SVM Model Tampering and Anchored Learning: A Case Study in Hebrew NP Chunking

Yoav Goldberg and Michael Elhadad
Computer Science Department
Ben Gurion University of the Negev
P.O.B 653 Be'er Sheva 84105, Israel
yoavg,elhadad@cs.bgu.ac.il

Abstract

We study the issue of porting a known NLP method to a language with little existing NLP resources, specifically Hebrew SVM-based chunking. We introduce two SVM-based methods – Model Tampering and Anchored Learning. These allow fine grained analysis of the learned SVM models, which provides guidance to identify errors in the training corpus, distinguish the role and interaction of lexical features and eventually construct a model with ∼10% error reduction. The resulting chunker is shown to be robust in the presence of noise in the training corpus, relies on less lexical features than was previously understood and achieves an F-measure performance of 92.2 on automatically PoS-tagged text. The SVM analysis methods also provide general insight on SVM-based chunking.

1 Introduction

While high-quality NLP corpora and tools are available in English, such resources are difficult to obtain in most other languages. Three challenges must be met when adapting results established in English to another language: (1) acquiring high quality annotated data; (2) adapting the English task definition to the nature of a different language, and (3) adapting the algorithm to the new language. This paper presents a case study in the adaptation of a well known task to a language with few NLP resources available. Specifically, we deal with SVM based Hebrew NP chunking. In (Goldberg et al., 2006), we established that the task is not trivially transferable to Hebrew, but reported that SVM based chunking (Kudo and Matsumoto, 2000) performs well. We extend that work and study the problem from 3 angles: (1) how to deal with a corpus that is smaller and with a higher level of noise than is available in English; we propose techniques that help identify 'suspicious' data points in the corpus, and identify how robust the model is in the presence of noise; (2) we compare the task definition in English and in Hebrew through quantitative evaluation of the differences between the two languages by analyzing the relative importance of features in the learned SVM models; and (3) we analyze the structure of learned SVM models to better understand the characteristics of the chunking problem in Hebrew.

While most work on chunking with machine learning techniques tend to treat the classification engine as a black-box, we try to investigate the resulting classification model in order to understand its inner working, strengths and weaknesses. We introduce two SVM-based methods – Model Tampering and Anchored Learning – and demonstrate how a fine-grained analysis of SVM models provides insights on all three accounts. The understanding of the relative contribution of each feature in the model helps us construct a better model, which achieves ∼10% error reduction in Hebrew chunking, as well as identify corpus errors. The methods also provide general insight on SVM-based chunking.

2 Previous Work

NP chunking is the task of marking the boundaries of simple noun-phrases in text. It is a well studied problem in English, and was the focus of CoNLL2000's Shared Task (Sang and Buchholz,

Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pages 224–231,
Prague, Czech Republic, June 2007.

2000). Early attempts at NP Chunking were rule learning systems, such as the Error Driven Pruning method of Pierce and Cardie (1998). Following Ramshaw and Marcus (1995), the current dominant approach is formulating chunking as a classification task, in which each word is classified as the (B)eginning, (I)nside or (O)outside of a chunk. Features for this classification usually involve local context features. Kudo and Matsumoto (2000) used SVM as a classification engine and achieved an F-Score of 93.79 on the shared task NPs. Since SVM is a binary classifier, to use it for the 3-class classification of the chunking task, 3 different classifiers $\{$B/I, B/O, I/O$\}$ were trained and their majority vote was taken.

NP chunks in the shared task data are BaseNPs, which are non-recursive NPs, a definition first proposed by Ramshaw and Marcus (1995). This definition yields good NP chunks for English. In (Goldberg et al., 2006) we argued that it is not applicable to Hebrew, mainly because of the prevalence of the Hebrew's construct state ($smixut$). $Smixut$ is similar to a noun-compound construct, but one that can join a noun (with a special morphological marking) with a full NP. It appears in about 40% of Hebrew NPs. We proposed an alternative definition (termed SimpleNP) for Hebrew NP chunks. A SimpleNP cannot contain embedded relatives, prepositions, VPs and NP-conjunctions (except when they are licensed by $smixut$). It can contain $smixut$, possessives (even when they are attached by the 'של/of' preposition) and partitives (and, therefore, allows for a limited amount of recursion). We applied this definition to the Hebrew Tree Bank (Sima'an et al., 2001), and constructed a moderate size corpus (about 5,000 sentences) for Hebrew SimpleNP chunking. SimpleNPs are different than English BaseNPs, and indeed some methods that work well for English performed poorly on Hebrew data. However, we found that chunking with SVM provides good result for Hebrew SimpleNPs. We analyzed that this success comes from SVM's ability to use lexical features, as well as two Hebrew morphological features, namely "number" and "construct-state".

One of the main issues when dealing with Hebrew chunking is that the available tree bank is rather small, and since it is quite new, and has not been used intensively, it contains a certain amount of inconsistencies and tagging errors. In addition, the identification of SimpleNPs from the tree bank also introduces some errors. Finally, we want to investigate chunking in a scenario where PoS tags are assigned automatically and chunks are then computed. The Hebrew PoS tagger we use introduces about 8% errors (compared with about 4% in English). We are, therefore, interested in identifying errors in the chunking corpus, and investigating how the chunker operates in the presence of noise in the PoS tag sequence.

3 Model Tampering

3.1 Notation and Technical Review

This section presents notation as well as a technical review of SVM chunking details relevant to the current study. Further details can be found in Kudo and Matsumoto (2000; 2003).

SVM (Vapnik, 1995) is a supervised binary classifier. The input to the learner is a set of l training samples $(x_1, y_1), \ldots, (x_l, y_l)$, $x \in R^n$, $y \in \{+1, -1\}$. x_i is an n dimensional feature vector representing the ith sample, and y_i is the label for that sample. The result of the learning process is the set SV of Support Vectors, the associated weights α_i, and a constant b. The Support Vectors are a subset of the training vectors, and together with the weights and b they define a hyperplane that optimally separates the training samples. The basic SVM formulation is of a linear classifier, but by introducing a kernel function K that non-linearly transforms the data from R^n into a higher dimensional space, SVM can be used to perform non-linear classification. SVM's decision function is: $y(x) = sgn\left(\sum_{j \in SV} y_j \alpha_j K(x_j, x) + b\right)$ where x is an n dimensional feature vector to be classified. In the linear case, K is a dot product operation and the sum $w = \sum y_j \alpha_j x_j$ is an n dimensional weight vector assigning weight for each of the n features. The other kernel function we consider in this paper is a polynomial kernel of degree 2: $K(x_i, x_j) = (x_i \cdot x_j + 1)^2$. When using binary valued features, this kernel function essentially implies that the classifier considers not only the explicitly specified features, but also all available pairs of features. In order to cope with inseparable data, the learning process of SVM allows for some misclassification, the amount of which is determined by a parameter C, which can be thought of as a penalty for each misclassified training sample.

In SVM based chunking, each word and its context is considered a learning sample. We refer to the word being classified as w_0, and to its part-of-speech (PoS) tag, morphology, and B/I/O tag as p_0, m_0 and t_0 respectively. The information considered for classification is $w_{-cw} \ldots w_{cw}$, $p_{-cp} \ldots p_{cp}$, $m_{-cm} \ldots m_{cm}$ and $t_{-ct} \ldots t_{-1}$. The feature vector F is an indexed list of all the features present in the corpus. A feature f_i of the form $w_{+1} = dog$ means that the word following the one being classified is 'dog'. Every learning sample is represented by an $n = |F|$ dimensional binary vector x. $x_i = 1$ iff the feature f_i is active in the given sample, and 0 otherwise. This encoding leads to extremely high dimensional vectors, due to the lexical features $w_{-cw} \ldots w_{cw}$.

3.2 Introducing Model Tampering

An important observation about SVM classifiers is that features which are not active in any of the Support Vectors have no effect on the classifier decision. We introduce Model Tampering, a procedure in which we change the Support Vectors in a model by forcing some values in the vectors to 0.

The result of this procedure is a new Model in which the deleted features never take part in the classification.

Model tampering is different than feature selection: on the one hand, it is a method that helps us identify irrelevant features in a model after training; on the other hand, and this is the key insight, removing features **after** training is not the same as removing them before training. The presence of the low-relevance features during training has an impact on the generalization performed by the learner as shown below.

3.3 The Role of Lexical Features

In Goldberg *et al.* (2006), we have established that using lexical features increases the chunking F-measure from 78 to over 92 on the Hebrew Treebank. We refine this observation by using Model Tampering, in order to assess the importance of lexical features in NP Chunking. We are interested in identifying which specific lexical items and contexts impact the chunking decision, and quantifying their effect. Our method is to train a chunking model on a given training corpus, tamper with the resulting model in various ways and measure the performance[1] of the tampered models on a test corpus.

3.4 Experimental Setting

We conducted experiments both for English and Hebrew chunking. For the Hebrew experiments, we use the corpora of (Goldberg et al., 2006). The first one is derived from the original Treebank by projecting the full syntactic tree, constructed manually, onto a set of NP chunks according to the SimpleNP rules. We refer to the resulting corpus as HEB_{Gold} since PoS tags are fully reliable. The HEB_{Err} version of the corpus is obtained by projecting the chunk boundaries on the sequence of PoS and morphology tags obtained by the automatic PoS tagger of Adler & Elhadad (2006). This corpus includes an error rate of about 8% on PoS tags. The first 500 sentences are used for testing, and the rest for training. The corpus contains 27K NP chunks. For the English experiments, we use the now-standard training and test sets that were introduced in (Marcus and Ramshaw, 1995)[2]. Training was done using Kudo's YAMCHA toolkit[3]. Both Hebrew and English models were trained using a polynomial kernel of degree 2, with $C = 1$. For English, the features used were: $w_{-2} \ldots w_2$, $p_{-2} \ldots p_2$, $t_{-2} \ldots t_{-1}$. The same features were used for Hebrew, with the addition of $m_{-2} \ldots m_2$. These are the same settings as in (Kudo and Matsumoto, 2000; Goldberg et al., 2006).

3.5 Tamperings

We experimented with the following tamperings:

TopN – We define *model feature count* to be the number of Support Vectors in which a feature is active in a given classifier. This tampering leaves in the model only the top N lexical features in each classifier, according to their count.

NoPOS – all the lexical features corresponding to a given part-of-speech are removed from the model. For example, in a NoJJ tampering, all the features of the form $w_i = X$ are removed from all the support vectors in which $p_i = JJ$ is active.

Loc≠i – all the lexical features with index i are removed from the model *e.g.*, in a Loc≠+2 tamper-

[1] The performance metric we use is the standard Precision/Recall/F measures, as computed by the conlleval program: http://www.cnts.ua.ac.be/conll2000/chunking/conlleval.txt

[2] ftp://ftp.cis.upenn.edu/pub/chunker

[3] http://chasen.org/∼taku/software/yamcha/

ing, features of the form $w_{+2} = X$ are removed).

Loc=i – all the lexical features with an index other than i are removed from the model.

3.6 Results and Discussion

Highlights of the results are presented in Tables (1-3). The numbers reported are F measures.

TopN	**HEB**$_{Gold}$	**HEB**$_{Err}$	**ENG**
ALL	93.58	92.48	93.79
N=0	78.32	76.27	90.10
N=10	90.21	88.68	90.24
N=50	91.78	90.85	91.22
N=100	92.25	91.62	91.72
N=500	93.60	92.23	93.12
N=1000	93.56	92.41	93.30

Table 1: Results of TopN Tampering.

The results of the TopN tamperings show that for both languages, most of the lexical features are irrelevant for the classification – the numbers achieved by using all the lexical features (about 30,000 in Hebrew and 75,000 in English) are very close to those obtained using only a few lexical features. This finding is very encouraging, and suggests that SVM based chunking is robust to corpus variations.

Another conclusion is that lexical features help balance the fact that PoS tags can be noisy: we know both HEB_{Err} and ENG include PoS tagging errors (about 8% in Hebrew and 4% in English). While in the case of “perfect” PoS tagging (HEB_{Gold}), a very small amount of lexical features is sufficient to reach the best F-result (500 out of 30,264), in the presence of PoS errors, more than the top 1000 lexical features are needed to reach the result obtained with all lexical features.

More striking is the fact that in Hebrew, the top 10 lexical features are responsible for an improvement of 12.4 in F-score. The words covered by these 10 features are the following: `Start of Sentence marker` and `comma`, `quote`, ‘`of`/של’, ‘`and`/ו’, ‘`the`/ה’ and ‘`in`/ב’.

This finding suggests that the Hebrew PoS tagset might not be informative enough for the chunking task, especially where punctuation [4] and prepositions are concerned. The results in Table 2 give further support for this claim.

[4] Unlike the WSJ PoS tagset in which most punctuations get unique tags, our tagset treat punctuation marks as one group.

NoPOS	**HEB**$_G$	**HEB**$_E$	**NoPOS**	**HEB**$_G$	**HEB**$_E$
Prep	**85.25**	**84.40**	Pronoun	92.97	92.14
Punct	**88.90**	**87.66**	Conjunction	92.31	91.67
Adverb	92.02	90.72	Determiner	92.55	91.39

Table 2: Results of Hebrew NoPOS Tampering. Other scores are $\geq$ 93.3(HEB$_G$), $\geq$ 92.2(HEB$_E$).

When removing lexical features of a specific PoS, the most dramatic loss of F-score is reached for Prepositions and Punctuation marks, followed by Adverbs, and Conjunctions. Strikingly, lexical information for most open-class PoS (including Proper Names and Nouns) has very little impact on Hebrew chunking performance.

From this observation, one could conclude that enriching a model based only on PoS with lexical features for only a few closed-class PoS (prepositions and punctuation) could provide appropriate results even with a simpler learning method, one that cannot deal with a large number of features. We tested this hypothesis by training the Error-Driven Pruning (EDP) method of (Cardie and Pierce, 1998) with an extended set of features. EDP with PoS features only produced an F-result of 76.3 on HEB_{Gold}. By adding lexical features only for prepositions {של כ ה ב מ}, one conjunction {ו} and punctuation, the F-score on HEB_{Gold} indeed jumps to 85.4. However, when applied on HEB_{Err}, EDP falls down again to 59.4. This striking disparity, by comparison, lets us appreciate the resilience of the SVM model to PoS tagging errors, and its generalization capability even with a reduced number of lexical features.

Another implication of this data is that commas and quotation marks play a major role in determining NP boundaries in Hebrew. In Goldberg *et al.* (2006), we noted the Hebrew Treebank is not consistent in its treatment of punctuation, and thus we evaluated the chunker only after performing normalization of chunk boundaries for punctuations. We now hypothesize that, since commas and quotation marks play such an important role in the classification, performing such normalization *before* the training stage might be beneficial. Indeed results on the normalized corpus show improvement of about 1.0 in F score on both HEB_{Err} and HEB_{Gold}. A 10-fold cross validation experiment on punctuation normalized HEB_{Err} resulted in an F-Score of 92.2, improving the results reported by (Goldberg et al.,

2006) on the same setting (91.4).

Loc=I	HEB$_E$	ENG	Loc≠I	HEB$_E$	ENG
-2	78.26	89.79	-2	91.62	93.87
-1	76.96	90.90	-1	91.86	93.03
0	90.33	92.37	0	79.44	91.16
1	76.90	90.47	1	92.33	93.30
2	76.55	90.06	2	92.18	93.65

Table 3: Results of Loc Tamperings.

We now turn to analyzing the importance of context positions (Table 3). For both languages, the most important lexical feature (by far) is at position 0, that is, the word currently being classified. For English, it is followed by positions 1 and -1, and then positions 2 and -2. For Hebrew, back context seems to have more effect than front context. In Hebrew, all the positions positively contribute to the decision, while in English removing $w_{2/-2}$ slightly improves the results (note also that including only feature $w_{2/-2}$ performs worse than with no lexical information in English).

3.7 The Real Role of Lexical Features

Model tampering (i.e., removing features after the learning stage) is not the same as learning without these features. This claim is verified empirically: training on the English corpus without the lexical features at position –2 yields worse results than with them (93.73 vs. 93.79) – while removing the w_{-2} features via tampering on a model trained with w_{-2} yields better results (93.87). Similarly, for all corpora, training using only the top 1,000 features (as defined in the Top1000 tampering) results in loss of about 2 in F-Score (ENG 92.02, HEB_{Err} 90.30, HEB_{Gold} 91.67), while tampering Top1000 yields a result very close to the best obtained (93.56, 92.41 or 93.3F).

This observation leads us to an interesting conclusion about the real role of lexical features in SVM based chunking: **rare events (features) are used to memorize hard examples**. Intuitively, by giving a heavy weight to rare events, the classifier learns specific rules such as "*if the word at position -2 is X and the PoS at position 2 is Y, then the current word is* **I***nside a noun-phrase*". Most of these rules are accidental – there is no real relation between the particular word-pos combination and the class of the current word, it just happens to be this way in the training samples. Marking the rare occurrences helps the learner achieve better generalization on the other, more common cases, which are similar to the outlier on most features, except the "irrelevant ones". As the events are rare, such rules usually have no effect on chunking accuracy: they simply never occur in the test data. This observation refines the common conception that SVM chunking does not suffer from irrelevant features: in chunking, SVM indeed generalizes well for the common cases but also over-fits the model on outliers.

Model tampering helps us design a model in two ways: (1) it is a way to "open the black box" obtained when training an SVM and to analyze the respective importance of features. In our case, this analysis allowed us to identify the importance of punctuation and prepositions and improve the model by defining more focused features (improving overall result by $\sim$1.0 F-point). (2) The analysis also led us to the conclusion that "feature selection" is complex in the case of SVM – irrelevant features help prevent over-generalization by forcing over-fitting on outliers.

We have also confirmed that the model learned remains robust in the presence of noise in the PoS tags and relies on only few lexical features. This verification is critical in the context of languages with few computational resources, as we expect the size of corpora and the quality of taggers to keep lagging behind that achieved in English.

4 Anchored Learning

We pursue the observation of how SVM deals with outliers by developing the *Anchored Learning* method. The idea behind Anchored Learning is to add a unique feature a_i (an *anchor*) to each training sample (we add as many new features to the model as there are training samples). These new features make our data linearly separable. The SVM learner can then use these anchors (which will never occur on the test data) to memorize the hard cases, decreasing this burden from "real" features.

We present two uses for Anchored Learning. The first is the identification of hard cases and corpus errors, and the second is a preliminary feature selection approach for SVM to improve chunking accuracy.

4.1 Mining for Errors and Hard Cases

Following the intuition that SVM gives more weight to anchor features of hard-to-classify cases, we can actively look for such cases by training an SVM chunker on anchored data (as the anchored data is guaranteed to be linearly separable, we can set a very high value to the C parameter, preventing any misclassification), and then investigating either the anchors whose weights[5] are above some threshold t or the top N heaviest anchors, and their corresponding corpus locations. **These locations are those that the learner considers hard to classify**. They can be either corpus errors, or genuinely hard cases.

This method is similar to the corpus error detection method presented by Nakagawa and Matsumoto (2002). They constructed an SVM model for PoS tagging, and considered Support Vectors with high α values to be indicative of suspicious corpus locations. These locations can be either outliers, or correctly labeled locations similar to an outlier. They then looked for similar corpus locations with a different label, to point out right-wrong pairs with high precision.

Using anchors improves their method in three aspects: (1) without anchors, similar examples are often indistinguishable to the SVM learner, and in case they have conflicting labels both examples will be given high weights. That is, both the regular case and the hard case will be considered as hard examples. Moreover, similar corpus errors might result in only one support vector that cover all the group of similar errors. Anchors mitigate these effects, resulting in better precision and recall. (2) The more errors there are in the corpus, the less linearly separable it is. Un-anchored learning on erroneous corpus can take unreasonable amount of time. (3) Anchors allow learning while removing some of the important features but still allow the process to converge in reasonable time. This lets us analyze which cases become hard to learn if we don't use certain features, or in other words: what problematic cases are solved by specific features.

The hard cases analysis achieved by anchored learning is different from the usual error analysis carried out on observed classification errors. The traditional methods give us intuitions about where the classifier **fails to generalize**, while the method we present here gives us intuition about what the classifier **considers hard to learn**, based on the training examples alone.

The intuition that "hard to learn" examples are suspect corpus errors is not new, and appears also in Abney *et al.* (1999) , who consider the "heaviest" samples in the final distribution of the AdaBoost algorithm to be the hardest to classify and thus likely corpus errors. While AdaBoost models are easy to interpret, this is not the case with SVM. Anchored learning allows us to extract the hard to learn cases from an SVM model. Interestingly, while both AdaBoost and SVM are 'large margin' based classifiers, there is less than 50% overlap in the hard cases for the two methods (in terms of mistakes on the test data, there were 234 mistakes shared by AdaBoost and SVM, 69 errors unique to SVM and 126 errors unique to AdaBoost)[6]. Analyzing the difference in what the two classifiers consider hard is interesting, and we will address it in future work. In the current work, we note that for finding corpus errors the two methods are complementary.

Experiment 1 – Locating Hard Cases

A linear SVM model (M_{full}) was trained on the training subset of the anchored, punctuation-normalized, HEB_{Gold} corpus, with the same features as in the previous experiments, and a C value of 9,999. Corpus locations corresponding to anchors with weights >1 were inspected. There were about 120 such locations out of 4,500 sentences used in the training set. Decreasing the threshold t would result in more cases. We analyzed these locations into 3 categories: corpus errors, cases that challenge the SimpleNP definition, and cases where the chunking decision is genuinely difficult to make in the absence of global syntactic context or world knowledge.

Corpus Errors: The analysis revealed the following corpus errors: we identified 29 hard cases related to conjunction and apposition (is the comma, colon or slash inside an NP or separating two distinct NPs). 14 of these hard cases were indeed mistakes in the corpus. This was anticipated, as we distinguished appositions and conjunctive commas using heuristics, since the Treebank marking of conjunctions is somewhat inconsistent.

In order to build the Chunk NP corpus, the syntactic trees of the Treebank were processed to derive chunks according to the SimpleNP definition. The hard cases analysis identified 18 instances where this

[5] As each anchor appear in only one support vector, we can treat the vector's α value as the anchor weight

[6] These numbers are for pairwise Linear SVM and AdaBoost classifiers trained on the same features.

transformation results in erroneous chunks. For example, null elements result in improper chunks, such as chunks containing only adverbs or only adjectives.

We also found 3 invalid sentences, 6 inconsistencies in the tagging of interrogatives with respect to chunk boundaries, as well as 34 other specific mistakes. Overall, more than half of the locations identified by the anchors were corpus errors. Looking for cases similar to the errors identified by anchors, we found 99 more locations, 77 of which were errors.

Refining the SimpleNP Definition: The hard cases analysis identified examples that challenge the SimpleNP definition proposed in Goldberg *et al.* (2006). The most notable cases are:

The 'et' marker : '*et*' is a syntactic marker of definite direct objects in Hebrew. It was regarded as a part of SimpleNPs in their definition. In some cases, this forces the resulting SimpleNP to be too inclusive:

[את הממשלה, הכנסת בית המשפט והתקשורת]

['et' (the government, the parliament and the media)]

Because in the Treebank the conjunction depends on '*et*' as a single constituent, it is fully embedded in the chunk. Such a conjunction should not be considered simple.

The של *preposition ('of')* marks generalized possession and was considered unambiguous and included in SimpleNPs. We found cases where 'של' causes PP attachment ambiguity:

[נשיא בית הדין] ל [משמעת] של [המשטרה]

[president-cons house-cons the-law] for [discipline] of [the police] / The Police Disciplinary Court President

Because 2 prepositions are involved in this NP, 'של' (*of*) and 'ל' (*for*), the 'של' part cannot be attached unambiguously to its head ('*court*'). It is unclear whether the 'ל' preposition should be given special treatment to allow it to enter simple NPs in certain contexts, or whether the inconsistent handling of the 'של' that results from the 'ל' inter-position is preferable.

Complex determiners and quantifiers: In many cases, complex determiners in Hebrew are multi-word expressions that include nouns. The inclusion of such determiners inside the SimpleNPs is not consistent.

Genuinely hard cases were also identified. These include prepositions, conjunctions and multi-word idioms (most of them are adjectives and prepositions which are made up of nouns and determiners, *e.g.*, as the word *unanimously* is expressed in Hebrew as the multi-word expression 'one mouth'). Also, some *adverbials* and *adjectives* are impossible to distinguish using only local context.

The anchors analysis helped us improve the chunking method on two accounts: (1) it identified corpus errors with high precision; (2) it made us focus on hard cases that challenge the linguistic definition of chunks we have adopted. Following these findings, we intend to refine the Hebrew SimpleNP definition, and create a new version of the Hebrew chunking corpus.

Experiment 2 – determining the role of contextual lexical features

The intent of this experiment is to understand the role of the contextual lexical features (w_i, $i \neq 0$). This is done by training 2 additional anchored linear SVM models, $M_{no-cont}$ and M_{near}. These are the same as M_{full} except for the lexical features used during training. $M_{no-cont}$ uses only w_0, while M_{near} uses w_0,w_{-1},w_{+1}.

Anchors are again used to locate the hard examples for each classifier, and the differences are examined. The examples that are hard for M_{near} but not for M_{full} are those solved by w_{-2},w_{+2}. Similarly, the examples that are hard for $M_{no-cont}$ but not for M_{near} are those solved by w_{-1},w_{+1}. Table 4 indicates the number of hard cases identified by the anchor method for each model. One way to interpret these figures, is that the introduction of features $w_{-1,+1}$ solves 5 times more hard cases than $w_{-2,+2}$.

Model	Number of hard cases ($t = 1$)	Hard cases for classifier B-I
M_{full}	120	2
M_{near}	320 (+ 200)	12
$M_{no-cont}$	1360 (+ 1040)	164

Table 4: Number of hard cases per model type.

Qualitative analysis of the hard cases solved by the contextual lexical features shows that they contribute mostly to the identification of chunk boundaries in cases of conjunction, apposition, attachment of adverbs and adjectives, and some multi-word expressions.

The number of hard cases specific to the B-I classifier indicates how the features contribute to the decision of splitting or continuing back-to-back NPs. Back-to-back NPs amount to 6% of the NPs in HEB_{Gold} and 8% of the NPs in ENG. However, while in English most of these cases are easily resolved, Hebrew phenomena such as null-equatives and free word order make them harder. To quantify the difference: 79% of the first words of the second NP in English belong to one of the closed classes POS, DT, WDT, PRP, WP – categories which mostly cannot appear in the middle of base NPs. In contrast, in Hebrew, 59% are Nouns, Numbers or Proper Names. Moreover, in English the ratio of unique first words to number of adjacent NPs is 0.068, while in Hebrew it is 0.47. That is, in Hebrew, almost every second such NP starts with a different word.

These figures explain why surrounding lexical information is needed by the learner in order to classify such cases. They also suggest that this learning is mostly superficial, that is, the learner just memorizes some examples, but these will not generalize well on test data. Indeed, the most common class of errors reported in Goldberg *et al.* , 2006 are of the split/merge type. These are followed by conjunction related errors, which suffer from the same problem. Morphological features of *smixut* and agreement can help to some extent, but this is still a limited solution. It seems that deciding the [NP][NP] case is beyond the capabilities of chunking with local context features alone, and more global features should be sought.

4.2 Facilitating Better Learning

This section presents preliminary results using Anchored Learning for better NP chunking. We present a setting (English Base NP chunking) in which selected features coupled together with anchored learning show an improvement over previous results.

Section 3.6 hinted that SVM based chunking might be hurt by using too many lexical features. Specifically, the features w_{-2},w_{+2} were shown to cause the chunker to overfit in English chunking. Learning without these features, however, yields lower results. This can be overcome by introducing anchors as a substitute. Anchors play the same role as rare features when learning, while lowering the chance of misleading the classifier on test data.

The results of the experiment using 5-fold cross validation on ENG indicate that the F-score improves on average from 93.95 to 94.10 when using anchors instead of $w_{\pm 2}$ (+0.15), while just ignoring the $w_{\pm 2}$ features drops the F-score by 0.10. The improvement is minor but consistent. Its implication is that anchors can substitute for "irrelevant" lexical features for better learning results. In future work, we will experiment with better informed sets of lexical features mixed with anchors.

5 Conclusion

We have introduced two novel methods to understand the inner structure of SVM-learned models. We have applied these techniques to Hebrew NP chunking, and demonstrated that the learned model is robust in the presence of noise in the PoS tags, and relies on only a few lexical features. We have identified corpus errors, better understood the nature of the task in Hebrew – and compared it quantitatively to the task in English.

The methods provide general insight in the way SVM classification works for chunking.

References

S. Abney, R. Schapire, and Y. Singer. 1999. Boosting applied to tagging and PP attachment. *EMNLP-1999.*

M. Adler and M. Elhadad. 2006. An unsupervised morpheme-based hmm for hebrew morphological disambiguation. In *COLING/ACL2006.*

C. Cardie and D. Pierce. 1998. Error-driven pruning of treebank grammars for base noun phrase identification. In *ACL-1998.*

Y. Goldberg, M. Adler, and M. Elhadad. 2006. Noun phrase chunking in hebrew: Influence of lexical and morphological features. In *COLING/ACL2006.*

T. Kudo and Y. Matsumoto. 2000. Use of support vector learning for chunk identification. In *CoNLL-2000.*

T. Kudo and Y. Matsumoto. 2003. Fast methods for kernel-based text analysis. In *ACL-2003.*

M. Marcus and L. Ramshaw. 1995. Text Chunking Using Transformation-Based Learning. In *Proc. of the 3rd ACL Workshop on Very Large Corpora.*

T. Nakagawa and Y. Matsumoto. 2002. Detecting errors in corpora using support vector machines. In *COLING-2002.*

Erik F. Tjong Kim Sang and S. Buchholz. 2000. Introduction to the conll-2000 shared task: chunking. In *CoNLL-2000.*

K. Sima'an, A. Itai, Y. Winter, A. Altman, and N. Nativ. 2001. Building a tree-bank of modern hebrew text. *Traitement Automatique des Langues*, 42(2).

V. Vapnik. 1995. *The nature of statistical learning theory.* Springer-Verlag New York, Inc.