Clustering Methods for Improving Language Models

CS224N Natural Language Processing
Final Project
June 6, 2007

Ethan Dreyfuss (SUID 0520-3963)
Ian Goodfellow (SUID 0531-0893)
Paul Baumstarck (SUID 0538-6251)

Introduction

One of the basic tools of natural language processing is a language model which assigns likelihoods to arbitrary strings, scoring how likely they are to properly be part of a language. An ideal language model assigns high likelihoods to grammatical sentences while assigning low likelihoods to improbable or plain nonsensical sentences. Some applications for language models include decoding and machine translation.

A dominant, baseline type of language model is the *n*-gram model. This calculates a sentence's probability as a product of the probabilities over every word in the sentence given the *n-1*-preceding words, i.e.,

$$P(w_1 w_2 \ldots w_m) = \prod_{i=1}^{m} P(w_m | w_{m-1} w_{m-2} \ldots w_{m-n+1})$$

Thus the *n*-gram model encodes the basic assumption that words in a sentence are determined by their immediately preceding words.

Trigram models (where *n*=3) have proven adequate for obtaining good language model performance. However *n*-gram models—because of their simplicity—are also very limited. Attempts to improve them by moving to arbitrarily higher-order values of *n* (e.g., 4- and 5-gram) have improved performance and accuracy but only insofar as throwing more data at an insufficient model will allow. We believe true improvement will come from increasing the quality of the model and making more use out of the existing structure.

Thus we will endeavor to improve trigram models by including more, richer data beyond the words alone. We will include clustering derived by various means of classification, machine learning, tagging, stemming, and parsing. We will also seek to improve the probability measure of the entire sentence by looking beyond traditional *n*-gram models entirely and developing measures of key, related pairs of words in the sentence (such as the subject and verb).

Language Modeling

1. Full IBM Predict Model

The Full IBM Predict Model combines word-based and cluster-based models by the following formula:

$$P_{\texttt{fullibmpredict}}(w|w_{i-2}w_{i-1}) = (\lambda P(W|w_{i-2}w_{i-1}) + (1-\lambda)P(W|W_{i-2}W_{i-1}))\times (\mu P(w|w_{i-2}w_{i-1}W) + (1-\mu)P(w|W_{i-2}W_{i-1}W)$$

(Goodman)

In his paper, Joshua Goodman found that this model proved to be superior to all of the models he tried. Thus we decided to implement this model in our system.

Our implementation achieves the above model through a mixture of sub-models that interpolate with lower-order models. Each order of model is based on maximum likelihood estimation from counts that is then smoothed by absolute discounting (See our earlier work for proofs that linear interpolation and absolute discounting yield valid probability distributions). The amount of discounting and the interpolation weights are learned from a holdout set. This method of smoothing is chosen to be relatively simple to enable us to complete a more complex language model and accomplish more clustering tasks within the time available for the project (as our focus is on clustering methods, not on smoothing methods). For ideal performance, (Goodman) indicates that we should have implemented Kneser-Ney smoothing which has been found to improve performance with clustering more than other methods such as Katz smoothing.

[Add figure showing dependency graph]

To avoid hard-coding each of these models, we wrote a small number of classes that can be instantiated to create the multiple nodes of the dependency graph. Inside Full IBM Predict, each of the four sub-models is an instance of the IntermediateModel class. The IntermediateModel class interpolates between a higher-order model and a lower-order model. The lowest-order model is a UnistringModel class, which models the frequency of individual words or clusters. To construct the Full IBM Predict model, we began with *P(w)* (the probability of a word) and *P(W)* (the probability of the word's cluster) and successively built models on top of these.

2. Subject-Verb / Verb-Object Model

An obvious problem with constraining ourselves to trigram models is that we cannot gather statistics on any words that do not fit within the scope of a trigram. But there are some words in the sentence that are intimately connected (such as subjects and their verbs, verbs and their objects) no matter how many words separate them. For motivation, consider the sentences,

Jack threw the big rock.

Jack threw the big problem.

From a trigram model's perspective, all of individual components "threw the large," "the big rock," and "the big problem" are reasonably likely. However the second sentence should intuitively be much less likely since "threw" and "problem" are not a meaningful subject-verb pair.

Addressing this problem with higher-order *n*-gram models is infeasible since subjects and verbs can be separated by many intervening words; thus the model that could contain them within an *n*-gram would be so sparse that it would be unlikely to yield improvements in this respect without prohibitive amounts of training data.

Instead we propose to segment out these prominent word combinations and build language models over them directly. For these combinations we chose to excerpt subjects paired with their verbs (called SV pairs) and verbs paired with their objects (both direct and indirect, called VO pairs). We chose these since every sentence is guaranteed to have some number of both of these and we believed we could identify them easily enough by examining the parse trees of tagged sentences.

We detected SV and VO pairs by a heuristic approach examining the parse tree of a sentence. For detecting SV pairs, we searched for "NP" tags followed by a neighboring "VP" tag. We searched through the noun phrase and extract the leftmost words with "NN" POS's and labeled them the subject, S. And we searched through the VP and extracted the leftmost word with a "VB" tag, labeling it the verb, V. For finding VO's, we searched for VP's neighbored by a following NP or PP. Then we extracted the leftmost "VB"-tagged word from the VP and the leftmost "NN"-tagged words from the NP (skipping any preposition).

An example output of our algorithm on a parse tree is given to show its effectiveness:

```
(ROOT
  (S
    (NP
      (NP
        (NP (NNPS Travelers) (POS 's))
        (NN employee) (NNS benefits) (NN group))
      (, ,)
      (SBAR
        (WHNP (WDT which))
        (S
          (VP (VBZ includes)
            (NP (PRP$ its) (NN group) (NN health) (NN insurance) (NNS operations)))))
      (, ,))
    (VP (VBD posted)
      (NP
        (NP (NNS earnings))
        (PP (IN of)
          (NP
            (QP ($ $) (CD 24) (CD million)))))
      (, ,)
      (PP (VBN compared)
        (PP (IN with)
          (NP
            (NP (DT a) (NN loss))
            (PP (IN of)
```

```
        (NP
          (QP ($ $) (CD 3) (CD million))))
      (NP (JJ last) (NN year))))))
  (. .)))
```

Full sentence:
Travelers 's employee benefits group , which includes its group health insurance operations , posted earnings of $ 24 million , compared with a loss of $ 3 million last year

Extracted SV/VO pairs:
(SV, Travelers, posted)
(VO, includes, operations)
(VO, posted, earnings)

Note how the algorithm successfully identified the main subject-verb pair of "Travelers/posted" even though the component words were separated by a distance of twelve words.

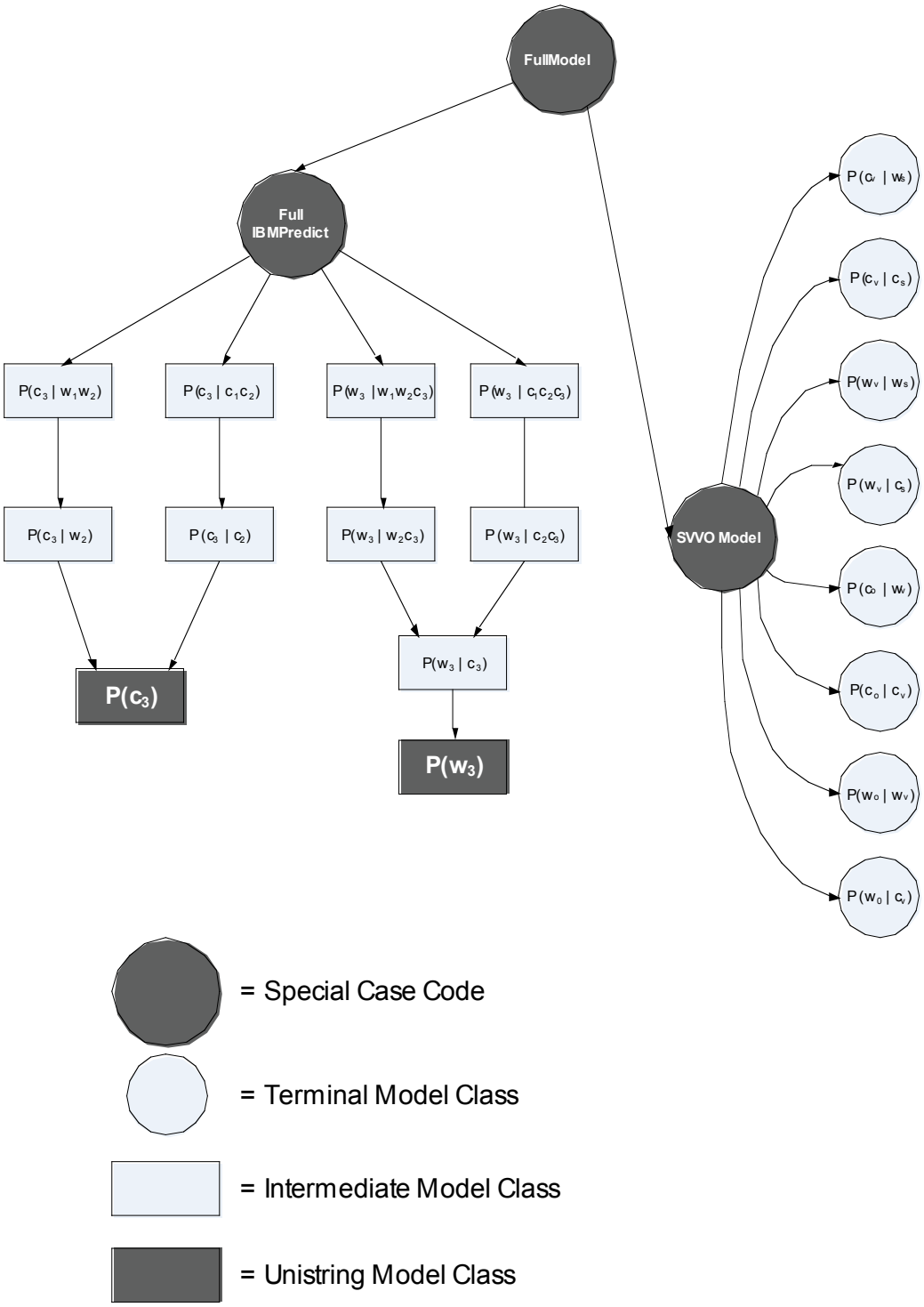
These relationships are formalized in our model with a formula similar to Full IBM Predict.

P(wv|ws) = (lambda * P(cv | ws) + (1.0 - lambda) * P(cv | cs))
* ((mu * P(wv | ws) + (1.0 - mu) * P(wv | cs))

where wv is a verb, ws is the subject of the verb, and cv and cs are their corresponding clusters. The formula for P (wo | wv) for verb-object pairs is similar. These models are not recursively enumerated or interpolated with lower-order models since they serve only to measure the appropriateness of the verb-object and verb-subject pairings. They are smoothed by absolute discounting, with the amount of discounting learned from the validation set.

3. Full dependency graph

The following graph demonstrates all of the dependencies in our recursive formulation of our language model, and demonstrates the ways that we designed our java classes to avoid code duplication:



Clustering

1. k-Means on Word Contexts

We designed and implemented our own k-means clustering method for finding

word clusters based on feature vectors meant to represent the word's context. We formalize the word's context as the vector of

$$P(w_i = w_j) \text{ for all possible } i \text{ and } j$$

where w_i is a variable in the language model and w_j is a word in the clustering dataset. For example, consider this toy dataset: "He kicked the rock. He kicked the ball. He threw the rock. He refuted the argument. He refuted the claim." To cluster for use with the variable w3 in a trigram model, our features are the probabilities of the w1 and w2 variables taking on specific values in the clustering set. For example, the features of the word "rock" are [P(w1 = kicked) = 0.5, P (w1 = threw)=0.5, P (w2 = the) = 1.0, and 0.0 for all other features]. The features for w1 are based on w2 and w3, the features for w2 are based on w1 and w3, and the features for a verb, object, or subject are based on the type of word it is paired with in that context.

We chose context as our feature for two reasons: one, because the language model is based on context, our clusters should discover the role that a word plays in the language model. And two, both syntax and semantics can be discovered from context: certain sequences of parts of speech are grammatical and others are not; likewise, certain combinations of semantic clusters of words are likely and others are not.

This is particularly obvious with verbs and nouns. For example, physical entities and physical nouns often go together. "He kicked the ball," though not in the toy dataset, should be an obviously valid sentence based on "ball" having a similar context to the word "rock." But the same could not be said for the test sentence "He kicked the idea." Granted, there are sentences that violate this heuristic, such as those involving negation ("you can't kick an idea"); however, the language model should discover their likelihood appropriately as long as they appear with nonzero frequency in the training set.

We chose k-means as our clustering algorithm because it provides hard clustering, that is, assignment of a word to a single cluster. This algorithm has two main drawbacks: it assumes that the clusters are hyperspherical in shape (when it uses Euclidean distance for its distance metric) and it must be instructed to find a specific number of clusters that may not fit the actual data well. We considered other algorithms but rejected them. Kruskal's minimum spanning tree algorithm, for instance, could discover arbitrary spaced clusters, but it would require computing $O(n^2)$ distances between the points. Expectation-Maximization would suffer from the same basic problems as k-means but would add the problem of not providing hard assignments, making the output unsuitable for use with the IBM Full Predict model. An added advantage of k-means clustering is that we were able to design an efficient sparse representation of the feature vectors. Our method of computing the distance from a data point to a cluster center only needs to consider those features that are non-zero for the data point, not every feature that is nonzero in either the point or the cluster center. For these reasons, we think k-means is a relatively good clustering algorithm for this problem.

We used Euclidean distance as our distance measure between points. We used 32 clusters in all cases for the most direct correspondence with the clustering based on Alexander Clark's code (see section 4). When run on the example dataset provided above, this method successfully puts "refuted" and "reiterated" in one w1 cluster, "kicked" and "threw" in another w1 cluster, "rock" and "ball" in one w3 cluster, and "claim" and "idea" in another w3 cluster. This should enable a trigram model to infer that a ball can be kicked but not an idea (in the most common sense of the verb).

2. Parse Labels as Cluster Assignments

We can consider a word's part of speech tag to be roughly correlated with some hypothetical cluster assignment, but there is no perfect correspondence since a word can be assigned to many different parts of speech based on context. For this project, we did not implement a part of speech tagger. However, since many high-quality part of speech taggers are available, we decided to experiment with using part of speech tags from parsed corpora as cluster labels in our language model.

3. WordNet

WordNet is a powerful tool which gives defined, "ground truth" relationships and groupings between many thousands of English words. Because of the specificity and surety of the data in WordNet, we desired to make use of this for the clustering task.

An intuitive way to cluster a word is by considering its hypernyms. Words related by hypernymy are those where one of the words has a broader meaning and includes the other word as a subset, such as how "milk" is a kind of "dairy product." Hyponymy is this same relationship in the opposite direction, so the hyponyms of "dairy product" include "milk," "cream," "half-and-half," "butter," "yogurt," "whey," "curd," "clabber," and "cheese" (words with a common hypernym are known as sister terms). By using a word's direct hypernym as its cluster, we preserve the original meaning of the word to a high degree and are guaranteed to have a cluster with well-related words.

Hypernyms for words can be obtained from WordNet. But, after deriving the direct hypernym for a word (which we called the 1st-order hypernym) we could next derive that word's hypernym and so obtain the 2nd-order hypernym for the original word. In general going higher up into the inherited hypernym chain for clustering does dilute the relation between the word and its cluster but it does also reduce the number of clusters.

But, since only nouns have hypernyms, to extend this method to more words we made use of the synsets, or sets of related words like the "sister words" as above. When searching for the cluster for a verb, adjective, or adverb (the other kinds of words that are in WordNet), we access the word's synset and select a single word from there to represent the whole synset. This performs the same function as clustering to 1st-order hypernyms as with nouns but it is not extendable to higher order (as the synset over these words is closed).

4. Alexander Clark's Part of Speech Induction

We decided to experiment with building off of existing clustering systems by using Alexander Clark's part of speech induction code, available at http://www.cs.rhul.ac.uk/home/alexc/, to cluster our own corpus. Though the motivation of Clark's work was to discover parts of speech, it is essentially general-purpose clustering. His code combines Ney-Essen clustering with a Bayes net model that uses morphological features of words. Ney-Essen clustering attempts to maximize the likelihood of the data in a trigram model by repeatedly changing the assignment of each word to a cluster (Clark). The advantage of using Clark's code is that he has already tuned it far more than we have had time to tune our own code for this assignment. The drawback is that the output of his algorithm is independent of the words' roles in the

model—that is, it does not produce different clusters for w1, w2, and w3 or for verb-object or verb-subject combinations.

We used the arguments provided with his code to cluster our own corpus. His suggestion of 32 clusters became the basis for clustering with our k-means method with a k constant of 32 (to enable more direct comparisons between our clustering method and his).

5. Stemming

A fast way to achieve object clustering is to use stemming, the process of recovering the root of a word by removing any suffixes or variable endings. For example, stemming turns all of the words "chewing," "chews," "chewed," and "chewy," into just the common root "chew." Stemming can also produce non-dictionary words as when it turns "repute" into "reput." This is because the word "reputable" must also stem back to "repute" and so the only part common to both words is "reput."

For stemming we implemented the well-known Porter Stemming algorithm designed in 1979. The algorithm is a compilation of many derived, heuristic rules for stemming Engilsh words. Many of the steps, in particular, include exhaustive enumerations of common English word suffixes and how they should be deleted.

For originality, we implemented the stemmer algorithm ourselves in [PorterStemmer.java]. But for verification we tested it against a version of the code available from Martin Porter's (the algorithm's designer) website. We verified our stemmer by testing that it duplicated the performance of the given stemmer on a corpus of over 100,000 words. (Porter)

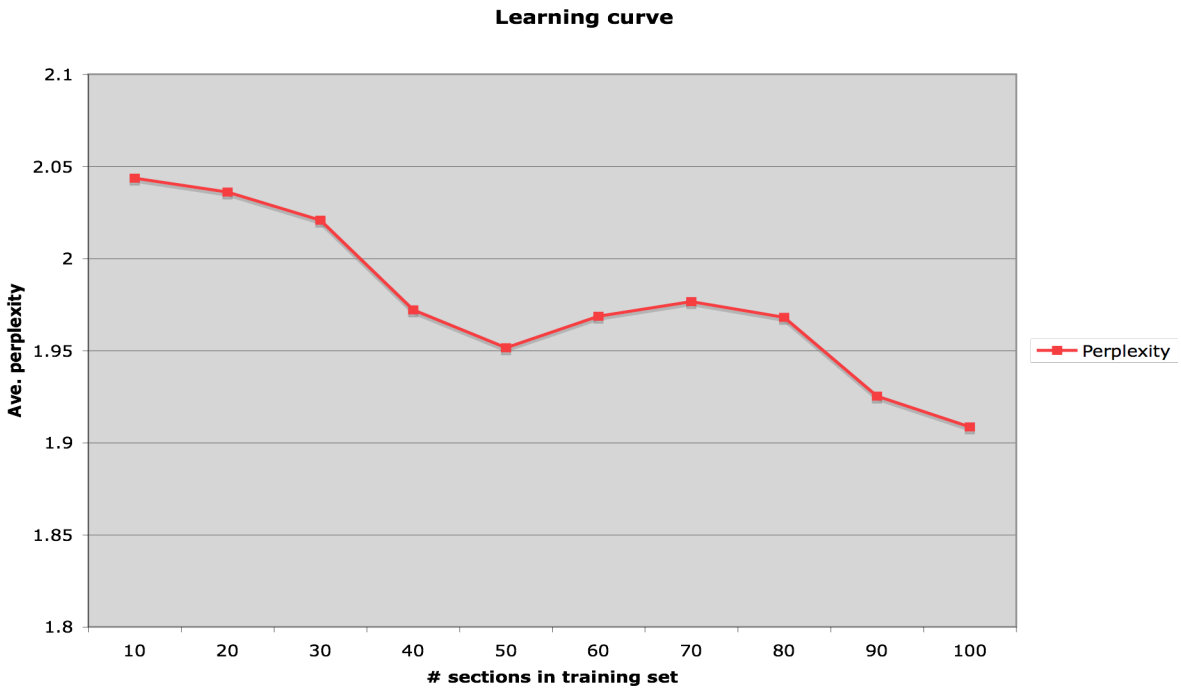
Datasets and evaluation methods

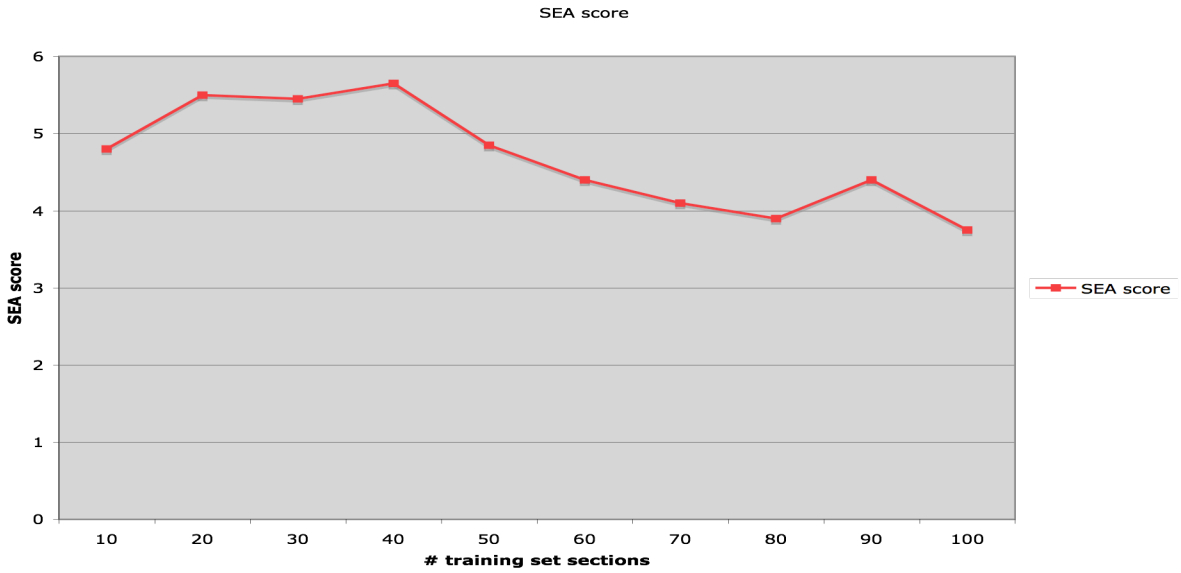
We performed all of our clustering on the first 100 sections of the BLLIP-WSJ corpus, which is approximately 14 million words. We originally planned to use BLLIP-WSJ for the language model as well, but eventually chose to use the Penn Treebank instead because we found better documentation of its parse tags. We used the conventional sections of the Treebank for training, validation, and development testing—that is, we train on sections 2-21, use sections 1 and 24 for validation. Our final test results are on the section conventionally used for final results, section 23.

We used perplexity as one measure of our model's performance, but it is a very deficient measurement because it cannot compare between models very well. For example, our SV-VO model returns very high probabilities because there are few verb-object and subject-verb pairs in a sentence, and thus interpolating with the SV-VO model yields extremely low perplexities without a proportional increase in discriminative ability. To counteract this, we devised our own Sentence Error Amount measure. This runs the model on several edited versions of a sentence, then returns the total edit distance of all sentences that the model prefers to the original. We then average this score over all of the sentences in the test set. This tests the model's ability to discriminate likely sentences from unlikely sentences. It is analogous to HUB Word Error Rate in that respect.

Results

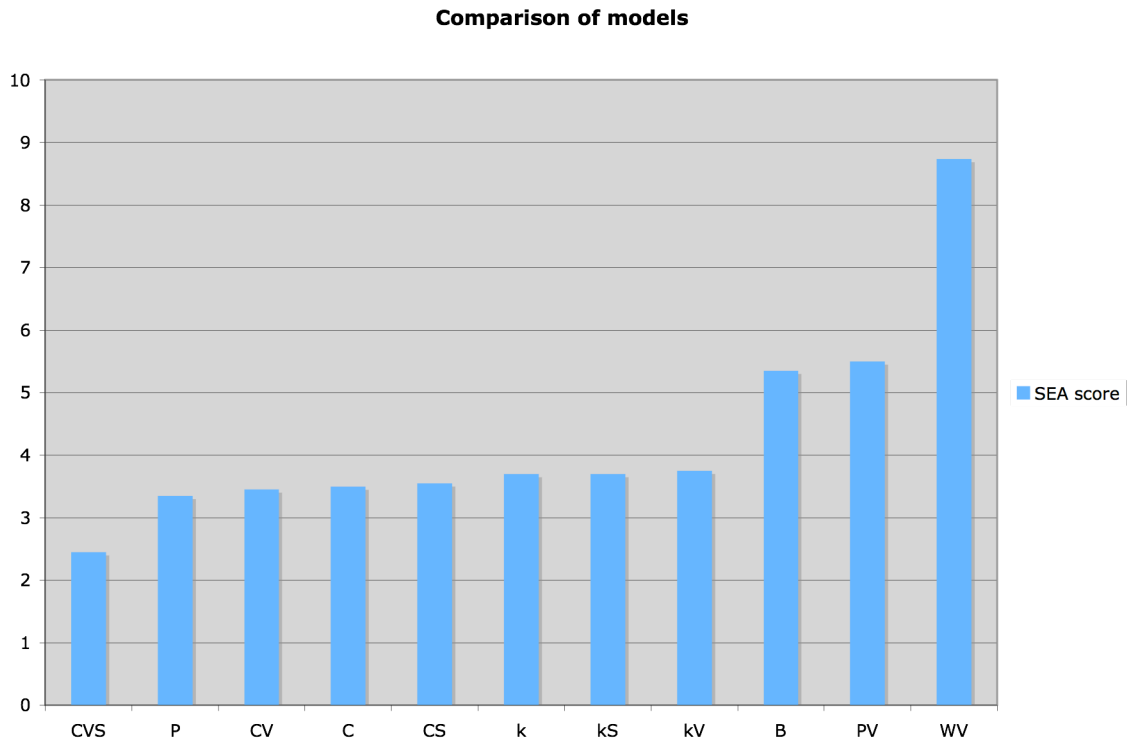
Our full model that interpolates between the SV-VO model and IBM Full Predict model shows clear improvements in performance with added training data. The follow graphs show perplexity and SEA score falling (improving) as the model trains on more sections of the Penn Treebank. Note that the perplexities have been scaled down greatly due to interpolation with the SV-VO model. These curves were produced using clustered derived from k-means clustering on trigram and SV-VO contexts and without the use of the Porter stemming algorithm.





The following bar graph compares the performance of several different models constructed by arranging different combinations of our features. The letter k represents k-means clustering, C represents Alexander Clark's Ney-Essen clustering, W represents

clusters from WordNet, and P represents clustering via the parse. V represents the use of the SV-VO model as well as IBM Full Predict. S represents the use of the Porter Stemmer. B is a baseline, a Paul's trigram model with backoff smoothing from PA1. We used the SEA score because comparison of perplexities would give our models an unfair advantage.



This shows that the majority of our models beat the baseline trigram. Our best model, CVS, used our most sophisticated automated method of clustering, and used our subject-verb and verb-object model as well as the Porter Stemmer. This suggests that all of our features were useful. Observe also that models that did not use the parse at all, such as kS, still outperform the baseline trigram model. This shows that models that still use very little computational resources at runtime (ie, that do not have to parse the sentence) can give noticeable improvements over the trigram model.

Given pre-existing work with clusters, we had expected that the way we used our clusters would be more important than the way we obtained them (Goodman). Our findings were contrary to that expectation. The extremely sparse cluster coverage provided by WordNet made it by far our worst model. Relatively good clusters such as those provided by Clark's code or gold-standard POS parses provided the best performance of any of our models. Specifically, we found that the high quality of Clark's clusters was more important than the context-sensitivity of our k-means clusters, even though our selected specifically for our model.

Error analysis is difficult due to the complexity of the model, but it is possible to make some generalizations. Consider the sentence "It is easy to say the specialist isn't doing his job." Even our best model, CVS, prefers, "It is easy to hold the time isn't doing his exchange." This demonstrates a weakness in our SV-VO model, which can't handle progressive verbs ("isn't doing") well. Learning subject-verb and verb-object agreement

requires more sophisticated tree-parsing than we originally expected, and due to the time pressure to get clustering processes running, we chose to begin clustering on subject and verb contexts without tuning our verb-object extractor as well as we would have liked.

We have also noticed that when the clustered models fail, they fail more nonsensically than do ordinary trigrams. This is presumably because the arbitrary number of clusters forces each cluster to contain several sub-clusters that would ideally be separated. For instance, one cluster from Clark's algorithm appeared to contain heads of state, periods of time, and hyphenated words. This reduces the descriptive power of the trigram model, because the value of each position in the sequence it learns is ambiguous. A better clustering algorithm that adapts to the data rather than imposing a set number of clusters on the data could alleviate this problem.

Discussion / future improvements

Given more time, this project could be improved in several ways. The short time span we had for the project made it difficult to run a satisfactory number of clustering and testing processes. The project has several relatively mundane flaws, such as that a bug in our code only tested the model on 20 sentences rather than the full section of the PTB. Due to the massive amount of time required to run all of our tests, we were unable to provide a better set of test results before the deadline.

Beyond these mundane glitches, there are several more elegant ways that our language models could be improved. Foremost, we would like to design a more mathematically principled way of combining the probabilities of individual subject-verb and verb-object pairs to yield a probability of the sentence. Also, as mentioned in the methods section, the language model should use Kneser-Ney smoothing (or some other method besides absolute discounting). Our clustering methods could be improved as well. As Professor Manning suggested in lecture, distance functions for word sense clustering that do not involve squared terms often perform better, so we would be interested in trying our k-means clustering algorithm with Manhattan instead of Euclidean distance (Manning). Lastly, there are other unsupervised clustering algorithms—such as mean-shift clustering—that can determine a natural number of clusters from the data and find clusters of arbitrary shape without the restrictive O(n^2) space requirement of Kruskal. Clustering by mean-shift might find more natural word clusters than both k-means and Ney-Essen.

Because we found that the stemmer and the SV-VO model improved performance, and because the clustering methods that yielded the best performance came from a machine learning approach rather than gold-standard lookup, we believe that our project demonstrates the value of continued investigation into the use of word clustering for language models.

References

Alexander Clark. 2003. Combining Distributional and Morphological Information for Part of Speech Induction. http://www.cs.rhul.ac.uk/home/alexc/papers/eacl2003.pdf

Joshu T. Goodman. 2001. A Bit of Progress in Language Modeling: Extended Version. Microsoft Research Technical Report. http://research.microsoft.com/~joshuago/longcombine.pdf

Manning, Christopher. CS224N lecture. Stanford University. 4 June 2007.

M.F. Porter, 1980, An algorithm for suffix stripping, *Program*, **14**(3) pp 130−137. http://www.tartarus.org/~martin/PorterStemmer/def.txt http://www.tartarus.org/~martin/PorterStemmer/