rec
babelfy	0.54	0.65	0.47	0.51	0.52	0.51	0.60	0.65	0.57
dandelion	0.59	0.76	0.49	0.35	0.50	0.27	0.43	0.52	0.37
dbspotlight	0.39	0.52	0.32	0.38	0.29	0.55	0.45	0.63	0.37
textrazor	0.54	0.77	0.42	0.61	0.54	0.71	0.57	0.78	0.45
ensemble	**0.65**	**0.86**	**0.53**	**0.67**	**0.69**	**0.64**	**0.68**	**0.76**	**0.61**
ensemble ($I = I_U$)	**0.59**	**0.77**	**0.48**	**0.59**	**0.59**	**0.59**	**0.55**	**0.59**	**0.51**
ensemble ($I = I_T$)	**0.42**	**0.44**	**0.40**	**0.41**	**0.42**	**0.40**	**0.49**	**0.51**	**0.47**

[12] GERBIL is a general Linked Data benchmarking that offers an easy-to-use web-based platform for the agile comparison of annotators using multiple datasets and uniform measuring approaches.

Table 8. GERBIL scores on OKE2016

	Micro scores			Macro scores		
	fsc	pre	rec	fsc	pre	rec
agdistis	0.50	0.50	0.50	0.52	0.52	0.52
aida	0.49	0.63	0.41	0.5	0.64	0.42
dexter	0.44	0.92	0.29	0.43	0.81	0.31
fox	0.48	0.77	0.35	0.47	0.69	0.37
freme ner	0.31	0.57	0.21	0.26	0.27	0.25
kea	0.64	0.67	0.61	0.63	0.66	0.61
pboh	0.69	0.69	0.69	0.69	0.69	0.69
ensemble	**0.66**	**0.88**	**0.52**	**0.65**	**0.86**	**0.53**

Table 9. GERBIL scores on AIDA-CoNLL

	Micro scores			Macro scores		
	fsc	pre	rec	fsc	pre	rec
agdistis	0.58	0.58	0.58	0.59	0.59	0.59
aida	0.00	0.00	0.00	0.00	0.00	0.00
dexter	0.51	0.76	0.38	0.47	0.75	0.36
fox	0.57	0.63	0.51	0.56	0.64	0.51
freme ner	0.38	0.62	0.27	0.29	0.30	0.27
kea	0.60	0.65	0.56	0.59	0.63	0.56
pboh	0.00	0.00	0.00	0.00	0.00	0.00
ensemble	**0.68**	**0.79**	**0.60**	**0.68**	**0.76**	**0.61**

ensemble still continues to have good performances. For the NexGenTV dataset, we cannot compare the other NERD extractors because the majority of them perform NED only for the English language.

6 Conclusion and Future Work

In this paper, we presented two multilingual ensemble methods which combine the responses of web services (extractors) performing Named Entity Recognition and Disambiguation. The method relies on two Neural Networks that outperform the single extractors respectively in NER and NED tasks. Furthermore, the NER network allows to avoid the manually type alignment between the type taxonomies of each extractor and the ground truth taxonomy. We demonstrated the importance of the features generation for the success of these ensemble methods. In terms of NER, the type features play most of the work in the ensemble. For the NED task, while entity features have the greater impact, only a combination with type features really improve the effectiveness of the ensemble method with respect to single extractor predictions.

As future work, we plan to enhance the input feature set with Part of Speech tags features that would be assigned to each token. We also aim to vary the neural network architecture, and in particular, we are planning to replace the dense layer receiving the surface features with a BiLSTM, which would also take in consideration the context in which the tokens are sequentially appearing. Finally, all the neural networks models have been trained when all extractors APIs were reachable. A training that involves some samples which simulates the extractors failures and unavailability would make the network models more robust to API failures.

Acknowledgements. This work has been partially supported by the French National Research Agency (ANR) within the ASRAEL project (ANR-15-CE23-0018), the French Fonds Unique Interministériel (FUI) within the NexGen-TV project and the European Union's Horizon 2020 research and innovation programme via the project MeMAD (GA 780069).

References

1. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606 (2016)
2. Chinchor, N.: Appendix B: MUC-7 test scores introduction. In: Seventh Message Understanding Conference (MUC-7), Fairfax, Virginia, USA (1998)
3. Finkel, J.R., Grenager, T., Manning, C.: Incorporating non-local information into information extraction systems by Gibbs sampling. In: 43rd Annual Meeting on Association for Computational Linguistics (ACL), Ann Arbor, Michigan, USA, pp. 363–370 (2005)
4. Ratinov, L., Roth, D.: Design challenges and misconceptions in named entity recognition. In: 13th Conference on Computational Natural Language Learning (CoNLL), Boulder, Colorado, USA, pp. 147–155, June 2009
5. Rizzo, G., Troncy, R.: NERD: a framework for unifying named entity recognition and disambiguation extraction tools. In: 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL), Avignon, France, pp. 73–76 (2012)
6. Rizzo, G., van Erp, M., Troncy, R.: Benchmarking the extraction and disambiguation of named entities on the semantic web. In: 9th International Conference on Language Resources and Evaluation (LREC), Reykjavik, Iceland (2014)
7. Rizzo, G., van Erp, M., Troncy, R.: Inductive entity typing alignment. In: 1st International Workshop on Linked Data for Information Extraction (LD4IE), Riva del Garda, Italy (2014)
8. Sebastiani, F.: Machine learning in automated text categorization. ACM Comput. Surv. **34**(1), 1–47 (2002)
9. Speck, R., Ngonga Ngomo, A.-C.: Ensemble learning of named entity recognition algorithms using multilayer perceptron for the multilingual web of data. In: 9th International Conference on Knowledge Capture (K-CAP), Austin, TX, USA (2017)

10. Speck, R., Ngonga Ngomo, A.-C.: Ensemble learning for named entity recognition. In: Mika, P. (ed.) ISWC 2014 Part I. LNCS, vol. 8796, pp. 519–534. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11964-9_33
11. van Erp, M., Rizzo, G., Troncy, R.: Learning with the web: spotting named entities on the intersection of NERD and machine learning. In: 3rd International Workshop on Making Sense of Microposts (#MSM), Concept Extraction Challenge, Rio de Janeiro, Brazil (2013)
12. Zhang, F., Yuan, N.J., Lian, D., Xie, X., Ma, W.-Y.: Collaborative knowledge base embedding for recommender systems. In: 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), San Francisco, California, USA, pp. 353–362 (2016)