A Comparison of Statistical and Rule-Induction Learners for Automatic Tagging of Time Expressions in English

Jordi Poveda, Mihai Surdeanu and Jordi Turmo
TALP Research Center
Technical University of Catalonia (UPC)
Barcelona, Spain
{jpoveda,surdeanu,turmo}@lsi.upc.edu

Abstract

Proper recognition and handling of temporal information contained in a text is key to understanding the flow of events depicted in the text and their accompanying circumstances. Consequently, time expression recognition and representation of the time information they convey in a suitable normalized form is an important task relevant to several problems in Natural Language Processing. In particular, such an analysis is largely significant for Information Extraction (IE), Question Answering (QA) and Automatic Summarization (AS). The most common approach to time expression recognition in the past has been the use of handmade extraction rules (grammars), which also served as the basis for normalization. Our aim is to explore the possibilities afforded by applying machine learning techniques to the recognition of time expressions. We focus on recognizing the appearances of time expressions in text (not normalization) and transform the problem into one of chunking, where the aim is to correctly assign Begin, Inside or Outside (BIO) tags to tokens. In this paper, we explain the knowledge representation used and compare the results obtained in our experiments with two different methods, one statistical (support vector machines) and one of rule induction (FOIL). Our empirical analysis shows that SVMs are superior.

1. Time Expression Recognition: An Overview

1.1. Problem Description

Chunking refers to any problem or task in NLP (natural language processing) which involves segmenting (i.e. identifying the boundaries of) the occurrences in a text of a certain type of entities. Chunking of *temporal expressions* is a part of the more complex task called *temporal expression recognition and normalization* (TERN), in which the aim is to identify the mentions in a text of expressions that denote time, and to capture their meaning by writing them in a canonical representation. A common way to represent these temporal expressions in text —as well as other entities of interest in information extraction, like named entities, events or relations— is to employ XML tags with convenient attributes to further qualify the entity (see Figure 1).

```
But even <TIMEX2 VAL="1999-07-22">
last Thursday </TIMEX2>, there were
signs of potential battles <TIMEX2
VAL="FUTURE_REF" ANCHOR_DIR="AFTER"
ANCHOR_VAL="1999-07-22"> ahead
</TIMEX2>.
```

Figure 1. XML annotation of time expressions using TIMEX2 tags

The TIMEX standard sets out guidelines for annotating and normalizing mentions of time expressions. These and a full description of temporal expressions can be found in [2].

There are two separate problems in the TERN task, one of identifying the *extension* of the mentions of temporal expressions in text (i.e. chunking) and the complementary problem of normalizing the value of the temporal expression. Here we are only concerned with the first of these two problems. Contrary to what it may at first seem, temporal expressions come into a wide variety of forms ([12]):

- Fully-specified time references: *16th June 2006*, *the twentieth century, Monday at 3pm.*
- Expressions whose reference depends on the context: *the previous month*, *three days after the meeting*, *February the following year*.
- Anaphoric expressions and expressions relative to the

time when the expression is written: *that day*, *yesterday*, *currently*, *then*.

- Durations or intervals: *a month*, *three days*, *some hours in the afternoon*.
- Frequencies or recurring times: *monthly*, *every other day, once a week*, *every first Sunday of a month*.
- Culturally dependent time denominations: *Easter*, *the month of Ramadan*, *St. Valentine*.
- Fuzzy or vaguely specified time references: *the future*, *some day*, *eventually*, *anytime you so desire*.

1.2. Relevant Literature

The TERN (Temporal Expression Recognition and Normalization) task has gathered attention only recently within the broader framework of Information Extraction (IE), and references to it in the research literature are still sparse. Temporal expression extraction has been most often approached as an intermediate step towards tackling the more difficult problems of event extraction and characterization, extraction of temporal relations (e.g. before, after, includes, ...), temporal ordering and timestamping of events, and temporal reasoning ([1], [6]).

Grammars continue to be the strategy most commonly resorted to for time analysis in documents (see [9] for an example of a grammatical treatment of recognition and co-reference resolution of time expressions). Boguraev & Ando (2005) [1] describe a hybrid approach to the extraction of time expressions, events and temporal relations, which combines grammatical parsing with machine learning. First, a cascade of finite-state grammars is used to target temporal expressions and to extract attributes for their normalization; separately, a combination of token, POS (part-of-speech), syntax and context, in addition to word profiling from unannotated corpora, are used as features in a risk-minimization classifier for event and temporal expression recognition.

Here we are rather interested in the more automatic treatment of chunking afforded by methods such as support vector machines (SVM) and inductive logic programming (ILP). Perrig (2005) [6] reports the results of various experiments using SVMs (Vapnik, 1995 [11]) for the recognition of time expressions, events, and temporal relations. The precision, recall and F-measure results are given for several configurations of features. Hacioglu et al. (2005) [3] have also applied SVMs for automatic chunking of time expressions in English and Chinese. Their approach involves combining proper token features (token, POS, syntactic, semantic and contextual) with external sources of knowledge (the output from third-party external classifiers used as additional features for the SVM classifier), some of them human-made rules, in order to boost the classification performance of the SVMs.

There is virtually no previous work that uses ILP for the recognition of temporal expressions. A relational approach for automatically learning rules for information extraction is described in Turmo & Rodriguez (2002) [10]. Their system EVIUS employs Inductive Logic Programming —in particular, FOIL— to automatically extract IE rules (for any type of entity, not just time expressions) in multiple domains, but they do not perform an explicit evaluation for temporal expression recognition. In this paper we introduce a consistent comparison of SVMs and ILP using the same corpus and the same feature sets.

The rest of this paper is organized as follows. Section 2 describes the scheme used to represent chunks and the set of token features we employ to tackle the time expression chunking problem with machine learning. Next, Section 3 introduces the two alternative approaches we have considered for this problem: support vector machines (statistical), and the FOIL ILP system (rule induction). How training and classification takes place is described for each of the two approaches. Section 4 is concerned with our experiments with SVMs, acting on several of the learning method parameters, and with FOIL. The results obtained with each method are discussed and compared in this section. Finally, Section 5 summarizes the conclusions.

2. Data Representation and Features

The chunking-related problems have traditionally been represented as the multiclass classification problem of assigning *BIO tags* to each token in a sequence of text (Figure 2). The tags stand for Begin (B), Inside (I) and Outside (O), and are enough for delimiting non-overlapping, non-recursive chunks (i.e. chunks that neither appear inside the boundaries of the extension of another larger chunk, nor overlap with another chunk). In this kind of data representation, the features used for classification refer to the individual tokens themselves.

```
But/O even/O last/B Thursday/I ,/O
there/O were/O signs/O of/O potential/O
battles/O ahead/B ./O
```

Figure 2. Chunk annotation of time expressions using BIO tags

An alternative *segment-based representation* involves considering sequences of consecutive tokens (i.e. segments), as opposed to individual tokens, and tagging each segment as either being a chunk of the desired type or not. In this type of representation, each segment of up to

a bounded length is assigned a distinct identifier and classification makes use of features that exploit relationships among the tokens in the segment (e.g. the order in which they appear), apart from features about the tokens themselves. We have chosen to use the token-level BIO representation because of three reasons: first, the complexity (in terms of volume of training data) involved in using BIO tags is much lower than for segments; second, the BIO representation has the potential to correctly tag chunks of any length, contrary to segments, for which an upper bound on the length must be fixed for tractability; and third, time expressions appearing inside of another larger time expression (which could be detected using a segment representation), although existing, are very infrequent.

Both training and classification involve a previous step of document preprocessing: tokenization and extraction of features for each token. The text is first tokenized (segmented into individual tokens) and then a number of features are computed for each token. Tokens correspond normally with individual words, but also include punctuation marks —which are separate tokens— and special constructs like the genitive "'s". The features used for token classification are:

- Lexical: The token form itself, the token in lowercase, the token that results from removing all letters from the token (e.g. 3 for "3pm"), the token that results from removing all letters and numbers (e.g. - - - for 1995-07-12).
- Morphological: The POS (Part Of Speech) tag (for instance, NN for noun, JJ for adjective, CD for cardinal number, MD for modal verb, ...).
- Syntactic: The tag of the basic syntactic chunk which the token belongs to (e.g. I-NP for inside noun phrase, B-VP for beginning of verb phrase, ...).
- Format features: These are flags that indicate if the token has certain format characteristics (*isAllCaps* like "THU", *isAllCapsOrDots* like "I.B.M", *isAllDigits* like "2004", *isAllDigitsOrDots* like "10.24", and *initialCap* like "February").
- Features that indicate if the token belongs to certain specific classes of words. These features are computed as a function indicating membership to a closed list of tokens, which defines the class: *isNumber* (e.g. *one*, *two*, *ten*, ...), *isMultiplier* (e.g. *hundred*, *thousands*, ...), *isDay* (e.g. *monday*, *mon*, *saturday*, *sat*, ...) and *isMonth* (e.g. *january*, *jan*, *june*, *jun.*, ...).
- Contextual features: All of the previous features, but in reference to the tokens occurring in a certain window to the left and right of the current token.
- The target classification: The BIO tags for a window of tokens in the left context (we call these dynamic features). We use the gold BIO tags in training and the predicted tags during testing.

The part-of-speech tags are computed using a linguistic processor, the TnT tagger[1]. The syntactic chunks are obtained using the YamCha toolkit, a general purpose tagger based on SVMs, trained with a model for English based on Penn TreeBank corpus. The rest of features are computed programmatically, and arranged into a tabular format.

3. Approaches

We have considered two different approaches to deal with the problem of time expression recognition, each corresponding to a different machine-learning paradigm. The first method considered is statistical and uses Support Vector Machines ([11]) as the underlying machine learning algorithm. SVMs are discriminative models that find the maximum-margin hyperplane that separates examples from two classes (positive and negative) and which, thanks to the use of kernel functions, allow for non-lineal separating surfaces. The second method belongs to the rule-induction paradigm, more specifically Inductive Logic Programming. We use FOIL ([8]), an ILP system, in order to learn classifiers to assign the B, I and O tags respectively. FOIL builds a hypothesis in the language of first-order logic predicates that attempts to describe the form of positive examples, based on heuristic search. The results of these two approaches are compared in Section 4.

3.1. Statistical Approach: the YamCha Tagger

For our experiments we have used the YamCha (Yet Another Multi-purpose CHunk Annotator) toolkit[2] ([4]), a free implementation of a multipurpose chunker that uses SVMs as the underlying algorithm for chunking and classification of chunks (more specifically, YamCha employs the free TinySVM library[3]). YamCha takes as input a tabular file where each line corresponds to the features of a token, as shown in Figure 3 (features other than the token form, POS tag, syntactic chunk and correct classification are omitted for simplicity).

At least, the first column must correspond to the token form (i.e. the "words"), and the last column indicates the target classification for that token (also known as the *golden tag*). The golden tags of the tokens in the context window can be used as dynamic features when training.

[1] http://www.coli.uni-saarland.de/~thorsten/tnt/
[2] http://chasen.org/~taku/software/yamcha/
[3] http://chasen.org/~taku/software/TinySVM/

	FORM	POS tag	SYNTAX	BIO tag
POS: -3	But	CC	O	O
POS: -2	even	RB	B-ADVP	O
POS: -1	last	JJ	B-NP	B-TIMEX
POS: 0	Thursday	NNP	I-NP	I-TIMEX
POS: +1	,	,	O	O
POS: +2	there	EX	B-NP	O
POS: +3	were	VBD	B-VP	O

Figure 3. Sample tokenized text with features (input to YamCha)

A SVM classifier requires that examples be represented as a vector of numeric features (that is, if the usual kernels involving dot products are to be used). Many features commonly used in NLP learning tasks are categorical (e.g. the token form and variations thereof like lowercase and patterns, the POS tag, the syntactic tag, ...); and moreover, some can have a domain of considerably many values — even several thousands—. Therefore, a transformation that maps categorical features into sets of *binary features* becomes a necessary step prior to training, which is performed internally by YamCha. The resulting feature vectors are very high-dimensional and sparse, with most binary features denoting the presence of a value of one of the original categorical features. With the use of contextual features, the number of dimensions grows accordingly.

SVMs are in principle binary classifiers, whereas ours is the multiclass classification problem of assigning a B, I or O tag to each token. YamCha supports both the one-vs-rest (training one binary classifier per class in the data, treating examples of one class as positive and the rest as negative), and one-vs-one strategies (training one binary classifier per each pair of classes, to discriminate between examples of the two). We chose to use one-vs-rest classification, which yields 3 classifiers, the same number as one-vs-one classification would in our particular case ($\binom{3}{2} = 3$). One often cited reason to employ one-vs-one classification has to do with efficiency, since in training classifiers to distinguish among examples from two particular classes in a pairwise fashion, the size of the training set for each individual classifier is greatly reduced, which diminishes overall training time. In our case with only 3 classes, the additional gain is not significant. Instead, we use one-vs-rest classification because our ultimate objective is to compare, under the same assumptions, the classification performance of SVMs with that of a rule-learning system whose knowledge representation is in the form of logic predicates (FOIL). A one-vs-one model does not have an obvious parallel in the hypothesis language of FOIL since, not being a discriminative learner, it attempts to reconstruct the set of positive cases for a single class rather than learn to distinguish among examples of two classes.

The combination of the margin outputs from each of the 3 classifiers in order to produce a coherent tag assignment is also handled internally by YamCha. The dynamic features (tag assignment for context tokens) are generated "on the fly" as classification proceeds. YamCha then uses dynamic programming based on the last N tokens (beam search) in order to select the final assignment of tags which maximizes the certainty score for the sequence of tokens (see [4] for details).

3.2. Rule-Induction Approach: FOIL

Inductive Logic Programming (ILP) refers to algorithms that extend traditional attribute-value concept learners (e.g. decision trees), by exploiting the *relations* among the examples' features and among examples themselves, and express them in the form of first-order logic predicates with variables. Target concepts are represented in the form of first-order *predicates* (also called *relations*). Hypotheses are definitions for each of the target predicates p_i, in the form of a set of logic clauses (rules) $p_i(X_1, \ldots, X_n) \leftarrow L_1, \ldots, L_m$, where each of the L_i are literals. A set of *training examples* $\mathcal{E}$, consisting of positive examples ($\mathcal{E}^+ \subseteq \mathcal{E}$), which satisfy the target predicate p_i and, optionally, negative examples ($\mathcal{E}^- \subset \mathcal{E}$), is supplied for the learner to induce a logic program. Each example is represented as a ground fact of the target predicate, i.e. as an n-tuple of constants $< x_1, \ldots, x_n >$ that define an assignment of values for the predicate arguments. In addition, the learner may be provided with *background knowledge* predicate definitions q_i, which serve as a vocabulary for the learner, in terms of which the hypothesis for target predicates p_i are constructed. The hypothesis must be consistent with both the training examples and the background knowledge. Refer to [5] for an in-depth discussion of ILP techniques, algorithms and various ILP systems in existence.

FOIL is an empirical (non-interactive, non-incremental) ILP system developed by Quinlan ([7], [8]). Arguments of relations in FOIL are typed, which means they take values over a particular domain. Representation of the background knowledge predicates q_i is restricted to extensional definitions of predicates in the form of a list of ground facts. Examples and, optionally, counterexamples for each of the tar-

get predicates must also be supplied as ground facts. From these elements, FOIL returns an hypothesis consisting of a set of function-free Horn clauses for each of the target predicates, whose body may contain as literals L_i any of the following: predicates from the background knowledge, any of the target predicates, a binding predicate ($X_i = X_j$), a predicate specifying an ordering relation for ordered types ($X_i < X_j$), or any of the former in the negated form $not\ L_i$.

Although in order to harness the full expressive power of the hypothesis language of FOIL (that is, first-order logic predicates), one should use a knowledge representation that takes into account possible relations among examples, we employ a *propositional* knowledge representation that mimics the one used with YamCha and which is explained in Section 2. The purpose of trying out this propositional approach is to compare how well ILP performs for this task in comparison with the SVMs by using the same corpus and the same set of features.

In order to transform this attribute-value representation of tokens into logic predicates that can be used as input to FOIL, each individual token is assigned an identifier. These token identifiers function as constant arguments for logic predicates. All logic predicates take a single argument: a token identifier (therefore we refer to this approach as *propositional*, because the predicates do not really express *relations*). Each possible feature of a token will be represented as a predicate of the background knowledge, with positive examples for the predicate being the identifiers of the tokens that have that feature. This way, we could have ground facts of the background knowledge such as those in Figure 4. Also, there will be three target predicates, which correspond to the three possible classifications of a token, namely: *begin_time_exp (X)*, *inside_time_exp(X)* and *outside_time_exp(X)*, which take a token identifier as argument. We used FOIL to train three individual classifiers to learn the B, I and O tag assignments respectively. As for the testing, we took advantage of the built-in inference engine of PROLOG.

Since we have trained three classifiers with FOIL independently of each other (one for each possible tag), each classifier provides an individual yes/no response for a token, which has to be combined with the responses from the remaining two in order to reach an agreement on the final tag assignment. Also, to maintain consistency, two restrictions apply: an I (inside) tag cannot follow an O (outside) tag, and the first token in a sentence cannot carry an I tag. We used the *confidence* of a rule as a measure of the evidence supporting each possible decision in order to establish a consensus among the classifiers' outputs:

$$conf(A \Leftarrow B) = \frac{\#(A \wedge B)}{\#B}$$

where $\#(A \wedge B)$ refers to the number of tokens (cases) that satisfy both the antecedent (B) and the consequent (A) of the rule, and $\#B$ refers to the number of tokens that satisfy the antecedent.

This quantity is computed for each clause by taking into account the ground facts (i.e. predicate facts derived from the tokens' features) about tokens from the training set. The *consensus* among the classifiers (B, I and O) makes use of the confidences of the rules that support each of the individual decisions. Because a 'yes' response from a classifier could be based on several rules (that is, the token in question could satisfy the antecedents of one or several clauses), each one having a different confidence value, we have tried two different approaches to computing the confidence of each possible decision (the results of both are reported in Section 4 below):

1. Considering the confidence of the best clause among those satisfied by the token.

2. Considering the sum of the confidences of all the clauses satisfied by the token.

The procedure for reaching the final agreement on the classification of a token is performed locally based only on information about the token in question and the tag decided upon for the previous token (thus, it is a form of greedy inference):

- If the three individual classifiers give a 'no' response, the token is assigned the O tag (which is largely the most often occurring tag).

- If one of the possible classifications is I (inside) and the token either appears at the start of a sentence, or the classification of the previous token was a O, the tag I is discarded from the options to consider —as it would create an inconsistent tag sequence— and the final classification is based on the remaining options alone.

- In any other case, the token is assigned the tag corresponding to the classification decision with the highest confidence.

One subtle detail regarding the testing stage is that, unlike in the training stage, it is not acceptable to take the correct classification tag for the context tokens in order to generate the dynamic features of the token. In order to produce coherent testing results, we need to generate this dynamic contextual features "on-the-fly": the corresponding ground facts about the previous tokens are added to the PROLOG base of facts as the classification takes place.

```
form_last(tok100).  // token 100 is 'last'
form_Thursday(tok101).  // token 101 is 'Thursday'
POS_NNP(tok101).  // token 101 is a proper noun
syn_I_NP(tok101).  // token 101 is inside a noun phrase
context_r1_form_Thursday(tok100).  // token right of tok100 is 'Thursday'
context_l1_B_NP(tok101).  // token left of tok101 is at the start of a noun phrase
                                    ...
```

Figure 4. Token features as ground facts of background knowledge predicates in FOIL

4 Experiments

For our experiments, we use the corpus of the ACE 2005 competition (Automatic Content Extraction) for training and testing. This corpus has the mentions of time expressions manually annotated using TIMEX tags. The composition of the corpus is 550 documents distributed in five categories (newswire, broadcast news, broadcast conversations, conversational telephone speech and weblogs), containing 257000 tokens and approx. 4650 time expression mentions. In all the experiments described in the following sections, the data partitions used for training and test contain a heterogeneous mixture of documents from the five categories, in equal proportions. Of the 257K tokens, 8809 tokens (a 3.42%) belong to time expressions.

Regarding our statistical approach with SVMs, we have carried out experiments in which we altered different parameters of learning: the feature set, the degree of the polynomial kernel, and the length of the context window. The motivation behind these additional experiments, whose results are developed below, was to quantify how variations in these three factors affected the classification performance. For this process, the ACE 2005 corpus was split in five partitions. Special care was taken in order to obtain balanced partitions with regard to the number of documents from each of the five categories. For the optimal configuration we repeat the experiments using 5-fold cross-validation, where five different models are constructed reserving each time a different partition for testing and using the remaining four for training. The rest of experiments do not use cross-validation owing to the rather long time required to train one SVM model for classification (approx. 8 hours with YamCha running in a computer cluster). Instead, for the remaining experiments which show the effects of altering parameters of learning, we have trained a single model for each different configuration, using the same data partitions for training and test as in Round 4 of the cross-validation (because the results for that round were closest to the average).

Lastly, we conducted an experiment with FOIL using the same data and feature set, according to the approach described in Section 3.2 above. In this case, due to reasons of temporal complexity, we did not perform a full cross-validation as above but a single round of training and test.

In all of the following tables, the columns *precision*, *recall* and F_1 *measure* refer to these three performance metrics as they are commonly understood in information extraction. The fourth column, *accuracy*, is a per-token correctness measure:

Precision: The rate of returned temporal expressions that are correctly identified (i.e. correctly tagged divided by total tagged).

Recall: The rate of existing temporal expressions that are correctly identified (i.e. correctly tagged divided by those that should have been tagged).

F_1 Score: It is the harmonic mean of the two previous values: $F_1 = \frac{2 \times Precision \times Recall}{(Precision + Recall)}$.

Accuracy: The percentage of correct BIO tag assignments predicted by the classifier at the token level (i.e. whether the predicted tag coincides with the target tag).

The precision, recall and F_1 scores are calculated in terms of full time expressions: only exact matches where the boundaries of the expression are correctly identified are counted as hits. Because the accuracy is computed over all tokens including the **O** tokens, which are easier to classify and significantly more common than **B** or **I** tokens, accuracy values are larger than F_1, precision, or recall scores

4.1. Optimal SVM Model

We report first the experiments with YamCha using the optimal combination of features, learning algorithm parameters and context window length.

Table 1 shows the results of performing five rounds of 5-fold cross-validation. For this experiment, the entire ACE corpus was split in 5 partitions, balanced with regard to the number of documents from each category (see Section 4 above). In each round, 4 of these 5 partitions were used as training data and the remaining one was reserved for testing the learned model. We employed the full set of token features described in Section 2, a polynomial kernel of degree 2, and a context window of 2 tokens left and right (only the left context is used for the dynamic feature).

	PREC	RECALL	F_1	ACC
Round 1	81.33	75.23	78.16	98.68
Round 2	77.74	70.46	73.92	98.60
Round 3	75.92	71.22	73.50	98.47
Round 4	80.05	73.71	76.75	98.65
Round 5	80.34	72.54	76.24	98.66
AVERAGE	**79.08**	**72.63**	**75.71**	**98.61**
STD DEV.	2.20	1.91	1.97	0.17

Table 1. 5-fold cross-validation in SVM with all features, quadratic kernel and a 2-token context

4.2. Effects of Varying the Kernel Degree

Table 2 shows the gain in performance achieved by varying the degree of the polynomial kernel. The increments between brackets indicate the difference with respect to the reference configuration: the quadratic kernel. As seen, the quadratic kernel achieves the best performance, and the best tradeoff between results and complexity. An explanation for this fact is that the lineal kernel is too simple to capture the complexity of this problem, and produces underfitting; whereas the cubic kernel is too complex, adds too many complex, unnecessary combinations of features and, hence, overfits. Besides, empirically, the quadratic kernel is known to produce the best results in NLP chunking and classification tasks.

4.3. Effects of Incremental Feature Sets

In order to study which types of features are relevant, we have designed three incremental models that use increasingly complex sets of features. The features used in each of these models are as follows:

- Model 1: the token form and the token form in lowercase.
- Model 2: Model 1 + POS tags + format features (isAllCaps, isAllCapsOrDots, isAllDigits, isAllDigitsOrDots, initialCap) + the token form removing alphabetic characters + the token form removing alphanumeric characters.
- Model 3: Model 2 + syntactic chunk tags + classes of words (isNumber, isMultiplier, isDay, isMonth).

Table 3 shows the effect of incrementally using a richer feature set. The increments between brackets indicate the difference with respect to the reference configuration: the maximal feature set (model 3). The results indicate that all of the features initially considered contribute to classification performance. A context window of 2 tokens left and right and a quadratic kernel were used in all three models.

4.4. Effects of Increasing the Context Window

And last, Table 4 shows the results obtained from increasing the size of the context window used for the contextual features. It can be observed that passing from a context of 2 tokens to the left and right to a context of 3 tokens in each direction goes in detriment of recall. Enlarging the context window causes the system to overfit on the training corpus, which, as expected, has a negative effect on the testing partition.

4.5. FOIL

We collected the set of clauses produced by FOIL for the three target predicates into one file of rules for PROLOG. In a similar manner, we produced a file of PROLOG facts which contained ground facts about the features of the tokens in the test partition. Next, we ran a PROLOG program to output our classifier's guess (that is, a B, I or O tag) for each of the test set tokens in sequence, alongside the correct tag for each token in a tabular file. And lastly, we used a PERL script that computes the precision, recall and F_1 measures to evaluate the results.

In the evaluation of FOIL, we did not conduct cross-validation on this experiment due to the high temporal cost of training a model with FOIL for our full data set. As our aim is to compare the results with the performance of the SVM on equal terms, instead of using a reduced training set, we conducted a single experiment with FOIL which uses the optimal configuration achieved with the SVM: the full set of features as in model 3 above, and a context window of 2 tokens left and right. The comparison is with respect to Partition 4 of the 5-fold cross-validation experiment with YamCha.

Table 5 shows the precision, recall and F_1 values achieved by the classifiers trained with FOIL. We include the results obtained with both strategies for computing the confidence of a classification decision ('best' indicates the confidence of the best clause is taken, 'sum' indicates that the sum of confidences of all the satisfied clauses is taken).

KERNEL	PREC	RECALL	F_1	ACC
pol. lineal	72.39 (-7.66)	70.08 (-3.63)	71.21 (-5.54)	98.25 (-0.40)
pol. quadratic	**80.05**	**73.71**	**76.75**	**98.65**
pol. cubic	81.30 (+1.25)	71.73 (-1.98)	76.21 (-0.54)	98.65 (+0.00)

Table 2. Effects on performance of varying the degree of the kernel (SVM)

FEATURES	PREC	RECALL	F_1	ACC
Model 1	80.00 (-0.05)	66.89 (-6.82)	72.86 (-3.89)	98.56 (-0.09)
Model 2	80.10 (+0.05)	71.73 (-1.98)	75.68 (-1.07)	98.60 (-0.05)
Model 3	**80.05**	**73.71**	**76.75**	**98.65**

Table 3. Effects on performance of incrementally extending the feature set (SVM)

4.6. Comparison of results

In Table 5, the quantities in bold correspond to the best result obtained by FOIL (with each of the two strategies for computing the confidence), and the numbers in parentheses are the score differences of the SVM classifier with respect to FOIL. It can be observed that the precision values are close in both cases, but there is consistently a drop in the recall value of FOIL with respect to that achieved by the SVM. This loss in recall damages the F_1 figures of FOIL considerably. We believe that this low recall of FOIL is due to an implicit bias of the learning method that makes it more susceptible to committing overfitting than, for instance, an SVM, provided that the two classifiers were trained with exactly the same data and feature sets.

The considerably higher recall of the SVMs can be explained by the fact that SVMs are max-margin classifiers. Furthermore, SVMs are able to work with many combinations of features at the same time, and find the best combinations without a penalty in efficiency, which confers them higher generalization ability regardless of the high dimensionality of the data. This is possible thanks to the use of the "kernel trick". Other algorithms, such as FOIL, rely on heuristics in order to select the best features and make the dimensionality of the data manageable, which has a negative impact on overall generalization.

4.7. Performance issues

Training has been conducted in a cluster of workstations with Pentium 4 3.20 GHz and 4 GBytes of RAM. Training a model with YamCha took approximately 8 hours, with slight variations from 6 to 12 hours depending on the degree of the polynomial kernel, the subset of features and the context window length.

We encountered that the amount of time that FOIL requires to train a model for a large dataset and number of features (as it is our case with the ACE corpus and the feature set described above) renders it impractical for real use. Training 3 classifiers (for the B, I and O tags) with FOIL took over three and a half weeks for each —the three of them were trained simultaneously—. FOIL's temporal complexity is of order $\mathcal{O}(\|\mathcal{B}\| \times \|\mathcal{E}^+ \cup \mathcal{E}^-\| \times A)$, where $\|\mathcal{B}\|$ is the number of background knowledge predicates, $\|\mathcal{E}^+ \cup \mathcal{E}^-\|$ is the number of examples and counterexamples for the target predicate and A is the maximum arity (number of arguments) of a predicate. Our training set consists of 192182 tokens, 3613 of which are tagged as B (start of temporal expression), 3187 as I (inside temporal expression) and 185364 as O (outside). In the propositional knowledge representation, the number of domain predicates other than the 3 target predicates (i.e. predicates corresponding to possible features of the individual tokens) is 159175, and all predicates have arity 1 (the only variable is the token identifier).

Nevertheless, we attempted to speed up the training of our model with FOIL, at the expense of filtering predicates from the background knowledge (thus reducing the number of available options for FOIL to construct literals for clauses) that had few positive examples, and reducing the number of counterexamples for the target predicates using an ad hoc measure of relevance of a counterexample with respect to the set of positive examples. We achieved some substantial reduction in training time in this way, but at the expense of considerable penalty in the classification results: a decrease of 8% in both precision and recall.

5. Conclusions

We have evaluated the performance achieved by the SVM and FOIL, and have observed that, while the precision scores lie within an acceptable range, the SVM significantly outperforms FOIL with respect to recall. The explanation is that SVMs are max-margin classifiers that generalize better on sparse corpora, such as the one used in this paper. We saw also that the execution time of FOIL for training a full

WINDOW	PREC	RECALL	F_1	ACC
-1 .. +1	74.47 (-5.58)	72.83 (-0.88)	73.64 (-3.11)	98.41 (-0.24)
-2 .. +2	**80.05**	**73.71**	**76.75**	**98.65**
-3 .. +3	80.30 (+0.25)	71.29 (-2.42)	75.52 (-1.23)	98.59 (-0.06)

Table 4. Effects of increasing the context window (SVM)

CLASSIFIER	PREC	RECALL	F_1	ACC
FOIL (best)	77.58	**52.15**	**62.37**	97.95
FOIL (sum)	**81.32**	50.28	62.13	**97.98**
SVM	80.05 (-1.27)	73.71 (+21.56)	76.75 (+14.38)	98.65 (+0.67)

Table 5. Performance of the propositional approach with FOIL vs. a SVM classifier

model under the assumptions that occupy us in this problem renders its usage impractical, and that simplifying the training data set in order to reduce execution time carries a non-negligible penalty associated in both precision and recall.

Our optimal model for temporal expression chunking with SVM yields a F_1 score of 75.71%. The best performance in a task like time expression recognition, which has an important syntactic component and, to a lesser extent, semantic, will still be achieved by a system that integrates handwritten knowledge of some kind (these systems consistently achieve F_1 scores above 90%). The main drawback of using grammars is that grammar rules have to be tailored specifically to the task at hand. This is one of the reasons why we wanted to try out a purely statistical method such as SVMs and observe the classification performance that such a method affords.

Two potentially positive aspects of SVMs, regarding their application to our problem, are: first, SVM's resistance to overfitting, even in the presence of very high dimensional feature spaces (as those that arise typically in NLP problems); second, the fact that SVMs allow one to freely experiment with different sets of features, and that by the use of non-linear kernels one can entrust the learning method with automatically finding interesting combinations of these features.

References

[1] B. Boguraev and R. K. Ando. Timeml-compliant text analysis for temporal reasoning. In *Proc. of the 19th International Joint Conference on Artificial Intelligence*, pages 997–1003, 2005.

[2] L. Ferro, L. Gerber, I. Mani, B. Sundheim, and G. Wilson. Tides standard for the annotation of temporal expressions v1.3. Technical report, MITRE Corporation, 2003.

[3] K. Hacioglu, Y. Chen, and B. Douglas. Automatic time expression labelling for english and chinese text. In *Proc. of the 6th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, pages 548–559. Springer, 2005.

[4] T. Kudo and Y. Matsumoto. Chunking with support vector machines. In *Proc. of 2nd Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 192–199, Pittsburgh, PA, 2001.

[5] N. Lavrač and S. Džeroksi. *Inductive Logic Programming: Techniques and Applications*. Prentice Hall, 1994.

[6] C. Perrig. Identification of time expressions, signals, events and temporal relations in texts. http://www.cs.lth.se/EDA171/Reports/2005/cyril.pdf, 2005.

[7] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[8] J. R. Quinlan and R. M. Cameron-Jones. Foil: A midterm report. In *Proc. European Conference on Machine Learning*, pages 3–20. Springer Verlag, 1993.

[9] E. Saquete, R. Muñoz, and P. Martínez-Barco. Event ordering using terseo system. In *Proc. of the 9th International Conference on Application of Natural Language to Information Systems (NLDB)*, pages 39–50. Springer, 2004.

[10] J. Turmo and H. Rodriguez. Learning rules for information extraction. *Natural Language Engineering*, 8:167–191, 2002.

[11] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, USA, 1995.

[12] G. Wilson, I. Mani, B. Sundheim, and L. Ferro. A multilingual approach to annotating and extracting temporal information. In *Proc. Workshop on Temporal and Spatial Information Processing Vol. 13*, pages 1–7, Morristown, NJ, USA, 2001. ACL.