Stacked generalization for information extraction

Georgios Sigletos[1,2], Georgios Paliouras[1], Constantine D. Spyropoulos[1], Takis Stamatopoulos[2]

Abstract. This paper defines a new stacked generalization framework in the context of information extraction (IE) from online sources. The proposed setting removes the constraint of applying classifiers at the base-level. A set of IE systems are trained instead to identify relevant fragments within text documents, which differs significantly from the task of classifying candidate text fragments as relevant or not. The templates filled by the base-level IE systems are *stacked*, forming a set of feature vectors for training a meta-level classifier. Thus, base-level IE systems are combined with a common classifier at meta-level. The proposed framework was evaluated on three Web domains, using well known IE approaches at base-level and a variety of classifiers at meta-level. Results demonstrate the added value obtained by combining the base-level IE systems in the new framework.

1 INTRODUCTION

One of the most attractive topics in supervised machine learning is learning how to combine the predictions of multiple classifiers. The motivation for doing this derives from the opportunity to obtain higher prediction accuracy, while treating classifiers as *black boxes*, i.e. without considering the details of their functionality.

Stacked generalization or *stacking* [1] deals with the task of learning a meta-level classifier to combine the predictions of multiple base-level classifiers. The set of base-level classifiers is generated by applying different learning algorithms to a given data set. Alternative combination methods like boosting [2] and bagging [3] deal with multiple classifiers, generated however by applying the same learning algorithm to different versions of the data.

Research on stacking has primarily focused on classification. Each instance in the domain of interest is represented by a vector $< x_1 ... x_n , y >$ of attribute values, where y is the *class* attribute, whose value we wish to predict. To classify a new instance, the predicted class values of the base-level classifiers form a new vector that is assigned the final class by the meta-classifier.

In this article we investigate the effectiveness of stacked generalization in the task of information extraction (IE) from online sources: a form of shallow text processing that extracts relevant text pieces to populate a predefined template. However IE is naturally an identification task, rather than a classification one [4]. A rich variety of IE systems, e.g. [5, 6, 7, 8] are typically trained to identify relevant text fragments, i.e. sequences of tokens, within documents. There is only a small number of approaches [9, 10] that enumerate a high proportion of all possible text fragments that can be found within a document and then model the IE task as a binary classification one. In this latter case, the task is to learn whether or not a candidate fragment fills some template-slot. However, there are several problems associated with this approach such as the exponential number of candidate fragments and the artificially large number of negative examples.

Thus, the main contribution of this article is a new stacking framework that accommodates IE systems at base-level that are not required to perform classification. Given a document, the templates populated by the base-level IE systems are *stacked* to a single template, wherefrom a set of feature vectors is assembled for training a meta-level classifier. At runtime, this classifier decides whether a candidate fragment, among the ones predicted by the base-level IE systems, is relevant or not. The proposed framework was experimentally evaluated in three Web domains, using well known IE approaches at base-level and a variety of classifiers at meta-level. Results show a superior performance of stacking against both base-level IE systems and voting, for all domains.

Section 2 presents some background on stacked generalization and information extraction at meta-level. Section 3 describes the proposed framework. Section 4 presents the experimental results. Finally we conclude in section 5, discussing further improvements.

2 BACKGROUND

2.1 Stacking

The key idea behind stacking is to learn a meta-level (or level-1) classifier based on the output of base-level (or level-0) classifiers, estimated via cross-validation as follows:

Let D a dataset consisting of feature vectors, also referred to as level-0 data, and $L^1 ... L^N$ a set of N different learning algorithms. During a J -fold cross-validation process, D is randomly split into J disjoint parts $D^1 ... D^J$ of equal size and similar class distribution. At each *j*th fold, $j = 1..J$, the $L^1 ... L^N$ algorithms are trained on the training set $D \setminus D^j$ and the induced classifiers are applied to the test part D^j . The concatenated class predictions of the induced classifiers on each vector x_i in D^j , followed by the original class value $y_i(x_i)$, form a new set MD^j of meta-level vectors.

At the end of the cross-validation process, the union $MD = \cup MD^j$, $j = 1..J$, constitutes the full meta-level dataset, also referred to as level-1 data, which is used for training a meta-level classifier C^M . The learning algorithm that is employed at meta-level could be one of the $L^1 ... L^N$ or a different one. Finally, the $L^1 ... L^N$ learning algorithms apply to the entire dataset D inducing the final base-level classifiers $C^1 ... C^N$ to be used at runtime.

[1] Institute of Informatics and Telecommunications, NCSR "Demokritos", 15310, Aghia Paraskevi, Attikis, GREECE, email: {sigletos, paliourg, costass}@iit.demokritos.gr

[2] Departments of Informatics and Telecommunications, University of Athens, Panepistimiopolis, Athens, GREECE, email {takis@di.uoa.gr}

In order to classify a new instance, the concatenated predictions of all base-level classifiers $C^1...C^N$ form a meta-level vector that is finally assigned a class value by the meta-level classifier.

In [11] an extension of stacking was proposed, where each classifier outputs a class probability distribution for every example, instead of a single class. In the same work, *multi-response linear regression models* (MLR) were used as a meta-level classifier that proved to be highly effective compared to other classifiers. Other recent approaches to stacking include work presented in [12, 13].

2.2 Information extraction at meta-level

Despite the growing interest in combining machine learning algorithms and the application to some natural language parsing tasks such as part-of-speech tagging [14], which is fundamentally a classification task, this topic has received little attention by the IE community. The only relevant work is described in [9] where the IE task is transformed into a classification one, as mentioned in Section 1, using a set of four base-level extractors. Having done that, a *multistrategy approach* based on *voting* is used. Although this approach could be upgraded to stacking, it inherits the problems of treating IE as a classification problem, as explained in Section 1.

On the other hand, a rich variety of IE approaches e.g. [5, 6, 7, 8] do not externally enumerate all possible text fragments within a page. Such systems typically generalize a set of pattern-matching extraction rules from positive examples. At runtime, the induced patterns apply within a document, trying to match relevant text fragments. Therefore, it would be desirable to design an alternative stacking framework that can accommodate such IE systems.

3 STACKED GENERALIZATION FOR INFORMATION EXTRACTION

3.1 Definition of the extraction task

Let $F = \{f_1...f_W\}$ a set of W extraction *fields*, defining a *template* T, and d a document annotated by the domain expert with *instances* of those fields. A field *instance* is a pair $< t(s,e), f >$, where $t(s,e)$ is a text fragment, with s and e the indices of the start and end tokens of the fragment in page's token table, and $f \in F$ the related field. A field is typically a *target-slot* in T, while $t(s,e)$ is a *slot-filler*. In this article we assume that T is filled with pairs $< t(s,e), f >$. Table 1 shows a part of a Web page describing laptop products where relevant text is highlighted in bold. Table 2 shows the hand-filled template for this page.

Table 1. Part of a Web page describing laptop products.

…**TransPort ZX**   <font size="1"> **15"**XGA **TFT** Display   **Intel Pentium III 600 MHZ** 256k Mobile processor   **256 MB** SDRAM up to 1GB …

Table 2. Hand-filled populated template for the page of Table 1.

T			Short description for field f
$t(s,e)$	s,e	*Field* f	
Transport ZX	47, 49	model	Name of laptop's model
15"	56, 58	screenSize	Size of laptop's screen
TFT	59, 60	screenType	Type of laptop's screen
Intel Pentium III	63, 67	procName	Name of laptop's processor
600 MHZ	67, 69	procSpeed	Speed of laptop's processor
256 MB	76, 78	Ram	Laptop's ram capacity

The IE task can be defined as follows: *given a new document* d *and an empty template* T, *find all possible instances for each extraction field and populate* T. An extended approach to IE is to group field instances into higher-level concepts, also referred as *multi-slot* extraction [8]. However, the simpler single-slot approach addressed here covers a wide range of IE tasks and motivated the development of a variety of learning algorithms [5, 6, 7, 9, 10].

3.2 Stacking information extraction systems

Let $L^1...L^N$ a set of N learning algorithms, designed for IE tasks and D an annotated corpus, e.g. of Web pages describing laptop products. Let $E^1...E^N$ the IE systems that were built by training $L^1...L^N$ on D. Finally, define $T^1..T^N$ a set of templates populated by $E^1...E^N$ from a document d in D. Table 3 shows two templates T^1, T^2 by two fictitious IE systems E^1, E^2 for the page of Table 1.

Table 3. Templates populated by two IE systems for the page of Table 1.

T^1			T^2		
$t(s,e)$	s,e	f	$t(s,e)$	s,e	f
Transport ZX	47, 49	model	Transport ZX	47,49	manuf
15"	56, 58	screenSize			
TFT	59, 60	screenType	TFT	59, 60	screenType
Intel Pentium III	63, 67	procName	Intel Pentium	63, 66	procName
600 MHZ	67, 69	procSpeed	600 MHZ	67, 69	procSpeed
256 MB	76, 78	ram	256 MB	76, 78	ram
1 GB	81, 83	ram	1 GB	81, 83	HDcapacity

Examining Table 3 we note some disagreement in the predictions of the two IE systems: for two text fragments ("Transport ZX", "1GB") the predicted fields by E^1, E^2 disagree. Comparing to the hand-filled template of Table 2, we conclude that "Transport ZX" has been correctly identified as *model* only by the first IE system, while "1GB" has been incorrectly identified by both systems. Furthermore, the fragment "15"" has been identified only by E^1, while E^2 did not identify it at all. Finally, there is an overlapping case: the fragment "Intel Pentium III" has been correctly identified by E^1 as *procName*, while E^2 incorrectly predicted the same field for "Intel Pentium". This disagreement is better observed in the *stacked* template of Table 4.

Table 4. Stacked template of T^1 and T^2

s,e	$t(s,e)$	*Field by* E^1	*Field by* E^2	*Correct field*
47, 49	Transport ZX	model	manuf	model
56, 58	15"	screenSize	-	screenSize
59, 60	TFT	screenType	screenType	screenType
63, 66	IntelPentium	-	procName	-
63, 67	IntelPentium III	procName	-	procName
…	…	…	…	…
81, 83	1 GB	ram	HDcapacity	-

Constructing the *stacked* template of Table 4 is a straightforward process: all fragments $t(s,e)$ identified by E^1, E^2 in T^1, T^2 are inserted into an initial pool. Duplicate fragments are removed; two text fragments differ if either their start or end index differs. For the remaining *distinct* fragments, the predicted fields by E^1, E^2 are collected and appended with the correct field (last column in Table 4), according to the hand-filled template of Table 2.

Examining Table 4, the desirable performance task is to automatically fill the last column with the correct fields. The simplest idea is to use *voting* on the predictions of the IE systems. An alternative approach is to *learn to predict* the correct field for each distinct text fragment. The idea suggested in this paper is to create a feature vector for each entry of Table 4, i.e. for each

distinct fragment $t(s,e)$, and use the new vectors for training a common classifier. Table 5 shows the new feature vectors.

Table 5. Meta-level feature vectors for the stacked template of Table 4.

		Feature vectors		
s,e	*t(s,e)*	*Features by* E^1	*Features by* E^2	*class*
47, 49	Transport ZX	model,	manuf,	model
56, 58	15"	screenSize,	?,	screenSize
59, 60	TFT	screenType,	screenType,	screenType
63, 66	Intel<b>Pentium	?,	procName,	false
63, 67	Intel<b>Pentium III	procName,	?,	procName
…	…	…	…	…
81, 83	1 GB	ram,	HDcapacity,	false

Absence of prediction by an IE system is indicated by "?". If a text fragment doesn't exist in the hand-filled template it is classified as *false*. The remaining issue is to construct the full set of *feature vectors* that will be used for training a meta-level classifier, from the base-level training set that consists of *annotated documents*. This disparity between base-level and meta-level datasets motivated us to propose a new variant of the cross-validation methodology that samples from documents, rather than from feature vectors, as described in section 2.1.

3.3 The new stacking framework

The key idea behind stacking for IE is to learn a meta-level classifier based on the output of base-level IE systems, estimated via cross-validation as follows:

At the *j*th fold, $j=1..J$, of cross-validation, the N learning algorithms $L^1...L^N$ are trained on the document set $D \setminus D^j$ and the induced IE systems $E^1(j)...E^N(j)$ are applied to the test set D^j. For each document d in D^j, let $T^1...T^N$ the populated templates by $E^1(j)...E^N(j)$ respectively. A *stacked* template ST is assembled from $T^1...T^N$, as shown in section 3.2. A new feature vector is produced for each entry in ST, which is added to the meta-level dataset MD^j. At the end of the cross-validation process, the union $MD = \cup MD^j$ constitutes the full meta-level dataset, which is used for the training of a meta-level classifier C^M. Finally, the N learning algorithms are retrained on the entire dataset D inducing the base-level IE systems $E^1...E^N$ to be used at runtime. Table 6 presents an algorithmic description of the new stacking framework.

The vectors in the new meta-level dataset MD belong to $W+1$ classes, where W the number of fields in the domain of interest, plus the value "false". A vector classified as "false" indicates that the corresponding text fragment doesn't exist in the hand-filled template, and thus none of the IE systems should have predicted a field for it (e.g. the "1 GB" in Table 5).

At runtime, given a new document d, the base-level IE systems are used to identify relevant instances and fill the corresponding templates. A stacked template is created by the individual ones. For each entry in the stacked template a feature vector is created and finally classified by the meta-level classifier C^M. If the vector is classified into one of the W fields (i.e. it isn't a "false" prediction), the corresponding entry $< t(s,e), f >$ is inserted in the final template for d, otherwise ("false" prediction) the entry is rejected. The stacking framework at runtime is shown in Figure 1.

A major issue concerning cross-validation methodologies is *stratification*. Unlike classification tasks where a similar distribution of classes is maintained at each fold, in IE there is a different distribution of field instances in each document. Therefore

Table 6. The new stacking framework for information extraction.

procedure stacking_for_IE (D, J, $L^1...L^N$, L^M) **begin**
 $D^1...D^J$ = partition of D into J sets of documents of equal size
 for j = 1 **to** J **do begin**
  MD^j = {}
  for i = 1 **to** N **do**
   $E^i(j)$ = the IE system obtained by training L^i on $D \setminus D^j$
  foreach document d in D^j **do begin**
   for i = 1 **to** N **do**
    T^i = template populated by applying $E^i(j)$ to d
   ST = *create_stacked_template* (d, $T^1...T^N$)
   foreach entry, i.e. for each distinct $t(s,e)$, in ST **do begin**
    for i = 1 **to** N **do**
     $f^i \in \{f_1,...f_W,"?"\}$ = the field by $E^i(j)$ for $t(s,e)$
    $f \in \{f_1,...f_W, false\}$ = the correct field for $t(s,e)$
    $MD^j = MD^j \cup \text{vector} < f^1,...f^N, f >$
   end
  end
 end *// end of cross-validation*
 $MD = \cup MD^j$, $j=1..J$
 C^M = meta-classifier obtained by applying L^M on MD
 // Train the base-level IE systems
 for i = 1 **to** N **do**
  E^i = the base-level IE system obtained by training L^i on D
end

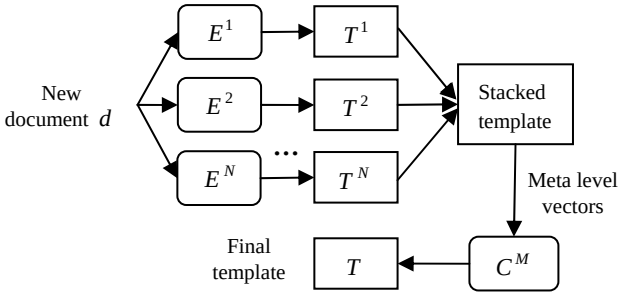

Figure 1. Stacking for information extraction at runtime.

it is extremely hard to even approximate the same distribution of fields in each fold. Despite the lack of explicit stratification, in our experiments we didn't encounter particular problems.

3.4 Stacking with confidence scores

Algorithms that learn pattern matching rules for IE typically determine an appropriate metric for evaluating the confidence of the patterns being learned. A straightforward extension of the proposed stacking framework is based on the idea that a predicted field for a text fragment $t(s,e)$ is accompanied by the confidence score of the pattern that matched that fragment. The exact procedure follows:

- Instead of predicting one of the W fields for each $t(s,e)$, each IE system generates a confidence score c^k for the field f^k, if a field is predicted at all. This is modeled by a W-element vector that contains zero values, except possibly for the *k*th position where c^k appears, i.e. $< 0,...c^k...0 >$.
- Each vector is *mapped* to a new one $< 0,...p^k...0 >$, where p^k is a probability estimate that corresponds to c^k and reflects the correctness of the prediction in a range between zero and one. The argument for performing this mapping is that confidence scores produced by different algorithms are not comparable nor

they bear any resemblance to probability estimates [9]. For example, in the $(LP)^2$ system [6] the confidence is measured through the number of wrong matches made by each pattern during training, while HMMs measure confidence by logarithmic values assigned by the *Viterbi* algorithm. The mapping of confidence scores to true probability estimates is done using a form of linear regression, as proposed in [9]. This allows us to adopt a similar multistrategy approach as that presented in [9] for comparison purposes.

- Finally, the output vectors by $E^1 ... E^N$ for $t(s,e)$ form a single one of $N * W$ elements, appended by the correct field.

The inner *foreach* loop in Table 6 is appropriately modified to handle the new setting. Table 7 shows the new meta-level vectors assembled by the stacked template of Table 4.

Table 7. Meta-level vectors using confidence scores.

	Feature vectors		
s,e	*Features by* E^1	*Features by* E^2	*class*
47, 49	0, 0, 0.92, 0, 0, 0, 0, 0,	0, 0.34, 0, 0, 0, 0, 0, 0,	model
56, 58	0, 0, 0, 0, 0, 0, 0.83, 0,	0, 0, 0, 0, 0, 0, 0, 0,	screenSize
59, 60	0, 0, 0, 0, 0, 0, 0, 0.85,	0, 0, 0, 0, 0, 0, 0, 0.91,	screenType
63, 66	0, 0, 0, 0, 0, 0, 0, 0,	0, 0, 0, 0.61, 0, 0, 0, 0,	false
63, 67	0, 0, 0, 0.67, 0, 0, 0, 0,	0, 0, 0, 0, 0, 0, 0, 0,	procName
…	…	…	…
81, 83	0, 0, 0, 0, 0, 0.55, 0, 0,	0.89, 0, 0, 0, 0, 0, 0, 0,	false

The same vector representation was used in the extension of stacking for classification tasks proposed in [11]. The difference is that class (or field) probability distributions are not typically produced by IE systems. Therefore, except for the places in the vectors that correspond to the predicted fields, all other values are set to zero.

3.5 Voting and multistrategy learning

Voting does not involve an internal cross-validation process and thus it is much faster than stacking. Given a page at runtime, a stacked template is again formed by the individual ones. For each entry in the stacked template, i.e. for each $t(s,e)$, the predicted fields are counted and the one with the highest count is selected. This is *majority voting* and in case of a tie, a random selection is performed among the winning fields.

Multistrategy learning was used in [9] in the form of voting using probability estimates, mapped from confidence scores. Since in stacking with confidence scores we employ a similar mapping process, as mentioned in section 3.4, we can use the same approach: each predicted field by some IE system for $t(s,e)$ is followed by a confidence score which is mapped to a probability estimate. The field with the highest estimate is finally selected and compared against the field predicted by stacking.

4 EXPERIMENTS

The aims of the experiments are to a) determine if stacking provides added value over the base-level IE systems, b) compare the simple approach to stacking (with fields) against stacking with confidence scores, c) compare stacking against majority voting and multistrategy learning d) compare different classifiers at meta-level.

4.1 Algorithms

At base-level we experimented with three well-known learning approaches for IE: the $(LP)^2$ system [6], a sequential covering rule-based learning approach, Hidden Markov Models (HMMs) [7], a stochastic finite-state approach for IE, and Boosted Wrapper Induction (BWI) [10]. At meta-level we experimented with six different classifiers, as implemented in the WEKA environment [15]: *J48*, an implementation of the C4.5 decision-tree inducer, *NaiveBayes*, the well known Naïve Bayes classifier, *IB1*, the 1-nearest-neighbour, *SMO*, an implementation of Support Vector Machines, *MLR*, a multi-response linear regression implementation, a setting commonly used in stacking for classification tasks [11, 12, 13], and finally the *LogitBoost* boosting algorithm using decision stumps as weak classifiers.

4.2 Domains

Experiments were conducted using three collections of Web pages describing three different domains. The first two collections consist of 101 pages describing CS courses and 96 pages describing research projects, and were constructed in the context of the WebKB project [16]. They were hand-filled for three and two extraction fields respectively: *crsNumber*, the number of the course, *crsTitle*, the course title, *crsInst*, the course instructor, *projTitle*, the project title and *projMember*, the name of a project member.

The third collection consists of 50 pages, describing laptop products that were collected from 25 vendor sites[1]. A total of 19 extraction fields were hand-filled for this domain, including the manufacturer of the laptop, the model name, the processor name, processor speed, ram, hard disk capacity, etc. The particular dataset was constructed in the context of building a shopping comparison agent that visits various vendor sites, extracts laptop descriptions and presents the results to the user.

As baseline for evaluating the proposed stacking framework we used the best results obtained by the three base-level IE systems in each dataset. Comparisons were also conducted against the best results obtained by voting and multistrategy learning as described in section 3.5. Finally, for the two WebKB datasets our results were compared against multistrategy learning results, as presented in [9].

4.3 Evaluation methodology and metrics

For the evaluation, cross-validation was used to obtain an unbiased estimate of performance on unseen data. For the laptop products domain, the corpus of 50 pages was randomly split into five equally populated parts. At each fold, a different part consisting of ten pages was kept for evaluation and the systems were trained on the remaining forty pages to induce the base-level IE systems and the meta-level classifier. Results on the unseen parts were averaged over all folds. Note that the cross-validation procedure used for evaluation is completely different from the cross-validation used for training, as presented in Table 6.

A different evaluation methodology was followed for the two WebKB domains, in order to achieve an objective comparison with the results reported for those domains in [9]. Each corpus was randomly split into two parts of equal size. The first part was used to induce the base-level IE systems and the meta-level classifiers. The second part was kept for evaluation. The process was repeated five times, averaging the results at the end.

As a performance measure we use the $F1$ evaluation metric, which is the harmonic mean of *recall* (R) and *precision* (P), defined as $F1 = 2RP/(R+P)$. Precision is the percentage of the predicted field instances that are correct, while recall is the

[1] Dataset is available in *http://www.iit.demokritos.gr/skel/crossmarc*

percentage of the annotated field instances (in the hand-filled templates) that were predicted by the system.

4.4 Results

Table 8 presents the best $F1$ scores of the base-level IE systems for each domain, compared to the best results of majority voting-multistrategy learning and the best meta-level classifiers, using the simple approach (with fields) and the confidence-score approach.

Table 8. Best $F1$ scores (%) of base-level IE systems, multistrategy setting and best meta-level classifiers, for each of the three domains.

	Base-level IE system	Majority Voting-Multistrategy	Stacking with fields	Stacking with conf. scores
CS courses	65,73	66,12	66,03	**71,93**
Projects	61,64	63,53	66,05	**71,41**
Laptops	63,81	64,38	68,46	**71,55**

For the CS courses domain $(LP)^2$ obtained the best results at base-level, while the HMMs obtain the best results for the other two domains. Table 8, shows a clear improvement in performance when using stacking with confidence scores against simple stacking, majority voting-multistrategy learning and the best base-level IE systems for all domains. Results in the third column are the best of multistrategy learning, which are higher than the best results of majority voting.

Table 9 shows the best $F1$ scores of all meta-level classifiers in the stacking with confidence scores approach over all three domains. The values in bold are the best $F1$ scores obtained for each domain. On average, the *LogitBoost* and *J48* classifiers obtained the best results for all domains, with the former being slightly better. The *LogitBoost* classifier performed best for the CS courses, and the laptop products domain, while *J48* performed best for the domain of research projects.

Table 9. Best $F1$ scores (%) of the meta-level classifiers over all domains.

	CS courses	Projects	Laptops	*Average*
J48	70,24	**71,41**	70,31	70,68
NaiveBayes	65,16	66,53	61,33	64,34
IB1	70,87	66,58	69,15	68,87
SMO	68,24	66,36	69,43	68,01
LogitBoost	**71,93**	70,67	**71,55**	**71,38**
MLR	70,50	65,19	69,72	68,47

Table 10 shows the best $F1$ scores per-field for the two WebKB datasets, in order to compare against the results presented in [9].

Table 10. Per-field best $F1$ scores (%) for the two WebKB datasets.

	Best Base	Multi-strategy	Stacking with confidence scores	Best Base [9]	Multi-strategy [9]
crsNumber	**94,46**	94,46	93,85	89,9	88,9
crsTitle	70,05	71,68	**74,26**	55,9	62,0
crsInst	48,21	48,98	**58,53**	48,1	49,8
projMember	65,00	66,38	**73,83**	41,1	45,5
projTitle	39,66	34,96	**40,15**	33,7	34,1

Experiments confirm the superiority of stacking with confidence scores, on four out of five fields. Stacking results are also better than the multistrategy learning results presented in [9]. This seems to be partially due to the higher performance of the base-level extractors that we used.

Note that the $F1$ scores in Tables 8 and 9 are based on precision and recall averaged over all instances of all fields. This allows an objective overall comparison among different IE systems. Experiments in stacking pairs of base-level IE systems were also conducted but did not lead to better results.

5 CONCLUSIONS AND FUTURE WORK

This paper presented a new framework for stacked generalization, appropriate for IE tasks and demonstrated its effectiveness. Experimental results have shown the superiority of the approach against single IE systems and combination of IE systems through voting and multistrategy learning.

Experiments will be continued using more datasets as well as other algorithms both at base-level and at meta-level. A more comprehensive use of the confidence scores generated by the individual IE systems, other than the one described in [9], will also be investigated, expecting to improve the new stacking framework. Longer-term plans include the application of stacking to handle harder forms of template-filling tasks.

6 ACKNOWLEDGEMENTS

The authors are grateful to Fabio Ciravegna for offering $(LP)^2$ and to Dayne Freitag for offering the two annotated WebKB datasets.

7 REFERENCES

[1] Wolpert, D., Stacked Generalization, *Neural Networks*,5(2): 241-260, 1992.

[2] Freund, Y., Shapire, R., Experiments with a new boosting algorithm. *In Proceedings of the 13th ICML*, 148-156, 1996.

[3] Breiman, L., Bagging Predictors, *Machine Learning*, 24(2): 123-140, 1996.

[4] Thompson, C.A.,, Califf, M.E., Mooney, R.J., Active Learning for Natural Language Parsing and Information Extraction, *In Proceedings of the 16th ICML*, Bled, Slovenia, 1999.

[5] Califf, M.E., Mooney R.J., Bottom-up Relational Learning of Pattern Matching Rules for Information Extraction, *JMLR*, (4), 177-210, 2003.

[6] Ciravegna, F., Adaptive Information Extraction from Text by Rule Induction and Generalization. *In Proc. of 17th IJCAI*, Seattle, 2001.

[7] Freitag, D., McCallum, A., Information Extraction using HMMs and schrinkage. AAAI-99 Workshop on machine learning for IE, 1999.

[8] Sonderland, S., Learning Information Extraction Rules for Semi-structured and Free Text, *Machine Learning*, 34-(1/3), 233-272, 1999.

[9] Freitag, D., Machine Learning for Information Extraction in Informal Domains, *Machine Learning*, 39, 169-202, 2000.

[10] Freitag, D., Kushmerick, N., Boosted Wrapper Induction, *In Proceedings of the 17th AAAI*, 59-66, 1999.

[11] Ting, K., Witten M., Issues in stacked generalization, *Journal of Artificial Intelligence Research*, 10, 271-289, 1999.

[12] Džeroski, S., Ženko, B., Is Combining Classifiers Better than Selecting the Best One? *Machine Learning*, 54(3): 255-273, 2004.

[13] Seewald, A., Towards understanding stacking, *PhD Thesis*, Dept. of Informatics, Technical University of Wien, Austria, 2003.

[14] Halteren, H., Zavrel, J., Daelemans, W., Improving accuracy in word class tagging through combination of machine learning systems, *Computational Linguistics*, 27 (2), 199-230, 2001.

[15] Witten, I., Frank, E., Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, *Morgan Kaufmann*, 2000.

[16] Craven, M., DiPasquo, D., Freitag, D., McCallum, A.K., Mitchell, T., Nigam, K., Slattery, S., Learning to extract symbolic knowledge from the World Wide Web. *In Proceedings of AAAI*, 1998.