Semester project

DHLAB - IC

Active learning for deep reference parsing

Author:
Mattia Martinelli

Supervisor:
Giovanni Colavizza

June 8, 2018

Abstract

In recent years, models based on deep neural networks (DNNs) have shown outstanding results in several domains, outperforming other machine learning techniques. However, such models often require large collections of labeled data to achieve state-of-the-art performance. Labeling data can be a costly and time-consuming process, and some research has been conducted on reducing training sets in a methodical manner. In this work, we show that an accuracy close to that of a fully-trained model can be achieved with only a fraction of the training data. We develop a lightweight CNN-CNN-LSTM model, consisting of convolutional (CNN) character and word encoders, and a long short term memory (LSTM) tag decoder; and we use it to parse bibliographic references from humanities literature. Active learning techniques are explored, and in particular we present an approach based on uncertainty sampling. We are able to nearly achieve the accuracy of our best models on two tasks with, respectively, 40% and 19% of the original training data.

Keywords: Reference parsing, deep learning, active learning, uncertainty sampling.

Contents

1 Introduction

Models based on deep neural networks (DNNs) have shown to outperform traditional machine learning methods in various domains, including natural language processing (NLP) [7, 10]. In particular, DNNs have been successfully deployed over the past few years in the tasks of named-entity recognition (NER) and part-of-speech (POS) tagging [2]. Traditional methods, including hidden markov models (HMMs) and conditional random fields (CRFs), are hard to engineer and not easy to adapt to variations of a given task [4]. Models based on DNNs inherently enable significant advantages, as they do neither require to craft hand-engineered features nor gather expert human knowledge in specific domains. Therefore, DNNs offer a considerably larger degree of flexibility; however, as a major drawback, they often require a large amount of labeled training data to achieve state-of-the-art performance [7].
It is not always possible to rely on a sufficiently large dataset to train a DNN model, as resources that need to be invested in labeling the dataset may be prohibitive. For some NER tasks, unlabeled data can be easily retrieved from the web, e.g. by means of scraping techniques. Conversely, annotation is an expensive procedure that often requires qualified operators and time investment. This problem has pushed researchers to explore methods that can reduce the quantity of labeled data needed by a model to achieve comparable performance as it was trained with a full dataset. A solution is *active learning*, that is, an ensemble of methods and techniques that allow a model to select the most valuable samples from a pool data, which can then be used for training. However, active learning poses the challenge of choosing the most suitable metric to determine which samples are more informative than the others to train a model [12].
In this work, we show that the quantity of training data can be effectively reduced to achieve results comparable to a fully-trained model. We develop a lightweight DNN model based on convolutional neural networks (CNNs) and long-short term memory (LSTM) networks, and we train it on a dataset of references from humanities literature, exploring different active learning techniques.
This report begins providing some theoretical background: Section 2 contains a brief literature review of our domains of interest, Section 3 describes in detail our tasks and dataset, and Section 4 explains our active learning approach. The rest is organized as follows: Section 5 describes our model, Section 6 shows the result of our experiments, and finally Section 7 proposes discussion and conclusion.

2 Related work

Our work covers different research areas, however, we mainly focus on the following topics: reference parsing in humanities literature, active learning, and active learning with DNNs. In this section we propose a brief literature review.

2.1 Reference parsing in humanities literature

Reference parsing is closely related to the NER task and consists in labeling each element of a reference to its semantics, such as *author, title, publisher*, and *year*. However, the variety of referencing styles and referred objects in humanities literature poses more challenges as compared to a regular NER problem. [3] has successfully deployed CRFs in the domain. [4] has improved results with a BiLSTM-CRF architecture, which has shown to outperform vanilla CRFs in different tasks. We base our work on the same tasks and dataset, and we aim to apply active learning techniques to minimize the quantity of data necessary to obtain results comparable to [4]. The tasks are explained in detail in Section 3.

2.2 Active learning

[12] offers an extensive literature review in the domain of active learning. However, applications of active learning to DNNs have been little explored and necessitate of solid theoretical analysis [13]. Some research in active learning with DNNs has been done in the field of image classification. [15] has applied uncertainty-based sampling for image classification with CNNs. However, active learning for NER tasks has not been yet much investigated. Shen et al. [13] addresses a NER task and claims to achieve state-of-the-art performance with only 25% of their dataset. Their work is based on two key ideas. Firstly, they develop a lightweight CNN-CNN-LSTM model to conduct their experiments. Since the model is retrained several times on different datasets, it needs to be considerably faster compared to a pure LSTM model. Although they experience a small loss of performance, the significant training speed up justifies their choice. Secondly, they propose uncertainty sampling as an active learning strategy. Uncertainty sampling is a simple and computationally cheap technique, and they have shown that it can effectively reduce the quantity of training data needed to train their model. Since our task is closely related to NER, we gather inspiration from their approach.

3 Task definition

Our work is based on the tasks that have been explored by [4], which include the extraction, detection and classification of bibliographic references from literature in arts and humanities. A bibliographic reference is defined as a contiguous sequence of text where all the necessary information on a citation is contained [4]. Citations may refer to primary or secondary sources of information, written by the same or another author. A reference is usually composed of several components, such as the *author*, *title* and *publisher* of the referred publication. An example of a reference is:

```
N. Mangini, I teatri di Venezia, Mursia, Milano, 1974.
```

This reference is composed of: author's name (*N. Mangini*), title (*I teatri di Venezia*), publisher (*Mursia*), and location (*Milano*) and year (*1974*) of publication. In this example, the components are separated by a comma and the author's name is abbreviated using initials followed by a dot. However, this might not always be the case. Extraction and classification of references in humanities pose a demanding challenge, as the elements of a reference show no fixed structure. References are often provided in the form of footnotes and are rarely collected in dedicated sections. As a consequence, the same reference might appear in another document with a different structure and/or with a different syntactic style, e.g. the author's name is never abbreviated.
From a technical perspective, a reference consists of a finite sequence of tokens. Each token can be a word, a number, or a punctuation sign. A reference can be also placed in a larger sequence of text, and in such a case it becomes important to distinguish the reference from the remaining text. [4] has investigated *reference mining*, defined by three main steps: **detection** of the reference, **extraction** of the reference from the remaining text, and **classification** of the tokens in the reference. These steps can be summarized with the following tasks: (which are also depicted in Fig. 1):

- Task 1: *Reference components*, that is, annotations. Each token is classified using a taxonomy of 27 specific tags (which is discussed in [3]). Tags are not uniformly represented in the dataset and show a skewed distribution.
- Task 2: *Reference typology.* The entire reference is classified as a *primary* or *secondary* source of information. In addition, tokens are classified as begin reference, inside reference, and end reference.
- Task 3: *Reference span*, that is, begin and end of the reference. Each token is classified using one of the following tags: *b-r* begin of reference, *e-r* end of reference, *i-r* inside reference, and *o-r* outside reference.

Our work exclusively addresses Task 1 and Task 3. We call these tasks *reference parsing*, as they focus on classifying single tokens of the reference. On the other hand, Task 2 involves a classification on the entire reference, i.e. every token of the reference accounts

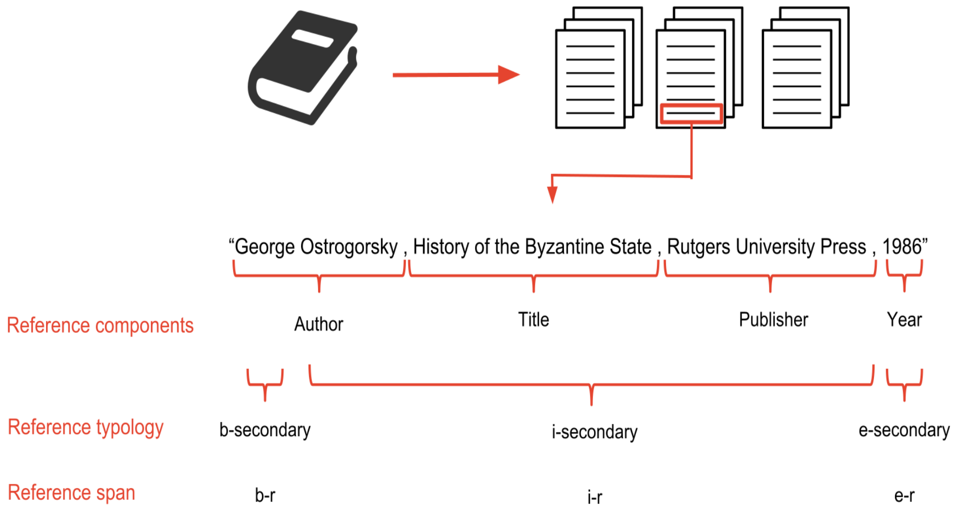


Figure 1: Each reference is annotated according to the three tasks: reference components (Task 1), reference topology (Task 2), and reference span (Task 3).

for the same label. Our active learning algorithm requires to compute probabilities of labeling single tokens independently, therefore, it cannot be applied to Task 2. More details are provided in Section 4.

3.1 Dataset structure

The dataset contains more than 40'000 annotated references from a corpus of publications on the historiography on Venice. The corpus includes books and journal articles published from the 19th century to 2014. Publications cover different languages: mostly Italian, followed by English, French, German, Spanish and Latin. The annotated corpus includes references taken from reference lists and footnotes, therefore, a considerable variety of referencing styles and referred sources is present. The dataset does not contain the full text of publications, but only the text lines where a reference (or part of it) appears; therefore some lines of text include out-of-reference tokens, preceding or following a reference (these tokens are important to learn to assign begin-end tags). Full details, including corpus acquisition and annotation sampling strategy and procedure, are given in [3] [1].
Every publication with annotated references is randomly allocated in a train, test or validation set, with an 80/10/10 split at the document level, respectively. The number of references in each set does not precisely follow the same proportion, however a document-level split is important in order to reduce reference style data snooping [4]. Additional details on the dataset structure are given in [4].

[1]The dataset can be downloaded here: https://github.com/dhlab-epfl/LinkedBooksReferenceParsing.

4 Active learning

Active learning is an ensemble of methods and techniques that aim at reducing in a methodical manner the quantity of data necessary to achieve performance comparable to a model trained with the entire data. Active learning falls into the broader category of semi-supervised approaches, as it is not fully automated and requires constant interaction with the user. Active learning has received attention in recent years due to the significantly large quantity of labeled training data required by supervised DNNs, and which may be expensive to obtain [11, 13, 15]. The key idea of active learning is to use the model to select which data instances it can train with at every iteration. Given a large set of unlabeled data, which is usually considerably cheaper to collect, the model (i.e. the learner) can query the user (i.e. the oracle) to know how the most valuable data, in terms of training efficacy, should be labeled. [12] claims that, if the selection heuristics is properly implemented, a model can perform better and with less training. However, as discussed later, the main challenge is to determine how the most valuable data is measured [13].

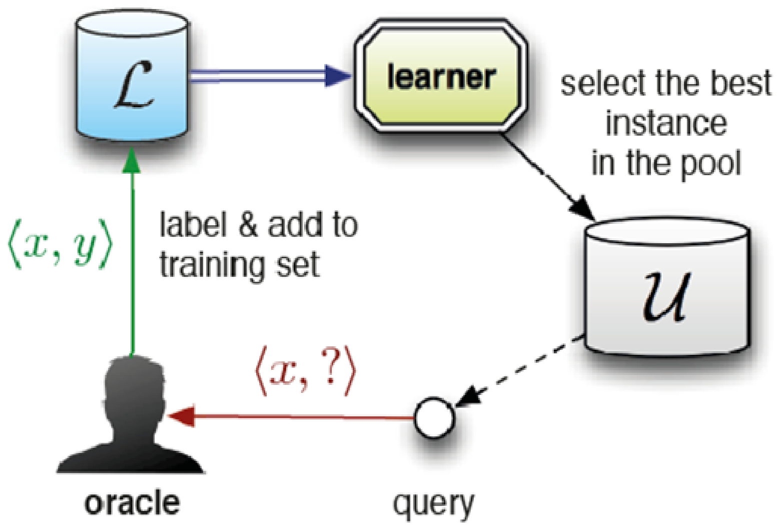


Figure 2: Pool-based sampling. Picture taken from [12].

According to how the model queries the user, [12] suggests that active learning frameworks can be classified into three categories:

- *Query synthesis*: the learner can request labels for any unlabeled data instance in the input space, including queries that the learner synthesizes *de novo*. The only assumption is that the learner has a definition of the input space available to it.

- *Stream-based selective sampling*: the assumption is that obtaining a single unlabeled instance is inexpensive, so it can first be sampled from the actual distribution and then the learner can decide whether or not to request its label. Each unlabeled instance is typically drawn one at a time from the input source, and the learner must decide whether to query or discard it.

- *Pool-based sampling:* the assumption is that a small set of labeled data with large pool of unlabeled data is already available. Queries are done on static pool of unlabeled data and processed in one time. The algorithm evaluates and ranks the entire collection of unlabeled data before selecting the best query.

Since we are already in possession of the full training dataset of [4], we have decided to build our model upon the pool-based sampling.

4.1 Uncertainty sample

Every active learning framework poses the challenge to select what unlabeled instances are most valuable for training the model. Since the primary goal is to minimize the number of queries asked to the user, the quantity of selected instances should be as small as possible. Uncertainty sampling is a popular approach for active learning with DNN models [13, 15]. With the uncertainty sampling setup, the model selects the unlabeled instances, in our case the tokens in references, according to its uncertainty on their predictions. The assumption is that the learner avoids querying the tokens on which it is already confident, and conversely, it focuses its attention on the unlabeled tokens that it finds confusing [12], three different metrics can be found in literature:

- *Least confidence:* selects the lowest probability on the best prediction. In formal terms, $\text{argmin}_x P_\theta(\hat{y}_1|x)$, that is, x s.t. the probability of its label $\hat{y}_1$ given the model parameters θ is minimized. It exclusively considers the information coming from the best prediction $\hat{y}_1$.

- *Output margin:* selects the lowest margin between the first and second best prediction. In formal terms, $\text{argmin}_x[P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x)]$, that is, x s.t. the distance between the probabilities of the two best predictions $\hat{y}_1$ and $\hat{y}_2$ is minimized, given model parameters θ. Compared to the previous measure, it also considers the information coming from the second best prediction.

- *Entropy:* selects the highest entropy over the tag prediction probability distribution, which can be see as the average information content. In formal terms, $\text{argmax}_x H_\theta(Y|x)$, that is, x s.t. the entropy over its N possible labels is maximized. The entropy $H_\theta(Y|x)$ is defined as $-\sum_n^N P_\theta(y_n|x) log P_\theta(y_n|x)$, where y_n represents a label and θ the model parameters.

The uncertainty sample approach is simple to implement and does not add much computation overhead. However, the only and simple hypothesis on which it relies, i.e. the learner can improve with the data on which it is least confident, might not be the best fit for every scenario. There are registered cases in which even a random sampling performs better than uncertainty sampling [12].

4.2 Preliminary experiments

We have tested the three uncertainty sample measures with the model proposed by [4]. The first objective of the test is to compare how the different measures affect the way the model selects to data. The second objective is to confirm whether the model still improves performance adding part of the test set to the train set, evaluating results on the validation set.

Reference	Position	Token	Target	Predicted	Probability
231	7	annuale	title	volume	0.1685
1940	22	disp	numbered_ref	numbered_ref	0.1708
234	8	Esercizii	title	attachment	0.1805
68	17	-	title	pagination	0.1833
234	10	-	title	o	0.1845

Table 1: Five least confident tokens in test dataset according to the least confidence metric.

Reference	Position	Token	Target	Predicted	Probability
871	13	Press	publisher	publisher	1.000
862	9	Press	publisher	publisher	1.000
896	9	Press	publisher	publisher	1.000
930	11	Press	publisher	publisher	1.000
871	12	University	publisher	publisher	1.000

Table 2: Five most confident tokens in test dataset according to the least confidence metric.

Table 1 and Table 2 show the results of uncertainty sampling with least confidence score on the test set. As we have access to the work of [4], we used their BiLSTM-CRF model trained on the Task 1 dataset to compute the scores. We observe that on the least confident tokens the model predictions are wrong, while on the most confident they are correct with probability close to 100%. In the latter case, the tokens are *Press* and *University*, which are common words in the dataset and distributed over different references. In addition, such words are very specific target of the label *publisher*. Conversely, the label *title* is the most common tag in the dataset (approximately 30%) and

	Precision	Recall	F1 Score
Baseline	0.8949	0.8904	0.8904
Low H	0.9031	0.9031	0.8973
High H	0.9531	0.9531	0.9514
Random	0.9352	0.9339	0.9331

Table 3: Task 1 results on test dataset

	Precision	Recall	F1 Score
Baseline	0.8911	0.8958	0.8902
Low H	0.8950	0.8997	0.8950
High H	0.8956	0.8959	0.8929
Random	0.8953	0.8970	0.8938

Table 4: Task 1 results on validation dataset

is associated to various tokens in different contexts.
We have then conducted a test with the BiLSTM-CRF model to study its training capacity. We have chosen the entropy as metric of uncertainty and we have added 50% of the test dataset to the training dataset in order to investigate if results on validation could be improved within an active learning framework.
Table 3 and Table 4 present the results of the experiment. High entropy, and therefore least confidence, apparently shows to improve the performance; however, the results on validation confirms that the model actually overfits the test dataset.
We therefore conclude that the model has reached its trained capacity because of a limitation in our dataset. There may be a set of tokens extremely difficult to classify due to noise or underrepresentation in the training dataset. There is no benefit in adding more references from our dataset, the most challenging references to classify must be investigated and new data must be collected.

4.3 Active learning algorithm

Our active learning algorithm is based on the pool-sampling approach, and we apply entropy as metric of uncertainty sampling. In order to compare how entropy affects the training process, the algorithm can sample instances with lowest entropy (most confident), highest entropy (least confident) or random. We assume that we cannot partially annotate a reference, and we consider complete references as the atomic unit of the dataset. Therefore, we compute the uncertainty score on the entire reference, defined as the average of the entropy of the l tokens in the reference.

$$H_{sequence} = \frac{1}{l}\sum_{i}^{l} H_i,\ H_i \textit{ is the entropy of the i-th token.}$$

Algorithm 1 shows the pseudo-code of our active learning algorithm. The training dataset is firstly split into the labeled dataset L and the unlabeled dataset U. In detail, we randomly sample the training dataset in such a way that a minimum number of tokens is guaranteed for each label. We therefore avoid to train the model with missing labels in the labeled dataset. We then perform N training cycles. At each cycle, the model is trained with the current labeled set, and entropy is computed on the unlabeled pool (with the same model parameters). The oracle provides the learner with the labels on the selected references. In our case we do not need to literally label the references since our entire dataset is already labeled. At the end of the cycle, the selected references are added to labeled set and removed from the unlabeled pool.

Algorithm 1 Pool-based entropy uncertainty sampling

```
1: create $U$, $L$                          ▷ Pool of unlabeled and labeled references
2: for n = 1,2,...,N do
3:     $\theta \leftarrow \mathbf{train}(L)$                   ▷ Train model $\theta$ on $L$
4:     $\bar{\mathbf{x}} \leftarrow \mathbf{score}(\theta, U)$            ▷ Uncertainty sample on $U$ with model $\theta$
5:     $\bar{\mathbf{y}} \leftarrow \mathbf{query}(\bar{\mathbf{x}})$                ▷ Query the oracle to get labeling
6:     $L \leftarrow L + (\bar{\mathbf{x}}, \bar{\mathbf{y}})$                ▷ Add $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ to $L$
7:     $U \leftarrow U - \bar{\mathbf{x}}$                    ▷ Remove $\bar{\mathbf{x}}$ from $U$
8: end for
```

5 Model

In NER applications, models based on LSTM DNNs have shown to outperform models based on other techniques [13]. However, LSTMs often require a considerable longer training time to achieve the highest accuracy. In addition, this time is proportional to the depth of the network. As our work requires numerous training cycles, we have developed a lightweight DNN model based on a small number CNN layers and a single LSTM layer with few hidden units .

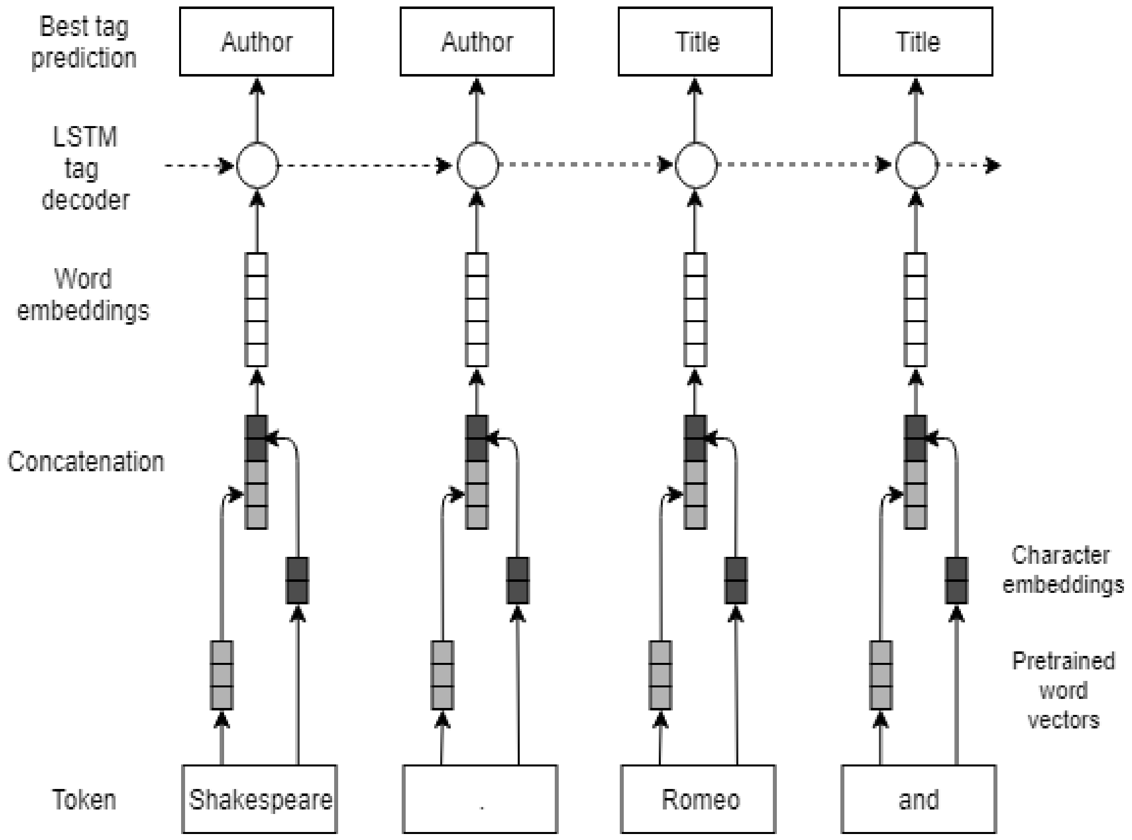


Figure 3: Our model is based on CNN-CNN-LSTM network.

The model, which has been previously proposed by [13], consists of three main components: the character-level encoder, the word-level encoder, and the tag decoder. Each component offers several hyper-parameters to tune, however, we rely on the same ones of [4, 13], as we assume that their work already includes a grid search on hyper-parameters. In addition, since our work is focused on active learning, we are not interested in fine-tuning our model.

5.1 Reference representation

References are represented as fixed length sequences of integers, where each unique token in text is mapped to a specific integer. Numbers in text, such as years or pages, are uniformed under a dedicated token (NUM), therefore, different numbers are not distinguished.

Reference	PAD	PAD	,	Giulietta	,	and
Representation	0	0	9	30	9	18

Table 5: Word representation of ”*, Giulietta, and*”. Each reference is represented as a fixed-length sequence of integers (in this case 6).

References are padded with the special PAD token to uniform their length. In detail, each input sequence is pre-pended with a sequence of 0s (which is a reserved index for padding) to ensure a fixed-length representation. Each reference is therefore extended to the length of the longest sequence in the text. Given list of N formatted references x_1, x_2, ... , x_N, the input value x_{ni} represents the numeric representation of the i-th word of the n-th reference.
To enable character-level encoding, each unique character in text is mapped to a specific integer, and tokens are encoded as sequences of characters. The characters are mapped case-sensitive, as character-level encoding aims at distinguishing unique character features such as proper nouns and beginning of sentences.

Token	PAD	PAD	PAD	PAD	E	l	l	e
Representation	0	0	0	0	6	15	15	10

Table 6: Character representation of ”*Elle*”. Each token is represented as a fixed-length sequence of integers (in this case 8).

The same padding operation is performed then with characters. The length of every character sequence is equal to the length of the longest token in the entire text. The input value x_{nij} represents the numeric representation of the j-th character in the i-th word of the n-th reference.

5.2 Character embeddings

Character embeddings enable the model to extrapolate specific character-level features, like prefix, suffix and spelling of words [4]. Conversely, a pure word-level encoder would integrate such features inside word embeddings, without the same influence on the model as if they were a dedicated part of the word representation. In addition, it would be more sensitive to OCR errors, as a misspelling of two identical words would result in two different encodings. Rare words benefit from character-level encoding, as a word-level encoding would poorly represent them. The model extracts character-level features by means of a CNN. Despite LSTMs slightly outperform CNNs as character-level encoders [6], the improvement is not relevant whereas the computational cost is significantly higher.

Layer	Out dimension
Input	$[l_w, l_c]$
Embedding	$[l_w, l_c, 100]$
Convolution 1D + Leaky ReLU	$[l_w, l_c, 50]$
Max Pooling 1D	$[l_w, l_c, 50]$

Table 7: List of layers of the character-level encoder.

Table 7 shows the architecture of the character embedding CNN. The network receives as input a list of sequences of dimension (l_w, l_c), where l_w is the length of word sequence and l_c is the length of the character sequence. The embedding layer transforms each input character into a dense vector of size 100. The convolutional 1D layer uses a set of 50 filters with width 3 and leaky ReLU activation function. Max-pooling with window width 2 and stride 1 is then applied on the output.

5.3 Word embeddings

Each word is represented as the concatenation of its character embedding with a latent embedding of the word itself. The latter is initialized with Word2vec embeddings [8]. Since our dataset is highly specialized and contains a large number of uncommon words, we assume that we cannot rely on purely pretrained embeddings [3]. Word2vec embeddings are trained on the full contents of all the publications from which our references were extracted. We use the Gensim Word2vec implementation [9], a window of 5 words, and the skip-gram model. The vectors are trained for all words appearing at least five times in the dataset, and less frequent words have been regrouped under a dedicated token (UNK). The word embeddings are then retrained with the model. The size of the embedding vector is 300.
The concatenation of the word and character embedding is then inputed to a CNN, which consists of two 1D convolutional layers with 800 filters of width 5 and leaky ReLU activation function. Between convolutional layers, a dropout layer with probability 0.5 is placed to reduce overfitting [14]. Max-pooling with window width 2 and stride 1 is then applied on the output.

5.4 Tag decoder

Word embeddings are inputed to a LSTM layer [5]. In a sequence of text, the prediction on the current word strongly depends on the predictions of the previous words. Re-

Layer	Out dimension
Input	$[l_w]$
Embedding	$[l_w, 300]$
Concatenation	
Convolution 1D + Leaky ReLU	$[l_w, 800]$
Dropout	
Convolution 1D + Leaky ReLU	$[l_w, 800]$
Max Pooling 1D	$[l_w, 800]$

Table 8: List of layers of the word-level encoder.

current neural networks perform better than other models in NER tasks because they can capture the sequential dependency of the input data. An RNN cell with sigmoid activation and softmax prediction can be described as follows:

$$\mathbf{h}^{(t)} = \sigma(\mathbf{b} \; + \; \mathbf{W}\mathbf{x}^{(t)} \; + \; \mathbf{U}\mathbf{h}^{(t-1)})$$
$$\hat{\mathbf{y}}^{(t)} = softmax(\mathbf{c} \; + \; \mathbf{V}\mathbf{h}^{(t)})$$

where $\mathbf{x}^{(t)}$ is the input word representation in position t of the current sequence, $\mathbf{h}^{(t)}$ represents the hidden state at the same position, $\mathbf{b}$ and $\mathbf{c}$ are bias vectors and $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$ are parameter matrices to be learned.
However, vanilla RNNs have shown short memory dependency and tendency to suffer from vanishing or exploding gradients [1]; therefore, attention has been recently given to LSTMs. A LSTM cell is composed of three gates: an input gate $\mathbf{i}$, a forget gate $\mathbf{f}$, and an output gate $\mathbf{o}$, in order to provide the cell with a means to retain information on previous states more effectively. An LSTM cell with softmax prediction, as implemented in Keras, can be described as follows [4]:

$$\mathbf{i}^{(t)} = \sigma(\mathbf{b}^i \; + \; \mathbf{W}^i\mathbf{x}^{(t)} \; + \; \mathbf{U}^i\mathbf{h}^{(t-1)})$$
$$\mathbf{f}^{(t)} = \sigma(\mathbf{b}^f \; + \; \mathbf{W}^f\mathbf{x}^{(t)} \; + \; \mathbf{U}^f\mathbf{h}^{(t-1)})$$
$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + \; \mathbf{i}^{(t)} \odot tanh(\mathbf{b}^c \; + \; \mathbf{W}^c\mathbf{x}^{(t)} + \mathbf{U}^c\mathbf{h}^{(t-1)})$$
$$\mathbf{o}^{(t)} = \sigma(\mathbf{b}^o \; + \; \mathbf{W}^o\mathbf{x}^{(t)} \; + \; \mathbf{U}^o\mathbf{h}^{(t-1)})$$
$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot tanh(\mathbf{c}^{(t)})$$
$$\hat{\mathbf{y}}^{(t)} = softmax(\mathbf{c} \; + \; \mathbf{V}\mathbf{h}^{(t)})$$

where σ is the element-wise hard-sigmoid function and $\odot$ is the element-wise product. As before, $\mathbf{x}^{(t)}$ is the input word representation in position t of the current sequence and $\mathbf{h}^{(t)}$ represents the hidden state at the same position. $\mathbf{x}^{(t)}$ represent the current cell state, as a function of the forget gate applied to the previous step cell state, and the input gate applied to a non-linear transformation (hyperbolic tangent in this case) of a vanilla RNN internal state. The final hidden state is then given by a product of

the output gate with a further non-linear transformation of the cell state. The different bias vectors **b** and **c** and matrices **W**, **U** and **V** are all learned parameters.
A dense layer is placed on the top of the LSTM, and to compute output probabilities, we apply a softmax function to the output. Intuitively, the softmax function maps a vector of integer numbers into a vector of probabilities of the same size. Higher probability is assigned to higher values whereas lower probability is assigned to lower values. Given a token $\bar{x}$ and K tags t_1, t_2, ..., t_K, the probability that $\bar{x}$ is labeled t_k is:

$$p(t_k|\bar{x}) = \frac{exp(o_k(\bar{x}))}{\sum_j exp(o_j(\bar{x}))}$$

where o_k is the k-th output of the dense layer.
As a final step, the model assigns to $\bar{x}$ the tag $\bar{t}$ with highest probability.

Layer	Out dimension
LSTM	$[l_w, 32]$
Dense	$[l_w, K]$
Softmax	$[l_w, K]$

Table 9: List of layers of the tag decoder.

6 Experiments

In this section we describe in details the experiments that we have conducted on our model. We use F1 score as the accuracy measure to compare models. Every model is trained for 15 epochs and early stopping is not enabled. We use the *RMSprop* optimizer, and we set batch size to 128 and learning rate to 0.001. As mentioned in Section 3, we have explored only on Task 1 and Task 3.

6.1 Hardware and implementation details

We work on a computing cluster with the following technical characteristics: 48 Intel Xeon E5-2680 2.50 GHz CPUs, 250 GB DRR4 1866 MHz RAM, and 2 GeForce GTX TITAN X GPUs. The model has been implemented in Keras and runs on Tensorflow GPU 1.4.0.

6.2 Model performance and comparison

We compare our CNN-CNN-LSTM model with the BiLSTM-CRF model proposed by [4]. In terms of training time, our model achieves an average speed up of 3x on a single epoch, when training of the full dataset. The accuracy of the two models is reported in Table 10 and Table 11. The BiLSTM-CRF model slightly outperforms our CNN-CNN-LSTM model, however, at significantly greater computational cost. Our error analysis, conducted on the confusion matrices of the two models, has not shown relevant difference in prediction between the two models.

	Precision	Recall	F1 Score	Avg epoch time (s)
Task 1	87.23 ± 0.20%	87.84 ± 0.32%	87.26 ± 0.20%	166
Task 3	94.30 ± 0.16%	94.80 ± 0.14%	94.41 ± 0.15%	164

Table 10: Accuracy of the CNN-CNN-LSTM model on validation dataset.

	Precision	Recall	F1 Score	Avg epoch time (s)
Task 1	90.06%	90.22%	89.66%	510
Task 3	95.15%	95.41%	95.09%	490

Table 11: Accuracy of the BiLSTM-CRF model on validation dataset. Scores taken from [4].

6.3 Active learning

We show the results of our active learning algorithm with the CNN-CNN-LSTM model. For each experiment, we run 15 cycles of training. The first training cycle is performed

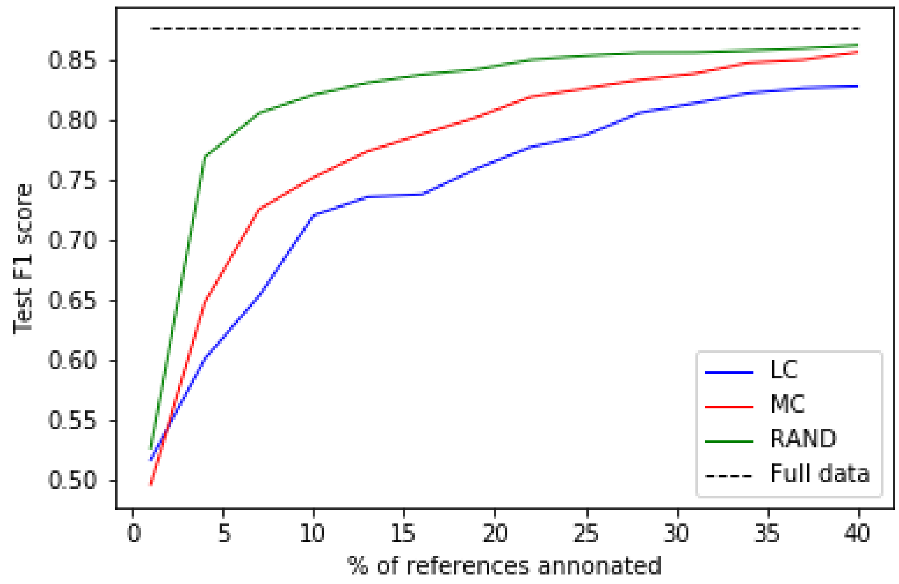


Figure 4: Task 1 F1 score on test dataset.

on a constrained random sampling of the dataset that guarantees a minimum number of tokens for each tag. The remaining sampling are based on entropy uncertainty with fixed policy, i.e. we do not change how we select data from the unlabeled pool between training cycles. In detail, in the first cycle the model is trained on 1.2% (Task 1) or 0.9% (Task 3) of the references in our training dataset, then at each cycle we add 3% of the references. We have tested our algorithm with three policies: lowest entropy (or MC, most confident), highest entropy (or LC, least confident), and random sampling (RAND).

Figure 4 shows the trend of the F1 score on the test set in Task 1. Random sampling grows steeply in the very first iterations, but the increase drops after few iterations. MC has a more uniform trend, and both metrics tend to converge to the same score when one-third of the dataset is reached. Conversely, LC shows a non-strictly monotone growth and does not achieve the same accuracy.

Sampling	Best F1	Average F1
Highest entropy (LC)	83.73%	82.39 ± 1.51%
Lowest entropy (MC)	85.49%	85.04 ± 0.48%
Random	85.78%	85.63 ± 0.14 %

Table 12: Task 1 F1 scores on validation dataset, using 34.2% of the total references.

Figure 5 depicts confidence margins, simply computed as the interval between the best and worst model at each iteration. As expected, the most noisy metric is LC, since it

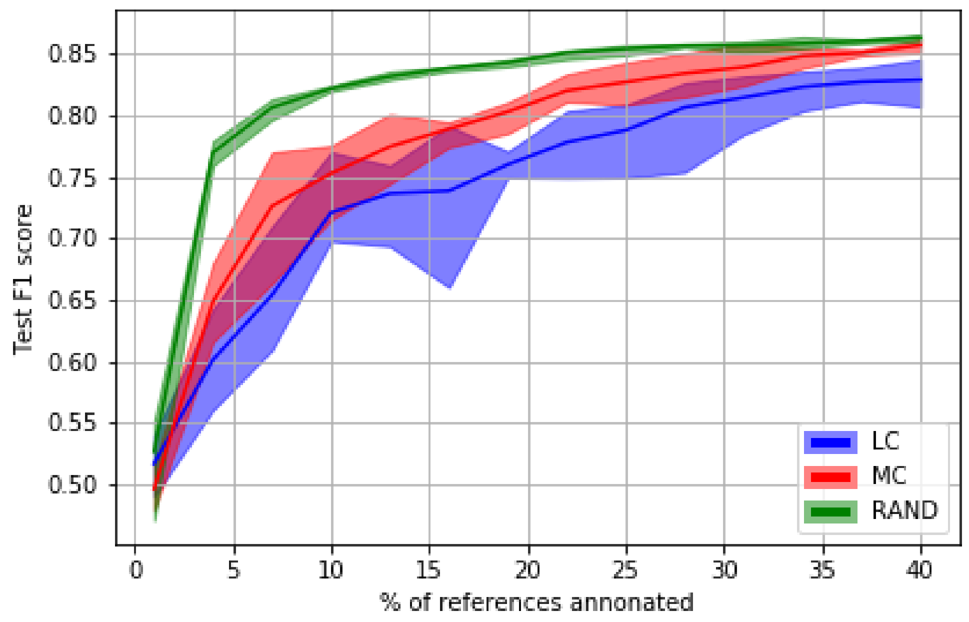


Figure 5: Task 1 F1 score on test dataset with confidence margins.

Sampling	Best F1	Average F1
Highest entropy (LC)	84.75%	83.39 ± 1.36%
Lowest entropy (MC)	86.48%	85.88 ± 0.59%
Random	85.86%	85.70 ± 0.24%

Table 13: Task 1 F1 scores on validation dataset, using 40.2% of the total references.

samples the highest-entropy data. MC tends to converge to more stable values after a few iterations, while random sampling shows to be extremely stable. More considerations are proposed later in the report.

Table 12 and Table 13 show results of the metrics on the validation dataset after querying, respectively, 33% and 39% of the references. The tables confirm the trend of Figure 4 and Figure 5. The LC metric is the least accurate, with worst results and highest margins. The MC and random metrics achieve similar results, and in the best case, MC achieves approximately 99% of the accuracy of the full-trained model. For Task 3, we simply report results without further investigation. Task 3 shows highly noisy learning curves, therefore, we do not report charts on the F1 test scores. The model achieves outstanding results with extremely low percentages of annotated references, 1% is enough to achieve more than 90% F1 on test data. We report the accuracy of the model on validation data in Table 14. We conclude that Task 3 tags are extremely redundant, as we can reach the same accuracy of the fully-trained model with just 19% of the sequences.

Sampling	Best F1	Average F1
Highest entropy (LC)	94.76%	94.41 ± 0.35%
Lowest entropy (MC)	93.62%	93.42 ± 0.20%
Random	94.19%	93.72 ± 0.70%

Table 14: Task 3 F1 scores on validation dataset, using 18.9% of the total references.

6.4 Data analysis

We investigate how the three different metrics affect the distribution of sampled data in Task 1. Tag distribution in training dataset (Figure 6) is considerably skewed, and it resembles a power law. In particular, the *title* tag accounts for the largest part of the dataset, especially in the test and validation datasets. Conversely, a large number of tags is considerably underrepresented in entire dataset.
The first iteration significantly affects tag distribution. Random sampling maintains its tag distribution as the original one, LC excessively focuses on underrepresented tags, and MC focuses on the most represented tags. As a result, random achieves the best accuracy after the first iteration as it trains on tags uniformly, while LC achieves the worst accuracy as it trains on tags that are not dominant in both training and test datasets. We further observe that, at every iteration, random sampling maintains a distribution that is nearly identical to the original. After eleven iterations, LC and MC tend to converge to the original distribution, explaining the improvement in results. After thirteen iterations, MC has reached a tag distribution reasonably close to the original dataset, while LC still maintains significant differences. Charts of tag distributions are showed in Annex B.

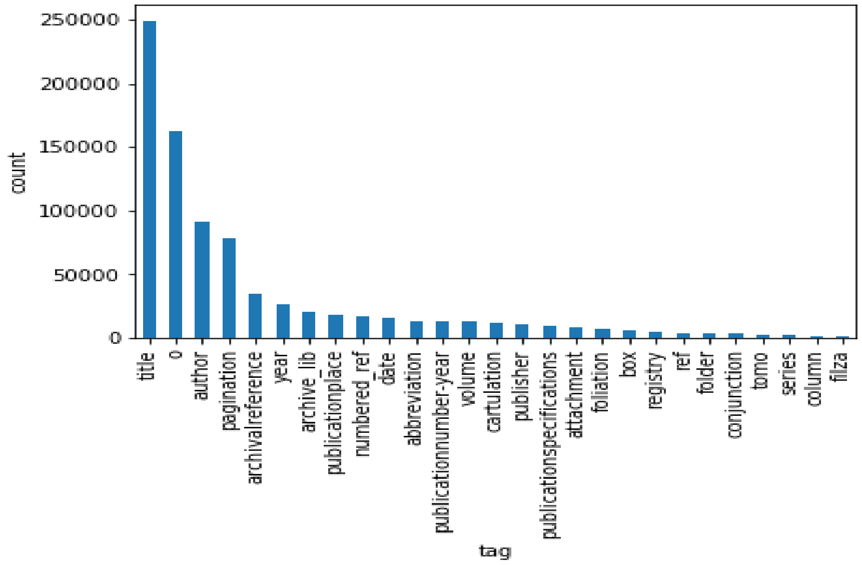


Figure 6: Distribution of Task 1 tags in training dataset.

7 Discussion

Our CNN-CNN-LSTM model reaches 87.26% (Task 1) and 94.41% (Task 3) of F1 score on our validation dataset, which is a result comparable to the one obtained by [4].
Uncertainty sampling allows us to achieve comparable results with only a fraction of our training dataset. In particular, for Task 1 the lowest entropy metric enables the model to match 99% of the F1 score on the fully trained model with just 40.2% of the original dataset, while random sampling allows us to match 98% of performance with 34.2% of the dataset. We observe that highest entropy is not the most suitable approach for our dataset, as it present a non-uniform distribution of the labels. A homogeneous dataset would benefit more from such an approach. For instance, our dataset could be split according to the classification of Task 2, since it represent an important discrimination of the reference nature. A dataset of primary references would consists in less underrepresented tags. Alternatively, if are allowed to collect additional data, we can request references with diversified annotations. Conversely, if we are not allowed to modify our dataset, or we cannot afford to gather new data, different sampling techniques can be explored. A solution is to perform a constrained sampling, i.e. we guarantee that our sampling algorithm promotes sequences with diversified tags. Therefore, we avoid to focus on over and under represented categories.
Active learning on Task 3 has shown outstanding results, matching the same accuracy of the fully trained model with just 19% of the data. However, the reason strictly relies in the dataset, which is extremely redundant for the given task.

7.1 Further development

We propose some further development to our work. As explained in Section 5, since our first objective is to study active learning, we decided to do not invest time in tuning our model; therefore, we have relied on the parameters provided by [13]. However, our architecture slightly differs from the one proposed by [13], consequently, improvements can be achieve with an appropriated fine-tuning. In particular, we suggest to tune the CNN layers (i.e. filter and pool size) and the number of hidden units in the LSTM layer.
Over the course of the project, we have investigated other strategies to improve our active learning algorithm, which we could not fully develop due to the shortage of time. Firstly, we have explored a hybrid uncertainty sampling technique which samples from both the most and the least confident instances. However, the ratio between these two quantities is a fundamental parameter that needs to be addressed in detail. Another important strategy is dataset generalization (described in Annex A). Dataset generalization allows the model to train on a broader and richer dataset, which may prove useful to uniform our labellings. In addition, dataset generalization can be obtained with cheap data, as compared with humanities data which is costly to collect.

7.2 Conclusion

In this report, we have discussed some methods that can prove useful to reduce the quantity of data necessary to train a model. We have shown that state-of-the-art accuracy in a specific NER task can be achieved with a CNN-CNN-LSTM model, which is significantly faster to train compared to a fully LSTM model. We have proposed uncertainty sample as an active learning technique, and we have discussed different uncertainty metrics. However, some research on different uncertainty metrics can still be done to improve our results.

References

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, Mar 1994.

[2] Jason P. C. Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *CoRR*, abs/1511.08308, 2015.

[3] Giovanni Colavizza and Matteo Romanello. Annotated References in the Historiography on Venice: 19th–21st centuries, 2017.

[4] Rodrigues Alves Danny, Giovanni Colavizza, and Frederic Kaplan. Deep Reference Mining from Scholarly Literature in the Arts and Humanities. *Frontiers in Research Metrics and Analytics, forthcoming*, 2017.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. 9:1735–80, 12 1997.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[9] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, 2010. ELRA.

[10] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[11] Ozan Sener and Silvio Savarese. A geometric approach to active learning for convolutional neural networks. *CoRR*, abs/1708.00489, 2017.

[12] Burr Settles. *Active Learning.* Morgan and Claypool Publishers, 2012.

[13] Yanyao Shen, Hyokun Yun, Zachary C. Lipton, Yakov Kronrod, and Animashree Anandkumar. Deep active learning for named entity recognition. *CoRR*, abs/1707.05928, 2017.

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[15] Keze Wang, Dongyu Zhang, Ya Li, Ruimao Zhang, and Liang Lin. Cost-effective active learning for deep image classification. *CoRR*, abs/1701.03551, 2017.

A. Dataset generalization

This annex describes an important component of our work that we could not fully develop: the generalization of our training dataset.
References from humanistic literature are usually collected from books and documents, which consequently must be parsed by means of OCR techniques. Annotations are not provided, and labeling references is an expensive and time-consuming procedure. Conversely, scientific literature lies in a utterly different context. Large collections of scientific references can be easily retrieved from the web, and they do not need to be parsed nor labeled. We have collected a corpus of more than 20'000 references from literature in computer science [2] to study if and how references from different domains can improve the accuracy of our model. Our assumption is that such references are significantly faster and cheaper to collect, and that our model is sufficiently flexible to train on a large amount of heterogeneous data without losing generalization (i.e. without overfitting the scientific references).

Methodology

We have collected the references in form of Bibtex files. Bibtex is a widely used reference management software for formatting lists of references in Latex documents[3]. An example of bibtex reference is the following:

@BookFisch86,
author = Max H. Fisch,
title = Peirce, Semeiotic, and Pragmatism: Essays,
publisher = Indiana University Press,
year = 1986,
note = edited by K.I.Ketner and C.J.W.Kloesel,
topics = area.semiotics,area.peirce,
address = Bloomington, IN

Bibtex references, in contrast with footnotes of humanistic publication, are rigorously defined and can be parsed automatically. Every element of the reference is annotated, and the taxonomy is similar to ours of Task 1. Bibtex files can be naturally converted into a dataset similar to our CoNLL training set. However, the task poses some challenges as Bibtex files may not follow coherent styling rules. In details, our conversion must guarantee that:

- References follow a fixed structure, that is, the taxonomy and the ordering of the tags is the same for every reference.
- Punctuation is treated uniformly, e.g. comma and dots are independent tokens.

[2]The full corpus of references can be found here: https://liinwww.ira.uka.de/bibliography/Introduction.html.
[3]https://en.wikipedia.org/wiki/BibTeX

- Latex encoding is properly formatted into UTF-8 encoding.
- The dataset is built form difference source and the distribution of token is not excessively skewed on specific elements, e.g. too many tokens of a particular journal in databases.

To guarantee a coherent taxonomy and structure of the references, we have defined the following schema:

author, title, publisher, volume, tomo, year, pagination

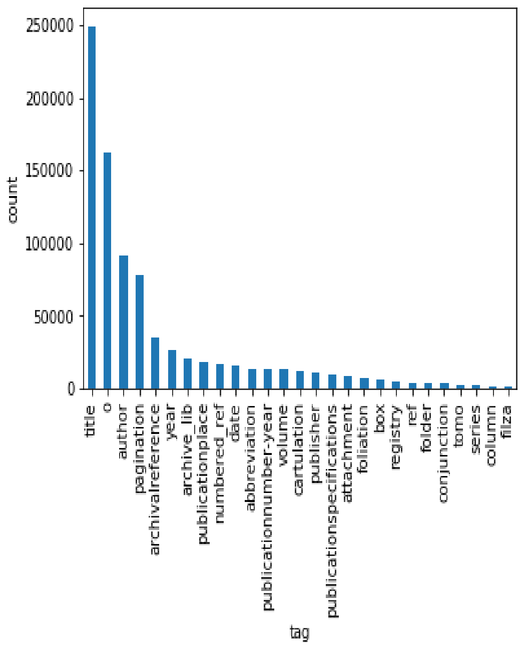
As some Bibtex tags named differently, we map tags in the following way:

author:	author
title:	title
journal:	publisher
volume:	volume
number:	tomo
year:	year
pages:	pagination

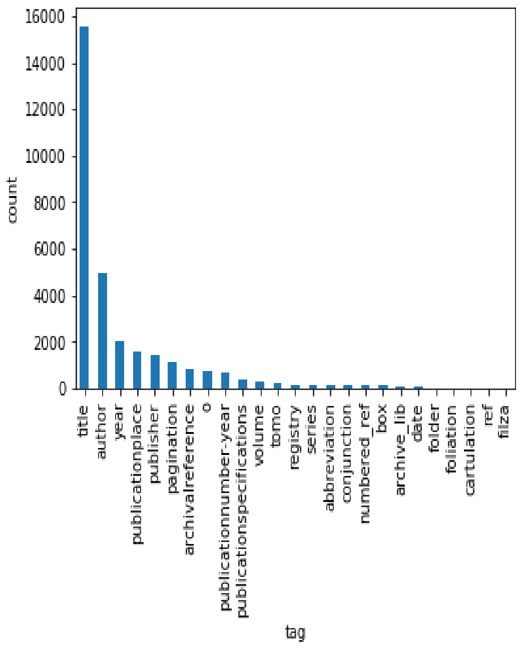
Preliminary experiments

We could not extensively investigate our generalized dataset, however, it has shown promising results. In particular, the lowest entropy uncertainty sampling can benefit from our dataset generalization, as it mainly focuses on the same tags (e.g. *author* and *title*). Some preliminary experiments have shown that our corpus of computer science references increases the F1 validation score of the lowest entropy setup from 84.41% to 85.22%, with only 28.2% of references from our humanities training dataset.

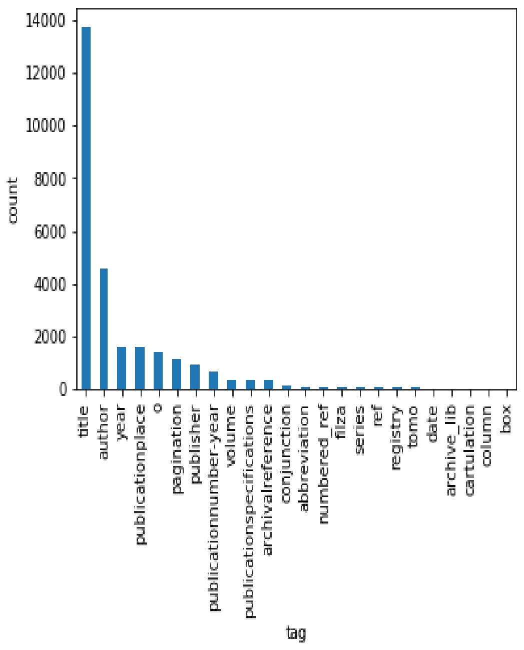
B. Task 1 tag distributions



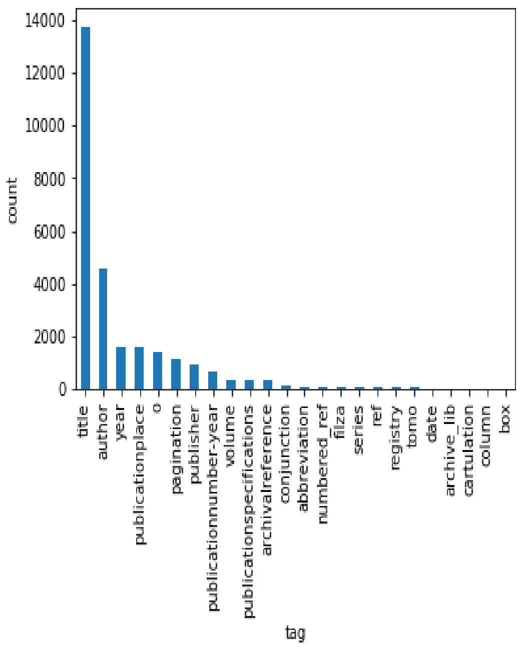
(a) Training dataset.



(b) Test dataset.

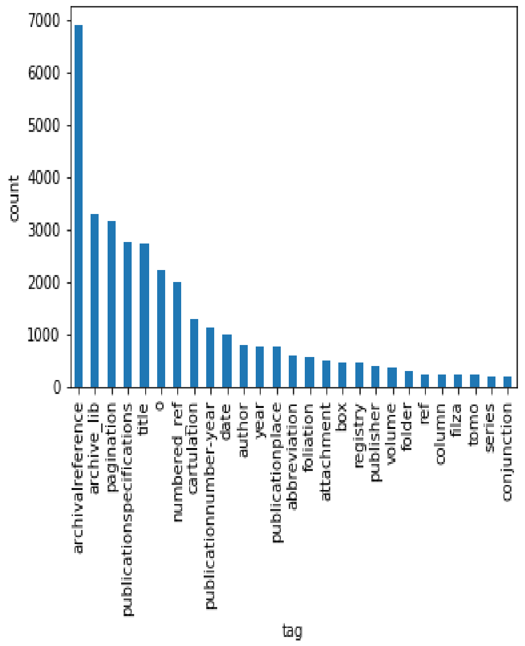


(c) Validation dataset.

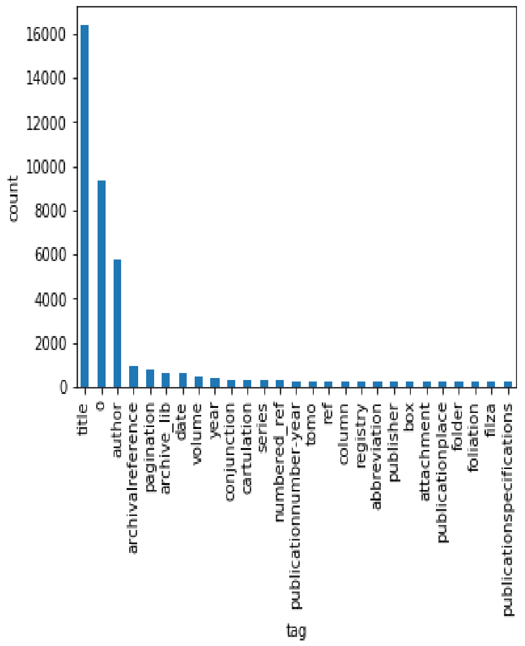


(d) Initial sampling on training set (1.2% of total sequences.).

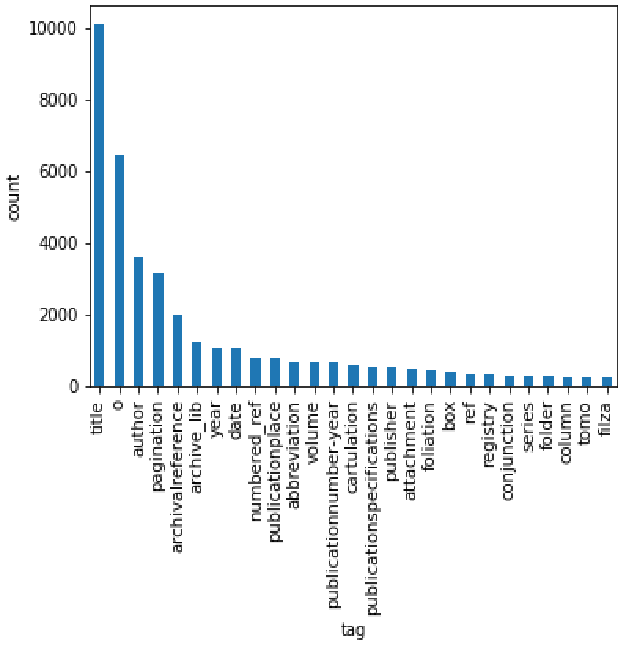
Figure 7: Tag distribution in datasets for Task 1.



(a) Highest entropy (LC).

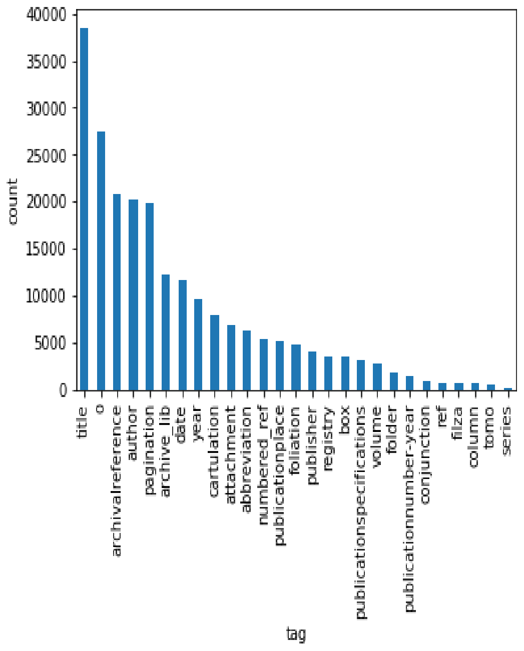


(b) Lowest entropy (MC).

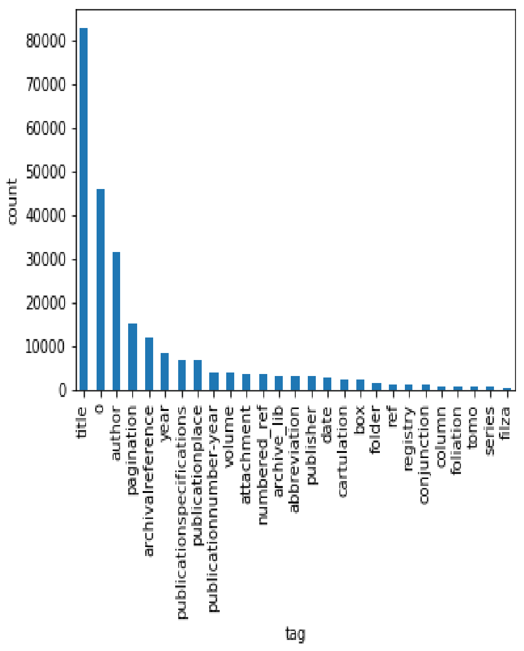


(c) Random.

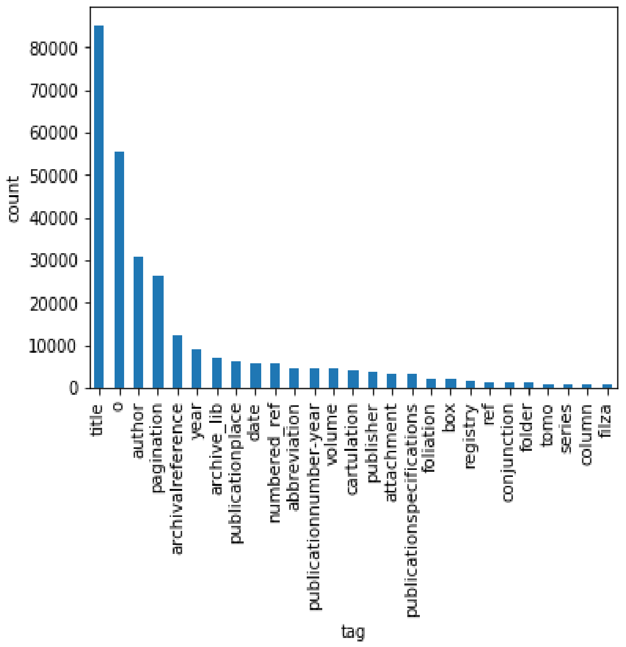
Figure 8: Tag distribution after first iteration. Annotated references: 4.2%



(a) Highest entropy (LC).

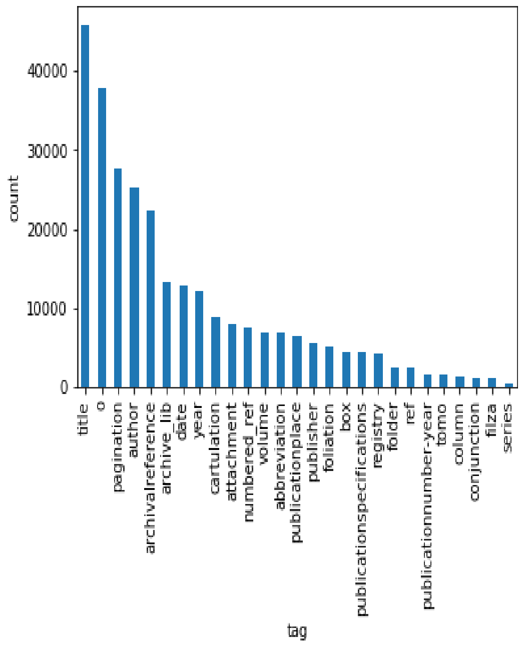


(b) Lowest entropy (MC).

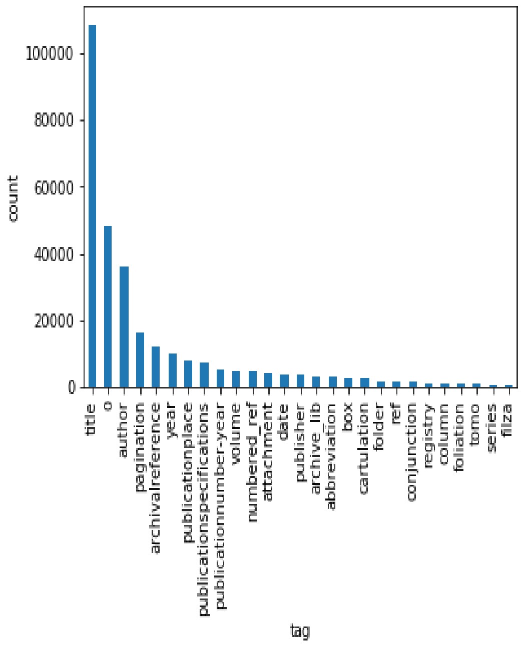


(c) Random.

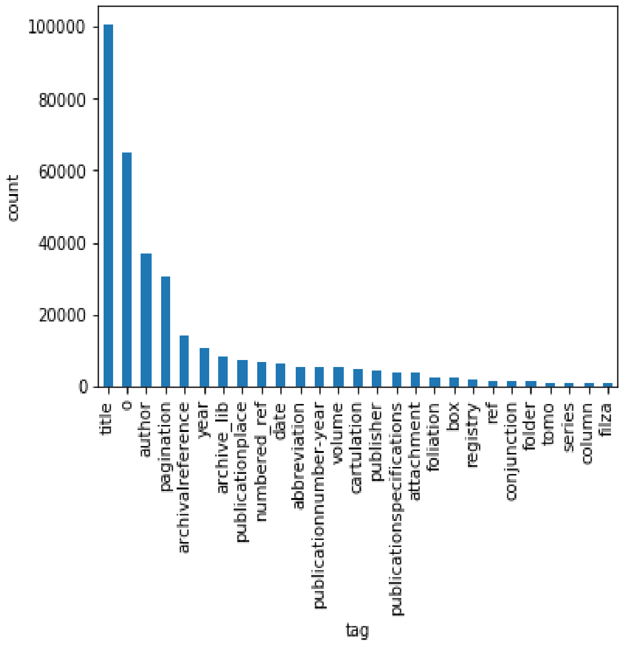
Figure 9: Tag distribution after 11 iterations. Annotated references: 34.2%



(a) Highest entropy (LC).



(b) Lowest entropy (MC).



(c) Random.

Figure 10: Tag distribution after 13 iterations . Annotated references: 40.2%