CS224N: Natural Language Processing David Kale & Gauhar Wadhera

Stanford University Wednesday, June 1^{st} 2005

Homework #FP: **Hidden State Language Modeling Based on Soft-membership Clustering of Words**
Due Date: 1^{st} June 2005
Collaborator(s): None

1 Introduction

The objective of this project is to analyze the performance of a class-based language model and compare it to the performance of traditional n-gram language models. Class-based language models are well-studied, as is the use of clustering to learn classes of words. However, it seems fairly standard across the literature to use hard-clustering i.e. assign each word to a single class and then to use these classes in a class n-gram language model. Also, word clustering seems to be often done in conjunction with document clustering, allowing document class to be used in determining word class and vice versa.

We hoped to do something a bit different from what appeared to be the standard approach. We, by no means, think our approach is novel, but we found relatively little literature whose work we seemed to be repeating. First and foremost, we clustered words entirely on part of speech information. Secondly, we believe that a single word's class can vary according to its context, suggesting a soft-membership clustering approach that would give a probability distribution over classes given a word and its context. Finally, since word class is a hidden variable, with the actual text representing the observables , we have attempted to construct HMM-based language models.

Ultimately, we found our learned hidden state class-based language model performed significantly worse than traditional n-gram word language models. We have ideas about how our approach might be improved with future research, but we are not convinced that such work is really warranted.

2 Design of Feature Space for words

Any clustering problem requires that the input be represented in some feature space. For example, points can be represented as a 3-D vector based on their coordinates. One of the key challenges in our case was to define the notion of similarity between words based on certain assumptions.

2.1 Semantic Similarity

According to our model, two words are similar if similar POS tags are assigned to them in text. However, each word can be assigned different POS tags based on context. A simple comparison metric would be to compare two words based on their most frequent tag. This approach yields a single dimensional feature space which makes it extremely quick to use for model construction and evaluation.

A more detailed approach entails using the probability distribution over POS tags as the feature space for words. The above space allows subtle effects to play a potentially important role in determining similarity. However, at the same time it increases the dimensionality of the feature space to the number of availabe POS tags.

For example, consider three words A, B and C. Let A occur as NN and JJ in text 30% and 70% respectively. Similarly, B occurs as NN and JJ in 55% and 45% cases repectively and C occurs in 80% and 20% of the cases as NN and VB respectively. In the case of using the most frequent tag, B and C seem to be more similar to each other. However, it really seems that it is more likely that A and B have greater similarity since they use the same set of POS tags with varying probabilities.

We only consider the second approach from the above two approaches. The above representation of each word is henceforth referred to as **Model 1** (a la IBM!). However, we take the notion of similarity of word based on POS tags a level further. Our intuition is that the chief reason for a single word being assigned different POS tags in different parts of text is due to the surrounding context. In order to determine the surrounding context we observe the POS tags of the previous "$nLeft$" and procedding "$nRight$" words.

Each combination of neighboring POS tags is defined as a **Neighborhood Profile** of a word. We represent each word, apart from a distribution over tags, as a distribution over all possible neighborhood profiles. This approach is henceforth referred to as **Model 2**. With N available POS tags the above distribution has potentially $N^{(nLeft+nRight)}$entries, making it seemingly infeasible. However, the sparsity of the problem works in our favor and makes its use a realistic proposition.

2.2 Computation of Distribution of POS tags

Model 1 and Model 2 share the probability distribution computed for each word over all the tags. We define some notation below,

Symbol	Meaning
N	Total No. of tokens
W	No .of unique words
N_{TW}	No. of unique ¡tag, word¿ pairs
$C_W(w)$	No. of occurences of word 'w'
$C_T(t)$	No. of occurence of tag 't'
$N_T(t)$	No. of words with $C(t,w) > 0$
$C(t,w)$	No. of occurences of word 'w' with tag 't'

We compute the distribution $P(t|w)$ for all words. In the case of infrequent words, we smooth their distributions using the distribution over POS tags given the entire dataset. This distribution i.e. $P(t|data)$ is a function of two factors.

1. Number of occurences of the tag.

2. Number of unique words having tag 't' for at least one occurence.

Also, we assign the distribution $P(t|data)$ to the distribution of unknown tokens. The distributions are computed as follows:

$$
\begin{aligned}
P(t|data) &= \alpha * \frac{C_T(t)}{N} + (1-\alpha) * \frac{N_T}{N_{TW}} \quad (\alpha = 0.5) \\
P_{ML}(t|w) &= \frac{C(t,w)}{C_W(w)} \\
P(t|w) &= \begin{cases} P_{ML}(t|w) & C_W(w) > K \\ \beta * P_{ML}(t|w) + (1-\beta) * P(t|data) & otherwise \end{cases} \\
P(t|*UNK*) &= P(t|data)
\end{aligned}
$$

2.3 Computation of Distribution of Neighboring POS tags

Model 2 also computes the distribution $P(< t_{i-1}^{i-nLeft}, t_{i+1}^{i+nRight+1} > |w_i)$. The notation is available below.

Symbol	Meaning
T	Set of all possible profiles $< t_{i-1}^{i-nLeft}, t_{i+1}^{i+nRight+1} >$
T_i	i^{th} possible profile
$C_W(w)$	No. of occurence of word 'w'
$C_T(t)$	No. of occurence of tag 't'
$C_{TW}(t,w)$	No. of occurences of word 'w' with tag 't'
$C_{NW}(n,w)$	No. of occurences of profile 'n' with word 'w'
$C_{NT}(n,t)$	No. of occurences of profile 'n' as neighbor of tag 't'
S_w	Set of tags 't' such that $C_{TW}(t,w) > 0$

We first compute the distribution $P(T_i|t)$ i.e. the distribution over profile for each tag. These are simple ML estimates computed as:

$$P(T_i|t) = \frac{C_{NT}(n,t)}{C_T(t)}$$

The above probabilities are used to compute the smoothed estimates of $P(T_i|w)$ including estimates for unknown words.

$$\begin{aligned} P_{ML}(T_i|w) &= \frac{C_{NW}(n,w)}{C_W(w)} \\ P(T_i|w) &= \begin{cases} P_{ML}(T_i|w) & C_W(w) > K \\ \gamma * P_{ML}(T_i|w) + (1-\gamma) * \sum_t P(t|w) * P(T_i|t) & otherwise \end{cases} \\ P(T_i|UNK) &= \sum_t P(t|data) * P(T_i|t) \end{aligned}$$

3 Soft Membership Clustering of Words via EM

Doing soft-membership clustering amounts to treating word-clas as a latent random variable C over which we maintain a conditional probability distribution $P(C|W)$ conditioned on the word W. Each word is represented by two distributions W_{tag} and W_{neigh} for the tag and neighborhood profile distrbutions as described before. Our goal, then, is to learn the $P(C|W)$ distribution. Such a task is a natural application of expectation-maximization (EM) to learn parameters for a good approximation of this distribution that maximizes the likelihood of the data. We have two different EM tasks, one for each model that we employ. For **Model 1**, the parameter we learn for each class 'j' is μ_j, which is the distribution over tags representing the class [1]. For **Model 2**, we learn an additional distribution θ_j , which is the mean representation of the neighborhood profile distributions of the class' constituent words. According to our model construction, as shall be shown, the former model is simply a special case of the latter. Hence, we derive our EM algorithm for Model 2 only.

The likelihood of our parameters on the data (with vocabulary V and number of classes to be learned C) is defined as

$$\begin{aligned} \ell(\mu,\theta) &= \sum_{i=1}^{|V|} \log p(w^{(i)};\theta,\mu) && (1) \\ &= \sum_{i=1}^{|V|} \log \sum_{j=1}^{C} p(w^{(i)}, c^{(i)}=j;\theta_j,\mu_j) && (2) \\ &= \sum_{i=1}^{|V|} \log \sum_{j=1}^{C} Q_i(c^{(i)}=j) \frac{p(w^{(i)}, c^{(i)}=j;\theta_j,\mu_j)}{Q_i(c^{(i)}=j)} && (3) \\ &\geq \sum_{i=1}^{|V|} \sum_{j=1}^{C} Q_i(c^{(i)}=j) \log \frac{p(w^{(i)}, c^{(i)}=j;\theta_j,\mu_j)}{Q_i(c^{(i)}=j)} && (4) \end{aligned}$$

with equation (4) by Jensen's inequality. Each expectation step (E-step) involves choosing an approximate distribution for classes for each word i, namely $Q_i(c^{(i)})$, based on the parameters θ and μ from the previous maximization step (M-step). The maximization step (M-step) involves holding Q constant and choosing new parameters θ' and μ' that maximize the likelihood of the data.

[1] similar to the means learned in Mixture of Gaussians EM

3.1 E-step

Here we choose our distribution $Q_i(c^{(i)})$, which we can also see as a set of weights $w_1^{(i)} \dots w_j^{(i)} l \dots w_C^{(i)}$ if we view the M-step as computing a weighted average. We also know that EM is monotonically increasing whenever $Q_i(c) = p(c^{(i)}|w^{(i)} : \theta, \mu)$. Hence, we compute the weights as a probability distribution i.e.

$$w_j^{(i)} = p(c^{(i)} = j|w^{(i)}; \theta_j, \mu_j)$$

In order to measure $p(c^{(i)} = j|w^{(i)}; \theta_j, \mu_j)$, we use the notion of distances between probability distributions as:

$$\begin{aligned} p_{unnormalized}(c^i = j|w^{(i)}; \theta_j, \mu_j) &= dist(w^{(i)}, \theta_j, \mu_j) - max_c(dist(w^{(i)}, \theta_c, \mu_c)) + \delta \\ p_(c^i = j|w^{(i)}; \theta_j, \mu_j) &= \frac{p_{unnormalized}(c^i = j|w^{(i)}; \theta_j, \mu_j)}{\sum_{k=1}^{C} p_{unnormalized}(c^i = k|w^{(i)}; \theta_k, \mu_k)} \end{aligned}$$

In order to compute the distance between the word and class distributions we consider the distributions to be vectors and compute the L1 norm between them i.e.

$$dist(w^{(i)}, \theta_j, \mu_j) = \alpha \|w_{tag}^{(i)} - \mu_j\|_1 + (1 - \alpha)\|w_{neigh}^{(i)} - \theta_j\|_1$$

It is clear that setting $\alpha = 1$ is equivalent to running **Model 1**, since it only takes into account the tag distributions.

3.2 M-step

Here we choose new parameters θ' and μ' for our distribution, which involves computing the weighted average distributions from all the points for each of the categories as:

$$\begin{aligned} \mu_j &= \frac{\sum_{i=1}^{V} w_j^{(i)} w_{tag}^{(i)}}{\sum_{i=1}^{V} w_j^{(i)}} \\ \theta_j &= \frac{\sum_{i=1}^{V} w_j^{(i)} w_{neigh}^{(i)}}{\sum_{i=1}^{V} w_j^{(i)}} \end{aligned}$$

3.3 Class Pruning

In addtion to generating a soft-clustering for words, EM also helps in pruning the number of classes required. After our EM converges, we compute, for each class 'C', the number of words

$$N_C = |w^{(i)} : (\arg\max_j w_j^{(i)}) = C|$$

. We removie all classes wot $N_C = 0$.

Our assumption in this case is that any given class which is not the closest class to any word, can be discarded. We renormalize all the probability distributions again, before proceeding to generating parameters for our HMM as described below.

4 Hidden Markov Models

According to the literature we review, most n-gram class models assign each word to a single category. Hence, we have a deterministic function $\pi : W \rightarrow C$. The probability of each sentence based on an n-gram class model is thus given as:

$$\begin{aligned} c_i &= \pi(w_i) \\ P(w_{i...n}|model) &= \prod_{i=1}^{n} P(w_i|c_i)P(c_i|c_{i-n+1...i-1}) \end{aligned}$$

However, we have attempted to compute a soft-assignment of words to categories. Hence, the previous context of categories can influence the category each word belongs to. The details of this approach are given below.

4.1 Parameter Estimation

In order to compute the HMM parameters, we use the results from the EM phase to estimate the transition, prior and emission probabilities.

4.1.1 Transition Probabilities

This set of parameters is learned after a certain amount of information loss. The two main constituents of input data used for learning the transitions are the probabilities $P(c|w)$ and the original untagged text. To compute the transitions, we replace each word by the class with the highest corresponding weight i.e $C(w) = argmax_c P(c|w)$.

In other words, we replace each word by its closest class and use the new text, which has a smaller vocabulary (all the classes) to learn n-gram class transitions based entirely on ML estimates.

4.1.2 Prior Probabilities

We simply add $(n-1)$ <START> symbols to the beginning of the class-based sentence. Hence, we compute the prior probabilities of each category 'C' as:

$$P_0(C) = \frac{C(< START_{n-1}, \ldots, START_1, C >)}{N}$$

where, N is the number of training sentences.

4.1.3 Emission Probabilities

Even though, we lose the soft-membership characteristic while learning the transitions, we do not do so in the case of learning emission probabilities. We estimate these as:

$$\begin{aligned} P(w) &= \frac{C_W(w)}{N} \\ P(w|c) &= \frac{P(c,w)}{P(c)} \\ &= \frac{P(c|w)P(w)}{\sum_w P(c,w)} \end{aligned}$$

Hence, using the weights learned during EM i.e. $P(c|w)$, and the ML estimates for word probabilities i.e. $P(w)$, we compute the emission probabilities for the HMM without losing the soft-membership characteristic.

4.2 Model Evaluation

Perplexity is a function of the probability of a sentence given the language model i.e. $P(w_{1...n}|model)$. In order to compute the probability of a sentence/string given an HMM, we use the Forward algorithm.

$$\begin{aligned} P(w_{1...n}|model) &= forward\_prob(w_{1...n}|model) \\ &= \sum_{c_1...c_n} P(w_{1...n}, c_{1...n}|model) \\ &= \sum_{c_1...c_n} \prod_{i=1}^{n} P(w_i|c_i)P(c_i|c_{i-n+1...i-1}) \end{aligned}$$

5 Markov Models

The above section described in detail the construction of HMMs for language modeling. However, we have also implemented the n-gram class models studied in literature. In order to use the HMM paradigm, we ensure that:

$$\forall w : |\{c : P(w|c) > 0\}| = 1$$

We implement this by computing the function π and using it to generate the emissions.

$$\begin{aligned} \pi(w) &= \arg\max_c P(c|w) \\ P(w|c) &= \frac{1\{\pi(w) = c\} * C_W(w)}{\sum_{w'} 1\{\pi(w') = c\} * C_W(w')} \end{aligned}$$

We also know that for any sentence,we have

$$P(w_{1...n}, c_{1...n}|model) \neq 0 \qquad iff \quad \forall w_i : c_i = \pi(w_i)$$

Therefore, we can still use the forward probability as an estimate for $P(w_{1...n}|model)$.

6 Implementation & Data

6.1 Implementation

The project was completed in 4 phases:

6.1.1 Creation of Training Data for EM

The module used the Stanford Tagger for POS tagging the training text. The resulting <word/tag> data is used to create the training data in terms of distributions for each word corresponding to its tags and nieghboring tags. The code was done in Perl due to its convenient text parsing structs.

6.1.2 EM

The training data prepared from the previous phase is used by a Java-based program that runs EM in order to generate the soft-membership clustering. We have used a few classes, provided to us, such as Counter and CounterMap as part of the code base for this module.

6.1.3 Computation of HMM Parameters

This module is integrated into the previous module and is, thus, also Java-based. It uses the soft-membership clustering learned from EM to estimate the emission, transition and prior probabilities for the HMM.

6.1.4 HMM

We have written a C++ wrapper for the C-based library **GHMM**. This module reads in the n^{th} order parameters and creates an equivalent 1^{st}-order HMM. The forward probabilities for each test sentence given the HMM are used to compute the perplexity over the test set.

6.2 Data

We have used the data from HW1, specifically the Treebank dataset. We have learned two sets of parameters, based on treebank-train-all and treebank-test. The test data to compute perplexities is treebank-validate.

7 Results

Using the datasets mentioned above, we trained each of our models on the POS-tagged words and POS-neighborhoods four times, once on data from *treebank-sentences-spoken-test.txt* and once on data from *treebank-sentences-spoken-train.txt.all*, learning a bigram model and trigram model on each. We did all of our testing on untagged sentences from *treebank-sentences-spoken-validate.txt*. For comparison we downloaded and installed the CMU-Cambridge Statistical Language Modeling toolkit (http://mi.eng.cam.ac.uk/ prc14/toolkit.html), which implements a relatively sophisticated ngram language model toolkit. Using this toolkit, we trained and tested bigram and trigram language models with no smoothing, absolute discounting, and witten-bell smoothing. We recorded a perplexity score on the validate dataset for each model in the able below. Note that columns represent different training sets and nth-orders, while different rows represent different models (CMU, markov model, hideen markov model, hidden markov model with POS neighborhoods). "Best" results for each training set and model type are bolded.

	training set, n-grams			
model	test.txt, 2	test.txt, 3	train.txt.all, 2	train.txt.all, 3
CMU	387.37	**373.60**	357.86	**306.14**
CMU, absolute discounting	395.62	379.96	361.68	308.30
CMU, witten bell smoothing	417.93	418.95	374.49	335.34
MM, POS tag	1571.95	**1556.45**	1605.02	**1568.17**
HMM, POS tag	1254.29	**1253.77**	1681.99	1680.97
HMM, with neighbors (1,1)	1306.78	1303.68	1690.35	1690.09
HMM, with neighbors (2,1)	1258.05	1256.73	1695.06	1693.60
HMM, with neighbors (3,1)	1274.47	1273.18	1623.00	**1621.20**

8 Discussion

From the above perplexity results, we are reasonably satisfied that our approach will not outperform a traditional language model, at least on datasets like the one we used. We confess that we are not fully certain as to the significance of the differences between the performance of our model and that of the CMU SLM toolkit; it is entirely possible that comparison on perplexity scores alone is not a sufficient comparison of different language models, particularly ones as different as a word-based n-gram model and class-based HMM model. More extensive testing is necessary to ascertain a more confident comparison and estimation of our approach's performance and power (or lack thereof). Nevertheless, we feel that our results suggest that soft-membership clustering of words based on parts of speech and part-of-speech neighborhoods is not a worthwhile or effective method for augmenting HMM-based language models. More extensive research into their pairing is warranted (assuming it has not already been done or is not already underway...we admit our literature review was far from exhaustive) only perhaps to ascertain why it seems to perform so poorly given that it seems to make intuitive sense.

As for ourselves, if we were to invest any additional time in investigating this approach, it would be in looking at the effects of

- *Different distance metrics*: our EM clustering relies heavily on a reasonable notion of "distance" between words (as described by their features), between classes, and between words and classes. We

feel that investing greater effort and focus into developing a robust and meaningful distance metric may pay off.

- *Data*: we feel that in the end, our datasets proved to be less rich and interesting that we'd hoped (we used them more out of convenience than out of principle decision). We feel that the language models we trained on them were rather brittle and sensitive to minor changes (in documented experimentation, we found that making slight alterations to either the training or test sets often greatly impacted perplexity and performance) and that our approach would benefit from training and testing on a larger, more varied corpus.
- *Feature selection*: Part-of-speech tags and neighborhoods seem useful for clustering words and yield natural, intuitive classes (after all, they are classes of words themselves), but the performance of our model suggests otherwise, at least for our approach. We are curious to know what other features might prove more useful. These features could be lexical, syntactic, semantic, or other (position in sentence, length, prefix, suffix, etc.).
- *Combining with other clustering*: Much research has been done on augmenting word clustering with document clustering so that topical information can be used to conditional word class distributions. Such an approach probably would not have proved effective for our dataset (since it was fairly unvaried in topic and was organized by sentence rather than document), but perhaps with other datasets this could prove interesting and useful.