ACL-05

Feature Engineering for Machine Learning in Natural Language Processing

Proceedings of the Workshop

29 June 2005
University of Michigan
Ann Arbor, Michigan, USA

Production and Manufacturing by
Omnipress Inc.
Post Office Box 7214
Madison, WI 53707-7214

Sponsorship gratefully received from
Microsoft Research
One Microsoft Way
Redmond, Washington 98052, USA

Order copies of this and other ACL proceedings from:

Association for Computational Linguistics (ACL)
75 Paterson Street, Suite 9
New Brunswick, NJ 08901
USA
Tel: +1-732-342-9100
Fax: +1-732-342-9339
acl@aclweb.org

Introduction

The ACL 2005 Workshop on Feature Engineering for Machine Learning in Natural Language Processing is an opportunity to explore the various dimensions of feature engineering for problems that are of interest to the ACL community. Feature Engineering encompasses feature design, feature selection, feature induction, studies of feature impact (including feature ablation studies), and related topics. In 2003, there was a NIPS workshop on feature engineering ("Feature Extraction and Feature Selection"), but the focus was not on NLP problems specifically. Also, although the various aspects of feature engineering have been dealt with at times in various ACL forums, until now, to our knowledge, the spotlight has never been shone directly on this topic specifically for NLP and language technology problems. We feel that now is the time to look more closely.

As experience with machine learning for solving natural language processing tasks accumulates in the field, practitioners are finding that feature engineering is as critical as the choice of machine learning algorithm, if not more so. Feature engineering significantly affects the performance of systems and deserves greater attention. Also, in the wake of the shift in our field away from knowledge engineering and of the successes of data-driven and statistical methods, researchers are likely to make further progress by incorporating additional, sometimes familiar, sources of knowledge as features. Feature design may benefit from expert insight even where the relative merits of features must be assessed through empirical techniques from data. Although some experience in the area of feature engineering is to be found in the theoretical machine learning community, the particular demands of natural language processing leave much to be discovered.

In the call for papers, we expressed our intent of bringing together practitioners of NLP, machine learning, information extraction, speech processing, and related fields with the goal of sharing experimental evidence for successful approaches to feature engineering. Judging by the quality and diversity of the submissions received, we believe we have succeeded, and the resulting program should be of great interest to many researchers in the ACL community. We hope that the workshop will contribute to the distillation of best practices and to the discovery of new sources of knowledge and features previously untapped.

We also extend an open invitation to the reader to continue investigation in all aspects of feature engineering for machine learning in NLP, including:

- Novel methods for discovering or inducing features, such as mining the web for closed classes, useful for indicator features.
- Comparative studies of different feature selection algorithms for NLP tasks.
- Error analysis tools that help researchers to identify ambiguous cases that could be disambiguated by the addition of features.
- Error analysis of various aspects of feature induction, selection, representation.
- Issues with representation, e.g., strategies for handling hierarchical representations, including decomposing to atomic features or by employing statistical relational learning.

- Techniques used in fields outside NLP that prove useful in NLP.
- The impact of feature selection and feature design on such practical considerations as training time, experimental design, domain independence, and evaluation.
- Analysis of feature engineering and its interaction with specific machine learning methods commonly used in NLP.
- Ensemble methods employing diverse types of features.
- Studies of methods for inducing a feature set, for example by iteratively expanding a base feature set.
- Issues with representing and combining real-valued and categorical features for NLP tasks.

We anticipate that contributions in these areas will move the field of NLP and language technologies forward, with greater system performance and further insight into our own data and perhaps language itself.

We wish to thank all of the researchers who submitted papers to the workshop. Also, thanks go to the entire program committee (see next page) and those who assisted them in their reviewing responsibilities.

Best regards,

Eric Ringger, Microsoft Research (USA)

20 May 2005

Organizer:

Eric Ringger, Microsoft Research (USA)

Program Committee:

Eric Ringger, Microsoft Research (USA)
Simon Corston-Oliver, Microsoft Research (USA)
Kevin Duh, University of Washington (USA)
Matthew Richardson, Microsoft Research (USA)
Oren Etzioni, University of Washington (USA)
Andrew McCallum, University of Massachusetts at Amherst (USA)
Dan Bikel, IBM Research (USA)
Olac Fuentes, INAOE (Mexico)
Christopher Manning, Stanford University (USA)
Kristina Toutanova , Stanford University (USA)
Hideki Isozaki, NTT Communication Science Laboratories (Japan)
Caroline Sporleder, University of Edinburgh (UK)

Additional Reviewers:

Thamar Solorio, INAOE (Mexico)

Invited Speaker:

Andrew McCallum, University of Massachusetts at Amherst (USA)

Table of Contents

Conference Program

Wednesday, June 29, 2005

8:45–9:00 Opening Remarks

Session W4.1: Classification

A Novel Machine Learning Approach for the Identification of Named Entity Relations
Tianfang Yao and Hans Uszkoreit

Feature Engineering and Post-Processing for Temporal Expression Recognition Using Conditional Random Fields
Sisay Fissaha Adafre and Maarten de Rijke

Temporal Feature Modification for Retrospective Categorization
Robert Liebscher and Richard K. Belew

10:30–11:00 Break

11:00–12:00 Invited Talk by Andrew McCallum

Using Semantic and Syntactic Graphs for Call Classification
Dilek Hakkani-Tür, Gokhan Tur and Ananlada Chotimongkol

12:30–14:00 Lunch

Session W4.2: Discourse and Syntax

Feature-Based Segmentation of Narrative Documents
David Kauchak and Francine Chen

Identifying non-referential it: a machine learning approach incorporating linguistically motivated patterns
Adriane Boyd, Whitney Gegg-Harrison and Donna Byron

Engineering of Syntactic Features for Shallow Semantic Parsing
Alessandro Moschitti, Bonaventura Coppola, Daniele Pighin and Roberto Basili

15:30–16:00 Break

Wednesday, June 29, 2005 (continued)

Session W4.3: Feature Sources

Automatic identification of sentiment vocabulary: exploiting low association with known sentiment terms
Michael Gamon and Anthony Aue

Studying Feature Generation from Various Data Representations for Answer Extraction
Dan Shen, Geert-Jan M. Kruijff and Dietrich Klakow

A Novel Machine Learning Approach for the Identification of Named Entity Relations

Tianfang Yao
Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, 200030, China
yao-tf@cs.sjtu.edu.cn

Hans Uszkoreit
Department of Computational Linguistics and Phonetics
Saarland University
Saarbruecken, 66041, Germany
uszkoreit@coli.uni-sb.de

Abstract

In this paper, a novel machine learning approach for the identification of named entity relations (NERs) called positive and negative case-based learning (PNCBL) is proposed. It pursues the improvement of the identification performance for NERs through simultaneously learning two opposite cases and automatically selecting effective multi-level linguistic features for NERs and non-NERs. This approach has been applied to the identification of domain-specific and cross-sentence NERs for Chinese texts. The experimental results have shown that the overall average recall, precision, and F-measure for 14 NERs are 78.50%, 63.92% and 70.46% respectively. In addition, the above F-measure has been enhanced from 63.61% to 70.46% due to adoption of both positive and negative cases.

1 Introduction

The investigation for Chinese information extraction is one of the topics of the project COLLATE dedicated to building up the German Competence Center for Language Technology. After accomplishing the task concerning named entity (NE) identification, we go on studying identification issues for named entity relations (NERs). As an initial step, we define 14 different NERs based on six identified NEs in a sports domain based Chinese named entity recognition system (Yao et al., 2003). In order to learn NERs, we annotate the output texts from this system with XML. Meanwhile, the NER annotation is performed by an interactive mode.

The goal of the learning is to capture valuable information from NER and non-NER patterns, which is implicated in different features and helps us identify NERs and non-NERs. Generally speaking, because not all features we predefine are important for each NER or non-NER, we should distinguish them by a reasonable measure mode. According to the selection criterion we propose - self-similarity, which is a quantitative measure for the concentrative degree of the same kind of NERs or non-NERs in the corresponding pattern library, the effective feature sets - general-character feature (GCF) sets for NERs and individual-character feature (ICF) sets for non-NERs are built. Moreover, the GCF and ICF feature weights serve as a pro portion determination of the features' degree of importance for identifying NERs against non-NERs. Subsequently, identification thresholds can also be determined.

In the NER identification, we may be confronted with the problem that an NER candidate in a new case matches more than one positive case, or both positive and negative cases. In such situations, we have to employ a vote to decide which existing

Proceedings of the ACL Workshop on Feature Engineering for Machine Learning in NLP, pages 1–8,
Ann Arbor, June 2005.

case environment is more similar to the new case. In addition, a number of special circumstances should be also considered, such as relation conflict and relation omission.

2 Definition of Relations

An NER may be a modifying / modified, dominating / dominated, combination, collocation or even cross-sentence constituent relationship between NEs. Considering the distribution of different kinds of NERs, we define 14 different NERs based on six identified NEs in the sports domain shown in Table 1.

NER Category	Explanation
PS_TM	The membership of a person in a sports team.
PS_CP	A person takes part in a sports competition.
PS_CPC	The origin location of a person.
PS_ID	A person and her / his position in a sports team or other occasions.
HT_VT	The home and visiting teams in a sports competition.
WT_LT	The winning and losing team name in a sports match.
DT_DT	The names of two teams which draw a match.
TM_CP	A team participates in a sports competition.
TM_CPC	It indicates where a sports team comes from.
ID_TM	The position of a person employed by a sports team.
CP_DA	The staged date for a sports competition.
CP_TI	The staged time for a sports competition.
CP_LOC	It gives the location where a sports match is held.
LOC_CPC	The location ownership (LOC belongs to CPC).

Table 1. NER Category

In order to further indicate the positions of NEs in an NER, we define a general frame for the above NERs and give the following example using this description:

Definition 1 (General Frame of NERs):
NamedEntityRelation (NamedEntity$_1$, ParagraphSentenceNamedEntityNo$_1$; NamedEntity$_2$, ParagraphSentenceNamedEntityNo$_2$)

Example 1:

广东宏远队[1]客场以 3 比 0 击败广州太阳神队。
The Guangdong Hongyuan Team defeated the Guangzhou Taiyangshen Team by 3: 0 in the guest field.

In the sentence we observe that there exist two NERs. According to the general frame, the first NER description is HT_VT(广州太阳神队 (Guangzhou Taiyangshen Team), 1-1-2; 广东宏远队(Guangdong Hongyuan Team), 1-1-1) and the other is WT_LT(广东宏远队 (Guangdong Hongyuan Team), 1-1-1; 广州太阳神(Guangzhou Taiyangshen Team), 1-1-2).

In this example, two NERs represent dominating / dominated and collocation relationships separately: namely, the first relation HT_VT gives the collocation relationship for the NE "Guangdong Hongyuan Team" and the noun "guest field". This implies that "Guangdong Hongyuan Team" is a guest team. Adversely, "Guangzhou Taiyangshen Team" is a host team; the second relation WT_LT indicates dominating / dominated relationship between "Guangdong Hongyuan Team" and "Guangzhou Taiyangshen Team" by the verb "defeat". Therefore, "Guangdong Hongyuan Team" and "Guangzhou Taiyangshen Team" are the winning and losing team, respectively.

3 Positive and Negative Case-Based Learning

The positive and negative case-based learning (PNCBL) belongs to supervised statistical learning methods (Nilsson, 1996). Actually, it is a variant of memory-based learning (Stanfill and Waltz, 1986; Daelemans, 1995; Daelemans et al., 2000). Unlike memory-based learning, PNCBL does not simply store cases in memory but transforms case forms into NER and non-NER patterns. Additionally, it stores not only positive cases, but also negative ones. Here, it should be clarified that the negative case we mean is a case in which two or more NEs do not stand in any relationships with each other, i.e, they bear non-relationships which are also investigated objects in which we are interested.

During the learning, depending on the average similarity of features and the self-similarity of NERs (also non-NERs), the system automatically selects general or individual-character features (GCFs or ICFs) to construct a feature set. It also determines different feature weights and identification thresholds for different NERs or non-NERs. Thus, the learning results provide an identification references for the forthcoming NER identification.

3.1 Relation Features

Relation features, by which we can effectively identify different NERs, are defined for capturing critical information of the Chinese language. According to the features, we can define NER / non-

[1] The underlining of Chinese words means that an NE consists of these words.

NER patterns. The following essential factors motivate our definition for relation features:

- The relation features should be selected from multiple linguistic levels, i.e., morphology, grammar and semantics (Cardie, 1996);
- They can help us to identify NERs using positive and negative case-based machine learning as their information do not only deal with NERs but also with non-NERs; and
- They should embody the crucial information of Chinese language processing (Dang et al., 2002), such as word order, the context of words, and particles etc.

There are a total of 13 relation features shown in Table 2, which are empirically defined according to the above motivations. It should be explained that in order to distinguish feature names from element names of the NER / non-NER patterns, we add a capital letter "F" in the ending of feature names. In addition, a sentence group in the following definitions can contain one or multiple sentences. In other words, a sentence group must end with a stop, semicolon, colon, exclamation mark, or question mark.

Feature Category	Explanation
SGTF	The type of a sentence group in which there exists a relation.
NESPF	The named entities of a relevant relation are located in the same sentence or different sentences.
NEOF	The order of the named entities of a relevant relation.
NEVPF	The relative position between the verbs and the named entities of a relevant relation. The verbs of a relevant relation mean that they occur in a sentence where the relation is embedded.
NECF	The context of named entities. The context only embodies a word or a character preceding or following the current named entity.
VSPF	The verbs are located in the same sentence or different sentences in which there is a relevant relation.
NEPPOF	The relative order between parts-of-speech of particles and named entities. The particles occur within the sentences where the relation is embedded.
NEPF	The parts-of-speech of the named entities of a relevant relation.
NECPF	The parts-of-speech of the context for the named entities associated with a relation.
SPF	The sequence of parts-of-speech for all sentence constituents within a relation range.
VVF	The valence expression of verbs in the sentence(s) where there is a relation embedded.
NECTF	The concepts of the named entities of a relevant relation from HowNet (Dong and Dong, 2000).
VCTF	The concepts of the verbs of a relevant relation from HowNet.

Table 2. Feature Category

In 13 features, three features (NECF, NECPF and NEPF) belong to morphological features, three features (NEOF, SPF and SGTF) are grammatical features, four features (NEPPOF, NESPF, NEVPF and VSPF) are associated with not only morphology but also grammar, and three features (NECTF, VCTF and VVF) are semantic features.

Every feature describes one or more properties of a relation. Through the feature similarity calculation, the quantitative similarity for two relations can be obtained, so that we can further determine whether a candidate relation is a real relation. Therefore, the feature definition plays an important role for the relation identification. For instance, NECF can capture the noun 客场 (the guest field, it means that the guest team attends a competition in the host team's residence.) and also determine that the closest NE by this noun is 广东宏远队 (the Guangdong Hongyuan Team). On the other hand, NEOF can fix the sequence of two relation-related NEs. Thus, another NE 广州太阳神队 (the Guangzhou Taiyangshen Team) is determined. Therefore, these two features reflect the properties of the relation HT_VT.

3.2 Relation and Non-Relation Patterns

A relation pattern describes the relationships between an NER and its features. In other words, it depicts the linguistic environment in which NERs exist.

Definition 2 (Relation Pattern): A relation pattern (RP) is defined as a 14-tuple: RP = (NO, RE, SC, SGT, NE, NEC, VERB, PAR, NEP, NECP, SP, VV, NECT, VCT) where *NO* represents the number of a RP; *RE* is a finite set of relation expressions; *SC* is a finite set for the words in the sentence group except for the words related to named entities; *SGT* is a sentence group type; *NE* is a finite set for named entities in the sentence group; *NEC* is a finite set that embodies the context of named entities; *VERB* is a finite set that includes the sequence numbers of verbs and corresponding verbs; *NEP* is a finite set of named entities and their POS tags; *NECP* is a finite set which contains the POS tags of the context for named entities; *SP* is a finite set in which there are the sequence numbers as well as corresponding POS tags and named entity numbers in a sentence group; *VV* is a finite set comprehending the posi-

tion of verbs in a sentence and its valence constraints from Lexical Sports Ontology which is developed by us; *NECT* is a finite set that has the concepts of named entities in a sentence group; and *VCT* is a finite set which gives the concepts of verbs in a sentence group.

Example 2:

据新华社北京 3 月 2 6 日电全国足球甲 B 联赛今天进行了第二轮赛事的 5 场比赛，广东宏远队客场以 3 比 0 击败广州太阳神队，成为唯一一支两战全胜的队伍，暂居积分榜榜首。

According to the news from Xinhua News Agency Beijing on March 26th: National Football Tournament (the First B League) today held five competitions of the second round, The Guangdong Hongyuan Team defeats the Guangzhou Taiyangshen Team by 3: 0 in the guest field, becoming the only team to win both matches, and temporarily occupying the first place of the entire competition.

Relation Pattern:
NO = 34;
RE = {(CP_DA, NE1-3, NE1-2), (CP_TI, NE1-3, NE1-4), …, (WT_LT, NE2-1, NE2-2)}
SC = {(1, 据, according_to, Empty, AccordingTo), (2, 新华社, Xinhua/Xinhua_News_agency, Empty, institution/news/ProperName/China), …, (42, 。, ., Empty, {punc})};
SGT = multi-sentences;
NE = {(NE1-1, 3, LN, {(1, 北京)}), (NE1-2, 4, Date, {(1, 3), (2, 月), (3, 2 6), (4, 日)}), ..., (NE2-2, 26, TN, {(1, 广州), (2, 太阳神), (3, 队)})};
NEC = {(NE1-1, 新华社, 3), (NE1-2, 北京, 电), ..., (NE2-2, 击败, ,) };
VERB = {(8, 进行), (25, 击败), ..., (39, 居)}
PAR = {(1, 据), (9, 了), ..., (38, 暂)};
NEP = {(NE1-1, {(1, N5)}), (NE1-2, {(1, M), (2, N), (3, M), (4, N)}), ..., (NE2-2, {(1, N5), (2, N), (3, N)})};
NECP = {(NE1-1, N, M), (NE1-2, N5, N), …, (NE2-2, V, W)};
SP = {(1, P), (2, N), (3, NE1-1), ..., (42, W)};
VV = {(V_8, {Agent|fact/compete|CT, -Time|time|DT}), (V_25, {Agent|human/mass|TN, Patient|human/mass|TN}),..., (V_39, {Agent|human/sport|PN, Agent|human/mass|TN})};
NECT = {(NE1-1, place/capital/ProperName/China), (NE1-2, Empty+celestial/unit/time+Empty+ celestial/time/time/morning), …, (NE2-2, place/city/ProperName/China+ Empty+community/human/mass)};
VCT = {(V_8, GoForward/GoOn/Vgoingon), (V_25, defeat), …, (V_39, reside/situated)}

Analogous to the definition of the relation pattern, a non-relation pattern is defined as follows:

Definition 3 (Non-Relation Pattern): A non-relation pattern (NRP) is also defined as a 14-tuple: NRP = (NO, NRE, SC, SGT, NE, NEC, VERB, PAR, NEP, NECP, SP, VV, NECT, VCT), where NRE is a finite set of non-relation expressions which specify the nonexistent relations in a sentence group. The definitions of the other elements are the same as the ones in the relation pattern. For example, if we build an NRP for the above sentence group in Example 2, the NRE is listed in the following:

NRE = {(CP_LOC, NE1-3, NE1-1), (TM_CPC, NE2-1, NE1-1), ..., (DT_DT, NE2-1, NE2-2)}

In this sentence group, the named entity (CT) 全国足球甲 B 联赛 (National Football Tournament (the First B League)) does not bear the relation CP_LOC to the named entity (LN) 北京 (Beijing). This LN only indicates the release location of the news from Xinhua News Agency.

As supporting means, the non-NER patterns also play an important role, because in the NER pattern library we collect sentence groups in which the NER exists. If a sentence group only includes non-NERs, obviously, it is excluded from the NER pattern library. Thus the impact of positive cases cannot replace the impact of negative cases. With the help of non-NER patterns, we can remove misidentified non-NERs and enhance the precision of NER identification.

3.3 Similarity Calculation

In the learning, the similarity calculation is a kernel measure for feature selection.

Definition 4 (Self-Similarity): The self-similarity of a kind of NERs or non-NERs in the corresponding library can be used to measure the concentrative degree of this kind of relations or non-relations. The value of the self-similarity is between 0 and 1. If the self-similarity value of a kind of relation or non-relation is close to 1, we can say that the concentrative degree of this kind of relation or non-relation is very "tight". Conversely, the concentrative degree of that is very "loose".

The calculation of the self-similarity for the same kind of NERs is equal to the calculation for the average similarity of the corresponding relation features. Suppose R(i) is a defined NER in the NER set ($1 \le i \le 14$). The average similarity for this kind of NERs is defined as follows:

$$Sim_{average}(R(i)) = \frac{\sum_{1 \le j,\, k \le m;\, j \ne k} Sim\,(R(i)_j, R(i)_k)}{Sum_{relation\_pair}(R(i)_j, R(i)_k)} \quad (1)$$

where Sim $(R(i)_j, R(i)_k)$ denotes the relation similarity between the same kind of relations, $R(i)_j$ and

$R(i)_k$. $1 \le j, k \le m, j \ne k$; m is the total number of the relation R(i) in the NER pattern library. The calculation of $Sim(R(i)_j, R(i)_k)$ depends on different features. $Sum_{relation\_pair}(R(i)_j, R(i)_k)$ is the sum of calculated relation pair number. They can be calculated using the following formulas:

$$Sim(R(i)_j, R(i)_k) = \frac{\sum_{t=1}^{Sumf} Sim(R(i)_j, R(i)_k)(f_t)}{Sum_f} \quad (2)$$

$$Sum_{relation\_pair}(R(i)_j, R(i)_k) = \begin{cases} 1 & m = 2 \\ \frac{m!}{(m-2)! * 2!} & m > 2 \end{cases} \quad (3)$$

In the formula (2), f_t is a feature in the feature set ($1 \le t \le 13$). Sum_f is the total number of features. The calculation formulas of $Sim(R(i)_j, R(i)_k)(f_t)$ depend on different features. For example, if f_t is equal to NECF, $Sim(R(i)_j, R(i)_k)(f_t)$ is shown as follows:

$$Sim(X(i)_j, X(i)_k)(NECF) = \begin{cases} 1 & \text{if all contexts of named entities for two relations are the same} \\ 0.75 & \text{if only a preceding or following context is not the same} \\ 0.5 & \text{if two preceding and / or following contexts are not the same} \\ 0.25 & \text{if three preceding and / or following contexts are not the same} \\ 0 & \text{if all contexts of named entities for two relations are not the same} \end{cases} \quad (4)$$

Notice that the similarity calculation for non-NERs is the same as the above calculations.

Before describing the learning algorithm, we want to define some fundamental conceptions related to the algorithm as follows:

Definition 5 (General-Character Feature): If the average similarity value of a feature in a relation is greater than or equal to the self-similarity of this relation, it is called a General-Character Feature (GCF). This feature reflects a common characteristic of this kind of relation.

Definition 6 (Individual-Character Feature): An Individual-Character Feature (ICF) means its average similarity value in a relation is less than or equal to the self-similarity of this relation. This feature depicts an individual property of this kind of relation.

Definition 7 (Feature Weight): The weight of a selected feature (GCF or ICF) denotes the important degree of the feature in GCF or ICF set. It is used for the similarity calculation of relations or non-relations during relation identification.

$$f(s)_w(R(i)) = \frac{Sim_{average}f(s)(R(i))}{\sum_{t=1}^{n} Sim_{average}f(t)(R(i))} \quad (5)$$

where R(i) is a defined relation in the NER set ($1 \le i \le 14$); n is the size of selected features, $1 \le s, t \le n$; and

$$Sim_{average}f(s)(R(i)) = \frac{\sum_{1 \le j, k \le m; j \ne k} Sim(R(i)_j, R(i)_k)(f(s))}{Sum_{relation\_pair}(R(i)_j, R(i)_k)} \quad (6)$$

$Sim(R(i)_j, R(i)_k)(f(s))$ computes the feature similarity of the feature f(s) between same kinds of relations, $R(i)_j$ and $R(i)_k$. $1 \le j, k \le m, j \ne k$; m is the total number of the relation R(i) in the NER pattern library. $Sum_{relation\_pair}(R(i)_j, R(i)_k)$ is the sum of calculated relation pair numbers, which can be calculated by the formula (3).

Definition 8 (Identification Threshold): If a candidate relation is regarded as a relation in the relation pattern library, the identification threshold of this relation indicates the minimal similarity value between them. It is calculated by the average of the sum of average similarity values for selected features:

$$IdenThrh(R(i)) = \frac{\sum_{t=1}^{n} Sim_{average}f(t)(R(i))}{n} \quad (7)$$

where n is the size of selected features, $1 \le t \le n$.

Finally, the PNCBL algorithm is described as follows:

1) Input annotated texts;
2) Transform XML format of texts into internal data format;
3) Build NER and non-NER patterns;
4) Store both types of patterns in hash tables and construct indexes for them;

5) Compute the average similarity for features and self-similarity for NERs and non-NERs;
6) Select GCFs and ICFs for NERs and non-NERs respectively;
7) Calculate weights for selected features;
8) Decide identification thresholds for every NER and non-NER;
9) Store the above learning results.

4 Relation Identification

Our approach to NER identification is based on PNCBL, it can utilize the outcome of learning for further identifying NERs and removing non-NERs.

4.1 Optimal Identification Tradeoff

During the NER identification, the GCFs of NER candidates match those of all of the same kind of NERs in the NER pattern library. Likewise, the ICFs of NER candidates compare to those of non-NERs in the non-NER pattern library. The computing formulas in this procedure are listed as follows:

$$\mathrm{Sim}\,(R(i)_{can}, R(i)_{j1}) = \sum_{k1=1}^{\mathrm{Sum}(GCF)_i} \{\, w_i\,(GCF_{k1}) * \mathrm{Sim}\,(R(i)_{can}, R(i)_{j1})\,(GCF_{k1}) \,\} \quad (8)$$

and

$$\mathrm{Sim}\,(R(i)_{can}, NR(i)_{j2}) = \sum_{k2=1}^{\mathrm{Sum}(ICF)_i} \{\, w_i\,(ICF_{k2}) * \mathrm{Sim}\,(R(i)_{can}, NR(i)_{j2})\,(ICF_{k2}) \,\} \quad (9)$$

where R(i) represents the NER_i, and NR(i) expresses the non-NER_i, $1 \le i \le 14$. $R(i)_{can}$ is defined as a NER_i candidate. $R(i)_{j1}$ and $NR(i)_{j2}$ are the j1-th NER_i in the NER pattern library and the j2-th non-NER_i in the non-NER pattern library. $1 \le j1 \le$ Sum (R(i)) and $1 \le j2 \le$ Sum (NR(i)). Sum (R(i)) and Sum (NR(i)) are the total number of R(i) in the NER pattern library and that of NR(i) in non-NER pattern library respectively. w_i (GCF_{k1}) and w_i (ICF_{k2}) mean the weight of the k1-th GCF for the NER_i and that of the k2-th ICF for the non-NER_i. Sum $(GCF)_i$ and Sum $(ICF)_i$ are the total number of GCF for NER_i and that of ICF for non-NER_i separately.

In matching results, we find that sometimes the similarity values of a number of NERs or non-NERs matched with NER candidates are all more than the identification threshold. Thus, we have to utilize a voting method to achieve an identification tradeoff in our approach. For an optimal tradeoff, we consider the final identification performance in two aspects: i.e., recall and precision. In order to enhance recall, as many correct NERs should be captured as possible; on the other hand, in order to increase precision, misidentified non-NERs should be removed as accurately as possible.

The voting refers to the similarity calculation results between an NER candidate and NER / non-NER patterns. It pays special attention to circumstances in which both results are very close. If this happens, it exploits multiple calculation results to measure and arrive at a final decision. Additionally, notice that the impact of non-NER patterns is to restrict possible misidentified non-NERs. On the other hand, the voting assigns different thresholds to different NER candidates (e.g. HT_VT, WT_LT, and DT_DT or other NERs). Because the former three NERs have the same kind of NEs, the identification for these NERs is more difficult than for others. Thus, when voting, the corresponding threshold should be set more strictly.

4.2 Resolving NER Conflicts

In fact, although the voting is able to use similarity computing results for yielding an optimal tradeoff, there still remain some problems to be resolved. The relation conflict is one of the problems, which means that contradictory NERs occur in identification results. For example:

(i) The same kind of relations with different argument position: e.g., the relations HT_VT,

> HT_VT(ne1, no1; ne2, no2) and HT_VT(ne2, no2; ne1, no1) occur in an identification result at the same time.

(ii) The different kinds of relations with same or different argument positions: e.g., the relations WT_LT and DT_DT,

> WT_LT(ne1, no1; ne2, no2) and DT_DT(ne1, no1; ne2, no2) appear simultaneously in an identification result.

The reason for a relation conflict lies in the simultaneous and successful matching of a pair of NER candidates whose NEs are the same kind. They do not compare and distinguish themselves further. Considering the impact of NER and non-NER patterns, we organize the conditions to remove one of the relations, which has lower average similarity value with NER patterns or higher average similarity value with non-NER patterns.

4.3 Inferring Missing NERs

Due to a variety of reasons, some relations that should appear in an identification result may be missing. However, we can utilize some of the identified NERs to infer them. Of course, the prerequisite of the inference is that we suppose identified NERs are correct and non-contradictory. For all identified NERs, we should first examine whether they contain missing NERs. After determining the type of missing NERs, we may infer them - containing the relation name and its arguments. For instance, in an identification result, two NERs are:

PS_ID (ne1, no1; ne2, no2) and PS_TM (ne1, no1; ne3, no3)

In the above NER expressions, ne1 is a personal name, ne2 is a personal identity, and ne3 is a team name, because if a person occupies a position, i.e., he / she has a corresponding identity in a sports team, that means the position or identity belongs to this sports team. Accordingly, we can infer the following NER:

ID_TM (ne2, no2; ne3, no3)

5 Experimental Results and Evaluation

The main resources used for learning and identification are NER and non-NER patterns. Before learning, the texts from the Jie Fang Daily[2] in 2001 were annotated based on the NE identification. During learning, both pattern libraries are established in terms of the annotated texts and Lexical Sports Ontology. They have 142 (534 NERs) and 98 (572 non-NERs) sentence groups, respectively.

To test the performance of our approach, we randomly choose 32 sentence groups from the Jie Fang Daily in 2002, which embody 117 different NER candidates.

For evaluating the effects of negative cases, we made two experiments. Table 3 shows the average and total average recall, precision, and F-measure for the identification of 14 NERs only by positive case-based learning. Table 4 demonstrates those by PNCBL. Comparing the experimental results, among 14 NERs, the F-measure values of the seven NERs (PS_ID, ID_TM, CP_TI, WT_LT, PS_CP, CP_DA, and DT_DT) in Table 4 are higher than those of corresponding NERs in Table 3; the F-measure values of three NERs (LOC_CPC, TM_CP, and PS_CP) have no variation; but the F-measure values of other four NERs (PS_TM, CP_LOC, TM_CPC, and HT_VT) in Table 4 are lower than those of corresponding NERs in Table 3. This shows the performances for half of NERs are improved due to the adoption of both positive and negative cases. Moreover, the total average F-measure is enhanced from 63.61% to 70.46% as a whole.

Relation Type	Average Recall	Average Precision	Average F-measure
LOC_CPC	100	91.67	95.65
TM_CP	100	87.50	93.33
PS_ID	100	84.62	91.67
PS_TM	100	72.73	84.21
CP_LOC	88.89	69.70	78.13
ID_TM	90.91	66.67	76.93
CP_TI	83.33	71.43	76.92
PS_CP	60	75	66.67
TM_CPC	100	42.50	59.65
HT_VT	71.43	38.46	50
WT_LT	80	30.77	44.45
PS_CPC	33.33	66.67	44.44
CP_DA	0	0	0
DT_DT	0	0	0
Total Ave.	**71.99**	**56.98**	**63.61**

Table 3: Identification Performance for 14 NERs only by Positive Case-Based Learning

Relation Type	Average Recall	Average Precision	Average F-measure
LOC_CPC	100	91.67	95.65
TM_CP	100	87.50	93.33
CP_TI	100	75	85.71
PS_CPC	100	68.75	81.48
ID_TM	90.91	68.19	77.93
PS_ID	72.22	81.67	76.65
CP_LOC	88.89	66.67	76.19
PS_TM	80	65	71.72
CP_DA	100	50	66.67
DT_DT	66.67	66.67	66.67
PS_CP	60	75	66.67
WT_LT	60	37.50	46.15
HT_VT	42.86	30	35.30
TM_CPC	37.50	31.25	34.09
Total Ave.	**78.50**	**63.92**	**70.46**

Table 4: Identification Performance for 14 NERs by PNCBL

Finally, we have to acknowledge that it is difficult to compare the performance of our method to others because the experimental conditions and corpus domains of other NER identification efforts are quite different from ours. Nevertheless, we would like to use the performance of Chinese NER identification using memory-based learning (MBL) (Zhang and Zhou, 2000) for a comparison with our approach in Table 5. In the table, we select similar NERs in our domain to correspond to the three types of the relations (*employee-of*, *product-of*, and *location-of*). From the table we can deduce that the

[2] This is a local newspaper in Shanghai, China.

identification performance of relations for PNCBL is roughly comparable to that of the MBL.

Method	Relation Type	Recall	Precision	F-measure
MBL&I	employee-of	75.60	92.30	83.12
	product-of	56.20	87.10	68.32
	location-of	67.20	75.60	71.15
PNCBL&I	PS_TM	80	65	71.72
	PS_CP	60	75	66.67
	PS_ID	72.22	81.67	76.65
	ID_TM	90.91	68.19	77.93
	TM_CP	100	87.50	93.33
	CP_LOC	88.89	66.67	76.19
	PS_CPC	100	68.75	81.48
	TM_CPC	37.50	31.25	34.09

Table 5: Performances for Relation Identification (PNCBL&I vs. MBL&I)

6 Conclusion

In this paper, we propose a novel machine learning and identification approach PNCBL&I. This approach exhibits the following advantages: (i) The defined negative cases are used to improve the NER identification performance as compared to only using positive cases; (ii) All of the tasks, building of NER and non-NER patterns, feature selection, feature weighting and identification threshold determination, are automatically completed. It is able to adapt the variation of NER and non-NER pattern library; (iii) The information provided by the relation features deals with multiple linguistic levels, depicts both NER and non-NER patterns, as well as satisfies the requirement of Chinese language processing; (iv) Self-similarity is a reasonable measure for the concentrative degree of the same kind of NERs or non-NERs, which can be used to select general-character and individual-character features for NERs and non-NERs respectively; (v) The strategies used for achieving an optimal NER identification tradeoff, resolving NER conflicts, and inferring missing NERs can further improve the performance for NER identification; (vi) It can be applied to sentence groups containing multiple sentences. Thus identified NERs are allowed to cross sentences boundaries.

The experimental results have shown that the method is appropriate and effective for improving the identification performance of NERs in Chinese.

Acknowledgement

This work is a part of the COLLATE project under contract no. 01INA01B, which is supported by the German Ministry for Education and Research.

References

C. Cardie. 1996. *Automating Feature Set Selection for Case-Based Learning of Linguistic Knowledge*. In Proc. of the Conference on Empirical Methods in Natural Language Processing. University of Pennsylvania, Philadelphia, USA.

W. Daelemans. 1995. *Memory-based lexical acquisition and processing*. In P. Steffens, editor, Machine Translations and the Lexicon, Lecture Notes in Artificial Intelligence, pages 85-98. Springer Verlag. Berlin, Germany.

W. Daelemans, A. Bosch, J. Zavrel, K. Van der Sloot, and A. Vanden Bosch. 2000. *TiMBL: Tilburg Memory Based Learner, Version 3.0, Reference Guide*. Technical Report ILK-00-01, ILK, Tilburg University. Tilburg, The Netherlands. http://ilk.kub.nl/~ilk/papers/ilk0001.ps.gz.

H. Dang, C. Chia, M. Palmer and F. Chiou. 2002. *Simple Features for Chinese Word Sence Disambiguation*. In Proc. of the 19th International Conference on Computational Linguistics (COLING 2002), pages 204-210. Taipei, Taiwan.

Z. Dong and Q. Dong. 2000. *HowNet*. http://www.keenage.com/zhiwang/e_zhiwang.html.

N. Nilsson. 1996. *Introduction to Machine Learning: An Early Draft of a Proposed Textbook*. Pages 175-188. http://robotics.stanford.edu/people/nilsson/mlbook.html.

C. Stanfill and D. Waltz. 1986. *Toward memory-based reasoning*. Communications of the ACM, Vol.29, No.12, pages 1213-1228.

T. Yao, W. Ding and G. Erbach. 2003. *CHINERS: A Chinese Named Entity Recognition System for the Sports Domain*. In: Proc. of the Second SIGHAN Workshop on Chinese Language Processing (ACL 2003 Workshop), pages 55-62. Sapporo, Japan.

Y. Zhang and J. Zhou. 2000. *A trainable method for extracting Chinese entity names and their relations*. In Proc. of the Second Chinese Language Processing Workshop (ACL 2000 Workshop), pages 66-72. Hongkong, China.

Feature Engineering and Post-Processing for Temporal Expression Recognition Using Conditional Random Fields

Sisay Fissaha Adafre **Maarten de Rijke**
Informatics Institute, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
sfissaha,mdr@science.uva.nl

Abstract

We present the results of feature engineering and post-processing experiments conducted on a temporal expression recognition task. The former explores the use of different kinds of tagging schemes and of exploiting a list of core temporal expressions during training. The latter is concerned with the use of this list for post-processing the output of a system based on conditional random fields.

We find that the incorporation of knowledge sources both for training and post-processing improves recall, while the use of extended tagging schemes may help to offset the (mildly) negative impact on precision. Each of these approaches addresses a different aspect of the overall recognition performance. Taken separately, the impact on the overall performance is low, but by combining the approaches we achieve both high precision and high recall scores.

1 Introduction

Temporal expressions (*timex*es) are natural language phrases that refer directly to time points or intervals. They not only convey temporal information on their own but also serve as anchors for locating events referred to in a text. Timex recognition is a named entity recognition (NER) task to which a variety of natural language processing and machine learning techniques have been applied. As with other NER tasks, timex recognition is naturally viewed as a sequence labeling task, easily lending itself to machine learning techniques such as conditional random fields (CRFs) (Lafferty et al., 2001).

A preliminary experiment showed that, using CRFs, a respectable recognition performance can easily be achieved with a straightforward baseline system that is based on a simple tagging scheme and requires very little tuning, yielding F-scores around 0.78 (exact match) or even 0.90 (partial match). Interestingly, these high scores are mainly due to high or even very high precision scores, while recall leaves much to be improved.

The main focus of this paper is on boosting recall while maintaining precision at an acceptable (i.e., high) level. We report on two types of experiments aimed at achieving this goal. One type concerns feature engineering and the other concerns post-processing the output of a machine learner. While we do exploit the special nature of timexes, for portability reasons we avoid using task-specific and richer linguistic features (POS, chunks, etc.). Instead, we focus on features and techniques that can readily be applied to other NER tasks.

Specifically, our feature engineering experiments have two facets. The first concerns identification of a set of simple features that results in high generalization ability (accuracy). Here, particular emphasis will be placed on the use of a list of core timexes as a feature. The assumption is that the performance of data-driven approaches for timex recognition can be improved by taking into account the peculiar properties of timexes. Timexes exhibit various patterns, ranging from regular patterns that can easily be captured using simple regular expressions to complex linguistic forms (phrases). While timexes are real-

Proceedings of the ACL Workshop on Feature Engineering for Machine Learning in NLP, pages 9–16,
Ann Arbor, June 2005.

ized in different phrase types, the core lexical items of timexes are restricted. This suggests that a list of core timexes can easily be compiled and used in machine learning-based timex recognition. One approach of integrating such a list is using them to generate features, but the availability of such a list also opens up other possibilities in feature design that we present in later sections.

The second aspect concerns the tagging scheme. As in most NER experiments, the task of recognizing timexes is reduced to tagging. Commonly used tagging schemes are Inside-Outside (IO) and Begin-Continue-End-Unique-Negative (BCEUN) (Borthwick et al., 1998). The IO tagging scheme, which we use as a baseline, assigns the tag I to a token if it is part of a timex and O otherwise. The richer BCEUN scheme assigns the five tags B, C, E, U, and N to tokens depending on whether the token is single-token timex (U), a non-timex (N), appears at the beginning (B), at the end (E) or inside a timex boundary (C). In this paper, we compare the IO, BCEUN and an extended form of the BCEUN tagging scheme. The extended scheme adds two tags, PRE and POST, to the BCEUN scheme, which correspond to tokens appearing to the left and to the right of a timex.

In contrast, our post-processing experiments investigate the application of the list of core timexes for filtering the output of a machine learner. The incorporation into the recognition process of explicit knowledge in the form of a list for post-processing requires a carefully designed strategy to ensure that the important properties of the trained model are kept intact as much as possible while at the same-time improving overall results. We present an approach for using a list for post-processing that exploits the knowledge embodied in the trained model.

The paper is organized as follows. In Section 2 we provide background material, both on the timex extraction task (§2.1) and on the machine learning techniques on which we build in this paper, conditional random fields (§2.2). Our ideas on engineering feature sets and tagging schemes are presented in Section 3, while we describe our method for exploiting the explicit knowledge contained in a list in Section 4. In Section 5, we describe the experimental setup and present the results of our experiments. Related work is briefly reviewed in Section 6, and we conclude in Section 7.

2 Background

2.1 Task Description

In recent years, temporal aspects of information access have received increasing amounts of attention, especially as it relates to news documents. In addition to factual content, news documents have a temporal context, reporting events that happened, are happening, or will happen in relation to the publication date. Temporal document *retrieval* concerns the inclusion of both the document publication date and the in-text temporal expressions in the retrieval model (Kalczynski and Chou, 2005). The task in which we are interested in this paper is identifying the latter type of expressions, i.e., *extraction* of temporal expressions. TERN, the Temporal Expression Recognition and Normalization Evaluation, is organized under the auspices of the Automatic Content Extraction program (ACE, `http://www.nist.gov/speech/tests/ace/`). The TERN evaluation provides specific guidelines for the identification and normalization of timexes, as well as tagged corpora for training and testing and evaluation software. These guidelines and resources were used for the experiments described below.

The TERN evaluation consisted of two distinct tasks: recognition and normalization. Timex recognition involves correctly detecting and delimiting timexes in text. Normalization involves assigning recognized timexes a fully qualified temporal value. Our focus in this paper is on the recognition task; it is defined, for human annotators, in the TIDES TIMEX2 annotation guidelines (Ferro et al., 2004). The recognition task is performed with respect to corpora of transcribed broadcast news speech and news wire texts from ACE 2002, ACE 2003, and ACE 2004, marked up in SGML format and, for the training set, hand-annotated for TIMEX2s. An official scorer that evaluates the recognition performance is provided as part of the TERN evaluation. It computes precision, recall, and F-measure both for TIMEX2 tags (i.e., for *overlap* with a gold standard TIMEX2 element) and for *extent* of TIMEX2 elements (i.e., exact match of entire timexes).

2.2 Conditional Random Fields

We view the recognition of timexes task as a sequence labeling task in which each token in the text

is classified as being either a timex or not. One machine learning technique that has recently been introduced to tackle the problem of labeling and segmenting sequence data is conditional random fields (CRFs, (Lafferty et al., 2001)). CRFs are conditional probability distributions that take the form of exponential models. The special case of linear chain CRFs, which takes the following form, has been widely used for sequence labeling tasks:

$$P(y \mid x) = \frac{1}{Z(x)} \exp\left(\sum_{t=1}\sum_{k} \lambda_k f_k(t, y_{t-1}, y_t, x)\right),$$

where $Z(x)$ is the normalization factor, $X = \{x_1, \ldots, x_n\}$ is the observation sequence, $Y = \{y_1, \ldots, y_T\}$ is the label sequences, f_k and λ_k are the feature functions and their weights respectively. An important property of these models is that probabilities are computed based on a set of feature functions, i.e., f_k (usually binary valued), which are defined on both the observation X and label sequences Y. These feature functions describe different aspect of the data and may overlap, providing a flexible way of describing the task.

CRFs have been shown to perform well in a number of natural language processing applications, such as POS tagging (Lafferty et al., 2001), shallow parsing or NP chunking (Sha and Pereira, 2003), and named entity recognition (McCallum and Li, 2003). In this paper, CRFs are applied to the recognition of timexes; in our experiments we used the minorThird implementation of CRFs (Cohen, 2004).

3 Feature Engineering

The success of applying CRFs depends on the quality of the set of features used and the tagging scheme chosen. Below, we discuss these two aspects in greater detail.

3.1 Feature sets

Our baseline feature set consists of simple lexical and character features. These features are derived from a context window of two words (left and right). Specifically, the features are the lowercase form of all the tokens in the span, with each token contributing a separate feature, and the tokens in the left and right context window constitute another set of features. These feature sets capture the lexical content and context of timexes. Additionally, character type pattern features (such as capitalization, digit sequence) of tokens in the timexes are used to capture the character patterns exhibited by some of the tokens in temporal expressions. These features constitute the *basic feature* set.

Another important feature is the list of core timexes. The list is obtained by first extracting the phrases with -TMP function tags from the PennTree bank, and taking the words in these phrases (Marcus et al., 1993). The resulting list is filtered for stopwords. Among others, the list of core timexes consists of the names of days of the week and months, temporal units 'day,' 'month,' 'year,' etc. This list is used to generate binary features. In addition, the list is used to guide the design of other complex features that may involve one or more of token-tag pairs in the context of the current token. One way of using the list for this purpose is to generate a feature that involves bi-grams tokens. In certain cases, information extracted from bi-grams, e.g. +Xx 99 (May 20), can be more informative than information generated from individual tokens. We refer to these features as the *list* feature set.

3.2 Tagging schemes

A second aspect of feature engineering that we consider in this paper concerns different tagging schemes. As mentioned previously, the task of recognizing timexes is reduced to a sequence-labeling task. We compare three tagging schemes, IO (our baseline), BCEUN, and BCEUN+PRE&POST. While the first two are relatively standard, the last one is an extension of the BCEUN scheme. The intuition underlying this tagging scheme is that the most relevant features for timex recognition are extracted from the immediate context of the timex, e.g., the word 'During' in (1) below.

(1) During <TIMEX2>the past week</TIMEX2>, the storm has pounded the city.
During-PRE the-B past-C week-E ,-POST the storm has pounded the city.

Therefore, instead of treating these elements uniformly as outside (N), which ignores their relative importance, we conjecture that it is worthwhile to

assign them a special category, like PRE and POST corresponding to the tokens immediately preceding and following a timex, and that this leads to improved results.

4 Post-processing Using a List

In this section, we describe the proposed method for incorporating a list of core lexical timexes for post-processing the output of a machine learner. As we will see below, although the baseline system (with the IO tagging scheme and the basic feature set) achieves a high accuracy, the recall scores leave much to be desired. One important problem that we have identified is that timexes headed by core lexical items on the list may be missed. This is either due to the fact that some of these lexical items are semantically ambiguous and appear in a non-temporal sense, or the training material does not cover the particular context. In such cases, a reliable list of core timexes can be used to identify the missing timexes. For the purposes of this paper, we have created a list containing mainly headwords of timexes. These words are called *trigger words* since they are good indicators of the presence of temporal expressions.

How can we use trigger words? Before describing our method in some detail, we briefly describe a more naive (and problematic) approach. Observe that trigger words usually appear in a text along with their complements or adjuncts. As a result, picking only these words will usually contribute to token recall but span precision is likely to drop. Furthermore, there is no principled way of deciding which one to pick (semantically ambiguous elements will also be picked). Let's make this more precise. The aim is to take into account the knowledge acquired by the trained model and to search for the next optimal sequence of tags, which assigns the missed timex a non-negative tag. However, searching for this sequence by taking the whole word sequence is impractical since the number of possible tag sequences (number of all possible paths in a viterbi search) is very large. But if one limits the search to a window of size n ($n < 6$), sequential search will be feasible. The method, then, works on the output of the system. We illustrate the method by using the example given in (2) below.

(2) The chairman arrived in the city yesterday, and will leave next week. The press conference will be held tomorrow afternoon.

Now, assume that (2) is a test instance (a two-sentence document), and that the system returns the following best sequence (3). For readability, the tag N is not shown on the words that are assigned negative tags in all the examples below.

(3) The chairman arrived in the city yesterday-U , and will leave next week . The press conference will be held tomorrow-B afternoon-E .

According to (3), the system recognizes only 'yesterday' and 'tomorrow afternoon' but misses 'next week'. Assuming our list of timexes contains the word 'week', it tells us that there is a missing temporal expression, headed by 'week.' The naive method is to go through the above output sequence and change the token-tag pair 'week-N' to 'week-U'. This procedure recognizes the token 'week' as a valid temporal expression, but this is not correct: the valid temporal expression is 'next week'.

We now describe a second approach to incorporating the knowledge contained in a list of core lexical timexes as a post-processing device. To illustrate our ideas, take the complete sequence in (3) and extract the following segment, which is a window of 7 tokens centered at 'week'.

(4) ...[will leave next week . The press] ...

We *reclassify* the tokens in (4) assuming the history contains the token 'and' (the token which appears to the left of this segment in the original sequence) and the associated parameters. Of course, the best sequence will still assign both 'next' and 'week' the N tag since the underlying parameters (feature sets and the associated weights) are the same as the ones in the system. However, since the word sequence in (4) is now short (contains only 7 words) we can maintain a list of all possible tag sequences for it and perform a sequential search for the next best sequence, which assigns the 'week' token a non-negative tag. Assume the new tag sequence looks as follows:

(5) ...[will leave next-B week-E . The press] ...

This tag sequence will then be placed back into the original sequence resulting in (6):

(6) The chairman arrived in the city yesterday-U , and will leave next-B week-E . The press conference will be held tomorrow-B afternoon-E .

In this case, all the temporal expressions will be extracted since the token sequence 'next week' is properly tagged. Of course, the above procedure can also return other, invalid sequences as in (7):

(7) a. . . . will leave next-B week-C . The press . . .
b. . . . will leave next week-C . The press . . .
c. . . . will leave next week-C .-E The press . . .

The final extraction step will not return any timex since none of the candidate sequences in (7) contains a valid tag sequence. The assumption here is that of all the tag sequences, which assign the token 'week' a non-negative tag, those tag sequences which contain the segment 'next-B week-E' are likely to receive a higher weight since the underlying system is trained to recognize temporal expressions and the phrase 'next week' is a likely temporal expression.

This way, we hypothesize, it is possible to exploit the knowledge embodied in the trained model. As pointed out previously, simply going through the list and picking only head words like 'week' will not guarantee that the extracted tokens form a valid temporal expression. On the other hand, the above heuristics, which relies on the trained model, is likely to pick the adjunct 'next'.

The post-processing method we have just outlined boils down to reclassifying a small segment of a complete sequence using the same parameters (feature sets and associated weights) as the original model, and keeping all possible candidate sequences and searching through them to find a valid sequence.

5 Experimental Evaluation

In this section we provide an experimental assessment of the feature engineering and post-processing methods introduced in Sections 3 and 4. Specifically, we want to determine what their impact is on the precision and recall scores of the baseline system, and how they can be combined to boost recall while keeping precision at an acceptable level.

5.1 Experimental data

The training data consists of 511 files, and the test data consists of 192 files; these files were made available in the 2004 Temporal Expression Recognition and Normalization Evaluation. The temporal expressions in the training files are marked with XML tags. The minorThird system takes care of automatically converting from XML format to the corresponding tagging schemes. A temporal expression enclosed by `<TIMEX2>` tags constitutes a span. The features in the training instances are generated by looking at the surface forms of the tokens in the spans and their surrounding contexts.

5.2 Experimental results

Richer feature sets Table 1 lists the results of the first part of our experiments. Specifically, for every tagging scheme, there are two sets of features, *basic* and *list*. The results are based on both exact-match and partial match between the spans in the gold standard and the spans in the output of the systems, as explained in Subsection 2.1. In both the exact and partial match criteria, the addition of the list features leads to an improvement in recall, and no change or a decrease in precision.

In sum, the feature addition helps recall more than it hurts precision, as the F score goes up nearly everywhere, except for the exact-match/baseline pair.

Tagging schemes In Table 1 we also list the extraction scores for the tagging schemes we consider, IO, BCEUN, and BCEUN+PRE&POST, as described in Section 3.2.

Let us first look at the impact of the different tagging schemes in combination with the basic feature set (rows 3, 5, 7). As we go from the baseline tagging scheme IO to the more complex BCEUN and BCEUN+PRE&POS, precision increases on the exact-match criterion but remains almost the same on the partial match criterion. Recall, on the other hand, does not show the same trend. BCEUN has the highest recall values followed by BCEUN+PRE&POST and finally IO. In general, IO based tagging seems to perform worse whereas BCEUN based tagging scores slightly above its extended tagging scheme BCEUN+PRE&POST.

Next, considering the combination of extending the feature set and moving to a richer tagging scheme (rows 4, 6, 8), we have very much the same pattern. In both the exact match and the partial match setting, BCEUN tops (or almost tops) the two

		Exact Match			Partial Match		
Tagging scheme	Features	Prec.	Rec.	F	Prec.	Rec.	F
IO (baseline)	basic	0.846	0.723	0.780	0.973	0.832	0.897
	basic + list	0.822	0.736	0.776	0.963	0.862	0.910
BCEUN	basic	0.874	0.768	0.817	0.974	0.856	0.911
	basic + list	0.872	**0.794**	**0.831**	0.974	**0.887**	**0.928**
BCEUN+PRE&POS	basic	**0.882**	0.749	0.810	**0.979**	0.831	0.899
	basic + list	0.869	0.785	0.825	0.975	0.881	0.925

Table 1: Timex: Results of training on basic and list features, and different tagging schemes. Highest scores (Precision, Recall, F-measure) are in bold face.

other schemes in both precision and recall.

In sum, the richer tagging schemes function as precision enhancing devices. The effect is clearly visible for the exact-match setting, but less so for partial matching. It is not the case that the learner trained on the richest tagging scheme outperforms all learners trained with poorer schemes.

Post-processing Table 2 shows the results of applying the post-processing method described in Section 4. One general pattern we observe in Table 2 is that the addition of the list features improves precision for IO and BCEUN tagging scheme and shows a minor reduction in precision for BCEUN+PRE&POS tagging scheme in both matching criteria. Similarly, in the presence of post-processing, the use of a more complex tagging scheme results in a better precision. On the other hand, recall shows a different pattern. The addition of list features improves recall both for BCEUN and BCEUN+PRE&POS, but hurts recall for the IO scheme for both matching criteria.

Comparing the results in Table 1 and Table 2, we see that post-processing is a recall enhancing device since all the recall values in Table 2 are higher than the recall values in Table 1. Precision values in Table 2, on the other hand, are lower than those of Table 1. Importantly, the use of a more complex tagging scheme such as BCEUN+PRE&POS, allows us to minimize the drop in precision. In general, the best result (on partial match) in Table 1 is achieved through the combination of BCEUN and basic&list features whereas the best result in Table 2 is achieved by the combination of BCEUN+PRE&POS and basic &list features. Although both have the same overall scores on the exact match criteria, the latter performs better on partial match criteria. This, in turn, shows that the combination of post-processing, and BCEUN+PRE&POS achieves better results.

Stepping back We have seen that the extended tagging scheme and the post-processing methods improve on different aspects of the overall performance. As mentioned previously, the extended tagging scheme is both recall and precision-oriented, while the post-processing method is primarily recall-oriented. Combining these two methods results in a system which maintains both these properties and achieves a better overall result. In order to see how these two methods complement each other it is sufficient to look at the highest scores for both precision and recall. The extended tagging scheme with basic features achieves the highest precision but has relatively low recall. On the other hand, the simplest form, the IO tagging scheme and basic features with post-processing, achieves the highest recall and the lowest precision in partial match. This shows that the IO tagging scheme with basic features imposes a minimal amount of constraints, which allows for most of the timexes in the list to be extracted. Put differently, it does not discriminate well between the valid vs invalid occurrences of timexes from the list in the text. At the other extreme, the extended tagging scheme with 7 tags imposes strict criteria on the type of words that constitute a timex, thereby restricting which occurrences of the timex in the list count as valid timexes.

In general, although the overall gain in score is limited, our feature engineering and post-processing efforts reveal some interesting facts. First, they show one possible way of using a list for post-processing.

		Exact Match			Partial Match		
Tagging scheme	Features	Prec.	Rec.	F	Prec.	Rec.	F
IO	basic (baseline)	0.846	0.723	0.780	**0.973**	0.832	0.897
	basic	0.756	0.780	0.768	0.902	**0.931**	0.916
	basic + list	0.772	0.752	0.762	0.930	0.906	0.918
BCEUN	basic	0.827	0.789	0.808	0.945	0.901	0.922
	basic + list	0.847	0.801	0.823	0.958	0.906	0.931
BCEUN+PRE&POS	basic	0.863	0.765	0.811	**0.973**	0.863	0.915
	basic + list	**0.861**	**0.804**	**0.831**	0.970	0.906	**0.937**

Table 2: Timex: Results of applying post-processing on the systems in Table 1. The baseline (from Table 1) is repeated for ease of reference; it does not use post-processing. Highest scores (Precision, Recall, F-measure) are in bold face.

This method is especially appropriate for situations where better recall is important. It offers a means of controlling the loss in precision (gain in recall) by allowing a systematic process of recovering missing expressions that exploits the knowledge already embodied in a probabilistically trained model, thereby reducing the extent to which we have to make random decisions. The method is particularly sensitive to the criterion (the quality of the list in the current experiment) used for post-processing.

6 Related Work

A large number of publications deals with extraction of temporal expressions; the task is often treated as part of a more involved task combining recognition and normalization of timexes. As a result, many timex interpretation systems are a mixture of both rule-based and machine learning approaches (Mani and Wilson, 2000). This is partly due to the fact that timex recognition is more amenable to data-driven methods whereas normalization is best handled using primarily rule-based methods. We focused on machine learning methods for the timex recognition task only. See (Katz et al., 2005) for an overview of methods used for addressing the TERN 2004 task.

In many machine learning-based named-entity recognition tasks dictionaries are used for improving results. They are commonly used to generate binary features. Sarawagi and Cohen (2004) showed that semi-CRFs models for NE recognition perform better than conventional CRFs. One advantage of semi-CRFs models is that the units that will be tagged are segments which may contain one or more tokens, rather than single tokens as is done in conventional CRFs. This in turn allows one to incorporate segment based-features, e.g., segment length, and also facilitates integration of external dictionaries since segments are more likely to match the entries of an external dictionary than tokens. In this paper, we stuck to conventional CRFs, which are computationally less expensive, and introduced post-processing techniques, which takes into account knowledge embodied in the trained model.

Kristjannson et al. (2004) introduced constrained CRFs (CCRFs), a model which returns an optimal label sequence that fulfills a set of constraints imposed by the user. The model is meant to be used in an interactive information extraction environment, in which the system extracts structured information (fields) from a text and presents it to the user, and the user makes the necessary correction and submits it back to the system. These corrections constitute an additional set of constraints for CCRFs. CCRFs re-computes the optimal sequence by taking these constraints into account. The method is shown to reduce the number of user interactions required in validating the extracted information. In a very limited sense our approach is similar to this work. The list of core lexical timexes that we use represents the set of constraints on the output of the underlying system. However, our method differs in the way in which the constraints are implemented. In our case, we take a segment of the whole sequence that contains a missing timex, and reclassify the words in this segment while keeping all possible tag sequences sorted based on their weights. We then

search for the next optimal sequence that assigns the missing timex a non-negative tag sequentially. On the other hand, Kristjannson et al. (2004) take the whole sequence and recompute an optimal sequence that satisfies the given constraints. The constraints are a set of states which the resulting optimal sequence should include.

7 Conclusion

In this paper we presented different feature engineering and post-processing approaches for improving the results of timex recognition task. The first approach explores the different set of features that can be used for training a CRF-based timex recognition system. The second investigates the effect of the different tagging scheme for timex recognition task. The final approach we considered applies a list of core timexes for post-processing the output of a CRF system. Each of these approaches addresses different aspects of the overall performance. The use of a list of timexes both during training and for post-processing resulted in improved recall whereas the use of a more complex tagging scheme results in better precision. Their individual overall contribution to the recognition performances is limited or even negative whereas their combination resulted in substantial improvements over the baseline.

While we exploited the special nature of timexes, we did avoid using linguistic features (POS, chunks, etc.), and we did so for portability reasons. We focused exclusively on features and techniques that can readily be applied to other named entity recognition tasks. For instance, the basic and list features can also be used in NER tasks such as PERSON, LOCATION, etc. Moreover, the way that we have used a list of core expressions for post-processing is also task-independent, and it can easily be applied for other NER tasks.

Acknowledgments

Sisay Fissaha Adafre was supported by the Netherlands Organization for Scientific Research (NWO) under project number 220-80-001. Maarten de Rijke was supported by grants from NWO, under project numbers 365-20-005, 612.069.006, 220-80-001, 612.000.106, 612.000.207, 612.066.302, 264-70-050, and 017.001.190.

References

[Borthwick et al.1998] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. 1998. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Workshop on Very Large Corpora, ACL*.

[Cohen2004] W. Cohen. 2004. Methods for identifying names and ontological relations in text using heuristics for inducing regularities from data. `http://minorthird.sourceforge.net`.

[Ferro et al.2004] L. Ferro, L. Gerber, I. Mani, and G. Wilson, 2004. *TIDES 2003 Standard for the Annotation of Temporal Expressions*. MITRE, April.

[Kalczynski and Chou2005] P.J. Kalczynski and A. Chou. 2005. Temporal document retrieval model for business news archives. *Information Processing and Management*, 41:635–650.

[Katz et al.2005] G. Katz, J. Pustejovsky, and F. Schilder, editors. 2005. *Proceedings Dagstuhl Workshop on Annotating, Extracting, and Reasoning about Time and Events*.

[Kristjannson et al.2004] T. Kristjannson, A. Culotta, P. Viola, and A. McCallum. 2004. Interactive information extraction with constrained conditional random fields. In *Nineteenth National Conference on Artificial Intelligence, AAAI*.

[Lafferty et al.2001] J. Lafferty, F. Pereira, and A. McCallum. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*.

[Mani and Wilson2000] I. Mani and G. Wilson. 2000. Robust temporal processing of news. In *Proceedings of the 38th ACL*.

[Marcus et al.1993] M.P. Marcus, B. Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19:313–330.

[McCallum and Li2003] A. McCallum and W. Li. 2003. Early results for Named Entity Recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the 7th CoNLL*.

[Sarawagi and Cohen2004] S. Sarawagi and W.W. Cohen. 2004. Semi-markov conditional random fields for information extraction. In *NIPs (to appear)*.

[Sha and Pereira2003] F. Sha and F. Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of Human Language Technology-NAACL*.

Temporal Feature Modification for Retrospective Categorization

Robert Liebscher and **Richard K. Belew**
Department of Cognitive Science
University of California, San Diego
{rliebsch|rik}@cogsci.ucsd.edu

Abstract

We show that the intelligent use of one small piece of contextual information–a document's publication date–can improve the performance of classifiers trained on a text categorization task. We focus on academic research documents, where the date of publication undoubtedly has an effect on an author's choice of words. To exploit this contextual feature, we propose the technique of *temporal feature modification*, which takes various sources of lexical change into account, including changes in term frequency, associative strength between terms and categories, and dynamic categorization systems. We present results of classification experiments using both full text papers and abstracts of conference proceedings, showing improved classification accuracy across the whole collection, with performance increases of greater than 40% when temporal features are exploited. The technique is fast, classifier-independent, and works well even when making only a few modifications.

1 Introduction

As they are normally conceived, many tasks relevant to Computational Linguistics (CL), such as text categorization, clustering, and information retrieval, ignore the *context* in which a document was written, focusing instead on the lexical *content* of the document. Numerous improvements have been made in such tasks when context is considered, for example the hyperlink or citation structure of a document collection (Cohn and Hofmann, 2001; Getoor et al., 2001). In this paper, we aim to show that the intelligent use of another dimension of context–a document's publication date–can improve the performance of classifiers trained on a text categorization task.

Traditional publications, such as academic papers and patents, have histories that span centuries. The World Wide Web is no longer a new frontier; over a decade of its contents have been archived (Kahle, 2005); Usenet and other electronic discussion boards have been around for several decades. These forums continue to increase their publication rates and show no signs of slowing. A cursory glance at any one of them at two different points in time can reveal widely varying content.

For a concrete example, we can ask, "What is Computational Linguistics *about*?" Some topics, such as machine translation, lie at the heart of the discipline and will always be of interest. Others are ephemeral or have reached theoretical upper bounds on performance. It is thus more appropriate to ask what CL is about at some point in time. Consider Table 1, which lists the top five unigrams that best distinguished the field at different six-year periods, as derived from the odds ratio measure (see Section 3.2) over the full text of the ACL proceedings.

1979-84	1985-90	1991-96	1997-02
system	phrase	discourse	word
natural	plan	tree	corpus
language	structure	algorithm	training
knowledge	logical	unification	model
database	interpret	plan	data

Table 1: ACL's most characteristic terms for four time periods.

While these changes are interesting in their own right for an historical linguist, we aim to show that they can also be exploited for practical purposes. We focus on a fairly homogeneous set of academic research documents, where the time of publication undoubtedly has an effect both on an author's choice of words and on a field's definition of underlying topical categories. A document must say something novel while building upon what has already been said. This dynamic generates a landscape of changing research language, where authors and disciplines constantly influence and alter the course of one another.

Proceedings of the ACL Workshop on Feature Engineering for Machine Learning in NLP, pages 17–23,
Ann Arbor, June 2005.

1.1 Motivations

Text Categorization (TC) systems are typically used to classify a stream of documents soon after they are produced, based upon a set of historical training data. It is common for some TC applications, such as topic tracking (see Section 5.2), to downweight older features, or the feature vectors of entire documents, while placing more emphasis on features that have recently shown increased importance through changes in frequency and discriminative ability.

Our task, which we call *retrospective categorization*, uses historical data in both the training and test sets. It is retrospective from the viewpoint of a current user browsing through previous writings that are categorized with respect to a "modern" interpretation. Our approach is motivated by three observations concerning lexical change over time, and our task is to modify features so that a text classifier can account for all three.

First, lexical changes can take place within a category. The text collections used in our experiments are from various conference proceedings of the Association of Computing Machinery, which uses a hierarchical classification system consisting of over 500 labels (see Section 2). As was suggested by the example of Table 1, even if classification labels remain constant over time, the terms that best characterize them can change to reflect evolving "meanings". We can expect that many of the terms most closely associated with a category like Computational Linguistics cannot be captured properly without explicitly addressing their temporal context.

Second, lexical changes can occur between categories. A term that is significant to one category can suddenly or gradually become of interest to another category. This is especially applicable in news corpora (see examples in Section 3), but also applies to academic research documents. Terminological "migrations" between topics in computer science, and across all of science, are common.

Third, any coherent document collection on a particular topic is sufficiently dynamic that, over time, its categorization system must be updated to reflect the changes in the world on which its texts are based. Although Computational Linguistics predates Artificial Intelligence (Kay, 2003), many now consider the former a subset of the latter. Within CL, technological and theoretical developments have continually altered the labels ascribed to particular works.

In the ACM's hierarchical Computing Classification System (see Section 2.1), several types of transformations are seen in the updates it received in 1983, 1987, 1991, and 1998.[1] In *bifurcations*, categories can be split apart. With *collapses*, categories that were formerly more fine-grained, but now do not receive much attention, can be combined. Finally, entirely new categories can be inserted into the hierarchy.

[1] http://acm.org/class/

2 Data

To make our experiments tractable and easily repeatable for different parameter combinations, we chose to train and test on two subsets of the ACM corpus. One subset consists of collections of abstracts from several different ACM conferences. The other includes the full text collection of documents from one conference.

2.1 The ACM hierarchy

All classifications were performed with respect to the ACM's Computing Classification System, 1998 version. This, the most recent version of the ACM-CCS, is a hierarchic classification scheme that potentially presents a wide range of hierarchic classification issues. Because the work reported here is focused on temporal aspects of text classification, we have adopted a strategy that effectively "flattens" the hierarchy. We interpret a document which has a *primary*[2] category at a narrow, low level in the hierarchy (e.g., `H.3.3.CLUSTERING`) as also classified at all broader, higher-level categories leading to the root (`H`, `H.3`, `H.3.3`). With this construction, the most refined categories will have fewer example documents, while broader categories will have more.

For each of the corpora considered, a threshold of 50 documents was set to guarantee a sufficient number of instances to train a classifier. Narrower branches of the full ACM-CCS tree were truncated if they contained insufficient numbers of examples, and these documents were associated with their parent nodes. For example, if H.3.3 contained 20 documents and H.3.4 contained 30, these would be "collapsed" into the H.3 category.

All of our corpora carry publication timestamp information involving time scales on the order of one to three decades. The field of computer science, not surprisingly, has been especially fortunate in that most of its publications have been recorded electronically. While obviously skewed relative to scientific and academic publishing more generally, we nevertheless find significant "micro-cultural" variation among the different special interest groups.

2.2 SIGIR full text

We have processed the annual proceedings of the Association for Computing Machinery's Special Interest Group in Information Retrieval (SIGIR) conference from its inception in 1978 to 2002. The collection contains over 1,000 documents, most of which are 6-10 page papers, though some are keynote addresses and 2-3 page poster

[2] Many ACM documents also are classified with additional "other" categories, but these were not used.

Corpus	Vocab size	No. docs	No. cats
SIGIR	16104	520	17
SIGCHI	4524	1910	20
SIGPLAN	6744	3123	22
DAC	6311	2707	20

Table 2: Corpus features

	Unlabeled	Expected
Proceedings	18.97%	7.73%
Periodicals	19.08%	11.54%
No. docs	24,567	8,703

Table 3: Missing classification labels in ACM

summaries. Every document is tagged with its year of publication. Unfortunately, only about half of the SIGIR documents bear category labels. The majority of these omissions fall within the 1978-1987 range, leaving us the remaining 15 years to work with.

2.3 Conference abstracts

We collected nearly 8,000 abstracts from the Special Interest Group in Programming Languages (SIGPLAN), the Special Interest Group in Computer-Human Interaction (SIGCHI) and the Design Automation Conference (DAC). Characteristics of these collections, and of the SIGIR texts, are shown in Table 2.

2.4 Missing labels in ACM

We derive the statistics below from the corpus of all documents published by the ACM between 1960 and 2003. The arguments can be applied to any corpus which has categorized documents, but for which there are classification gaps in the record.

The first column of Table 3 shows that nearly one fifth of all ACM documents, from both conference proceedings and periodicals, do not possess category labels. We define a document's label as "expected" when more than half of the other documents in its publication (one conference proceeding or one issue of a periodical) are labeled, and if there are more than ten total. The second column lists the percentage of documents where we expected a label but did not find one.

3 Methods

Text categorization (TC) is the problem of assigning documents to one or more pre-defined categories. As Section 1 demonstrated, the terms which best characterize a category can change through time, so it is not unreasonable to assume that intelligent use of temporal context will prove useful in TC.

Imagine the example of sorting several decades of articles from the *Los Angeles Times* into the categories `ENTERTAINMENT`, `BUSINESS`, `SPORTS`, `POLITICS`, and `WEATHER`. Suppose we come across the term `schwarzenegger` in a training document. In the 1970s, during his career as a professional bodybuilder, Arnold Schwarzenegger's name would be a strong indicator of a `SPORTS` document. During his film career in the 1980s-1990s, his name would be most likely to appear in an `ENTERTAINMENT` document. After 2003, at the outset of his term as California's governor, the `POLITICS` and `BUSINESS` categories would be the most likely candidates. We refer to `schwarzenegger` as a *temporally perturbed* term, because its distribution across categories varies greatly with time.

Documents containing temporally perturbed terms hold valuable information, but this is lost in a statistical analysis based purely on the average distribution of terms across categories, irrespective of temporal context. This information can be recovered with a technique we call *temporal feature modification* (TFM). We first outline a formal model for its use.

3.1 A term generator framework

One obvious way to introduce temporal information into the categorization task is to simply provide the year of publication as a new lexical feature. Preliminary experiments (not reported here) showed that this method had virtually no effect on classification performance. When the date features were "emphasized" with higher frequencies, classification performance declined.

Instead, we proceed from the perspective of a simplified *language generator* model (e.g. (Blei et al., 2003)). We imagine that the first step in the production of a document involves an author choosing a category `C`. Each term `k` (word, bigram, phrase, etc.) is accorded a unique generator G^k that determines the distribution of `k` across categories, and therefore its likelihood to appear in category `C`. The model assumes that all authors share the same generator for each term, and that the generators do not change over time. We are particularly interested in identifying temporally perturbed lexical generators that violate this assumption.

External events at time *t* can perturb the generator of `k`, causing $\Pr(C|k_t)$ to be different relative to the background $\Pr(C|k)$ computed over the entire corpus. If the perturbation is significant, we want to separate the instances of `k` at time *t* from all other instances.

Returning to our earlier example, we would treat a generic, atemporal occurrence of `schwarzenegger` and the *pseudo-term* "`schwarzenegger`*+2003*" as though they were actually *different* terms, because they were produced by two different generators. We hypothesize that separating the analysis of the two can improve

our estimates of the true Pr(C|k), both in 2003 and in other years.

3.2 TFM Procedure

The generator model motivates a procedure we outline below for flagging certain lexemes with explicit temporal information that distinguish them so as to contrast them with those generated by the underlying atemporal alternatives. This procedure makes use of the (log) odds ratio for feature selection:

$OddsRatio(C,k) = log(\frac{p_k(1-q_k)}{q_k(1-p_k)})$

where p is Pr(k|C), the probability that term k is present, given category C, and q is Pr(k|!C).

The odds ratio between a term and a category is a measure of the associated strength of the two, for it measures the likelihood that a term will occur frequently within a category and (relatively) infrequently outside. Odds ratio happens to perform very well in feature selection tests; see (Mladenic, 1998) for details on its use and variations. Ultimately, it is an arbitrary choice and could be replaced by any method that measures term-category strength.

The following pseudocode describes the process of temporal feature modification:

```
VOCABULARY ADDITIONS:
for each class C:
 for each time (year) t:
  PreModList(C,t,L) = OddsRatio(C,t,L)
  ModifyList(t) =
   DecisionRule(PreModList(C,t,L)
  for each term k in ModifyList(t):
   Add pseudo-term "k+t" to Vocab

DOCUMENT MODIFICATIONS:
for each document:
 t = time (year) of doc
 for each term k:
  if "k+t" in Vocab:
   Replace k with "k+t"
 Classify modified document
```

PreModList(C,*t*,L) is a list of the top L terms that, by the odds ratio measure, are highly associated with category C at time *t*. (In our case, time is divided annually, because this is the finest resolution we have for many of the documents in our corpus.) We test the hypothesis that these come from a perturbed generator at time *t*, as opposed to the atemporal generator G^k, by comparing the odds ratios of term-category pairs in a PreModList at time *t* with the same pairs across the entire corpus. Terms which pass this test are added to the final ModifyList(*t*) for time *t*. For the results that we report, DecisionRule is a simple ratio test with threshold factor *f*. Suppose *f* is 2.0: if the odds ratio between C and k is twice as great at time *t* as it is atemporally, the decision rule is "passed".

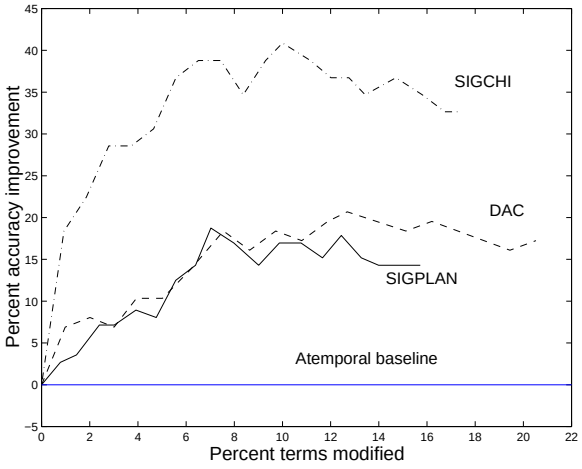

Figure 1: Improvement in categorization performance with TFM, using the best parameter combinations for each corpus.

The generator G^k is then considered perturbed at time *t* and k is added to ModifyList(*t*). In the training and testing phases, the documents are modified so that a term k is replaced with the pseudo-term "k+*t*" if it passed the ratio test.

3.3 Text categorization details

The TC parameters held constant in our experiments are: Stoplist, Porter stemming, and Laplacian smoothing. Other parameters were varied: four different classifiers, three unique minimum vocabulary frequencies, unigrams and bigrams, and four threshold factors *f*. 10-fold cross validation was used for parameter selection, and 10% of the corpus was held out for testing purposes. Both of these sets were distributed evenly across time.

4 Results

Table 4 shows the parameter combinations, chosen by ten-fold cross-validation, that exhibited the greatest increase in categorization performance for each corpus.

Using these parameters, Figure 1 shows the improvement in accuracy for different percentages of terms modified on the test sets. The average accuracies (across all parameter combinations) when no terms are modified are less than stellar, ranging from 26.70% (SIGCHI) to 37.50% (SIGPLAN), due to the difficulty of the task (20-22 similar categories; each document can only belong to one). Our aim here, however, is simply to show improvement. A baseline of 0.0 in the plot indicates accuracy without any temporal modifications.

Figure 2 shows the accuracy on an absolute scale when TFM is applied to the full text SIGIR corpus. Performance increased from the atemporal baseline of 28.85%

Corpus	Improvement	Classifier	n-gram size	Vocab frequency min.	Ratio threshold *f*
SIGIR	33.32%	Naive Bayes	Bigram	2	2.0
SIGCHI	40.82%	TF.IDF	Bigram	10	1.0
SIGPLAN	18.74%	KNN	Unigram	10	1.5
DAC	20.69%	KNN	Unigram	2	1.0

Table 4: Top parameter combinations for TFM by improvement in classification accuracy. *Vocab frequency min.* is the minimum number of times a term must appear in the corpus in order to be included.

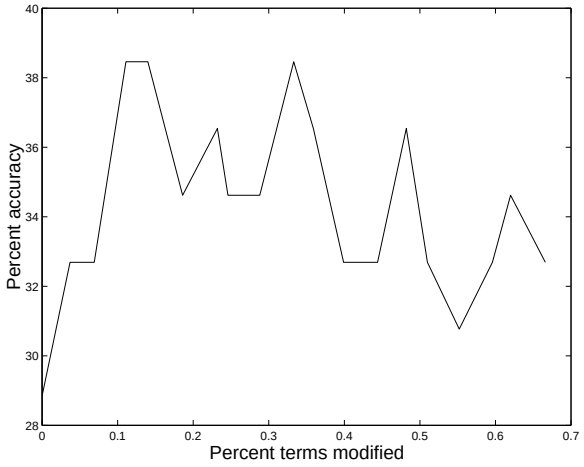

Figure 2: Absolute categorization performance with TFM for the SIGIR full text corpus.

correct to a maximum of 38.46% when only 1.11% of the terms were modified. The *ModifyLists* for each category and year averaged slightly fewer than two terms each.

In most cases, the technique performs best when making relatively few modifications: the left sides of each figure show a rapid performance increase, followed by a gradual decline as more terms are modified. After requiring the one-time computation of odds ratios in the training set for each category/year, TFM is very fast and requires negligible extra storage space. This is important when computing time is at a premium and enormous corpora such as the ACM full text collection are used. It is also useful for quickly testing potential enhancements to the process, some of which are discussed in Section 6.

The results indicate that L in PreModList(C,*t*,L) need not exceed single digits, and that performance asymptotes as the number of terms modified increases. As this happens, more infrequent terms are judged to have been produced by perturbed generators, thus making their true distributions difficult to compute (for the years in which they are not modified) due to an insufficient number of examples.

4.1 General description of results

A quantitative average of all results, using all parameter combinations, is not very meaningful, so we provide a qualitative description of the results not shown in Table 4 and Figures 1 and 2. Of the 96 different parameter combinations tested on four different corpora, 83.33% resulted in overall increases in performance. The greatest increase peaked at 40.82% improvement over baseline (atemporal) accuracy, while the greatest decrease dropped performance by only 8.31%.

5 Related Work

The use of metadata and other complementary (non-content) information to improve text categorization is an interesting and well-known problem. The specific use of temporal information, even if only implicitly, for tasks closely related to TC has been explored through adaptive information filtering (AIF) and topic detection and tracking (TDT).

5.1 Adaptive Information Filtering

There exists a large body of work on information filtering, which "is concerned with the problem of delivering useful information to a user while preventing an overload of irrelevant information" (Lam et al., 1996). Of particular interest here is *adaptive* information filtering (AIF), which handles the problems of concept *drift* (a gradual change in the data set a classifier must learn from) and concept *shift* (a more radical change).

Klinkenberg and Renz test eight different classifiers on their abilities to adapt to changing user preferences for news documents (Klinkenberg and Renz, 1998). They try different "data management techniques" for the concept drift scenario, selectively altering the size of the set of examples (the *adaptive window*) that a classifier trains on using a heuristic that accounts for the degree of dissimilarity between the current batch of examples and previous batches. Klinkenberg and Joachims later abandon this approach because it relies on "complicated heuristics", and instead concentrate their analysis on support vector machines (Klinkenberg and Joachims, 2000).

Stanley uses an innovative approach that eschews the need for an adaptive window of training examples, and

instead relies on a voting system for decision trees (Stanley, 2001). The weight of each classifier's vote (classification) is proportional to its record in predicting classifications for previous examples. He notes that this technique does not rely on decision trees; rather, any combination of classifiers can be inserted into the system.

The concept drift and shift scenarios used in the published literature are often unrealistic and not based upon actual user data. Topic Detection and Tracking, described in the following section, must work not with the behavior of one individual, but with texts that report on real external events and are not subject to artificial manipulation. This multifaceted, unsupervised character of TDT makes it a more appropriate precursor with which to compare our work.

5.2 Topic Detection and Tracking

Franz et al. note that Topic Detection and Tracking (TDT) is fundamentally different from AIF in that the "adaptive filtering task focuses on performance improvements driven by feedback from real-time human relevance assessments. TDT systems, on the other hand, are designed to run autonomously without human feedback" (Franz et al., 2001). Having roots in information retrieval, text categorization, and information filtering, the initial TDT studies used broadcast news transcripts and written news corpora to accomplish tasks ranging from news story clustering to boundary segmentation. Of most relevance to the present work is the *topic tracking* task. In this task, given a small number (1-4) of training stories known to be about a particular event, the system must make a binary decision about whether each story in an incoming stream is about that event.

Many TDT systems make use of temporal information, at least implicitly. Some employ a least recently used (Chen and Ku, 2002) or decay (Allan et al., 2002) function to restrict the lexicon available to the system at any given point in time to those terms most likely to be of use in the topic tracking task.

There are many projects with a foundation in TDT that go beyond the initial tasks and corpora. For example, TDT-inspired language modeling techniques have been used to train a system to make intelligent stock trades based upon temporal analysis of financial texts (Lavrenko et al., 2000). Retrospective *timeline* generation has also become popular, as exhibited by Google's *Zeitgeist* feature and browsers of TDT news corpora (Swan and Allan, 2000; Swan and Jensen, 2000).

The first five years of TDT research are nicely summarized by Allan (Allan, 2002).

6 Summary and Future Work

In this paper, we have demonstrated a feature modification technique that accounts for three kinds of lexical changes in a set of documents with category labels. Within a category, the distribution of terms can change to reflect the changing nature of the category. Terms can also "migrate" between categories. Finally, the categorization system itself can change, leading to necessary lexical changes in the categories that do not find themselves with altered labels. Temporal feature modification (TFM) accounts for these changes and improves performance on the retrospective categorization task as it is applied to subsets of the Association for Computing Machinery's document collection.

While the results presented in this paper indicate that TFM can improve classification accuracy, we would like to demonstrate that its mechanism truly incorporates *changes* in the lexical content of categories, such as those outlined in Section 1.1. A simple baseline comparison would pit TFM against a procedure in which the corpus is divided into slices temporally, and a classifier is trained and tested on each slice individually. Due to changes in community interest in certain topics, and in the structure of the hierarchy, some categories are heavily represented in certain (temporal) parts of the corpus and virtually absent elsewhere. Thus, the chance of finding every category represented in a single year is very low. For our corpora, this did not even occur once.

The "bare bones" version of TFM presented here is intended as a proof-of-concept. Many of the parameters and procedures can be set arbitrarily. For initial feature selection, we used odds ratio because it exhibits good performance in TC (Mladenic, 1998), but it could be replaced by another method such as information gain, mutual information, or simple term/category probabilities. The ratio test is not a very sophisticated way to choose which terms should be modified, and presently only detects the *surges* in the use of a term, while ignoring the (admittedly rare) declines.

In experiments on a Usenet corpus (not reported here) that was more balanced in terms of documents per category and per year, we found that allowing different terms to "compete" for modification was more effective than the egalitarian practice of choosing L terms from each category/year. There is no reason to believe that each category/year is equally likely to contribute temporally perturbed terms.

We would also like to exploit temporal *contiguity*. The present implementation treats time slices as independent entities, which precludes the possibility of discovering temporal trends in the data. One way to incorporate trends *implicitly* is to run a smoothing filter across the temporally aligned frequencies. Also, we treat each slice at annual resolution. Initial tests show that aggregating two or more years into one slice improves performance for some corpora, particularly those with temporally sparse data such as DAC.

Acknowledgements

Many thanks to the anonymous reviewers for their helpful comments and suggestions.

References

J. Allan, V. Lavrenko, and R. Swan. 2002. Explorations within topic tracking and detection. In J. Allan, editor, *Topic Detection and Tracking: Event-based Information Organization*, pages 197–224. Kluwer Academic Publishers.

J. Allan. 2002. Introduction to topic detection and tracking. In J. Allan, editor, *Topic Detection and Tracking: Event-based Information Organization*, pages 1–16. Kluwer Academic Publishers.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1002.

H. Chen and L. Ku. 2002. An nlp & ir approach to topic detection. In J. Allan, editor, *Topic Detection and Tracking: Event-based information organization*, pages 243–264. Kluwer Academic Publishers.

D. Cohn and T. Hofmann. 2001. The missing link: a probabilistic model of document content and hyperlink connectivity. In *Advances in Neural Information Processing Systems*, pages 430–436. MIT Press.

M. Franz, T. Ward, J.S. McCarley, and W. Zhu. 2001. Unsupervised and supervised clustering for topic tracking. In *Proceedings of the Special Interest Group in Information Retrieval*, pages 310–317.

L. Getoor, E. Segal, B. Taskar, and D. Koller. 2001. Probabilistic models of text and link structure for hypertext classification (2001). In *Proceedings of the 2001 IJCAI Workshop on Text Learning: Beyond Supervision*.

Brewster Kahle. 2005. The internet archive. http://www.archive.org/.

Martin Kay. 2003. Introduction. In Ruslan Mitkov, editor, *The Oxford Handbook of Computational Linguistics*, pages xvii–xx. Oxford University Press.

R. Klinkenberg and T. Joachims. 2000. Detecting concept drift with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, page 11. Morgan Kaufmann.

R. Klinkenberg and I. Renz. 1998. Adaptive information filtering: Learning in the presence of concept drifts. In *AAAI/ICML workshop on learning for text categorization*.

W. Lam, S. Mukhopadhyay, J. Mostafa, and M. Palakal. 1996. Detection of shifts in user interests for personalized information filtering. In *Proceedings of the Special Interest Group in Information Retrieval*, pages 317–326.

V. Lavrenko, M. Schmill, D. Lawrie, P. Ogilvie, D. Jensen, and J. Allan. 2000. Mining of concurrent text and time series. In *6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Text Mining Workshop*, pages 37–44.

D. Mladenic. 1998. *Machine Learning on non-homogeneous, distributed text data*. Ph.D. thesis, University of Ljubljana, Slovenia.

K.O. Stanley. 2001. Learning concept drift with a committee of decision trees. Computer Science Department, University of Texas-Austin.

R. Swan and J. Allan. 2000. Automatic generation of overview timelines. In *Proceedings of the Special Interest Group in Information Retrieval*, pages 47–55.

R. Swan and D. Jensen. 2000. Timemines: Constructing timelines with statistical models of word usage. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Using Semantic and Syntactic Graphs for Call Classification

Dilek Hakkani-Tür **Gokhan Tur**
AT&T Labs – Research
Florham Park, NJ, 07932
{dtur,gtur}@research.att.com

Ananlada Chotimongkol
Carnegie Mellon University
Pittsburgh, PA 15213
ananlada@cs.cmu.edu

Abstract

In this paper, we introduce a new data representation format for language processing, the syntactic and semantic graphs (SSGs), and show its use for call classification in spoken dialog systems. For each sentence or utterance, these graphs include lexical information (words), syntactic information (such as the part of speech tags of the words and the syntactic parse of the utterance), and semantic information (such as the named entities and semantic role labels). In our experiments, we used written language as the training data while computing SSGs and tested on spoken language. In spite of this mismatch, we have shown that this is a very promising approach for classifying complex examples, and by using SSGs it is possible to reduce the call classification error rate by 4.74% relative.

1 Introduction

Goal-oriented spoken dialog systems aim to identify intents of humans, expressed in natural language, and take actions accordingly to satisfy their requests. The intent of each speaker is identified using a natural language understanding component. This step can be seen as a multi-label, multi-class call classification problem for customer care applications (Gorin et al., 1997; Chu-Carroll and Carpenter, 1999; Gupta et al., To appear, among others). As an example, consider the utterance *I would like to know my account balance*, from a financial domain customer care application. Assuming that the utterance is recognized correctly by the automatic speech recognizer (ASR), the corresponding intent (call-type) would be *Request(Balance)* and the action would be telling the balance to the user after prompting for the account number or routing this call to the billing department.

Typically these application specific call-types are pre-designed and large amounts of utterances manually labeled with call-types are used for training call classification systems. For classification, generally word n-grams are used as features: In the *How May I Help You?*^{SM} (*HMIHY*) call routing system, selected word n-grams, namely *salient phrases*, which are salient to certain call-types play an important role (Gorin et al., 1997). For instance, for the above example, the salient phrase "account balance" is strongly associated with the call-type *Request(Balance)*. Instead of using salient phrases, one can leave the decision of determining useful features (word n-grams) to a classification algorithm used as described in (Di Fabbrizio et al., 2002) and (Gupta et al., To appear). An alternative would be using a vector space model for classification where call-types and utterances are represented as vectors including word n-grams (Chu-Carroll and Carpenter, 1999).

Call classification is similar to text categorization, except the following:

- The utterances are much shorter than typical documents used for text categorization (such as broadcast news or newspaper articles).

Proceedings of the ACL Workshop on Feature Engineering for Machine Learning in NLP, pages 24–31,
Ann Arbor, June 2005.

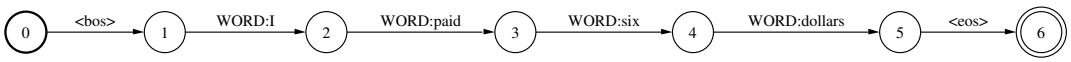

Figure 1: An example utterance represented as a single path FSM.

- Since it deals with spontaneous speech, the utterances frequently include disfluencies or are ungrammatical, and
- ASR output is very noisy, typically one out of every four words is misrecognized (Riccardi and Hakkani-Tür, 2003).

Even though the shortness of the utterances may imply the easiness of the call classification task, unfortunately this is not the case. The call classification error rates typically range between 15% to 30% depending on the application (Gupta et al., To appear). This is mainly due to the data sparseness problem because of the nature of the input. Even for simple call-types like *Request(Balance)*, there are many ways of uttering the same intent. For instance, in one of the applications we used in our experiments, as a response to the greeting prompt, there are 2,697 unique utterances out of 3,547 utterances for that call-type. Some examples include:

- *I would like to know my account balance*
- *How much do I owe you*
- *How much is my bill*
- *What is my current bill*
- *account balance*
- *You can help me by telling me what my phone bill is*
- ...

Given this data sparseness, current classification approaches require an extensive amount of labeled data in order to train a call classification system with a reasonable performance. In this paper, we present methods for extending the classifier's feature set by generalizing word sequences using syntactic and semantic information represented in compact graphs, called syntactic and semantic graphs (SSGs). For each sentence or utterance, these graphs include lexical information (words), syntactic information (such as the part of speech tags of the words and the syntactic parse of the utterance), and semantic information (such as the named entities and semantic role labels). The generalization is expected to help reduce the data sparseness problem by applying various groupings on word sequences. Furthermore, the classifier is provided with additional syntactic and semantic information which might be useful for the call classification task.

In the following section, we describe the syntactic and semantic graphs. In Section 3, we describe our approach for call classification using SSGs. In Section 4, we present the computation of syntactic and semantic information for SSGs. In the last Section, we present our experiments and results using a spoken dialog system AT&T VoiceTone® Spoken Dialog System (Gupta et al., To appear).

2 Semantic and Syntactic Graphs

Consider the typical case, where only lexical information, i.e. word n-grams are used for call classification. This is equivalent to representing the words in an utterance as a directed acyclic graph where the words are the labels of the transitions and then extracting the transition n-grams from it. Figure 1 shows the graph for the example sentence *I paid six dollars*, where <*bos*> and <*eos*> denote the beginning and end of the sentence, respectively.

Syntactic and semantic graphs are also directed acyclic graphs, formed by adding transitions encoding syntactic and semantic categories of words or word sequences to the word graph. The first additional information is the part of speech tags of the words. In the graph, as a parallel transition for each word of the utterance, the part of speech category of the word is added, as shown in Figure 2 for the example sentence. Note that, the word is prefixed by the token *WORD:* and the part-of-speech tag is prefixed by the token *POS:*, in order to distinguish between different types of transitions in the graph.

The other type of information that is encoded in these graphs is the syntactic parse of each utterance, namely the syntactic phrases with their head words. For example in the sentence *I paid six dollars*, *six dollars* is a noun phrase with the head word *dollars*. In Figure 2, the labels of the transitions for syntactic phrases are prefixed by the token *PHRASE:*. There-

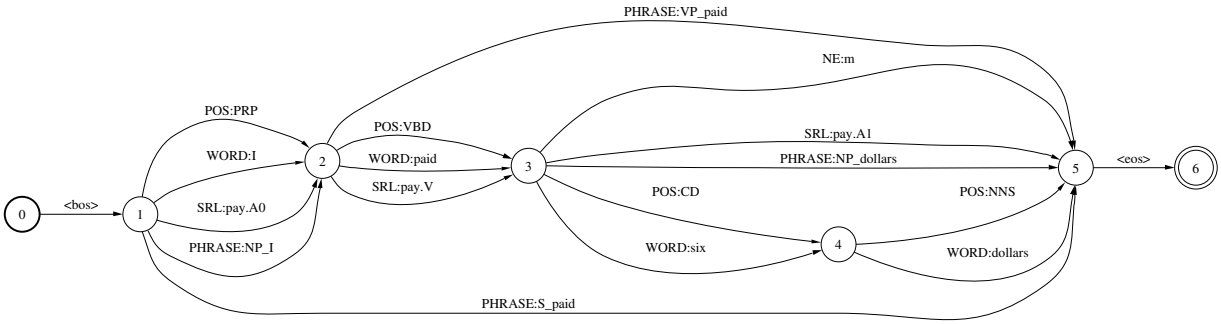

Figure 2: The SSG for the utterance *I paid six dollars*, where words (*WORD:*), part-of-speech tags (*POS:*), syntactic parse (*PHRASE:*), named entities (*NE:*) and semantic roles (*SRL:*) are included.

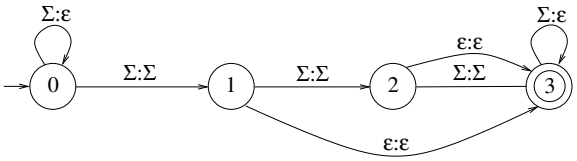

Figure 3: The FST used to extract unigram, bigram and trigrams. Σ represents the alphabet, ε represents the epsilon transition.

fore, *six dollars* is also represented by the transition labeled *PHRASE:NP_dollars*. As an alternative, one may drop the head word of the phrase from the representation, or insert an epsilon transition parallel to the transitions of the modifiers of the head word to eliminate them from some n-grams.

Generic named entity tags, such as person, location and organization names and task-dependent named entity tags, such as drug names in a medical domain, are also incorporated into the graph, where applicable. For instance, for the example sentence, *six dollars* is a monetary amount, so the arc *NE:m* is inserted parallel to that sequence.

As another source of semantic information, semantic role labels of the utterance components are incorporated to the SSGs. The semantic role labels represent the predicate/argument structure of each sentence: Given a predicate, the goal is to identify all of its arguments and their semantic roles. For example, in the example sentence the predicate is *pay*, the agent of this predicate is *I* and the amount is *six dollars*. In the graph, the labels of the transitions for semantic roles are prefixed by the token *SRL:* and the corresponding predicate. For example, the sequence *six dollars* is the amount of the predicate *pay*, and this is shown by the transition with label *SRL:pay.A1* following the PropBank notation (Kingsbury et al., 2002)[1].

In this work, we were only able to incorporate part-of-speech tags, syntactic parses, named entity tags and semantic role labels in the syntactic and semantic graphs. Insertion of further information such as supertags (Bangalore and Joshi, 1999) or word stems can also be beneficial for further processing.

3 Using SSGs for Call Classification

In this paper we propose extracting all n-grams from the SSGs to use them for call classification. The n-grams in an utterance SSG can be extracted by converting it to a finite state transducer (FST), F_i. Each transition of F_i has the labels of the arcs on the SSG as input and output symbols[2]. Composing this FST with another FST, F_N, representing all the possible n-grams, forms the FST, F_X, which includes all n-grams in the SSG:

$$F_X = F_i \circ F_N$$

[1] *A1* or *Arg1* indicates the object of the predicate, in this case the amount.

[2] Instead of the standard notation where ":" is used to separate the input and output symbols in finite state transducers, we use ":" to separate the type of the token and its value.

Then, extracting the n-grams in the SSG is equivalent to enumerating all paths of F_X. For $n = 3$, F_N is shown in Figure 3. The alphabet Σ contains all the symbols in F_i.

We expect the SSGs to help call classification because of the following reasons:

- First of all, the additional information is expected to provide some generalization, by allowing new n-grams to be encoded in the utterance graph since SSGs provide syntactic and semantic groupings. For example, the words *a* and *the* both have the part-of-speech tag category *DT* (determiner), or all the numbers are mapped to a cardinal number (*CD*), like the *six* in the example sentence. So the bigrams *WORD:six WORD:dollars* and *POS:CD WORD:dollars* will both be in the SSG. Similarly the sentences *I paid six dollars* and *I paid seventy five dollars and sixty five cents* will both have the trigram *WORD:I WORD:paid NE:m* in their SSGs.
- The head words of the syntactic phrases and predicate of the arguments are included in the SSGs. This enables the classifier to handle long distance dependencies better than using other simpler methods, such as extracting all gappy n-grams. For example, consider the following two utterances: *I need a copy of my bill* and *I need a copy of a past due bill*. As shown in Figures 4 and 5, the n-gram *WORD:copy WORD:of PHRASE:NP_bill* appears for both utterances, since both subsequences *my bill* and *a past due bill* are nothing but noun phrases with the head word *bill*.
- Another motivation is that, when using simply the word n-grams in an utterance, the classifier is only given lexical information. Now the classifier is provided with more and different information using these extra syntactic and semantic features. For example, a named entity of type monetary amount may be strongly associated with some call-type.
- Furthermore, there is a close relationship between the call-types and semantic roles. For example, if the predicate is *order* this is most probably the call-type *Order(Item)* in a retail domain application. The simple n-gram approach would consider all the appearances of the unigram *order* as equal. However consider the utterance *I'd like to check an order* of a different call-type, where the *order* is not a predicate but an object. Word n-gram features will fail to capture this distinction.

Once the SSG of an utterance is formed, all the n-grams are extracted as features, and the decision of which one to select/use is left to the classifier.

4 Computation of the SSGs

In this section, the tools used to compute the information in SSGs are described and their performances on manually transcribed spoken dialog utterances are presented. All of these components may be improved independently, for the specific application domain.

4.1 Part-of-Speech Tagger

Part-of-speech tagging has been very well studied in the literature for many languages, and the approaches vary from rule-based to HMM-based and classifier-based (Church, 1988; Brill, 1995, among others) tagging. In our framework, we employ a simple HMM-based tagger, where the most probable tag sequence, $\hat{T}$, given the words, W, is output (Weischedel et al., 1993):

$$\hat{T} = \underset{T}{\operatorname{argmax}}\, P(T|W) = \underset{T}{\operatorname{argmax}}\, P(W|T)P(T)$$

Since we do not have enough data which is manually tagged with part-of-speech tags for our applications, we used Penn Treebank (Marcus et al., 1994) as our training set. Penn Treebank includes data from Wall Street Journal, Brown, ATIS, and Switchboard corpora. The final two sets are the most useful for our domain, since they are also from spoken language and include disfluencies. As a test set, we manually labeled 2,000 words of user utterances from an AT&T VoiceTone spoken dialog system application, and we achieved an accuracy of 94.95% on manually transcribed utterances. When we examined the errors, we have seen that the frequent word *please* is mis-labeled or frequently occurs as a verb in the training data, even when it is not. Given that the latest literature on POS tagging using Penn Treebank reports an accuracy of around 97% with in-domain

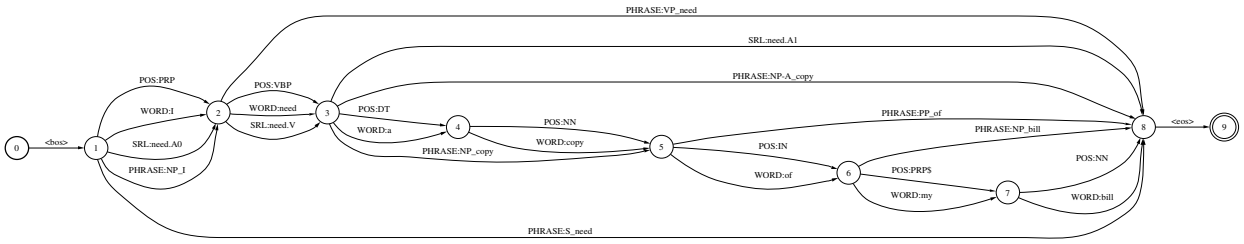

Figure 4: An example SSG for the utterance *I need a copy of my bill*.

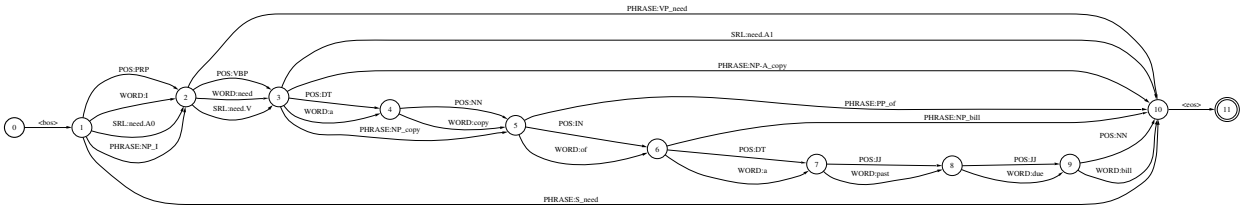

Figure 5: An example SSG for the utterance *I need a copy of a past due bill*.

training data (van Halteren et al., 2001), we achieve a very reasonable performance, considering these errors.

4.2 Syntactic Parser

For syntactic parsing, we use the Collins' parser (Collins, 1999), which is reported to give over 88% labeled recall and precision on Wall Street Journal portion of the Penn Treebank. We use Buchholz's `chunklink` script to extract information from the parse trees[3]. Since we do not have any data from our domain, we do not have a performance figure for this task for our domain.

4.3 Named Entity Extractor

For named entity extraction, we tried using a simple HMM-based approach, a simplified version of BBN's name finder (Bikel et al., 1999), and a classifier-based tagger using Boostexter (Schapire and Singer, 2000). In the simple HMM-based approach, which is the same as the part-of-speech tagging, the goal is to find the tag sequence, $\hat{T}$, which maximizes $P(T|W)$ for the word sequence, W. The tags in this case are named entity categories (such as *P* and *p* for Person names, *O* and *o* for Organization names, etc. where upper-case indicates the first word in the named entity) or *NA* if the word is not a part of a named entity. In the simplified version of BBN's name finder, the states of the model were word/tag combinations, where the tag t_i for word w_i is the named entity category of each word. Transition probabilities consisted of trigram probabilities $P(w_i/t_i|w_{i-1}/t_{i-1}, w_{i-2}/t_{i-2})$ over these combined tokens. In the final version, we extended this model with an unknown words model (Hakkani-Tür et al., 1999). In the classifier-based approach, we used simple features such as the current word and surrounding 4 words, binary tags indicating if the word considered contains any digits or is formed from digits, and features checking capitalization (Carreras et al., 2003).

To test these approaches, we have used data from an AT&T VoiceTone spoken dialog system application for a pharmaceutical domain, where some of the named entity categories were person, organization, drug name, prescription number, and date. The training and test sets contained around 11,000 and 5,000 utterances, respectively. Table 1 summarizes the overall F-measure results as well as F-measure for the most frequent named entity categories. Overall, the classifier based approach resulted in the best performance, so it is also used for the call classification experiments.

4.4 Semantic Role Labeling

The goal of semantic role labeling is to extract all the constituents which fill a semantic role of a target verb. Typical semantic arguments include Agent, Patient, Instrument, etc. and also adjuncts such as Locative, Temporal, Manner, Cause, etc. In this

[3] http://ilk.kub.nl/∼sabine/chunklink/chunklink_2-2-2000_for_conll.pl

Category	Count	HMM	IF	Boostexter
Org.	132	62.0	73.8	70.9
Person	150	45.0	62.4	54.4
Date	178	51.4	61.9	72.0
Drug	220	65.7	62.3	63.1
Overall	836	54.5	56.8	64.0

Table 1: F-Measure results for named entity extraction with various approaches. HMM is the simple HMM-based approach, IF is the simplified version of BBN's name finder with an unknown words model.

work, we use the semantic roles and annotations from the PropBank corpus (Kingsbury et al., 2002), where the arguments are given mnemonic names, such as Arg0, Arg1, Arg-LOC, etc. For example, for the sentence *I have bought myself a blue jacket from your summer catalog for twenty five dollars last week*, the agent (buyer, or Arg0) is *I*, the predicate is *buy*, the thing bought (Arg1) is *a blue jacket*, the seller or source (Arg2) is *from your summer catalog*, the price paid (Arg3) is *twenty five dollars*, the benefactive (Arg4) is *myself*, and the date (ArgM-TMP) is *last week*[4].

Semantic role labeling can be viewed as a multi-class classification problem. Given a word (or phrase) and its features, the goal is to output the most probable semantic label. For semantic role labeling, we have used the exact same feature set that Hacioglu *et al.* (2004) have used, since their system performed the best among others in the CoNLL-2004 shared task (Carreras and Màrquez, 2004). We have used Boostexter (Schapire and Singer, 2000) as the classifier. The features include token-level features (such as the current (head) word, its part-of-speech tag, base phrase type and position, etc.), predicate-level features (such as the predicate's lemma, frequency, part-of-speech tag, etc.) and argument-level features which capture the relationship between the token (head word/phrase) and the predicate (such as the syntactic path between the token and the predicate, their distance, token position relative to the predicate, etc.).

In order to evaluate the performance of semantic role labeling, we have manually annotated 285 utterances from an AT&T VoiceTone spoken dialog system application for a retail domain. The utterances include 645 predicates (2.3 predicates/utterance). First we have computed recall and precision rates for evaluating the predicate identification performance. The precision is found to be 93.04% and recall is 91.16%. More than 90% of false alarms for predicate extraction are due to the word *please*, which is very frequent in customer care domain and erroneously tagged as explained above. Most of the false rejections are due to disfluencies and ungrammatical utterances. For example in the utterance *I'd like to order place an order*, the predicate *place* is tagged as a noun erroneously, probably because of the preceding verb *order*. Then we have evaluated the argument labeling performance. We have used a stricter measure than the CoNLL-2004 shared task. The labeling is correct if both the boundary and the role of all the arguments of a predicate are correct. In our test set, we have found out that our SRL tool correctly tags all arguments of 57.4% of the predicates.

5 Call Classification Experiments and Results

In order to evaluate our approach, we carried out call classification experiments using human-machine dialogs collected by the spoken dialog system used for customer care. We have only considered utterances which are responses to the greeting prompt *How may I help you?* in order not to deal with confirmation and clarification utterances. We first describe this data, and then give the results obtained by the semantic classifier. We have performed our tests using the Boostexter tool, an implementation of the Boosting algorithm, which iteratively selects the most discriminative features for a given task (Schapire and Singer, 2000).

5.1 Data

Table 2 summarizes the characteristics of our application including the amount of training and test data, total number of call-types, average utterance length, and call-type perplexity. Perplexity is computed using the prior distribution over all the call-types in the training data.

5.2 Results

For call classification, we have generated SSGs for the training and test set utterances using the tools

[4]See http://www.cis.upenn.edu/∼dgildea/Verbs for more details

Training Data Size	3,725 utterances
Test Data Size	1,954 utterances
Number of Call-Types	79
Call-Type Perplexity	28.86
Average Utterance Length	12.08 words

Table 2: Characteristics of the data used in the experiments.

	Baseline	Using SSG	Increase
Unigram	2,303	6,875	2.99 times
Bigram	15,621	112,653	7.21 times
Trigram	34,185	705,673	20.64 times
Total	52,109	825,201	15.84 times

Table 3: A comparison of number of features.

described above. When n-grams are extracted from these SSGs, instead of the word graphs (Baseline), there is a huge increase in the number of features given to the classifier, as seen in Table 3. The classifier has now 15 times more features to work with. Although one can apply a feature selection approach before classification as frequently done in the machine learning community, we left the burden of analyzing 825,201 features to the classifier.

Table 4 presents the percentage of the features selected by Boostexter using SSGs for each information category. As expected the lexical information is the most frequently used, and 54.06% of the selected features have at least one word in its n-gram. The total is more than 100%, since some features contain more than one category, as in the bigram feature example: *POS:DT WORD:bill*. This shows the use of other information sources as well as words.

Table 5 presents our results for call classification. As the evaluation metric, we use the top class error rate (TCER), which is the ratio of utterances, where the top scoring call-type is not one of the true call-types assigned to each utterance by the human labelers. The baseline TCER on the test set using only word n-grams is 23.80%. When we extract features from the SSGs, we see a 2.14% relative decrease in the error rate down to 23.29%. When we analyze these results, we have seen that:

- For "easy to classify" utterances, the classifier already assigns a high score to the true call-type using just word n-grams.
- The syntactic and semantic features extracted from the SSGs are not 100% accurate, as presented earlier. So, although many of these features have been useful, there is certain amount of noise introduced in the call classification training data.
- The particular classifier we use, namely Boosting, is known to handle large feature spaces poorer than some others, such as SVMs. This is especially important with 15 times more features.

Category		Frequency
Lexical	Words	54.06%
Syntactic	Part-of-Speech	49.98%
	Syntactic Parse	27.10%
Semantic	Named Entity	1.70%
	Semantic Role Label	11.74%

Table 4: The percentage of the features selected by the classifier for each information category

	Baseline	SSGs	Decrease
All utterances	23.80%	23.29%	2.14%
Low confidence utterances	68.77%	62.16%	9.61%
All utterances (Cascaded)	23.80%	22.67%	4.74%

Table 5: Call classification error rates using words and SSGs.

Due to this analysis, we have focused on a subset of utterances, namely utterances with low confidence scores, i.e. cases where the score given to the top scoring call-type by the baseline model is below a certain threshold. In this subset we had 333 utterances, which is about 17% of the test set. As expected the error rates are much higher than the overall and we get much larger improvement in performance when we use SSGs. The baseline for this set is 68.77%, and using extra features, this reduces to 62.16% which is a 9.61% relative reduction in the error rate.

This final experiment suggests a cascaded approach for exploiting SSGs for call classification.

That is, first the baseline word n-gram based classifier is used to classify all the utterances, then if this model fails to commit on a call-type, we perform extra feature extraction using SSGs, and use the classification model trained with SSGs. This cascaded approach reduced the overall error rate of all utterances from 23.80% to 22.67%, which is 4.74% relative reduction in error rate.

6 Conclusions

In this paper, we have introduced syntactic and semantic graphs (SSGs) for speech and language processing. We have described their use for the task of call classification. We have presented results showing 4.74% improvement, using utterances collected from AT&T VoiceTone spoken dialog system. SSGs can also be useful for text classification and other similar language processing applications. Our future work includes feature selection prior to classification and developing methods that are more robust to ASR errors while computing the SSGs. We also plan to improve the syntactic and semantic processing components by adapting the models with some amount of labeled in-domain spoken dialog data.

References

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2), June.

Daniel M. Bikel, Richard Schwartz, and Ralph M. Weischedel. 1999. An algorithm that learns what's in a name. *Machine Learning Journal Special Issue on Natural Language Learning*, 34(1-3):211–231.

Eric Brill. 1995. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 21(4):543–565, December.

Xavier Carreras and Lluís Màrquez. 2004. Introduction to the CoNLL-2004 shared task: Semantic role labeling. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, Boston, MA, May.

Xavier Carreras, Lluís Màrquez, and Lluís Padró. 2003. A simple named entity extractor using AdaBoost. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, Edmonton, Canada.

Jennifer Chu-Carroll and Bob Carpenter. 1999. Vector-based natural language call routing. *Computational Linguistics*, 25(3):361–388.

Kenneth W. Church. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Second Conference on Applied Natural Language Processing (ANLP)*, pages 136–143, Austin, Texas.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Computer and Information Science, Philadelphia, PA.

Giuseppe Di Fabbrizio , Dawn Dutton, Narendra Gupta, Barbara Hollister, Mazin Rahim, Giuseppe Riccardi, Robert Schapire, and Juergen Schroeter. 2002. AT&T help desk. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Denver, CO, September.

Allen L. Gorin, Giuseppe Riccardi, and Jerry H. Wright. 1997. How May I Help You? . *Speech Communication*, 23:113–127.

Narendra Gupta, Gokhan Tur, Dilek Hakkani-Tür, Srinivas Bangalore, Giuseppe Riccardi, and Mazin Rahim. To appear. The AT&T spoken language understanding system. *IEEE Transactions on Speech and Audio Processing*.

Kadri Hacioglu, Sameer Pradhan, Wayne Ward, James H. Martin, and Dan Jurafsky. 2004. Semantic role labeling by tagging syntactic chunks. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, Boston, MA, May.

Dilek Hakkani-Tür, Gokhan Tur, Andreas Stolcke, and Elizabeth Shriberg. 1999. Combining words and prosody for information extraction from speech. In *Proceedings of the EUROSPEECH'99,* 6^{th} *European Conference on Speech Communication and Technology*, Budapest, Hungary, September.

Paul Kingsbury, Mitch Marcus, and Martha Palmer. 2002. Adding semantic annotation to the Penn TreeBank. In *Proceedings of the Human Language Technology Conference (HLT)*, San Diego, CA, March.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Giuseppe Riccardi and Dilek Hakkani-Tür. 2003. Active and unsupervised learning for automatic speech recognition. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Geneva, Switzerland, September.

Robert E. Schapire and Yoram Singer. 2000. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2-3):135–168.

Hans van Halteren, Jakub Zavrel, and Walter Daelemans. 2001. Improving accuracy in word class tagging through combination of machine learning systems. *Computational Linguistics*, 27(2):199–230.

Ralph Weischedel, Richard Schwartz, Jeff Palmucci, Marie Meteer, and Lance Ramshaw. 1993. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics, Special Issue on Using Large Corpora*, 19(2):361–382, June.

Feature-Based Segmentation of Narrative Documents

David Kauchak
Palo Alto Research Center and
University of California, San Diego
San Diego, CA 92093
dkauchak@cs.ucsd.edu

Francine Chen
Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto, CA 94304
fchen@parc.com

Abstract

In this paper we examine topic segmentation of narrative documents, which are characterized by long passages of text with few headings. We first present results suggesting that previous topic segmentation approaches are not appropriate for narrative text. We then present a feature-based method that combines features from diverse sources as well as learned features. Applied to narrative books and encyclopedia articles, our method shows results that are significantly better than previous segmentation approaches. An analysis of individual features is also provided and the benefit of generalization using outside resources is shown.

1 Introduction

Many long text documents, such as magazine articles, narrative books and news articles contain few section headings. The number of books in narrative style that are available in digital form is rapidly increasing through projects such as Project Gutenberg and the Million Book Project at Carnegie Mellon University. Access to these collections is becoming easier with directories such as the Online Books Page at the University of Pennsylvania.

As text analysis and retrieval moves from retrieval of documents to retrieval of document passages, the ability to segment documents into smaller, coherent regions enables more precise retrieval of meaningful portions of text (Hearst, 1994) and improved question answering. Segmentation also has applications in other areas of information access, including document navigation (Choi, 2000), anaphora and ellipsis resolution, and text summarization (Kozima, 1993).

Research projects on text segmentation have focused on broadcast news stories (Beeferman et al., 1999), expository texts (Hearst, 1994) and synthetic texts (Li and Yamanishi, 2000; Brants et al., 2002). Broadcast news stories contain cues that are indicative of a new story, such as "coming up", or phrases that introduce a reporter, which are not applicable to written text. In expository texts and synthetic texts, there is repetition of terms within a topical segment, so that the similarity of "blocks" of text is a useful indicator of topic change. *Synthetic* texts are created by concatenating stories, and exhibit stronger topic changes than the subtopic changes within a document; consequently, algorithms based on the similarity of text blocks work well on these texts.

In contrast to these earlier works, we present a method for segmenting *narrative* documents. In this domain there is little repetition of words and the segmentation cues are weaker than in broadcast news stories, resulting in poor performance from previous methods.

We present a feature-based approach, where the features are more strongly engineered using linguistic knowledge than in earlier approaches. The key to most feature-based approaches, particularly in NLP tasks where there is a broad range of possible feature sources, is identifying appropriate features. Selecting features in this domain presents a number of interesting challenges. First, features used in previous methods are not sufficient for solving this problem. We explore a number of different sources of information for extracting features, many previously unused. Second, the sparse nature of text and the high

Proceedings of the ACL Workshop on Feature Engineering for Machine Learning in NLP, pages 32–39,
Ann Arbor, June 2005.

cost of obtaining training data requires generalization using outside resources. Finally, we incorporate features from non-traditional resources such as lexical chains where features must be extracted from the underlying knowledge representation.

2 Previous Approaches

Previous topic segmentation methods fall into three groups: similarity based, lexical chain based, and feature based. In this section we give a brief overview of each of these groups.

2.1 Similarity-based

One popular method is to generate similarities between blocks of text (such as blocks of words, sentences or paragraphs) and then identify section boundaries where dips in the similarities occur.

The cosine similarity measure between term vectors is used by Hearst (1994) to define the similarity between blocks. She notes that the largest dips in similarity correspond to defined boundaries. Brants et al. (2002) learn a PLSA model using EM to smooth the term vectors. The model is parameterized by introducing a latent variable, representing the possible "topics". They show good performance on a number of different synthetic data sets.

Kozima and Furugori (1994) use another similarity metric they call "lexical cohesion". The "cohesiveness" of a pair of words is calculated by spreading activation on a semantic network as well as word frequency. They showed that dips in lexical cohesion plots had some correlation with human subject boundary decisions on one short story.

2.2 Lexical Chains

Semantic networks define relationships between words such as synonymy, specialization/generalization and part/whole. Stokes et al. (2002) use these relationships to construct lexical chains. A lexical chain is a sequence of lexicographically related word occurrences where every word occurs within a set distance from the previous word. A boundary is identified where a large numbers of lexical chains begin and end. They showed that lexical chains were useful for determining the text structure on a set of magazine articles, though they did not provide empirical results.

2.3 Feature-based

Beeferman et al. (1999) use an exponential model and generate features using a maximum entropy selection criterion. Most features learned are cue-based features that identify a boundary based on the occurrence of words or phrases. They also include a feature that measures the difference in performance of a "long range" vs. "short range" model. When the short range model outperforms the long range model, this indicates a boundary. Their method performed well on a number of broadcast news data sets, including the CNN data set from TDT 1997.

Reynar (1999) describes a maximum entropy model that combines hand selected features, including: broadcast news domain cues, number of content word bigrams, number of named entities, number of content words that are WordNet synonyms in the left and right regions, percentage of content words in the right segment that are first uses, whether pronouns occur in the first five words, and whether a word frequency based algorithm predicts a boundary. He found that for the HUB-4 corpus, which is composed of transcribed broadcasts, that the combined feature model performed better than TextTiling.

Mochizuki et al. (1998) use a combination of linguistic cues to segment Japanese text. Although a number of cues do not apply to English (e.g., topical markers), they also use anaphoric expressions and lexical chains as cues. Their study was small, but did indicate that lexical chains are a useful cue in some domains.

These studies indicate that a combination of features can be useful for segmentation. However, Mochizuki et al. (1998) analyzed Japanese texts, and Reynar (1999) and Beeferman et al. (1999) evaluated on broadcast news stories, which have many cues that narrative texts do not. Beeferman et al. (1999) also evaluated on concatenated Wall Street Journal articles, which have stronger topic changes than within a document. In our work, we examine the use of linguistic features for segmentation of narrative text in English.

3 Properties of Narrative Text

Characterizing data set properties is the first step towards deriving useful features. The approaches in the previous section performed well on broad-

Table 1: Previous approaches evaluated on narrative data from *Biohazard*

Model	Word Error	Sent. Error	Window Diff
random	0.486	0.490	0.541
TextTiling	0.481	0.497	0.526
PLSA	0.480	0.521	0.559

cast news, expository and synthetic data sets. Many properties of these documents are not shared by narrative documents. These properties include: 1) cue phrases, such as "welcome back" and "joining us" that feature-based methods used in broadcast news, 2) strong topic shifts, as in synthetic documents created by concatenating newswire articles, and 3) large data sets such that the training data and testing data appeared to come from similar distributions.

In this paper we examine two narrative-style books: *Biohazard* by Ken Alibek and *The Demon in the Freezer* by Richard Preston. These books are segmented by the author into sections. We manually examined these author identified boundaries and they are reasonable. We take these sections as true locations of segment boundaries. We split *Biohazard* into three parts, two for experimentation (exp1 and exp2) and the third as a holdout for testing. *Demon in the Freezer* was reserved for testing. *Biohazard* contains 213 true and 5858 possible boundaries. *Demon* has 119 true and 4466 possible boundaries. Locations between sentences are considered possible boundaries and were determined automatically.

We present an analysis of properties of the book *Biohazard* by Ken Alibek as an exemplar of narrative documents (for this section, test=exp1 and train=exp2). These properties are different from previous expository data sets and will result in poor performance for the algorithms mentioned in Section 2. These properties help guide us in deriving features that may be useful for segmenting narrative text.

Vocabulary The book contains a single topic with a number of sub-topics. These changing topics, combined with the varied use of words for narrative documents, results in many unseen terms in the test set. 25% of the content words in the test set do not occur in the training set and a third of the words in the test set occur two times or less in the training set. This causes problems for those methods that learn a model of the training data such as Brants et al. (2002) and Beeferman et al. (1999) because, without outside resources, the information in the training data is not sufficient to generalize to the test set.

Boundary words Many feature-based methods rely on cues at the boundaries (Beeferman et al., 1999; Reynar, 1999). 474 content terms occur in the first sentence of boundaries in the training set. Of these terms, 103 occur at the boundaries of the test set. However, of those terms that occur *significantly* at a training set boundary (where significant is determined by a likelihood-ratio test with a significance level of 0.1), only 9 occur at test boundaries. No words occur significantly at a training boundary AND also significantly at a test boundary.

Segment similarity Table 1 shows that two similarity-based methods that perform well on synthetic and expository text perform poorly (i.e., on par with random) on *Biohazard*. The poor performance occurs because block similarities provide little information about the actual segment boundaries on this data set. We examined the average similarity for two adjacent regions within a segment versus the average similarity for two adjacent regions that cross a segment boundary. If the similarity scores were useful, the within segment scores would be higher than across segment scores. Similarities were generated using the PLSA model, averaging over multiple models with between 8 and 20 latent classes. The average similarity score within a segment was 0.903 with a standard deviation of 0.074 and the average score across a segment boundary was 0.914 with a standard deviation of 0.041. In this case, the across boundary similarity is actually higher. Similar values were observed for the cosine similarities used by the TextTiling algorithm, as well as with other numbers of latent topics for the PLSA model. For all cases examined, there was little difference between inter-segment similarity and across-boundary similarity, and there was always a large standard deviation.

Lexical chains Lexical chains were identified as synonyms (and exact matches) occurring within a distance of one-twentieth the average segment length and with a maximum chain length equal to the average segment length (other values were ex-

amined with similar results). Stokes et al. (2002) suggest that high concentrations of lexical chain beginnings and endings are indicative of a boundary location. On the narrative data, of the 219 overall chains, only 2 begin at a boundary and only 1 ends at a boundary. A more general heuristic identifies boundaries where there is an increase in the number of chains beginning and ending near a possible boundary while also minimizing chains that span boundaries. Even this heuristic does not appear indicative on this data set. Over 20% of the chains actually cross segment boundaries. We also measured the average distance from a boundary and the nearest beginning and ending of a chain if a chain begins/ends within that segment. If the chains are a good feature, then these should be relatively small. The average segment length is 185 words, but the average distance to the closest beginning chain is 39 words away and closest ending chain is 36 words away. Given an average of 4 chains per segment, the beginning and ending of chains were not concentrated near boundary locations in our narrative data, and therefore not indicative of boundaries.

4 Feature-Based Segmentation

We pose the problem of segmentation as a classification problem. Sentences are automatically identified and each boundary between sentences is a possible segmentation point. In the classification framework, each segmentation point becomes an example. We examine both support vector machines (SVMlight (Joachims, 1999)) and boosted decision stumps (Weka (Witten and Frank, 2000)) for our learning algorithm. SVMs have shown good performance on a variety of problems, including natural language tasks (Cristianini and Shawe-Taylor, 2000), but require careful feature selection. Classification using boosted decisions stumps can be a helpful tool for analyzing the usefulness of individual features. Examining multiple classification methods helps avoid focusing on the biases of a particular learning method.

4.1 Example Reweighting

One problem with formulating the segmentation problem as a classification problem is that there are many more negative than positive examples. To discourage the learning algorithm from classifying all results as negative and to instead focus on the positive examples, the training data must be reweighted.

We set the weight of positive vs. negative examples so that the number of boundaries after testing agrees with the expected number of segments based on the training data. This is done by iteratively adjusting the weighting factor while re-training and retesting until the predicted number of segments on the test set is approximately the expected number. The expected number of segments is the number of sentences in the test set divided by the number of sentences per segment in the training data. This value can also be weighted based on prior knowledge.

4.2 Preprocessing

A number of preprocessing steps are applied to the books to help increase the informativeness of the texts. The book texts were obtained using OCR methods with human correction. The text is preprocessed by tokenizing, removing stop words, and stemming using the Inxight LinguistiX morphological analyzer. Paragraphs are identified using formatting information. Sentences are identified using the TnT tokenizer and parts of speech with the TnT part of speech tagger (Brants, 2000) with the standard English *Wall Street Journal* n-grams. Named entities are identified using finite state technology (Beesley and Karttunen, 2003) to identify various entities including: person, location, disease and organization. Many of these preprocessing steps help provide salient features for use during segmentation.

4.3 Engineered Features

Segmenting narrative documents raises a number of interesting challenges. First, labeling data is extremely time consuming. Therefore, outside resources are required to better generalize from the training data. WordNet is used to identify words that are similar and tend to occur at boundaries for the "word group" feature. Second, some sources of information, in particular entity chains, do not fit into the standard feature based paradigm. This requires extracting features from the underlying information source. Extracting these features represents a tradeoff between information content and generalizability. In the case of entity chains, we extract features that characterize the occurrence distribution of the

entity chains. Finally, the "word groups" and "entity groups" feature groups generate candidate features and a selection process is required to select useful features. We found that a likelihood ratio test for significance worked well for identifying those features that would be useful for classification. Throughout this section, when we use the term "significant" we are referring to significant with respect to the likelihood ratio test (with a significance level of 0.1).

We selected features both a priori and dynamically during training (i.e., word groups and entity groups are selected dynamically). Feature selection has been used by previous segmentation methods (Beeferman et al., 1999) as a way of adapting better to the data. In our approach, knowledge about the task is used more strongly in defining the feature types, and the selection of features is performed prior to the classification step. We also used mutual information, statistical tests of significance and classification performance on a development data set to identify useful features.

Word groups In Section 3 we showed that there are not consistent cue phrases at boundaries. To generalize better, we identify word *groups* that occur significantly at boundaries. A word group is all words that have the same parent in the WordNet hierarchy. A binary feature is used for each learned group based on the occurrence of at least one of the words in the group. Groups found include months, days, temporal phrases, military rankings and country names.

Entity groups For each entity group (i.e. named entities such as person, city, or disease tagged by the named entity extractor) that occurs significantly at a boundary, a feature indicating whether or not an entity of that group occurs in the sentence is used.

Full name The named entity extraction system tags persons named in the document. A rough co-reference resolution was performed by grouping together references that share at least one token (e.g., "General Yury Tikhonovich Kalinin" and "Kalinin"). The full name of a person is the longest reference of a group referring to the same person. This feature indicates whether or not the sentence contains a full name.

Entity chains Word relationships work well when the documents have disjoint topics; however, when topics are similar, words tend to relate too easily. We propose a more stringent chaining method called entity chains. Entity chains are constructed in the same fashion as lexical chains, except we consider named entities. Two entities are considered related (i.e. in the same chain) if they refer to the same entity. We construct entity chains and extract features that characterize these chains: How many chains start/end at this sentence? How many chains cross over this sentence/previous sentence/next sentence? Distance to the nearest dip/peak in the number of chains? Size of that dip/peak?

Pronoun Does the sentence contain a pronoun? Does the sentence contain a pronoun within 5 words of the beginning of the sentence?

Numbers During training, the patterns of numbers that occur significantly at boundaries are selected. Patterns considered are any number and any number with a specified length. The feature then checks if that pattern appears in the sentence. A commonly found pattern is the number pattern of length 4, which often refers to a year.

Conversation Is this sentence part of a conversation, i.e. does this sentence contain "direct speech"? This is determined by tracking beginning and ending quotes. Quoted regions and single sentences between two quoted regions are considered part of a conversation.

Paragraph Is this the beginning of a paragraph?

5 Experiments

In this section, we examine a number of narrative segmentation tasks with different segmentation methods. The only data used during development was the first two thirds from *Biohazard* (exp1 and exp2). All other data sets were only examined after the algorithm was developed and were used for testing purposes. Unless stated otherwise, results for the feature based method are using the SVM classifier.[1]

5.1 Evaluation Measures

We use three segmentation evaluation metrics that have been recently developed to account for "close but not exact" placement of hypothesized boundaries: word error probability, sentence error probability, and WindowDiff. Word error probability

[1] SVM and boosted decision stump performance is similar. For brevity, only SVM results are shown for most results.

Table 2: Experiments with *Biohazard*

	Word Error	Sent. Error	Window Diff	Sent err improv
Biohazard				
random (sent.)	0.488	0.485	0.539	——-
random (para.)	0.481	0.477	0.531	(base)
Biohazard				
exp1 → holdout	0.367	0.357	0.427	25%
exp2 → holdout	0.344	0.325	0.395	32%
3x cross validtn.	0.355	0.332	0.404	24%
Train *Biohazard* Test *Demon*	0.387	0.364	0.473	25%

(Beeferman et al., 1999) estimates the probability that a randomly chosen pair of words k words apart is incorrectly classified, i.e. a false positive or false negative of being in the same segment. In contrast to the standard classification measures of precision and recall, which would consider a "close" hypothesized boundary (e.g., off by one sentence) to be incorrect, word error probability gently penalizes "close" hypothesized boundaries. We also compute the sentence error probability, which estimates the probability that a randomly chosen pair of sentences s sentences apart is incorrectly classified. k and s are chosen to be half the average length of a section in the test data. WindowDiff (Pevzner and Hearst, 2002) uses a sliding window over the data and measures the difference between the number of hypothesized boundaries and the actual boundaries within the window. This metric handles several criticisms of the word error probability metric.

5.2 Segmenting Narrative Books

Table 2 shows the results of the SVM-segmenter on *Biohazard* and *Demon in the Freezer*. A baseline performance for segmentation algorithms is whether the algorithm performs better than naive segmenting algorithms: choose no boundaries, choose all boundaries and choose randomly. Choosing all boundaries results in word and sentence error probabilities of approximately 55%. Choosing no boundaries is about 45%. Table 2 also shows the results for random placement of the **correct** number of segments. Both random boundaries at sentence locations and random boundaries at paragraph locations are shown (values shown are the averages of 500 random runs). Similar results were obtained for random segmentation of the *Demon* data.

Table 3: Performance on Groliers articles

	Word Error	Sent. Error	Window Diff
random	0.482	0.483	0.532
TextTile	0.407	0.412	0.479
PLSA	0.420	0.435	0.507
features (stumps)	0.387	0.400	0.495
features (SVM)	0.385	0.398	0.503

For *Biohazard* the holdout set was not used during development. When trained on either of the development thirds of the text (i.e., exp1 or exp2) and tested on the test set, a substantial improvement is seen over random. 3-fold cross validation was done by training on two-thirds of the data and testing on the other third. Recalling from Table 1 that both PLSA and TextTiling result in performance similar to random even when given the correct number of segments, we note that all of the single train/test splits performed better than any of the naive algorithms and previous methods examined.

To examine the ability of our algorithm to perform on unseen data, we trained on the entire *Biohazard* book and tested on *Demon in the Freezer*. Performance on *Demon in the Freezer* is only slightly worse than the *Biohazard* results and is still much better than the baseline algorithms as well as previous methods. This is encouraging since *Demon* was not used during development, is written by a different author and has a segment length distribution that is different than *Biohazard* (average segment length of 30 vs. 18 in *Biohazard*).

5.3 Segmenting Articles

Unfortunately, obtaining a large number of narrative books with meaningful labeled segmentation is difficult. To evaluate our algorithm on a larger data set as well as a wider variety of styles similar to narrative documents, we also examine 1000 articles from Groliers Encyclopedia that contain subsections denoted by major and minor headings, which we consider to be the true segment boundaries. The articles contained 8,922 true and 102,116 possible boundaries. We randomly split the articles in half, and perform two-fold cross-validation as recommended by Dietterich (1998). Using 500 articles from one half of the pair for testing, 50 articles are randomly selected from the other half for training. We used

Table 4: Ave. human performance (Hearst, 1994)

	Word Error (%)	Sent. Error (%)	Window Diff (%)
Sequoia	0.275	0.272	0.351
Earth	0.219	0.221	0.268
Quantum	0.179	0167	0.316
Magellan	0.147	0.147	0.157

a subset of only 50 articles due to the high cost of labeling data. Each split yields two test sets of 500 articles and two training sets. This procedure of two-fold cross-validation is performed five times, for a total of 10 training and 10 corresponding test sets. Significance is then evaluated using the t-test.

The results for segmenting Groliers Encyclopedia articles are given in Table 3. We compare the performance of different segmentation models: two feature-based models (SVMs, boosted decision stumps), two similarity-based models (PLSA-based segmentation, TextTiling), and randomly selecting segmentation points. All segmentation systems are given the estimated number of segmentation points based based on the training data. The feature based approaches are significantly[2] better than either PLSA, TextTiling or random segmentation. For our selected features, boosted stump performance is similar to using an SVM, which reinforces our intuition that the selected features (and not just classification method) are appropriate for this problem.

Table 1 indicates that the previous TextTiling and PLSA-based approaches perform close to random on narrative text. Our experiments show a performance improvement of $>$24% by our feature-based system, and significant improvement over other methods on the Groliers data. Hearst (1994) examined the task of identifying the paragraph boundaries in *expository* text. We provide analysis of this data set here to emphasize that identifying segments in natural text is a difficult problem and since current evaluation methods were not used when this data was initially presented. Human performance on this task is in the 15%-35% error rate. Hearst asked seven human judges to label the paragraph boundaries of four different texts. Since no ground truth was available, true boundaries were identified by those boundaries that had a majority vote as a boundary. Table 4 shows the average human performance for each text. We show these results not for direct comparison with our methods, but to highlight that even human segmentation on a related task does not achieve particularly low error rates.

[2]For both SVM and stumps at a level of 0.005 using a t-test except SVM_TextTile-WindowDiff (at 0.05) and stumps_TextTile-WindowDiff and SVM/stumps_PLSA-WindowDiff (not significantly different)

Table 5: Feature occurrences at boundary and non-boundary locations

	boundary	non-boundary
Paragraph	74	621
Entity groups	44	407
Word groups	39	505
Numbers	16	59
Full name	2	109
Conversation	0	510
Pronoun	8	742
Pronoun ≤ 5	1	330

5.4 Analysis of Features

The top section of Table 5 shows features that are intuitively hypothesized to be positively correlated with boundaries and the bottom section shows negatively correlated. For this analysis, exp1 from *Alibek* was used for training and the holdout set for testing. There are 74 actual boundaries and 2086 possibly locations. Two features have perfect recall: paragraph and conversation. Every true section boundary is at a paragraph and no section boundaries are within conversation regions. Both the word group and entity group features have good correlation with boundary locations and also generalized well to the training data by occurring in over half of the positive test examples.

The benefit of generalization using outside resources can be seen by comparing the boundary words found using word groups versus those found only in the training set as in Section 3. Using word groups triples the number of significant words found in the training set that occur in the test set. Also, the number of shared words that occur significantly in both the training and test set goes from none to 9. More importantly, significant words occur in 37 of the test segments instead of none without the groups.

6 Discussion and Summary

Based on properties of narrative text, we proposed and investigated a set of features for segmenting narrative text. We posed the problem of segmentation as a feature-based classification problem, which presented a number of challenges: many different feature sources, generalization from outside resources for sparse data, and feature extraction from non-traditional information sources.

Feature selection and analyzing feature interaction is crucial for this type of application. The paragraph feature has perfect recall in that all boundaries occur at paragraph boundaries. Surprisingly, for certain train/test splits of the data, the performance of the algorithm was actually better without the paragraph feature than with it. We hypothesize that the noisiness of the data is causing the classifier to learn incorrect correlations.

In addition to feature selection issues, posing the problem as a classification problem loses the sequential nature of the data. This can produce very unlikely segment lengths, such as a single sentence. We alleviated this by selecting features that capture properties of the sequence. For example, the entity chains features represent some of this type of information. However, models for complex sequential data should be examined as possible better methods.

We evaluated our algorithm on two books and encyclopedia articles, observing significantly better performance than randomly selecting the correct number of segmentation points, as well as two popular, previous approaches, PLSA and TextTiling.

Acknowledgments

We thank Marti Hearst for the human subject performance data and the anonymous reviewers for their very helpful comments. Funded in part by the Advanced Research and Development Activity NIMD program (MDA904-03-C-0404).

References

Doug Beeferman, Adam Berger, and John Lafferty. 1999. Statistical models for text segmentation. *Machine Learning*, 34:177–210.

Kenneth R. Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI Publications, Palo Alto, CA.

Thorsten Brants, Francine Chen, and Ioannis Tsochantaridis. 2002. Topic-based document segmentation with probabilistic latent semantic analysis. In *Proceedings of CIKM*, pg. 211–218.

Thorsten Brants. 2000. TnT – a statistical part-of-speech tagger. In *Proceedings of the Applied NLP Conference*.

Freddy Choi. 2000. Improving the efficiency of speech interfaces for text navigation. In *Proceedings of IEEE Colloquium: Speech and Language Processing for Disabled and Elderly People*.

Nello Cristianini and John Shawe-Taylor. 2000. *An Introduction to Support Vector Machines*. Cambridge University Press.

Thomas Dietterich. 1998. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10:1895–1923.

Marti A. Hearst. 1994. Multi-paragraph segmentation of expository text. In *Meeting of ACL*, pg. 9–16.

Thorsten Joachims, 1999. *Advances in Kernel Methods - Support Vector Learning*, chapter Making large-Scale SVM Learning Practical. MIT-Press.

Hideki Kozima and Teiji Furugori. 1994. Segmenting narrative text into coherent scenes. In *Literary and Linguistic Computing*, volume 9, pg. 13–19.

Hideki Kozima. 1993. Text segmentation based on similarity between words. In *Meeting of ACL*, pg. 286–288.

Hang Li and Kenji Yamanishi. 2000. Topic analysis using a finite mixture model. In *Proceedings of Joint SIGDAT Conference of EMNLP and Very Large Corpora*, pg. 35–44.

Hajime Mochizuki, Takeo Honda, and Manabu Okumura. 1998. Text segmentation with multiple surface linguistic cues. In *COLING-ACL*, pg. 881–885.

Lev Pevzner and Marti Hearst. 2002. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, pg. 19–36.

Jeffrey Reynar. 1999. Statistical models for topic segmentation. In *Proceedings of ACL*, pg. 357–364.

Nicola Stokes, Joe Carthy, and Alex Smeaton. 2002. Segmenting broadcast news streams using lexical chains. In *Proceedings of Starting AI Researchers Symposium, (STAIRS 2002)*, pg. 145–154.

Ian H. Witten and Eibe Frank. 2000. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann.

Identifying non-referential *it*: a machine learning approach incorporating linguistically motivated patterns

Adriane Boyd
Department of Linguistics
The Ohio State University
1712 Neil Ave.
Columbus, OH 43210
adriane@ling.osu.edu

Whitney Gegg-Harrison & Donna Byron
Department of Computer Science and Engineering
The Ohio State University
2015 Neil Ave.
Columbus, OH 43210
{geggharr,dbyron}@cse.osu.edu

Abstract

In this paper, we present a machine learning system for identifying non-referential *it*. Types of non-referential *it* are examined to determine relevant linguistic patterns. The patterns are incorporated as features in a machine learning system which performs a binary classification of *it* as referential or non-referential in a POS-tagged corpus. The selection of relevant, generalized patterns leads to a significant improvement in performance.

1 Introduction

The automatic classification of *it* as either referential or non-referential is a topic that has been relatively ignored in the computational linguistics literature, with only a handful of papers mentioning approaches to the problem. With the term "non-referential *it*", we mean to refer to those instances of *it* which do not introduce a new referent. In the previous literature these have been called "pleonastic", "expletive", and "non-anaphoric". It is important to be able to identify instances of non-referential *it* to generate the correct semantic interpretation of an utterance. For example, one step of this task is to associate pronouns with their referents. In an automated pronoun resolution system, it is useful to be able to skip over these instances of *it* rather than attempt an unnecessary search for a referent for them, only to end up with inaccurate results. The task of identifying non-referential *it* could be incorporated into a part-of-speech tagger or parser, or viewed as an initial step in semantic interpretation.

We develop a linguistically-motivated classification for non-referential *it* which includes four types of non-referential *it*: extrapositional, cleft, weather/condition/time/place, and idiomatic, each of which will be discussed in more detail in Section 2. A subset of the BNC Sampler Corpus (Burnard, 1995) was chosen for our task because of its extended tagset and high tagging accuracy. Non-referential *it* makes up a significant proportion of the occurrences of *it* in our corpus, which contains a selection of written texts of various genres, approximately one-third prose fiction, one-third newspaper text, and one-third other non-fiction. In our corpus, there are 2337 instances of *it*, 646 of which are non-referential (28%). *It* appears in over 10% of the sentences in our corpus. The corpus is described in further detail in Section 3.

Previous research on this topic is fairly limited. Paice and Husk (1987) introduces a rule-based method for identifying non-referential *it* and Lappin and Leass (1994) and Denber (1998) describe rule-based components of their pronoun resolution systems which identify non-referential *it*. Evans (2001) describes a machine learning system which classifies *it* into seven types based on the type of referent. Their approaches are described in detail in Section 4. In Section 5 we describe our system which combines and extends elements of the systems developed by Paice and Husk (1987) and Evans (2001), and the results are presented in Section 6.

The authors would like to thank the GE Foundation Faculty for the Future grant for their support of this project. We would also like to thank Detmar Meurers and Erhard Hinrichs for their helpful advice and feedback.

Proceedings of the ACL Workshop on Feature Engineering for Machine Learning in NLP, pages 40–47,
Ann Arbor, June 2005.

2 Classification

The first step is to create a classification system for all instances of *it*. Though the goal is the binary classification of *it* as referential or non-referential, an annotation scheme is used which gives more detail about each instance of non-referential *it*, since they occur in a number of constructions. The main types of non-referential *it* are taken from the *Cambridge Grammar of the English Language* in the section on "Special uses of *it*", Section 2.5, Huddleston and Pullum (2002). Five main uses are outlined: extrapositional, cleft, weather/condition/time/place, idiomatic, and predicative. As noted in the *Cambridge Grammar*, predicative *it* seems to be more referential that the other types of non-referential *it*. Predicative *it* can typically be replaced with a demonstrative pronoun. Consider the example: *It is a dreary day. It* can be replaced with *This* with no change in grammaticality and no significant change in meaning: *This is a dreary day.* In contrast, replacing the other types of *it* with *this* results in nonsense, e.g., **This seems that the king is displeased.*

For our purposes, if a particular *it* can be replaced with a demonstrative pronoun and the resulting sentence is still grammatical and has no significant change in meaning, this *it* is referential and therefore annotated as referential. The demonstrative pronoun replacement test is not quite perfect (e.g., **This is a dreary day in Paris*), but no such instances of predicative *it* were found in the corpus so predicative *it* is always classified as referential. This leaves four types of *it*, each of which are described in detail below. The main examples for each type are taken from the corpus. See Section 3 for details about the corpus.

2.1 Extrapositional

When an element of a sentence is extraposed, *it* is often inserted as a placeholder in the original position of the now extraposed element. Most often, *it* appears in the subject position, but it can also appear as an object. Example (1) lists a few instances of extrapositional *it* from our corpus.

(1) a. **It** has been confirmed this week that political parties will no longer get financial subsidies.

b. She also made **it** clear that Conductive Education is not the only method.

c. You lead life, **it** seems to me, like some ritual that demands unerring performance.

The extraposed element is typically a subordinate clause, and the type of clause depends on lexical properties of verbs and adjectives in the sentence, see (2).

(2) * It was difficult that X.
It was difficult to X.

* It was clear to X.
It was clear that X.

As (1c) shows, extrapositional *it* can also appear as part of a truncated extrapositional phrase as a kind of parenthetical comment embedded in a sentence.

2.2 Cleft

It appears as the subject of *it*-cleft sentences. When an *it*-cleft sentence is formed, the foregrounded phrase becomes the complement of the verb *be* and the rest of sentence is backgrounded in a relative clause. The foregrounded phrase in a cleft sentence can be a noun phrase, prepositional phrase, adjective phrase, adverb phrase, non-finite clause, or content clause.

(3) a. It was **the military district commander** who stepped in to avoid bloodshed. (*noun phrase*)

b. It is **on this point** that the views of the SACP and some Soviet policymakers divide. (*prepositional phrase*)

c. 'Tis **glad** I am to 'ear it, me lord. (*adjective phrase*)

Additionally, the foregrounded phrase can sometimes be fronted:

(4) **He** it was who ushered in the new head of state.

More context than the immediate sentence is needed to accurately identify *it*-cleft sentences. First, clefts with a foregrounded noun phrase are ambiguous between cleft sentences (5a) and sentences where the noun phrase and relative clause form a constituent (5b).

(5) a. A: I heard that the general stepped in to avoid bloodshed.
B: No, it was the military district commander who stepped in.

b. A: Was that the general being interviewed on the news?
B: No, it was the military district commander who stepped in to avoid bloodshed.

Due to this ambiguity, we expect that it may be difficult to classify clefts. In addition, there are difficulties because the relative clause does not always appear in full. In various situations the relative pronoun can be omitted, the relative clause can be reduced, or the relative clause can be omitted entirely.

2.3 Weather/Condition/Time/Place

It appears as the subject of weather and other related predicates involving condition, time, and place/distance:

(6) a. It was snowing steadily outside.

b. It was about midnight.

c. It was no distance to Mutton House.

d. It was definitely not dark.

2.4 Idiomatic

In idioms, *it* can appear as the subject, object, or object of a preposition.

(7) a. After three weeks it was my turn to go to the delivery ward at Fulmer.

b. Cool it!

c. They have not had an easy time of it.

2.5 General Notes

Non-referential *it* is most often the subject of a sentence, but in extrapositional and idiomatic cases, it can also be the object. Idioms are the only cases where non-referential *it* is found as the object of a preposition.

3 Corpus

The BNC Sampler Corpus (Burnard, 1995) was chosen for its extended tagset and high tagging accuracy. The C7 tagset used for this corpus has a unique tag for *it*, which made the task of identifying all occurrences of *it* very simple. We chose a subset consisting of 350,000 tokens from written texts in variety of genres. The breakdown by text type can be seen in Table 1.

Prose fiction	32%
Newspaper text	38%
Other non-fiction	30%

Table 1: Text types in our corpus

	# of Instances	% of Inst.
Extrapositional	477	20.4%
Cleft	119	5.1%
Weather	69	2.9%
Idiomatic	46	2.0%
Referential	1626	69.6%
Total	2337	100%

Table 2: Instances of *it* in our corpus

The two lead authors independently annotated each occurence with one of the labels shown in Table 2 and then came to a joint decision on the final annotation. The breakdown of the instances of *it* in our corpus is shown in Table 2. There are 2337 occurrences of *it*, 646 of which are non-referential (28%). Ten percent of the corpus, taken from all sections, was set aside as test data. The remaining section, which contains 2100 instances of *it*, became our training data.

4 Previous Research

Paice and Husk (1987) reports a simple rule-based system that was used to identify non-referential *it* in the technical section of the Lancaster-Oslo/Bergen Corpus. Because of the type of text, the distribution of types of non-referential *it* is somewhat limited, so they only found it necessary to write rules to match extrapositional and cleft *it* (although they do mention two idioms found in the corpus). The corpus was plain text, so their rules match words and punctuation directly.

Their patterns find *it* as a left bracket and search for a right bracket related to the extrapositional and cleft grammatical patterns (*to*, *that*, etc.). For the extrapositional instances, there are lists of words which are matched in between *it* and the right

Accuracy	92%
Precision	93%
Recall	97%

Table 3: Paice and Husk (1987): Results

Accuracy	79%
Precision	80%
Recall	31%

Table 4: Replicating Paice and Husk (1987)

bracket. The word lists are task-status words (STATUS), state-of-knowledge words (STATE), and a list of prepositions and related words (PREP), which is used to rule out right brackets that could potentially be objects of prepositions. Patterns such as "it STATUS to" and "it !PREP that" were created. The left bracket can be at most 27 words from the right bracket and there can be either zero or two or more commas or dashes between the left and right brackets. Additionally, their system had a rule to match parenthetical *it*: there is a match when *it* appears immediately following a comma and another comma follows within four words. Their results, shown in Table 3, are impressive.

We replicated their system and ran it on our testing data, see Table 4. Given the differences in text types, it is not surprising that their system did not perform as well on our corpus. The low recall seems to show the limitations of fixed word lists, while the reasonably high precision shows that the simple patterns tend to be accurate in the cases where they apply.

Lappin and Leass (1994) and Denber (1998) mention integrating small sets of rules to match non-referential *it* into their larger pronoun resolution systems. Lappin and Leass use two words lists and a short set of rules. One word list is modal adjectives (*necessary*, *possible*, *likely*, etc.) and the other is cognitive verbs (*recommend*, *think*, *believe*, etc.). Their rules are as follows:

It is Modaladj that S
It is Modaladj (for NP) to VP
It is Cogv-ed that S
It seems/appears/means/follows (that) S
NP makes/finds it Modaladj (for NP) to VP

Accuracy	71%
Precision	73%
Recall	69%

Table 5: Evans (2001): Results, Binary Classification

It is time to VP
It is thanks to NP that S

Their rules are mainly concerned with extrapositional *it* and they give no mention of cleft *it*. They give no direct results for this component of their system, so it is not possible to give a comparison. Denber (1998) includes a slightly revised and extended version of Lappin and Leass's system and adds in detection of weather/time *it*. He suggests using WordNet to extend word lists.

Evans (2001) begins by noting that a significant percentage of instances of *it* do not have simple nominal referents and describes a system which uses a memory-based learning (MBL) algorithm to perform a 7-way classification of *it* by type of referent. We consider two of his categories, *pleonastic* and *stereotypic/idiomatic*, to be non-referential. Evans created a corpus with texts from the BNC and SUSANNE corpora and chose to use a memory-based learning algorithm. A memory-based learning algorithm classifies new instances on the basis of their similarity to instances seen in the training data. Evans chose the k-nearest neighbor algorithm from the Tilburg Memory-Based Learner (TiMBL) package (Daelemans et al., 2003) with approximately 35 features relevant to the 7-way classification. Although his system was created for the 7-way classification task, he recognizes the importance of the binary referential/non-referential distinction and gives the results for the binary classification of pleonastic *it*, see Table 5. His results for the classification of idiomatic *it* (33% precision and 0.7% recall) show the limitations of a machine learning system given sparse data.

We replicated Evans's system with a simplified set of features to perform the referential/non-referential classification of *it*. We did not include features that would require chunking or features that seemed relevant only for distinguishing kinds of referential *it*. The following thirteen features are used:

Accuracy	76%
Precision	57%
Recall	60%

Table 6: Replicating Evans (2001)

1-8. four preceding and following POS tags
9-10. lemmas of the preceding and following verbs
11. lemma of the following adjective
12. presence of *that* following
13. presence of an immediately preceding preposition

Using our training and testing data with the same algorithm from TiMBL, we obtained results similar to Evans's, shown in Table 6. The slightly higher accuracy is likely due to corpus differences or the reduced feature set which ignores features largely relevant to other types of *it*.

Current state-of-the-art reference resolution systems typically include filters for non-referential noun phrases. An example of such a system is Ng and Cardie (2002), which shows the improvement in reference resolution when non-referential noun phrases are identified. Results are not given for the specific task of identifying non-referential *it*, so a direct comparison is not possible.

5 Method

As seen in the previous section, both rule-based and machine learning methods have been shown to be fairly effective at identifying non-referential *it*. Rule-based methods look for the grammatical patterns known to be associated with non-referential *it* but are limited by fixed word lists; machine learning methods can handle open classes of words, but are less able to generalize about the grammatical patterns associated with non-referential *it* from a small training set.

Evans's memory-based learning system showed a slight integration of rules into the machine learning system by using features such as the presence of following *that*. Given the descriptions of types of non-referential *it* from Section 2, it is possible to create more specific rules which detect the fixed grammatical patterns associated with non-referential *it* such as *it VERB that* or *it VERB ADJ to*. Many of these patterns are similar to Paice and Husk's, but having part-of-speech tags allows us to create more general rules without reference to specific lexical items. If the results of these rule matches are integrated as features in the training data for a memory-based learning system along with relevant verb and adjective lemmas, it becomes possible to incorporate knowledge about grammatical patterns without creating fixed word lists. The following sections examine each type of non-referential *it* and describe the patterns and features that can be used to help automatically identify each type.

5.1 Extrapositional *it*

Extrapositional *it* appears in a number of fairly fixed patterns, nine of which are shown below. Intervening tokens are allowed between the words in the patterns. **F4-6** are more general versions of **F1-3** but are not as indicative of non-referential *it*, so it useful to keep them separate even though ones that match **F1-3** will also match **F4-6**. **F7** applies when *it* is the object of a verb. To simplify patterns like **F8**, all verbs in the sentence are lemmatized with *morpha* (Minnen et al., 2001) before the pattern matching begins.

F1 *it* VERB ADJ *that*
F2 *it* VERB ADJ *what/which/where/whether/why/how*
F3 *it* VERB ADJ *to*
F4 *it* VERB *that*
F5 *it* VERB *what/which/where/whether/why/how*
F6 *it* VERB *to*
F7 *it* ADJ *that/to*
F8 *it be/seem as if*
F9 *it* VERB COMMA

For each item above, the feature consists of the distance (number of tokens) between *it* and the end of the match (the right bracket such *that* or *to*). By using the distance as the feature, it is possible to avoid specifying a cutoff point for the end of a match. The memory-based learning algorithm can adapt to the training data. As discussed in Section 2.1, extraposition is often lexically triggered, so the specific verbs and adjectives in the sentence are important for its classification. For this reason, it is necessary to include information about the surrounding verbs and adjectives. The nearby full verbs

(as opposed to auxiliary and modal verbs) are likely to give the most information, so we add features for the immediately preceding full verb (for **F7**), the following full verb (for **F1**-**F6**), and the following adjective (for **F1-3,7**). The verbs were lemmatized with *morpha* and added as features along with the following adjective.

F10 lemma of immediately preceding full verb
F11 lemma of following full verb within current sentence
F12 following adjective within current sentence

5.2 Cleft *it*

Two patterns are used for cleft *it*:

F13 *it be who/which/that*
F14 *it who/which/that*

As mentioned in the previous section, all verbs in the sentence are lemmatized before matching. Likewise, these features are the distance between *it* and the right bracket. Feature **F14** is used to match a cleft *it* in a phrase with inverted word order.

5.3 Weather/Condition/Time/Place *it*

Ideally, the possible weather predicates could be learned automatically from the following verbs, adjectives, and nouns, but the list is so open that it is better in practice to specify a fixed list. The weather/time/place/condition predicates were taken from the training set and put into a fixed list. Some generalizations were made (e.g., adding the names of all months, weekdays, and seasons), but the list contains mainly the words found in the training set. There are 46 words in the list. As Denber mentioned, WordNet could be used to extend this list. A feature is added for the distance to the nearest weather token.

The following verb lemma feature (**F10**) added for extrapositional *it* is the lemma of the following full verb, but in many cases the verb following weather *it* is the verb *be*, so we also added a binary feature for whether the following verb is *be*.

F15 distance to nearest weather token
F16 whether the following verb is *be*

5.4 Idiomatic *it*

Idioms can be identified by fixed patterns. All verbs in the sentence are lemmatized and the following patterns, all found as idioms in our training data, are used:

if/when it come to	*pull it off*
as it happen	*fall to it*
call it a NOUN	*ask for it*
on the face of it	*be it not for*
have it not been for	*like it or not*

Short idiom patterns such as “cool it” and “watch it” were found to overgeneralize, so only idioms including at least three words were used. A binary feature was added for whether an idiom pattern was matched for the given instance of *it* (**F17**). In addition, two common fixed patterns were included as a separate feature:

it be ... time
it be ... my/X’s turn

F17 whether an idiom pattern was matched
F18 whether an additional fixed pattern was matched

5.5 Additional Restrictions

There are a few additional restrictions on the pattern matches involving length and punctuation. The first restriction is on the distance between the instance of *it* and the right bracket (*that*, *to*, *who*, etc.). On the basis of their corpus, Paice and Husk decided that the right bracket could be at most 27 words away from *it*. Instead of choosing a fixed distance, features based on pattern matches are the distance (number of tokens) between *it* and the right bracket.

The system looks for a pattern match between *it* and the end of the sentence. The end of a sentence is considered to be punctuation matching any of the following: . ; : ? !)] . (Right parenthesis or bracket is only included if a matching left parenthesis or bracket has not been found before it.) If there is anything in paired parentheses in the remainder of the sentence, it is omitted. Quotes are not consistent indicators of a break in a sentence, so they are ignored. If the end of a sentence is not located within 50 tokens, the sentence is truncated at that point and the system looks for the patterns within those tokens.

As Paice and Husk noted, the presence of a single comma or dash between *it* and the right bracket is a good sign that the right bracket is not relevant to whether the instance of *it* is non-referential. When there are either zero or two or more commas or dashes it is difficult to come to any conclusion without more information. Therefore, when the total comma count or total dash count between *it* and the right bracket is one, the pattern match is ignored.

Additionally, unless *it* occurs in an idiom, it is also never the object of a preposition, so there is an additional feature for whether *it* is preceded by a preposition.

F19 whether the previous word is a preposition

Finally, the single preceding and five following simplified part-of-speech tags were also included. The part-of-speech tags were simplified to their first character in the C7 tagset, adverb (R) and negative (X) words were ignored, and only the first instance in a sequence of tokens of the same simplified type (e.g., the first of two consecutive verbs) was included in the set of following tags.

F20-25 surrounding POS tags, simplified

6 Results

Training and testing data were generated from our corpus using the the 25 features described in the previous section. Given Evans's success and the limited amount of training data, we chose to also use TiMBL's k-nearest neighbor algorithm (IB1). In TiMBL, the distance metric can be calculated in a number of ways for each feature. The numeric features use the numeric metric and the remaining features (lemmas, POS tags) use the default overlap metric. Best performance is achieved with gain ratio weighting and the consideration of 2 nearest distances (neighbors). Because of overlap in the features for various types of non-referential *it* and sparse data for cleft, weather, and idiomatic *it*, all types of non-referential *it* were considered at the same time and the output was a binary classification of each instance of *it* as referential or non-referential. The results for our TiMBL classifier (MBL) are shown in Table 7 alongside our results using a decision tree algorithm (DT, described below) and the results from our replication of Evans (2001). All three systems were trained and evaluated with the same data.

	Our MBL Classifier	Our DT Classifier	Repl. of Evans
Accuracy	88%	81%	76%
Precision	82%	82%	57%
Recall	71%	42%	60%

Table 7: Results

Extrapositional	81%
Cleft	45%
Weather	57%
Idiomatic	60%
Referential	94%

Table 8: Recall by Type for MBL Classifier

All three systems perform a binary classification of each instance of *it* as referential or non-referential, but each instance of non-referential *it* was additionally tagged for type, so the recall for each type can be calculated. The recall by type can been seen in Table 8 for our MBL system. Given that the memory-based learning algorithm is using previously seen instances to classify new ones, it makes sense that the most frequent types have the highest recall. As mentioned in Section 2.2, clefts can be difficult to identify.

Decision tree algorithms seem suited to this kind of task and have been used previously, but C4.5 (Quinlan, 1993) decision tree algorithm did not perform as well as TiMBL on our data, compare the TiMBL results (MBL) with the C4.5 results (DT) in Table 7. This may be because the verb and adjective lemma features (**F10**-**F12**) had hundreds of possible values and were not as useful in a decision tree as in the memory-based learning algorithm.

With the addition of more relevant, generalized grammatical patterns, the precision and accuracy have increased significantly, but the same cannot be said for recall. Because many of the patterns are designed to match specific function words as the right bracket, cases where the right bracket is omitted (e.g., extraposed clauses with no overt complementizers, truncated clefts, clefts with reduced relative clauses) are difficult to match. Other problematic cases include sentences with a lot of intervening

material between *it* and the right bracket or simple idioms which cannot be easily differentiated. The results for cleft, weather, and idiomatic *it* may also be due in part to sparse data. When only 2% of the instances of *it* are of a certain type, there are fewer than one hundred training instances, and it can be difficult for the memory-based learning method to be very successful.

7 Conclusion

The accurate classification of *it* as referential or non-referential is important for natural language tasks such as reference resolution (Ng and Cardie, 2002). Through an examination of the types of constructions containing non-referential *it*, we are able to develop a set of detailed grammatical patterns associated with non-referential *it*. In previous rule-based systems, word lists were created for the verbs and adjectives which often occur in these patterns. Such a system can be limited because it is unable to adapt to new texts, but the basic grammatical patterns are still reasonably consistent indicators of non-referential *it*. Given a POS-tagged corpus, the relevant linguistic patterns can be generalized over part-of-speech tags, reducing the dependence on brittle word lists. A machine learning algorithm is able to adapt to new texts and new words, but it is less able to generalize about the linguistic patterns from a small training set. To be able to use our knowledge of relevant linguistic patterns without having to specify lists of words as indicators of certain types of *it*, we developed a machine learning system which incorporates the relevant patterns as features alongside part-of-speech and lexical information. Two short lists are still used to help identify weather *it* and a few idioms. The k-nearest neighbors algorithm from the Tilburg Memory Based Learner is used with 25 features and achieved 88% accuracy, 82% precision, and 71% recall for the binary classification of *it* as referential or non-referential.

Our classifier outperforms previous systems in both accuracy and precision, but recall is still a problem. Many instances of non-referential *it* are difficult to identify because typical clues such as complementizers and relative pronouns can be omitted. Because of this, subordinate and relative clauses cannot be consistently identified given only a POS-tagged corpus. Improvements could be made in the future by integrating chunking or parsing into the pattern-matching features used in the system. This would help in identifying extrapositional and cleft *it*. Knowledge about context beyond the sentence level will be needed to accurately identify certain types of cleft, weather, and idiomatic constructions.

References

L. Burnard, 1995. *Users reference guide for the British National Corpus*. Oxford.

Walter Daelemans, Jakub Zavrel, Ko van der Sloot, and Antal van den Bosch. 2003. TiMBL: Tilburg Memory Based Learner, version 5.0, Reference Guide. ILK Technical Report 03-10. Technical report.

Michel Denber. 1998. Automatic resolution of anaphora in English. Technical report, Imaging Science Divison, Eastman Kodak Co.

Richard Evans. 2001. Applying machine learning toward an automatic classification of It. *Literary and Linguistic Computing*, 16(1):45 – 57.

Rodney D. Huddleston and Geoffrey K. Pullum. 2002. *The Cambridge Grammar of the English Language*. Cambridge University Press, Cambridge.

Shalom Lappin and Herbert J. Leass. 1994. An Algorithm for Pronominal Anaphora Resolution. *Computational Linguistics*, 20(4):535–561.

Guido Minnen, John Caroll, and Darren Pearce. 2001. Applied morphological processing of English. *Natural Language Engineering*, 7(3):207–223.

Vincent Ng and Claire Cardie. 2002. Identifying anaphoric and non-anaphoric noun phrases to improve coreference resolution. *Proceedings of the 19th International Conference on Computational Linguistics (COLING-2002)*.

C. D. Paice and G. D. Husk. 1987. Towards an automatic recognition of anaphoric features in English text; the impersonal pronoun 'it'. *Computer Speech and Language*, 2:109 – 132.

J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Engineering of Syntactic Features for Shallow Semantic Parsing

Alessandro Moschitti[◇] **Bonaventura Coppola**[†‡] **Daniele Pighin**[◇] **Roberto Basili**[◇]

[◇] DISP - University of Rome "Tor Vergata", Rome, Italy
{moschitti, pighin, basili}@info.uniroma2.it

[†] ITC-Irst, [‡] DIT - University of Trento, Povo-Trento, Italy
coppolab@itc.it

Abstract

Recent natural language learning research has shown that structural kernels can be effectively used to induce accurate models of linguistic phenomena.

In this paper, we show that the above properties hold on a novel task related to predicate argument classification. A tree kernel for selecting the subtrees which encodes argument structures is applied. Experiments with Support Vector Machines on large data sets (i.e. the PropBank collection) show that such kernel improves the recognition of argument boundaries.

1 Introduction

The design of features for natural language processing tasks is, in general, a critical problem. The inherent complexity of linguistic phenomena, often characterized by structured data, makes difficult to find effective linear feature representations for the target learning models.

In many cases, the traditional feature selection techniques (Kohavi and Sommerfield, 1995) are not so useful since the critical problem relates to feature generation rather than selection. For example, the design of features for a natural language syntactic parse-tree re-ranking problem (Collins, 2000) cannot be carried out without a deep knowledge about automatic syntactic parsing. The modeling of syntactic/semantic based features should take into account linguistic aspects to detect the interesting context, e.g. the ancestor nodes or the semantic dependencies (Toutanova et al., 2004).

A viable alternative has been proposed in (Collins and Duffy, 2002), where convolution kernels were used to implicitly define a tree substructure space. The selection of the relevant structural features was left to the voted perceptron learning algorithm. Another interesting model for parsing re-ranking based on tree kernel is presented in (Taskar et al., 2004). The good results show that tree kernels are very promising for automatic feature engineering, especially when the available knowledge about the phenomenon is limited.

Along the same line, automatic learning tasks that rely on syntactic information may take advantage of a tree kernel approach. One of such tasks is the automatic boundary detection of predicate arguments of the kind defined in PropBank (Kingsbury and Palmer, 2002). For this purpose, given a predicate p in a sentence s, we can define the notion of *predicate argument spanning trees* ($PAST$s) as those syntactic subtrees of s which *exactly cover* all and only the p's arguments (see Section 4.1). The set of non-spanning trees can be then associated with all the remaining subtrees of s.

An automatic classifier which recognizes the spanning trees can potentially be used to detect the predicate argument boundaries. Unfortunately, the application of such classifier to all possible sentence subtrees would require an exponential execution time. As a consequence, we can use it only to decide for a reduced set of subtrees associated with a corresponding set of candidate boundaries. Notice how these can be detected by previous approaches

Proceedings of the ACL Workshop on Feature Engineering for Machine Learning in NLP, pages 48–56,
Ann Arbor, June 2005.

(e.g. (Pradhan et al., 2004)) in which a traditional boundary classifier (tbc) labels the parse-tree nodes as potential arguments ($\mathcal{PA}$). Such classifiers, generally, are not sensitive to the overall argument structure. On the contrary, a $PAST$ classifier ($past_c$) can consider the overall argument structure encoded in the associated subtree. This is induced by the $\mathcal{PA}$ subsets.

The feature design for the $PAST$ representation is not simple. Tree kernels are a viable alternative that allows the learning algorithm to measure the similarity between two $PAST$s in term of all possible tree substructures.

In this paper, we designed and experimented a boundary classifier for predicate argument labeling based on two phases: (1) a first annotation of potential arguments by using a high recall tbc and (2) a $PAST$ classification step aiming to select the correct substructures associated with potential arguments. Both classifiers are based on Support Vector Machines learning. The $past_c$ uses the tree kernel function defined in (Collins and Duffy, 2002). The results show that the $PAST$ classification can be learned with high accuracy (the f-measure is about 89%) and the impact on the overall boundary detection accuracy is good.

In the remainder of this paper, Section 2 introduces the Semantic Role Labeling problem along with the boundary detection subtask. Section 3 defines the SVMs using the linear kernel and the parse tree kernel for boundary detection. Section 4 describes our boundary detection algorithm. Section 5 shows the preliminary comparative results between the traditional and the two-step boundary detection. Finally, Section 7 summarizes the conclusions.

2 Automated Semantic Role Labeling

One of the largest resources of manually annotated predicate argument structures has been developed in the PropBank (PB) project. The PB corpus contains 300,000 words annotated with predicative information on top of the Penn Treebank 2 Wall Street Journal texts. For any given predicate, the expected arguments are labeled sequentially from *Arg0* to *Arg9*, *ArgA* and *ArgM*. Figure 1 shows an example of the PB predicate annotation of the sentence: `John rented a room in Boston.`

Predicates in PB are only embodied by verbs whereas most of the times *Arg0* is the *subject*, *Arg1* is the *direct object* and *ArgM* indicates *locations*, as in our example.

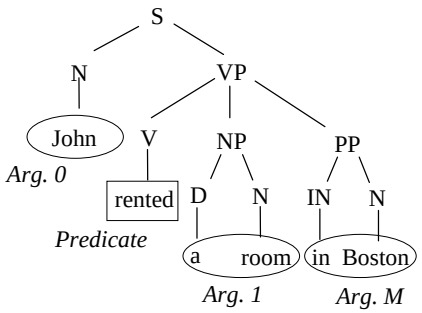

Figure 1: A predicate argument structure in a parse-tree representation.

Several machine learning approaches for automatic predicate argument extraction have been developed, e.g. (Gildea and Jurasfky, 2002; Gildea and Palmer, 2002; Gildea and Hockenmaier, 2003; Pradhan et al., 2004). Their common characteristic is the adoption of feature spaces that model predicate-argument structures in a flat feature representation. In the next section, we present the common parse tree-based approach to this problem.

2.1 Predicate Argument Extraction

Given a sentence in natural language, all the predicates associated with the verbs have to be identified along with their arguments. This problem is usually divided in two subtasks: (a) the detection of the target argument boundaries, i.e. the span of its words in the sentence, and (b) the classification of the argument type, e.g. *Arg0* or *ArgM* in PropBank or *Agent* and *Goal* in FrameNet.

The standard approach to learn both the detection and the classification of predicate arguments is summarized by the following steps:

1. Given a sentence from the *training-set*, generate a full syntactic parse-tree;
2. let $\mathcal{P}$ and $\mathcal{A}$ be the set of predicates and the set of parse-tree nodes (i.e. the potential arguments), respectively;
3. for each pair $< p, a > \in \mathcal{P} \times \mathcal{A}$:
   - extract the feature representation set, $F_{p,a}$;

- if the subtree rooted in a covers exactly the words of one argument of p, put $F_{p,a}$ in T^+ (positive examples), otherwise put it in T^- (negative examples).

For instance, in Figure 1, for each combination of the predicate *rent* with the nodes N, S, VP, V, NP, PP, D or IN the instances $F_{rent,a}$ are generated. In case the node a exactly covers "John", "a room" or "in Boston", it will be a positive instance otherwise it will be a negative one, e.g. $F_{rent,IN}$.

The T^+ and T^- sets are used to train the boundary classifier. To train the multi-class classifier T^+ can be reorganized as positive $T^+_{arg_i}$ and negative $T^-_{arg_i}$ examples for each argument i. In this way, an individual ONE-vs-ALL classifier for each argument i can be trained. We adopted this solution, according to (Pradhan et al., 2004), since it is simple and effective. In the classification phase, given an unseen sentence, all its $F_{p,a}$ are generated and classified by each individual classifier C_i. The argument associated with the maximum among the scores provided by the individual classifiers is eventually selected.

2.2 Standard feature space

The discovery of relevant features is, as usual, a complex task. However, there is a common consensus on the set of basic features. These standard features, firstly proposed in (Gildea and Jurasfky, 2002), refer to unstructured information derived from parse trees, i.e. *Phrase Type*, *Predicate Word*, *Head Word*, *Governing Category*, *Position* and *Voice*. For example, the *Phrase Type* indicates the syntactic type of the phrase labeled as a predicate argument, e.g. NP for *Arg1* in Figure 1. The *Parse Tree Path* contains the path in the parse tree between the predicate and the argument phrase, expressed as a sequence of nonterminal labels linked by direction (up or down) symbols, e.g. V ↑ VP ↓ NP for *Arg1* in Figure 1. The *Predicate Word* is the surface form of the verbal predicate, e.g. *rent* for all arguments.

In the next section we describe the SVM approach and the basic kernel theory for the predicate argument classification.

3 Learning predicate structures via Support Vector Machines

Given a vector space in $\Re^n$ and a set of positive and negative points, SVMs classify vectors according to a separating hyperplane, $H(\vec{x}) = \vec{w} \times \vec{x} + b = 0$, where $\vec{w} \in \Re^n$ and $b \in \Re$ are learned by applying the *Structural Risk Minimization principle* (Vapnik, 1995).

To apply the SVM algorithm to Predicate Argument Classification, we need a function $\phi : \mathcal{F} \rightarrow \Re^n$ to map our features space $\mathcal{F} = \{f_1, .., f_{|\mathcal{F}|}\}$ and our predicate/argument pair representation, $F_{p,a} = F_z$, into $\Re^n$, such that:

$$F_z \rightarrow \phi(F_z) = (\phi_1(F_z), .., \phi_n(F_z))$$

From the kernel theory we have that:

$$H(\vec{x}) = \Big(\sum_{i=1..l} \alpha_i \vec{x}_i \Big) \cdot \vec{x} + b =$$

$$\sum_{i=1..l} \alpha_i \vec{x}_i \cdot \vec{x} + b = \sum_{i=1..l} \alpha_i \phi(F_i) \cdot \phi(F_z) + b.$$

where, $F_i \; \forall i \in \{1, .., l\}$ are the training instances and the product $K(F_i, F_z) = <\phi(F_i) \cdot \phi(F_z)>$ is the kernel function associated with the mapping ϕ.

The simplest mapping that we can apply is $\phi(F_z) = \vec{z} = (z_1, ..., z_n)$ where $z_i = 1$ *if* $f_i \in F_z$ and $z_i = 0$ otherwise, i.e. the characteristic vector of the set F_z with respect to $\mathcal{F}$. If we choose the scalar product as a kernel function we obtain the linear kernel $K_L(F_x, F_z) = \vec{x} \cdot \vec{z}$.

An interesting property is that we do not need to evaluate the ϕ function to compute the above vector. Only the $K(\vec{x}, \vec{z})$ values are in fact required. This allows us to derive efficient classifiers in a huge (possible infinite) feature space, provided that the kernel is processed in an efficient way. This property is also exploited to design convolution kernel like those based on tree structures.

3.1 The tree kernel function

The main idea of the tree kernels is the modeling of a $K_T(T_1, T_2)$ function which computes the number of common substructures between two trees T_1 and T_2.

Given the set of substructures (fragments) $\{f_1, f_2, ..\} = \mathcal{F}$ extracted from all the trees of the training set, we define the indicator function $I_i(n)$

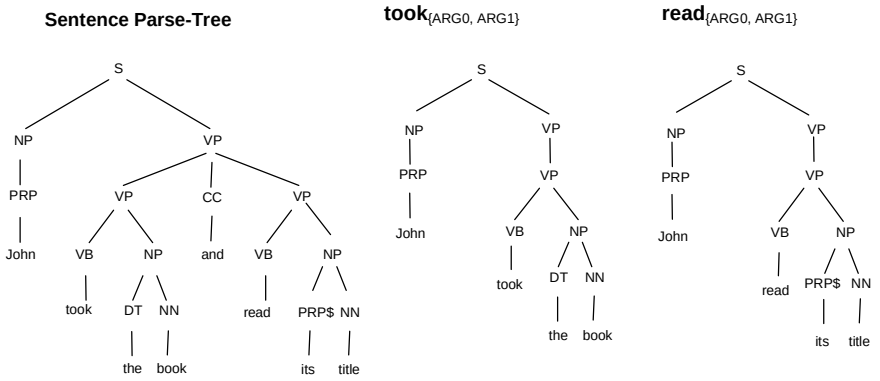

Figure 2: A sentence parse tree with two predicative tree structures ($PAST$s)

which is equal 1 if the target f_i is rooted at node n and 0 otherwise. It follows that:

$$K_T(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) \quad (1)$$

where N_{T_1} and N_{T_2} are the sets of the T_1's and T_2's nodes, respectively and $\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} I_i(n_1) I_i(n_2)$. This latter is equal to the number of common fragments rooted at the n_1 and n_2 nodes. We can compute Δ as follows:

1. if the productions at n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
2. if the productions at n_1 and n_2 are the same, and n_1 and n_2 have only leaf children (i.e. they are pre-terminals symbols) then $\Delta(n_1, n_2) = 1$;
3. if the productions at n_1 and n_2 are the same, and n_1 and n_2 are not pre-terminals then

$$\Delta(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (1 + \Delta(c_{n_1}^j, c_{n_2}^j)) \quad (2)$$

where $nc(n_1)$ is the number of the children of n_1 and c_n^j is the j-th child of the node n. Note that, as the productions are the same, $nc(n_1) = nc(n_2)$.

The above kernel has the drawback of assigning higher weights to larger structures[1]. In order to overcome this problem we scale the relative importance of the tree fragments imposing a parameter λ in conditions 2 and 3 as follows: $\Delta(n_x, n_z) = \lambda$ and $\Delta(n_x, n_z) = \lambda \prod_{j=1}^{nc(n_x)} (1 + \Delta(c_{n_1}^j, c_{n_2}^j))$.

[1]In order to approach this problem and to map similarity scores in the [0,1] range, a normalization in the kernel space, i.e. $K'_T(T_1, T_2) = \frac{K_T(T_1, T_2)}{\sqrt{K_T(T_1, T_1) \times K_T(T_2, T_2)}}$. is always applied

4 Boundary detection via argument spanning

Section 2 has shown that traditional argument boundary classifiers rely only on features extracted from the current potential argument node. In order to take into account a complete argument structure information, the classifier should select a set of parse-tree nodes and consider them as potential arguments of the target predicate. The number of all possible subsets is exponential in the number of the parse-tree nodes of the sentence, thus, we need to cut the search space. For such purpose, a traditional boundary classifier can be applied to select the set of potential arguments $\mathcal{PA}$. The reduced number of $\mathcal{PA}$ subsets can be associated with sentence subtrees which in turn can be classified by using tree kernel functions. These measure if a subtree is *compatible* or not with the subtree of a correct predicate argument structure.

4.1 The Predicate Argument Spanning Trees ($PAST$s)

We consider the predicate argument structures annotated in PropBank along with the corresponding TreeBank data as our object space. Given the target predicate p in a sentence parse tree T and a subset $s = \{n_1, .., n_k\}$ of the T's nodes, N_T, we define as the spanning tree root r the lowest common ancestor of $n_1, .., n_k$. The node spanning tree (NST), p_s is the subtree rooted in r, from which the nodes that are neither ancestors nor descendants of any n_i are removed.

Since predicate arguments are associated with tree nodes, we can define the *predicate argu-*

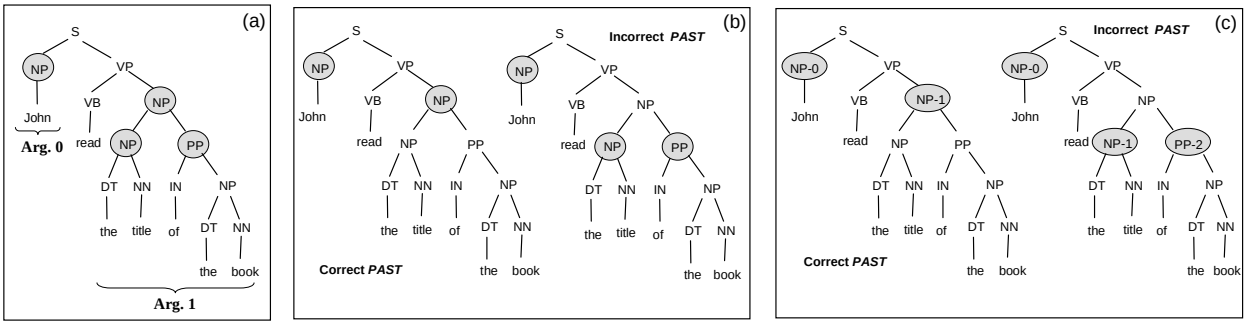

Figure 3: Two-step boundary classifier.

ment spanning tree ($PAST$) of a predicate argument set, $\{a_1, .., a_n\}$, as the NST over such nodes, i.e. $p_{\{a_1,..,a_n\}}$. A $PAST$ corresponds to the *minimal* subparse tree whose leaves are all and only the word sequence compounding the arguments. For example, Figure 2 shows the parse tree of the sentence `"John took the book and read its title"`. $took_{\{ARG_0,ARG_1\}}$ and $read_{\{ARG_0,ARG_1\}}$ are two $PAST$ structures associated with the two predicates *took* and *read*, respectively. All the other NSTs are not valid $PAST$s.

Notice that, labeling $p_s, \forall s \subseteq N_T$ with a $PAST$ classifier ($past_c$) corresponds to solve the boundary problem. The critical points for the application of this strategy are: (1) how to design suitable features for the $PAST$ characterization. This new problem requires a careful linguistic investigation about the significant properties of the argument spanning trees and (2) how to deal with the exponential number of NSTs.

For the first problem, the use of tree kernels over the $PAST$s can be an alternative to the manual features design as the learning machine, (e.g. SVMs) can select the most relevant features from a high dimensional feature space. In other words, we can use Eq. 1 to estimate the similarity between two $PAST$s avoiding to define explicit features. The same idea has been successfully applied to the parse-tree reranking task (Taskar et al., 2004; Collins and Duffy, 2002) and predicate argument classification (Moschitti, 2004).

For the second problem, i.e. the high computational complexity, we can cut the search space by using a traditional boundary classifier (tbc), e.g. (Pradhan et al., 2004), which provides a small set of potential argument nodes. Let $\mathcal{PA}$ be the set of nodes located by tbc as arguments. We may consider the set $\mathcal{P}$ of the NSTs associated with any subset of $\mathcal{PA}$, i.e. $\mathcal{P} = \{p_s : s \subseteq \mathcal{PA}\}$. However, also the classification of $\mathcal{P}$ may be computationally problematic since theoretically there are $|\mathcal{P}| = 2^{|\mathcal{PA}|}$ members.

In order to have a very efficient procedure, we applied $past_c$ to only the $\mathcal{PA}$ sets associated with incorrect $PAST$s. A way to detect such incorrect NSTs is to look for a node pair $<n_1, n_2> \in \mathcal{PA} \times \mathcal{PA}$ of *overlapping nodes*, i.e. n_1 is ancestor of n_2 or viceversa. After we have detected such nodes, we create two node sets $PA_1 = \mathcal{PA} - \{n_1\}$ and $PA_2 = \mathcal{PA} - \{n_2\}$ and classify them with the $past_c$ to select the correct set of argument boundaries. This procedure can be generalized to a set of overlapping nodes O greater than 2 as reported in Appendix 1.

Note that the algorithm selects a maximal set of non-overlapping nodes, i.e. the first that is generated. Additionally, the worst case is rather rare thus the algorithm is very fast on average.

The Figure 3 shows a working example of the multi-stage classifier. In Frame (a), tbc labels as potential arguments (gray color) three overlapping nodes (in Arg.1). The overlap resolution algorithm proposes two solutions (Frame (b)) of which only one is correct. In fact, according to the second solution the propositional phrase "*of the book*" would incorrectly be attached to the verbal predicate, i.e. in contrast with the parse tree. The $past_c$, applied

to the two NSTs, should detect this inconsistency and provide the correct output. Note that, during the learning, we generate the non-overlapping structures in the same way to derive the positive and negative examples.

4.2 Engineering Tree Fragment Features

In the Frame (b) of Figure 3, we show one of the possible cases which $past_c$ should deal with. The critical problem is that the two NSTs are perfectly identical, thus, it is not possible to discern between them using only their parse-tree fragments.

The solution to engineer novel features is to simply add the boundary information provided by the tbc to the NSTs. We mark with a progressive number the phrase type corresponding to an argument node, starting from the leftmost argument. For example, in the first NST of Frame (c), we mark as `NP-0` and `NP-1` the first and second argument nodes whereas in the second NST we have an hypothesis of three arguments on the `NP`, `NP` and `PP` nodes. We trasform them in `NP-0`, `NP-1` and `PP-2`.

This simple modification enables the tree kernel to generate features useful to distinguish between two identical parse trees associated with different argument structures. For example, for the first NST the fragments `[NP-1 [NP][PP]]`, `[NP [DT][NN]]` and `[PP [IN][NP]]` are generated. They do not match anymore with the `[NP-0 [NP][PP]]`, `[NP-1 [DT][NN]]` and `[PP-2 [IN][NP]]` fragments of the second NST.

In order to verify the relevance of our model, the next section provides empirical evidence about the effectiveness of our approach.

5 The Experiments

The experiments were carried out with the SVM-light-TK software available at `http://ai-nlp.info.uniroma2.it/moschitti/` which encodes the tree kernels in the SVM-light software (Joachims, 1999). For tbc, we used the linear kernel with a regularization parameter (option `-c`) equal to 1 and a cost-factor (option `-j`) of 10 to have a higher Recall. For the $past_c$ we used $\lambda = 0.4$ (see (Moschitti, 2004)).

As referring dataset, we used the PropBank corpora available at `www.cis.upenn.edu/~ace`, along with the Penn TreeBank 2 (`www.cis.upenn.edu/~treebank`) (Marcus et al., 1993). This corpus contains about 53,700 sentences and a fixed split between training and testing which has been used in other researches, e.g. (Pradhan et al., 2004; Gildea and Palmer, 2002). We did not include continuation and co-referring arguments in our experiments.

We used sections from 02 to 07 (54,443 argument nodes and 1,343,046 non-argument nodes) to train the traditional boundary classifier (tbc). Then, we applied it to classify the sections from 08 to 21 (125,443 argument nodes vs. 3,010,673 non-argument nodes). As results we obtained 2,988 NSTs containing at least an overlapping node pair out of the total 65,212 predicate structures (according to the tbc decisions). From the 2,988 overlapping structures we extracted 3,624 positive and 4,461 negative NSTs, that we used to train the $past_c$.

The performance was evaluated with the F_1 measure[2] over the section 23. This contains 10,406 argument nodes out of 249,879 parse tree nodes. By applying the tbc classifier we derived 235 overlapping $NSTs$, from which we extracted 204 $PAST$s and 385 incorrect predicate argument structures. On such test data, the performance of $past_c$ was very high, i.e. 87.08% in Precision and 89.22% in Recall.

Using the $past_c$ we removed from the tbc the $\mathcal{PA}$ that cause overlaps. To measure the impact on the boundary identification performance, we compared it with three different boundary classification baselines:

- tbc: overlaps are ignored and no decision is taken. This provides an upper bound for the recall as no potential argument is rejected for later labeling. Notice that, in presence of overlapping nodes, the sentence cannot be annotated correctly.

- RND: one among the non-overlapping structures with maximal number of arguments is randomly selected.

[2] F_1 assigns equal importance to Precision P and Recall R, i.e. $F_1 = \frac{2P \times R}{P+R}$.

	tbc			tbc+RND			tbc+Heu			tbc+$past_c$		
	P	R	F	P	R	F	P	R	F	P	R	F
All Struct.	92.21	98.76	95.37	93.55	97.31	95.39	92.96	97.32	95.10	94.40	98.42	96.36
Overl. Struct.	98.29	65.8	78.83	74.00	72.27	73.13	68.12	75.23	71.50	89.61	92.68	91.11

Table 1: Two-steps boundary classification performance using the traditional boundary classifier tbc, the random selection of non-overlapping structures (RND), the heuristic to select the most suitable non-overlapping node set (Heu) and the predicate argument spanning tree classifier ($past_c$).

- Heu (heuristic): one of the NSTs which contain the nodes with the lowest overlapping score is chosen. This score counts the number of overlapping node pairs in the NST. For example, in Figure 3.(a) we have a NP that overlaps with two nodes NP and PP, thus it is assigned a score of 2.

The third row of Table 1 shows the results of tbc, $tbc + RND$, $tbc + Heu$ and $tbc + past_c$ in the columns 2,3,4 and 5, respectively. We note that:

- The tbc F_1 is slightly higher than the result obtained in (Pradhan et al., 2004), i.e. 95.37% vs. 93.8% on same training/testing conditions, i.e. (same PropBank version, same training and testing split and same machine learning algorithm). This is explained by the fact that we did not include the continuations and the co-referring arguments that are more difficult to detect.

- Both RND and Heu do not improve the tbc result. This can be explained by observing that in the 50% of the cases a correct node is removed.

- When, to select the correct node, the $past_c$ is used, the F_1 increases of 1.49%, i.e. (96.86 vs. 95.37). This is a very good result considering that to increase the very high baseline of tbc is hard.

In order to give a fairer evaluation of our approach we tested the above classifiers on the overlapping structures only, i.e. we measured the $past_c$ improvement on all and only the structures that required its application. Such reduced test set contains 642 argument nodes and 15,408 non-argument nodes. The fourth row of Table 1 reports the classifier performance on such task. We note that the $past_c$ improves the other heuristics of about 20%.

6 Related Work

Recently, many kernels for natural language applications have been designed. In what follows, we highlight their difference and properties.

The tree kernel used in this article was proposed in (Collins and Duffy, 2002) for syntactic parsing reranking. It was experimented with the Voted Perceptron and was shown to improve the syntactic parsing. A refinement of such technique was presented in (Taskar et al., 2004). The substructures produced by the proposed tree kernel were bound to local properties of the target parse tree and more lexical information was added to the overall kernel function.

In (Zelenko et al., 2003), two kernels over syntactic shallow parser structures were devised for the extraction of linguistic relations, e.g. *person-affiliation*. To measure the similarity between two nodes, the *contiguous string kernel* and the *sparse string kernel* (Lodhi et al., 2000) were used. The former can be reduced to the contiguous substring kernel whereas the latter can be transformed in the non-contiguous string kernel. The high running time complexity, caused by the general form of the fragments, limited the experiments on data-set of just 200 news items.

In (Cumby and Roth, 2003), it is proposed a description language that models feature descriptors to generate different feature type. The descriptors, which are quantified logical prepositions, are instantiated by means of a *concept graph* which encodes the structural data. In the case of relation extraction the *concept graph* is associated with a syntactic shallow parse and the extracted propositional features express fragments of a such syntactic structure. The experiments over the named entity class categorization show that when the description language selects an adequate set of tree fragments the Voted Perceptron algorithm increases its classification accuracy.

In (Culotta and Sorensen, 2004) a dependency

tree kernel is used to detect the Named Entity classes in natural language texts. The major novelty was the combination of the contiguous and sparse kernels with the word kernel. The results show that the contiguous outperforms the sparse kernel and the *bag-of-words*.

7 Conclusions

The feature design for new natural language learning tasks is difficult. We can take advantage from the kernel methods to model our intuitive knowledge about the target linguistic phenomenon. In this paper we have shown that we can exploit the properties of tree kernels to engineer syntactic features for the predicate argument boundary detection task.

Preliminary results on gold standard trees suggest that (1) the information related to the whole predicate argument structure is important and (2) tree kernel can be used to generate syntactic features.

In the future, we would like to use an approach similar to the $PAST$ classifier on parses provided by different parsing models to detect boundary and to classify semantic role more accurately .

Acknowledgements

We wish to thank Ana-Maria Giuglea for her help in the design and implementation of the basic Semantic Role Labeling system that we used in the experiments.

References

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL02*.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of ICML 2000*.

Aron Culotta and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 423–429, Barcelona, Spain, July.

Chad Cumby and Dan Roth. 2003. Kernel methods for relational learning. In *Proceedings of the Twentieth International Conference (ICML 2003)*, Washington, DC, USA.

Daniel Gildea and Julia Hockenmaier. 2003. Identifying semantic roles using combinatory categorial grammar. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, Sapporo, Japan.

Daniel Gildea and Daniel Jurasfky. 2002. Automatic labeling of semantic roles. *Computational Linguistic*, 28(3):496–530.

Daniel Gildea and Martha Palmer. 2002. The necessity of parsing for predicate argument recognition. In *Proceedings of the 40th Annual Conference of the Association for Computational Linguistics (ACL-02)*, Philadelphia, PA, USA.

T. Joachims. 1999. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*.

Paul Kingsbury and Martha Palmer. 2002. From Treebank to PropBank. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC-2002)*, Las Palmas, Spain.

Ron Kohavi and Dan Sommerfield. 1995. Feature subset selection using the wrapper model: Overfitting and dynamic search space topology. In *The First International Conference on Knowledge Discovery and Data Mining*, pages 192–197. AAAI Press, Menlo Park, California, August. Journal version in AIJ.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Christopher Watkins. 2000. Text classification using string kernels. In *NIPS*, pages 563–569.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19:313–330.

Alessandro Moschitti. 2004. A study on convolution kernels for shallow semantic parsing. In *proceedings of the 42th Conference on Association for Computational Linguistic (ACL-2004)*, Barcelona, Spain.

Sameer Pradhan, Kadri Hacioglu, Valeri Krugler, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2005. Support vector learning for semantic argument classification. *to appear in Machine Learning Journal*.

Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 1–8, Barcelona, Spain, July. Association for Computational Linguistics.

Kristina Toutanova, Penka Markova, and Christopher D. Manning. 2004. The leaf projection path view of parse trees: Exploring string kernels for hpsg parse selection. In *Proceedings of EMNLP 2004*.

V. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer.

D. Zelenko, C. Aone, and A. Richardella. 2003. Kernel methods for relation extraction. *Journal of Machine Learning Research.*

Appendix 1: Generalized Boundary Selection Algorithm

Let O be the set of overlapping nodes of $\mathcal{PA}$, and NO the set of non overlapping nodes of $\mathcal{PA}$.
Let $subs^{(-1)}(A) = \{B | B \in 2^A, |B| = |A| - 1\}$.
Let $\hat{O} = subs^{(-1)}(O)$.

while($true$)
begin

1. $\mathcal{H} = \emptyset$
2. $\forall o \in \hat{O}$:
   (a) **If** o does not include any overlapping node pair
   then $\mathcal{H} = \mathcal{H} \cup \{o\}$
3. **If** $\mathcal{H} \neq \emptyset$ **then**:
   (a) Let $\hat{s}$ =$argmax_{o \in H}\ past_c(p_{NO \cup o})$, where $p_{NO \cup o}$ represents the node spanning tree compatible with o, and the $past_c(p_{NO \cup o})$ is the score provided by the $PAST$ SVM categorizer on it
   (b) **If** $past_c(\hat{s}) > 0$ **then RETURN**($\hat{s}$)
4. **If** $\hat{O} = \{\emptyset\}$ **then RETURN**(NO)
5. **Else**:
   (a) $\hat{O} = \hat{O} - \mathcal{H}$
   (b) $\hat{O} = \bigcup_{o \in \hat{O}} subs^{(-1)}(o)$

end

Automatic identification of sentiment vocabulary: exploiting low association with known sentiment terms

Michael Gamon
Natural Language Processing Group
Microsoft Research
mgamon@microsoft.com

Anthony Aue
Natural Language Processing Group
Microsoft Research
anthaue@microsoft.com

Abstract

We describe an extension to the technique for the automatic identification and labeling of sentiment terms described in Turney (2002) and Turney and Littman (2002). Their basic assumption is that sentiment terms of similar orientation tend to co-occur at the document level. We add a second assumption, namely that sentiment terms of opposite orientation tend **not** to co-occur at the sentence level. This additional assumption allows us to identify sentiment-bearing terms very reliably. We then use these newly identified terms in various scenarios for the sentiment classification of sentences. We show that our approach outperforms Turney's original approach. Combining our approach with a Naive Bayes bootstrapping method yields a further small improvement of classifier performance. We finally compare our results to precision and recall figures that can be obtained on the same data set with labeled data.

1 Introduction

The field of sentiment classification has received considerable attention from researchers in recent years (Pang and Lee 2002, Pang et al. 2004, Turney 2002, Turney and Littman 2002, Wiebe et al. 2001, Bai et al. 2004, Yu and Hatzivassiloglou 2003 and many others). The identification and classification of sentiment constitutes a problem that is orthogonal to the usual task of text classification. Whereas in traditional text classification the focus is on topic identification, in sentiment classification the focus is on the assessment of the writer's sentiment toward the topic.

Movie and product reviews have been the main focus of many of the recent studies in this area (Pang and Lee 2002, Pang et al. 2004, Turney 2002, Turney and Littman 2002). Typically, these reviews are classified at the document level, and the class labels are "positive" and "negative". In this work, in contrast, we narrow the scope of investigation to the sentence level and expand the set of labels, making a threefold distinction between "positive", "neutral", and "negative". The narrowing of scope is motivated by the fact that for realistic text mining on customer feedback, the document level is too coarse, as described in Gamon et al. (2005). The expansion of the label set is also motivated by real-world concerns; while it is a given that review text expresses positive or negative sentiment, in many cases it is necessary to also identify the cases that don't carry strong expressions of sentiment at all.

Traditional approaches to text classification require large amounts of labeled training data. Acquisition of such data can be costly and time-consuming. Due to the highly domain-specific nature of the sentiment classification task, moving from one domain to another typically requires the acquisition of a new set of training data. For this reason, unsupervised or very weakly supervised methods for sentiment classification are especially

Proceedings of the ACL Workshop on Feature Engineering for Machine Learning in NLP, pages 57–64,
Ann Arbor, June 2005.

desirable.[1] Our focus, therefore, is on methods that require very little data annotation.

We describe a method to automatically identify the sentiment vocabulary in a domain. This method rests on three special properties of the sentiment domain:

1. the presence of certain words can serve as a proxy for the class label
2. sentiment terms of *similar* orientation tend to co-occur
3. sentiment terms of *opposite* orientation tend to not co-occur at the sentence level.

Turney (2002) and Turney and Littman (2002) exploit the first two generalizations for unsupervised sentiment classification of movie reviews. They use the two terms *excellent* and *poor* as seed terms to determine the semantic orientation of other terms. These seed terms can be viewed as proxies for the class labels "positive" and "negative", allowing for the exploitation of otherwise unlabeled data: Terms that tend to co-occur with *excellent* in documents tend to be of positive orientation, and vice versa for *poor*. Turney (2002) starts from a small (2 word) set of terms with known orientation (*excellent* and *poor*). Given a set of terms with unknown sentiment orientation, Turney (2002) then uses the PMI-IR algorithm (Turney 2001) to issue queries to the web and determine, for each of these terms, its pointwise mutual information (PMI) with the two seed words across a large set of documents. Term candidates are constrained to be adjectives, which tend to be the strongest bearers of sentiment. The sentiment orientation (SO) of a term is then determined by the difference between its association (PMI) with the positive seed term *excellent* and its association with the negative seed term *poor*. The resulting list of terms and associated sentiment orientations can then be used to implement a classifier: semantic orientation of the terms in a document of unknown sentiment is added up, and if the overall score is positive, the document is classified as being of positive sentiment, otherwise it is classified as negative.

Yu and Hatzivassiloglou (2003) extend this approach by (1) applying it at the sentence level (instead of the document-level), (2) taking into account non-adjectival parts-of-speech, and (3) using larger sets of seed words. Their classification goal also differs from Turney's: it is to distinguish opinion sentences from factual statements.

Turney et al.'s approach is based on the assumption that sentiment terms of similar orientation tend to co-occur in documents. Our approach takes advantage of a second assumption: At the sentence level, sentiment terms of opposite orientation tend *not* to co-occur. This is, of course, an assumption that will only hold in general, with exceptions. Basically, the assumption is that sentences of the following form:

I dislike X.
I really like X.

are more frequent than "mixed sentiment" sentences such as

I dislike X but I really like Y.

It has been our experience that this generalization does hold often enough to be useful.

We propose to utilize this assumption to identify a set of sentiment terms in a domain. We select the terms that have the lowest PMI scores on the sentence level with respect to a set of manually selected seed words. If our assumption about low association at the sentence level is correct, this set of low-scoring terms will be particularly rich in sentiment terms. We can then use this newly identified set to:

(1) use Turney's method to find the orientation for the terms and employ the terms and their scores in a classifier, and
(2) use Turney's method to find the orientation for the terms and add the new terms as additional seed terms for a second iteration

As opposed to Turney (2002), we do not use the web as a resource to find associations, rather we apply the method directly to in-domain data. This has the disadvantage of not being able to apply the classification to any arbitrary domain. It is worth noting, however, that even in Turney (2002) the choice of seed words is explicitly motivated by domain properties of movie reviews.

In the remainder of the paper we will describe results from various experiments based on this assumption. We also show how we can combine this method with a Naive Bayes bootstrapping approach that takes further advantage of the unlabeled data (Nigam et al. 2000).

[1] For domain-specificity of sentiment classification see Engström (2004) and Aue and Gamon (2005).

2 Data

For our experiments we used a set of car reviews from the MSN Autos web site. The data consist of 406,818 customer car reviews written over a four-year period. Aside from filtering out examples containing profanity, the data was not edited. The reviews range in length from a single sentence (56% of all cases) to 50 sentences (a single review). Less than 1% of reviews contain ten or more sentences. There are almost 900,000 sentences in total. When customers submitted reviews to the website, they were asked for a recommendation on a scale of 1 (negative) to 10 (positive). The average score was very high, at 8.3, yielding a strong skew in favor of positive class labels. We annotated a randomly-selected sample of 3,000 sentences for sentiment. Each sentence was viewed in isolation and classified as positive, negative or neutral. The neutral category was applied to sentences with no discernible sentiment, as well as to sentences that expressed both positive and negative sentiment. Three annotators had pair-wise agreement scores (Cohen's Kappa score, Cohen 1960) of 70.10%, 71.78% and 79.93%, suggesting that the task of sentiment classification on the sentence level is feasible but difficult even for people. This set of data was split into a development test set of 400 sentences and a blind test set of 2600 sentences.

Sentences are represented as vectors of binary unigram features. The total number of observed unigram features is 72988. In order to restrict the number of features to a manageable size, we disregard features that occur less than 10 times in the corpus. With this restriction we obtain a reduced feature set of 13317 features.

3 Experimental Setup

Our experiments were performed as follows: We started with a small set of manually-selected and annotated seed terms. We used 4 positive and 6 negative seed terms. We decided to use a few more negative seed words because of the inherent positive skew in the data that makes the identification of negative sentences particularly hard. The terms we used are:

positive:	**negative:**
good	bad
excellent	lousy
love	terrible
happy	hate
	suck
	unreliable

There was no tuning of the set of initial seed terms; the 10 words were originally chosen intuitively, as words that we observed frequently when manually inspecting the data.

We then used these seed terms in two basic ways: (1) We used them as seeds for a Turney-style determination of the semantic orientation of words in the corpus (semantic orientation, or SO method). As mentioned above, this process is based on the assumption that terms of similar orientation tend to co-occur. (2) We used them to mine sentiment vocabulary from the unlabeled data using the additional assumption that sentiment terms of opposite orientation tend not to co-occur at the sentence level (sentiment mining, or SM method). This method yields a set of sentiment terms, but no orientation for that set of terms. We continue by using the SO method to find the semantic orientation for this set of sentiment terms, effectively using SM as a feature selection method for sentiment terminology.

Pseudo-code for the SO and SM approaches is provided in Figure 1 and Figure 2. As a first step for both SO and SM methods (not shown in the pseudocode), PMI needs to be calculated for each pair (f, s) of feature *f* and seed word *s* over the collection of feature vectors.

Semantic Orientation (SO) method to find semantic orientation for a set of features F, given a set of feature vectors V:

```
FOREACH feature f in F:
    FOREACH positive seed word spos
        PosScore(f) = PosScore(f)+PMI(f, spos)
    FOREACH negative seed word sneg
        NegScore(f)+PMI(f, sneg)
    normalize scores by number of positive/negative
    seed features:
        PosScore(f) = PosScore(f)/number of
        positive seed features
        NegScore(f) = NegScore(f)/number of
        negative seed features
    calculate overall semantic orientation (SO) for f:
        SO = PosScore(f)-NegScore(f)
```

Figure 1: SO method for determining semantic orientation

Mining for sentiment terms (SM) in a set of features F:

```
FOREACH feature f in F:
        FOREACH seed word s
                IF PMI(f, s) < Score(f):
                        Score(f) = PMI(f, s)

Select the top n% of features f that have the lowest Score(f)
```

Figure 2: SM method for mining sentiment terms

In the first scenario (using straightforward SO), features *F* range over all observed features in the data (modulo the aforementioned count cutoff of 10). In the second scenario (SM + SO), features *F* range over the *n*% of features with the lowest PMI scores with respect to any of the seed words that were identified using the sentiment mining technique in Figure 2.

The result of both SO and SM+SO is a list of unigram features which have an associated semantic orientation score, indicating their sentiment orientation: the higher the score, the more "positive" a term, and vice versa.

This list of features and associated scores can be used to construct a simple classifier: for each sentence with unknown sentiment, we take the sum of the semantic orientation scores for all of the unigrams in that sentence. This overall score determines the classification of the sentence as "positive", "neutral" or "negative" as shown in Figure 3.

Scoring and classifying sentence vectors:

```
(1) assigning a sentence score:
FOREACH feature f in sentence vector v:
        Score(v) = Score(v) + SO(f)

(2) assigning a class label based on the sentence score:
IF Score(v) > threshold1:
        Class(v) = "positive"
ELSE IF Score(v) < threshold1 AND Score(v) > threshold2:
        Class(v) = "neutral"
ELSE
        Class(v) = "negative"
```

Figure 3: Using SO scores for sentence scoring and classification

The two thresholds used in classification need to be determined empirically by taking the distribution of class values in the corpus into account. For our experiments we simply took the distribution of class labels in the 400 sentence development test set as an approximation of the overall class label distribution: we determined that distribution to be 15.5% for negative sentences, 21.5% for neutral sentences, and 63.0% for positive sentences. Scores for all sentence vectors in the corpus are then collected using the scoring part of the algorithm in Figure 3. The scores are sorted and the thresholds are determined as the cutoffs for the top 63% and bottom 15.5% of scores respectively.

4 Results

4.1. Comparing SO and SM+SO

In our first set of experiments we manipulated the following parameters:

1. the choice of SO or SM+SO method
2. the choice of *n* when selecting the *n*% semantic terms with lowest PMI score in the SM method

The tables below show the results of classifying sentence vectors using the unigram features and associated scores produced by SO and SO+SM. We used the 2,600-sentence manually-annotated test set described previously to establish these numbers. Since the data exhibit a strong skew in favor of the positive class label, we measure performance not in terms of accuracy but in terms of average precision and recall across the three class labels, as suggested in (Manning and Schütze 2002).

	Avg precision	Avg recall
SO	0.4481	0.4511

Table 1: Using the SO approach.

Table 1 shows results of using the SO method on the data. Table 2 presents the results of combining the SM and SO methods for different values of *n*. The best results are shown in boldface.

As a comparison between Table 1 and Table 2 shows, the highest average precision and recall scores were obtained by combining the SM and SO methods. Using SM as a feature selection mechanism also reduces the number of features significantly. While the SO method employed on sentence-level vectors uses 13,000 features, the best-performing SM+SO combination uses only 20% of this feature set, indicating that SM is indeed effective in selecting the most important sentiment-bearing terms.

We also determined that the positive impact of SM is not just a matter of reducing the number of features. If SO - without the SM feature selection step - is reduced to a comparable number of features by taking the top features according to absolute score, average precision is at 0.4445 and average recall at 0.4464.

	N=10		N=20		N=30		N=40		N=50	
	Avg prec	Avg rec	Avg prec	Avg rec	Avg prec	Avg rec	Avg prec	Avg rec	Avg prec	Avg rec
SM+SO SO from document level	0.4351	0.4377	**0.4568**	**0.4605**	0.4528	0.4557	0.4457	0.4478	0.4451	0.4475

Table 2: combining SM and SO.

Sentiment terms in top 100 SM terms	**Sentiment terms in top 100 SO terms**
excellent, terrible, broke, junk, alright, bargain, grin, highest, exceptional, exceeded, horrible, loved, waste, ok, death, leaking, outstanding, cracked, rebate, warped, hooked, sorry, refuses, excellant, satisfying, died, biggest, competitive, delight, avoid, awful, garbage, loud, okay, competent, upscale, dated, mistake, sucks, superior, high, kill, neither	excellent, happy, stylish, sporty, smooth, love, quiet, overall, pleased, plenty, dependable, solid, roomy, safe, good, easy, smaller, luxury, comfortable, style, loaded, space, classy, handling, joy, small, comfort, size, perfect, performance, room, choice, recommended, package, compliments, awesome, unique, fun, holds, comfortably, extremely, value, free, satisfied, little, recommend, limited, great, pleasure
Non sentiment terms in top 100 SM terms	**Non sentiment terms in top 100 SO terms**
alternative, wont, below, surprisingly, maintained, choosing, comparing, legal, vibration, seemed, claim, demands, assistance, knew, engineering, accelleration, ended, salesperson, performed, started, midsize, site, gonna, lets, plugs, industry, alternator, month, told, vette, 180, powertrain, write, mos, walk, causing, lift, es, segment, $250, 300m, wanna, february, mod, $50, nhtsa, suburbans, manufactured, tiburon, $10, f150, 5000, posted, tt, him, saw, jan,	condition, very, handles, milage, definitely, definately, far, drives, shape, color, price, provides, options, driving, rides, sports, heated, ride, sport, forward, expected, fairly, anyone, test, fits, storage, range, family, sedan, trunk, young, weve, black, college, suv, midsize, coupe, 30, shopping, kids, player, saturn, bose, truck, town, am, leather, stereo, car, husband

Table 3: the top 100 terms identified by SM and SO

Table 3 shows the top 100 terms that were identified by each SM and SO methods. The terms are categorized into sentiment-bearing and non-sentiment bearing terms by human judgment. The two sets seem to differ in both strength and orientation of the identified terms. The SM-identified words have a higher density of negative terms (22 out of 43 versus 2 out of 49 for the SO-identified terms). The SM-identified terms also express sentiment more strongly, but this conclusion is more tentative since it may be a consequence of the higher density of negative terms.

4.2. Multiple iterations: increasing the number of seed features by SM+SO

In a second set of experiments, we assessed the question of whether it is possible to use multiple iterations of the SM+SO method to gradually build the list of seed words. We do this by adding the top n% of features selected by SM, along with their orientation as determined by SO, to the initial set of seed words. The procedure for this round of experiments is as follows:

- take the top n% of features identified by SM (we used n=1 for the reported re-

sults, since preliminary experiments with other values for n did not improve results)

- perform SO for these features to determine their orientation
- take the top 15.5% negative and top 63% positive (according to class label distribution in the development test set) of the features and add them as negative/positive seed features respectively

This iteration increases the number of seed features from the original 10 manually-selected features to a total of 111 seed features.

With this enhanced set of seed features we then re-ran a subset of the experiments in Table 2. Results are shown in Table 4. Increasing the number of seed features through the SM feature selection method increases precision and recall by several percentage points. In particular, precision and recall for negative sentences are boosted.

	Avg precision	Avg recall
SM + SO, n=10, SO from document vectors	0.4826	0.48.76
SM + SO, n=30, SO from document vectors	**0.4957**	**0.4995**
SM + SO, n=50, SO from document vectors	0.4914	0.4952

Table 4: Using 2 iterations to increase the seed feature set

We also confirmed that these results are truly attributable to the use of the SM method for the first iteration. If we take an equivalent number of features with strongest semantic orientation according to the SO method and add them to the list of seed features, our results degrade significantly (the resulting classifier performance is significantly different at the 99.9% level as established by the McNemar test). This is further evidence that SM is indeed an effective method for selecting sentiment terms.

4.3. Using the SO classifier to bootstrap a Naive Bayes classifier

In a third set of experiments, we tried to improve on the results of the SO classifier by combining it with the bootstrapping approach described in (Nigam et al. 2000). The basic idea here is to use the SO classifier to label a subset of the data D_L. This labeled subset of the data is then used to bootstrap a Naive Bayes (NB) classifier on the remaining unlabeled data D_U using the Expectation Maximization (EM) algorithm:

(1) An initial naive Bayes classifier with parameters θ is trained on the documents in D_L.
(2) This initial classifier is used to estimate a probability distribution over all classes for each of the documents in D_U. (E-Step)
(3) The labeled and unlabeled data are then used to estimate parameters for a new classifier. (M-Step)

Steps 2 and 3 are repeated until convergence is achieved when the difference in the joint probability of the data and the parameters falls below the configurable threshold ε between iterations. Another free parameter, λ, can be used to control how much weight is given to the unlabeled data.

For our experiments we used classifiers from the best SM+SO combination (2 iterations at n=30) from Table 4 above to label 30% of the total data. Table 5 shows the average precision and recall numbers for the converged NB classifier.[2] In addition to improving average precision and recall, the resulting classifier also has the advantage of producing class probabilities instead of simple scores.[3]

	Avg precision	Avg recall
Bootstrapped NB classifier	0.5167	0.52

Table 5: Results obtained by bootstrapping a NB classifier

4.4. Results from supervised learning: using small sets of labeled data

Given infinite resources, we can always annotate enough data to train a classifier using a supervised algorithm that will outperform unsupervised or weakly-supervised methods. Which approach to take depends entirely on how much time and money are available and on the accuracy requirements for the task at hand.

[2] In this experiment, λ was set to 0.1 and ε was set to 0.05.
[3] We also experimented with labeling the whole data set with the best of our SO score classifiers, and then training a linear Support Vector Machine classifier on the data. The results were considerably worse than any of the reported numbers, so they are not included in this paper.

To help situate the precision and recall numbers presented in the tables above, we trained Support Vector Machines (SVMs) using small amounts of labeled data. SVMs were trained with 500, 1000, 2000, and 2500 labeled sentences. Annotating 2500 sentences represents approximately eight person-hours of work. The results can be found in Table 5. We were pleasantly surprised at how well the unsupervised classifiers described above perform in comparison to state-of-the-art supervised methods (albeit trained on small amounts of data).

Labeled examples	Avg. Precision	Avg. Recall
500	.4878	.4967
1000	.5161	.5105
2000	.5297	.5256
2500	.5017	.5083

Table 6: Average precision and recall for SVMs for small numbers of labeled examples

4.5. Results on the movie domain

We also performed a small set of experiments on the movie domain using Pang and Lee's 2004 data set. This set consists of 2000 reviews, 1000 each of very positive and very negative reviews. Since this data set is balanced and the task is only a two-way classification between positive and negative reviews, we only report accuracy numbers here.

	accuracy	Training data
Turney (2002)	66%	unsupervised
Pang & Lee (2004)	87.15%	supervised
Aue & Gamon (2005)	91.4%	supervised
SO	73.95%	unsupervised
SM+SO to increase seed words, then SO	74.85%	weakly supervised

Table 7: Classification accuracy on the movie review domain

Turney (2002) achieves 66% accuracy on the movie review domain using the PMI-IR algorithm to gather association scores from the web. Pang and Lee (2004) report 87.15% accuracy using a unigram-based SVM classifier combined with subjectivity detection. Aue and Gamon (2005) use a simple linear SVM classifier based on unigrams, combined with LLR-based feature reduction, to achieve 91.4% accuracy. Using the Turney SO method on in-domain data instead of web data achieves 73.95% accuracy (using the same two seed words that Turney does). Using one iteration of SM+SO to increase the number of seed words, followed by finding SO scores for all words with respect to the enhanced seed word set, yields a slightly higher accuracy of 74.85%. With additional parameter tuning, this number can be pushed to 76.4%, at which point we achieve statistical significance at the 0.95 level according to the McNemar test, indicating that there is more room here for improvement. Any reduction of the number of overall features in this domain leads to decreased accuracy, contrary to what we observed in the car review domain. We attribute this observation to the smaller data set.

5 Discussion

5.1 A note on statistical significance

We used the McNemar test to assess whether two classifiers are performing significantly differently. This test establishes whether the accuracy of two classifiers differs significantly - it does not guarantee significance for precision and recall differences. For the latter, other tests have been proposed (e.g. Chinchor 1995), but time constraints prohibited us from implementing any of those more computationally costly tests.

For the results presented in the previous sections the McNemar test established statistical significance at the 0.99 level over baseline (i.e. the SO results in Table 1) for the multiple iterations results (Table 4) and the bootstrapping approach (Table 5), but not for the SM+SO approach (Table 2).

5.2 Future work

This exploratory set of experiments indicates a number of interesting directions for future work. A shortcoming of the present work is the manual tuning of cutoff parameters. This problem could be alleviated in at least two possible ways:

First, using a general combination of the ranking of terms according to SM and SO. In other words, calculate the semantic weight of a term as a combination of SO and its rank in the SM scores.

Secondly, following a suggestion by an anonymous reviewer, the Naive Bayes bootstrapping approach could be used in a feedback loop to inform the SO score estimation in the absence of a manually annotated parameter tuning set.

5.3 Summary

Our results demonstrate that the SM method can serve as a valid tool to mine sentiment-rich vocabulary in a domain. SM will yield a list of terms that are likely to have a strong sentiment orientation. SO can then be used to find the polarity for the selected features by association with the sentiment terms of known polarity in the seed word list. Performing this process iteratively by first enhancing the set of seed words through SM+SO yields the best results. While this approach does not compare to the results that can be achieved by supervised learning with large amounts of labeled data, it does improve on results obtained by using SO alone.

We believe that this result is relevant in two respects. First, by improving average precision and recall on the classification task, we move closer to the goal of unsupervised sentiment classification. This is a very important goal in itself given the need for "out of the box" sentiment techniques in business intelligence and the notorious difficulty of rapidly adapting to a new domain (Engström 2004, Aue and Gamon 2005). Second, the exploratory results reported here may indicate a general source of information for feature selection in natural language tasks: features that have a tendency to be in complementary distribution (especially in smaller linguistic units such as sentences) may often form a class that shares certain properties. In other words, it is not only the strong association scores that should be exploited but also the particularly weak (negative) associations.

References

Anthony Aue and Michael Gamon (2005): "Customizing Sentiment Classifiers to a New Domain: A Case Study. Under review.

Xue Bai, Rema Padman, and Edoardo Airoldi. (2004). Sentiment Extraction from Unstructured Text Using Tabu Search-Enhanced Markov Blanket. In: Proceedings of the International Workshop on Mining for and from the Semantic Web (MSW 2004), pp 24-35.

Nancy A. Chinchor (1995): Statistical significance of MUC-6 results. Proceedings of the Sixth Message Understanding Conference, pp. 39-44.

J. Cohen (1960): "A coefficient of agreement for nominal scales." In: Educational and Psychological measurements 20, pp. 37–46

Charlotta Engström. 2004. *Topic dependence in Sentiment Classification.* MPhil thesis, University of Cambridge.

Michael Gamon, Anthony Aue, Simon Corston-Oliver, and Eric Ringger. (2005): "Pulse: Mining Customer Opinions from Free Text". Under review.

Christopher D. Manning and Hinrich Schütze (2002): Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, London.

Kamal Nigam, Andrew McCallum, Sebastian Thrun and Tom Mitchell (2000): Text Classification from Labeled and Unlabeled Documents using EM. In: Machine Learning 39 (2/3), pp. 103-134.

Bo Pang, Lillian Lee and Shivakumar Vaithyanathan (2002): "Thumbs up? Sentiment Classification using Machine Learning Techniques". Proceedings of EMNLP 2002, pp. 79-86.

Bo Pang and Lillian Lee. (2004). A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts. Proceedings of ACL 2004, pp.217-278.

Peter D. Turney (2001): "Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL." In Proceedings of the Twelfth European Conference on Machine Learning, pp. 491-502.

Peter D. Turney (2002): "Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews". In: Proceedings of ACL 2002, pp. 417-424.

Peter D. Turney and M. L. Littman (2002): "Unsupervised Learning of Semantic Orientation from a Hundred-Billion-Word Corpus." Technical report ERC-1094 (NRC 44929), National Research Council of Canada.

Janyce Wiebe, Theresa Wilson and Matthew Bell (2001): "Identifying Collocations for Recognizing Opinions". In: Proceedings of the ACL/EACL Workshop on Collocation.

Hong Yu and Vasileios Hatzivassiloglou (2003): "Towards Answering opinion Questions: Separating Facts from Opinions and Identifying the Polarity of Opinion Sentences". In: Proceedings of EMNLP 2003.

Studying Feature Generation from Various Data Representations for Answer Extraction

Dan Shen[†‡] **Geert-Jan M. Kruijff**[†] **Dietrich Klakow**[‡]

[†] Department of Computational Linguistics
Saarland University
Building 17,Postfach 15 11 50
66041 Saarbruecken, Germany

[‡] Lehrstuhl Sprach Signal Verarbeitung
Saarland University
Building 17, Postfach 15 11 50
66041 Saarbruecken, Germany

{dshen,gj}@coli.uni-sb.de
{dietrich.klakow}@lsv.uni-saarland.de

Abstract

In this paper, we study how to generate features from various data representations, such as surface texts and parse trees, for answer extraction. Besides the features generated from the surface texts, we mainly discuss the feature generation in the parse trees. We propose and compare three methods, including feature vector, string kernel and tree kernel, to represent the syntactic features in Support Vector Machines. The experiment on the TREC question answering task shows that the features generated from the more structured data representations significantly improve the performance based on the features generated from the surface texts. Furthermore, the contribution of the individual feature will be discussed in detail.

1 Introduction

Open domain question answering (QA), as defined by the TREC competitions (Voorhees, 2003), represents an advanced application of natural language processing (NLP). It aims to find exact answers to open-domain natural language questions in a large document collection. For example:
Q2131: Who is the mayor of San Francisco?
Answer: Willie Brown

A typical QA system usually consists of three basic modules: 1. Question Processing (QP) Module, which finds some useful information from the questions, such as expected answer type and key words. 2. Information Retrieval (IR) Module, which searches a document collection to retrieve a set of relevant sentences using the question key words. 3. Answer Extraction (AE) Module, which analyzes the relevant sentences using the information provided by the QP module and identify the answer phrase.

In recent years, QA systems trend to be more and more complex, since many other NLP techniques, such as named entity recognition, parsing, semantic analysis, reasoning, and external resources, such as WordNet, web, databases, are incorporated. The various techniques and resources may provide the indicative evidences to find the correct answers. These evidences are further combined by using a pipeline structure, a scoring function or a machine learning method.

In the machine learning framework, it is critical but not trivial to generate the features from the various resources which may be represented as surface texts, syntactic structures and logic forms, etc. The complexity of feature generation strongly depends on the complexity of data representation. Many previous QA systems (Echihabi et al., 2003; Ravichandran, et al., 2003; Ittycheriah and Roukos, 2002; Ittycheriah, 2001; Xu et al., 2002) have well studied the features in the surface texts. In this paper, we will use the answer extraction module of QA as a case study to further explore how to generate the features for the more complex sentence representations, such as parse tree. Since parsing gives the deeper understanding of the sentence, the features generated from the parse tree are expected to improve the performance based on the features generated from the surface text. The answer ex-

Proceedings of the ACL Workshop on Feature Engineering for Machine Learning in NLP, pages 65–72,
Ann Arbor, June 2005.

traction module is built using Support Vector Machines (SVM). We propose three methods to represent the features in the parse tree: 1. features are designed by domain experts, extracted from the parse tree and represented as a feature vector; 2. the parse tree is transformed to a node sequence and a string kernel is employed; 3. the parse tree is retained as the original representation and a tree kernel is employed.

Although many textual features have been used in the others' AE modules, it is not clear that how much contribution the individual feature makes. In this paper, we will discuss the effectiveness of each individual textual feature in detail. We further evaluate the effectiveness of the syntactic features we proposed. Our experiments using TREC questions show that the syntactic features improve the performance by 7.57 MRR based on the textual features. It indicates that the new features based on a deeper language understanding are necessary to push further the machine learning-based QA technology. Furthermore, the three representations of the syntactic features are compared. We find that keeping the original data representation by using the data-specific kernel function in SVM may capture the more comprehensive evidences than the predefined features. Although the features we generated are specific to the answer extraction task, the comparison between the different feature representations may be helpful to explore the syntactic features for the other NLP applications.

2 Related Work

In the machine learning framework, it is crucial to capture the useful evidences for the task and integrate them effectively in the model. Many researchers have explored the rich textual features for the answer extraction.

IBM (Ittycheriah and Roukos, 2002; Ittycheriah, 2001) used a Maximum Entropy model to integrate the rich features, including query expansion features, focus matching features, answer candidate co-occurrence features, certain word frequency features, named entity features, dependency relation features, linguistic motivated features and surface patterns. ISI's (Echihabi et al. 2003; Echihabi and Marcu, 2003) statistical-based AE module implemented a noisy-channel model to explain how a given sentence tagged with an answer can be rewritten into a question through a sequence of stochastic operations. (Ravichandran et al., 2003) compared two maximum entropy-based QA systems, which view the AE as a classification problem and a re-ranking problem respectively, based on the word frequency features, expected answer class features, question word absent features and word match features. BBN (Xu et al. 2002) used a HMM-based IR system to score the answer candidates based on the answer contexts. They further re-ranked the scored answer candidates using the constraint features, such as whether a numerical answer quantifies the correct noun, whether the answer is of the correct location sub-type and whether the answer satisfies the verb arguments of the questions. (Suzuki et al. 2002) explored the answer extraction using SVM.

However, in the previous statistical-based AE modules, most of the features were extracted from the surface texts which are mainly based on the key words/phrases matching and the key word frequency statistics. These features only capture the surface-based information for the proper answers and may not provide the deeper understanding of the sentences. In addition, the contribution of the individual feature has not been evaluated by them. As for the features extracted from the structured texts, such as parse trees, only a few works explored some predefined syntactic relation features by partial matching. In this paper, we will explore the syntactic features in the parse trees and compare the different feature representations in SVM. Moreover, the contributions of the different features will be discussed in detail.

3 Answer Extraction

Given a question Q and a set of relevant sentences *SentSet* which is returned by the IR module, we consider all of the base noun phrases and the words in the base noun phrases as answer candidates ac_i. For example, for the question "*Q1956: What country is the largest in square miles?*", we extract the answer candidates { *Russia*, *largest country*, *largest*, *country*, *world*, *Canada*, *No.2.*} in the sentence "*I recently read that Russia is the largest country in the world, with Canada No. 2.*" The goal of the AE module is to choose the most probable answer from a set of answer candidates $\{ac_1, ac_2, ...ac_m\}$ for the question Q.

We regard the answer extraction as a classification problem, which classify each question and

answer candidate pair <Q, ac_i> into the positive class (the correct answer) and the negative class (the incorrect answer), based on some features. The predication for each <Q, ac_i> is made independently by the classifier, then, the *ac* with the most confident positive prediction is chosen as the answer for Q. SVM have shown the excellent performance for the binary classification, therefore, we employ it to classify the answer candidates.

Answer extraction is not a trivial task, since it involves several components each of which is fairly sophisticated, including named entity recognition, syntactic / semantic parsing, question analysis, etc. These components may provide some indicative evidences for the proper answers. Before generating the features, we process the sentences as follows:

1. tag the answer sentences with named entities.
2. parse the question and the answer sentences using the Collins' parser (Collin, 1996).
3. extract the key words from the questions, such as the target words, query words and verbs.

In the following sections, we will briefly introduce the machine learning algorithm. Then, we will discuss the features in detail, including the motivations and representations of the features.

4 Support Vector Machines

Support Vector Machines (SVM) (Vapnik, 1995) have strong theoretical motivation in statistical learning theory and achieve excellent generalization performance in many language processing applications, such as text classification (Joachims, 1998).

SVM constructs a binary classifier that predict whether an instance $\boldsymbol{x}$ ($\boldsymbol{w} \in \boldsymbol{R}^n$) is positive ($f(\boldsymbol{x}) = 1$) or negative ($f(\boldsymbol{x}) = -1$), where, an instance may be represented as a feature vector or a structure like sequence of characters or tree. In the simplest case (linearly separable instances), the decision $f(\boldsymbol{x}) = sgn(\boldsymbol{w} \cdot \boldsymbol{x} + b)$ is made based on a separating hyperplane $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ ($\boldsymbol{w} \in \boldsymbol{R}^n$, $b \in \boldsymbol{R}$). All instances lying on one side of the hyperplane are classified to a positive class, while others are classified to a negative class.

Given a set of labeled training instances $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_m, y_m)\}$, where $\boldsymbol{x}_i \in \boldsymbol{R}^n$ and $y_i = \{1, -1\}$, SVM is to find the optimal hyperplane that separates the positive and negative training instances with a maximal margin. The margin is defined as the distance from the separating hyperplane to the closest positive (negative) training instances. SVM is trained by solving a dual quadratic programming problem.

Practically, the instances are non-linearly separable. For this case, we need project the instances in the original space $\boldsymbol{R}^n$ to a higher dimensional space $\boldsymbol{R}^N$ based on the kernel function $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = <\Phi(\boldsymbol{x}_1), \Phi(\boldsymbol{x}_2)>$,where, $\Phi(\boldsymbol{x}): \boldsymbol{R}^n \rightarrow \boldsymbol{R}^N$ is a project function of the instance. By this means, a linear separation will be made in the new space. Corresponding to the original space $\boldsymbol{R}^n$, a non-linear separating surface is found. The kernel function has to be defined based on the Mercer's condition. Generally, the following kernel functions are widely used.

Polynomial kernel: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i \cdot \boldsymbol{x}_j + 1)^p$

Gaussian RBF kernel: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 / 2\sigma^2}$

5 Textual Features

Since the features extracted from the surface texts have been well explored by many QA systems (Echihabi et al., 2003; Ravichandran, et al., 2003; Ittycheriah and Roukos, 2002; Ittycheriah, 2001; Xu et al., 2002), we will not focus on the textual feature generation in this paper. Only four types of the basic features are used:

1. **Syntactic Tag Features**: the features capture the syntactic/POS information of the words in the answer candidates. For the certain question, such as "*Q1903: How many time zones are there in the world?*", if the answer candidate consists of the words with the syntactic tags "*CD NN*", it is more likely to be the proper answer.
2. **Orthographic Features**: the features capture the surface format of the answer candidates, such as capitalization, digits and lengths, etc. These features are motivated by the observations, such as, the length of the answers are often less than 3 words for the factoid questions; the answers may not be the subsequences of the questions; the answers often contain digits for the certain questions.
3. **Named Entity Features**: the features capture the named entity information of the answer

candidates. They are very effective for the *who*, *when* and *where* questions, such as, For "*Q1950: Who created the literary character Phineas Fogg?*", the answer "*Jules Verne*" is tagged as a PERSON name in the sentences "*Jules Verne 's Phileas Fogg made literary history when he traveled around the world in 80 days in 1873*.". For the certain question target, if the answer candidate is tagged as the certain type of named entity, one feature fires.

4. **Triggers**: some trigger words are collected for the certain questions. For examples, for "*Q2156: How fast does Randy Johnson throw?*", the trigger word "*mph*" for the question words "*how fast*" may help to identify the answer "*98-mph*" in "*Johnson throws a 98-mph fastball*".

6 Syntactic Features

In this section, we will discuss the feature generation in the parse trees. Since parsing outputs the highly structured data representation of the sentence, the features generated from the parse trees may provide the more linguistic-motivated explanation for the proper answers. However, it is not trivial to find the informative evidences from a parse tree.

The motivation of the syntactic features in our task is that the proper answers often have the certain syntactic relations with the question key words. Table 1 shows some examples of the typical syntactic relations between the proper answers (*a*) and the question target words (*qtarget*). Furthermore, the syntactic relations between the answers and the different types of question key words vary a lot. Therefore, we capture the relation features for the different types of question words respectively. The question words are divided into four types:

- Target word, which indicates the expected answer type, such as "*city*" in "*Q: What city is Disneyland in?*".
- Head word, which is extracted from *how* questions and indicates the expected answer head, such as "*dog*" in "*Q210: How many dogs pull …?*"
- Subject words, which are the base noun phrases of the question except the target word and the head word.
- Verb, which is the main verb of the question.

To our knowledge, the syntactic relation features between the answers and the question key words haven't been explored in the previous machine learning-based QA systems. Next, we will propose three methods to represent the syntactic relation features in SVM.

6.1 Feature Vector

It is the commonly used feature representation in most of the machine learning algorithms. We predefine a set of syntactic relation features, which is an enumeration of some useful evidences of the answer candidates (ac) and the question key words in the parse trees. 20 syntactic features are manually designed in the task. Some examples of the features are listed as follows,

- if the ac node is the same of the qtarget node, one feature fires.
- if the ac node is the sibling of the qtarget node, one feature fires.
- if the ac node the child of the qsubject node, one feature fires.

The limitation of the manually designed features is that they only capture the evidences in the local context of the answer candidates and the question key words. However, some question words, such as subject words, often have the long range syntac-

1. *a* node is the same as the *qtarget* node and *qtarget* is the hypernym of *a*.
Q: What *city* is Disneyland in? S: Not bad for a struggling actor who was working at *Tokyo* Disneyland a few years ago.
2. *a* node is the parent of *qtarget* node.
Q: What is the name of the *airport* in Dallas Ft. Worth? S: Wednesday morning, the low temperature at the Dallas-Fort Worth International *Airport* was 81 degrees.
3. *a* node is the sibling of the *qtarget* node.
Q: What *book* did Rachel Carson write in 1962? S: In her 1962 *book* Silent Spring, Rachel Carson, a marine biologist, chronicled DDT 's poisonous effects, ….

Table 1: Examples of the typical relations between answer and question target word. In Q, the italic word is question target word. In S, the italic word is the question target word which is mapped in the answer sentence; the underlined word is the proper answer for the question Q.

tic relations with the answers. To overcome the limitation, we will propose some special kernels which may keep the original data representation instead of explicitly enumerate the features, to explore a much larger feature space.

6.2 String Kernel

The second method represents the syntactic relation as a linked node sequence and incorporates a string kernel in SVM to handle the sequence.

We extract a path from the node of the answer candidate to the node of the question key word in the parse tree. The path is represented as a node sequence linked by symbols indicating upward or downward movement through the tree. For example, in Figure 1, the path from the answer candidate node "*211,456 miles*" to the question subject word node "*the moon*" is " $NPB\uparrow ADVP\uparrow VP\uparrow S\downarrow NPB$ ", where " $\uparrow$ " and " $\downarrow$ " indicate upward movement and downward movement in the parse tree. By this means, we represent the object from the original parse tree to the node sequence. Each character of the sequence is a syntactic/POS tag of the node. Next, a string kernel will be adapted to our task to calculate the similarity between two node sequences.

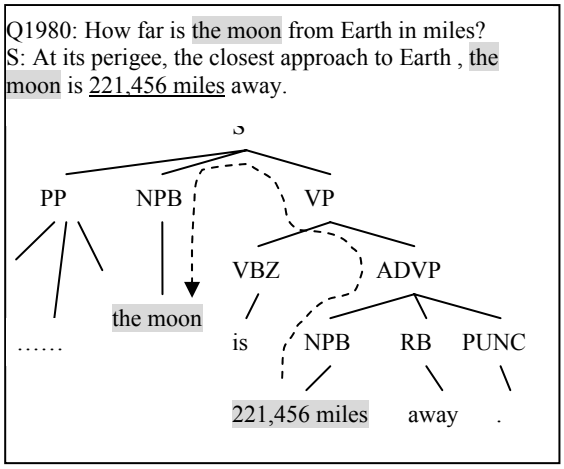

Figure 1: An example of the path from the answer candidate node to the question subject word node

(Haussler, 1999) first described a convolution kernel over the strings. (Lodhi et al., 2000) applied the string kernel to the text classification. (Leslie et al., 2002) further proposed a spectrum kernel, which is simpler and more efficient than the previous string kernels, for protein classification problem. In their tasks, the string kernels achieved the better performance compared with the human-defined features.

The string kernel is to calculate the similarity between two strings. It is based on the observation that the more common substrings the strings have, the more similar they are. The string kernel we used is similar to (Leslie et al., 2002). It is defined as the sum of the weighted common substrings. The substring is weighted by an exponentially decaying factor λ (set 0.5 in the experiment) of its length k. For efficiency, we only consider the substrings which length are less than 3. Different from (Leslie et al., 2002), the characters (syntactic/POS tag) of the string are linked with each other. Therefore, the matching between two substrings will consider the linking information. Two identical substrings will not only have the same syntactic tag sequences but also have the same linking symbols. For example, for the node sequences $NP\uparrow VP\uparrow VP\uparrow S\downarrow NP$ and $NP\uparrow NP\uparrow VP\downarrow NP$, there is a matched substring (k = 2): $NP\uparrow VP$.

6.3 Tree Kernel

The third method keeps the original representation of the syntactic relation in the parse tree and incorporates a tree kernel in SVM.

Tree kernels are the structure-driven kernels to calculate the similarity between two trees. They have been successfully accepted in the NLP applications. (Collins and Duffy, 2002) defined a kernel on parse tree and used it to improve parsing. (Collins, 2002) extended the approach to POS tagging and named entity recognition. (Zelenko et al., 2003; Culotta and Sorensen, 2004) further explored tree kernels for relation extraction.

We define an object (a relation tree) as the smallest tree which covers one answer candidate node and one question key word node. Suppose that a relation tree T has nodes $\{t_0, t_1, ..., t_n\}$ and each node t_i is attached with a set of attributes $\{a_0, a_1, ..., a_m\}$, which represents the local characteristics of t_i. In our task, the set of the attributes includes Type attributes, Orthographic attributes and Relation Role attributes, as shown in Table 2. The core idea of the tree kernel $K(T_1, T_2)$ is that the similarity between two trees T_1 and T_2 is

the sum of the similarity between their subtrees. It is calculated by dynamic programming and captures the long-range syntactic relations between two nodes. The kernel we use is similar to (Zelenko et al., 2003) except that we define a task-specific matching function and similarity function, which are two primitive functions to calculate the similarity between two nodes in terms of their attributes.

Matching function

$$m(t_i, t_j) = \begin{cases} 1 & \text{if } t_i.type = t_j.type \text{ and } t_i.role = t_j.role \\ 0 & \text{otherwise} \end{cases}$$

Similarity function

$$s(t_i, t_j) = \sum_{a \in \{a_0, \ldots, a_m\}} f(t_i.a, t_j.a)$$

where, $f(t_i.a, t_j.a)$ is a compatibility function between two feature values

$$f(t_i.a, t_j.a) = \begin{cases} 1 & \text{if } t_i.a = t_j.a \\ 0 & \text{otherwise} \end{cases}$$

Figure 2 shows two examples of the relation tree *T1_ac#targetword* and *T2_ac#targetword*. The kernel we used matches the following pairs of the nodes <t_0, w_0>, <t_1, w_2>, <t_2, w_2> and <t_4, w_1>.

Attributes		Examples
Type	POS tag	CD, NNP, NN…
	syntactic tag	NP, VP, …
Ortho-graphic	Is Digit?	DIG, DIGALL
	Is Capitalized?	CAP, CAPALL
	length of phrase	LNG1, LNG2#3, LNGgt3
Role1	Is answer candidate?	true, false
Role2	Is question key words?	true, false

Table 2: Attributes of the nodes

7 Experiments

We apply the AE module to the TREC QA task. To evaluate the features in the AE module independently, we suppose that the IR module has got 100% precision and only passes those sentences containing the proper answers to the AE module. The AE module is to identify the proper answers from the given sentence collection.

We use the questions of TREC8, 9, 2001 and 2002 for training and the questions of TREC2003 for testing. The following steps are used to generate the data:

1. Retrieve the relevant documents for each question based on the TREC judgments.
2. Extract the sentences, which match both the proper answer and at least one question key word, from these documents.
3. Tag the proper answer in the sentences based on the TREC answer patterns

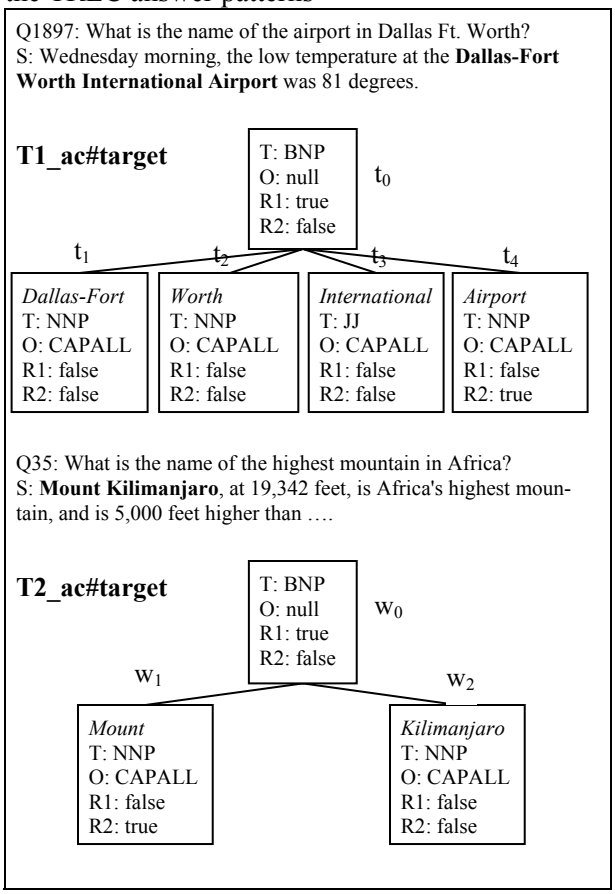

Figure 2: Two objects representing the relations between answer candidates and target words.

In TREC 2003, there are 413 factoid questions in which 51 questions (NIL questions) are not returned with the proper answers by TREC. According to our data generation process, we cannot provide data for those NIL questions because we cannot get the sentence collections. Therefore, the AE module will fail on all of the NIL questions and the number of the valid questions should be 362 (413 – 51). In the experiment, we still test the module on the whole question set (413 questions) to keep consistent with the other's work. The training set contains 1252 questions. The performance of our system is evaluated using the mean reciprocal rank (MRR). Furthermore, we also list the percentages of the correct answers respectively

in terms of the top 5 answers and the top 1 answer returned. We employ the SVM^{Light} (Joachims, 1999) to incorporate the features and classify the answer candidates. No post-processes are used to adjust the answers in the experiments.

Firstly, we evaluate the effectiveness of the textual features, described in Section 5. We incorporate them into SVM using the three kernel functions: linear kernel, polynomial kernel and RBF kernel, which are introduced in Section 4. Table 3 shows the performance for the different kernels. The RBF kernel (46.24 MRR) significantly outperforms the linear kernel (33.72 MRR) and the polynomial kernel (40.45 MRR). Therefore, we will use the RBF kernel in the rest experiments.

	Top1	Top5	MRR
linear	31.28	37.91	33.72
polynomial	37.91	44.55	40.45
RBF	42.67	51.58	46.24

Table 3: Performance for kernels

In order to evaluate the contribution of the individual feature, we test out module using different feature combinations, as shown in Table 4. Several findings are concluded:

1. With only the syntactic tag features $F_{syn.}$, the module achieves a basic level MRR of 31.38. The questions "*Q1903: How many time zones are there in the world?*" is correctly answered from the sentence "*The world is divided into 24 time zones.*".
2. The orthographic features $F_{orth.}$ show the positive effect with 7.12 MRR improvement based on $F_{syn.}$. They help to find the proper answer "*Grover Cleveland*" for the question "*Q2049: What president served 2 nonconsecutive terms?*" from the sentence "*Grover Cleveland is the forgotten two-term American president.*", while $F_{syn.}$ wrongly identify "*president*" as the answer.
3. The named entity features F_{ne} are also beneficial as they make the 4.46 MRR increase based on $F_{syn.}$+$F_{orth.}$ For the question "*Q2076: What company owns the soft drink brand "Gatorade"?*", F_{ne} find the proper answer "*Quaker Oats*" in the sentence "*Marineau , 53 , had distinguished himself by turning the sports drink Gatorade into a mass consumer brand while an executive at Quaker Oats During his 18-month…*", while $F_{syn.}$+$F_{orth.}$ return the wrong answer "*Marineau*".
4. The trigger features F_{trg} lead to an improvement of 3.28 MRR based on $F_{syn.}$+F_{orth}+F_{ne}. They correctly answer more questions. For the question "*Q1937: How fast can a nuclear submarine travel?*", F_{trg} return the proper answer "*25 knots*" from the sentence "*The submarine , 360 feet (109.8 meters) long , has 129 crew members and travels at 25 knots.*", but the previous features fail on it.

F_{syn}	$F_{orth.}$	F_{ne}	F_{trg}	Top1	Top5	MRR
√				26.50	38.92	31.38
√	√			34.69	43.61	38.50
√	√	√		39.85	47.82	42.96
√	√	√	√	42.67	51.58	46.24

Table 4: Performance for feature combinations

Next, we will evaluate the effectiveness of the syntactic features, described in Section 6. Table 5 compares the three feature representation methods, *FeatureVector*, *StringKernel* and *TreeKernel*.

- *FeatureVector* (Section 6.1). We predefine some features in the syntactic tree and present them as a feature vector. The syntactic features are added with the textual features and the RBF kernel is used to cope with them.
- *StringKernel* (Section 6.2). No features are predefined. We transform the syntactic relations between answer candidates and question key words to node sequences and a string kernel is proposed to cope with the sequences. Then we add the string kernel for the syntactic relations and the RBF kernel for the textual features.
- *TreeKernel* (Section 6.3). No features are predefined. We keep the original representations of the syntactic relations and propose a tree kernel to cope with the relation trees. Then we add the tree kernel and the RBF kernel.

	Top1	Top2	MRR
$F_{syn.}$+$F_{orth.}$+F_{ne}+F_{trg}	42.67	51.58	46.24
FeatureVector	46.19	53.69	49.28
StringKernel	48.99	55.83	52.29
TreeKernel	50.41	57.46	53.81

Table 5: Performance for syntactic feature representations

Table 5 shows the performances of *FeatureVector*, *StringKernel* and *TreeKernel.* All of them improve the performance based on the textual features ($F_{syn.}$+$F_{orth.}$+F_{ne}+F_{trg}) by 3.04 MRR, 6.05 MRR and 7.57 MRR respectively. The probable reason may be that the features generated from the structured data representation may capture the

more linguistic-motivated evidences for the proper answers. For example, the syntactic features help to find the answer "*nitrogen*" for the question "*Q2139: What gas is 78 percent of the earth 's atmosphere?*" in the sentence "*One thing they haven't found in the moon's atmosphere so far is nitrogen, the gas that makes up more than three-quarters of the Earth's atmosphere.*", while the textual features fail on it. Furthermore, the *StringKernel* (+3.01MRR) and *TreeKernel* (+4.53MRR) achieve the higher performance than *FeatureVector,* which may be explained that keeping the original data representations by incorporating the data-specific kernels in SVM may capture the more comprehensive evidences than the predefined features. Moreover, *TreeKernel* slightly outperforms *StringKernel* by 1.52 MRR. The reason may be that when we transform the representation of the syntactic relation from the tree to the node sequence, some information may be lost, such as the sibling node of the answer candidates. Sometimes the information is useful to find the proper answers.

8 Conclusion

In this paper, we study the feature generation based on the various data representations, such as surface text and parse tree, for the answer extraction. We generate the syntactic tag features, orthographic features, named entity features and trigger features from the surface texts. We further explore the feature generation from the parse trees which provide the more linguistic-motivated evidences for the task. We propose three methods, including feature vector, string kernel and tree kernel, to represent the syntactic features in Support Vector Machines. The experiment on the TREC question answering task shows that the syntactic features significantly improve the performance by 7.57MRR based on the textual features. Furthermore, keeping the original data representation using a data-specific kernel achieves the better performance than the explicitly enumerated features in SVM.

References

M. Collins. 1996. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of ACL-96*, pages 184-191.

M. Collins. 2002. New Ranking Algorithms for Parsing and Tagging: Kernel over Discrete Structures, and the Voted Perceptron. In *Proceedings of ACL-2002*.

M. Collins and N. Duffy. 2002. Convolution Kernels for Natural Language. *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press.

A. Culotta and J. Sorensen. 2004. Dependency Tree Kernels for Relation Extraction. In *Proceedings of ACL-2004*.

A. Echihabi, U. Hermjakob, E. Hovy, D. Marcu, E. Melz, D. Ravichandran. 2003. Multiple-Engine Question Answering in TextMap. In *Proceedings of the TREC-2003 Conference*, NIST.

A. Echihabi, D. Marcu. 2003. A Noisy-Channel Approach to Question Answering. In Proceedings of the ACL-2003.

D. Haussler. 1999. Convolution Kernels on Discrete Structures. Technical Report UCS-CRL-99-10, University of California, Santa Cruz.

A. Ittycheriah and S. Roukos. 2002. IBM's Statistical Question Answering System – TREC 11. In *Proceedings of the TREC-2002 Conference*, NIST.

A. Ittycheriah. 2001. Trainable Question Answering System. Ph.D. Dissertation, Rutgers, The State University of New Jersey, New Brunswick, NJ.

T. Joachims. 1999. Making large-Scale SVM Learning Practical. *Advances in Kernel Methods - Support Vector Learning*, MIT-Press, 1999.

T. Joachims. 1998. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In Proceedings of the European Conference on Machine Learning, Springer.

C. Leslie, E. Eskin and W. S. Noble. 2002. The spectrum kernel: A string kernel for SVM protein classification. Proceedings of the Pacific Biocomputing Symposium.

H. Lodhi, J. S. Taylor, N. Cristianini and C. J. C. H. Watkins. 2000. Text Classification using String Kernels. In *NIPS*, pages 563-569.

D. Ravichandran, E. Hovy and F. J. Och. 2003. Statistical QA – Classifier vs. Re-ranker: What's the difference? In *Proceedings of Workshop on Mulingual Summarization and Question Answering*, ACL 2003.

J. Suzuki, Y. Sasaki, and E. Maeda. 2002. SVM Answer Selection for Open-domain Question Answering. In Proc. of COLING 2002, pages 974–980.

V. N. Vapnik. 1998. Statistical Learning Theory. Springer.

E.M. Voorhees. 2003. Overview of the TREC 2003 Question Answering Track. In *Proceedings of the TREC-2003 Conference*, NIST.

J. Xu, A. Licuanan, J. May, S. Miller and R. Weischedel. 2002. TREC 2002 QA at BBN: Answer Selection and Confidence Estimation. In *Proceedings of the TREC-2002 Conference*, NIST.

D. Zelenko, C. Aone and A. Richardella. 2003. Kernel Methods for Relation Extraction. *Journal of Machine Learning Research*, pages 1083-1106.

Author Index