Ordering Prenominal Modifiers with a Reranking Approach

Jenny Liu
MIT CSAIL
jyliu@csail.mit.edu

Aria Haghighi
MIT CSAIL
me@aria42.com

Abstract

In this work, we present a novel approach to the generation task of ordering prenominal modifiers. We take a maximum entropy reranking approach to the problem which admits arbitrary features on a permutation of modifiers, exploiting hundreds of thousands of features in total. We compare our error rates to the state-of-the-art and to a strong Google n-gram count baseline. 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 a maximum error reduction of 68.4% and average error reduction across all test sets of 41.8% compared to our Google n-gram count baseline.

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: 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.

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 for more examples). In this work, we consider the task of ordering an unordered set of prenominal modifiers so that they sound fluent to native language speakers. This is an important task for natural language generation systems.

Much linguistic research has investigated the semantic constraints behind prenominal modifier orderings. One common line of research suggests that modifiers can be organized 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". Using $>$ to denote precedence of semantic groups, some commonly proposed orderings are: *quality* $>$ *size* $>$ *shape* $>$ *color* $>$ *provenance* (Sproat and Shih, 1991), *age* $>$ *color* $>$ *participle* $>$ *provenance* $>$ *noun* $>$ *denominal* (Quirk et al., 1974), and *value* $>$ *dimension* $>$ *physical property* $>$ *speed* $>$ *human propensity* $>$ *age* $>$ *color* (Dixon, 1977). However, correctly classifying modifiers into these groups can be difficult 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 class 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 counting methods and clustering algorithms, and Malouf (2000) expands upon Shaw and Hatzivassiloglou's work.

This paper describes a computational solution to the problem that uses relevant features to model the modifier ordering process. By mapping a set of features across the training data and using a maximum entropy reranking model, we can learn optimal weights for these features and then order each set of modifiers in the test data according to our features and the learned weights. This approach has not been used before to solve the prenominal modifier ordering problem, and as we demonstrate, vastly outperforms the state-of-the-art, especially for sequences of longer lengths.

Section 2 of this paper describes previous computational approaches. In Section 3 we present the details of our maximum entropy reranking approach. Section 4 covers the evaluation methods we used, and Section 5 presents our results. In Section 6 we compare our approach to previous methods, and in Section 7 we discuss future work and improvements that could be made to our system.

2 Related Work

Mitchell (2009) orders sequences of at most 4 modifiers and defines nine classes that express the broad positional preferences of modifiers, where position 1 is closest to the noun phrase (NP) head and position 4 is farthest from it. Classes 1 through 4 comprise those modifiers that prefer only to be in positions 1 through 4, respectively. Class 5 through 7 modifiers prefer positions 1-2, 2-3, and 3-4, respectively, while class 8 modifiers prefer positions 1-3, and finally, class 9 modifiers prefer positions 2-4. Mitchell counts how often each word type appears in each of these positions in the training corpus. If any modifier's probability of taking a certain position is greater than a uniform distribution would allow, then it is said to prefer that position. Each word type is then assigned a class, with a global ordering defined over the nine classes.

Given a set of modifiers to order, if the entire set has been seen at training time, Mitchell's system looks up the class of each modifier and then orders the sequence based on the predefined ordering for the classes. When two modifiers have the same class, 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.

Shaw and Hatzivassiloglou (1999) use corpus-based counting methods as well. For a corpus with w word types, they define a $w \times w$ matrix where $Count[A, B]$ indicates how often modifier A precedes modifier B. Given two modifiers a and b to order, they compare $Count[a, b]$ 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 $< a, b >$ for $Count[a, b]$ number of times. If this probability is above a certain threshold then they say that a precedes b. Shaw and Hatzivassiloglou also use a transitivity method to fill out parts of the *Count* table where bigrams are not actually seen in the training data but their counts can be inferred from other entries in the table, and they use a clustering method to group together modifiers with similar positional preferences.

These methods have proven to work 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 also 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 reranking approach that tries to develop a more comprehensive model of the modifier ordering process to avoid the sparsity issues that previous approaches have faced.

3 Model

We treat the problem of prenominal modifier ordering as a reranking 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 reranking approach (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 3.1, 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 techniques.

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 binary feature that tells us whether 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.

3.1 Feature Selection

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. For instance, perhaps the majority of words that end with *-ly* are adverbs and should usually be positioned farthest from the head noun, so we can define an indicator function that captures this feature as follows:

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

We create a feature of this form for every possible modifier position i from 1 to 4.

We might also 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:

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

See Table 2 for a summary of our features. Many of the features we use are similar to those in Dunlop et al. (2010), which uses a feature-based multiple sequence alignment approach to order modifiers.

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 $w \epsilon W$, where W is the set of all word types in the training data.
Ends In *e**	Modifier in position i ends in suffix $e \epsilon E$, where E = {*-al -ble -ed -er -est -ic -ing -ive -ly -ian*}
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 2: Features Used In Our Model. Features with an asterisk (*) are created for all possible modifier positions i from 1 to 4.

4 Experiments

4.1 Data Preprocessing and Selection

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 NPs with more than 5 modifiers, we kept those with 2-5 modifiers and with tags *NN* or *NNS* for the head noun. We also kept NPs with only 1 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 3 for a summary of the number of NPs of lengths 1-5 extracted from the four corpora.

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 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: <*beautiful blue*>, <*blue Macedonian*>, and <*beautiful Macedonian*>, along with the 3-gram <*beautiful blue Macedonian*>. 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 $< M, H >$, where H is the head noun of an NP and M is the modifier closest to it. In the example "the beautiful blue Macedonian vase," we would increment the count of $< Macedonian, vase >$ in the table. The n-gram and $< M, H >$ counts are used to compute numeric fea-

Number of Sequences (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 (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 3: Number of NPs extracted from our data for NP sequences with 1 to 5 modifiers.

ture values.

4.2 Google n-gram Baseline

The Google n-gram corpus is a collection of n-gram counts drawn from public webpages with a total of one trillion tokens – around 1 billion each of unique 3-grams, 4-grams, and 5-grams, and around 300,000 unique bigrams. We created a Google n-gram baseline that takes a set of modifiers B, determines the Google n-gram count for each possible permutation in $\pi(B)$, and selects the permutation with the highest n-gram count as the winning ordering x^*. We will refer to this baseline as GOOGLE N-GRAM.

4.3 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 test data sequence, 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.

4.4 Evaluation

To evaluate our system (MAXENT) and our baselines, we partitioned the corpora into training and testing data. For each NP in the test data, we generated a set of modifiers and looked at the predicted orderings of the MAXENT, CLASS BASED, and GOOGLE N-GRAM methods. We considered a predicted sequence ordering to be correct if it matches the original ordering of the modifiers in the corpus. We ran four trials, the first holding out the Brown corpus and using it as the test set, the second holding out the WSJ corpus, the third holding out the Switchboard corpus, and the fourth holding out a randomly selected tenth of the NANC. For each trial we used the rest of the data as our training set.

5 Results

The MAXENT model consistently outperforms CLASS BASED across all test corpora and sequence lengths for both tokens and types, except when testing on the Brown and Switchboard corpora for modifier sequences of length 5, for which neither approach is able to make any correct predictions. However, there are only 3 sequences total of length 5 in the Brown and Swichboard corpora combined.

Test Corpus		**Token Accuracy (%)**					**Type Accuracy (%)**				
		2	**3**	**4**	**5**	**Total**	**2**	**3**	**4**	**5**	**Total**
Brown	GOOGLE N-GRAM	82.4	35.9	12.5	0	79.1	81.8	36.3	12.5	0	78.4
	CLASS BASED	79.3	54.3	25.0	0	77.3	78.9	54.9	25.0	0	77.0
	MAXENT	**89.4**	**70.7**	**87.5**	0	**88.1**	**89.1**	**70.3**	**87.5**	0	**87.8**
WSJ	GOOGLE N-GRAM	84.8	53.5	31.4	71.8	79.4	82.6	49.7	23.1	16.7	76.0
	CLASS BASED	85.5	51.6	16.6	0.6	78.5	85.1	50.1	19.2	0	78.0
	MAXENT	**95.9**	**84.1**	**71.2**	**80.1**	**93.5**	**94.7**	**81.9**	**70.3**	**45.2**	**92.0**
Switchboard	GOOGLE N-GRAM	**92.8**	68.4	0	0	**90.3**	**91.7**	68.1	0	0	**88.8**
	CLASS BASED	80.1	52.6	0	0	77.3	79.1	53.1	0	0	75.9
	MAXENT	91.4	**74.6**	**25.0**	0	89.6	90.3	**75.2**	**25.0**	0	88.4
One Tenth of	GOOGLE N-GRAM	86.8	55.8	27.7	43.0	81.1	79.2	44.6	20.5	12.3	70.4
NANC	CLASS BASED	86.1	54.7	20.1	1.9	80.0	80.3	51.0	18.4	3.3	74.5
	MAXENT	**95.2**	**83.8**	**71.6**	**62.2**	**93.0**	**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: Token and 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 corpuses: 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.

MAXENT also outperforms the GOOGLE N-GRAM baseline for almost all test corpora and sequence lengths. For the Switchboard test corpus token and type accuracies, the GOOGLE N-GRAM baseline is more accurate than MAXENT for sequences of length 2 and overall, but the accuracy of MAXENT is competitive with that of GOOGLE N-GRAM.

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 across modifier 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 also 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) – this tradeoff is well worth the increase we attain in system performance.

6 Analysis

MAXENT seems to outperform 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 categories, with each category representing a different set of positional preferences. However, many of the modifiers in the training data get classified to the same category, and CLASS BASED makes a random choice when faced with orderings of modifiers all in the same category. When applying CLASS BASED

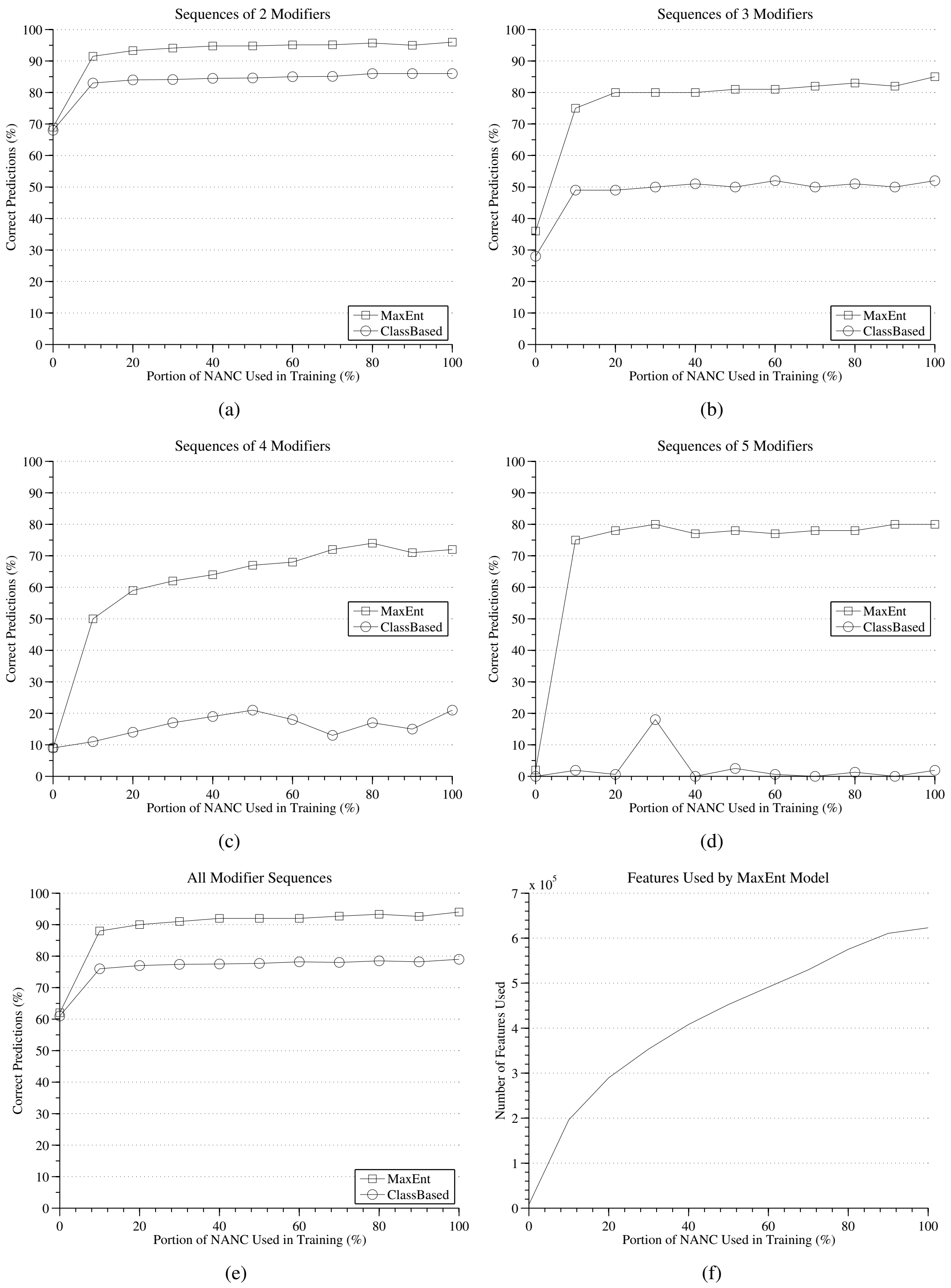

Figure 1: Learning curves for the MAXENT and CLASS BASED approaches. We start by training each approach on just the Brown and Switchboard corpora while testing on WSJ. We incrementally add portions of the NANC corpus. Graphs (a) through (d) break down the total correct predictions by the number of modifiers in a sequence, while graph (e) gives accuracies over modifier sequences of all lengths. Prediction percentages are for sequence tokens. Graph (f) shows the number of features active in the MaxEnt model as the training data scales up.

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. 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 show up in the Google n-gram corpus, so the GOOGLE N-GRAM method 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 we may be overcounting certain modifier permutations that can function as different parts of a sentence.

We also compared the effect that increasing the amount of training data has when using the CLASS BASED and MAXENT methods by initially training each system with 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 (see Figure 1) show that we are able to benefit from the additional data much more than the CLASS BASED approach can, since we do not have a fixed set of classes limiting the amount of information the model can learn. In addition, adding the first tenth of NANC made the biggest difference in increasing accuracy for both approaches.

7 Conclusion

The straightforward maximum entropy reranking approach is able to significantly outperform previous computational approaches by allowing for a richer model of the prenominal modifier ordering process. Future work could include adding more features to the model and conducting ablation testing. In addition, 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.

Acknowledgements

Many thanks to Margaret Mitchell, Regina Barzilay, Xiao Chen, and members of the CSAIL NLP group for their help and suggestions.

References

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

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

A. Dunlop, M. Mitchell, and B. Roark. 2010. 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.

R. Malouf. 2000. 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.

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

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

J. Shaw and V. Hatzivassiloglou. 1999. 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.

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

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