Adaptive Information Extraction at UCD

Aidan Finn, Brian McLernon and Nicholas Kushmerick
Computer Science Department, University College Dublin
{*aidan.finn, brian.mclernon, nick*}*@ucd.ie*

Introduction

Information extraction (IE) is the process of identifying a set of pre-defined relevant items in text documents. In this paper we describe two systems for adaptive information extraction that are currently under development at UCD.

The first system, ELIE (Finn & Kushmerick 2004), uses an SVM based two-level approach to learning. Our ELIE algorithm first tries to classify every document position as the start of a fragment to be extracted, the end of a fragment, or neither. Typically, these extracted positions have high precision but mediocre recall. To increase recall, positions nearby these extracted positions are labelled by a second set of classifiers that are biased for high recall. For example, several positions "downstream" from each extracted start position are classified in order to find the end of the given fragment.

ELIE is fully supervised. The second system, TPLEX, employs a semi-supervised learning algorithm to generate extraction patterns from a mixture of labelled and unlabelled data. TPLEX bootstraps the learning process from a set of seed examples. The examples are used to populate initial pattern sets for each target category, with patterns matching the start and end positions of the seeds. Each pattern is then generalised to produce more patterns, which are inturn applied to the corpus. The algorithm then employs a recursive scoring metric: the goodness of a pattern is a function of the goodness of the fragments that it extracts, and the goodness of a fragment is a function of the goodness of the patterns that extract it.

ELIE

We treat tasks with multiple fields as multiple independent single-field extraction tasks i.e. we only extract one field at a time. We treat the identification of field start and end positions as distinct token classification tasks. All tokens that begin a labeled field are positive instances for the start classifier, while all the other tokens become negative instances for this classifier. Similarly, the positive examples for the end classifier are the last tokens of each labeled field, and the other instances are negative examples.

Features and encoding.

Each instance has a set of features that describe the given token. In addition to the actual token itself we encode several additional pieces of information about the token. Each token is tagged with its corresponding POS using Brill's POS tagger. We also represent chunking information about the tokens by grouping the POS tags into noun-phrases and verb-phrases. We add the values associated with the token in a gazetteer. The gazetteer is a user-defined dictionary that contains lists of first-names and last-names taken from the U.S. census bureau, a list of countries and cities, time identifiers (am, pm), titles (Jr., Mr), and a list of location identifiers used by the U.S. postal service (street, boulevard). Orthographic features include whether the token is upper-case, lower-case, capitalized, alphabetic, numeric or punctuation.

Relational information is encoded using additional features. To represent an instance, we encode all these features for that particular token. In addition, for a fixed window size of w, we add the same features for the previous w tokens and the next w tokens. For example, if we use a window size of 1, then each instance has a feature to represent the token for that instance, the token of the preceding instance and the token of the next instance. Similarly, there are features to represent the POS, gaz and orthographic information of the current instance and the previous and next instances.

Learning with ELIE

The ELIE[1] algorithm has two distinct phases. In the first phase, ELIE simply learns to detect the start and end of fragments to be extracted. Our experiments demonstrate that this first phase generally has high precision but low recall. The second phase is designed to increase recall. We find that very often false negatives are "almost" extracted (the start but not the end is correctly identified, or the end but not the start). In the second phase ELIE is trained to detect either the end of a fragment given its beginning, or the beginning of a fragment given its end.

Level One (L1) learning. The L1 learner treats IE as a standard classification task, augmented with a simple mechanism to attach predicted start and end tags.

Firstly the set of training examples are converted to a set of instances for the start and end using the features described above. Each token in each training document becomes a single instance, and is either a positive or negative example of a start or end tag. Each of these instances is encoded with

[1] ELIE is available at http://smi.ucd.ie/aidan

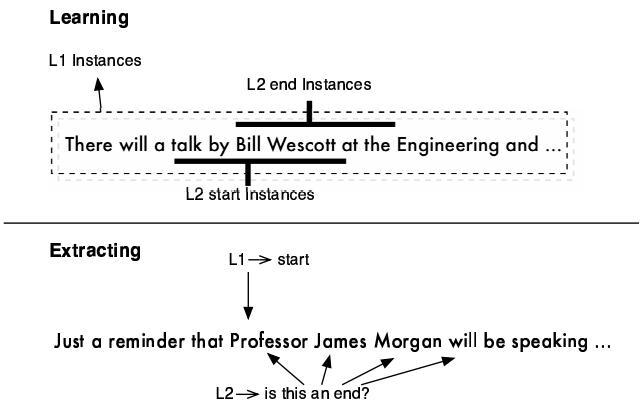

Figure 1: ELIE: L1 and L2

feature information for the particular token in question and the tokens surrounding it. Then the attributes are filtered according to information gain. These instances are passed to a learning algorithm which uses them to learn a model. At the end of the L1 training phase we have models for start and end tags and all the start-end pairs.

The start-end pairs are passed to a tag-matcher component which is charged with matching start and end tags. Our experiments involve a tag-matcher which derives a histogram based on the number of tokens between each start and end tag in the training data. When matching predictions, the probability of a start-tag being paired with an end-tag is estimated as the proportion with which a field of that length occurred in the training data.

Level two (L2) learning. The L1 learner builds its model based on a very large number of negative instances and a small number of positive instances. Therefore the prior probability that an arbitrary instance is a boundary is very small. This very low prior probability means that the L1 model is much more likely to produce false negatives than false positives.

The L2 learner is learned from training data in which the prior probability that a given instance is a boundary is much higher than for the L1 learner and the number of irrelevant instances is vastly reduced. This "focused" training data is constructed as follows. When building the L2 start model, we take only the instances that occur a fixed distance before an end tag. Similarly, for the L2 end model, we use only instances that occur a fixed distance after a start tag.

Fig. 1 shows an example of the the instances used by L1 and L2 with a lookahead/lookback of 3. In this example the token "Bill" is the start of a field and the token "Wescott" is the end of a field. When building the L1 classifiers we use all the available instances. When building the L2 start model we use the end token and the 3 tokens preceding it. When building the end model we use the start token and the three tokens following it. Note that these L2 instances are encoded in the same way as for L1; the difference is simply that the L2 learner is only allowed to look at a small subset of the available training data. When extracting, the L2 end classifier is only applied to the three tokens following the token which L1 tagged as a start and the token itself. Similarly the L2 start classifier is only applied to instances tagged as an end by L1 and the three preceding tokens.

This technique for selecting training data means that the L2 models are likely to have much higher recall but lower precision than L1 models. If we were to blindly apply the L2 model to the entire document, it would generate a lot of false positives. Therefore, as shown in Fig. 1, the reason we can use the L2 model to improve performance is that we only apply it to regions of documents where the L1 model has made a prediction. Specifically, during extraction, the L2 classifiers use the predictions of the L1 models to identify parts of the document that are predicted to contain fields.

The two level approach takes advantage of the fact that at L1 we have two highly dependent learners, each with very high precision. Thus a prediction by one of them indicates with very high probability that the second should make a prediction. When training the L2 classifier, we drastically alter the prior probabilities of the training data by using only the instances within a fixed distance before or after an annotated start or end. This L2 classifier is much more likely to make predictions as it was trained on a much smaller set of negative instances. Thus it is more likely to identify starts or ends that the L1 classifier missed.

Experiments

We evaluate our algorithm on three standard benchmark datasets - the seminar announcements ("SA") dataset , the job postings ("Jobs") dataset, and the Reuters corporate acquisitions ("Reuters") - consisting of 31 fields in total.

We used a 50:50 split of the dataset repeated 10 times. All experiments use a window of length 3 tokens, and L2 lookahead/lookback of 10 tokens. On the SA dataset, this typically gives a set of approximately 80 thousand training instances (a few hundred of which are positive) and approximately 50 thousand attributes. These experiments have all features enabled initially, and then the top 5000 features ranked by information gain are used for learning the model.

We compare our system against BWI, RAPIER, LP^2 and SNOW-IE using the available published results for each system. Figure 2 shows precision and recall for $Elie_{L2}$compared to that of other IE systems on the 31 fields. The horizontal axis shows the performance of our system at L2 while the vertical axis shown the performance of competitors for each field. In general precision is comparable to competitor systems while recall is superior.

TPLEX

Like most adaptive information extraction algorithms, ELIE is fully supervised. In contrast, our TPLEX algorithm is semi-supervised: it uses patterns mined from labelled documents to find possible fragments in unlabelled documents, with which it can confirm the quality of the initial patterns, as well as discover additional patterns. This process iterates until convergence.

TPLEX is based on the NOMEN algorithm (Yangarber, Lin, & Grishman 2002). NOMEN is a semisupervised information extraction algorithm, but it has a very simple it-

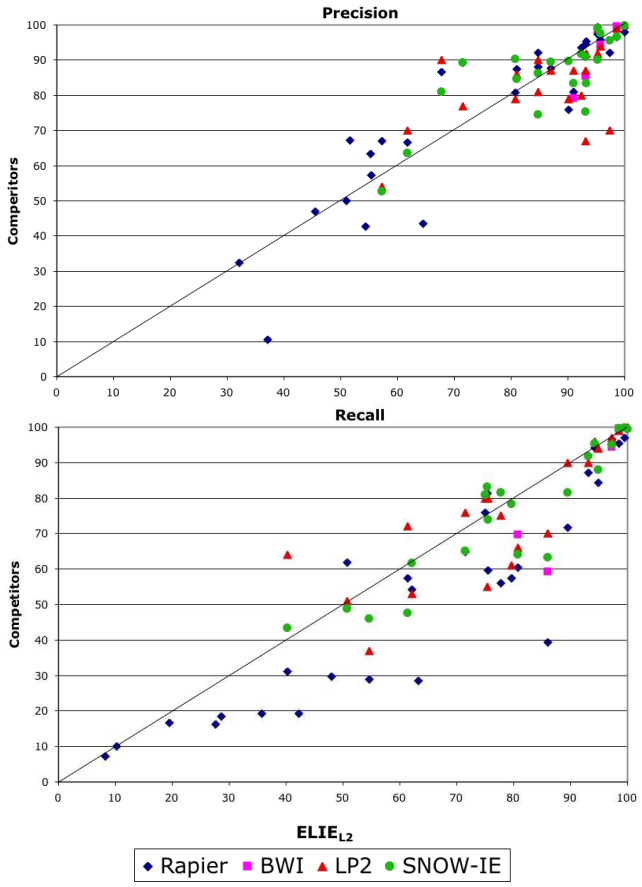

Figure 2: Comparison of $Elie_{L2}$with other IE systems on three benchmark IE tasks

erative structure: at each step, a very small number of high-quality new fragments are extracted, which are treated in the next step as equivalent to seeds from the labeled documents. NOMEN has a number of parameters which must be carefully tuned to ensure that it does not over-generalise.

We have extended NOMEN by allowing it to make finer-grained (as opposed to binary) scoring decisions at each iteration. NOMEN's binary scoring mechanism works well in dense corpora with substantial redundancy (ie, many occurrences of the same fragment). But many information extraction tasks feature sparse corpora with little or no redundancy. Instead of definitively assigning a fragment to a given field, we calculate an estimate of the probability that it belongs to the field.

Unlike ELIE, TPLEX is a true multi-field extraction algorithm in that it extracts multiple fields simultaneously. By doing this, information learned for one field can be used to enforce information learned about another field. For example, if a fragment has a high probability for one field, then our method automatically ensures that is has a low probability for all other fields.

TPLEX employs a recursive definition of "goodness" in which good patterns reinforce good fragments, and good fragments reinforce good patterns. Specifically, the goodness of a pattern is a function of the goodness of the fragments that it extracts, and the goodness of a fragment is a function of the goodness of the patterns that extract it.

Learning with TPLEX

As with ELIE, TPLEX extracts fragments by identifying probable fragment start and end positions, which are then assembled into complete fragments. TPLEX's patterns are therefore *boundary detectors* which identify one end of the field or the other. TPLEX's boundary detectors are similar to those learned by BWI (Freitag & Kushmerick 2000). A boundary detector d has two parts, a left pattern ℓ and a right pattern r; $d = (\ell, r)$. Each of these patterns is a sequence of tokens, where each token is either a literal or a wildcard. For example, the boundary detector ([`will be <Punc>`][`<Upper> <FName> <Any>`]) would correctly find the start of a name in an utterance such as "will be: Dr Robert Boyle" and "will be, Robert Frederick (IBM)", but it will fail to identify the start of the name in "will be Dr. Robert Boyle". The boundary detectors that find the beginnings of fields are called the pre-patterns, and the detectors that find the ends of fields are called the post-patterns.

As input, TPLEX requires a set of tagged seed documents for training, and an untagged corpus for learning. The seed documents are used to initialize the pre-pattern and post-pattern sets for each of the target fields. We then grow the initial sets of patterns in each field by generalizing each pattern with special tokens such as the wildcard token, punctuation token, number token and capitalized token. All patterns are then applied to the entire corpus, including the seed documents. When a pattern matches a new position, the tokens at that position are converted into a maximally-specilaized pattern, which is added to the pattern set. New patterns are in turn generalized and the whole process iterates until no new patterns or fragments are discovered.

The locations in the corpus where the patterns match are regarded as potential target positions. Pre-patterns indicate potential start positions for target fragments while post-patterns indicate end positions. When all of the patterns have been matched against the corpus, each field will have a corresponding set of potential start and end positions.

Positions are denoted by r, patterns are denoted by p, and fields are denoted by f. The special field index X indicates "should not be extracted at all". Let F be the set of all fields, including X.

The labelled training data consists of a set of positions $R = \{\ldots, r, \ldots\}$, and a labelling function $T : R \rightarrow F$ for each such position. $T(r) = f$ indicates that position r is labelled with field f in the training data. $T(r) = X$ means that r is not labelled in the training data (ie, r is a negative example for all fields).

The unlabelled test data consists of an additional set of positions U. The learning task is to extend the domain of T to U—ie, to generalize from $T(r)$ for $r \in R$, to $T(r)$ for $r \in U$.

Pattern and Position Scoring

When the patterns and positions for the fields have been identified we must score them. As previously stated $\text{score}_f(r)$ can be interpreted as the probability that r's field is f. Note that $\text{score}_X(r)$ is perfectly well-defined: this is simply the probability that r shouldn't be extracted. As it is a probability, $\text{score}_f(r)$ is a distribution over the fields (ie,$\sum_{f \in F} \text{score}_f(r) = 1$).

$\text{score}_f(p)$ is the probability that p extracts positions whose field is f, with $\text{score}_X(p)$ being the probability that p extracts "junk". Again, $\text{score}_f(p)$ is a distribution over the fields (ie, $\sum_{f \in F} \text{score}_f(p) = 1$).

Below we will describe in detail the recursive manner in which we defined $\text{score}_f(r)$ in terms of $\text{score}_f(p)$, and vice versa. Given that definition, we want to find fixed-point values for $\text{score}_f(p)$ and $\text{score}_f(r)$. To achieve this, we initialize the scores, and then iterate through the scoring process (ie, calculate scores at step $t+1$ from scores at step t). This process repeats until convergence.

Initialization. As the scores of the patterns and positions of a field are recursively dependant, we must assign initial scores to one or the other. Initially the only elements that we can classify with certainty are the seed fragments. So we initialise the scoring function by assigning scores to the positions for each of the fields. In this way it is then possible to score the patterns based on these initial scores.

From the labelled training data, we derive the prior probability $\pi(f)$ that a randomly selected position belongs to field $f \in F$:

$$\pi(f) = \frac{|\{r \in R \,|\, T(r) = f\}|}{|R|}$$

Note that $\pi(X) = 1 - \sum_{f \neq X} \pi(f)$ is simply the prior probability that a randomly selected position should not be extracted at all.

Given the priors $\pi(f)$, we can score each potential position r in field f:

$$\text{score}_f^0(r) = \begin{cases} \pi(f) & \text{if } r \in U \\ 1 & \text{if } r \in R \wedge T(r) = f \\ 0 & \text{if } r \in R \wedge T(r) \neq f \end{cases}$$

As required, $\text{score}_f^0(r)$ is a distribution: if $r \in U$ then $\text{score}_f^0(r)$ is a distribution because $\pi(f)$ is; if $r \in R$ then $\text{score}_f^0(r)$ is a distribution because it is 1 for $f = T(r)$ and 0 for all other f.

Iteration After initializing the scores of the positions we can begin the iterative process of scoring the patterns and the positions. To compute the score of a pattern p for field f we compute a positive score, $\text{pos}_f(p)$; a negative score, $\text{neg}_f(p)$; and an unknown score, $\text{unk}f(p)$. $\text{pos}_f(p)$ can be thought of as a measure of the benefit of p to f, while $\text{neg}_f(p)$ measures the harm of p to f, and $\text{unk}f(p)$ is the uncertainty of p to f.

These quantities are defined as follows: $\text{pos}_f(p)$ is the average score for field f of positions extracted by p:

$$\text{pos}_f(p) = \frac{1}{Z_p} \sum_{p \to r} \text{score}_f^t(r),$$

where Z_p is normalization constant to ensure that $\text{pos}_f(p)$ is a distribution. For each field f and pattern p, $\text{neg}_f(p)$ is the extent to which p extracts positions whose field is not f:

$$\text{neg}_f(p) = \frac{1}{Z_p} \sum_{f' \neq f} \text{pos}_{f'}(p).$$

Finally, $\text{unk}(p)$ measures the degree to which p extract positions who field is unknown:

$$\text{unk}(p) = \frac{1}{|\{p \to r\}|} \sum_{p \to r} \text{unk}(r),$$

where $\text{unk}(r)$ measures the degree to which position r is unknown. To be completely ignorant of a position's field is to fall back on the prior field probabilities $\pi(f)$. Therefore, we calculate $\text{unk}(r)$ by comparing $\text{score}_f^t(r)$ and prior $\pi(f)$:

$$\text{unk}(r) = 1 - \frac{1}{Z_r} \sum_f (\text{score}_f^t(r) - \pi(f))^2$$

where Z_r is a normalization constant to ensure that $\text{unk}(r) > 0$; in our implementation we use $Z_r = \max_r \sum_f (\text{score}_f^t(r) - \pi(f))^2$.

For each field f and pattern p, $\text{score}_f^t(p)$ is defined in terms of $\text{pos}_f(p)$, $\text{neg}_f(p)$ and $\text{unk}(p)$ as follows:

$$\text{score}_f^{t+1}(p) = \frac{1}{Z_p} \cdot \frac{A}{A+B+C} \cdot \text{pos}_f(p),$$

where $A = \alpha_{\text{pos}} \text{pos}_f(p)$, $B = \alpha_{\text{neg}} \text{neg}_f(p)$, and $C = \alpha_{\text{unk}} \text{unk}(p)$. α_{pos}, α_{neg} and α_{unk} are weights that determine the relative importance of the positive, negative and unknown perspectives on p's quality; in out implementation we set $\alpha_{\text{pos}} = \alpha_{\text{neg}} = \alpha_{\text{unk}} \text{unk}(p) = 1/3$.

Finally, we complete the iterative step by calculating a revised score for each non-seed position from these pattern scores:

$$\text{score}_f^{t+1}(r) = \begin{cases} \text{score}_f^t(r) & \text{if } r \in R \\ \frac{1}{Z_r} \cdot \sum_{p \to r} \text{score}_f^t(p) & \text{if } r \in U. \end{cases}$$

Fragment Identification

When the positions scores have converged, we identify complete fragments within the corpus by matching pre-positions with post-positions. To do this we compute the length probabilities for the fragments of field f based on the lengths of the seed fragments of f. Suppose that positions r_1 has been identified as a possible start for field f, and position r_2 has been identified as a possible field f end, and let $P_f(\ell)$ be the fraction of field f seed fragments with length ℓ. Then the fragment (r_1, r_2) is assigned a score of:

$$\text{score}_f(r_1) \cdot \text{score}_f(r_2) \cdot P_f(r_2 - r_1).$$

One problem with this approach is that many overlapping fragments are extracted (ie, have non-zero score). Of course,

we know that if two fragments overlap, at least one must be wrong. We resolve overlapping fragments by calculating the set of non-overlapping fragments that maximises the total score while also accounting for the expected rate of occurrence of fragments from each field in a given document. It is computationally infeasible to find the optimal subset, so we use a greedy search.

At present TPLEX is still in development. Results will be presented at a later date.

References

Finn, A., and Kushmerick, N. 2004. Multi-level boundary classification for information extraction. In *European Conference on Machine Learning*.

Freitag, D., and Kushmerick, N. 2000. Boosted wrapper induction. In *AAAI/IAAI*.

Yangarber, R.; Lin, W.; and Grishman, R. 2002. Unsupervised learning of generalised names. In *19th International Conference on Computational Linguistics*.