Institute for Natural Language Processing

University of Stuttgart
Pfaffenwaldring 5b
D–70569 Stuttgart

Diploma Thesis No. 3399

Towards Robust Cross-Domain Domain Adaptation for Part-of-Speech Tagging

Tobias Schnabel

Course of Study:	Computer Science
Examiner:	Prof. Dr. Hinrich Schütze
Supervisor:	Prof. Dr. Hinrich Schütze
Commenced:	2012-10-22
Completed:	2013-04-23
CR-Classification:	I.2.7

Abstract

Most systems in natural language processing experience a substantial loss in performance when the data that the system is tested with differs significantly from the data that the system has been trained on. Systems for part-of-speech (POS) tagging, for example, are typically trained on newspaper texts but are often applied to texts of other domains such as medical texts. Domain adaptation (DA) techniques seek to improve such systems so that they are able to achieve consistently good performance – independent of the domains at hand.

We investigate the robustness of domain adaptation representations and methods across target domains using part-of-speech tagging as a case study. We find that there is no single representation and method that works equally well for all target domains. In particular, there are large differences between target domains that are more similar to the source domain and those that are less similar.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Domain adaptation (DA) is the problem of adapting a statistical classifier that was trained on a source domain to a target domain (TD) for which no or little training data is available. Many applications are in need of domain adaptation since annotating training data usually is an expensive task. Also, being able to analyze large data sets gets more and more important in various fields. In natural language processing (NLP), for example, researchers aim at extracting information from the huge amounts of data available on the web. In these scenarios, robust techniques are needed due to the variability in such data.

We present a case study that investigates the robustness of domain adaptation across six different target domains for part-of-speech (POS) tagging, an important preprocessing task in many NLP applications. In part-of-speech tagging, we need to assign each word of a sentence its corresponding grammatical category, called *POS tag*. The two main information sources in POS tagging are *context* – which parts-of-speech are possible in a particular syntactic context – and *lexical bias* – the prior probability distribution of parts-of-speech for each word. We address domain adaptation for lexical bias in this work, focusing on unknown words; they are most difficult to handle in domain adaptation because no direct information about their possible parts-of-speech is available in the source domain training set. Since typical target domains contain a high percentage of unknown words, a substantial gain in the overall performance can be achieved by improving tagging for these words.

We address a problem setting where – in addition to *labeled source domain data* – a large amount of *unlabeled target domain data* is available, but *no labeled target domain data*. This setting is often called unsupervised domain adaptation (cf. Daumé III, 2007).

Most prior work on domain adaptation has either been on a single target domain, on two or more tasks – which results in an experimental setup in which two variables change at the same time, task and target domain – or has not systematically investigated how robust different features and different domain-adaptation approaches are.

We make three contributions in this work. First, we systematically investigate the cross-target-domain robustness of different representations and methods. We show that there are some elements of domain-adaptation setups used in the literature that are robust across target domains – e.g., the use of distributional information – but that many others are not, including dimensionality reduction and shape information.

Second, we present an analysis that shows that there are two important factors that influence cross-target-domain variation: (i) the magnitude of the difference in distributional properties between source domain and target domain – more similar target domains require other methods than less similar target domains and (ii) the evaluation measures used for performance. Since in unsupervised domain adaptation we optimize learning criteria on a source domain that can be quite different from the target domains, different target domain evaluation measures can diverge more in domain adaptation than in standard supervised learning settings when comparing learning methods.

Our third contribution is that we show that if we succeed in selecting an appropriate domain-adaptation method for a target domain, then performance improves significantly. We establish baselines for unknown words for the five target domains of the SANCL 2012 shared task and present the best domain-adaptation results for unknown words on the Penn BioTreebank. Our improvements on this data set (by 10% compared to published results) are largely due to a new domain-adaptation technique we call training set filtering. We restrict the training set to long words whose distribution is more similar to unknown words than that of words in general.

The next chapter introduces all basic methodologies used in this work. Chapter 3 presents a case study on part-of-speech tagging for one fixed target domain. We test several sources of information for their robustness in Chapter 4 and Chapter 5, again, only for one domain. We use the most promising sources of information for a comparison across six different target domains in Chapter 6. Chapter 7 discusses related work. Chapter 8 concludes and presents possible future research directions.

1.1 Data

In this work, we study data from one source domain and six target domains. While the preliminary experiments in Chapter 3 to 5 are conducted using data from just one target domain, we evaluate our methods on six different target domains in Chapter 6.

In the source domain, we have access to a labeled training set and an additional training set containing unlabeled data. We use the suffixes -train and -ul to discriminate among the latter. In any target domain, we use two distinct data sets, called *development set* and *test set*. We tune any parameters that an algorithm may have on the development set and make changes to the algorithm as necessary. For the final results, we run the final version of each algorithm without further changes on the test set. We use the suffixes -dev and -test in addition to the domain name to refer to a specific dataset.

1.1.1 Source Domain

Our source domain is the Penn Treebank (Marcus et al., 1993) of Wall Street Journal (WSJ) text. Following Blitzer et al. (2006), we use sections 2-21 for training our part-of-speech models. We also use 100,000 WSJ sentences from 1988 as unlabeled data in training.

1.1.2 Target Domains

We evaluate on six different target domains. The first target domain is the Penn BioTreebank data set distributed by Blitzer. It consists of development and test sets of 500 sentences each and an unlabeled set of 100,000 sentences of BIO text on cancer.

The remaining five target domains (newsgroups, weblogs, reviews, answers, emails) are from the SANCL shared task (Petrov and McDonald, 2012). We will use WEB to refer to these five target domains collectively. Each WEB target domain has an unlabeled training set of 100,000 sentences and development and test sets of about 1000 labeled

dataset	domain	oov rate
WEB	answers	8.53
	emails	10.55
	newsgroups	10.34
	reviews	6.84
	weblogs	8.44
BIO	bio	19.86

Table 1.1: Percentages of out-of-vocabulary words in the development set of each target domain.

sentences each. WEB and BIO tag sets differ slightly; we use them as published without modifications to make our results directly comparable to the benchmarks.

We define the *target domain repository* (denoted by $\mathcal{D}_T^{u,l}$) for a target domain as the union of development set and unlabeled data available for that target domain. $\mathcal{D}_{S,T}^{u,l}$ is the union of the source data (labeled and unlabeled WSJ) and $\mathcal{D}_T^{u,l}$.

By looking at Table 1.1, we can see that all target domains contain relatively high percentages of unknown words. While domains from the WEB data set contain roughly 7-11% out-of-vocabulary (OOV) words, almost 20% of all words in BIO are unknown. We can make two observations here. First, as all target domains have a considerably amount of unknown words, any tagger striving for high accuracies should be able to tag them correctly. This also supports our claim in the introduction above; focusing on unknown words is indeed a promising approach for improving tagging across domains. Second, these numbers suggest that target domains from WEB behave relatively similar to each other with respect to their OOV rates, but there are rather large differences between texts from WEB and BIO. One explanation might be that texts from WEB are user-generated content whereas BIO contains peer-reviewed MEDLINE abstracts. We discuss these discrepancies in Section 6.3 more thoroughly.

Chapter 2

Fundamentals

This chapter briefly covers the most important concepts used throughout this work. Section 2.1 introduces the main task of this work, part-of-speech tagging. Most state-of-the-art taggers incorporate statistical information from the data when building a model. We discuss these so-called stochastic taggers in Section 2.2. For our experiments, we frame part-of-speech tagging also as classification by omitting sequence information. Section 2.3 describes this approach in greater detail.

The remaining sections cover two general methods that we apply to POS-tagging: domain adaptation and singular value decomposition. Section 2.4 gives a definition of the task of domain adaptation and reviews previous work on it. Singular value decomposition, a technique that can project data from a high-dimensional vector space into a lower dimensional one, is addressed in Section 2.5. We will later apply this technique to our feature vectors in order to obtain more compact representations of them.

2.1 Part-of-Speech Tagging

In part-of-speech (POS) tagging, we are given a set of part-of-speech tags (or *tagset*) and a sentence split into separate tokens. The task is to assign the single tokens of a sentence their correct part-of-speech tags. Although there is no exact definition for parts-of-speech, they are often motivated by grouping words with similar syntactic or morphological behaviors into classes. Depending on whether one can add new words to a class, classes are called open or closed. Determiners, for example, make up a closed class as there are only three of them in English ("a", "an" and "the"), whereas one can always add new names to the open class of proper nouns.

Although many different tagsets exist for English, the Penn Treebank tagset (Marcus et al., 1993) has become a de facto standard in tagging as most results get published for it. It comprises 45 different part-of-speech tags which are shown in Table 2.1. However, in some scenarios, a finer tag set is needed. The Treebank tagset is often augmented by new tags in such cases in order to be able to make finer-grained distinctions.

POS tagging is often part of the preprocessing pipeline of larger frameworks. A prominent example is the use of POS tags to recover the syntactical structure of a sentence in parsing. POS tags can also help with speech synthesis to disambiguate between different pronunciations, like adult as a noun or adult as an adjective. Knowing a word's POS tag can also be useful in stemming as it can tell us which suffixes a word can take. In addition, POS tags are often included as features in classification tasks such as word sense disambiguation (WSD) or coreference resolution. Yet another use of POS tags is in information retrieval, where, words in the query might be filtered based on their tags.

What makes tagging a non-trivial task is tag ambiguity. Jurafsky and Martin (2008) report that about 18.5% of all words in the vocabulary of the Brown corpus are ambiguous. Moreover, over 40% of the tokens have more than one possible tag. Most of the ambiguous tokens, however, are easy to tag because not all tags are equally likely for a word. A common approach is to use a word's context, i.e. the tags of its neighboring words, in such ambiguous cases. Context provides an algorithm with essential information because POS tags also reflect a word's syntactic properties. A typical example for meaningful context information is whether a word's left neighbor is a determiner. Given the word "bank", we can then disambiguate between "bank" as a noun and "bank" as a verb.

Most tagging algorithms can be broadly put into two categories: linguistic taggers and stochastic taggers. Linguistic taggers, like the EngCG tagger (Voutilainen, 1995) usually involve a large set of hand-crafted rules and a lexicon. After looking up all possible parts-of-speech for a word, constraints from the ruleset are used to filter out all invalid assignments. Coming up with a good set of rules is not only a tedious task; once finished, it needs to be redone when moving on to other languages or application domains. These are just two reasons why linguistic taggers have become less popular in recent years. Indeed, all current state-of-the-art taggers are stochastic taggers (ACL Wiki, 2013).

Stochastic taggers are able to handle the issues of linguistic taggers more gracefully; they are able to generalize from training examples themselves. Also, as long as the underlying modeling assumptions are not violated, stochastic taggers can be applied to new languages at no or only little extra cost. Stochastic taggers are sometimes also called data-driven, as they extract their knowledge solely from a set of labeled training examples. Given a sentence, stochastic taggers are then able to estimate the probability for each possible assignment of tags to the sentence. Finally, the tagger chooses the assignment that was most likely for this sentence. We introduce three popular stochastic tagging algorithms in Section 2.2: Hidden Markov Models, Maximum Entropy Markov Models and Conditional Random Fields.

2.1.1 Practical Issues

When developing new methods and testing new approaches, it is common practice to have a rather simple method to compare results with. In POS-tagging, one frequently used baseline is assigning the most frequent tag to each known word and the overall most frequent tag to all unknown ones. This baseline usually does fairly well; it achieves accuracies in the range of 90-92% as reported by Charniak (1997). Although DeRose (1988) found that roughly 40% of the tokens in the Brown corpus were ambiguous, many ambiguous words have only one prevailing tag which makes the most frequent tag baseline become so competitive.

A problem that all taggers have to face in practice is unknown words. Even large corpora won't contain all proper nouns or acronyms possible. A simple approach is to assume that each tag is equally likely for an unknown word, forcing the tagger to rely solely on the contextual information it has. However, it appears unfair to assume that each tag occurs with equal probability for an unknown word, for example because unknown words are very unlikely to have tags from a closed class like determiners. Thus, Dermatas and Kokkinakis (1995) and Baayen and Sproat (1996) both suggest to use the tag distribution of words occurring only once in the corpus, also known as *hapax legomena*, as an approximation for the tag distribution of unknown words.

tag	description	example	tag	description	example
CC	coordin. conj.	and, but	SYM	symbol	+,%, &
CD	cardinal number	one, three	TO	“to”	to
DT	determiner	a, the	UH	interjection	ah, oops
EX	existential ’there’	there	VB	verb, base form	eat
FW	foreign word	mea culpa	VBD	verb, past tense	ate
IN	prep./sub-conj	of, in, by	VBG	verb, gerund	eating
JJ	adjective	yellow	VBN	verb, past part.	eaten
JJR	adj., comparative	bigger	VBP	verb, non-3sg pres	eat
JJS	adj., superlative	wildest	VBZ	verb, 3sg pres	eats
LS	list item marker	I, 2, One	WDT	wh-determiner	which, that
MD	modal	should	WP	wh-pronoun	what, who
NN	noun, sing. or mass	llama	WP$	possessive wh-	whose
NNS	noun, plural	llamas	WRB	wh-adverb	how, where
NNP	proper noun, sing.	IBM	$	dollar sign	$
NNPS	proper noun, pl.	Carolinas	#	pound sign	#
PDT	predeterminer	alt, both	“	left quote	‘ or “
POS	possessive ending	’s	”	right quote	’ or ”
PRP	personal pron.	I, you, he	(	left parenthesis	[, (, {, <
PRP$	possessive pron.	one’s	)	right parenthesis	],), }, >
RB	adverb	quickly	,	comma	,
RBR	adverb, comp.	faster	.	sent.-final punc	. ! ?
RBS	adverb, super.	fastest	;	mid-sent. punc	: ; - – ...
RP	particle	up, off			

Table 2.1: The Penn Treebank tagset. Examples were borrowed from Jurafsky and Martin (2008).

More sophisticated methods for unknown-word handling try to harness a word's morphological or orthographic features. Morphological features, such as suffixes can be a good indicator for certain tags like adverbs. Orthographic features also often provide useful information. For example, an uppercase word is more likely to be a proper noun due to spelling conventions. Other common orthographic features look for special characters in a word, such as hyphens or digits (Toutanova et al., 2003).

2.2 Stochastic Taggers

Stochastic taggers interpret POS-tagging as a special case in the general framework of sequence prediction. In sequence prediction, we are given a sequence of observations $\mathbf{x} = (\tilde{x}_1, \ldots, \tilde{x}_T)$ belonging to a sequence of unknown states $\mathbf{y} = (y_1, \ldots, y_T)$. Our goal is to find the sequence of states $\mathbf{y}$ that is most likely for a given sequence of observations $\mathbf{x}$. In other words, we wish to find

$$\begin{aligned} h(\mathbf{x}) &= \mathrm{argmax}_{\mathbf{y}}\, P(\mathbf{y} \mid \mathbf{x}) && (2.1)\\ &= \mathrm{argmax}_{\mathbf{y}}\, P(\mathbf{x}, \mathbf{y}) && (2.2)\\ &= \mathrm{argmax}_{\mathbf{y}}\, P(\mathbf{x} \mid \mathbf{y})\, P(\mathbf{y}). && (2.3) \end{aligned}$$

In the context of POS-tagging, the sequence of observations corresponds to the tokens of a sentence and the sequence of states to the tags. Finding the argmax will thus return the most likely sequence of tags for a sentence. Note that the last equation gives us three equivalent ways of expressing a model for POS-tagging. For example, we could try to model the joint distribution $P(X, Y)$ directly. However, this is rather inefficient as we would have to model all dependencies in the joint distribution. This is why most stochastic taggers either build a model for the conditional probability $P(Y \mid X)$ or just find separate models for both the prior probability $P(Y)$ and class conditional probability $P(X \mid Y)$.

When working with stochastic taggers, one usually keeps three distinct data sets around. Each data set contains a number of manually tagged sentences. The first data set, called *training set*, is used to train the tagger, e.g. is used to find estimates for the probability distributions. In the next step, the *development set* is used to perhaps tune some parameters. Finally, the tagger is evaluated on the *test set* with no further parameter changes. Typically, most of the available data is used for training (about 70% to 80%); the remaining data is withheld for the development and test set.

2.2.1 Hidden Markov Models

Hidden Markov Models are one of the best known stochastic algorithms in part-of-speech tagging. They became popular in the mid 1980s as researchers started building systems using HMMs for tag disambiguation (i.e. Garside, 1987; DeRose, 1988). Although HMMs are rarely used in their basic form nowadays, it is important to understand the principles of HMMs as background for more sophisticated tagging algorithms. In this section, we introduce HMMs with focus on part-of-speech tagging. For a more general introduction to HMMs, see Russell and Norvig (2010).

Hidden Markov Models belong to the group of generative models as they assume the data to be generated according to a joint distribution $P(X, Y)$. HMMs model $P(X, Y)$

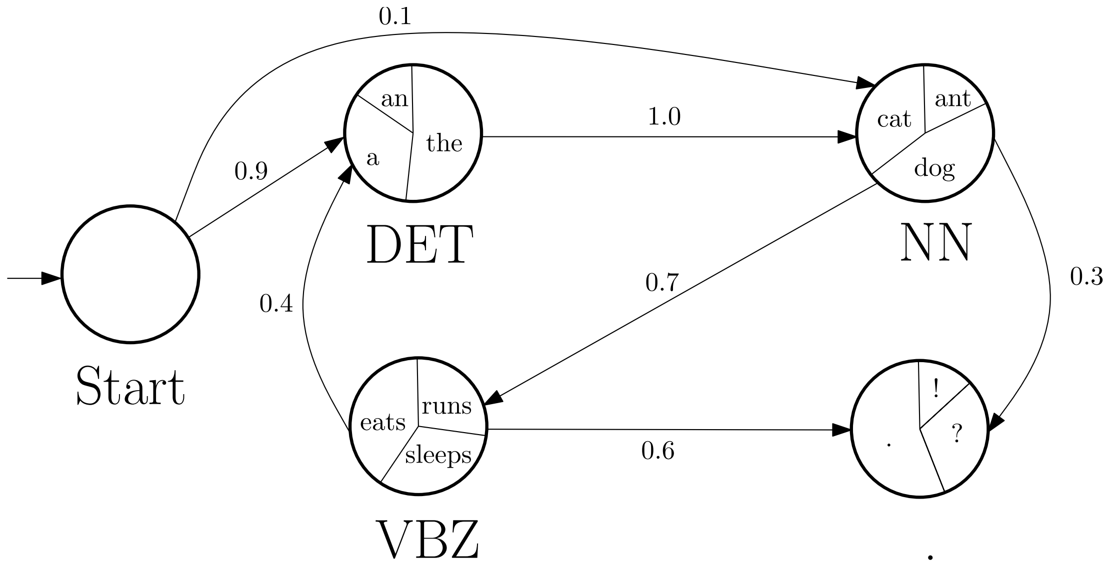

Figure 2.1: Graphical representation of a simple first-order Hidden Markov Model. The states correspond to single POS tags. The numbers on the arcs indicate the probabilities for transitioning from one state to another. Each state can emit a number of words; the probability of emitting a word in a specific state is represented by the area in the pie chart. The process of generating data is equivalent to a random walk on this graph. Beginning in the start state, we would repeatedly choose a transition to a new state and then a word to emit in this new state.

as two separate distributions following Equation 2.3: a prior probability $P(Y)$ and a conditional probability $P(X \mid Y)$. Additionally, HMMs make some simplifying assumptions in order to compute the probability distributions efficiently. More specifically, they use the following assumptions:

1. The probability of observing a part-of-speech tag is only dependent on the last $n-1$ tags. In language modeling, this corresponds to having an n-gram model for the sequence of tags. The probability of transition from one state to another is also referred to as transition probability. If only the last tag is conditioned on, the resulting HMM is called first-order. The prior probability for a tag sequence in such a first-order HMM is computed as:

$$P(\mathbf{y}) = P(y_1) \prod_{t=2}^{T} P(y_t \mid y_{t-1}).$$

2. The probability of observing a token only depends on the current tag and is independent of all other tokens:

$$P(\mathbf{x} \mid \mathbf{y}) = \prod_{t=1}^{T} P(\tilde{x}_t \mid y_t).$$

Plugging in the two assumptions into equation 2.3 yields the final equation for finding the most likely tag sequence with a first-order HMM:

$$h(\mathbf{x}) = \text{argmax}_{\mathbf{y}}\, P(\mathbf{x} \mid \mathbf{y}) \cdot P(\mathbf{y}) \tag{2.4}$$

$$= \text{argmax}_{\mathbf{y}}\, P(y_1)P(\tilde{x}_1 \mid y_1) \prod_{t=2}^{T} P(y_t \mid y_{t-1})\, P(\tilde{x}_t \mid y_t). \tag{2.5}$$

Given a sentence $\mathbf{x}$, how can we then find the most likely sequence of tags for it? A rather naive approach would try out every single combination of tags $\mathbf{y}$ and then return the one that yielded the maximum probability. However, if a sentence is of length T and we have N different POS-tags, then there are N^T possible sequences of tags we need to consider. In other words, the run-time of this naive algorithm increases exponentially with the sentence length, making tagging unfeasible for practical purposes. As it turns out, there is a faster algorithm for finding the most likely sequence of tags. The Viterbi algorithm, proposed by Viterbi (1967), has become one of the standard algorithms in tagging. It decomposes equation 2.5 into a set of recurrence equations and then applies dynamic programming to them in order to compute them efficiently. Its final run-time is $O(TN^2)$, i.e. it grows only linearly in the length of a sentence.

Let's take an example to get an intuition for the probability distribution that a HMM models. Figure 2.1 shows a simplistic HMM with only four POS-tags and a small vocabulary. To compute the probability of an annotated sentence, we begin in the start state and then move from one state to the other in the same order that the tags appear in the sentence. In each state except the start state, we have to multiply the probability of observing the current word in this state and the probability of transitioning from the previous state to the current state with the running total probability.

Say we want to compute the probability for observing the annotated sentence

"The/DET dog/NN sleeps/VBZ ./."

Thus, we want to calculate $P(\mathbf{x}, \mathbf{y})$ for $\mathbf{x} =$"The dog sleeps." and $\mathbf{y} =$"DET NN VBZ .". Following the instructions from the previous paragraph yields:

$$\begin{aligned} P(\mathbf{x}, \mathbf{y}) = & P(\text{DET} \mid \text{START})\, P(\text{the} \mid \text{DET})\, P(\text{NN} \mid \text{DET})\, P(\text{dog} \mid \text{NN}) \\ & P(\text{VBZ} \mid \text{NN})\, P(\text{sleeps} \mid \text{VBZ})\, P(. \mid \text{VBZ})\, P(. \mid .) \end{aligned}$$

One could now plug in numbers for the probabilities using Figure 2.1. We leave it to the reader to choose concrete numbers for the emission probabilities as they are only given graphically.

2.2.2 Maximum Entropy Markov Models

Traditional HMMs have two shortcomings. Recall that our goal was to find the sequence of tags that is most likely for a given sequence of words. Instead of maximizing $P(Y \mid X)$ directly, HMMs decompose this probability and make an effort to estimate the prior probability $P(Y)$ and conditional probability $P(X \mid Y)$ separately. The problem here is that the $P(Y)$ needs to be modeled although it isn't actually needed. Moreover, HMMs assume features X to be independent of each other when modeling $P(X \mid Y)$. Also, if one has additional features that one would like to integrate with the tagger, there is no natural way to do so with HMMs. Maximum Entropy Models or MEMMs introduced by

McCallum et al. (2000) are designed to overcome these drawbacks. First, they directly model the conditional probability $P(Y \mid X)$ in

$$h(\mathbf{x}) = \mathrm{argmax}_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x}) \quad (2.6)$$

$$= \mathrm{argmax}_{\mathbf{y}} P(y_1 \mid \tilde{\mathbf{x}}_1) \prod_{t=2}^{T} P(y_t \mid y_{t-1}, \tilde{\mathbf{x}}_t), \quad (2.7)$$

making no further independence assumptions on the features X.

Second, MEMMs are able to include knowledge into the tagging process by training a local classifier for each state transition. Note that $\tilde{\mathbf{x}}_t$ is a vector here, indicating that we are free to include information from any timestep t in the current observation vector $\tilde{\mathbf{x}}_t$. We could for example include the first word $\tilde{x}_1$ of a sentence in $\tilde{\mathbf{x}}_t$, or information about the next word $\tilde{x}_{t+1}$. For each tag y_{prev} in Y, we build a so-called Maximum Entropy model that estimates the probability of a transition from y_{prev} to a tag y as follows:

$$P(y \mid y_{prev}, \tilde{\mathbf{x}}) = \frac{1}{Z(\tilde{\mathbf{x}}, y_{prev})} \exp\left(\sum_i w_i f_i(\tilde{\mathbf{x}}, y)\right). \quad (2.8)$$

That is, given a previous tag y_{prev} and observation $\tilde{\mathbf{x}}$, the local classifier gives us a probability distribution over the successor tags y. $Z(\tilde{\mathbf{x}}, y_{prev})$ is used as a normalization function, ensuring that the single probabilities sum up to 1. The features are modeled as $f_i(\tilde{\mathbf{x}}, y)$ here and can include information from the current tag y and current input observation $\tilde{\mathbf{x}}$. Note that each feature function $f_i(\tilde{\mathbf{x}}, y)$ is implicitly conditioned on y_{prev} because we construct separate features for each $y_{prev} \in Y$. MEMMs allow us to define a large range of features, encompassing the ones we used for HMMs. For example, we can model the transition probabilities of a HMM as:

$$f_k(\tilde{\mathbf{x}}, y) = \begin{cases} 1 & \text{if } y = k \\ 0 & \text{otherwise} \end{cases} \quad (2.9)$$

for each state k. Similarly, we can define functions that model the emission probabilities for each state-observation pair (k, s) as follows:

$$f_{ks}(\tilde{\mathbf{x}}, y) = \begin{cases} 1 & \text{if } y = k \text{ AND } \tilde{\mathbf{x}} = s \\ 0 & \text{otherwise} \end{cases} \quad (2.10)$$

All features are multiplied by a corresponding weight w_i. The weights for each classifier can be found in a supervised training step so as to maximize the likelihood of the training data.

Per-state normalization of the probabilities with $Z(\tilde{\mathbf{x}}, y_{prev})$ leads to a problematic side-effect called *label bias* (Lafferty et al., 2001). In a nutshell, tags that have only few outgoing transitions will tend to ignore their inputs. In the extreme case, a tag with only one successor tag will always return a probability value of 1, completely independent of its input observations. Lafferty et al. (2001) showed that due to this effect, MEMMs can perform significantly worse on synthetic POS-datasets than HMMs which do not suffer from this problem.

2.2.3 Conditional Random Fields

Conditional Random Fields (CRFs) are another popular approach to sequence classification. CRFs form a special case of undirected graphical models. Dependencies between

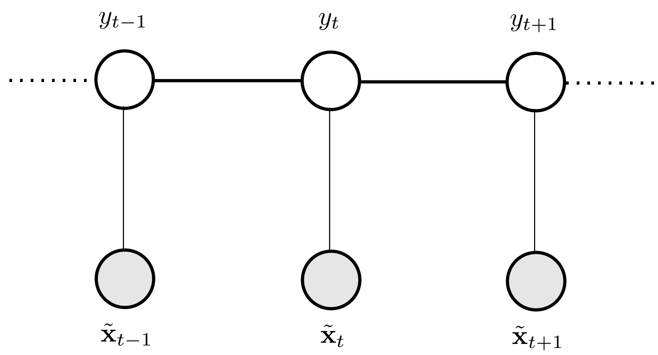

Figure 2.2: Graphical structure of a linear-chain Conditional Random Field (CRF). Random variables are represented by nodes. An edge between two random variables indicates conditional dependency.

random variables in such models are often represented by graphs. Although the structure might almost be arbitrary with CRFs, most common in POS-tagging are linear-chain CRFs as shown in Figure 2.2. The general structure resembles the one of a HMM; indeed, it can be shown that any HMM can be expressed as a CRF (Sutton and McCallum, 2006). CRFs are also similar to MEMMs, however, they avoid the label bias problem since they employ global normalization. Instead of training separate local models like MEMMs for each state, CRFs train a single model for the entire sequence. The conditional probability of a sequence $\mathbf{y}$ given the observations $\mathbf{x}$ is:

$$P\left(\mathbf{y} \mid \mathbf{x}\right) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \exp\left(\sum_i w_i f_i(\tilde{\mathbf{x}}_t, y_t, y_{t-1})\right). \tag{2.11}$$

Again, $Z(\mathbf{x})$ is a normalization function making sure that the distribution sums to 1. The feature functions $f_i(\tilde{\mathbf{x}}_t, y_t, y_{t-1})$ can encode information from three sources: Information from the current observation $\tilde{\mathbf{x}}_t$ and information from the current state y_t and previous state y_{t-1}. Linear-chain CRFs allow us to define similar features as MEMMs. For example, we can define features to model the transition probabilities of a HMM as:

$$f_{kl}\left(\tilde{\mathbf{x}}_t, y_t, y_{t-1}\right) = \begin{cases} 1 & \text{if } y_t = k \text{ AND } y_{t-1} = l \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

for each transition pair (k, l). Likewise, we can model the emission probabilities for each state-observation pair (k, s) as follows:

$$f_{ks}\left(\tilde{\mathbf{x}}_t, y_t, y_{t-1}\right) = \begin{cases} 1 & \text{if } y_t = k \text{ AND } \tilde{x}_t = s \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

Although CRFs offer many advantages, they also suffer from a couple of drawbacks. For example, having more powerful features increases the risk of overfitting the training data. While many regularization methods exist, all of them introduce new parameters that need to be tuned. Another weakness is that CRFs are computationally expensive to train. When training CRFs, probabilities need to be optimized for whole sequences using some form of the backward-forward algorithm. With MEMMs, however, one can estimate each of the local distributions in isolation.

2.3 Tagging as Classification

Classification refers to the task of determining to which class $y \in Y$ an object $\mathbf{x} \in X$ belongs. If objects can only belong to two classes, e.g. $Y = \{-1, +1\}$, the classification task is called binary. If Y contains more than 2 classes, the resulting problem is called multiclass classification. Classification in the standard setting is a supervised task, e.g. the classifier is given a set of M labeled training examples $\mathcal{D}^l = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)\}$ to learn from. During training, the classifier finds a model for the data using the training instances provided as input. Once the training is finished, we can classify unknown objects with the help of this model.

Remember that stochastic taggers look for a likely sequence of tags which, at the same time, is also likely for the sequence of observations. One idea that we use in this work is to tag words using classification rather than sequence classification. We can frame tagging as multiclass classification if we do not incorporate sequence information. This means that we train a classifier only on local information about a word that does not include the POS tags of neighboring words, thereby completely neglecting transition probabilities. Vice versa, most local classifiers can be extended with Viterbi decoding to obtain a global model, like support vector machines (Giménez and Màrquez, 2004) or MaxEnt classifiers for MEMMs (McCallum et al., 2000).

2.3.1 k-Nearest-Neighbors

The k-nearest-neighbor (k-NN) method is one of the simplest classification algorithms. An unknown object is simply classified by assigning it the majority class label among its k nearest neighbors. More formally, let $\text{count}_k(\mathbf{x}, y)$ be the function that retrieves the number training instances with class label y among the k nearest neighbors of $\mathbf{x}$ using some similarity measure sim. We then determine the most frequent label of the neighbors:

$$h_k(\mathbf{x}) = \text{argmax}_{y \in Y} \text{count}_k(\mathbf{x}, y). \tag{2.14}$$

There are two important hyperparameters to tune. The choice of the metric sim can have a huge impact on performance. Depending on sim, different instances are included in the set of nearest neighbors. Common choices of the similarity metric involve the Euclidean metric or cosine similarity. Cosine similarity is often used in text mining application as feature representations are sparse. It measures the cosine of the angle between two vectors $\mathbf{x}$ and $\mathbf{y}$ as

$$\text{sim}_{cos}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \, \|\mathbf{y}\|}.$$

Additionally, we need to select an appropriate value for k, the number of neighbors to consider. The larger the value of k, the greater the smoothing effect on the data gets. While this can help to eliminate noise in the data, having a large value for k will also cause boundaries between classes to be less distinct. As the optimal value of k is problem-specific, it is often determined on a held-out development set.

Compared to other classification methods, k-NN has a couple of favorable properties. As mentioned before, k-NN is easy to implement. This is also due to the fact that k-NN is a lazy method, meaning that it doesn't require training; it merely has to store all training instances. Another amenable property of k-NN is that it extends naturally to multiclass classification. The algorithm simply returns the majority class label oblivious of the number of classes.

One obvious weakness of k-NN is the computational effort when classifying an object. In order to find an object's nearest neighbors, k-NN needs to compute the distance between the input object and all other data points. However, many approaches exist to ease the computational efforts of k-NN. Some approaches try to store the training examples more intelligently to speed-up classification, for example using tree-like structures (Bentley, 1980). Other approaches aim at reducing the total number of data points to be stored, like condensed k-NN (Hart, 1968).

Despite its conceptual simplicity, k-NN still gets used in practice today. Søgaard (2011) used a variant of the condensed k-NN algorithm on top of the output of two taggers, one unsupervised tagger and one supervised tagger. His results are currently ranked first among all state-of-the-art taggers, yielding an accuracy of 97.50% on the Wall Street Journal.

2.3.2 Support Vector Machines

Another widely used classification method is the support vector machine (SVM). In its basic form, an SVM is a linear binary classifier, meaning that it determines a hyperplane with normal vector $\mathbf{w}$ and offset $\mathbf{b}$ separating all positive training instances from the negative ones. After determining that hyperplane, we can classify a new object $\mathbf{x}$ as follows:

$$h_{\mathbf{w},\mathbf{b}}(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{x} \cdot \mathbf{w} + \mathbf{b} > 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.15}$$

Sometimes, there is more than one hyperplane that would separate the training data. SVMs thus employ a second criterion for choosing an optimal hyperplane. They try to find the hyperplane yielding the largest possible margin between the two classes. This is thought to make the SVM less susceptible to noise as small deviations in the input can be tolerated. Since not all data in practice can be separated by a linear hyperplane, soft-margin SVMs relax the classification problem by introducing slack variables ξ. These slack variables allow for misclassified instances during training. More specifically, soft-margin SVMs solve the following optimization problem:

$$\min_{\mathbf{w},\mathbf{b},\xi} \; \frac{1}{2}\mathbf{w} \cdot \mathbf{w} + C \sum_{i=1}^{M} \xi_i \tag{2.16}$$

$$\begin{aligned} \text{subject to } & y_1(\mathbf{w} \cdot \mathbf{x}_1 + \mathbf{b}) \geq 1 - \xi_1 \quad, \xi_1 \geq 0 \\ & \cdots \\ & y_M(\mathbf{w} \cdot \mathbf{x}_M + \mathbf{b}) \geq 1 - \xi_M, \xi_M \geq 0 \end{aligned} \tag{2.17}$$

Equation 2.16 basically says that we are striving for a large margin (which is equivalent to minimizing $\frac{1}{2}\mathbf{w} \cdot \mathbf{w}$), while, at the same time, violating the constraints as little as possible. The constraints express the objective to have each training instance $(\mathbf{x}_i, y_i)$ classified correctly. Note that we have to choose C, a hyperparameter controlling how strongly we penalize incorrectly classified instances during training. If C is large, we will usually make fewer mistakes during training; however, we get this at the cost of having smaller margins. Vice versa, a small value for C will increase training error rates but will give us larger margins. In practice, one often determines an optimal value for C on a held-out development set.

Up to now, we have only presented SVMs in the context of binary classification. However, there are also numerous solutions for multiclass problems. One approach is to restate the optimization objective so that it directly allows for multiple classes as proposed by Crammer and Singer (2002). However, more frequent in practice is the reduction of multiclass classification to multiple binary classification tasks. Again, there are multiple ways of performing this reduction. One way is the one-against-all paradigm where we train one classifier for each class $y \in Y$. Input to each classifier is instances of a particular class labeled as positive examples, and all remaining instances being labeled as negative examples. We then assign an unknown object the class whose classifier had the greatest confidence. Another option is the one-vs-one paradigm where we train $\binom{|Y|}{2}$ classifier, one for each pair of class labels. Each classifier then votes for one of its two classes; the final label is determined by taking the class with the most votes. Although both approaches yield comparable classification results in practice, Hsu and Lin (2002) found the one-vs-one paradigm to be favorable due to shorter training times.

2.4 Domain Adaptation

Domain adaptation (DA) refers to the situation in which we have large amounts of labeled data from the domain we train our model in, the *source domain*, but only little or no labeled data from the domain where we want to apply our model, the *target domain* (Blitzer et al., 2006). This scenario often occurs in practice as building and annotating large corpora is usually a tedious and expensive task. In domain adaptation, we aim at developing techniques which are able to perform well in the target domain despite the lack of labeled training data.

Although many fields are in need of domain adaptation, we focus on applications of domain adaptation in natural language processing (NLP). One application of domain adaptation in NLP is spelling correction. There, we train a model on data collected from many users but may use domain adaptation to personalize the corrector for an individual user. Another example is in part-of-speech tagging where we may have newspaper articles as the source domain and biomedical texts as the target domain. As the tagger only has reliable information about words of the source domain, tagging performance is often significantly worse in the target new domain. We will observe similar tendencies in our case study in Chapter 3.

Note that the term “domain” refers to a somewhat loose concept; it does not necessarily have to depict a specific genre like movie reviews or recipes (Bandyopadhyay, 2012). Instead, we should understand the concept of a domain as the setting where we have data stemming from different distributions. Using this definition, we can see that the concept of a domain is a more continuous one as almost all real-word data exhibits some inconsistencies; even texts of the same author may vary from day to day. Whether we attribute inconsistencies in the data to noise or to different underlying distributions is somewhat arbitrary, although implications are different. Despite these problems, we try to give a more formal definition of the domain adaptation task in the next section.

2.4.1 Notation and Setting

We follow the notation introduced by Jiang (2008) and Chen et al. (2011) to describe the domain adaptation setting. Similar to classification, we use X to denote the random variable of an observation and Y for the random variable associated with the output labels. We assume that the data from the source domain is generated by an underlying joint distribution $P_S(X, Y)$. Likewise, we assume an unknown joint distribution generating data in the target domain $P_T(X, Y)$. Finally, let $\mathcal{D}_T^l$ and $\mathcal{D}_T^u$ denote the labeled training data and unlabeled training data from the source domain respectively. The data sets $\mathcal{D}_S^l$ and $\mathcal{D}_S^u$ are defined similarly for the source domain.

The problem in which we have both labeled data (and possibly unlabeled data) in the source and target domain, e.g. $\mathcal{D}_S^l$ and $\mathcal{D}_T^l$, is called *supervised domain adaptation*. If we only have access to labeled data in the source domain $\mathcal{D}_S^l$ and to unlabeled data $\mathcal{D}_S^u$ and $\mathcal{D}_T^u$, the problem is referred to as *unsupervised domain adaptation*. Usually, unsupervised adaptation is considered to be the harder task of the two, as links between the two domains can only be learned from unlabeled data (Daumé III et al., 2010).

Although creating an annotated data set $\mathcal{D}_T^l$ can be accelerated using active learning (Ringger et al., 2007), we cannot ensure that we will always have access to such data. Unlabeled data, in contrast, can be obtained easily in most cases. Most texts are available in an electronic version nowadays; in addition, we can extract large text collections from the web. Thus, we chose to focus on unsupervised domain adaptation in this work since we believe that the unsupervised domain-adaptation setting is more realistic than the supervised one. Consequently, we include only previous work in the survey below which can be applied in an unsupervised setting.

2.4.2 Unsupervised Approaches

Existing methods for unsupervised domain adaptation can be broadly put into three categories. Methods using representational learning, approaches which perform instance weighting, and classical bootstrapping methods. While methods in the first group are often specifically tailored to a task, approaches from the last two groups are usually applicable to a wider range of problems.

Representation learning. The basic idea of these approaches is to find robust representations which behave similarly across domains. Once a good feature representation is found, any supervised model using these features should be able to generalize well from the target domain. Since many approaches are application-specific, we focus on the ones that have been applied to the task of part-of-speech tagging.

Schütze (1995) showed that context is an important source of information for part-of-speech tagging. As part-of-speech tags are also motivated by syntactic behavior, knowing how a word is used in running text can provide valuable information. For example, when trying to tag a word, we could enumerate all contexts in which it occurs. Now, if we scan through this list and find that “the” occurred as a left neighbor in the corpus, we could infer that chances are high that it is a noun. We call a word’s neighbors *indicator words*, so if we refer to a word’s left indicator tokens, we mean the set of words that occurred to the left of a word in the corpus.

A well-known approach in representation learning is structural correspondence learning (SCL). Blitzer et al. (2006) applied SCL to the task of part-of-speech tagging. SCL exploits correlated features by defining a mapping from the full feature space to a space called the *pivot feature space*. Each dimension in the pivot feature space corresponds to the result of a binary classification problem. Input to each classifier is all unlabeled training samples. Each classifier gets assigned a separate problem which can be solved in an unsupervised manner. Blitzer, for example, uses pivot features of the form “this word’s left neighbor is w”, where w is some word in the vocabulary. There, pivot features rely on distributional information as well, however, they only use them indirectly when constructing the classification problems. Pivot features are supposed to bridge the gap between the source and target domain. If two words u and v have similar values for a pivot feature, they are assumed to also behave similarly with respect to their part-of-speech tags.

Although Blitzer et al. (2006) reported improved results for a POS-tagger using structural correspondence learning, there are some limitations to his approach. First, we need to define an appropriate set of pivot features. If we choose pivot features irrelevant to the task, these features could introduce additional noise to the data. Also, pivot features need to be frequent enough to have sufficient training data for the corresponding classification tasks. However, if pivot features are too frequent, they won't allow a supervised tagger to make fine-grained distinctions. Lastly, there are a couple of hyperparameters to set which may increase the risk of overfitting the training data.

Huang and Yates (2009) directly incorporate distributional information in their POS-tagging framework. They construct two feature vectors for each word; one encompassing all left neighbors of a word, and one encompassing all right neighbors. These feature vectors serve as input to a CRF tagger, either in their raw form or after applying dimension reduction to them. Like Schütze (1995), they examine the effects of using SVD as a dimensionality reduction technique. Besides SVD, they propose a HMM-based method for constructing a lower dimensional representation. They find better accuracies for their HMM method than Blitzer et al. (2006), however, they do not compare them against a CRF baseline using SVD feature vectors or raw feature vectors.

In their later work, Huang and Yates (2010) explore two modifications to their former approaches. First, they examine the benefits of using HMMs with multiple layers that are thought to increase the expressiveness of the feature space. The second extension they test is training by contrastive estimation which is able to handle unannotated data (Smith and Eisner, 2005). Using their augmented HMM, they were able to beat all previous results on the same POS-tagging task. In yet another work, Huang and Yates (2012) argue that finding an optimal feature representation is computationally intractable. Based on that argument, they propose a new framework which allows prior knowledge to be integrated into representation learning. They do this by introducing additional optimization criteria on the latent states of their HMMs. By doing so, one can incorporate various biases in the representation learning task helping to improve prediction.

Umansky-Pesin et al. (2010) use distributional information in a web-based framework. When encountering an unknown word, the algorithm runs a couple of web-queries to collect contexts in which this unknown word occurs. Contexts are then used to build feature vectors similar to Huang and Yates (2009). Finally, a MEMM tagger uses this information to predict the tag of this unknown word. Unlike other methods, this approach does not require a corpus from the new domain. However, using the whole web as corpus might add irrelevant contexts for the target domain at hand.

Bootstrapping. In bootstrapping (Yarowsky, 1995), we are given a small set of labeled data and a rather large set of unlabeled data. To improve performance, the learning algorithm augments its own training set with instances from the unlabeled data set. Note that the two bootstrapping methods we present here, self-training and co-training, can also be applied to scenarios other than domain adaptation.

Self-training (McClosky et al., 2006) can be used in conjunction with most learning algorithms. In order to make use of unlabeled data, an initial model is trained on the labeled training set. This model is then applied to the unlabeled data set. All newly labeled instances where the algorithm was most confident on get added to the training set and the model gets retrained. This process is iterated until some user-defined stopping criterion is reached. Huang et al. (2009) found that self-training might increase performance of HMM in POS tagging, but there are also cases in which a tagger didn't benefit from it. Huang and Yates (2010), however, reported small gains in accuracy when combining self-training and a CRF tagging model.

Co-training (Blum and Mitchell, 1998) is in the same spirit, but uses at least two different set of features, called *views*. During training, a model for each view is constructed and applied to the unlabeled data set. The unlabeled data set is then annotated and the most confident predictions from each view get added to the training set. Co-training works best when the views define conditionally independent sets of features and the learning algorithm can make accurate predictions using each view alone. Chen et al. (2011) present a variant of co-training for domain adaptation. In each round of their algorithm, both new training instances from the unlabeled data and new features get added to the existing sets. Kübler and Baucom (2011) use another variant of co-training for POS-tagging. They train three different taggers in the source domain and add all sentences from the target domain to the training set on which all three taggers agree. They report slight but statistically significant increases in accuracy for POS-tagging dialogue data.

Instance weighting. The general idea of instance weighting is to account for the differences between the distributions $P_S(X, Y)$ and $P_T(X, Y)$ by using instance-specific weights during training. Instance weighting techniques factor the joint distribution of X and Y either into $P(X, Y) = P(X \mid Y)P(Y)$ or $P(X, Y) = P(Y \mid X)P(X)$. They also assume the conditional probabilities to be the same in both domains, $P_S(X \mid Y) = P_T(X \mid Y)$ and $P_S(Y \mid X) = P_T(Y \mid X)$. This gives us two possibilities for explaining the differences between $P_S(X, Y)$ and $P_T(X, Y)$. Previous work on instance weighting takes two main perspectives: either $P(X)$ or $P(Y)$ is assumed to vary across domains.

Covariate shift assumes that problems are governed by differing observation distributions, e.g. $P_S(X)$ and $P_T(X)$. Jiang and Zhai (2007) propose a framework that integrates prior knowledge from different data sets into the learning objective by weights. By including knowledge from $\mathcal{D}_T^u$, they were able to increase accuracy on their POS data set by 1.2%. Bickel et al. (2007) state a new learning objective function that is able to determine weights for each training example directly. Unfortunately, there are no results available for POS tagging with this approach.

The second scenario is the one in which class labels have different prior probabilities, e.g. $P_S(Y)$ deviates from $P_T(Y)$. This problem can, for example, arise in POS-tagging when certain tags hardly occur in the source domain, such as proper nouns. Chan and Ng (2005) examine two approaches to this problem for the task of word sense disambiguation. Similar to the concept of bootstrapping, they use the posterior probability $P_S(Y \mid X)$ in an intermediate step to find better estimates of the prior probabilities $P_T(Y)$. One limitation of their approach is, though, that their method is strongly coupled with the underlying learning algorithm.

2.5 Singular Value Decomposition

When working with feature representations, one often has to deal with data of high dimensionality. Besides computational issues arising from holding data in memory and increased algorithm runtimes, high dimensional data is very likely to contain redundancies. It is thus desirable to have a procedure which is able to automatically find an appropriate representation in a lower-dimensional space. There is a whole zoo of dimension reduction techniques; each of which makes different assumptions about the true nature of the data.

One prominent technique is singular value decomposition (SVD). Before we discuss the mathematical details of it, we give a short overview of the algorithm and introduce some technical terms. Given a real matrix $m \times n$ M, SVD factors M into three different matrices U, Σ and V^T:

$$M = U\Sigma V^T \tag{2.18}$$

where U is an $m \times m$ orthonormal matrix, S an $n \times n$ diagonal matrix with only non-negative entries and V^T an $n \times n$ orthonormal matrix. The columns of U are the eigenvectors of MM^T, while the columns of V^T are the eigenvectors of M^TM. Respectively, they represent left and right singular vectors of M. It should also be noted that by convention, the diagonal entries s_i of Σ are sorted such that $s_1 \geq s_2 \geq \ldots \geq s_n$. So far, the matrix M has just been decomposed into three other matrices. There has been no approximation involved, as their product is equal to the original matrix. However, we can now find an approximation of M by taking the first d singular values of Σ, where $d << r$, and discard the rest, which gives us the equivalent of only keeping the d largest singular values. This means Σ will be replaced by Σ_d with $s_i = 0$ for $i > d$. The matrix M will thus be approximated by

$$M \approx M_d = U\Sigma_d V^T. \tag{2.19}$$

The low-rank approximation M_d to M is in fact the best solution when minimizing

$$\left\| M - \tilde{M} \right\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{T} |M_{ij} - \tilde{M}_{ij}|^2}$$

under the constraint that $\mathrm{rank}(\tilde{M}) = d$. This fact is also known as the *Eckart-Young theorem*, proven in 1936.

A typical application of SVD is in machine learning where it is often used to perform Principal Component Analysis, a related dimensionality reduction technique that we discuss in the next section. Another popular application of SVD is in the field of information retrieval where it is used to fight the problems of synonymy and semantic relatedness. After creating a so-called term-document matrix, a low-rank approximation of this matrix is computed via SVD. This method is also known as Latent Semantic Analysis (Deerwester et al., 1990). The working hypothesis there is that if words often occur together in a document, they will also be likely to be semantically related to each other. When reducing the number of dimensions, information is preserved best if related words are consolidated rather than unrelated words get merged. One hopes that the resulting dimension-reduced matrix will contain the most relevant information possible in its compact state, but will eliminate noise.

When using SVD as a dimensionality reduction technique, one should keep in mind the limitations it has. First, SVD is an unsupervised algorithm and relies on appropriate scaling of the feature values. If a feature has a lot of variance in the original feature space, SVD will try to best approximate its variance in the reduced space as well. Moreover, some of the variance in the data could also stem from noise. When the singular values capturing the noise are large enough, noise will also be present in the reduced space (Shlens, 2005). Hence, we cannot rely on SVD as a proper method for noise elimination. Lastly, SVD is only able to model linear dependencies within the data and suffers when

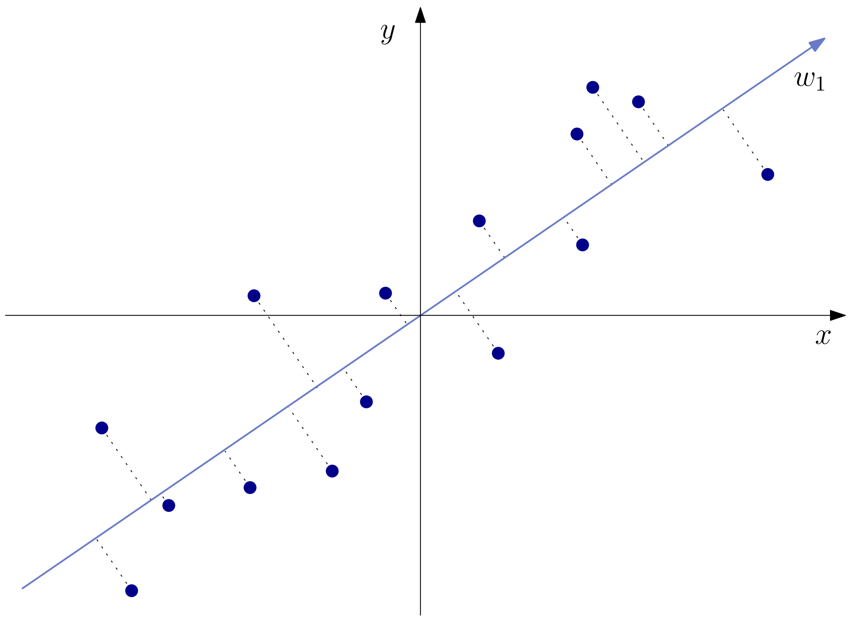

Figure 2.3: PCA applied to a set of 2D points. When reducing the number of dimensions to one, points get projected onto the first principal component w_1. The vector w_1 points in the direction of largest variance or equivalently, lies in the direction that minimizes the mean squared projection error indicated by the dotted lines.

there are non-linear structures. However, various generalizations exist that are able to overcome the shortcomings of SVD (Gorban et al., 2008).

An issue which we haven't addressed yet is choosing a good value for d. If d is too large, we still capture a lot of the subtleties in the data. However, we are striving for a representation of the data with less redundancy and therefore wish to be able to generalize from the data. If d is too small, we risk missing important concepts of the data. It can be shown that the size of a singular value is proportional to the amount of variance in direction of the left singular vector. The approximation error we make is also proportional to the sum of the omitted singular values s_i for $i > d$ (Everitt and Hothorn, 2011).

This gives us one of several heuristics for determining an appropriate value for d. We can simply specify the amount of variance to be preserved in the data. We then choose the set of singular values whose elements sum up to the desired variance. A second guideline involves plotting the singular values and looking for a knee in the plot as proposed by Cattell (1966). One can then keep all singular values before the knee. The rationale behind this is that one expects that the d largest singular values represent the most important concepts in the data; smaller values are supposed to be noise. As argued above, knowing how much variance of the original data is retained in the subspace may help determine the number of dimensions which should be kept. However, variance does not necessarily have to be the kind of information needed for a specific task. Thus, the number of dimensions is also often set to a fixed number or is determined d on a validation set.

2.5.1 Principal Component Analysis

Another dimensionality reduction technique is principal component analysis (PCA). It is closely related to SVD as we will show below. Indeed, because of the intimate relationship between the two techniques, we can use PCA to provide some intuition for what SVD does when being applied to set of data points. PCA can be motivated using two equivalent objectives (Hastie et al., 2008). First, PCA tries to find orthogonal directions in the feature space with largest variance, called principal components. Second, PCA will compute the linear subspace which yields the minimum squared projection error if the number of dimensions is reduced. If one keeps the full number of dimensions, PCA will just correspond to a rotation of the original vector basis so as to fulfill the first objective. The example in Figure 2.3 shows how PCA performs a reduction from two dimensions to one.

PCA takes a data matrix X as input, where each of the n columns corresponds to the feature vector of a sample. X is assumed to have zero empirical mean. This can be ensured by subtracting the mean from all data points in a preprocessing step. It then determines a new basis for the data by computing the eigenvector matrix W of the covariance matrix $\frac{1}{n}XX^T$. This allows us to factor the covariance matrix as:

$$\frac{1}{n}XX^T = WDW^T. \tag{2.20}$$

Let's define $Y = \frac{1}{\sqrt{n}}X$ for convenience. As any real matrix, Y also has a singular value decomposition. Thus, we can express the left-hand side of the last equation as follows:

$$\begin{aligned}
\frac{1}{n}XX^T &= \frac{1}{\sqrt{n}}X\frac{1}{\sqrt{n}}X^T && (2.21)\\
&= YY^T && (2.22)\\
&= U\Sigma V^T\left(U\Sigma V^T\right)^T && (2.23)\\
&= U\Sigma V^T V\Sigma U^T && (2.24)\\
&= U\Sigma^2 U^T && (2.25)
\end{aligned}$$

The last line uses the fact that V is orthonormal. Now it's easy to see the intimate relationship between PCA and SVD. D corresponds to Σ^2 and W corresponds to U. Hence, we could just perform an SVD on $Y = \frac{1}{\sqrt{n}}X$ instead of computing and decomposing the whole covariance matrix. Indeed, SVD is often used in practice to perform PCA. Apart from being easier, there are also numerical reasons for why SVD is preferable to directly computing PCA.

Sometimes, the terms SVD and PCA are used interchangeably. Part of this confusion comes from the different uses of the term SVD. Effectively, SVD is just a matrix factorization method, but sometimes it is also used to denote the low-rank matrix approximation induced by SVD. If we talk about SVD as a dimensionality reduction technique applied to a data matrix, then the only real difference is that PCA works on the centered data matrix, e.g. a matrix with zero mean.

Chapter 3

Case Study: Bio

The goal of this chapter is to motivate a couple of the problems that can occur in domain adaptation with part-of-speech tagging. Although we examine just one target domain in this case study, namely BIO, many of the problems we find here are typical for domain adaptation and also occur in other scenarios.

3.1 Experiment

To see how well a conventional approach performs, we trained a discriminative HMM-based part-of-speech tagger (Schmid and Laws, 2008) on the WSJ data set $\mathcal{D}_S^l$. Table 3.1 shows the tagging performance on BIO-dev. The overall accuracy on BIO-dev with 89.1% is fairly low compared to the results for single-domain tasks, where most state-of-the-art taggers achieve around 96-97% on WSJ data.

However, one can also see from the table that there is a huge difference between known words and unknown words. The accuracy on known words is near state-of-the-art. Tagging unknown words seems to be a more challenging task; accuracies on them are about 28% lower than on known words. Also, most of the errors that the tagger made are due to unknown words, namely 62.3%. This suggests that unknown word handling is a crucial component in building robust taggers. If we improve tagging on unknown words, we should be able to improve overall accuracies as well.

3.1.1 Common Errors

As our basic tagger wasn't too successful, we wanted to see where most of the errors occur in the target domain. Table 3.1.1 lists the most frequent errors. Note that there are some unseen tags we will never be able to get right, for instance HYPH, # and AFX. They have been added to the Penn Treebank tagset in order to account for the special needs of the bio domain.

	unknown words	known words	all
Errors	870	527	1397
	(62.3%)	(37.7%)	(100%)
Accuracy	66.5%	94.7%	89.1%

Table 3.1: Performance of a basic POS tagger on BIO-dev

tags		errors	
correct	predicted	absolute	relative
NN	NNP	334	2.59%
NN	JJ	249	1.93%
HYPH	:	102	0.79%
NN	NNS	83	0.64%
#	:	68	0.53%
VBN	JJ	40	0.31%
JJ	NN	46	0.36%
SYM	various	78	0.60%
various	SYM	1	0.01%

Table 3.2: Commonly made errors on BIO-dev using a basic HMM-based tagger. The percentages shown in the right-most column are relative to the total number of tokens.

We can also see that most errors are due to nouns being falsely tagged as proper nouns. The opposite rarely happened (although not listed here). One reason for that might be that WSJ contains a lot of proper nouns, so the transition probabilities might be biased towards the recognition of proper nouns. We further analyze this issue in Section 3.2.3. Next are nouns that get misclassified as adjectives. This might be due to their relative similar contexts in the BIO domain. Distinguishing normal nouns from plural nouns seems hard as well. Again, a reason for that might be that they occur in very similar contexts. If context doesn't suffice to make a correct prediction for certain words, we might want to incorporate other sources of information. We discuss several possible feature sets in Chapter 5.

3.2 Inter-Domain Differences

As mentioned in Section 2.4, we can view domain adaptation as the problem of having data from two different distributions $P_S(X,Y)$ and $P_T(X,Y)$. We present three phenomena in this section that can contribute to such differences. First are symbols as an example for tags which have different uses in both domains. From a more formal perspective, this can cause $P_S(Y \mid X)$ and $P_T(Y \mid X)$ to deviate. We then look at how different tag distributions are in both domains, meaning that we compare $P_S(Y)$ and $P_T(Y)$. Finally, we look at the transition probabilities of tags in both corpora and discuss a couple of critical cases.

3.2.1 Symbols

Symbols are normally easy to classify, as they often consist of non-alphanumeric characters. Yet, they seem to be notoriously hard to get right as only 23% (22 out of 96) of them are assigned the right label in BIO-dev. The main problem seems to be the different uses of the SYM tag in both domains. In BIO-dev, 96 words (0.74%) have a SYM tag, whereas in the WSJ training corpus, SYM occurred only 55 times (0.01%). Examining the contexts in which SYM occurs in both domains provides further insight. Most often

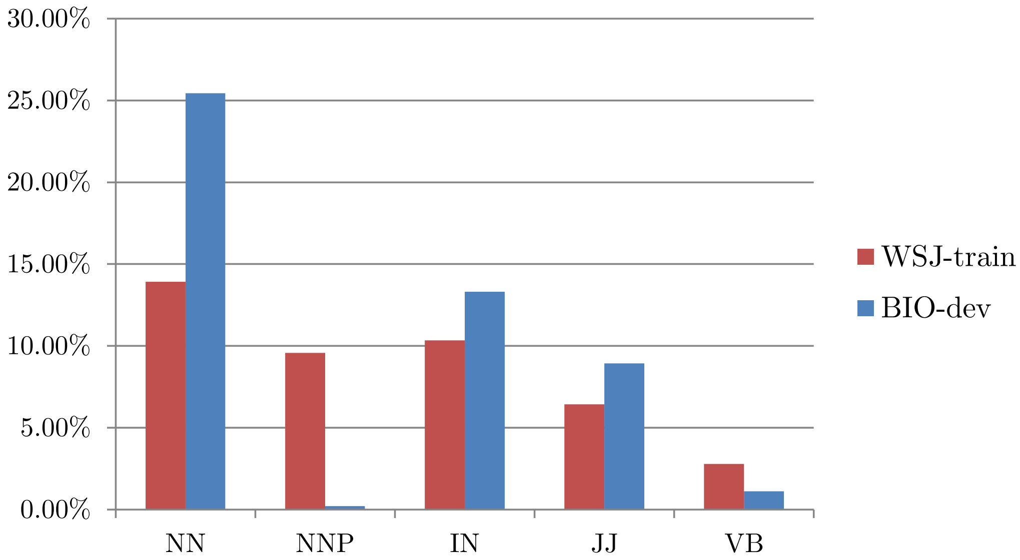

Figure 3.1: Tag distribution in the source and target domain. Only the top five frequencies with the greatest discrepancies are shown.

(33 out of 55 times), SYM is used in WSJ-train to denote an enumeration symbol at the beginning of a sentence. Next with 14 occurrences are symbols that indicate footnotes.

In BIO-dev, the most frequent symbol is "/", occurring 36 times. It is only used between two nouns, mostly between two technical terms or between two numbers. Examples for the two different uses are "SH3/SH2 activator" and "5/6". The second most frequent symbols are "<" and "=" which are used when specifying a p-value. The equals sign also occurs in WSJ, but in a completely different context. It is used to add the translation to a foreign word, for instance in "Homerun=jonron".

To summarize, symbols have greatly varying contexts in both domains as they often serve special purposes. The key to a good performance in the target domain is to find features that are able to make a connection between the two domains. For symbols, however, it seems almost impossible because they hardly occur in the source domain.

3.2.2 Tag Distribution

Remember that one of the two scenarios of instance weighting in Section 2.4 addressed the problem of having different class priors in the source and target domain. In part-of-speech tagging, this corresponds to having different prior probabilities for the POS tags in both domains. Differing prior probabilities for a tag can be problematic because a tagger is biased for the distribution observed during training. Hence, we need strong features during testing to overcome these biases.

Figure 3.1 shows how differently some tags occur in WSJ and BIO. One striking difference is the one between nouns and proper nouns. Normal nouns are roughly twice as frequent in BIO-dev as in WSJ-train. Moreover, proper nouns seem to play almost no role in BIO, whereas they are quite common in the WSJ domain. These differences can shed some light on why nouns were mostly confused with proper nouns in Table 3.1.1. As there are almost no proper nouns in BIO and nouns and proper nouns occur in relatively similar contexts, chances are high nouns get tagged as proper nouns.

There seem to be multiple reasons for the large discrepancies between nouns and proper nouns in both domains. One simple reason is that in the WSJ texts, proper nouns are often linked to company names or people's names. BIO texts mostly cover results from medical studies and contain fewer proper names. Another explanation are different annotation guidelines. In WSJ, for example, "DNA" is mostly annotated as NNP, whereas in BIO, is labeled as NN. If annotation guidelines differ, domain adaptation becomes an almost intractable task. However, we show later that we can still improve results on BIO by incorporating other information. Hence, we believe that different annotation guidelines, though present, do not dominate here.

There are other differences in tag frequencies for IN and VB. However, they don't seem to cause to many errors as they don't show up in Table 3.1.1. Adjectives and prepositions seem more frequent in BIO due to their use in technical descriptions, for example in "endometrial carcinoma", "patients with" or "inactivation of".

3.2.3 Transition Probabilities

Traditional stochastic taggers usually rely on two major sources of information: context information in the form of transition probabilities and lexical information in the form of emission probabilities. While information pertaining to tag transitions often helps in cases where source and target domain do not differ, one can doubt whether the same information is also useful in an unsupervised domain adaptation setting.

In domain adaptation, transition probabilities in the target domain may differ significantly from the source domain. In some scenarios, this could degrade tagging performance. For example, certain transitions that are unlikely in the source domain are also assumed to be unlikely in the target domain. This could confuse the tagger while trying to find a matching sequence of tags for the sentence.

Diverging transition probabilities are especially critical with unknown words or unknown tags. When the tagger encounters an unknown word, it has only little or no lexical information for this word. Thus, the tagger is forced to rely on the transition probabilities estimated in the source domain. If this information is erroneous, it is very likely the tagger will make a wrong prediction as well. A somewhat related problem pose unknown tags. With unknown tags in the target domain, the tagger is forced to choose between existing tags. There might not be a tag corresponding to the new tag. Most likely, tag probabilities will be different for the tag chosen and the unknown tag. Moreover, if the tagger gets a tag wrong, any incorrect tag might influence other tags in the sentence as well. The hypothesis here is that performance will be degraded by incorrect information from neighboring tags just as it benefits from correct tags of neighboring words.

In order to investigate these issues further, we computed transition probabilities for all tags in WSJ-train as well as in BIO-dev. We found in Section 3.1.1 that the tagger mostly confused nouns with proper nouns. This is why we chose transition probabilities of the latter for a closer look. Table 3.3 shows the transition probabilities for nouns and proper nouns which differ significantly in both domains.

There are two critical scenarios for transition probabilities we can infer from the Table. The first scenario is when tag transitions are typical in one domain but not in the other, like NN after a SYM tag or NN subsequent to #. Note that the # tag is a special tag only used in the bio domain. We can see that nouns are roughly twice as likely to occur after gerunds or superlative adjectives in BIO-dev as they are in WSJ-train. The problem is that the lower the transition probability is, the higher the emission probability must

tags		$P(t_i \mid t_{i-1})$	
t_i	t_{i-1}	in WSJ-train	in BIO-dev
NN	JJS	0.34	0.18
NN	WP$	0.38	0.00
NN	SYM	0.07	0.39
NN	#	0.00	0.35
NN	VBG	0.13	0.30
NN	FW	0.08	0.29
NN	IN	0.11	0.32
NN	(	0.06	0.62
NN	CC	0.12	0.40
NNP	NNPS	0.29	0.00
NNP	.	0.18	0.00
NNP	NNP	0.38	0.19
NNP	FW	0.21	0.00
NNP	IN	0.15	0.00
NNP	(	0.33	0.01
NNP	CC	0.16	0.01

Table 3.3: Transition probabilities for selected tags in WSJ-train and BIO-dev. Only tags whose probabilities differ by more than 0.1 in both corpora are shown. The transition probability for a pair of tags was estimated by the count of bigram(t_{i-1}, t_i) in the respective corpus divided by the frequency of t_{i-1}.

be in order to still be able to make a correct prediction.

The second problematic scenario appears when transition probabilities are inversely correlated to each other in both domains. This can be particularly well seen with the FW, IN, (and CC tags. A noun is more likely in the bio domain after these tags than a proper noun, whereas in WSJ everything is flipped. In other words, a model based on the transition probabilities of WSJ would be biased towards predicting a proper noun although one would expect a noun in the bio domain. One striking instance of a tag with reversed transition probabilities is (. It seems to be a reliable indicator for a noun in BIO-dev with a probability of 0.62 whereas it rarely occurs in the context of a noun in WSJ.

3.3 Summary

We saw in this case study that unknown words are responsible for the majority of errors in the target domain. Accuracies on unknown words are low accordingly. Also, as was mentioned before in Section 1.1.2, target domains contain relatively high percentages of unknown words. Hence, focusing on unknown words seems a very promising approach in domain adaptation for POS-tagging. We also saw that some errors are due to different annotation guidelines or different uses of tags. Unfortunately, there is nothing we can do about this in an unsupervised setting.

Moreover, we found that transition probabilities are also different for the two domains at hand. We reasoned that transition probabilities might even hurt performance in some cases. This was part of the reason why we will use a word-based approach as our standard set-up for all experiments in this work. Instead of tagging whole sentences, we tag each word separately, independent from its context. Apart from being easier, this approach will allow us to observe which information helps most in isolation.

Another challenge in domain adaptation is different prior probabilities for tags. The findings of this chapter give rise to the idea of using some form of instance weighting during training. In the extreme case, instance weighting could set weights for some training examples to zero, meaning that we leave out certain examples during training. We examine a step we call (training set) *filtering* based on this idea in Chapter 6. Another idea might be to restrict the set of possible tags for unknown words, since unknown words are mostly from open classes like proper nouns, adjectives or adverbs.

Chapter 4

Exploring Distributional Features

In this chapter, we investigate whether distributional features are effective representations for domain adaptation in POS tagging. Our goal is to find features that represent properties which are meaningful across multiple domains. These feature are then used as input to a supervised tagging algorithm. We hope that eventually, these features will help remedy problems that are common to domain adaptation, such as unknown words. Recall that for the methodologies in this work, we assume that we have labeled data in the source domain and are able to gather large amounts of unlabeled data in both the source and target domain, a setting that is sometimes referred to as unsupervised domain adaptation (Daumé III, 2007).

One of the key ideas in the design of distributional features is to characterize words by the contexts they appear in. In doing so, we need nothing but frequency information to build our features. This information can be extracted from a corpus in an unsupervised fashion, e.g. we do not need to provide any annotated data to the algorithm. We hypothesize that words which occur in similar contexts will also exhibit commonalities on a higher level, i.e. on a grammatical level. This should enable a tagger to relate words from both domains to each other, even though vocabularies or word usages might differ.

Our experimental setup is motivated by two observations: (i) texts from other domains usually contain high percentages of unknown words and (ii) transition probabilities might not be a reliable source of information. Thus, we adopt a word-based approach that classifies each word independent of the sentence it is part of. Also, we report results mainly on unknown words as they can benefit the most from external information. As for the experiments in Chapter 5, we only examine the combination of WSJ as the target and BIO as the source domain here. This is because a lot of previous work has addressed this particular setting; moreover, we only want to be able to quickly evaluate our experiments.

After describing the construction process, we introduce an experiment in order to evaluate the usefulness of our design decisions in a simple setting. We study several extensions and alternatives to this basic experiment in Sections 4.5 and 4.6. We look at possible ways of leveraging information about the differences between domains in Section 4.7. Furthermore, we attempt to answer the following series of questions in this chapter:

1. Are the feature representations presented here useful for domain adaptation?
2. What is the effect of applying dimensionality reduction techniques such as SVD to our features?

3. Where are the limitations of the approaches taken?
4. What are interesting parameters to be included in future experiments?

4.1 Constructing Feature Vectors

There are many kinds of contextual properties one could leverage for building feature vectors. The first set of features described here, distribution features, looks at the tokens directly to the left and right of a word. The second set of features, cluster features, is more complex as its construction is divided into two steps. First, a cluster is constructed for each token. The final cluster feature for a word then describes how many of the member words of a cluster occur next to the word.

We have already talked about different types of context information, but not about which specific contexts we would like to consider. If we would take every word appearing in the context of a word w into account, our feature vectors would be highly dimensioned. Moreover, it is questionable whether this information would help or if it would only add more noise to our feature vectors. Thus, we decided to use a restricted set of words to characterize w, which we call *indicator words*. The co-occurrence of these indicator words alongside a word w is used to represent w. For now, we shall assume that we are given a set $T = \{t_1, t_2, \ldots, t_n\}$ of indicator words. We leave the full details on how to choose those words for Section 4.2. If not specified differently, we use $n = 250$ indicator words.

A feature vector for a word w is the concatenation of a left and a right vector, $f_{\text{left}}(w)$ and $f_{\text{right}}(w)$ respectively. More formally, if $f(w)$ denotes the function that maps a word w to its feature vector, then

$$f(w) = \begin{bmatrix} f_{\text{left}}(w) \\ f_{\text{right}}(w) \end{bmatrix}$$

Both, the right and the left vector of a word, contains n entries, one for each indicator word:

$$\begin{aligned} f_{\text{left}}(w) &= [x_1(w)\ x_2(w)\ \ldots\ x_n(w)]^T \\ f_{\text{right}}(w) &= [x'_1(w)\ x'_2(w)\ \ldots\ x'_n(w)]^T . \end{aligned}$$

4.1.1 Distribution Features

This idea has been adopted from Schütze (1995). There, a word is characterized by the tokens that occur in its vicinity. Entry i in the left (right) distribution vector of word w is the number of times that the indicator word t_i occurred immediately to the left (right) of w:

$$x_i(w) = 1 + \log(\text{freq}(\text{bigram}(t_i, w)))$$

The right vectors are constructed analogously. The log-scaling of the frequencies will give lower weight to frequent terms. Similar to term weighting in information retrieval, this step is supposed to prevent frequent terms from dominating the feature vectors. As a final step, both the left and right vectors are length-normalized in order to have them weighted equally in the final feature vector.

	left cluster	right cluster
the	NN 0.16 VBN 0.10 VB 0.15 VBZ 0.08 VBG 0.16 VBD 0.08 NNS 0.08	JJ 0.21 NN 0.33 NNS 0.14 NNP 0.17
says	NN 0.18 NNP 0.72	JJ 0.06 NN 0.11 NNS 0.06 NNP 0.65
billion	CD 0.99	JJ 0.16 NN 0.38 IN 0.07 NNS 0.08 NNP 0.07
cells	JJ 0.26 NN 0.27 VBN 0.09 VBG 0.06 CD 0.10	JJ 0.08 RB 0.12 NN 0.08 VB 0.09 VBN 0.18 VBZ 0.08 IN 0.06 VBG 0.08 VBD 0.10

Table 4.1: Tag distribution for words within the left or right cluster of a word. The number behind each tag is the relative frequency of words with this tag in the corresponding cluster. All tags with frequencies below 0.05 were omitted.

4.1.2 Cluster Features

Another idea for designing features that are suited for domain adaptation is characterizing words by cluster memberships. Given an indicator word t_i, we define two clusters for t_i which encompass all words that occur to the right (left) of it. More precisely,

$$C_{\text{right},i} = \left\{ v \mid \text{bigram}(t_i, v) \in \mathcal{D}^{u,l}_{S,T} \right\}.$$

The features for a word w are then the scaled sum of the frequencies of words in the corresponding cluster occurring to the left (right) of w:

$$x_i(w) = \log \left(\sum_{c \in C_{\text{right},i}} \text{freq}(\text{bigram}(c, w)) \right) + 1.$$

The same idea can be used to construct the left feature vectors. Again, the left and right vectors are normalized to unit length. To get a better idea why these features might be useful, we looked at the tag distribution of all known words in the clusters. We used the macro-average over tags from both WSJ-train and BIO-dev to compute the distributions. Table 4.1.2 shows the distributions of some sample clusters.

Let's take an example that further motivates our cluster features. Suppose we would like to have a feature that says "my left neighbors are only adjectives and nouns". Unfortunately, this information isn't available to us in an unsupervised task. However, we can find clusters that will most likely carry this information. Looking at the tags of words in the right cluster of "the", one can see that "the" is mostly followed by adjectives or nouns. Hence, if those words occur frequently to the left of a word w, we can tell that its left neighbors are mostly adjectives or nouns.

Table 4.1.2 contains more illustrative examples. Left neighbors of "says" are typically nouns or proper nouns. "Billion" is almost only preceded by numbers. Although clusters often represent syntactical properties, there are some cases where their membersets are very diverse and may carry less useful information. One example for this is the right cluster of "cells" where tags are more or less distributed equally.

4.1.3 Singular Value Decomposition

We utilize singular value decomposition to reduce feature space dimensionality and to eliminate possibly irrelevant information. The number of dimensions balances the amount of information to be retained versus the exactness of the approximation. We set the number of dimensions d to be equal to 100 for the following experiments, although the value of d may change results. However, choosing a good value for d appears to be less critical for the final supervised task in Chapter 6 as the algorithm can adjust the weights for each dimension as needed.

We apply SVD to our token-feature matrix M which is constructed by concatenating all non-zero feature vectors of all words in the combined corpus vertically:

$$M = \begin{bmatrix} f(w_1)^T \\ f(w_2)^T \\ \dots \\ f(w_m)^T \end{bmatrix}$$

We perform an SVD on M and keep the 100 largest singular values. The matrix M will thus be approximated by

$$M \approx M_{100} = U\Sigma_{100}V^T.$$

The SVD defines a mapping from the original feature space onto a reduced space $\mathbb{R}^d$. We can map new words using a step called fold-in to this lower dimensional space:

$$f_{\text{reduced}}(w) = f(w)^T V \Sigma_{100}^{-1}.$$

The reduced vector is also length-normalized. Since it would be costly to do this mapping at run time, we precompute the reduced feature vectors for all tokens and store them in a file.

4.2 Indicator Words

As already mentioned in Section 4.1, the main purpose of indicator words is to restrict the set of contexts we take into account. Selecting a different set of indicator words will automatically yield a different feature representation for a word. We present an experiment later in this chapter that tests different selection methods for their effectiveness.

There are two basic considerations to make when choosing a set of indicator words. First, we want an indicator word to occur frequently in the corpus to allow for reliable estimation. However, if an indicator word is too frequent it might not provide enough discriminative information. The methods presented here neglect this issue, as they assume that the final features will be weighted appropriately. The second consideration is specific to the task of domain adaptation. We aim to learn a tagging model on the source domain that is also useful for tagging in the target domain. On the one hand, we would like indicator words to be frequent in the source domain, as we can only learn weights for features we have seen. On the other hand, we want indicator words to also occur frequently in the target domain. Otherwise, we won't be able to relate words from the target domain to words from the source domain.

The three following methods address this issue in different ways. Naive selection simply draws the most frequent terms from a combined corpus. Intersection selection tries to find tokens that occur both frequently in the source and target domain. Lastly, source selection picks those tokens that are most frequent in the source domain.

4.2.1 Naive Selection

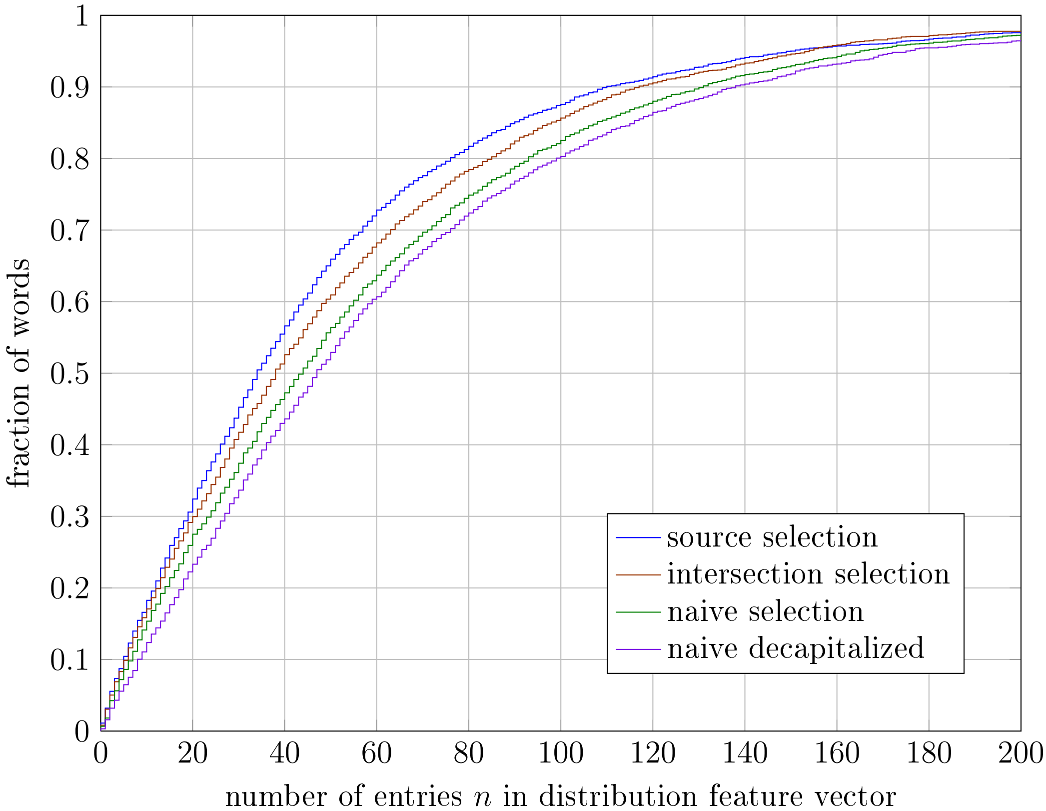

Figure 4.1: Fraction of words in the vocabulary of BIO-dev that have distribution feature vectors with $\leq n$ non-zero components. Each feature vector had 500 dimensions. We compare three different methods for selecting indicator words (naive selection, intersection selection and source selection) and show the effects of ignoring capitalization.

This methods computes the n most frequent terms in $\mathcal{D}_{S,T}^{u,l}$ which is the union of all labeled and unlabeled data from the source and target domain. Indicator word t_i is then the token with frequency rank i in this corpus. One advantage to this method is that on average, our distribution feature vectors using these indicator words will be denser in the target domain than with other methods as Figure 4.1 shows. The reason for this is rather simple. The indicator words chosen here are the most frequent terms in the corpus and we are very likely to encounter them when constructing the final feature vectors. However, it could be more significant to have dense vectors for words in a specific domain and not to optimize for a global criterion. This motivates the following two selection methods.

4.2.2 Intersection Selection

If we naively choose the n most frequent terms from $\mathcal{D}_{S,T}^{u,l}$, we can run into some problems. First, sizes of the single corpora can be different. This could cause terms of one domain to dominate the set of indicator terms even if their relative frequency in the domain is low. Another issue is that we want to represent words by indicator words that are common in both domains. In the extreme case, indicator words would only occur in one domain. As we try to learn a common set of weights, words from one domain can't be related to words from another domain. Even SVD would not help with independently occurring features as it wouldn't be able to make a connection between them.

To get a better idea of how our set of indicator terms with naive selection looks like, we looked at the frequencies of terms selected in both domains. We hypothesized that good features are features which occur frequently in both domains. We used the following measure to assess how unevenly an indicator word t occurs in both domains:

$$bal(t) = \frac{\text{freq}_{\text{WSJ-train}}(t) - \text{freq}_{\text{BIO-dev}}(t)}{\text{freq}_{\text{WSJ-train}}(t) + \text{freq}_{\text{BIO-dev}}(t)}$$

with $\text{freq}(t)$ being the frequencies of t in the respective corpora. If w only occurs in one corpus, $bal(t)$ will be 1 or -1 respectively. If it has the same frequency in both corpora, $bal(t)$ will be zero. We'll refer to a feature as being unbalanced, if $|bal(t)| \geq 0.85$. For the naive selection method, we found 98 out of the 250 indicator words to be unbalanced. 48 of them had $bal(w) \geq 0.85$. This means that they were hardly present in the bio domain. Similarly, 50 of them hardly occurred in the WSJ domain.

The observations above motivate our new selection method. Rather than selecting the most frequent words from $\mathcal{D}^{u,l}_{S,T}$, we use a separate corpus for each domain. We then retrieve a list of the respective vocabularies sorted by term frequency in descending order. We simultaneously move down these lists and add terms to our feature set if they are in the intersection of both lists up to the current position. This process ends as soon as we have selected n indicator words.

We used the same measure as in the second paragraph to evaluate the new feature set. Instead of having 98 unbalanced indicator words, now only three of the 250 words are unbalanced. We hope that this will help in achieving a better tagging performance. Figure 4.1 shows one potential drawback of this method: Distribution feature vectors now got sparser, e.g. contain less entries. However, the number of entries doesn't necessarily have to correlate with the amount of meaningful information that a vector carries.

4.2.3 Source Selection

Given a word from the target domain, we try to find words in the source domain which behave similarly in respect to their part-of-speech labels. One could argue that only tokens from the source domain are important, since our supervised learning method will estimate the weights for each feature based on them. This leads to our third feature selection method where we just select the n most frequent tokens of $\mathcal{D}^{l}_{S}$ as indicator words. However, this yields sparser distribution feature vectors for words in the target domain since the selected features will be less frequent. Indeed, among the three methods, source selection creates the sparsest feature vectors. It is shown as the upper blue line in Figure 4.1.

4.3 Design Choices

So far, we haven't discussed three important issues that come up during feature design. The first one is inconsistent capitalization in the corpus. We found that words in BIO are often capitalized since they are part of abstracts containing a lot of short sentences. Of course, we could lower-case all words in the corpus but this approach would bring other implications along. Section 4.3.1 examines this issue more thoroughly and looks at two possible solutions.

Another problem is words which do not occur in the unlabeled data set. Section 4.3.2 presents two approaches to handling such cases. Lastly, we have to decide which data to use when constructing feature vectors. Section 4.3.3 discusses two options with their advantages and disadvantages.

4.3.1 Capitalization

A problem occurs when one is not able to get enough information for the capitalized version of a word, even though its uncapitalized version with the same part-of-speech tag appears frequently. Capitalized and uncapitalized versions of the same word can appear for various reasons. They occur naturally at the beginning of sentences when every word is capitalized. There might also be other instances due to typing errors or different spelling preferences. The last group consists of words which are part of a named entity and are therefore capitalized. If capitalization is ignored, we will not be able to account for the differences between capitalized words and their uncapitalized counterparts (compare "to excel" vs. "Microsoft Excel" or "united" vs. "The United States of America").

In order to investigate this issue further, we took a look at the corpus. From the 147,607 words in the vocabulary of our corpus, 13,429 occur both capitalized and uncapitalized. We found 12,793 ($\sim$ 8.4%) words which have distribution feature vectors with length zero. When we lower-cased all words in the corpus, our vocabulary size dropped to 129,739. We also had a lower percentage of zero vectors, namely 6.6%.

For the task of domain adaptation, it seems reasonable to strive for dense feature vectors in the target domain. We hypothesize that a dense vector will also contain more useful information. This is why we looked at the sparseness of feature vectors in the target domain at hand. Figure 4.1 shows the cumulative distribution function for all words in BIO-dev. When we ignored capitalization, feature vectors had more non-zero entries. The average number of entries per feature vector increased from 59.3 to 64.1.

To conclude, we found that ignoring capitalization may remedy the problem of sparseness. However, we will lose the ability to discriminate words from named entities or abbreviations. Another possibility for handling capitalization could be to use the feature vector of the uncapitalized version of a word whenever it occurs more often than the original word. We will reexamine this issue in Section 4.5.

4.3.2 Unknown Word Handling

We refer to an unknown word here as a word which occurs during testing and which we haven't built a feature vector for. One approach is to also include the testing set in the corpus which is used to construct the feature vectors. In doing so, we'll have a feature vector for every word in the testing data. Another possible approach for handling unknown words is to compute the feature vector on the testing set that the unknown word is in. Although this solution is a bit cleaner as it doesn't make use of the testing data in the source domain, we chose the first approach since we wanted to keep things simple at this stage.

4.3.3 Representation Data

Another question that comes up during feature design is which data a word's representation should be based on: on a combination of source and target domain $\mathcal{D}^{u,l}_{S,T}$ or only on the target domain $\mathcal{D}_T^{u,l}$. We call this data the *representation data*. Note that representation data is only used to create distribution vectors for each word in a preprocessing step. After this step, the tagging algorithm doesn't need to know about the underlying structure of these feature vectors; distributional features are then treated like any other input features.

The first option is to use data from both the target and source domain $\mathcal{D}_{S,T}^{u,l}$ for building distribution vectors. One obvious advantage is that we make use of all data we have and we can collect sufficient information for words in the source domain. However, our unlabeled data sets $\mathcal{D}_S^u$ and $\mathcal{D}_T^u$ might have varying sizes. This raises the question about how to weight distributional information from both data sets as one data set might dominate. Even if both data sets have the same size we might want to weight training instances from both domains differently.

Building feature vectors based on distributional information only from the target domain $\mathcal{D}_T^{u,l}$ does not suffer from this problem. Moreover, leaving out data from the source domain has another possibly beneficial effect. Words in the source domain that do not occur in the target domain will not have feature representation under $\mathcal{D}_T^{u,l}$ and are thus excluded from the labeled training set. Again, this is some subtle form of filtering or instance weighting where we skip all examples which do not occur in the target domain. As a consequence, however, we will have fewer training examples which is usually observed to degrade accuracies with supervised methods.

4.4 A Sandbox Task

Our experiments' goal is to gauge the usefulness of distributional information for part-of-speech tagging. Instead of running a traditional sequence tagger, we decided to look at the performance on a simpler task. This helps us better understand how our features work and how sensitive results are to different feature sets.

We focus on tagging unknown words as our case study on BIO in Chapter 3 showed that most errors were made on them. The task is to correctly predict a word's majority tag in BIO given the majority tags of all labeled training instances in $\mathcal{D}_S^l$. Our tagging algorithm is a simple k-NN classifier which proceeds in three steps:

1. Find the k most similar words in WSJ-train to the input token using cosine similarity.
2. Assign the tag that occurred most often for a word in WSJ-train to each of the k words.
3. Return the most frequent tag among the k words.

4.4.1 Results

For this experiment, all unknown words from BIO-dev are retrieved. Multiple occurrences of the same word are treated as individual instances. We create distribution feature vectors as described in Section 4.1.1 for all words in $\mathcal{D}_{S,T}^{u,l}$ using $\mathcal{D}_{S,T}^{u,l}$ as representation data. Similarities are computed between feature vectors in the original feature space as well in the 100-dimensional feature space resulting from the SVD. Table 4.4.1 shows the results for the k-NN tagging task. Except for $k = 1$ with intersection indicator

	naive		intersection		source	
	original	svd	original	svd	original	svd
k=1	59.67	51.62	51.96	52.81	59.94	54.43
k=5	58.55	54.35	58.82	53.35	62.10	55.78
k=10	61.36	56.32	54.97	50.96	61.75	55.70
k=25	63.71	56.16	56.47	49.54	60.82	55.12
k=50	**64.33**	59.44	56.90	50.54	60.48	54.89

Table 4.2: Accuracies on unknown words of BIO-dev using a k-NN classifier with distribution feature vectors and different sets of indicator words. Each feature vector had 500 dimensions in the original feature space and 100 dimensions in the reduced space from the SVD.

words, we find SVD vectors to give lower accuracies than unreduced feature vectors. One explanation might be that as with latent semantic indexing (LSI), recall increases at the cost of precision.

We can also observe that as was hypothesized in Section 4.2, there are strong dependencies between the set of indicator words used and the experimental outcome. Despite their name, naive indicator words work best. This might be due to the effect that they create denser feature vectors in the target domain which helps in finding similar words in the WSJ domain. Intersection selection gives the poorest results overall; their best accuracy value is still worse than almost any value for naive indicator words. Second best is source selection. Although the maximum accuracy for it is below the model using naive selection, source selection is more effective than the latter when neighborhood sizes are small.

Our best performing k-NN model employs the tags of the 50 most similar tokens in the original feature space. It achieves an accuracy of 64.33% on unknown words, which is fairly close to the performance of our best supervised model with an accuracy of 66.5%. One thing to keep in mind, though, is that a supervised tagger can adjust the weights for each dimension of the feature vector separately whereas our k-NN tagger weighted all features equally when computing similarities. We therefore hope to be able to further improve results by incorporating some form of feature weighting.

4.4.2 Error Analysis

We investigated the errors of our currently best performing model in order to gain more insight in possible error patterns. In total, 926 of the 2596 unknown words in BIO-dev were mistagged by this model. As before, we considered multiple occurrences of the same word as separate instances. An analysis of the mistagged words of the k-NN model with $k = 50$ and naive selection of indicator words suggested five major categories for errors:

- Tentative errors: These words were correctly tagged by another well performing model. We chose the source selection model with $k = 10$ from above for the comparison. Roughly 10.6% of all unknown words in BIO-dev fall into this category. There are only two possible explanations why the latter model outperformed the currently best model. First, it uses another set of indicator words which might be more appropriate for some words. Second, for different values of k, neighborhood sizes vary and this leads of course to a different classification output. If the errors are due to the first condition, a supervised algorithm could be employed to fine-tune the weight for each feature. Having a different set of indicator words would thus be a special case of weighting: The corresponding weights would be set to zero.

- Unknown tag errors: BIO-dev contains unseen tags due to the special annotation needs in that domain. We also put all words with SYM tags in this category, because the SYM tag is almost absent in the WSJ domain and can thus be regarded as unseen. There are 4.5% of all unknown words that have unseen tags. As we assume that our set of tags is closed, there is nothing we can do about these errors.

- Rare word errors: 6.0% of all unknown words occur less than 10 times in $\mathcal{D}_{S,T}^{u,l}$ which comprises all data that we have. The threshold chosen here is rather flexible than fixed; it was motivated by looking at the feature vectors of some infrequent words. If a word is rare, it is hard to estimate its distribution in a reliable manner. It seems challenging to find a remedy for those errors; however, context might help in some cases if we perform sequence classification instead of word-based classification.

- Sparse feature vector errors: This category comprises all words that occur more than ten times in $\mathcal{D}_{S,T}^{u,l}$, but have less than or equal to nine entries. About 2.2% of all unknown words belong to this category. Again, the problem with those words is to gather enough distributional information. One example for a word in this category is the adjective "single-strand". It occurs 14 times in the corpus and has the following feature vector:

Feature	Value
by{left}	0.7329
,{left}	0.4756
the{left}	0.2809
-{left}	0.2809
used{left}	0.2809

  By looking at the entries, we can tell that its distribution vector contains little information for finding a good neighbor. One potential cure for this situation might be bootstrapping. We could try to find features in the target domain which behave similarly as features in the source domain. This could be done by defining a similarity measure on the clusters of Section 4.1.2 and combine indicator words that define similar clusters into new features.

- Similar context errors: These are all errors, that weren't put into one of the preceding categories. Words in this category make up 12.3% of all unknown words. Obviously, these errors are due to the confusion with words having the wrong tag. Instead of retrieving words that would have the right tag, other similar words with the wrong tag are returned. Thus, either the similarity measure or the features themselves are not able to separate words with different tags properly. This leads to two possible approaches: finding a better similarity measure or more informative features. An improved similarity measure could be found by tuning the weights of each dimension by some supervised algorithm. More informative features could be constructed by using combinations of the existing features. The new features could thus express more complex relationships like "both 'the' and 'of' occurred to the left of a word w". Two other ways of constructing more informative features are discussed in Chapter 5.

4.5 Further Variations

Motivated by the findings of the previous section, we want to examine the effects of our design decisions more closely. We compare six different sets of features. The first one, naive binary, uses the distribution features from section 4.1.1 with a modified scaling. Instead of transforming features with a log function, we just use a binary threshold function that returns one if the feature is greater than zero and zero otherwise.

Second, we are interested in whether more features would help. The naive more features model uses an enhanced set of indicator words. It constructs distribution feature vectors using $n = 500$ indicator words instead of $n = 250$ tokens. Consequently, the final feature vectors of this model have 1000 dimensions.

The third set of features uses distribution features and cluster features. Both sets of features have the same set of indicator words which is retrieved via the source selection method. As both feature vectors contain 500 entries, the final feature vectors have 1000 entries.

The fourth and fifth set of features both use cluster features and indicator words which have been selected by the naive method. The fourth set of features employs distribution features with the default scaling, whereas the fifth feature set has the same transformation to its distribution vectors applied as the first one.

The last set of features is equal to the fifth except for a special handling of capitalized words. If a word is more frequent in its uncapitalized form, we use the feature vector of its uncapitalized counterpart instead.

4.5.1 Results

Table 4.5.1 reports the results for the six feature sets described above. We can see that the distribution features with source indicator words, as well as the distribution features with naive indicator words benefited from adding cluster features. It is also interesting to note that they achieved best results in the reduced feature space of the SVD. This is different to the other feature sets, where the dimension reduction didn't improve accuracy.

Binary distribution features yielded the best performance gain. Our best performing model used those along with cluster features. Its accuracy of 69.30% on unknown words is better than anything we were able to get with a supervised tagger. This is remarkable because our k-NN tagger didn't make use of the context information of the word to tag. Having more features didn't help which might be because of additional features introducing noise, rather than meaningful information. The results also prove that our special treatment of capitalized words paid off. We were able to increase accuracy even further by almost 1.7%. Part of this success, though, might be because unknown words became known ones as soon as they were uncapitalized.

The results also show great effects of the feature scaling method on the results. To further illustrate how feature scaling affects results, here are the 10 most similar tokens in WSJ-train for the noun "jaundice" for distributional features using tf-scaling and binary scaling:

	naive binary		naive more features		source + cluster	
	original	svd	original	svd	original	svd
k=1	57.32	48.46	57.09	52.04	60.59	59.17
k=5	63.87	55.70	59.09	54.08	63.98	64.52
k=10	66.87	55.32	59.59	58.17	63.79	67.18
k=25	66.72	54.85	63.48	58.98	63.33	65.76
k=50	**67.68**	58.28	61.56	60.79	64.02	67.10
	naive + cluster		**naive bin. + cluster**		**naive bin. + cl. + mod**	
	original	svd	original	svd	original	svd
k=1	61.09	62.40	58.55	53.78	59.90	55.70
k=5	59.98	65.10	65.49	56.82	67.06	58.82
k=10	63.14	66.14	67.84	54.89	69.57	56.59
k=25	65.56	68.30	**69.30**	58.74	**70.99**	60.52
k=50	66.56	**68.49**	68.18	60.94	69.80	62.75

Table 4.3: Accuracies on unknown words of BIO-dev using a k-NN classifier with modified features sets.

```
naive.printMostSimilar('jaundice');
Token             Similarity       Tags in WSJ-train
minivans          0.7716           NNS 1.00
vegetables        0.7614           NNS 1.00
vomiting          0.7467           VBG 0.50 NN 0.50
resins            0.7385           NNS 1.00
printers          0.7380           NNS 1.00
bridges           0.7293           NNS 1.00
composting        0.7261           NN 1.00
insulation        0.7220           NN 1.00
self-destructive  0.7204           JJ 1.00
transmissions     0.7198           NNS 1.00

naiveBinary.printMostSimilar('jaundice');
Token             Similarity       Tags in WSJ-train
insulation        0.7833           NN 1.00
Newsday           0.7462           NNP 1.00
resins            0.7385           NNS 1.00
cheese            0.7035           NN 1.00
bass              0.6963           NN 1.00
rodents           0.6963           NNS 1.00
trunk             0.6963           NN 1.00
executions        0.6838           NNS 1.00
```

	$\mathcal{D}_{S,T}^{u,l}$	$\mathcal{D}_{T}^{u}$	$\mathcal{D}_{T}^{u,l}$
k=1	54.28	53.27	54.08
k=5	**56.74**	**57.51**	58.20
k=10	53.16	56.12	**58.71**
k=25	51.77	55.66	55.74
k=50	50.39	53.47	55.39

Table 4.4: Accuracies on unknown words of BIO-dev using a k-NN classifier with different representation data sets.

```
tilts              0.6742            VBZ 1.00
epilepsy           0.6690            NN 1.00
```

We can make several observations here. Distributional features seem to return words with tags which are close to the correct tag. What prevents "jaundice" from being labeled as a noun with the first feature set are nouns in plural form. By using binary distribution features, words with the correct tag get moved up. Some words newly appear in the ranking, for example "cheese" and "bass". Also, we can't find an adjective among the 10 most similar tokens anymore.

4.5.2 Representation Data

In all of our previous experiments, we have used $\mathcal{D}_{S,T}^{u,l}$ when computing feature vectors. In this section, we investigate what the effects are of using varying representation datasets as basis for our feature vectors. For the results presented in Table 4.4, we computed feature vectors on $\mathcal{D}_{S,T}^{u,l}$, $\mathcal{D}_{T}^{u,l}$ and $\mathcal{D}_{T}^{u}$. Instead of using only $n = 250$ indicator words as before, we opted for using all indicator tokens. This is because we do not want to include differences in our comparison that might be due to a different selection of indicator tokens. As we know from Section 4.4, results are rather sensitive to the method used for selecting indicator words.

Except for $k = 1$, we can see that results get constantly better from the left to the right. Models that use target-only data usually do better than the model that used all data. This is most likely caused by the filtering effect that computing vectors on target-only data has; proper nouns which do not occur in the target domain are filtered out. Thus, the k-NN classifier is biased for nouns which are the most frequent class in BIO. Compared to the model that used all available data from the source domain $\mathcal{D}_{T}^{u,l}$, the model that did not use labeled data from the target domain $\mathcal{D}_{T}^{l}$ is constantly worse. We believe that this is due to words in $\mathcal{D}_{T}^{l}$ which do not occur in $\mathcal{D}_{T}^{u}$. For those words, we get a feature vector that is zero which, in turn, can lead to problems during classification.

Another interesting observation we can make here is that using all indicator tokens leads to significantly worse performance. While our naive model using the top 250 indicator words has an accuracy of 64.33%, the same model with all indicator tokens only reaches 56.74%. Intuitively, a supervised algorithm should not suffer dramatically when adding more features. One explanation is that features are not weighted optimally and we would need to scale features appropriately before applying k-NN. We discuss this issue further in Section 4.8.

	naive		naive binary		naive bin. + cluster	
	original	svd	original	svd	original	svd
k=1	85.69	84.07	85.21	83.43	85.48	84.52
k=5	85.46	84.62	86.53	84.89	86.87	85.13
k=10	86.03	85.01	87.14	84.81	87.35	84.74
k=25	86.50	84.98	87.11	84.72	**87.64**	85.52
k=50	86.63	85.64	87.30	85.41	87.42	85.96

Table 4.5: Accuracies on the full BIO-dev dataset using a k-NN classifier with different feature sets.

4.5.3 Full Dataset

We used three well performing sets of features from above and tested them on the full BIO-dev dataset. This should serve as a baseline against which we can compare our discriminative tagger of Chapter 3. We used the same k-NN classifier as in the previous experiments, but had it only enabled for unknown words. When we encountered a known word, we just assigned it its most frequent tag in WSJ-train.

Table 4.5 presents results for this task. As expected, the best performing model on unknown words is also the best performing one on the full dataset. Its accuracy of 87.64% is relatively close to the 89.1% of our discriminative baseline tagger. Again, the supervised tagger has access to more information as it is trained on structured input data. This gives further evidence that a word-based approach can produce competitive results in some scenarios.

4.6 Tag Vectors

Instead of collecting distributional information in a word-based manner as we have done in our previous experiments, we investigate another possible approach in this section: *tag vectors*. The basic idea is to share distributional information only among training instances with the same tag. During classification, we do not look for similar words but rather for tags with a similar distribution. We hope that this can help in cases where words are ambiguous and usages differ significantly in both domains.

We present two different approaches for designing tag vectors which we call *dense tag vectors* and *sparse tag vectors*. Dense tag vectors collect information across all words in the corpus with the same tag and are therefore expected to contain more entries than word-based distribution vectors. The second approach computes separate tag vectors for each word, leading to sparser vectors as information gets split among the different tags of a word. The two approaches make different homogeneity assumptions on contexts. Dense tag vectors assume that contexts of words with the same tags are homogeneous and can thus be grouped together, while sparse tag vectors assume that contexts vary with both the word and the tag of a word.

Both kinds of tag vectors are used as training examples with a k-NN classifier. The classifier retrieves the word-based distribution vector of the word to be labeled and compares it with tag vectors. Finally, it outputs the tags of the closest tag vectors. Note

tag	tokens classified	freq. in WSJ-train
LS	28.5%	0.01%
JJS	14.9%	0.20%
UH	12.6%	0.01%
FW	9.5%	0.02%
)	8.7%	0.14%

Table 4.6: Tags that were assigned most frequently to unknown words of BIO-dev by a k-NN algorithm using dense tag vectors.

	acc.
k=1	33.82
k=5	40.33
k=10	38.14
k=25	**46.19**
k=50	45.99

Table 4.7: Accuracies on unknown words of BIO-dev using a k-NN classifier with sparse tag vectors.

that since tag vectors can only be trained on labeled data, we are restricted to just use data from the source domain.

4.6.1 Dense Tag Vectors

With dense feature vectors, we simply compute one feature vector for each tag. Each dimension in the left (right) tag vector will be the sum of counts of the form bigram(t_i, tag_i), where t_i is the indicator word with number i. As we only have one vector per tag, the k-NN algorithm is modified to just return the tag of the closest vector, e.g. k = 1.

The results we got with this approach were fairly poor. Accuracies on unknown words of BIO-dev were close to 2.0%. Table 4.6 provides a possible explanation for this phenomenon. The algorithm preferred vectors of infrequent tags in WSJ-train. The problem with rare tags is that their corresponding tag vectors are sparser than the vectors of frequent tags. This is because rare tags don't occur in as many contexts as frequent tags. Thus, if an indicator words occurs in the context of a rare tag, higher weight will be given to it in the final tag vector due to normalization. Word-based feature vectors which have a non-zero value in the same dimension will therefore tend to prefer such sparse vectors.

4.6.2 Sparse Tag Vectors

Sparse tag vectors are basically word-based distribution vectors, but they are computed for each word and tag pair separately. As an example, let's consider the word "like" with tags VB and IN. We construct two separate feature vectors, one for all occurrences of

	same maj. tag	diff. maj. tag
Euclid. distance	0.44 ± 0.11	0.48 ± 0.09
Cosine similarity	0.89 ± 0.05	0.87 ± 0.05

Table 4.8: Mean distances/similarities between distributional feature vectors of words occurring in both domains. The corresponding standard deviations are shown behind the plus-minus signs. The first column only considers words with the same majority tag in both domains. Similarly, the second column computed the mean only over words with different majority tags.

"like" as VB and one for all occurrences as IN. We hope that using these tag vectors will enable the k-NN algorithm to find better matches for ambiguous words.

Table 4.7 contains results for sparse tag vectors. We found this approach to work rather poorly as accuracies were generally below 50%. The main problem with sparse tag vectors is that – as the name implies – data sparseness increases. The labeled data that we have gets split among all tags that a word may have. Although having separate vectors for each word-tag combination seems not appropriate for unknown words, it could help disambiguate known words. We study more approaches for tagging known words in the next section.

4.7 Exploiting Inter-Domain Differences

In this section, we investigate the idea of exploiting differences between the source and target domain. For each word that occurs in both domains, we can get two separate feature vectors for it by using either the $\mathcal{D}_S^{u,l}$ or $\mathcal{D}_T^{u,l}$ as representation data. The hypothesis we examine here is that if the majority tags of a word deviate in both domains, distribution feature vectors should also be different.

Our experimental setup was as follows. We extracted all words from the corpus that occurred more than 5 times in BIO-dev to have a somewhat reliable estimate for the majority tag. We then compared feature vectors that were computed on source-only data $\mathcal{D}_S^{u,l}$ with feature vectors computed on target-only data $\mathcal{D}_T^{u,l}$. We chose to make the following simplification. Instead of comparing the full feature vectors, we selected only those dimensions, whose values were non-zero in both vectors. Our hypothesis was that if we take the full feature vectors, the differences we will find could also be due to a difference in vocabularies.

We computed the mean of the cosine similarity as well as the mean of the Euclidean distance for words with the same majority tags and different tags in both domains. If our hypothesis holds, feature vectors of words should have a greater similarity (shorter distance) if they behave similarly in both domains, e.g. have the same majority tag. As Table 4.8 shows, we found some evidence for this hypothesis. Similarity increases if only feature vectors of words with the majority tag are considered. However, standard deviations are quite large. Hence, it is questionable whether we could apply a threshold to find out whether a word is actually used differently in both domains.

token	frequency	tags in BIO-dev	tags in WSJ-train
found	37	VBN 0.86 VBD 0.14	VBD 0.65 VBN 0.35
DNA	27	NN 1.00	NNP 0.67 NN 0.33
detected	24	VBN 0.96 VBD 0.04	VBD 0.56 VBN 0.44
both	17	CC 0.65 DT 0.35	DT 0.91 PDT 0.03 CC 0.06 CD 0.00
observed	12	VBN 0.92 VBD 0.08	VBD 0.69 VBN 0.15 JJ 0.15
examined	10	VBN 0.60 VBD 0.40	VBD 0.67 VBN 0.33
more	10	RBR 0.90 JJR 0.10	JJR 0.58 JJ 0.00 RBR 0.41
reported	10	VBN 1.00	VBD 0.73 VBN 0.25 JJ 0.02
show	9	VBP 0.67 VB 0.33	NN 0.40 VB 0.37 VBP 0.23
RA	8	NN 1.00	NNP 1.00

Table 4.9: Tag distributions in BIO-dev and WSJ-train of known words which have different majority tags, e.g. they get mistagged by the current approach. They are sorted by their frequencies in BIO-dev.

	$\mathcal{D}^{u,l}_{S,T}$	$\mathcal{D}^{u}_{T}$	$\mathcal{D}^{u,l}_{T}$	naive
k=1	92.26	92.24	92.24	92.24
k=5	74.72	73.92	73.84	73.06
k=10	69.48	67.00	67.37	69.65
k=25	66.51	64.87	64.77	66.52
k=50	64.05	58.55	58.25	65.22

Table 4.10: Accuracies on known words of BIO-dev using a k-NN classifier with distribution feature vectors based on different representation datasets.

4.7.1 Known Words

So far, we have mainly focused on tagging unknown words. In this section, however, we want to examine whether distributional information can also help with known words. Our approach in Section 4.5.3 used a word's majority tag if it encountered a known word; otherwise the k-NN classifier was invoked. With this approach, we were able to get an accuracy of 92.26% (9505 out of 10302) on known words of BIO-dev. Superior to that would be a word-based tagger that assigns all known words their majority tag in the target domain. Such a tagger could achieve an accuracy of 96.51% on known words of BIO-dev. This means that we could theoretically increase accuracy by more than 4% if we were able to correctly predict every word's majority tag in BIO.

We looked at all known words that had different majority tags in WSJ and BIO in order to find out where we lose accuracy. Table 4.9 lists the frequencies and tag distributions in both corpora for mistagged words of BIO-dev. We can see that there are mainly two sources of error. Words that are annotated differently due to deviating conventions make up the first group. Examples for these words are "DNA" and "RA", both of them are labeled as proper nouns in WSJ-train but are treated as normal nouns in BIO-dev. As we assume homogenous annotation guidelines, there is nothing we can do about this problem.

The second and larger group of errors encompasses words with different uses across the two domains. Verbs like "detect", "observe" or "examine" are mostly used in their past tense form in the WSJ domain, for example in "Reuters reported that ...". In BIO, they occur mainly as past participles like in "as previously reported". Similarly, "both" is most often used as a conjunction in BIO-dev, whereas it mainly serves as a determiner in WSJ-train.

We re-ran our k-NN tagger on all known words of BIO-dev to see how well our existing approach works for known words. Feature vectors were constructed using three different representation data sets, $\mathcal{D}^{u,l}_{S,T}$, $\mathcal{D}^{u}_{T}$ and $\mathcal{D}^{u,l}_{T}$. We also included the standard model with naive selection in the comparison. Results are reported in Table 4.10. The results in the first row correspond to majority tag selection. Increasing the neighborhood sizes decreases accuracies for all models, as more and more incorrect tags are taken into account. It seems that majority tagging is still the best word-based method for known word tagging. Moreover, we couldn't replicate the results from the previous section. The results also suggest that the way in which feature vectors are computed appears to be less important when tagging known words.

Another approach could involve constructing feature vectors on each domain separately. As we have separate feature vectors for known words in both domains, we can compare them and only deviate from majority tagging if their distribution vectors differ too much. Given that the differences are not due to different vocabularies, this method could help spot critical words. However, the results of the section above suggest that we cannot reliably use the differences between the two vectors in order to predict whether their majority tags also differ.

4.8 Discussion

We were able to answer all of the questions at the beginning of this chapter with the help of our experiments.

Are the feature representations presented here useful for domain adaptation?

We found that distribution and cluster features can effectively help with out-of-domain POS-tagging, especially with unknown words. We achieved accuracies close to 71% on unknown words with a simple word-based tagger using distributional features. This tagger even outperformed our supervised baseline tagger on unknown words. However, we did not compare it against a supervised tagger which receives the same feature vectors as input. We include a CRF-based tagger in our final evaluation in Chapter 6 to allow a better comparison.

We tested tag vectors as an alternative to word-based distribution features. Tag vectors were introduced to facilitate treatment of ambiguous words. However, both variants of tag vectors performed rather poorly, indicating that the default word-based approach still is the best option. Moreover, tag vectors are conceptually more complex and introduce additional problems due to data sparseness.

What is the effect of applying dimensionality reduction techniques such as SVD to our features?

The effect of applying SVD to our feature vectors was less clear. Although SVD did not help in many cases, models with cluster features benefited from it. Maybe the information that cluster features introduce is better captured by the linear reduction that the SVD performs. However, we didn't test for multiple values of d, the number of dimensions. Also, applying a dimensionality reduction technique might not be appropriate in the case of unknown words where subtle differences might be of special relevance.

Where are the limitations of the approaches taken?

There are various limitations to our approaches. One natural problem arises when there is not enough data to have sufficient context information for a word, an issue that often occurs for rare words. In these cases, we could try to also consider a word's neighbors when trying to tag it. Schütze (1995) used distributional information of the left and right neighbor of a word to incorporate this information into the feature vectors. Another idea is to include other meaningful information in the set of features. Chapter 5 looks at two possibly useful feature sets.

Our k-NN approach did not work well on known words. This might be due to the fact that accuracies on known words were already high. Moreover, there are differences in distribution between unknown words and known words. Most unknown words in BIO were nouns, proper nouns or adjectives whereas known words had a more diverse tag distribution. Known words also encompass frequent words with a number of closed-class tags, for example determiners or prepositions. Treating them the same way as unknown words doesn't seem to be an appropriate approach here. Another idea was to only classify known words if their majority tags differ in the source and target domain. However, we were not able to find a reliable criterion that could reliably predict such cases.

Yet another difficulty is finding the right weights for each feature. Our k-NN classifier was highly sensitive to the set of features we used. Also, finding a good value for k, the number of neighbors to be considered, was crucial for good performance. For these reasons, we use support vector machines (SVMs) for our final experiment in Chapter 6.

SVMs are able to scale features automatically; hence, we expect classification to be more stable with them.

What are interesting parameters to be included in future experiments?

We found two groups of parameters that had interesting effects on performance. Parameters in the first group, SVD and capitalization, can be thought of as extensions to the basic approach. Our experiments suggested that capitalization might be an important factor in some domains. If domain data has highly inconsistent spelling conventions, it might me better to lowercase all data in a preprocessing step. As mentioned before, the effects of SVD were dependent on the feature sets at hand. Hence, we think that the effects of SVD should be further examined in future experiments.

In the second group are all parameters directly concerning feature vectors, like weighting. Our experiments showed that results were sensitive to the feature transformation method used; transforming features with a binary function instead of tf-weighting gave improved results. Another commonly used weighting scheme that should be tested in future experiments is tf-idf which weights down frequent contexts. We also observed that representation data plays an important role in the design process. Although we found better accuracies for feature vectors that were constructed from target-only data, this result might not transfer to other domains and needs to be studied further. Last is the set of indicator words used. Our three selection methods yielded quite different results. Best was the naive selection method which simply chose the top n most frequent tokens of the corpus. As adding more features also changed results, we need to investigate this variable further. We compare the naive selection method with full feature vectors using all indicator words in Chapter 6.

Chapter 5

Exploring Other Features

We have seen in the previous chapter that distributional features do not help in all cases. Issues arise when unknown words are relatively rare in the unlabeled data set; which is reasonable given that unknown words are assumed to be rare in general. Also, words that have similar contexts but different tags pose a problem. In both cases, we could benefit from additional information. Ideally, we would like new features to be complementary to an existing feature set, so we can benefit in cases where the old features did not provide enough information.

Two common sets of features that have been used for part-of-speech tagging in literature are morphological features (we call them *shape features*) and orthographical features (Giménez and Màrquez, 2004). Shape features are supposed to help in situations in which words are derived using some set of rules. As those rules are often closely coupled with grammatical behavior, we can use this information to infer a word's part-of-speech tag. Orthographic features are thought to exploit information about the spelling in a word. In many languages, we can find characters that serve special purposes, like digits. In some cases, the characters of a word coincide with certain part-of-speech tags, for example a token consisting just of digits is typically a cardinal number. Using orthographic information, we can build additional features that are supposed behave relatively complementary to shape features.

5.1 Shape Features

Many part-of-speech taggers incorporate morphological information to build robust features (Tseng et al., 2005; Miller et al., 2007). Common feature choices include a word's suffixes or prefixes. Suffixes are likely to be helpful because regular processes of inflectional and derivational morphology do not change in English when going from one domain to the next. Words ending in -ing are more likely to be gerunds than words that do not etc. Traditionally, these features are used as input features to a supervised tagging algorithm which will weight them accordingly.

In this section, we present a new way of leveraging morphological information. First, we construct morphological feature vectors for each word and then apply our standard k-NN-classifier to find similar words in the training set. We hope that similarity in this feature space will extend to similarity on a grammatical (e.g. part-of-speech) level as well.

	normalized vectors		unnormalized vectors	
	all words	$\lvert w\rvert \geq 5$	all words	$\lvert w\rvert \geq 5$
k=1	44.53	43.37	43.95	42.87
k=3	42.64	42.60	40.25	41.22
k=5	46.22	45.69	42.84	44.80
k=10	42.87	48.77	44.38	43.91
k=15	43.91	**49.69**	44.95	45.38
k=25	43.72	45.34	44.14	44.84
k=50	39.64	42.14	41.56	42.37

Table 5.1: Accuracies on unknown words of BIO-dev using a k-NN classifier with shape features. Besides the standard training set WSJ-train, the k-NN algorithm also used a subset of it corresponding to the "$|w| \geq 5$" caption.

5.1.1 Construction

We start by building a list of all suffixes of a given corpus. Each of these suffixes corresponds to a dimension in the resulting feature vector. Given a word w, we set each dimension in its feature vector to one if the corresponding suffix is present in w and to zero otherwise. Our resulting list of suffixes had 465,046 entries that were created by adding every possible suffix from the 147,606 words of BIO and WSJ, e.g. $\mathcal{D}^{u,l}_{S,T}$. Because the average word length of our vocabulary was about 8.2, a lot of suffixes have to occur at least two times. These findings indicate that we should be able to find a sufficient number of words that share a suffix.

Our experimental setup is similar as in our previous experiments in Chapter 4. Remember that for finding the tag of an input word w, we looked at the tags of the k most similar words in WSJ-train. We test two modifications of the original algorithm that are designed to account for special properties of the new feature vectors. First, instead of using the standard cosine similarity, we omit normalizing the vectors. Hence, the resulting similarity measure is nothing more than the dot product of two vectors. We rationalize that in order to find morphologically similar words, matches with words of greater length shouldn't be penalized by normalization. The second modification involves restricting the words in our training set WSJ-train so as to require words to have a minimum length. We call this step training set *filtering*. The motivation for this is that we want to avoid the suffixes of a word to match stop words.

5.1.2 Results

Table 5.1 presents results for all four possible combinations of our modifications. We can see that using normalized feature vectors usually yielded better results than using unnormalized ones. Also, leaving out words with less than 5 characters increased accuracy in the vast majority of cases. We believe that this effect is mainly due to the elimination of words with closed-class tags and rare tags, such as SYM. Best accuracies were found for values of k around 10 or 15. These values for k are typically lower than the optimal values of k with distributional feature vectors. This seems reasonable because most suffixes are not too frequent in the corpus. Hence, it is a good idea to only consider a relatively small number of neighbors with morphological feature vectors. Our best model yielded an accuracy of almost 50%. Keeping in mind the relatively little amount of information encoded in the feature vectors, we find this result to be rather surprising.

The following examples are thought to provide some intuition for tagging with the help of shape features. Generally, shape features work well when they are characteristic of certain part-of-speech tags, for example a word ending in -ially will most likely be an adverb. Here is another example for a word with a relatively indicative suffix:

```
MorphFeatures.printMostSimilar('Nonepithelial')
Token            Similarity       Tags in WSJ-train
trial            0.3721            NN 1.00
Trial            0.3721            NNP 0.75 NN 0.25
aerial           0.3397            JJ 1.00
burial           0.3397            NN 1.00
denial           0.3397            NN 1.00
facial           0.3397            JJ 1.00
genial           0.3397            JJ 1.00
jovial           0.3397            JJ 1.00
racial           0.3397            JJ 1.00
Racial           0.3397            JJ 1.00
```

We can see that the suffix -ial causes the algorithm to retrieve mostly adjectives. However, there are also cases where a suffix provides less useful information like in the following example:

```
MorphFeatures.printMostSimilar('stabilizer')
Token            Similarity       Tags in WSJ-train
fertilizer       0.6000            NN 1.00
Pfizer           0.5164            NNP 1.00
oxidizer         0.4472            NN 1.00
Nazer            0.4243            NNP 1.00
organizer        0.4216            NN 1.00
Blazer           0.3873            NNP 1.00
buzzer           0.3873            NN 1.00
Frazer           0.3873            NNP 1.00
Glazer           0.3873            NNP 1.00
Sulzer           0.3873            NNP 1.00
```

Although there are a couple of words with the right tags among the ten most similar words, the k-NN approach failed to work because WSJ-train contained too many proper nouns with the same ending. A potential solution could be to exclude capitalized words from the comparison, if a word isn't capitalized itself.

5.2 Orthographic Features

Information about the characters in a word can be indicative of a number of part-of-speech classes. For example, capital letters at the beginning of a word often signal proper nouns. Another simple example is accents that can be typical for foreign words. However,

feature set	BIO-dev	WSJ-train
baseline	89.05	98.30
no-svd	89.04	98.29
svd	89.22	98.31
svd-context	**89.23**	**98.36**

Table 5.2: Accuracies on the development and training set for varying sets of orthographic features.

crafting good features that exploit these properties by hand seems daunting because of the large number of possibilities. We need to come up with a good set of rules that captures all important concepts, i.e. special characters, symbols and capitalization. Also, most orthographic features are language specific and need to be re-designed when moving on to a new language. We present an approach here that doesn't suffer from these limitations as the design of the features is unsupervised.

5.2.1 Constructing Feature Vectors

Our approach to leverage character-based information was similar to the construction of distributional features. We construct a vector for each word that holds the distribution of characters in this word. That is, the number of dimensions of a each feature vector corresponds to the number of different characters found in the corpus. Entry i in the feature vector is the number of times character c_i occurs in a word. The resulting vector is then length-normalized.

An extension we test is SVD to reduce the number of dimensions. The motivation behind this the following. We hope that the SVD will find dimensions which represent important concepts by leveraging correlations in the input data. If characters occur together in words frequently, we assume that there is some underlying concept connecting those characters. We hope that the SVD is eventually able to uncover concepts like "alphanumeric characters" or "digits". We apply SVD to the feature matrix of all words in the corpus $\mathcal{D}^{u,l}_{S,T}$. For the experiment carried out here, we set the number of dimensions to 50.

We test three different feature sets that were used as additional input features to the discriminative tagger from Chapter 3. We chose to run the full experiment because we expect orthographical features to convey less meaningful information than the previous feature sets. The feature sets are

- no-svd: We just use the raw feature vectors that contained the character distribution of a word.
- svd: These were exactly the dimension-reduced feature vectors described above.
- svd-context: In addition to the svd feature vector of the current word w, we added the svd feature vectors of the words to the left and right of w.

5.2.2 Results

The results in Table 5.2.2 show that we are able to achieve an improvement of 0.18% over the baseline using orthographic features. We get the best results for the models which use SVD feature vectors. Adding information from the preceding and following token does only result in a marginal improvement on the development set, while the accuracy on the training set further increases, possibly indicating overfitting.

Another interesting observation we can make here is that using just unreduced orthographic features does not help; they rather cause performance to decrease a bit. A reason for this might be that noise and sparsity in the feature vectors prevent the tagger from learning stable concepts.

5.3 Discussion

Our experiments showed that there are two sets of features which can provide valuable information for unknown word tagging. Our word-based k-NN tagger achieved accuracies around 50% using just shape features. We combine shape features with distributional features in our final experiments in Chapter 6 and examine their robustness across several target domains. Another important idea that we reuse there is the one of training set filtering. Filtering is thought to remove all misleading training instances. Here, we used a fixed threshold b for cutting off words of length less than b which may not be optimal for all domains. We study the effect of this parameter b more closely in Chapter 6 as well.

Orthographic features can increase accuracy in some cases. However, orthographic information can be highly dependent on the domain. While capitalized words in WSJ are mostly proper nouns, BIO texts have rather inconsistent spelling conventions. In such cases, orthographic information might be misleading. Also, the overall performance gains were small. Hence, we decided to leave out these features for the final comparison of Chapter 6. It would be interesting for future experiments, though, to compare handcrafted orthographic features with the features extracted via SVD.

Chapter 6

Feature Robustness Across Domains

So far, we have been investigating the effectiveness of several feature sets in isolation. We found that both distributional features and shape features carry information that can help with unknown word tagging. Moreover, our experiments in the previous two chapters revealed that there are various parameters which can affect performance; some of them are likely to influence each other. We have also seen that removing certain instances from the training set can improve performance in some cases.

In order to come up with recommendations for designing features for domain adaptation in part-of-speech tagging, we present two experiments in Section 6.2 which investigate all important parameters systematically. In the first experiment, we try to find default values for all parameters that are likely to affect each other. In the second experiment, we use the settings from the first experiment to test further extensions that are thought to behave more independently.

There are three major differences to the experiments in the previous chapters. First, we use two different sets of features, distributional and shape features, in combination. Second, we evaluate our approaches on multiple target domains instead of only one. This is supposed to both test the cross-domain robustness of our features as well as to prevent overfitting to one specific target domain. Lastly, we use support vector machines for classification, thereby dismissing our k-NN classifier. We opted in favor of SVMs because the underlying statistical framework is more rigorous; moreover, they are able to scale features themselves which should lead to more robust results.

6.1 Experimental Data and Setup

Our source domain is the Penn Treebank (Marcus et al., 1993) of Wall Street Journal (WSJ) text as in our previous experiments. This time, however, we evaluate on six different target domains (TDs). The first target domain is the Penn BioTreebank data set (BIO). The remaining five target domains (newsgroups, weblogs, reviews, answers, emails) are from the SANCL shared task (Petrov and McDonald, 2012).

Like the experiments in Chapter 4 and 5, we adopt a simple approach of *word classification*. Again, the objects to be classified are words and the classes are the parts-of-speech of the source domain. The gold label of a word in training is the majority tag in the source domain. A prediction for an unknown word is then made by computing its feature representation and applying the learned classifier.

We adopt word classification instead of the more common sequence labeling setup because word classification is much more efficient to train and allows us to run a large number of experiments efficiently. Our experiments demonstrate that word classification accuracies are comparable with or higher than sequence labeling in part-of-speech tagging for unknown words.

We use LIBSVM (Chang and Lin, 2011) to train $\binom{k}{2}$ one-vs-one classifiers on the training set, where k is the number of part-of-speech tags in the training set. The SVMs were trained with untuned default parameters; in particular, $c = 1$. For sequence classification, we use CRFSuite (Okazaki, 2007), a Conditional Random Field (CRF) toolkit. Apart from the word features described below, we use the base feature set of Huang and Yates (2009) for CRFs, including features for state, emission and transition probabilities. CRFs are trained until convergence with a limit of 300 training iterations.

6.1.1 Features

Two sources of information have shown to be valuable in our experiments in Chapter 4 and 5 when predicting the part-of-speech of an unknown word in an unsupervised setting: the word itself (sequence of letters, shape etc) and the context(s) in which it occurs. Motivated by these results, we create a feature representation for each word that has three components: left context information, right context information and shape information. We will refer to left/right context information as distributional information. Let f be the function that maps a word w to its (full) feature vector. We then define f as follows:

$$f(w) = \begin{bmatrix} f_{\text{left}}(w) \\ f_{\text{right}}(w) \\ f_{\text{shape}}(w) \end{bmatrix}$$

Based on the intuition that each of the three sources of information is equally important, each of the three component vectors is normalized to unit length.

For both distributional and shape features, we have a choice of either using *all possible features or a subset consisting of the most frequent features.* We directly compare these two possibilities, using recommended values from the literature for the subset condition: the 250 most frequent features (indicator words) for distributional vectors (Schütze, 1995) and the 100 most frequent features (suffixes) for shape vectors (Müller et al., 2012). Each component vector has an additional binary feature that is set to 1 if the rest of the vector is zero, and 0 otherwise.

Distributional features

Like in our experiments in Chapter 4, the i^{th} entry x_i of $f_{\text{left}}(w)$ is the number of times that the *indicator word* t_i occurs immediately to the left of w:

$$x_i = \text{freq}\left(\text{bigram}(t_i, w)\right)$$

where t_i is the word with frequency rank i in the corpus. $f_{\text{right}}(w)$ is defined analogously.

Many different ways of defining and transforming distributional features have been proposed in the literature. We systematically investigate the following variables that our previous experiments proved to be important:

(i) weighting
(ii) dimensionality reduction
(iii) selection of data that distributional vectors are based on

We experiment with three different *weighting functions* that transform non-zero counts as follows.

(i) tf: $w_{\text{tf}}(x_i) = 1 + \log(x_i)$

(ii) tf-idf: $w_{\text{tf-idf}}(x_i) = (N/\log \mathrm{df}_{t_i})(1 + \log(x))$ (where N is the total number of words and df_{t_i} the number of words that indicator word t_i is a non-zero feature of)

(iii) binary: $w_{\text{bin}}(x_i) = 1$

Tf weighting gives lower weight to frequent terms to prevent them from dominating feature vector comparisons. The motivation for binary weighting is that the counts might not be relevant at all when computing similarities – the key information is whether an indicator has been observed or not. The intuition behind idf weighting is that indicators that occur with a large number of words should be downweighted because they are less informative than those that occur with few.

Transformation operations like *dimensionality reduction* (Deerwester et al., 1990) and autoencoding (Hinton et al., 2006) can be effective in improving generalization in machine learning, in particular in nonstandard settings like domain adaptation where a labeled random sample of the target domain is not available. We test singular value decomposition (SVD) here because it has been used in prior work on part-of-speech tagging (Huang and Yates, 2009). We apply SVD to the matrix of all feature vectors and keep the dimensions corresponding to the $d = 100$ largest singular values.

We compute distributional vectors either on target data only (i.e., on $\mathcal{D}_T^{u,l}$) or on the union of source and target data (i.e., $\mathcal{D}_{S,T}^{u,l}$). We compare these two alternatives and show in our experiments that source distributional information does not consistently improve performance.

Shape features

For a selected suffix s, we simply set the dimension corresponding to s in $f_{\text{shape}}(w)$ to 1 if w ends in s and to 0 otherwise. We either select all suffixes or the top 100, depending on the experiment.

In addition to suffixes, we investigate two other representational variables related to shape that we found to have an effect on performance: case and digits. For case, we compare keeping case information as is with converting all uppercase characters to lowercase characters. For digits, we compare keeping digits as is with converting all digits to the digit 0; e.g., \$1,643 is converted to \$1,000. We call these two transformations *case normalization* and *digit normalization*.

6.1.2 Training Set Filtering

The key challenge in domain adaptation is that the distributions of source and target are different. The experiments in Chapter 5 suggested a new approach that could make the distributions more similar: the elimination of all short words from the training set. We call this (training set) *filtering*. The reason this is promising is that longer words are more likely to be examples of productive linguistic processes than short words – even if this is only a statistical tendency with many exceptions. We show below that training on long words improves accuracy by several percentage points on one target domain.

6.2 Experimental Results

We train $\binom{k}{2}$ binary SVM classifiers on the feature representations we just defined. The training set consists of all words that occur in the WSJ training set (in condition $\mathcal{D}^{u,l}_{S,T}$) or all words that occur in both WSJ and $\mathcal{D}^{u,l}_{T}$ (in condition $\mathcal{D}^{u,l}_{T}$). An unknown word is classified by building its feature vector, running the classifiers on it and then assigning it to the part-of-speech class returned by the LIBSVM one-vs-one setup.

We divide our experiments into two parts. In the *basic experiment*, we investigate four parameters of the model that are likely to interact with each other: dimensionality of shape vectors (ALL vs. 100 most frequent suffixes), dimensionality of distributional vectors (ALL vs. 250 most frequent indicator words), use of dimensionality reduction (SVD: yes or no) and weighting of distributional vectors (bin, tf, tf-idf).

In the *extended experiment*, we then investigate the effect of other parameters on the best performing model from the basic experiment: distributional vectors based on $\mathcal{D}^{u,l}_{S,T}$ vs $\mathcal{D}^{u,l}_{T}$, case normalization, digit normalization, completely omitting either shape or distributional information and training set filtering. For the basic experiment, these parameters are set to the following values: distributional vectors are computed on $\mathcal{D}^{u,l}_{T}$, case normalization is used, digit normalization is not used, and the training set is not filtered (i.e., all words are included in the training set).

6.2.1 Basic Experiment

Table 6.1 gives the results of the basic experiment: the 24 possible combinations of number of shape features, number of distributional features, use of dimensionality reduction and weighting scheme. In each column, the best three accuracies are underlined and the best accuracy is doubly underlined; the results significantly different from the best result are marked with a dagger.[1]

The goal of the basic experiment is to exhaustively investigate combinations of the four parameters that we suspect to have the strongest interaction with each other and then find a parameter combination that is a good basis for testing the remaining parameters in the extended experiment. The guiding principle in this investigation is that when in doubt, we select the simpler or default setting for the extended experiment in order to make as few assumptions as possible.

For the number of shape features, ALL generally does better than 100. Five target domains have their best result for ALL: reviews, blog, answers, email (line 14) and BIO (line 24). The exception is grp (best result on line 3). The reason seems to be that the newsgroups target domain contains a larger number of unknown words with suffixes that do not support part-of-speech generalization well. E.g., the suffixes -ding, -eding, -eeding, -breeding of a newsgroup name like "alt.animals.horses.breeding" (mistagged as VBG, gold tag: NN) are misleading. Despite these problems, the best 100 result for newsgroups is not significantly better than the best ALL result (lines 3 vs. 20). This argues for using the setting ALL for the extended experiment.

For a number of distributional features, there is a similar tendency for the WEB target domains (grp, reviews, blog, answers, email) to do slightly better for fewer features (250) than ALL features. However, BIO clearly benefits from using the full dimensionality of the distributional feature space: all 250 results are statistically worse than the best

[1] $p < .05$, 2-sample test for equality of proportions with continuity correction. We use the same test and level for all significance results in this chapter.

	shape	dist	svd	weight	grp	reviews	blog	answers	email	BIO
1	100	250	n	bin	56.88	63.92	67.13†	52.14	63.30	65.64†
2				tf	56.50	65.67	70.33	52.47	64.37	63.14†
3				tf-idf	57.14	65.83	70.23	51.86†	64.14	64.94†
4			y	bin	52.52†	54.68†	62.74†	47.81†	60.08†	70.29†
5				tf	54.42	58.18†	68.01†	48.14†	61.70†	69.70†
6				tf-idf	54.73	57.44†	68.75†	48.93†	61.38†	70.95†
7		ALL	n	bin	55.98	63.60	68.70†	52.14	62.87	68.92†
8				tf	56.58	64.67	70.82	51.02†	63.52	65.72†
9				tf-idf	56.15	63.50	68.85†	50.09†	61.87†	68.61†
10			y	bin	52.05†	52.82†	60.67†	41.95†	59.82†	68.57†
11				tf	53.65†	57.23†	66.24†	43.02†	61.22†	69.82†
12				tf-idf	54.21†	55.47†	64.17†	42.50†	58.52†	69.11†
13	ALL	250	n	bin	56.02	65.04	70.77	54.05	64.37	68.45†
14				tf	55.59	66.05	72.45	55.03	64.43	64.82†
15				tf-idf	55.93	65.99	72.10	54.98	63.98	65.87†
16			y	bin	52.48†	56.16†	65.50†	43.48†	59.79†	70.64†
17				tf	53.26†	59.46†	68.95†	48.51†	60.60†	68.68†
18				tf-idf	54.16†	59.56†	68.70†	44.18†	60.66†	69.35†
19		ALL	n	bin	56.06	63.55	68.85†	54.38	59.85†	66.22†
20				tf	56.62	64.61	71.86	54.28	61.05†	65.64†
21				tf-idf	56.15	63.07	69.74	52.65	59.95†	65.25†
22			y	bin	52.35†	55.74†	62.89†	41.95†	58.68†	71.07†
23				tf	53.99†	59.83†	68.16†	43.62†	60.37†	69.93†
24				tf-idf	54.81	58.98†	68.65†	41.95†	58.68†	74.39

Table 6.1: Accuracy of unknown word classification in the basic experiment. The performance of the best (three best) parameter combinations per column are doubly (singly) underlined. A dagger indicates a result significantly worse than the column's best result.

ALL result and the gap to the best 250 result is large (line 24 vs line 6, a difference of $74.39 - 70.95 = 3.44$). The gap between best 250 result and best ALL result is smaller for the other five target domains (although only slightly smaller for email) and for each of the five target domains there is an ALL result that is statistically indistinguishable from the best 250 result. For this reason, we choose dist=ALL for the extended experiment. Simply using ALL indicator words also has the advantage of eliminating the need to optimize an additional parameter, the number of indicator words selected.

In a way similar to distributional features, the behaviors of WEB and BIO target domains also diverge for dimensionality reduction. The top three results for the WEB target domains are always achieved without SVD (lines 1, 3, 13, 14, 15, 19, 20), the top three results for the BIO target domain are all SVD results (lines 6, 22, 24). We opt for the simpler option (no SVD) for the extended experiment in the absence of strong consistent cross-target-domain evidence for the need of dimensionality reduction. We will also see in the extended experiment that we can recover and surpass the best BIO result (74.39, line 24) by optimizing other parameters.

The results on weighting argue against using binary weighting: the six best results in the Table all use tf weighting, either by itself or in conjunction with idf (lines 3, 14, 24). Apparently, the distinction between lower and higher frequencies of indicator word occurrences is beneficial for unknown word classification. Whether tf or tf-idf is better, is less clear. For two target domains, tf-idf yields the best result (grp on line 3, BIO on line 24), for four target domains tf (reviews, blog, answers, email: line 14). The difference between best tf-idf and best tf result is not significant for grp and we will show below that we can get tf results for BIO that are better than the best tf-idf result of 74.39 in Table 6.1. For this reason, we choose the setting tf for the extended experiment. Again, we are selecting the simpler of two options (tf vs tf-idf) when faced with somewhat mixed evidence.

In summary, based on the results of the base experiment, we choose the following settings for the extended experiment: shape = ALL, dist = ALL, svd = n, weight = tf. For shape, dist, and svd this is the simpler of two possible settings. For weighting, we choose tf (instead of the simpler binary option) because of clear evidence that some form of frequency weighting is beneficial across target domains. These settings correspond to line 20 in Table 6.1. This line is repeated as the baseline on line 1 in Table 6.2.

6.2.2 Extended Experiment

In the extended experiment, we investigate the effect of additional parameters. Results are shown in Table 6.2. Underlining conventions and statistical test setup are identical to Table 6.1. The CRF baseline used a parameter setting similar to word classification with two exceptions: we set dist=250 because we were not able to run dist=ALL due to memory limitations; and we convert all features to binary due to space restrictions.

Using sequence classification instead of word classification for unknown word prediction does not consistently improve results (line 2). For grp and answers, the CRF achieves the best overall accuracy, but the difference to the baseline is not significant. For the other four target domains, the best result occurs in a different parameter setting. For BIO, a large drop in performance occurs (from 65.64 to 56.62), perhaps suggesting that word classification is more robust than sequence classification for unknown words.

Calculating distributional vectors on both source and target (as opposed to target only) has similarly inconsistent effects (line 3). Performance compared to the baseline

		grp	reviews	blog	answers	email	BIO
1	baseline	56.62	$\underline{64.61}$	$\underline{71.86}$	54.28	61.05†	65.64†
2	CRF	$\underline{58.18}$	64.51	70.48	$\underline{56.52}$	$\underline{63.10}$	56.62†
3	$\mathcal{D}^{u,l}_{S,T}$	55.50	64.13	$\underline{\underline{72.50}}$	$\underline{55.31}$	62.91	65.17†
4	no case NRM	52.83†	64.45	70.68	52.00†	59.27†	67.51†
5	digit NRM	$\underline{56.80}$	$\underline{64.61}$	$\underline{72.01}$	54.05	$\underline{\underline{63.88}}$	68.61†
6	shape only, ALL	48.77†	45.32†	56.58†	39.90†	49.19†	52.52†
7	shape only, 100	47.69†	39.16†	51.90†	36.17†	47.24†	50.14†
8	dist only, ALL	52.05†	63.34	68.21†	47.07†	53.06†	73.41†
9	dist only, 250	51.49†	64.13	66.34†	45.76†	54.13†	72.86†
10	$\lvert w\rvert > 1$	56.58	$\underline{\underline{64.67}}$	71.81	$\underline{54.84}$	60.83†	65.99†
11	$\lvert w\rvert > 2$	$\underline{57.06}$	$\underline{64.61}$	71.56	54.38	$\underline{63.17}$	68.61†
12	$\lvert w\rvert > 3$	55.33	60.89†	69.69	48.79†	62.39	73.84†
13	$\lvert w\rvert > 4$	52.87†	60.10†	67.67†	47.53†	53.06†	77.66
14	$\lvert w\rvert > 5$	53.09†	59.35†	66.58†	44.37†	51.69†	77.66
15	$\lvert w\rvert > 6$	52.27†	58.55†	66.93†	43.25†	49.74†	$\underline{77.74}$
16	$\lvert w\rvert > 7$	51.96†	56.64†	63.18†	40.46†	47.17†	$\underline{78.41}$
17	$\lvert w\rvert > 8$	49.59†	56.16†	58.26†	39.06†	44.31†	$\underline{\underline{79.77}}$
18	$\lvert w\rvert > 9$	46.87†	52.82†	55.54†	33.94†	42.69†	74.58†
19	$\lvert w\rvert > 10$	43.42†	51.22†	52.54†	33.33†	39.24†	76.10†

Table 6.2: Extended experiment. The effect of various parameter changes on accuracy of unknown word classification. “NRM” = “normalization.

decreases for four target domains and increases for two. Based on this evidence, source-data distributional information is not robust cross-target-domain and should probably not be used.

Omitting case normalization (line 4) consistently hurts for WEB target domains, but helps for BIO. In other words, for BIO it is better not to case-normalize words. This result is plausible because case conventions vary considerably in different target domains. Whether keeping case distinctions is helpful or not depends on how similar source and target are in this respect and is therefore not stable in its effect across target domains.

Digit normalization (line 5) has a minor positive or negative effect on the first four target domains, but increases accuracy by more than 2% in the last two, email and BIO. The makeup of the WSJ tag set makes it unlikely that differences between digits could result in part-of-speech differences that are predictable in unsupervised domain adaptation. This argues for using digit normalization when WSJ is the source domain.

The clearest result of the Table is that distributional information is necessary for good performance. Performance compared to the baseline drops in all cases and all accuracies on lines 6&7 are significantly worse than the best result. Moreover, distributional features seem to encode more meaningful information for POS tagging than shape features; results on lines 6&7 are consistently lower than results on lines 8&9.

The evaluation is similarly consistent for shape information in the WEB target domains (lines 8 and 9). All accuracies are below the baseline, with some of the drops being quite large, e.g., about 7% for answers and email. Surprisingly, omitting shape information results in a large *increase* of accuracy for the BIO target domain. We will further investigate this puzzling result below.

Finally, training set filtering – only training the classifier on words above a threshold length k – is beneficial for all target domains except for blog; and even for blog, moderate filtering has only a negligible negative effect on accuracy (lines 10–11). In principle, the idea of restricting training to longer words because they are most likely to be representative of unknown words seems to be a good one. However, the effect of filtering is sensitive to the threshold length k. We leave it to future work to find properties of the target domain that could be used as diagnostics for finding a good value for k.

The motivation of splitting the experiments into basic experiment and extended experiments was to find a stable point in parameter space for the parameters that are most likely to interact and then look at the effect of the remaining parameters using this stable point as starting point. In Table 6.2, we see that for the WEB target domains, all variations of experimental conditions either hurt performance or produce only small positive changes in accuracy in comparison to the baseline. This is evidence that our strategy of splitting experiments into basic and extended was sound for these target domains.

		BIO
1	baseline	$73.41^{\dagger}$
3	$\mathcal{D}^{u,l}_{S,T}$	$67.94^{\dagger}$
4	no case NRM	$72.39^{\dagger}$
5	digit NRM	$74.15^{\dagger}$
10	$\lvert w\rvert > 1$	$73.96^{\dagger}$
11	$\lvert w\rvert > 2$	$75.24^{\dagger}$
12	$\lvert w\rvert > 3$	$81.30^{\dagger}$
13	$\lvert w\rvert > 4$	$81.88^{\dagger}$
14	$\lvert w\rvert > 5$	$\underline{82.98}$
15	$\lvert w\rvert > 6$	82.47
16	$\lvert w\rvert > 7$	$\underline{\underline{84.46}}$
17	$\lvert w\rvert > 8$	$\underline{83.09}$
18	$\lvert w\rvert > 9$	$79.03^{\dagger}$
19	$\lvert w\rvert > 10$	$80.52^{\dagger}$

Table 6.3: Extended experiment for BIO without shape information. Dist=ALL.

However, the situation for BIO is different. Two parameter changes result in large performance gains for BIO: omitting shape information (increase by 8%, lines 1 vs 8) and filtering out short training words (increase by 14%, lines 1 vs 17). This indicates that the base configuration of the extended experiment is not a good starting point for exploring parameter variation for BIO.

For this reason, we repeat parts of the extended experiment without any shape information. As we would expect, we obtain results for WEB target domains that are consistently worse than those in Table 6.2 (not shown), with one exception: a slight increase for $|w| > 8$ in email. However, the results for BIO are much improved as shown in Table 6.3.

We conclude that shape information is not robust across target domains in domain adaptation. Shape information is helpful for the WEB target domains, but it decreases performance by about 10% for BIO. We will analyze the reason for this discrepancy in the next section.

As a last set of experiments, we run the optimal parameter combination ($|w| > 7$ in Table 6.3, 84.46) on the BIO test set and obtained an accuracy of 88.13. This is more than 10% higher than the best number for unknown word prediction on BIO published up to this point (76.3 by Huang and Yates (2010)). For the experimental conditions with the best WEB results in Table 6.2 (double underlining), we get the following test accuracies: grp=56.66, reviews=67.79, blog=64.80, answers=66.51, email=65.51. These are either better than dev or slightly worse except for blog; the blog result can be explained by the fact that the blog base model (line 1) also is a lot worse on test than on dev (66.08 vs 71.86). We interpret these test set results as indicating that we did not overfit to the development set in our experiments.

6.2.3 Summary

We have investigated the cross-target-domain robustness of a number of configurational choices in domain adaptation for POS tagging. Based on our results, the following choices are relatively robust across target domains: using ALL indicator words (as opposed to a subset) for distributional features, no dimensionality reduction, tf weighting, digit normalization, target-only distributional features, and formalization of the problem of unknown word prediction as word classification (as opposed to sequence classification).

We found other choices to be dependent on the target domain, in particular the use of shape features, case normalization and training set filtering.

The most important lesson from these results is that many aspects of domain adaptation are highly dependent on the target domain. Given our results, it is unlikely that a single domain-adaptation setup will work in general. Instead, criteria need to be developed that allow us to predict which features and methods work for different target domains.

6.3 Analysis and Discussion

The biggest target-domain differences we found in the experiments are those between WEB and BIO: they behave differently with respect to dimensionality reduction (bad for WEB, good for BIO), shape information (good for WEB, bad for BIO) and sequence classification (neutral for WEB, bad for BIO).

One hypothesis that could explain these results is that the difference between BIO and WSJ is larger than the difference between WEB and WSJ. For example, dimensionality reduction creates more generalized representations, which may be appropriate for target domains with large source-target differences like BIO; and WSJ suffixes may not be helpful for BIO because biomedical terminology has suffixes specific to scientific vocabulary and is rare in newspaper text. In contrast, WEB suffixes may not diverge as much from WSJ since both are "non-technical" genres.

TD	tags	suffixes	transitions
grp	.009	.275	.068
reviews	.057	.352	.212
blog	.009	.295	.074
answers	.048	.337	.158
email	.036	.273	.139
BIO	.096	.496	.385

Table 6.4: Jensen-Shannon divergences between WSJ and target domains

		grp	reviews	blog	answers	email	BIO
1	baseline	32.77	38.89	43.48	30.52	34.26	40.06
2	CRF	38.74	42.71	46.63	38.08	36.21	39.03
3	$\mathcal{D}^{u,l}_{S,T}$	32.87	38.55	44.75	33.19	35.30	41.42
4	no case NRM	27.08	39.82	39.54	25.80	27.33	39.98
5	digit NRM	32.80	39.09	43.68	30.47	34.69	37.72
6	shape only, ALL	18.02	21.25	24.61	16.25	16.37	26.55
8	dist only, ALL	27.70	38.39	34.38	22.11	29.71	37.01
10	$\|w\| > 1$	32.73	39.48	43.54	30.60	34.20	35.32
11	$\|w\| > 2$	33.33	37.38	43.52	30.02	34.66	35.05
13	$\|w\| > 4$	26.37	28.92	37.68	22.33	24.14	37.55

Table 6.5: Selected conditions of the extended experiment (Table 6.2), evaluated using macroaveraged F_1

One way to assess the difference between two domains is to compare various characteristic probability distributions. The distance of two domains under a representation R has been shown to be important for domain adaptation (Ben-David et al., 2007). Similar to Huang and Yates (2010), we use Jensen-Shannon (JS) divergence as a measure of divergence. Table 6.4 shows the JS divergences between WSJ and the six target domains for different distributions.

The results confirm our hypothesis. BIO is indeed more different from WSJ than the other target domains. Tag distribution divergence is 0.096 for BIO and ranges from 0.009 to 0.057 for WEB. Suffix distribution divergence of BIO is 0.496, almost 50% more than reviews, the WEB target domain with highest suffix divergence. The underlying probability distributions here are $P(\text{suffix}|t)$, where $t \in \{\text{NN, NNP, JJ}\}$ – most unknown words are in these three classes and accuracy is therefore mostly a measure of accuracy on NN, NNP and JJ. Finally, transition probability divergence of BIO for NN, NNP, JJ is also much larger than for WEB. The distribution investigated here is $P(t_{i-1}|t_i)$; we compute the divergence between, say, BIO and WSJ for the three tags and then average the three divergences.

We do not have space to show detailed results on all tags, but the divergences are more similar for closed class POS. E.g., there is virtually no difference in transition probability divergence for modals between BIO and WEB. This observation prompted us to investigate whether some target-domain differences might depend on the evaluation measure used. Accuracy – a type of microaveraging – is mostly an evaluation of the classes that are frequent for unknown words: NN, NNP, JJ. If most of the higher divergence of BIO is caused by these categories, then a macroaveraged evaluation, which gives equal weight to each part-of-speech tag, should show less divergence.

This is indeed the case as the macroaveraged results in Table 6.3 show. These results are more consistent across target domains than those evaluated with accuracy. Removing shape and distributional information now hurts performance for all target domains (lines 6&8). WEB and BIO behave more similarly with respect to training set filtering: the large outliers for BIO we obtained in the accuracy evaluation are gone. Source domain distributional information has a more beneficial effect on F_1 than on accuracy, probably because the classification of parts-of-speech that are more stable across target domains like verbs and adverbs benefits from source domain information. The CRF produces the best result for all WEB target domains. For less frequent POS classes (those that dominate the macroaveraged measure, especially verbal POS), sequence information and "long-distance" context is probably more stable and can be exploited better than for NN, NNP and JJ. However, there is still a drop-off from the baseline for BIO; we attribute this to the larger differences in the transition probabilities for BIO vs WEB (Table 6.4); the sequence classifier is at a disadvantage for BIO, even on a macroaveraged measure, because the transition probabilities change a lot.

It is important to note that even though F_1 results are more consistent for domain adaptation, accuracy is the appropriate measure to use for POS tagging: the usefulness of a tagger to downstream components in the processing pipeline is better assessed by accuracy than by F_1.

Chapter 7

Related Work

Most work on part-of-speech tagging takes a standard supervised approach and assumes that source and target are the same (e.g., Toutanova et al., 2003). At the other end of the spectrum is a completely unsupervised setting (e.g., Goldwater and Griffiths, 2007). Other researchers have addressed the task of adapting a known tagging dictionary to a target domain (e.g., Merialdo, 1994; Smith and Eisner, 2005), which we view as complementary to methods for words about whose tags nothing is known. Subramanya et al. (2010) perform domain adaptation without using any unlabeled target-domain text. All of these applications scenarios are reasonable; however, it can be argued that the scenario we address is – apart from standard supervised learning – perhaps more typical of what occurs in practice: there is labeled source domain text available for training; there is plenty of unlabeled target-domain text available; and there is a substantial number of target-domain words that do not occur in the source domain. Frequently, researchers make the assumption that a small labeled target text has been created (e.g., Daumé III, 2007); in the process, a small number of unknown words may also be labeled, but this is not an alternative to handling unknown words in general.

Work by Das and Petrov (2011) is also a form of domain adaptation for part-of-speech tagging, using universal POS tag sets and parallel corpora. It is likely that best performance for target domains without training data can be achieved by combining our approach with a multilingual approach if appropriate parallel data is available. Ganchev et al. (2012) use another source of additional information, search logs. Again, it should be possible to integrate search-log based features into our framework.

Blitzer et al. (2006) learn correspondences between features in source and target. Our results suggest that completely ignoring source features (and only using source labels) may be a more robust approach for unknown words. Also, their tagging results are only for one target domain, namely BIO. It would be interesting to see how robust his method behaves across different domains.

Cholakov et al. (2011) point out that improving tagging accuracy does not necessarily improve the performance of downstream elements of the processing pipeline. However, improved unknown word classification will have a positive impact on most downstream components.

Choi and Palmer (2012) present a more practical approach to unsupervised domain adaptation. They train two separate models on the available data, a generalized one and a domain-specific one. The generalized model omits words that are rare in the input data and occur only within certain sections or documents whereas the domain-specific one uses almost all words. Since their approach is not conditioned on the underlying tagging model, it would be interesting to integrate their approach with ours.

Van Asch and Daelemans (2010) find linear correlation between tagging performance and similarity of domains. They compare several similarity metrics between the unigram models of various domains. Although most metrics show a linear relation between tagging accuracy and similarity, they obtain the most consistent results for the Rényi divergence. We found similar behavior using Jensen Shannon divergence as metric. However, we computed divergence on probability distributions that made use of labeled data in the target domain. It would be interesting to see whether unlabeled data would suffice to predict tagging performance in our case as well.

Huang and Yates (2009) evaluate CRFs with distributional features. Besides raw feature vectors, they examine lower dimensional feature representations using SVD or a special HMM-based method. In our experiments, we did not find an advantage to using SVD.

Huang and Yates (2010) address the problem of predicting part-of-speech of unknown words using sequence labeling. Huang and Yates (2012) extend this work by inducing latent states that are shown to improve prediction. As we argued above, a word classification approach has several advantages compared to a sequence labeling approach. Since latent sequence states can be viewed as a form of dimensionality reduction, it would be interesting to compare them to the non-sequence-based dimensionality reduction (SVD) we have investigated in our experiments.

Zhang and Kordoni (2006) use a classification approach for predicting part-of-speech for in-domain unknown words. They achieve an accuracy of 61.3%. Due to differences in the data sets used, these results are not directly comparable with ours.

Nakagawa et al. (2001) also apply support vector machines to unknown word handling in POS tagging. Their task was to predict an unknown word's tag in context. The set of features they used comprised a word's neighboring tags and words as well as prefixes and suffixes of up to length 4. They found improved accuracies for their SVM approach compared to a traditional HMM-based tagger. Another interesting result of their work is that they found sequence information (e.g. the POS tags and lexical identities of a word's neighbors) to provide the least information for unknown word tagging. The largest increase in performance they observed was due to orthographic and morphological information. This further motivates our approach of finding robust features that do not make use of sequence information. It should also be noted that our classification task differed in one important aspect from theirs; we tried predict a word's majority tag using global information, thereby completely neglecting a word's local context.

Miller et al. (2007) and Cucerzan and Yarowsky (2000) have both investigated the use of suffixes for domain adaptation. Miller et al. characterized words by a list of hand-built suffix classes that they appear in. They then used a 5-NN classifier along with a custom similarity measure to find initial lexical probabilities for all words. These probabilities were adjusted by EM training of a HMM model in a final step. We also ran preliminary experiments with k-NN, but found that one-vs-one SVM performs better.

Cucerzan and Yarowsky (2000) use distribution as a backoff strategy if no helpful suffix information is available. They address unknown word prediction for new languages. We have found that for within-language prediction, distributional information is generally more robust than shape information, including suffixes.

Finally, we introduce a new simple and effective domain-adaptation technique – training set filtering. For domain adaptation on WSJ→BIO, we report accuracies that are 10% higher for unknown words than previous work, largely due to this new technique.

Chapter 8

Conclusions and Future Work

In this work, we have investigated the robustness of domain-adaptation representations and methods for part-of-speech tagging and shown that there are large differences in robustness across target domains that need to be taken into account when performing domain adaptation for a target domain. We found that the divergence between source and target is an important predictor of what elements of domain adaptation will work; e.g., higher divergence makes it more likely that generalization mechanisms like dimensionality reduction will be beneficial.

In future work, we would like to develop statistical measures of source-target divergence that accurately predict whether a feature type or domain-adaptation technique supports high-performance domain adaptation for a particular target domain. Similar to Van Asch and Daelemans (2010), we want to solely use unlabeled data for estimating the performance of certain feature sets. One idea to do so is bootstrapping on features. We annotate data in the target domain with an initial model and then use this annotated data to evaluate the effectiveness of the features at hand. Finally, we use the features that proved to be most consistent with both domains. Another approach could apply an unsupervised tagger to both the source and target domain. Features that correlate with the classes chosen by the unsupervised tagger are then assumed to be good features for domain adaptation as well. The advantage to this approach is that it doesn't need labeled data in the source domain; hence, we can incorporate larger amounts of data into the training process.

Another interesting question to be examined is how hand-built feature sets compare to automatically designed features. As hand-built features are usually tailored specific datasets, we believe that finding features in an unsupervised way might be more effective in domain adaptation. Morphological features, for example, often contain a list of linguistically motivated suffixes and prefixes. Our shape features, in contrast, simply used all suffixes occurring in a corpus. Our unsupervised approach to leveraging orthographic information also showed promising results. Hence, we think it would be interesting to test the effectiveness of unsupervised features vs hand-encoded features in future experiments.

A last possible research direction could try to build on the concept of filtering. We found that excluding short words from the training set helps in some cases. Filtering is based on the observation that word length can also be an important clue for determining a word's POS tag (Cozens, 1998). For example, short words will have a higher probability for being a symbol, determiner or preposition. Most previous work used word length as an additional input feature, but it might well be the case that features change with word length as well. Non-alphanumeric characters, for instance, are usually more frequent with short words. In this case, it would be better to train separate classifiers for different word lengths or to have separate features for words of different lengths.

References

ACL Wiki (accessed March 7, 2013). Pos tagging (state of the art). Available at: http://aclweb.org/aclwiki/index.php?title=POS_Tagging_(State_of_the_art).

Baayen, H. and Sproat, R. (1996). Estimating lexical priors for low-frequency morphologically ambiguous forms. *Computational Linguistics*, 22(2):155–166.

Bandyopadhyay, S. (2012). *Emerging Applications of Natural Language Processing: Concepts and New Research*. IGI Global.

Ben-David, S., Blitzer, J., Crammer, K., and Sokolova, M. (2007). Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems 19*. MIT Press.

Bentley, J. L. (1980). Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229.

Bickel, S., Brückner, M., and Scheffer, T. (2007). Discriminative learning for differing training and test distributions. In *Proceedings of the 24th International Conference on Machine Learning*, pages 81–88.

Blitzer, J., Mcdonald, R., and Pereira, F. (2006). Domain adaptation with structural correspondence learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 120–128.

Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pages 92–100.

Cattell, R. B. (1966). The scree test for the number of factors. *Multivariate Behavioral Research*, 1(2):245–276.

Chan, Y. S. and Ng, H. T. (2005). Word sense disambiguation with distribution estimation. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1010–1015.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Charniak, E. (1997). Statistical techniques for natural language parsing. *AI Magazine*, 18:33–44.

Chen, M., Weinberger, K. Q., and Blitzer, J. (2011). Co-training for domain adaptation. In *Proceedings of Advances in Neural Information Processing Systems 24*, pages 1–9.

Choi, J. D. and Palmer, M. (2012). Fast and robust part-of-speech tagging using dynamic model selection. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, pages 363–367.

Cholakov, K., Noord, G. v., Kordoni, V., and Zhang, Y. (2011). An empirical comparison of unknown word prediction methods. In *Proceedings of the 5th International Joint Conference on Natural Language Processing*, pages 767–775.

Cozens, S. (1998). Primitive part-of-speech tagging using word length and sentential structure. *arXiv preprint cmp-lg/9808011*.

Crammer, K. and Singer, Y. (2002). On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292.

Cucerzan, S. and Yarowsky, D. (2000). Language independent, minimally supervised induction of lexical probabilities. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 270–277.

Das, D. and Petrov, S. (2011). Unsupervised part-of-speech tagging with bilingual graph-based projections. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 600–609.

Daumé III, H. (2007). Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 256–263.

Daumé III, H., Kumar, A., and Saha, A. (2010). Frustratingly easy semi-supervised domain adaptation. In *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, pages 53–59.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Dermatas, E. and Kokkinakis, G. (1995). Automatic stochastic tagging of natural language texts. *Computational Linguistics*, 21(2):137–163.

DeRose, S. J. (1988). Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14(1):31–39.

Everitt, B. and Hothorn, T. (2011). *Introduction to applied multivariate analysis with R.* Springer, Berlin.

Ganchev, K., Hall, K., McDonald, R., and Petrov, S. (2012). Using search-logs to improve query tagging. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, pages 238–242.

Garside, R. (1987). The CLAWS word-tagging system. In Garside, R., Leech, G., and Sampson, G., editors, *The Computational Analysis of English: A Corpus-Based Approach*, pages 30–41. Longman, London.

Giménez, J. and Màrquez, L. (2004). SVMTool: A general pos tagger generator based on support vector machines. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*, pages 43–46.

Goldwater, S. and Griffiths, T. (2007). A fully bayesian approach to unsupervised part-of-speech tagging. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 744–751.

Gorban, A., Kégl, B., and Wunsch, D. (2008). *Principal Manifolds for Data Visualization and Dimension Reduction*. Lecture Notes in Computational Science and Engineering. Springer London, Limited.

Hart, P. (1968). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(3):515–516.

Hastie, T., Tibshirani, R., and Friedman, J. (2008). *The elements of statistical learning: data mining, inference and prediction*. Springer, 2nd edition.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425.

Huang, F. and Yates, A. (2009). Distributional representations for handling sparsity in supervised sequence-labeling. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the Association of Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, pages 495–503.

Huang, F. and Yates, A. (2010). Exploring representation-learning approaches to domain adaptation. In *Proceedings of the Workshop on Domain Adaptation for Natural Language Processing*, pages 23–30.

Huang, F. and Yates, A. (2012). Biased representation learning for domain adaptation. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1313–1323.

Huang, Z., Eidelman, V., and Harper, M. (2009). Improving a simple bigram hmm part-of-speech tagger by latent annotation and self-training. In *Proceedings of the 10th Annual Meeting of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Short Papers*, pages 213–216.

Jiang, J. (2008). *Domain adaptation in natural language processing*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA. AAI3337811.

Jiang, J. and Zhai, C. (2007). Instance weighting for domain adaptation in NLP. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 264–271.

Jurafsky, D. and Martin, J. H. (2008). *Speech and Language Processing*. Pearson Prentice Hall, 2nd edition.

Kübler, S. and Baucom, E. (2011). Fast domain adaptation for part of speech tagging for dialogues. In Angelova, G., Bontcheva, K., Mitkov, R., and Nicolov, N., editors, *Recent Advances in Natural Language Processing*, pages 41–48.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The Penn treebank. *Computational Linguistics*, 19(2):313–330.

McCallum, A., Freitag, D., and Pereira, F. C. N. (2000). Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the 17th International Conference on Machine Learning*, pages 591–598.

McClosky, D., Charniak, E., and Johnson, M. (2006). Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 337–344.

Merialdo, B. (1994). Tagging english text with a probabilistic model. *Computational Linguistics*, 20(2):155–171.

Miller, J., Torii, M., and Shanker, V. K. (2007). Building domain-specific taggers without annotated (domain) data. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1103–1111.

Müller, T., Schütze, H., and Schmid, H. (2012). A comparative investigation of morphological language modeling for the languages of the european union. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 386–395.

Nakagawa, T., Kudoh, T., and Matsumoto, Y. (2001). Unknown word guessing and part-of-speech tagging using support vector machines. In *In Proceedings of the 6th Natural Language Processing Pacific Rim Symposium*, pages 325–331.

Okazaki, N. (2007). CRFsuite: A fast implementation of conditional random fields (CRFs). Available at: `http://www.chokkan.org/software/crfsuite/`.

Petrov, S. and McDonald, R. (2012). Overview of the 2012 Shared Task on Parsing the Web. Notes of the 1st Workshop on Syntactic Analysis of Non-Canonical Language.

Ringger, E., McClanahan, P., Haertel, R., Busby, G., Carmen, M., Carroll, J., Seppi, K., and Lonsdale, D. (2007). Active learning for part-of-speech tagging: accelerating corpus annotation. In *Proceedings of the Linguistic Annotation Workshop*, pages 101–108.

Russell, S. J. and Norvig, P. (2010). *Artificial Intelligence - A Modern Approach*. Pearson Education, 3rd edition.

Schmid, H. and Laws, F. (2008). Estimation of conditional probabilities with decision trees and an application to fine-grained pos tagging. In *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, pages 777–784.

Schütze, H. (1995). Distributional part-of-speech tagging. In *Proceedings of the 7th Conference on European Chapter of the Association for Computational Linguistics*, pages 141–148.

Shlens, J. (2005). A tutorial on principal component analysis. *Systems Neurobiology Laboratory, Salk Institute for Biological Studies.*

Smith, N. A. and Eisner, J. (2005). Contrastive estimation: training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 354–362.

Søgaard, A. (2011). Semisupervised condensed nearest neighbor for part-of-speech tagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short papers - Volume 2*, pages 48–52.

Subramanya, A., Petrov, S., and Pereira, F. (2010). Efficient graph-based semi-supervised learning of structured tagging models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 167–176.

Sutton, C. and McCallum, A. (2006). *An introduction to conditional random fields for relational learning.* Introduction to statistical relational learning. MIT Press.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, pages 173–180.

Tseng, H., Jurafsky, D., and Manning, C. (2005). Morphological features help POS tagging of unknown words across language varieties. In *Proceedings of the 4th Special Interest Group of the Association for Computational Linguistics Workshop on Chinese Language Processing*, pages 32–39.

Umansky-Pesin, S., Reichart, R., and Rappoport, A. (2010). A multi-domain web-based algorithm for POS tagging of unknown words. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 1274–1282.

Van Asch, V. and Daelemans, W. (2010). Using domain similarity for performance estimation. In *Proceedings of the Workshop on Domain Adaptation for Natural Language Processing*, pages 31–36.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260 –269.

Voutilainen, A. (1995). A syntax-based part-of-speech analyser. In *Proceedings of the 7th Conference on European Chapter of the Association for Computational Linguistics*, pages 157–164.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, pages 189–196.

Zhang, Y. and Kordoni, V. (2006). Automated deep lexical acquisition for robust open texts processing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, pages 275–280.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature