N.N. (Eds.): Proceedings of IJCNLP '13
6th International Joint Conference on Natural Language Processing, Nagoya, Japan, October 14-18

Learning Efficient Information Extraction on Heterogeneous Texts

Henning Wachsmuth
University of Paderborn, s-lab
Paderborn, Germany
hwachsmuth@s-lab.upb.de

Benno Stein
Bauhaus-Universität Weimar
Weimar, Germany
benno.stein@uni-weimar.de

Gregor Engels
University of Paderborn, s-lab
Paderborn, Germany
engels@upb.de

Abstract

From an efficiency viewpoint, information extraction means to filter the relevant portions of natural language texts as fast as possible. Given an extraction task, different pipelines of algorithms can be devised that provide the same precision and recall but that vary in their run-time due to different pipeline schedules. While recent research investigated how to determine the run-time optimal schedule for a collection or a stream of texts, this paper goes one step beyond: we analyze the run-times of efficient schedules *as a function of the heterogeneity* of the texts and we show how this heterogeneity is characterized from a data perspective. For extraction tasks on heterogeneous big data, we present a self-supervised online adaptation approach that learns to predict the optimal schedule depending on the input text. Our evaluation suggests that the approach will significantly improve efficiency on collections and streams of texts of high heterogeneity.

1 Introduction

Information extraction analyzes natural language text in order to find relevant information about entities and the events they participate in. An extraction task often requires to fill event templates with considerable numbers of slots. Such a task implies several analysis steps, e.g. certain types of entity and relation extraction, and it is therefore typically tackled with a pipeline $\Pi = \langle \mathbf{A}, \pi \rangle$, where $\mathbf{A}$ is a set of extraction algorithms and π a schedule that prescribes the order of algorithm application. The information sought for is anchored in text units of a certain size, e.g. in a sentence or paragraph.

In times of big data, the run-time efficiency of information extraction receives much attention in research and industry (Chiticariu et al., 2010). Among others, a growing need for business intelligence can be regarded as the driving force behind. This trend is equally observed by consulting companies who see evolving markets for predictive analytics (Harper, 2011), by global software players who exploit big data for decision making (White, 2011), and by researchers who seek to annotate tables at web scale (Limaye et al., 2010).

Generally, a pipeline $\Pi = \langle \mathbf{A}, \pi \rangle$ can be sped up by parallelization (Agichtein, 2005), if given enough resources, or by using faster but less effective algorithms (Al-Rfou' and Skiena, 2012). In addition, information extraction can always be approached as a filtering task as discussed in detail in (Wachsmuth et al., 2013b): By filling a template slot, each algorithm in $\mathbf{A}$ implicitly classifies certain units of an input text as relevant. Only these units need to be filtered for the next algorithm in π. As a result, a smart schedule π will often significantly improve the overall extraction efficiency. If the input requirements of all algorithms in $\mathbf{A}$ are met within π, the effectiveness of Π (in terms of both precision and recall) will be maintained, since the output of Π exactly lies in the filtered text units (Wachsmuth and Stein, 2012).[1]

When given a big data filtering task, the designer of a pipeline faces two challenges: (1) How to determine the most efficient schedule for a set of extraction algorithms and a collection or a stream of texts? (2) How to maintain efficiency under heterogeneous text characteristics? With regard to the former challenge we resort to existing research (cf. Section 2). The latter becomes an issue where input texts are not fully known or come from different sources as in the web. Moreover, streams of texts can undergo substantial changes in the distri-

[1] The simplest filtering task is to extract a relation between two entity types, such as *<ORG> was founded in <TIME>*. E.g., the sentence *"Google was established by two Stanford students."* needs not to be filtered for relation extraction, as it contains no time entity. The schedule of the two implied entity recognition steps will affect the extraction efficiency.

bution of relevant information. We argue that such kinds of uncertainty and lack of a-priori knowledge cannot be tackled offline, but they require to learn and to adapt to the characteristics of input texts to avoid a noticeable efficiency loss.

1.1 Contributions and Outline

In this paper, we analyze to what extent the heterogeneity of natural language texts in the distribution of relevant information affects the efficiency of an information extraction pipeline. For a high heterogeneity, we propose an adaptation of the pipeline's schedule, which we address with online learning. Our learning algorithm maps basic linguistic characteristics of a text to run-times of pipelines and chooses the pipeline with the lowest predicted runtime. The algorithm learns self-supervised and it is language-independent. To measure the impact of heterogeneity, we evaluate the algorithm on precisely constructed text corpora of different heterogeneity. Our contributions are three-fold:

1. We develop a self-supervised online adaptation algorithm that learns the efficiency of information extraction pipelines (Section 3).
2. We quantify the heterogeneity of natural language texts with regard to the distribution of relevant information (Section 4).
3. We evaluate the need for online adaptation in efficient information extraction as a function of the heterogeneity of input texts (Section 5).

2 Related Work

One line of research on extraction efficiency refers to *declarative information extraction* (Shen et al., 2007). In particular, Krishnamurthy et al. (2009) created SYSTEMT to address the needs of enterprise extraction applications. SYSTEMT involves optimization strategies such as the ordering and integration of analysis steps (Reiss et al., 2008), but it is restricted to rule-based extraction.

In (Wachsmuth et al., 2011a), we introduced a method that optimizes the schedule of an arbitrary set of extraction algorithms. This method captures much optimization potential and it can be automated using techniques from artificial intelligence (Wachsmuth et al., 2013a). Unlike the approach in this paper, however, the method does not handle variances in the characteristics of input texts.

Our approach applies to all extraction tasks with dependencies between the relevant types of information. We target at template filling, which consists in relating a number of entities to events of predefined types (Cunningham, 2006). Recent research, e.g. (Jean-Louis et al., 2011), and major evaluation tracks, e.g. (Kim et al., 2011), show the ongoing importance of template filling.

We consider extraction pipelines that perform filtering, which have a long tradition (Cowie and Lehnert, 1996). Sarawagi (2008) sees the efficient filtering of relevant portions of input texts as a main challenge. In the pipelines we focus on, each algorithm takes on one analysis (Grishman, 1997). Other approaches such as *joint information extraction* (Choi et al., 2006) can be effective, but they are not suitable when efficiency is important.

van Noord (2009) trades parsing efficiency for parsing effectiveness by learning a heuristic filtering of useful parses. In contrast, we develop a self-supervised online learning algorithm to achieve efficient extraction without reducing effectiveness. While our approach works with every predefined relation and event type, arbitrary binary relations are found in self-supervised *open information extraction* (Fader et al., 2011). Self-supervised learning aims to fully overcome manual text labeling, mostly for learning language like McClosky et al. (2010). To our knowledge, we are the first to apply it for predicting extraction efficiency.

3 Learning Efficient Extraction

The run-time efficiency of an information extraction pipeline depends on the distribution of relevant information in its input texts (Wachsmuth and Stein, 2012). For situations where this distribution varies, we now present an approach that chooses a pipeline schedule depending on the text at hand. To maintain precision and recall, we consider only schedules that fulfill the input requirements of all algorithms employed in a pipeline.

3.1 Splitting the Pipeline

Most extraction tasks require some analyses (e.g. tokenization) to be performed on the whole input texts, as they are needed for most or all subsequent analyses. We exploit this notion in that we use the results of the first algorithms in a pipeline to predict the best schedule of the remaining algorithms. To this end, we split a pipeline into a fixed first part and a variable second part. We call the first part the *prefix pipeline*, denoted as Π_{pre}, and each second part $\Pi_1, \ldots, \Pi_k$ a *main pipeline*.

In general, k has an upper bound of $m!$ where m is the number of algorithms in a main pipeline.

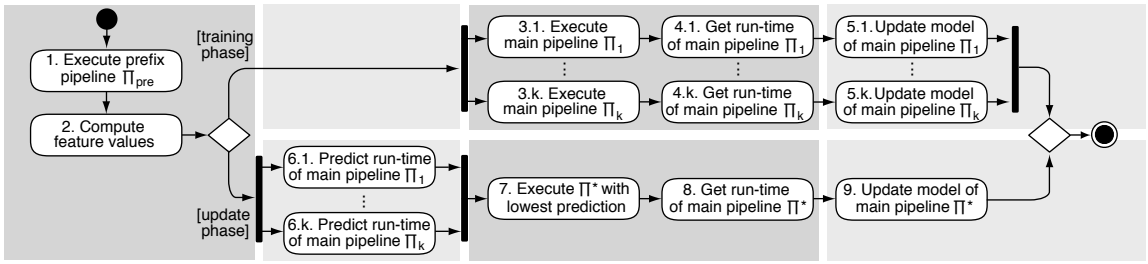

Figure 1: Illustration of the online adaptation algorithm with a prefix pipeline Π_{pre} and k main pipelines $\Pi_1, \ldots, \Pi_k$ for one input text, which either goes through the training phase or through the update phase.

Due to the algorithms' input constraints, however, k is normally much lower in practice. Also, there might be ways to restrict the set of candidate schedules to a reasonable selection, which is itself a non-trivial problem that is beyond the scope of this paper. In the following, we simply assume that $k \leq m!$ main pipelines are given.

3.2 Self-Supervised Learning of Run-times

For a collection or a stream of texts $\mathbf{D}$, our goal is to determine the most efficient main pipeline for each text in $\mathbf{D}$. We approach this goal as an online regression problem by learning to predict the run-time per text unit $t(\Pi_i)$ of each main pipeline $\Pi_i \in \{\Pi_1, \ldots, \Pi_k\}$. Based on the results of Π_{pre}, we represent each text $D \in \mathbf{D}$ as a feature vector $(x_1, ..., x_p)$ in order to create a regression model for each $t(\Pi_i)$. Concretely, we map the feature values $x_1^{(D)}, ..., x_p^{(D)}$ for D to a predicted run-time $\tilde{t}(\Pi_i)$. Then, D is processed by the main pipeline Π^* with the lowest prediction.

In this manner, learning can be approached self-supervised, as all training data is generated automatically: the feature values and the observed run-time $t(\Pi^*)$ of Π^* on D serve as a new training instance, and the prediction error is given by the difference between $\tilde{t}(\Pi^*)$ and $t(\Pi^*)$. Still, an explicit training set that is processed by all main pipelines helps to create initial regression models.

3.3 The Online Adaptation Algorithm

Let a prefix pipeline Π_{pre}, a set of main pipelines $\Pi_1, \ldots, \Pi_k$, and a collection or a stream of texts $\mathbf{D}$ be given. Then $\mathbf{D}$ is split into two parts $\mathbf{D_T}$ and $\mathbf{D_U}$ to serve the following two phases of the *online adaptation algorithm:*

1. *Training.* On each text $D \in \mathbf{D_T}$, execute Π_{pre} and each $\Pi_i \in \{\Pi_1, \ldots, \Pi_k\}$. Update the regression model of each Π_i wrt. the results of Π_{pre} and the run-time $t(\Pi_i)$ of Π_i on D.
2. *Update.* On each text $D \in \mathbf{D_U}$, execute Π_{pre} and predict $\tilde{t}(\Pi_i)$ for all $\Pi_i \in \{\Pi_1, \ldots, \Pi_k\}$. Execute the Π^* with the lowest prediction and update its regression model wrt. the results of Π_{pre} and the run-time $t(\Pi^*)$ of Π^* on D.

Figure 1 illustrates the online adaptation algorithm on one single text. An intuitive extension is to iteratively schedule each extraction algorithm separately. This would allow us to use detailed knowledge in later predictions. Since the first predictions are most decisive, however, we do not consider the iterative scenario here for simplicity.

3.4 Baselines and the Gold Standard

One way to evaluate our approach is to compare it to each pipeline (Π_{pre}, Π_i), $\Pi_i \in \{\Pi_1, \ldots, \Pi_k\}$. In practice, relying on such a fixed pipeline involves the danger of choosing a slow one. Hence, we also consider two baseline approaches below:

1. *Random baseline.* For each text $D \in \mathbf{D_U}$, choose a main pipeline (pseudo-) randomly.
2. *Optimal baseline.* For each text $D \in \mathbf{D_U}$, choose the main pipeline that has achieved the best overall run-time on $\mathbf{D_T}$.

The optimal baseline serves as a strong competitor on homogeneous collections and streams of texts, where it will often find the run-time optimal fixed pipeline. In contrast, the random baseline appears rather weak, but it will never fail completely.

Besides, we compute the *gold standard* below, i.e., an oracle that knows the fastest main pipeline for each text beforehand. The gold standard defines the upper ceiling for a set of main pipelines, thereby quantifying the general optimization potential. In that, it helps to evaluate whether online adaptation is suitable for the input texts at hand.

4 The Heterogeneity of Texts

The introduced algorithm is domain-independent and language-independent. It targets at situations

where input texts are heterogeneous in content or style, as is typical for the results of an exploratory web search. From an extraction perspective, the heterogeneity of a collection or a stream of texts $\mathbf{D}$ can be regarded as the extent to which the texts in $\mathbf{D}$ vary in the distribution of instances of the relevant *information types* $C_1, \ldots, C_m$, i.e., the entity, relation, and event types to be extracted.

As motivated in Section 1, text units need to be analyzed only if they may contain instances of all relevant information types. Hence, a pipeline's efficiency depends on the *density* $\rho_i(\mathbf{D})$ of each type C_i in the input texts in $\mathbf{D}$. Here, the density corresponds to the fraction of text units in $\mathbf{D}$ that contain an instance of C_i. The density $\rho_i(D)$ of C_i in a single text $D \in \mathbf{D}$ can be defined accordingly. Now, differences in the run-time per text unit of a pipeline mainly result from varying densities $\rho_i(D)$. In this regard, the heterogeneity of $\mathbf{D}$ can be quantified by measuring the variance of all densities in the texts in $\mathbf{D}$. The outlined considerations give rise to the following measure:

Averaged Deviation Let $\mathbf{C} = \{C_1, \ldots, C_m\}$ be the set of relevant information types for an extraction task, and let $\sigma_i(\mathbf{D})$ be the standard deviation of the density $\rho_i(\mathbf{D})$ of $C_i \in \mathbf{C}$ in a collection or a stream of texts $\mathbf{D}$. Then, the averaged deviation of $\mathbf{C}$ in $\mathbf{D}$ is

$$\sigma(\mathbf{C}|\mathbf{D}) = \frac{1}{m} \cdot \sum_{i=1}^{m} \sigma_i(\mathbf{D}).$$

We compute exact values $\sigma(\mathbf{C}|\mathbf{D})$ in Section 5 to measure the impact of heterogeneity. In general, the averaged deviation can also be estimated on a sample of texts. For illustration, Table 1 lists the deviations for the three most common named entity types in the German part of the *CoNLL'03 corpus* (Tjong Kim Sang and De Meulder, 2003), in the *Revenue corpus* (Wachsmuth et al., 2010), in a sample of the German Wikipedia (the first 10,000 articles according to internal page ID), and in the *LFA-11 smartphone corpus*, which is a web crawl of blog posts (Wachsmuth and Bujna, 2011). Here, we recognized entities using *Stanford NER* (Finkel et al., 2005; Faruqui and Padó, 2010).

Different from other sampling-based efficiency estimations, cf. (Wang et al., 2011), the averaged deviation does *not* measure the typical characteristics of input texts, but it quantifies how much these characteristics vary. By that, it helps pipeline designers to decide whether an online adaptation of pipeline schedules is needed to ensure efficient extraction. However, its current form leaves unclear how to compare deviations across tasks. In future work, a solution will be to normalize the averaged deviation—either with respect to a reference corpus or with respect to the given task, e.g. to a situation where all schedules perform equally well.

Information type C_i	$\sigma_i(\mathbf{D}_{\text{co}})$	$\sigma_i(\mathbf{D}_{\text{rv}})$	$\sigma_i(\mathbf{D}_{\text{wk}})$	$\sigma_i(\mathbf{D}_{\text{bp}})$
Person entities	18.4%	11.1%	15.9%	16.6%
Organization entities	18.1%	16.0%	14.1%	23.4%
Location entities	16.6%	10.9%	16.0%	15.3%
Averaged deviation	**17.7%**	**12.7%**	**15.3%**	**18.4%**

Table 1: The standard deviation σ_i of the density of three entity types in the CoNLL'03 corpus $\mathbf{D}_{\text{co}}$, the Revenue corpus $\mathbf{D}_{\text{rv}}$, a sample of 10,000 Wikipedia articles $\mathbf{D}_{\text{wk}}$, and a crawl of blog posts $\mathbf{D}_{\text{bp}}$. The bottom line shows their averaged deviations.

4.1 Text Corpora of Different Heterogeneity

For a careful evaluation of online adapation, we need input texts that refer to different levels of heterogeneity while being appropriate for analyzing a single and sufficiently complex filtering task at the same time. Most corpora for extraction tasks are too small to create reasonable subsets of different heterogeneity like $\mathbf{D}_{\text{co}}$ and $\mathbf{D}_{\text{rv}}$. As an alternative, a web crawl such as $\mathbf{D}_{\text{bp}}$ typically yields high heterogeneity, but it tends to include a large fraction of task-irrelevant texts. This conceals which efficiency differences are due to scheduling and, thus, is not suitable for controlled experiments.

To address this difficulty, we also use precisely constructed text corpora below, which consist of both original texts from existing corpora and artificially modified versions of these texts. Concretely, we modified a text by randomly duplicating one of its sentences, ensuring that each text in a corpus comprises a unique set of sentences while being grammatically valid. Thereby, we limit the online adaptation algorithm to a certain degree in learning linguistic features from the texts, but we gain that we can measure the benefit of online adaptation as a function of the averaged deviation.

5 Evaluation

We now present controlled experiments with the online adaptation algorithm on text corpora of different heterogeneity. The goal is to show the circumstances under which online adaptation will be needed for efficiency and, conversely, when a runtime optimal fixed pipeline appears sufficient.

5.1 Experimental Set-up

We consider the filtering task to extract financial forecast statements with resolvable time information and a monetary value for an organization. An example for such a statement is *"Apple's annual revenues could hit $400 billion by 2015"*. Accordingly, we have a set $\mathbf{C}$ of five relevant information types: time entities, money entities, organization entities, financial statements, and forecast events.

Algorithms Table 2 outlines the eight algorithms that we used in all experiments. We employed the *UIMA tokenizer*[2] to generate tokens and sentences, and the *TreeTagger* for part-of-speech tagging and chunking (Schmid, 1995). As in (Wachsmuth et al., 2011a), we relied on regexes for money and time recognition, while we applied support vector machines SD and FD for event detection. Organization names were extracted with the CRF-based *Stanford NER* (cf. Section 4). Finally, we implemented a rule-based time normalizer TN.

While, in general, our approach works for each kind of extraction algorithm, all algorithms in Table 2 perform only in-sentence extraction.

Pipelines As stated in Section 3, we split all pipelines into two parts. The prefix pipeline consists of the two algorithms used for preprocessing only: Π_{pre} = (UT, TT). For the reasons given below, we evaluated the following $k = 3$ main pipelines:

$$\Pi_1 = (\text{TR, FD, MR, SD, TN, OR})$$
$$\Pi_2 = (\text{TR, MR, FD, TN, OR, SD})$$
$$\Pi_3 = (\text{MR, TR, FD, OR, SD, TN})$$

Only 108 of the 6! = 720 possible main pipelines fulfill all dependencies listed in Table 2. Based on the method from (Wachsmuth et al., 2011a), we found that main pipelines starting with TR, MR, and FD (which are significantly faster than OR, SD, and TN) dominate others. We selected Π_1, Π_2, and Π_3, as they target at very different distributions of relevant information. While $k = 3$ appears small, it allows for a concise evaluation and suffices to discuss the impact of online adaptation. Still, we evaluate all 108 main pipelines in Section 5.4.

Features For generality, we restricted our view to simple features that neither require a preceding run over the training set nor exploit knowledge about the employed algorithms: (1) *Lexical statistics*, namely, the average and maximum number of characters in a token and of tokens in a sentence as well as the length of the text, (2) the average *run-times* per sentence of each algorithm in Π_{pre}, and (3) the frequencies of all *part-of-speech tags*.

In the feature evaluation in Section 5.4, we also have two further types that capture general characteristics of entities: (4) The frequency of each unigram and bigram of all *chunk tags* and (5) the frequencies of *regex matches* of a regex for arbitrary numbers and of a regex for upper-case words.[3]

Algorithm A		$t(A)$	**depends on**
UT	UIMA tokenizer	0.06 ms	–
TT	TreeTagger	0.59 ms	UT
TR	Time recognition	0.36 ms	UT
MR	Money recognition	0.64 ms	UT
OR	Organization recognition	2.52 ms	UT, TT
SD	Financial statement detection	3.95 ms	UT, TR, MR
FD	Forecast event detection	0.29 ms	UT, TT, TR
TN	Time normalization	0.95 ms	UT, TR

Table 2: Each evaluated algorithm A with its estimated average run-time per sentence $t(A)$ and the algorithm A depends on.

Learning Algorithm For run-time prediction, we applied *Stochastic Gradient Descent (SGD)* from *Weka 3.7.5* (Hall et al., 2009). After some preliminary tests, we set the learning rate of SGD to 0.01 for all experiments. Accordingly, we always used 10^{-5} for regularization and we always let SGD iterate 10 epochs over the training texts.

Datasets We constructed four partly artificial corpora $\mathbf{D_0}, \ldots, \mathbf{D_3}$ as motivated in Section 4.1. In case of $\mathbf{D_0}$, we randomly mixed 1500 texts of the German CoNLL'03 corpus and the Revenue corpus (cf. Section 4).[4] For $\mathbf{D_1}$, we took the 300 texts from $\mathbf{D_0}$ with the highest differences in the density of relevant information. We created modified versions of these texts in order to obtain a corpus size of 1500. $\mathbf{D_2}$ and $\mathbf{D_3}$ were built analogously for the 200 and 100 highest-difference texts, respectively. Table 3 lists all averaged deviations for the text unit "sentence". Where not stated otherwise, we used the first 500 texts of each corpus for training and the remaining 1000 for testing.

Efficiency The efficiency of all pipelines on each text was measured as the *run-time in milliseconds per sentence*, averaged over 10 runs. All run-times and their standard deviations were saved. For determinism, we loaded these run-times during eval-

[2] UIMA tokenizer, http://uima.apache.org/sandbox.html.

[3] We experimented with further regexes, but their impact was low. Therefore, we do not report on them in this paper.

[4] Notice that the evaluated set of features does not target at characteristics that are specific to the German language.

Information type C_i	$\sigma_i(\mathbf{D_0})$	$\sigma_i(\mathbf{D_1})$	$\sigma_i(\mathbf{D_2})$	$\sigma_i(\mathbf{D_3})$
Time entities	19.1%	22.5%	24.6%	25.9%
Money entities	19.8%	19.1%	20.4%	22.3%
Organization entities	19.3%	21.6%	22.4%	25.0%
Financial statements	7.1%	7.8%	8.9%	10.6%
Forecast events	3.8%	5.9%	6.7%	8.5%
Averaged deviation	**13.8%**	**15.4%**	**16.6%**	**18.5%**

Table 3: The standard deviation $\sigma_i(\mathbf{D})$ of the density of each relevant information types $C_i \in \mathbf{C}$ for each corpus $\mathbf{D} \in \{\mathbf{D_0}, \ldots, \mathbf{D_3}\}$ as well as the averaged deviation $\sigma(\mathbf{C}|\mathbf{D})$ of $\mathbf{C}$ in each $\mathbf{D}$.

uation instead of executing pipelines.[5] In case of the online adaptation algorithm, we also computed the *mean run-time prediction error* and the *classification error*, i.e., the fraction of texts the best main pipeline was not found for.

Effectiveness We omit to evaluate effectiveness here for lack of relevance: Our experiment setting, which is similar to (Wachsmuth et al., 2011a), yields no trade-off between efficiency and effectiveness, since we only consider schedules that fulfill all dependencies in Table 2. Thus, all pipelines always achieve the same precision and recall.

System and Software All experiments were conducted on a *2 GHz Intel Core 2 Duo MacBook* with 4 GB memory. The Java source code and the precomputed run-time files used in the evaluation can be accessed at http://www.arguana.com.

5.2 The Impact of Heterogeneity

We ran the online adaptation algorithm and the baselines from Section 3 on the test set of each of the corpora $\mathbf{D_0}, \ldots, \mathbf{D_3}$. Both the algorithm and the optimal baseline were trained on the respective training sets. Figure 2 illustrates the run-times of the main pipelines for each approach as a function of the averaged deviation and compares them to the gold standard. The shown confidence intervals result from the run-times' standard deviations σ, which ranged between 0.029ms and 0.043ms.

On the least heterogeneous corpus $\mathbf{D_0}$ with an averaged deviation of 13.8%, the online adaptation algorithm achieved an average run-time of 0.98ms per sentence, which is faster than the random baseline but slower than the optimal baseline at a low confidence level. For $\sigma(\mathbf{C}|\mathbf{D_1}) = 15.4\%$, the online adaptation algorithm succeeded with 0.87ms per sentence as opposed to 0.9ms of the optimal baseline. This gap gets significantly larger under higher averaged deviation. At 18.5%, both baselines are clearly outperformed, taking 37% and 40% more time on average, respectively.[6]

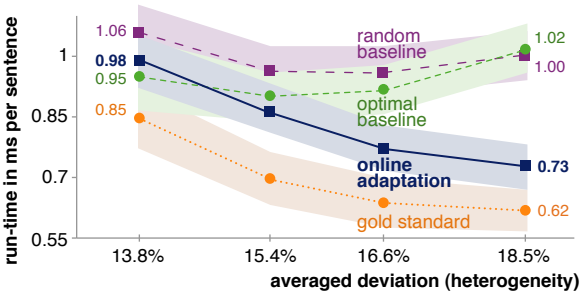

Figure 2: The average run-times of the main pipelines of both baseline approaches, the online adaptation algorithm, and the gold standard as a function of the averaged deviation. The background areas denote the 95% confidence intervals ($\pm 2\sigma$).

One reason for the weak result of online adaptation on $\mathbf{D_0}$ lies in the low optimization potential for that corpus: the main pipeline of the optimal baseline took only 12% more time on average than the gold standard (0.95ms vs. 0.85ms), which implies very small differences in the main pipelines' run-times. This does not only render online adaptation hard but also unnecessary. Conversely, Figure 3 shows an optimization potential of over 50% for $\mathbf{D_3}$. A reasonable hypothesis is therefore that online adaptation succeeds only on collections and streams of texts of high heterogeneity as indicated by large differences in the pipelines' run-times.

5.3 Run-time and Error Analysis

Figure 3 details the run-times of the three fixed pipelines, the online adaptation algorithm, and the gold standard on $\mathbf{D_0}$ and $\mathbf{D_3}$. The small black indicators denote the standard deviations.

In total, the fixed pipelines are 16% to 25% slower than the online adaptation algorithm on $\mathbf{D_3}$. The algorithm's run-time mainly breaks down into 0.51ms of the prefix pipeline and 0.73ms of the main pipelines, while the time for feature computations (0.03ms) and regression (0.01ms) is almost negligible. A similar situation is observed for $\mathbf{D_0}$. Here, online adaptation is worse only than (Π_{pre}, Π_1), which did best on 598 of the 1000 test texts. (Π_{pre}, Π_2) and (Π_{pre}, Π_3) had the lowest run-time on 229 and 216 texts, respectively.[7]

[5] Section 5.4 shows the effects of errors of measurement.

[6] On the training set of $\mathbf{D_3}$, the optimal baseline did not find the fastest main pipeline. This might be coincidence, but is, of course, more likely under higher heterogeneity.

[7] The numbers of texts sum up to more than 1000 because on some texts different pipelines performed equally well.

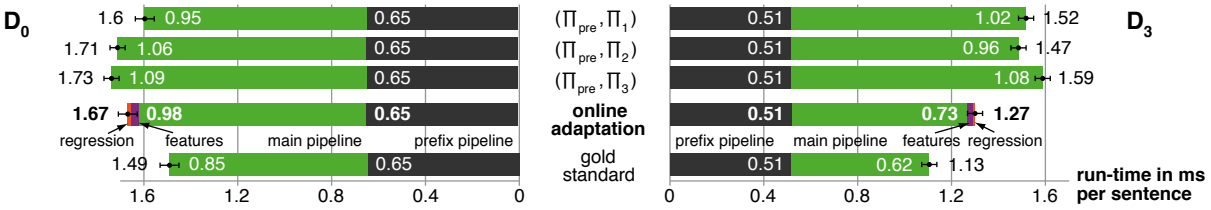

Figure 3: Average run-times of the fixed pipelines and online adaptation on the test sets of $\mathbf{D_0}$ and $\mathbf{D_3}$.

The online adaptation algorithm did not succeed on $\mathbf{D_0}$, since its mean run-time prediction error of 0.45ms was almost half as high as the average run-times to be predicted, which is too inaccurate under the small differences in the pipelines' run-times. As a result, for only 39% of the input texts, the best pipeline was chosen (i.e., a classification error of 0.61). However, the impact on $\mathbf{D_3}$ does not emanate from a low mean prediction error (that was in fact 0.24ms higher), but the classification error was reduced to 0.45. Consequently, the main reason lies in larger differences in the pipelines' run-times, which supports our hypothesis.

An insightful linguistic phenomenon is that the prefix pipeline Π_{pre} took significantly more time per sentence on $\mathbf{D_0}$ than on $\mathbf{D_3}$. Since the run-times of both algorithms in Π_{pre} scale linearly with the number of input tokens, the average sentence length of $\mathbf{D_0}$ must exceed that of $\mathbf{D_3}$. The reason is that shorter sentences tend to contain less relevant information. Hence, many sentences can be discarded after a few analysis steps, which increases the need for input-dependent scheduling.

5.4 Parameter Analysis

To give further evidence for the hypothesis from Section 5.2 and to test the applicability of online adaptation, we evaluated some major parameters:

Impact of Features For each of the five feature types in isolation, we trained a regression model on $\mathbf{D_0}$ and analyzed its impact. The *lexical statistics* achieved the lowest classification error (0.41), followed by the *run-times* (0.53). In terms of run-time prediction, the *regex matches* (0.46ms) and the *part-of-speech tags* (0.48ms) performed best, whereas the *chunk tags* failed both in classification (0.57) and prediction (0.58ms).[8] We used feature types 1–3 in all other experiments, since computing them took only 0.05ms per sentence on average. In contrast, the *regex matches* needed 0.16ms alone, which exceeds the difference between the optimal baseline and the gold standard on $\mathbf{D_0}$ (cf. Section 5.2) and, thus, renders the regex matches useless in the given setting.

[8]The low correlation of the prediction and classification error seems counterintuitive, but it indicates the limitations of these measures: E.g, a small prediction error can still be problematic if run-times differ only slightly, while a high classification error may have few negative effects in this case. If both errors are small, however, this normally implies success.

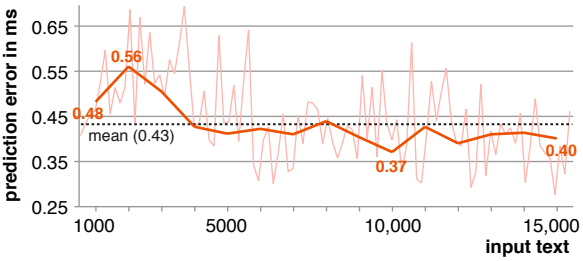

Figure 4: The mean run-time prediction error for the pipelines chosen by the online adaptation algorithm on 15,000 modified versions of the texts in $\mathbf{D_0}$ for training size 1. The values of the two interpolated learning curves denote the mean of 1000 and 100 consecutive predictions, respectively.

The *regex matches* emphasize the obvious need for a scheduling mechanism that avoids spending more time than can be saved later on. At the same time, such a mechanism should capture characteristics of a text that reliably model its complexity, which the evaluated features did not fully achieve.

Impact of Training Size We evaluated the online adaptation algorithm on $\mathbf{D_0}$ for nine training sizes between 1 and 5000. As the training set of $\mathbf{D_0}$ is limited, we created modified versions of its texts where needed (cf. Section 4.1). Online adaptation always did better than the random baseline but not than the optimal baseline except for training size 1. In case of 1000 or more training texts, the algorithm mimicked the optimal baseline, i.e., it chose Π_1 for about 90% of the texts.

Online Learning For training size 1, we ran the online adaptation algorithm on 15,000 modified versions of the texts in $\mathbf{D_0}$. Figure 4 shows two levels of detail of the algorithm's learning curve in its update phase. As the bold curve conveys, the mean run-time prediction error decreases on the

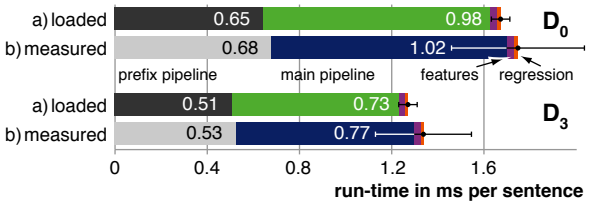

Figure 5: The average run-times and standard deviations of the online adaptation algorithm on $\mathbf{D_0}$ and $\mathbf{D_3}$ when run-times are a) loaded from a precomputed file or b) measured during execution.

first 5000 texts to an area around 0.4ms where it stays most of the time afterwards, though the light curve discovers many outliers. Still, online learning apparently seems to work well.

Overall Optimization Potential To measure the overall optimization potential of scheduling, we made an experiment with all $k = 108$ correct main pipelines on $\mathbf{D_0}$ (cf. Section 5.1).[9] The resulting average run-times of the fixed main pipelines span from 0.79ms per sentence in case of $\Pi_{best} = (\text{TR}, \text{FD}, \text{MR}, \text{OR}, \text{TN}, \text{SD})$ to 3.27ms of $\Pi_{worst} = (\text{OR}, \text{TR}, \text{TN}, \text{MR}, \text{SD}, \text{FD})$. This shows that the efficiency loss of choosing a wrong schedule can be very high. The online adaptation algorithm achieved 0.86ms with a mean run-time prediction error of 0.37ms. Altogether, 21 of the 108 schedules were used on the 1000 test texts, and the best schedule was chosen for 30% of the texts.

Real Execution As mentioned, we loaded all run-times from a precomputed file, which is not possible in case of real execution. In Figure 5, we compare the results of loading run-times to the run-times measured during the execution of the online adaptation algorithm. The measured values mainly differ in terms of larger standard deviations, i.e., 0.16ms on $\mathbf{D_0}$ and 0.12ms on $\mathbf{D_3}$. This seems to have a fairly negative effect on the main pipelines' run-times. However, the measured prefix pipeline run-times also exceed the saved ones, which suggests that the effect is due to a higher system load only. In any case, the measured run-time on $\mathbf{D_3}$ indicates that the online adaptation algorithm applies for practical applications.

Conclusion

In this paper, we analyze the efficiency of information extraction pipelines as a function of the heterogeneity of their input texts. In particular, we quantify heterogeneity with regard to the distribution of relevant information and we provide a self-supervised online adaptation algorithm that learns which pipeline schedule to choose for what input text in order to optimize efficiency while maintaining precision and recall. On this basis, we investigate the need for pipelines that adapt to their input within time-critical extraction tasks.

Our experiments suggest that the benefit of online adaptation is significant on heterogeneous collections and streams of texts: The online adaptation algorithm achieves gains of about 30% over the most efficient fixed schedule, which we see as important in times of big data. Conversely, when relevant information is uniformly distributed, finding an efficient fixed schedule appears sufficient, as approached in (Wachsmuth et al., 2013a).

A setting still to be evaluated refers to streams of input texts whose characteristics change slowly over time. Also, other extraction tasks may yield more insights. In order to decide how to approach a task at hand, a better understanding of the processing complexity of collections and streams of texts is required, to which our research contributes substantial building blocks.

In general, our self-supervised learning concept can be transferred to each natural language processing task that meets two basic conditions: (1) The task can be approached in different manners where each approach performs best for certain situations or inputs. (2) The performance of each approach can be measured or it is clear by definition.

[9] Notice that the training time increases linear to k, so a high k implies a high training overhead. For space reasons, we omit to report on training time here at all. However, training time will always be amortized in large-scale scenarios.

Acknowledgments

This work was partly funded by the German Federal Ministry of Education and Research (BMBF) under contract number 01IS11016A.

References

Eugene Agichtein. 2005. Scaling Information Extraction to Large Document Collections. *Bulletin of the IEEE Computer Society TCDE*, 28:3–10.

Rami Al-Rfou' and Steven Skiena 2012. SpeedRead: A Fast Named Entity Recognition Pipeline. In *Proc. of the 24th COLING*, pages 51–66.

Laura Chiticariu, Yunyao Li, Sriram Raghavan, and Frederick R. Reiss. 2010. Enterprise Information Extraction: Recent Developments and Open Challenges. In *Proc. of the 16th COMAD*, pages 1257–1258.

Yejin Choi, Eric Breck, and Claire Cardie. 2006. Joint Extraction of Entities and Relations for Opinion Recognition. In *Proc. of the 2006 EMNLP*, pages 431–439.

Hamish Cunningham. 2006. Information Extraction, Automatic. *Encyclopedia of Language & Linguistics*, 4:665–677.

Jim Cowie and Wendy Lehnert. 1996. Information Extraction. *Communications of the ACM*, 39(1):80–91.

Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying Relations for Open Information Extraction. In *Proc. of the 2011 EMNLP*, pages 1535–1545.

Jenny R. Finkel, Trond Grenager, and Christopher D. Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proc. of the 43nd Annual Meeting of the ACL*, pages 363–370.

Manaal Faruqui and Sebastian Padó. 2010. Training and Evaluating a German Named Entity Recognizer with Semantic Generalization. In *Proc. of KONVENS 2010*, pages 129–133.

Ralph Grishman. 1997. Information Extraction: Techniques and Challenges, In *International Summer School on Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, pages 10–27.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1):10–18.

Fern Harper. 2011. *Predictive Analytics: The Hurwitz Victory Index Report*, Hurwitz & Associates.

Ludovic Jean-Louis, Romaric Besançon, and Olivier Ferret. Text Segmentation and Graph-based Method for Template Filling in Information Extraction. In *Proc. of the 5th IJCNLP*, pages 723–731, 2011.

Jin-D. Kim, Sampo Pyysalo, Tomoko Ohta, Robert Bossy, and Jun'ichi Tsujii. 2011. Overview of BioNLP Shared Task 2011. In *Proc. of the BioNLP 2011 Workshop Companion Volume for Shared Task*, pages 1–6.

Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick R. Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu. 2009. SystemT: A System for Declarative Information Extraction. In *SIGMOD Records*, 37(4):7–13.

Girija Limaye, Sunita Sarawagi, Soumen Chakrabarti. 2010. Annotating and Searching Web Tables using Entities, Types and Relationships. In *Proc. of the VLDB Endowment*, 3(1):1338–1347.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*, MIT Press, Cambridge, MA, USA.

David McClosky, Eugene Charniak, and Mark Johnson. 2010. Automatic Domain Adaptation for Parsing. In *Proc. of the HLT/NAACL*, pages 28–36.

Gertjan van Noord. 2009. Learning Efficient Parsing. In *Proc. of the 12th EACL*, pages 817–825.

Frederick R. Reiss, Sriram Raghavan, Rajasekar Krishnamurthy, Huaiyu Zhu, and Shivakumar Vaithyanathan. 2008. An Algebraic Approach to Rule-Based Information Extraction. In *Proc. of the 2008 IEEE 24th ICDE*, pages 933–942.

Sunita Sarawagi. 2008. Information Extraction. *Foundations and Trends in Databases*, 1(3):261–377.

Helmut Schmid. 1995. Improvements in Part-of-Speech Tagging with an Application to German. In *Proc. of the ACL SIGDAT-Workshop*, pages 47–50.

Warren Shen, AnHai Doan, Jeffrey F. Naughton, and Raghu Ramakrishnan. 2007. Declarative Information Extraction using Datalog with Embedded Extraction Predicates. In *Proc. of the 33rd VLDB*, pages 1033–1044.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-independent Named Entity Recognition In *Proc. of the 7th CoNLL*, pages 142–147.

Henning Wachsmuth, Peter Prettenhofer, and Benno Stein. 2010. Efficient Statement Identification for Automatic Market Forecasting. In *Proc. of the 23rd COLING*, pages 1128–1136.

Henning Wachsmuth, Benno Stein, and Gregor Engels. 2011. Constructing Efficient Information Extraction Pipelines. In *Proc. of the 20th ACM CIKM*, pages 2237–2240.

Henning Wachsmuth and Kathrin Bujna. 2011. Back to the Roots of Genres: Text Classification by Language Function. In *Proc. of the 5th IJCNLP*, pages 632–640.

Henning Wachsmuth and Benno Stein. 2012. Optimal Scheduling of Information Extraction Algorithms. In *Proc. of the 24th COLING: Posters*, pages 1281–1290.

Henning Wachsmuth, Mirko Rose, and Gregor Engels. 2013. Automatic Pipeline Construction for Real-Time Annotation. In *Proc. of the 14th CICLing*, pages 38–49.

Henning Wachsmuth, Benno Stein, and Gregor Engels. 2013. Information Extraction as a Filtering Task. To appear in *Proc. of the 22th ACM CIKM*.

Daisy Z. Wang, Long Wei, Yunyao Li, Frederick R. Reiss, and Shivakumar Vaithyanathan. 2011. Selectivity Estimation for Extraction Operators over Text Data. In *Proc. of the 2011 IEEE 27th ICDE*, pages 685–696.

Colin White. 2011. *Using Big Data for Smarter Decision Making*, BI Research.