Part III

Algorithms and Models

10 Modeling Information Extraction as a Classification Task

10.1 Idea and Concept

One of the core statistical techniques is *classification*. A *classifier* operates over a set of classes (class labels) C; classifiers are *trained* from a list of tuples (F, c), where each F is a feature vector representing an instance of class $c \in C$. For application, a classifier must be able to map any (known or unknown) feature vector F to a class $c \in C$ which is the most likely class for this feature vector (as estimated by the classifier).

We can model information extraction as a classification task if we break down the task of extracting information from a text into a series of decisions that a classifier can handle; and the task of training an information extractor from an annotated training text into a series of operations for training the classifier.

For understanding how to do this, we regard a text as a sequence of *tokens*. For the purposes of information extraction, each token might be part of an attribute value; two or more sequential tokens might be part of the same attribute value. (This means we assume that no overlapping or non-continuous attribute values occur, as already stated in Chap. 8). This can be expressed by a sequence of *states* which is as long as the sequence of tokens. Each *state* is a 3-tuple (*attribute*, *begin?*, *end?*) that describes whether the corresponding *token* is part of a attribute value and whether the value begins and/or ends with this token (*attribute* is set to a *null* value iff the token is not part of an attribute value; *begin?* and *end?* are boolean values).

Modeled this way, the goal of IE is to determine the most likely state sequence for a given token sequence (text); IE systems are trained from token sequences where the corresponding state sequence is already known and can be used for training (annotated training texts).

Now we can model IE as sequence of classification tasks where the goal of each classification task is to determine the most likely *state* for a *token*. But classifiers work on feature vectors F and class labels c. Hence, for being able to use a classifier for this purpose, we need a way to convert states into class labels (for training the classifier) and vice versa (for applying the classifier and interpreting the results), and to convert tokens into feature vectors.

Converting states into class labels and vice versa is not difficult, but there are different ways to do it. While these different ways are generally equivalent in expressive power (as long as they can express all legal state sequences), they will result in different label sequences corresponding to the same state sequence. Since the classifier needs to learn and predict *label* sequences without having any information about the underlying state sequence, differences in the generated label sequences might affect the accuracy

of classifier predictions and hence the extraction quality reached by the resulting IE system. We refer to the different ways of converting between states and labels as *tagging strategies* (or labeling strategies); they will be covered in the next section.

Conversion of tokens to feature vectors for the classifier is another issue that needs to be handled. Obviously, a very trivial way of doing this would be to convert each token into single-element feature vector that just contains the token itself as only element. But this would provide almost no useful information to the classifier, since no information about the textual context surrounding the token would be available for classification, and neither would any linguistic, semantic or morphological information about the token itself be available. Clearly, classifiers will have an advantage if they can work on richer feature vectors that do provide at least some additional information about a token and the context in which it occurs. We refer to such feature vectors as *context representations* and will cover them in detail in Chapter 12, after covering the third essential ingredient, which is the used classifier (classification algorithm) itself.

To resume, approaches modeling information extraction as a classification task require three components:

1. A *tagging strategy* that is used for translating the sequence of states describing the attribute values to extract into a series of token labels, and vice versa. The relevant tagging strategies will be presented below (Sec. 10.2).
2. A *classification algorithm* (classifier) that predicts a label for each token during the application phase. The classifier must be trainable so it can learn the token labels in the training phase. The algorithm we use by default will be introduced in Chap. 11.
3. A feature extractor that converts the context of each token into a feature vector for the classification algorithm. The *context representations* we provide by default will be explained and motivated in Chap. 12.

In our approach, each of these components is independent from the others and from the rest of the system. Each component can be modified or replaced, leaving the rest of the setup unchanged.

Other systems pursuing similar approaches have been discussed in Sec. 4.4. These previous classification-based IE approach have combined a specific tagging strategy with a specific classification algorithm and specific other parameter settings, making it hard to detect how each of these choices influences the results. To allow systematic research into each of these choices, we have designed our system in such a way that it allows utilizing any tagging strategy with any classification algorithm (provided that a suitable implementation or adapter exists). This makes it possible to compare strategies or algorithms in an identical setting.

10.2 Tagging Strategies

Tagging strategies (or labeling strategies) are necessary for translating between sequence of states s_i describing the attribute values to extract and sequences of class

labels c_i for a classifier to handle. Each strategy need to define a translation operation $c_i \leftarrow label(s_i)$ which converts a state into a label and a translation operation $s_i \leftarrow state(c_i)$ which converts a state into a label. The former operation is needed for training the classifier from an annotated text (where states are known), the latter is used for extracting information from an un-annotated text by invoking the classifier for each token and converting the resulting label sequences into state sequences.

We assume that tagging strategies are stateful, so the result of a translation operation from or to a state s_i might depend on the preceding state s_{i-1}. It does not matter if tagging strategies are unable to determine the value of the *end?* field of a state s_i correctly since this value can be be reconstructed by comparing s_i and the following state s_{i+1}.

The most trivial (*Triv*) strategy would be to use a single class for each of the attributes and an additional class "O" class for all *other* tokens. However, this does not work correctly if two values of the same attribute immediately follow each other, e.g., if the names of two SPEAKERs are separated by a linebreak only. In such a case, both names would be collapsed into a single attribute value, since the trivial strategy lacks a way to mark the begin of the second attribute value, meaning that its *state* translation operation is unable to correctly determine the value of the *begin?* field in such cases.

For this reason (as well as for improved classification accuracy), various more complex strategies are employed that use distinct classes to mark the first and/or last token of an attribute value. The two variations of *IOB* tagging are probably most common: the variant usually called *IOB2* classifies each token as the begin of a value of a certain attribute (B-*type*, where *type* is the name of the attribute), as a continuation of the previously started attribute value, if any (I-*type*), or as not belonging to any attribute value (O)[1]. The *IOB1* strategy differs from *IOB2* in using B-*type* only if necessary to avoid ambiguity (i.e., if two values of the same attribute immediately follow each other); otherwise I-*type* is used even at the beginning of attribute values. While the *Triv* strategy uses only $n+1$ classes for n attributes, *IOB* tagging requires $2n+1$ classes.

BIE tagging differs from *IOB* in using an additional class for the last token of each attribute value. One class is used for the first token of an attribute value (B-*type*), one for inner tokens (I-*type*) and another one for the last token (E-*type*). A fourth class BE-*type* is used to mark attribute values consisting of a single token (which is thus both begin and end). Thus *BIE* requires $4n+1$ classes.

A disadvantage of the *BIE* strategy is the high number of classes it uses (twice as many as *IOB1|2*). This can be addressed by introducing a new strategy, *BIA* (or *Begin/After* tagging). Instead of using a separate class for the last token of an attribute value, *BIA* marks the first token *after* an attribute value as A-*type* (unless it is the begin of a new attribute value). Begin (B-*type*) and continuation (I-*type*) of attribute values are marked in the same way as by *IOB2*. *BIA* requires $3n+1$ classes, n less than *BIE* since no special treatment of single-token attribute values is necessary.

[1] Note that the actual names used to identify classes do not matter and can deviate from those used in the explanation; what matters is the chosen partitioning of tokens into classes.

Strategy	Triv	IOB2	IOB1	BIE	BIA	BE
Special class for first token	–	+	(+)[a]	+	+	+
Special class for last token	–	–	–	+	–	+
Special class for token after last	–	–	–	–	+	–
Number of classes	$n+1$	$2n+1$	$2n+1$	$4n+1$	$3n+1$	$2 \times (n+1)$
Number of classifiers	1	1	1	1	1	2

[a] Only if required for disambiguation

Table 10.1: Properties of Tagging Strategies

Text	Our	meeting	with	Mr.	Irfan	Ali
Triv	O	O	O	speaker	speaker	speaker
IOB2	O	O	O	B-speaker	I-speaker	I-speaker
IOB1	O	O	O	I-speaker	I-speaker	I-speaker
BIE	O	O	O	B-speaker	I-speaker	E-speaker
BIA	O	O	O	B-speaker	I-speaker	I-speaker
BE	O/O	O/O	O/O	B-speaker/O	O/O	O/E-speaker
Text	will	be	at	1:30	pm	in ...
Triv	O	O	O	stime	stime	O
IOB2	O	O	O	B-stime	I-stime	O
IOB1	O	O	O	I-stime	I-stime	O
BIE	O	O	O	B-stime	E-stime	O
BIA	A-speaker	O	O	B-stime	I-stime	A-stime
BE	O/O	O/O	O	B-stime/O	O/E-stime	O/O

Table 10.2: Labeling Example

The strategies discussed so far require only a single classification decision for each token (though often multiple binary classifiers are used concurrently instead of a single multi-class classifier to improve classification accuracy). Another option is to use two separate classifiers, one for finding the begin and another one for finding the end of attribute values. *Begin/End* (*BE*) tagging requires $n+1$ classes for each of the two classifiers (B-*type* + O for the first, E-*type* + O for the second). In this case, there is no distinction between inner and outer (other) tokens. Complete attribute values are found by combining the most suitable begin/end pairs of the same attribute, typically by taking the length distribution of attribute values into account.

Table 10.1 lists the properties of all strategies side by side. Table 10.2 shows the labels generated by each strategy for an example text fragment.

Which tagging strategies should be appropriate for which situations will be discussed in Chap. 19 after evaluating the various strategies.