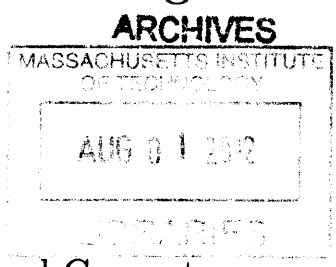
Ordering Prenominal Modifiers with a Ranking Approach



by

Jingyi Liu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2011

Author .....
Department of Electrical Engineering and Computer Science
August 2, 2011

Certified by .....
Regina Barzilay
Associate Professor, CSAIL
Thesis Supervisor

Accepted by .....
Dennis Freeman
Chairman, Masters of Engineering Thesis Committee

Ordering Prenominal Modifiers with a Ranking Approach

by

Jingyi Liu

Submitted to the Department of Electrical Engineering and Computer Science
on August 2, 2011, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis we present a solution to the natural language processing task of ordering prenominal modifiers, a problem that has applications from machine translation to natural language generation. In machine translation, constraints on modifier orderings vary from language to language so some reordering of modifiers may be necessary. In natural language generation, a representation of an object and its properties often needs to be formulated into a concrete noun phrase. We detail a novel approach that frames this task as a ranking problem amongst the permutations of a set of modifiers, admitting arbitrary features on each candidate permutation and exploiting hundreds of thousands of features in total. We compare our methods to a state-of-the-art class based ordering approach and a strong baseline that makes use of the Google n-gram corpus. We attain a maximum error reduction of 69.8% and average error reduction across all test sets of 59.1% compared to the state-of-the-art, and we attain a maximum error reduction of 68.4% and average error reduction across all test sets of 41.8% compared to our Google n-gram baseline. Finally, we present an analysis of our approach as compared to our baselines and describe several potential improvements to our system.

Thesis Supervisor: Regina Barzilay
Title: Associate Professor, CSAIL

Acknowledgments

I would like to thank Regina Barzilay, my thesis advisor, for her support and advice. I have really enjoyed working in her group these past few semesters. Many thanks to Aria Haghighi, who advised me on the modifier ordering project, and Roi Reichart, who advised me on the automatic summarization project. Their ideas have really kept me going and have taught me so much, and I'm really glad I had the opportunity to work with them. Thanks to Christy Sauper, my officemate, and the rest of the CSAIL NLP research group, for answering my multitude of questions and giving me many thoughtful comments on my ACL paper and presentation. Thanks to Margaret Mitchell for so patiently responding to all of my emails about her work. Thanks to my friends for being so supportive and keeping me laughing. Finally, many thanks to my parents for their advice and support. I couldn't have done it without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Speakers rarely have difficulty correctly ordering modifiers such as adjectives, adverbs, or gerunds when describing some noun. The phrase "beautiful blue Macedonian vase" sounds very natural, whereas changing the modifier ordering to "blue Macedonian beautiful vase" is awkward (see Table 1.1 for more examples). In this thesis we consider the task of ordering an unordered set of prenominal modifiers so that they sound fluent to native language speakers. This is a problem that has applications in natural language generation and machine translation. From the generation perspective, moving from a representation of an object and its properties to a concrete noun phrase requires proper ordering of the modifiers describing the object. In the translation setting, languages may have different constraints on modifier orderings (for example,

a. the vegetarian French lawyer ✓
b. the French vegetarian lawyer
a. the beautiful small black purse ✓
b. the beautiful black small purse
c. the small beautiful black purse
d. the small black beautiful purse

Table 1.1: Examples of restrictions on modifier orderings from Teodorescu (2006). The most natural sounding ordering is in bold, followed by other possibilities that may only be appropriate in certain situations.

quality > *size* > *shape* > *color* > *provenance*
(Sproat and Shih 1991)
age > *color* > *participle* > *provenance* > *noun* > *denominal*
(Quirk 1974)
value > *dimension* > *physical property* > *speed* > *human propensity* > *age* > *color*
(Dixon 1977)

Table 1.2: Examples of orderings on semantic groups, where > denotes precedence.

in French, adjectives may come before and after a noun while in English this is not the case), and some reordering may need to be done during the translation process.

Much linguistic research has investigated the semantic constraints behind prenominal modifier orderings. One common line of research suggests that modifiers can be grouped by the underlying semantic property they describe and that there is an ordering on semantic properties which in turn restricts modifier orderings. For instance, Sproat and Shih (1991) contend that the *size* property precedes the *color* property and thus "small black cat" sounds more fluent than "black small cat". See Table 1.2 for examples of full-length ordering constraints on semantic groups.

However, generating a list of semantic groups is a difficult task, and classifying modifiers into these groups is challenging as well and may be domain dependent or constrained by the context in which the modifier is being used. In addition, these methods do not specify how to order modifiers within the same group or modifiers that do not fit into any of the specified groups.

There have also been a variety of corpus-based, computational approaches. Mitchell (2009) uses a class-based approach in which modifiers are grouped into classes based on which positions they prefer in the training corpus, with a predefined ordering imposed on these classes. Shaw and Hatzivassiloglou (1999) developed three different approaches to the problem that use n-gram counting methods and clustering algorithms, and Malouf (2009) expands upon Shaw and Hatzivassiloglou's work. Dunlop, Mitchell, and Roark (2010) use a feature-based multiple sequence alignment approach.

In this thesis we describe a computational solution that views the modifier ordering

task as a ranking problem. All the permutations of a set of modifiers are enumerated and put in competition with each other, with a set of features used to determine a score for each candidate permutation. At training time a set of feature weights is learned such that the correct sequences in the training data receive the highest scores, and at testing time the candidate permutation with the highest score is chosen as the correct ordering. This approach has not been used before to solve the prenominal modifier ordering problem, and as we shall demonstrate, vastly outperforms the state-of-the-art, especially for sequences of longer lengths.

Chapter 2 describes previous approaches to the problem. Chapter 3 explains the details of our maximum entropy ranking approach. Chapter 4 covers the evaluation methods used and presents our results. Chapter 5 gives an analysis of the ranking approach and a comparison to previous methods, and Chapter 6 discusses the strengths and weaknesses of the approach as well as potential improvements to the system.

Chapter 2

Past Work

In the following chapter we survey past approaches to the modifier ordering problem, all of which are supervised.

2.1 n-gram Counting (Shaw and Hatzivassiloglou 1999)

Shaw and Hatzivassiloglou use n-gram counting methods. For a training corpus with w word types, they fill out a $w \times w$ matrix, COUNT, where COUNT$[A, B]$ indicates how often modifier type A precedes modifier type B. Given two modifiers a and b to order, they look at COUNT$[a, b]$ and COUNT$[b, a]$ in their training data. Assuming a null hypothesis that the probability of either ordering is 0.5, they use a binomial distribution to compute the probability of seeing the ordering $\langle a,\ b\rangle$ for COUNT$[a, b]$ number of times. If this probability is above a certain threshold then they say that a precedes b.

Because the bigram counts from the training corpus are extremely sparse, Shaw and Hatzivassiloglou also use a transitivity method to fill out empty parts of the COUNT table if their counts can be inferred from other entries, and they use a clustering method to group together modifiers with similar positional preferences.

2.2 Multiple Sequence Alignment (Dunlop et al. 2010)

Dunlop et al. employ multiple sequence alignment (MSA) techniques to the modifier ordering problem. MSA is typically used to order large numbers of DNA sequences so that as much overlap as possible is attained amongst the sequences, allowing for substitutions and insertions of new columns when this helps achieve more overlap. Generally, there is a predefined cost for substituting one token for another and a different cost for inserting a new column into the alignment. Dunlop et al. apply MSA to the ordering problem by treating modifiers as tokens. Given a set of modifiers to order from the test data, they generate all permutations of the set and pick the one that aligns with lowest cost to all of the training data via MSA. Because modifiers vary much more than nucleotides, they define a feature based cost function, which pushes the MSA towards aligning tokens with similar features into the same column. Example features used in their system are the value of the modifier itself, whether the modifier contains a numeral, whether the modifier is capitalized, and whether the modifier is hyphenated.

2.3 Positional Preference Classes (Mitchell 2009)

	"exquisite	blue	handcrafted	Macedonian	vase"
Position	4	3	2	1	

Table 2.1: Our notation for referring to modifier positions.

Because we will use Mitchell's 2009 work *Class-Based Ordering of Prenominal Modifiers* as a baseline for our system, we will describe it in more detail than the other previous work. It is worth introducing some notation, which will be used throughout the rest of this thesis. As depicted in Table 2.1, we will refer to position 1

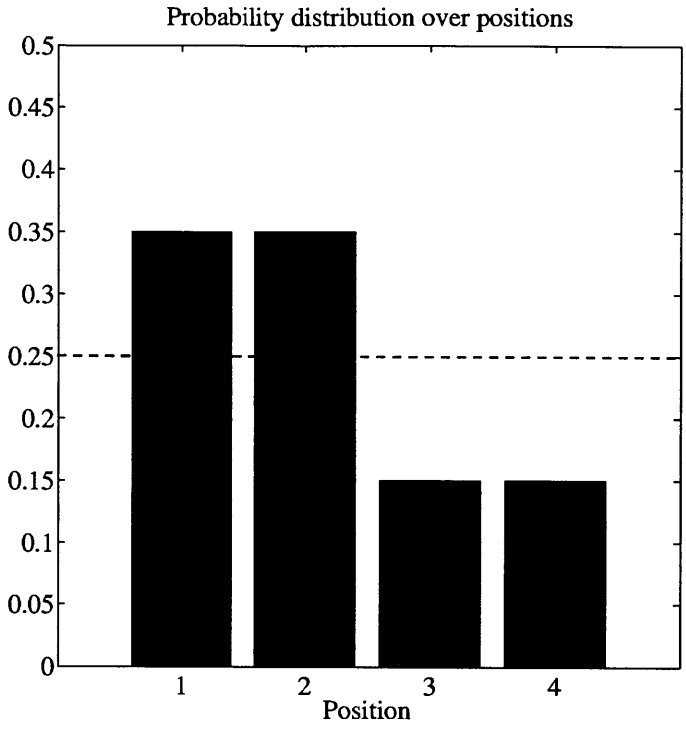


Figure 2-1: An example distribution across positions for a modifier. Because at training time this modifier is seen in positions 1 and 2 more than a uniform distribution would allow, it is assigned to Class 1-2, which comprises those modifiers that prefer to be in either position 1 or 2.

in a modifier sequence as that closest to the head noun, position 2 as the next closest, and so on. In addition, modifier generation starts from the right and moves to the left, so for the sequence in Table 2.1, we say that "Macedonian" is generated before "handcrafted", which is generated before "blue", and so on.

Mitchell orders sequences of at most 4 modifiers, and she defines nine classes that express the broad positional preferences of modifiers. The first four classes, Class 1, Class 2, Class 3, and Class 4, comprise those modifiers that prefer only to be in positions 1 through 4, respectively. Then there are classes that are more flexible, Class 1-2, Class 2-3, and Class 3-4, which prefer positions 1-2, 2-3, and 3-4, respectively. Finally, Class 1-2-3 prefers positions 1-3 and Class 2-3-4 prefers positions 2-4. Mitchell counts how often each word type appears in each of positions

.	.	.
.	.	.
.	.	.
1	→	2
2	→	3
3	→	4
1-2	→	2-3
2-3	→	3-4
1-2-3	→	2-3-4
2	→	2-3-4
.	.	.
.	.	.
.	.	.

Table 2.2: Some example rules in Mitchell's system for ordering modifiers of different classes, where → means "generated before".

1 through 4 in the training corpus. If any word type's probability of taking a certain position is greater than a uniform distribution would allow, then it is said to prefer that position, and each word type is then assigned a class. For example, if a modifier has the distribution over positions pictured in Figure 2-1, then it would be assigned to Class 1-2, those modifiers that prefer to be in positions 1 or 2.

Given a set of modifiers to order, if each modifier in the set has been seen at training time, Mitchell's system looks up the class of each modifier and then orders the sequence based on a set of rules, with a list of some example rules in Table 2.2. Some straightforward rules are that modifiers of Class 1 are generated before those of Class 2, and modifiers of Class 2 are generated before those of Class 3. When comparing classes that are more flexible, the Class with the lower average preferred position is generated first, so modifiers of Class 1-2 are generated before those of Class 2-3, and Modifiers of Class 2 are generated before those of Class 2-3-4. When two or more modifiers have the same class then multiple permutations satisfy the ordering rules, and in these cases the system picks between the possibilities randomly. If a

modifier was not seen at training time and thus cannot be said to belong to a specific class, the system favors orderings where modifiers whose classes are known are as close to their classes' preferred positions as possible.

2.4 Limitations of Past Work

These methods work fairly well, but they also suffer from sparsity issues in the training data. Mitchell reports a prediction accuracy of 78.59% for NPs of all lengths, but the accuracy of her approach is greatly reduced when two modifiers fall into the same class, since the system cannot make an informed decision in those cases. In addition, if a modifier is not seen in the training data, the system is unable to assign it a class, which limits accuracy. Shaw and Hatzivassiloglou report a highest accuracy of 94.93% and a lowest accuracy of 65.93%, but since their methods depend heavily on bigram counts in the training corpus, they are also limited in how informed their decisions can be if modifiers in the test data are not present at training time. In this next section, we describe our maximum entropy ranking approach that tries to develop a more comprehensive model of the modifier ordering process to avoid the sparsity issues that previous approaches have faced.

Chapter 3

Maximum Entropy Ranking

3.1 Motivation

The principle of maximum entropy and feature selection has been applied to various problems in NLP, from sentence boundary detection (Reynar and Ratnaparkhi 1997) to prepositional phrase attachment (Ratnaparkhi et al. 1994). The approach involves finding suitable features to model a linguistic process and then determining optimal weights for the features. In Reynar's sentence boundary detection work, each ".", "!", and "?" in the text is a candidate for a sentence boundary, and a set of relevant features on the context of each candidate is created. If the maximum entropy model determines that the probability a given candidate is a sentence boundary is greater than 0.5, then that candidate is assigned to be a boundary location. In Ratnaparkhi's PP attachment work, various candidate PP attachments are scored based on a set of important features, with the highest scoring attachment chosen as the winning one. We will use a similar approach as Ratnaparkhi's PP attachment work by scoring the permutations of a set of modifiers based on a set of relevant features. The next section describes our model and features used in more detail.

3.2 Model

We treat the problem of prenominal modifier ordering as a ranking problem. Given a set B of prenominal modifiers and a noun phrase head H which B modifies, we define $\pi(B)$ to be the set of all possible permutations, or orderings, of B. We suppose that for a set B there is some $x^* \in \pi(B)$ which represents a "correct" natural-sounding ordering of the modifiers in B.

At test time, we choose an ordering $x \in \pi(B)$ using a maximum entropy ranking approach as described in Collins and Koo (2005). Our distribution over orderings $x \in \pi(B)$ is given by:

$$P(x|H,B,W) = \frac{\exp\{W^T\phi(B,H,x)\}}{\sum_{x' \in \pi(B)} \exp\{W^T\phi(B,H,x')\}}$$

where $\phi(B,H,x)$ is a feature vector over a particular ordering of B and W is a learned weight vector over features. We describe the set of features in Section 3.3, but note that we are free under this formulation to use arbitrary features on the full ordering x of B as well as the head noun H, which we implicitly condition on throughout. Since the size of the set of prenominal modifiers B is typically less than six, enumerating $\pi(B)$ is not expensive.

At training time, our data consists of sequences of prenominal orderings and their corresponding nominal heads. We treat each sequence as a training example where the labeled ordering $x^* \in \pi(B)$ is the one we observe. This allows us to extract any number of 'labeled' examples from part-of-speech text. Concretely, at training time, we select W to maximize:

$$\mathcal{L}(W) = \left(\prod_{(B,H,x^*)} P(x^*|H,B,W) \right) - \frac{\|W\|^2}{2\sigma^2}$$

where the first term represents our observed data likelihood and the second the ℓ_2 regularization, where σ^2 is a fixed hyperparameter; we fix the value of σ^2 to 0.5 throughout. We optimize this objective using standard L-BFGS optimization tech-

niques.

The key to the success of our approach is using the flexibility afforded by having arbitrary features $\phi(B, H, x)$ to capture all the salient elements of the prenominal ordering data. These features can be used to create a richer model of the modifier ordering process than previous corpus-based counting approaches. In addition, we can encapsulate previous approaches in terms of features in our model. Mitchell's class-based approach can be expressed as a feature that tells us how well a given permuation satisfies the class ordering constraints in her model. Previous counting approaches can be expressed as a real-valued feature that, given all n-grams generated by a permutation of modifiers, returns the count of all these n-grams in the original training data.

See Figure 3-1 for a a high level overview of what happens at test time.

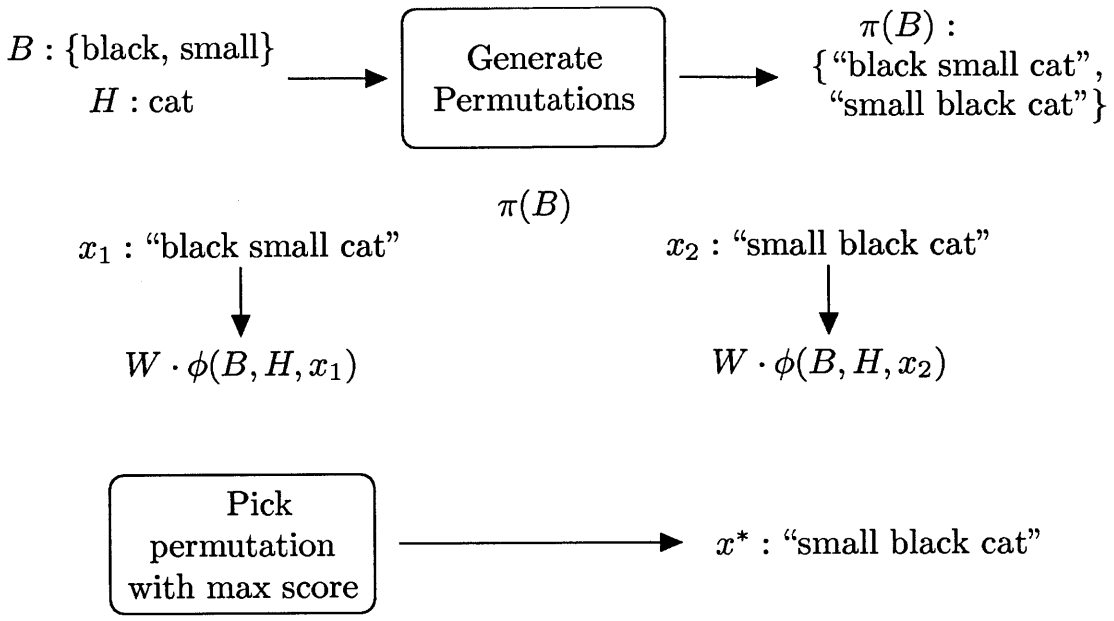


Figure 3-1: An overview of the testing time process. Given a set of modifiers, we generate all possible permutations of the set. Then we calculate a score for each candidate permutation, which depends on the feature vector $\phi(B, H, x)$ and the optimal feature weights W learned at test time. The permutation with the highest score is chosen as our x^*.

3.3 Features

Our features are of the form $\phi(B, H, x)$ as expressed in the model above, and we include both indicator features and real-valued numeric features in our model. We attempt to capture aspects of the modifier permutations that may be significant in the ordering process. We will describe a few of our most significant features. Table 3.1 contains the full list of features used in our model.

3.3.1 n-gram Count Feature

Very simply, we expect permutations that contain n-grams previously seen in the training data to be more natural sounding than other permutations that generate n-grams that have not been seen before. We can express this as a real-valued feature:

$$\phi(B, H, x) = \begin{cases} \text{count in training data of} \\ \text{all } n\text{-grams present in } x \end{cases}$$

We can create a separate feature of this form for bigrams, 3-grams, 4-grams, and 5-grams. See Figure 3-2 for an example of the bigram feature applied to a candidate permutation.

3.3.2 Head Noun and Closest Modifier Count Feature

The n-gram counts from the training data are extremely sparse, so we can supplement these with other counts from our training data. We can define a real-valued feature that extracts the head noun and its closest modifier from a candidate permutation and counts how often this bigram shows up in the training data. This feature is expressed as follows:

$$\phi(B, H, x) = \begin{cases} \text{count in training data of} \\ \langle M,\ H \rangle \text{ where } M \text{ is the} \\ \text{closest modifier to } H \end{cases}$$

See Figure 3-3 for an example of this feature applied to a candidate permutation.

3.3.3 Word Value Feature

We expect certain modifiers to prefer specific positions in an ordered sequence, and we can try to capture these preferences at a finer grain than the class based model does. We define an indicator feature that asks if a given word w appears in position i of a candidate permutation, as follows:

$$\phi(B, H, x) = \begin{cases} 1 & \text{if the modifier in position } i \text{ of ordering } x \text{ is the word } w \\ 0 & \text{otherwise} \end{cases}$$

We create a feature of this form for every word type w present in our training corpus and for positions 1 through 4. As an example, the feature for w = Macedonian and $i = 1$ applied to "exquisite blue Macedonian vase" has value 1 and applied to "blue Macedonian exquisite vase" has value 0.

3.3.4 Word Ending Feature

Similar to the n-gram count feature, the word value feature suffers from sparsity issues, and we attempt to supplement this by creating a similar feature that looks at word endings as opposed to values. From a linguistics perspective this makes sense as well – for example, perhaps the majority of words that end with *-ly* are adverbs and should usually be positioned farthest from the head noun, so a word ending feature that asks if the word in a given position of a candidate permutation ends in *-ly* should be useful. We can define this indicator feature as follows:

$$\phi(B, H, x) = \begin{cases} 1 & \text{if the modifier in position } i \text{ of ordering } x \text{ ends in } e \\ 0 & \text{otherwise} \end{cases}$$

The word ending e is drawn from the following list: {*-al -ble -ed -er -est -ian -ic -ing -ive -ly*}. We create a feature of this form for every e and every possible modifier position i from 1 to 4. As an example, the feature for e = *-ian* and $i = 1$ applied to "exquisite blue Macedonian vase" has value 1 and applied to "blue Macedonian exquisite vase" has value 0.

3.3.5 Mitchell Model Feature

We can also encapsulate previous approaches as features in our model. One real-valued feature we use is based on the Mitchell class based model. At training time, we can determine the class of every modifier present in our training data. Then we can look at all bigrams present in a candidate permutation and add a point to our feature for every bigram whose class ordering follows the rules of the class based model. We can express this feature as follows:

$$\phi(B, H, x) = \begin{cases} \text{number of bigrams in} \\ x \text{ that follow the class} \\ \text{based ordering rules} \end{cases}$$

See Figure 3-4 for an example of this feature applied to a candidate permutation.

BIGRAM COUNT FEATURE

$\phi_{bigram}(B, H, x) =$
how often do all bigrams present
in x appear in our training data

"exquisite blue Macedonian vase"

⟨exquisite, blue⟩ $\longrightarrow$ 14

⟨exquisite, Macedonian⟩ $\longrightarrow$ 3

⟨blue, Macedonian⟩ $\longrightarrow$ 4

Feature Value = 14 + 3 + 4 = 21

Training Data Statistics

⟨ancient stone⟩	12
⟨blue Macedonian⟩	4
.	.
.	.
.	.
⟨exquisite blue⟩	14
⟨exquisite Macedonian⟩	3
.	.
.	.
.	.
⟨zany spontaneous⟩	1

Figure 3-2: An example of the bigram count feature applied to candidate permutation "exquisite blue Macedonian vase". At training time we collect statistics on how often each bigram type is present in the training data. To find the value of this feature at testing time, we take all bigrams present in the permutation, allowing for skips (so ⟨exquisite Macedonian⟩ is included) and sum up their counts from our statistics.

CLOSEST MODIFIER AND HEAD NOUN COUNT FEATURE

$\phi_{headnoun}(B, H, x) =$
how often do head noun and closest modifier of x appear in training data together

"exquisite blue Macedonian vase"

⟨Macedonian vase⟩ $\longrightarrow$ 5

Feature Value = 5

Training Data Statistics

⟨ancient building⟩	10
⟨blue ocean⟩	25
.	.
.	.
.	.
⟨Macedonian vase⟩	5
⟨Macedonian village⟩	7
.	.
.	.
.	.
⟨zealous woman⟩	9

Figure 3-3: An example of the closest modifier and head noun count feature. At training time we collect statistics on how often each head noun and the modifier closest to it are present in the training data. The value of the feature at testing time is simply a single lookup into our table of statistics, which for candidate permutation "exquisite blue Macedonian vase" is the count of ⟨Macedonian vase⟩.

CLASS BASED FEATURE

$\phi_{classbased}(B, H, x) =$
how many bigrams in x satisfy the
CLASS BASED ordering constraints

"exquisite blue Macedonian vase"

exquisite $\longrightarrow$ Class 3

Macedonian $\longrightarrow$ Class 2

blue $\longrightarrow$ Class 2

$\langle$exquisite, blue$\rangle + 1$

$\langle$exquisite, Macedonian$\rangle + 1$

$\langle$blue, Macedonian$\rangle + 0$

Feature Value = 1 + 1 + 0 = 2

Ordering Rules

.	.	.
.	.	.
.	.	.
1	→	2
2	→	3
3	→	4
1-2	→	2-3
2-3	→	3-4
1-2-3	→	2-3-4
2	→	2-3-4
.	.	.
.	.	.
.	.	.

Figure 3-4: An example of the class based feature. At training time, we run the Mitchell model to generate class assignments for all modifiers in the training data. Given a candidate permutation, we can generate all its bigrams, allowing for skips, and then count how many of these bigrams follow the ordering rules. For example, for "exquisite blue Macedonian vase", ⟨exquisite, blue⟩ and ⟨exquisite, Macedonian⟩ satisfy the rule that Class 2 is generated before Class 3 so we add a point to the feature value for each bigram. However, the third bigram ⟨blue, Macedonian⟩ contains two modifiers of the same class, so we do not add anything to the feature value for this bigram.

Numeric Features	
n-gram Count	If N is the set of all n-grams present in the permutation, returns the sum of the counts of each element of N in the training data. A separate feature is created for 2-gms through 5-gms.
Count of Head Noun and Closest Modifier	Returns the count of $< M, H >$ in the training data where H is the head noun and M is the modifier closest to H.
Length of Modifier*	Returns the length of modifier in position i
Indicator Features	
Hyphenated*	Modifier in position i contains a hyphen.
Is Word w*	Modifier in position i is word $w \in W$, where W is the set of all word types in the training data.
Ends In e*	Modifier in position i ends in suffix $e \in E$, where $E = \{$*-al -ble -ed -er -est -ian -ic -ing -ive -ly*$\}$
Is A Color*	Modifier in position i is a color, where we use a list of common colors
Starts With a Number*	Modifier in position i starts with a number
Is a Number*	Modifier in position i is a number
Satisfies Mitchell Class Ordering	The permutation's class ordering satisfies the Mitchell class ordering constraints

Table 3.1: Summary of features used in our model. Features with an asterisk (*) are created for all possible modifier positions i from 1 to 4.

Chapter 4

Experiments

4.1 Dataset

We extracted all noun phrases from four corpora: the Brown, Switchboard, and Wall Street Journal corpora from the Penn Treebank, and the North American Newswire corpus (NANC). Since there were very few noun phrases (NPs) with more than five modifiers, we kept those with 2-5 modifiers and with tags *NN* or *NNS* for the head noun. We also kept NPs with only one modifier to be used for generating ⟨*modifier, head noun*⟩ bigram counts at training time. We then filtered all these NPs as follows: If the NP contained a *PRP*, *IN*, *CD*, or *DT* tag and the corresponding modifier was farthest away from the head noun, we removed this modifier and kept the rest of the NP. If the modifier was not the farthest away from the head noun, we discarded the NP. If the NP contained a *POS* tag we only kept the part of the phrase up to this tag. Our final set of NPs had tags from the following list: *JJ*, *NN*, *NNP*, *NNS*, *JJS*, *JJR*, *VBG*, *VBN*, *RB*, *NNPS*, *RBS*. See Table 4.1 for a summary of the number of NPs of lengths 1-5 extracted from the four corpora. Note that we had a large number of sequences in our data because of the NANC corpus, but most of these sequences contained two or three modifiers, with very few sequences containing four or five.

Our system makes several passes over the data during the training process. In the first pass, we collect statistics about the data, to be used later on when calculating our numeric features. To collect the statistics, we take each NP in the training

Number of Sequences by Token

	1	2	3	4	5	Total
Brown	11,265	1,398	92	8	2	12,765
WSJ	36,313	9,073	1,399	229	156	47,170
Switchboard	10,325	1,170	114	4	1	11,614
NANC	15,456,670	3,399,882	543,894	80,447	14,840	19,495,733

Number of Sequences by Type

	1	2	3	4	5	Total
Brown	4,071	1,336	91	8	2	5,508
WSJ	7,177	6,687	1,205	182	42	15,293
Switchboard	2,122	950	113	4	1	3,190
NANC	241,965	876,144	264,503	48,060	8,451	1,439,123

Table 4.1: Number of NP sequences extracted from our corpora, sorted by the number of modifiers in the NP.

data and consider all possible 2-gms through 5-gms that are present in the NP's modifier sequence, allowing for non-consecutive n-grams. For example, the NP "the beautiful blue Macedonian vase" generates the following bigrams: $\langle$*beautiful blue*$\rangle$, $\langle$*blue Macedonian*$\rangle$, and $\langle$*beautiful Macedonian*$\rangle$, along with the 3-gram $\langle$*beautiful blue Macedonian*$\rangle$. We keep a table mapping each unique n-gram to the number of times it has been seen in the training data. In addition, we also store a table that keeps track of bigram counts for $\langle M, H \rangle$, where H is the head noun of an NP and M is the modifier closest to it. For the example "the beautiful blue Macedonian vase", we would increment the count of $\langle$*Macedonian vase*$\rangle$ in the table. These $\langle M, H \rangle$ counts are also used to compute numeric feature values.

4.2 Baselines

4.2.1 Google n-gram Baseline

The Google n-gram corpus is a collection of n-gram counts drawn from public web-pages with a total of one trillion tokens, around 1 billion each of unique 3-grams, 4-grams, and 5-grams, and around 300 million unique bigrams. We created a Google n-gram baseline that takes a set of modifiers B, determines the count in the Google n-gram corpus for each possible permutation in $\pi(B)$, and selects the permutation with the highest count as the winning ordering x^*. We will refer to this baseline as GOOGLE N-GRAM.

4.2.2 Mitchell's Class-Based Ordering of Prenominal Modifiers (2009)

Mitchell's original system was evaluated using only three corpora for both training and testing data: Brown, Switchboard, and WSJ. In addition, the evaluation presented by Mitchell's work considers a prediction to be correct if the ordering of classes in that prediction is the same as the ordering of classes in the original sequence from the test data, where a class refers to the positional preference groupings defined in the model. We use a more stringent evaluation as described in the next section. We implemented our own version of Mitchell's system that duplicates the model and methods but allows us to scale up to a larger training set and to apply our own evaluation techniques. We will refer to this baseline as CLASS BASED.

4.3 Experiments

We ran four experiments where we held out part of our data and trained on the rest of it. The four test sets we used were the Brown corpus, the Switchboard corpus, the WSJ corpus, and a randomly chosen tenth of the NANC corpus. For each experiment we recorded the prediction accuracies by token and by type, and we considered the

prediction to be correct only if it exactly matched the original sequence present in the test data.

4.4 Results

When considering all sequence lengths, MAXENT consistently outperforms CLASS BASED across all test corpora and sequence lengths for both tokens and types, and MAXENT also outperforms GOOGLE N-GRAM across all test corpora and sequence lengths except when testing on the Switchboard corpus. MAXENT does a few percentage points better when testing on the WSJ and NANC corpora than when testing on Switchboard or Brown (see Figure 4-1). We suspect that the content of the corpora may have an effect on the numbers – Switchboard is composed of transcribed conversations, Brown is a mix of literature and newswire, and NANC and WSJ are solely newswire. Because NANC is such a big corpus, it heavily biases our training data in the newswire direction, resulting in higher numbers when testing on WSJ or testing on the held out section of NANC. In addition, GOOGLE N-GRAM may be performing better then MAXENT on the Switchboard corpus because the web tends to mirror conversational text more than our training corpora do.

However, Figure 4-1 does not tell the complete story, as the accuracies decrease very quickly for longer sequences as can be seen in Figure 4-2, which shows prediction accuracies when testing on the held out portion of NANC by n-gram length. However, the accuracies for MAXENT have the smallest decrease as the sequence length increases.

If we examine the error reduction between MAXENT and CLASS BASED, we attain a maximum error reduction of 69.8% for the WSJ test corpus for sequence tokens, and an average error reduction of 59.1% across all test corpora for tokens. MAXENT also attains a maximum error reduction of 68.4% for the WSJ test corpus and an average error reduction of 41.8% when compared to GOOGLE N-GRAM.

It should be noted that on average the MAXENT model takes three hours to train with several hundred thousand features mapped across the training data (the exact

number used during each test run is listed in Table 4.2) – this tradeoff is well worth the increase we attain in system performance.

Our complete results by token and type are listed in Tables 4.2 and 4.3.

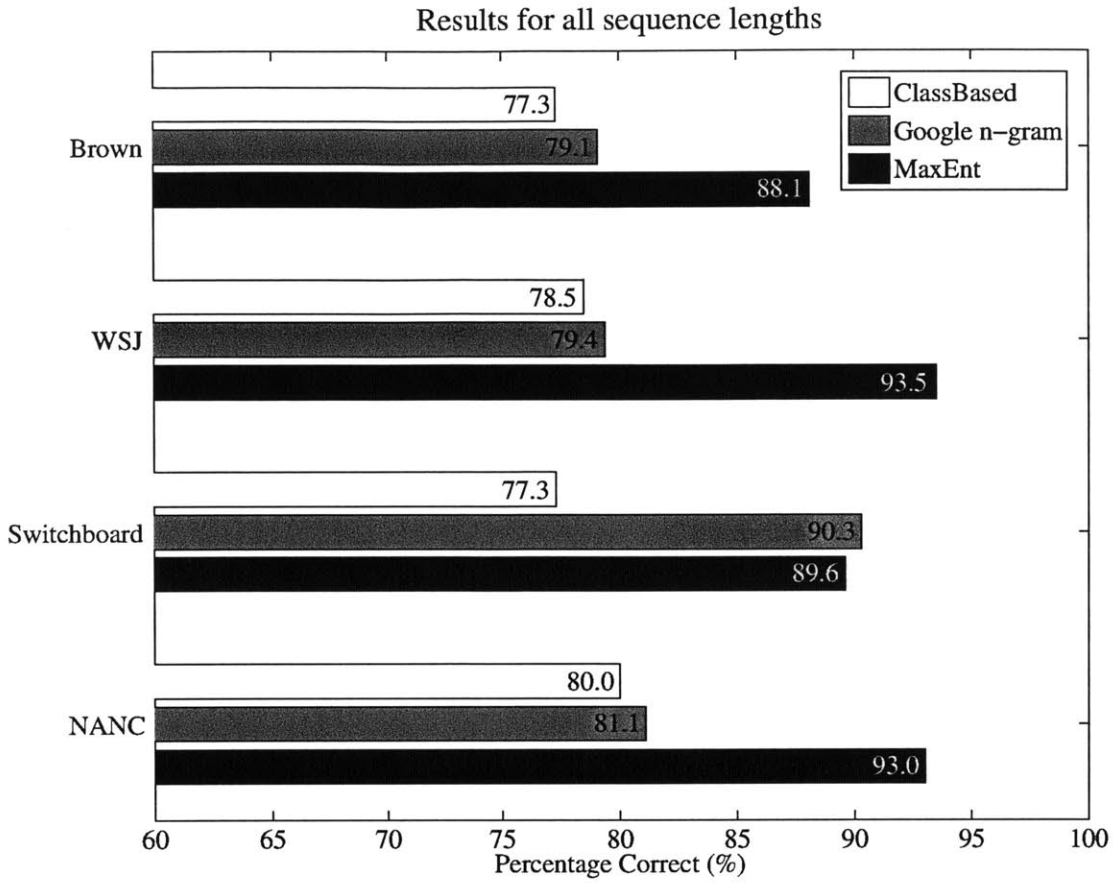


Figure 4-1: Results for sequence tokens of all lengths. Our data consisted of the NANC, Brown, WSJ, and Switchboard corpora. We used each corpus as test data (with the exception of NANC, which we randomly selected a tenth of to use for testing) and trained on everything else, and the table shows the results for each approach on each test corpus.

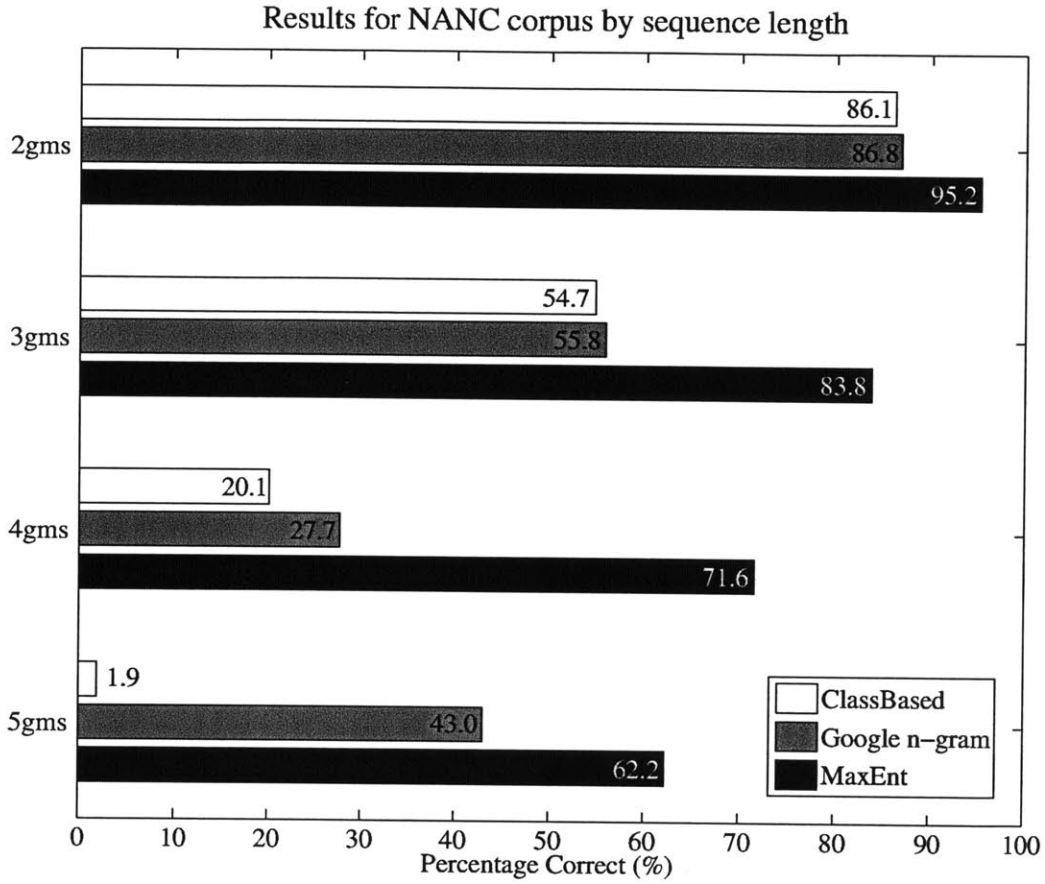


Figure 4-2: Results of testing on a tenth of the NANC corpus and training on everything else, broken down by sequence length. While accuracy quickly degraded as sequence length increased, the MAXENT approach saw the smallest decrease in accuracy.

Test Corpus		Token Accuracy (%)				
		2	**3**	**4**	**5**	**Total**
Brown	CLASS BASED	79.3	54.3	25.0	0	77.3
	GOOGLE N-GRAM	82.4	35.9	12.5	0	79.1
	MAXENT	**89.4**	**70.7**	**87.5**	0	**88.1**
WSJ	CLASS BASED	85.5	51.6	16.6	0.6	78.5
	GOOGLE N-GRAM	84.8	53.5	31.4	71.8	79.4
	MAXENT	**95.9**	**84.1**	**71.2**	**80.1**	**93.5**
Switchboard	CLASS BASED	80.1	52.6	0	0	77.3
	GOOGLE N-GRAM	**92.8**	68.4	0	0	**90.3**
	MAXENT	91.4	**74.6**	**25.0**	0	89.6
One Tenth ofNANC	CLASS BASED	86.1	54.7	20.1	1.9	80.0
	GOOGLE N-GRAM	86.8	55.8	27.7	43.0	81.1
	MAXENT	**95.2**	**83.8**	**71.6**	**62.2**	**93.0**

Test Corpus	Number of Features Used In MaxEnt Model
Brown	655,536
WSJ	654,473
Switchboard	655,791
NANC	565,905

Table 4.2: Token prediction accuracies for the GOOGLE N-GRAM, MAXENT, and CLASS BASED approaches for modifier sequences of lengths 2-5. Our data consisted of four corpora: Brown, Switchboard, WSJ, and NANC. The test data was held out and each approach was trained on the rest of the data. Winning scores are in bold. The number of features used during training for the MAXENT approach for each test corpus is also listed.

Test Corpus		Type Accuracy (%)				
		2	3	4	5	Total
Brown	CLASS BASED	78.9	54.9	25.0	0	77.0
	GOOGLE N-GRAM	81.8	36.3	12.5	0	78.4
	MAXENT	**89.1**	**70.3**	**87.5**	0	**87.8**
WSJ	CLASS BASED	85.1	50.1	19.2	0	78.0
	GOOGLE N-GRAM	82.6	49.7	23.1	16.7	76.0
	MAXENT	**94.7**	**81.9**	**70.3**	**45.2**	**92.0**
Switchboard	CLASS BASED	79.1	53.1	0	0	75.9
	GOOGLE N-GRAM	**91.7**	68.1	0	0	**88.8**
	MAXENT	90.3	**75.2**	**25.0**	0	88.4
One Tenth of NANC	CLASS BASED	80.3	51.0	18.4	3.3	74.5
	GOOGLE N-GRAM	79.2	44.6	20.5	12.3	70.4
	MAXENT	**91.6**	**78.8**	**63.8**	**44.4**	**88.0**

Test Corpus	Number of Features Used In MaxEnt Model
Brown	655,536
WSJ	654,473
Switchboard	655,791
NANC	565,905

Table 4.3: Type prediction accuracies for the GOOGLE N-GRAM, MAXENT, and CLASS BASED approaches for modifier sequences of lengths 2-5. Our data consisted of four corpora: Brown, Switchboard, WSJ, and NANC. The test data was held out and each approach was trained on the rest of the data. Winning scores are in bold. The number of features used during training for the MAXENT approach for each test corpus is also listed.

Chapter 5

Analysis

MAXENT outperforms the CLASS BASED baseline because it learns more from the training data. The CLASS BASED model classifies each modifier in the training data into one of nine broad classes, with each class representing a different set of positional preferences. However, because there are only nine classes, many of the modifiers in the training data get classified to the same class. At testing time, if an unordered set contains two or more modifiers of the same class then there is more than one candidate permutation that satisifes the ordering constraints of the model, so CLASS BASED is forced to make a random choice. When applying CLASS BASED to WSJ as the test data and training on the other corpora, 74.7% of the incorrect predictions contained at least 2 modifiers that were of the same positional preferences class. In contrast, MAXENT allows us to learn much more from the training data because it is not limited by a fixed number of classes, and as a result, we see much higher numbers when trained and tested on the same data as CLASS BASED.

The GOOGLE N-GRAM method does better than the CLASS BASED approach because it contains n-gram counts for more data than the WSJ, Brown, Switchboard, and NANC corpora combined. However, GOOGLE N-GRAM suffers from sparsity issues as well when testing on less common modifier combinations. For example, our data contains rarely heard sequences such as "Italian state-owned holding company" or "armed Namibian nationalist guerrillas." While MAXENT determines the correct ordering for both of these examples, none of the permutations of either example

Top Value Weights	
4-gram count in training data	9.13
...	...
3-gram count in training data	4.15
...	...
5-gram count in training data	3.43
...	...
does the permutation satisfy CLASS BASED ordering constraints	1.80
...	...
2-gram count in training data	1.53
...	...
does modifier farthest from head end in *-ly*	0.99
...	...
does modifier farthest from head end in *-ble*	0.88
...	...
does modifier farthest from head end in *-est*	0.71
...	...
count of ⟨M, H⟩ in training data where H is the head and M is the modifier closest to it	0.63

Table 5.1: Some of the highest value weights of the MAXENT model when trained on the NANC, Brown, and Switchboard corpora.

show up in the Google n-gram corpus, so GOOGLE N-GRAM is forced to randomly select from the six possibilities. In addition, the Google n-gram corpus is composed of sentence fragments that may not necessarily be NPs, so its counts for modifier permutations that can function as not just NPs but other parts of a sentence may be inflated.

5.1 Learned Weights

Table 5.1 shows some of the highest value learned weights in the MAXENT model when trained on the NANC, Brown, and Switchboard corpora. The table shows only a sampling of the highest value learned weights, as there are several features that come in between the lines of the table (mostly indicator features for whether the modifier in a specific position of a candidate permutation is equal to a specific word). The n-

gram count features have some of the highest weights, followed by features that look at word endings of modifiers in specific positions and a feature that counts how often the head noun and the modifier closest to it show up together in the training data. These feature weights capture the essence of our MAXENT model – we memorize what we can from the training data, and the high values for these feature weights let the n-gram count feature dominate when we have already seen a test set in its entirety at training time. However, because the training data is extremely sparse, we need to supplement these n-gram counts, and we use linguistic intuition to create features that capture other aspects of the modifier orderings that don't rely on entire sequences being present in our training data.

5.2 Examples

In this section, we discuss a few modifier sequences present in our data and the ordering that MAXENT and each of our baselines determined to be most fluent.

5.2.1 "significant generating capacity"

The sequence "significant generating capacity" is present in the WSJ corpus with the results shown in Figure 5-1. The CLASS BASED approach assigns Class 1-2-3 to word type "generating" and Class 2-3-4 to word type "significant," and because the average preferred position of Class 1-2-3 is closer to the head noun than that of Class 2-3-4, the CLASS BASED approach correctly deduces that "generating" should come closer to the head noun. However, the GOOGLE N-GRAM approach gets this sequence wrong, because the count for the bigram "generating significant" is much higher than that for "significant generating". This is a case of inflated counts – "significant generating" is clearly the right choice when it comes to modifier sequences and "generating significant" should never occur, but because "generating significant" functions as another part of a sentence with a very high count (for instance, "An event of colossal proportions occurred, generating significant interest from the media"), the counts in the Google n-gram corpus are muddled and thus produce an incorrect result

for our application.

MAXENT is able to pick the right candidate permutation with high confidence and assigns a probability of 0.999 to "significant generating capacity" even though neither candidate permutation is seen at training time by leveraging other pieces of evidence, and the important contributing features are listed in Figure 5-1. An important feature for both candidate permutations is the position of the word "significant". The permutation "generating significant capacity" sets off the feature "significant is word in position 1", which has a negative weight because the modifier "significant" rarely comes right next to the head noun. In addition, the count of "significant capacity" is lower than that of "generating capacity" in the training data, which helps boost the probability of the correct permutation as well. Finally, because the CLASS BASED approach gets this example correct, we are able to use this as a signal in our model as well.

"significant generating capacity"

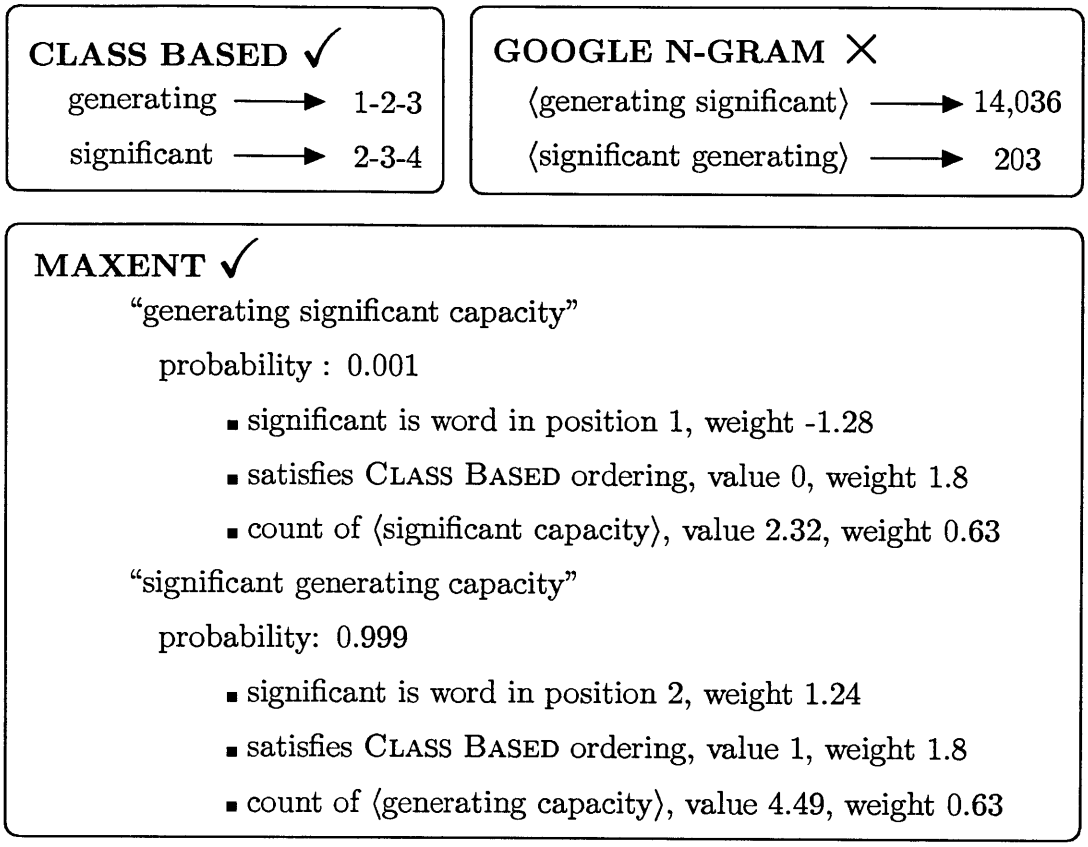


Figure 5-1: A modifier sequence found in the WSJ corpus, with the results of the three approaches. MAXENT and CLASS BASED are able to correctly order the modifiers, but GOOGLE N-GRAM does not because the counts are skewed by the fact that "generating significant" serves as another part of a sentence with very high occurrence.

5.2.2 "blue wool swimming trunks"

The sequence "blue wool swimming trunks" is present in the Brown corpus with the results shown in Figure 5-2. The CLASS BASED approach determines that "blue" should be farthest from the head noun but assigns the same class to both "wool" and "swimming". It is thus unable to determine whether the best ordering is "blue wool swimming trunks" or "blue swimming wool trunks", and must guess randomly. None of the permutations of this set of modifiers is present in the Google n-gram corpus, so GOOGLE N-GRAM does not get this example right either.

The MAXENT approach correctly picks out the right candidate permutation and assigns it a probability of 0.83, and the features that contribute towards biasing the model to choose the right permutation are listed in Figure 5-2. As in the previous example, none of the candidate permutations are present in the training data so we don't have 3-gram counts for any of them. However, we can leverage several features that look at smaller pieces of the permutation. For instance, because we've seen "swimming trunks" several times in the training data, this pushes us towards picking a permutation with the word "swimming" closest to the head noun. In addition, some of the bigrams present in "blue wool swimming" are also found in the training data even if the entire 3-gram is not, which helps us. And the CLASS BASED feature helps here as well, as it boosts the probability of candidate permutations with "blue" farthest from the head noun.

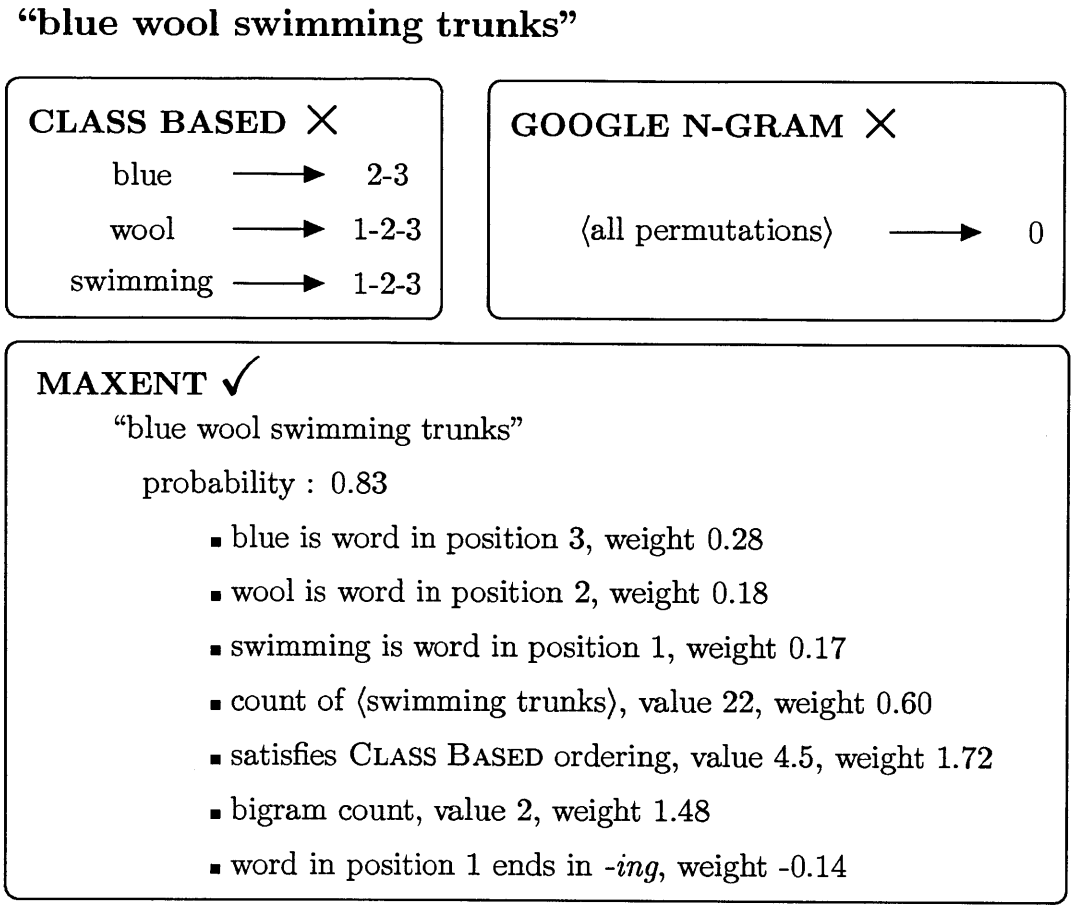
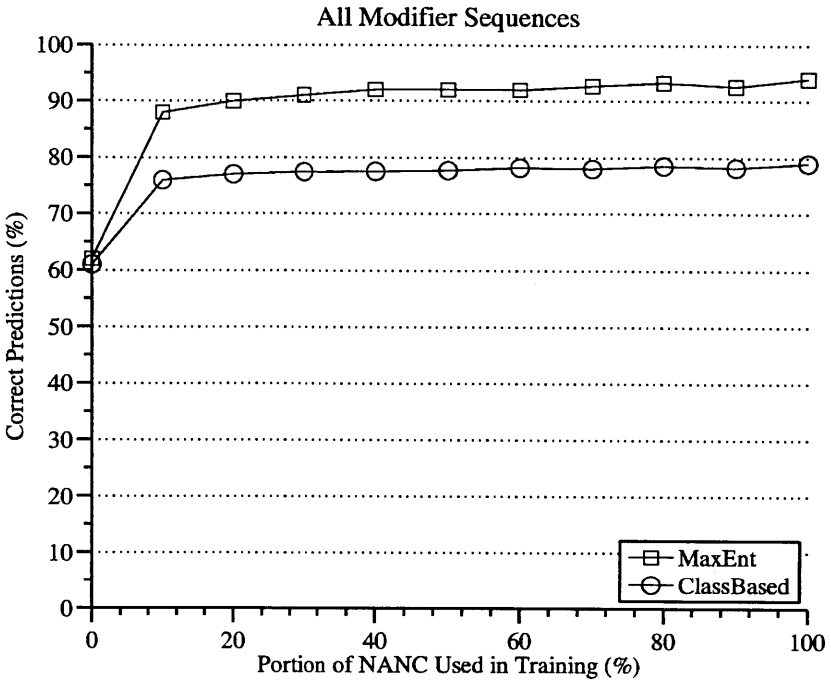


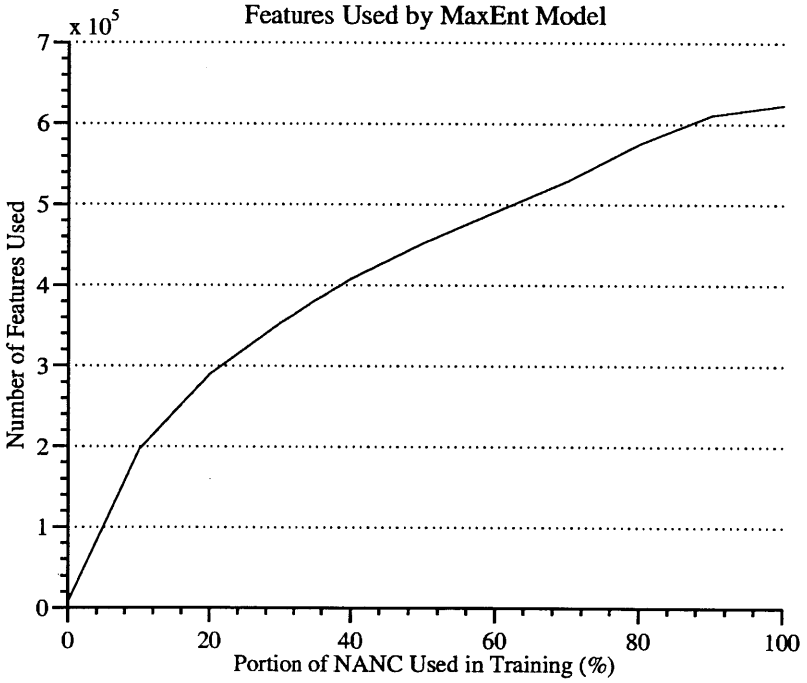
Figure 5-2: A modifier sequence found in the Brown corpus, with the results of the three approaches. The GOOGLE N-GRAM approach does not get this sequence correct because of sparsity issues and CLASS BASED assigns the same class to two of the modifiers and is forced to make a random guess. The MAXENT approach correctly orders the modifiers, with a list of the features that are used.

5.3 Learning Curves

We compared the effects of increasing the amount of training data when using the CLASS BASED and MAXENT methods by initially training each system on just the Brown and Switchboard corpora and testing on WSJ. Then we incrementally added portions of NANC, one tenth at a time, until the training set included all of it. The results for sequences of all lengths as well as the number of features used by the MAXENT approach as the training data scales up are shown in Figure 5-3. Figure 5-4 shows the results broken down by sequence length. These learning curves illustrate that while the MAXENT and CLASS BASED approaches have comparable accuracies when trained on small datasets, the MAXENT approach is able to benefit from additional data much more than the CLASS BASED approach can. The MAXENT approach does not have a fixed set of classes limiting the amount of information the model can learn, and the difference between the two approaches as the training set gets larger is even more stark on sequences of longer lengths.

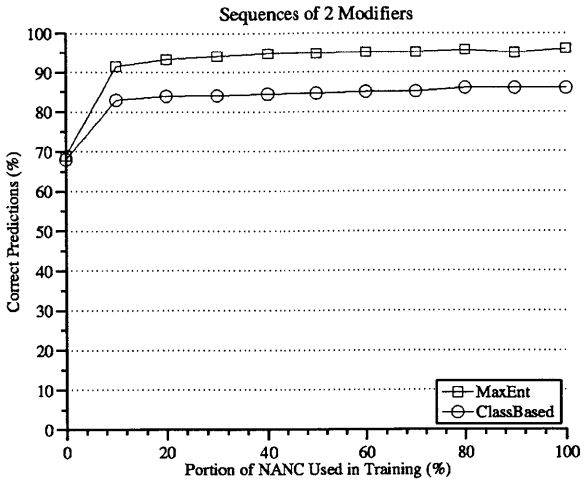


(a)

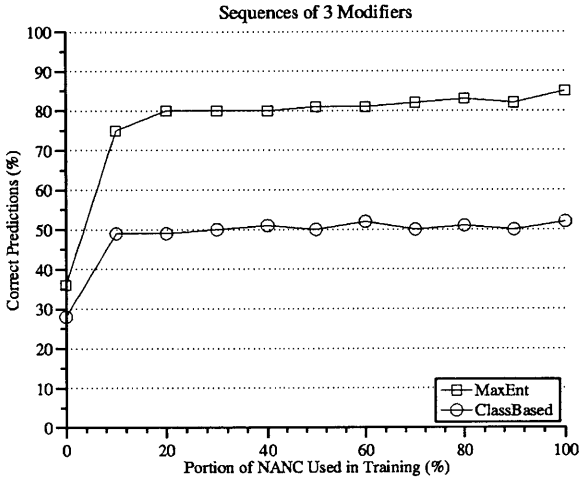


(b)

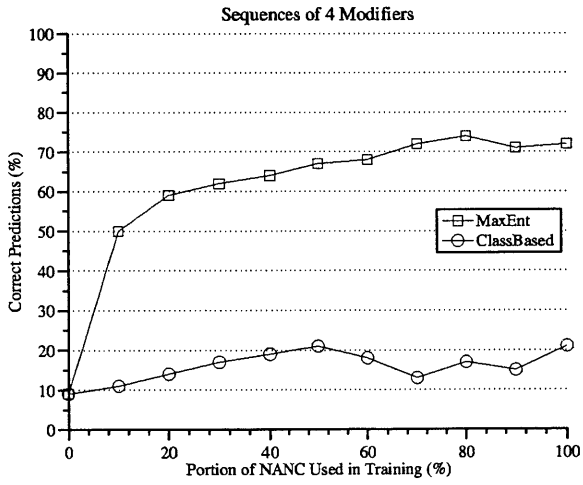
Figure 5-3: Learning curves for the MAXENT and CLASS BASED approaches. We start by training each approach on just the Brown and Switchboard corpora while testing on WSJ. We incrementally add portions of the NANC corpus. Graph (a) shows accuracies over modifier sequences of all lengths while graph (b) shows the number of features active in the MaxEnt model as the training data scales up.



(a)



(b)



(c)

Figure 5-4: Learning curves for the MAXENT and CLASS BASED approaches for sequence lengths of 2, 3, and 4.

Chapter 6

Conclusion

The maximum entropy ranking approach is able to significantly outperform previous approaches by allowing for a richer model of the modifier ordering process. The strength of this model lies in its ability to incorporate arbitrary features. This allows us to create simple n-gram count features to memorize what we can from the training data while also adding in features that capture linguistic intuitions about modifier orderings to attain higher accuracy on the tail of our test corpora. In addition, we can also incorporate previous approaches very compactly as features in our model (an example being the CLASS BASED feature we use), allowing us to draw from the strengths of past approaches. Our model is simple and easy to train with off-the-shelf techniques and takes only a few hours to train, which we think is well worth the increase in accuracy we attain compared to previous approaches.

However, there are still many improvements that could be made to our system. As discussed in Chapter 5, because the Google n-gram corpus contains counts for all sentence fragments of web documents and not just modifier sequences, the counts in this corpus are not necessarily well suited for picking out the correct permutation of a set of modifiers. A tagged version of the Google n-gram corpus is set to be released, and we could update our GOOGLE N-GRAM baseline to avoid this counting issue by taking advantage of the tags and only looking at sequences explicitly tagged as NPs.

We could also use the Google n-gram counts as a feature in our model, and Google has also released a new Google Books corpus with n-gram counts from select books

(Michel et al. 2011), and we could use these counts as a feature as well.

It might be helpful to build a language model based on the n-gram counts of the various corpora we're working with and to use this to generate n-gram counts, which could counteract the sparsity issues that merely using corpus n-gram counts creates.

It would also be beneficial to look at the punctuation present in our training sequences, as currently all punctuation is stripped from any noun phrases we extract. Seeing both the phrases "exquisite blue Macedonian vase" and "exquisite, Macedonian blue vase" could clue us into the fact that while the first phrase is the most correct when treating the three modifiers as separate, when using "Macedonian blue" as a hue, the latter phrase is the correct one.

In addition, it should be noted that our methods only work in unmarked contexts. In certain cases, "the brown big dog" may be considered more correct than "the big brown dog" if the emphasis is placed on the word "brown", in which case the phrase would be used for disambiguation. Our methods ignore these cases and only work in neutral situations.

Finally, while many sets of modifiers have stringent ordering requirements, some variations on orderings, such as "former famous actor" vs. "famous former actor," are acceptable in both forms and have different meanings. It may be beneficial to extend the model to discover these ambiguities.

Bibliography

[1] T. Brants and A. Franz. The google web 1t 5-gram corpus version 1.1. *LDC2006T13*, 2006.

[2] R.H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[3] M. Collins and T. Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70, 2005.

[4] R. M. W. Dixon. Where Have all the Adjectives Gone? *Studies in Language*, 1(1):19–80, 1977.

[5] A. Dunlop, M. Mitchell, and B. Roark. Prenominal modifier ordering via multiple sequence alignment. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 600–608. Association for Computational Linguistics, 2010.

[6] R. Malouf. The order of prenominal adjectives in natural language generation. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 85–92. Association for Computational Linguistics, 2000.

[7] J.B. Michel, Y.K. Shen, A.P. Aiden, A. Veres, M.K. Gray, J.P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, et al. Quantitative analysis of culture using millions of digitized books. *Science*, 331(6014):176, 2011.

[8] M. Mitchell. Class-based ordering of prenominal modifiers. In *Proceedings of the 12th European Workshop on Natural Language Generation*, pages 50–57. Association for Computational Linguistics, 2009.

[9] R. Quirk, S. Greenbaum, R.A. Close, and R. Quirk. *A university grammar of English*, volume 1985. Longman London, 1974.

[10] A. Ratnaparkhi, J. Reynar, and S. Roukos. A maximum entropy model for prepositional phrase attachment. In *Proceedings of the workshop on Human Language Technology*, pages 250–255. Association for Computational Linguistics, 1994.

[11] Jeffrey C. Reynar and Adwait Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *In Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 16–19, 1997.

[12] J. Shaw and V. Hatzivassiloglou. Ordering among premodifiers. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 135–143. Association for Computational Linguistics, 1999.

[13] R. Sproat and C. Shih. The cross-linguistic distribution of adjective ordering restrictions. *Interdisciplinary approaches to language*, pages 565–593, 1991.

[14] A. Teodorescu. Adjective Ordering Restrictions Revisited. In *Proceedings of the 25th West Coast Conference on Formal Linguistics*, pages 399–407. West Coast Conference on Formal Linguistics, 2006.