**CMPT 825: Natural Language Processing** **Spring 2008**

Lecture 1 — Feb 6, 2008

*Lecturer: Anoop Sarkar* *Scribe: Javad Safaei*

# 1.1 Hiding a Semantic Hierarchy in a Markov Model [1]

## 1.1.1 General Concepts

We know that in logic a predicate is a relation between its arguments. In other words, a predicate defines constraints between its arguments. A predicate $\rho(v, r, c)$ is called *selectional restriction* where $v$ is a verb, $r$ is a role or an object and $c$ is called a class which is a noun. *Selectional preference* $\sigma : (v, r, c) \rightarrow a$ is a function of these predicates to a real number. Where $a$ shows the degree of dependency of these arguments with each other. This paper introduces a new method for induction of selectional preference. The training data for this task is British National Corpus. There are two major problems in this training data; it is noisy and contains ambiguity. The noise is usually from parsing and tagging, and ambiguity is within the language. For example the word bread can be used as money or food. In addition to this corpus the proposed algorithm uses semantic hierarchy for induction of selectional preferences. In the following picture an example of semantic hierarchy is given.

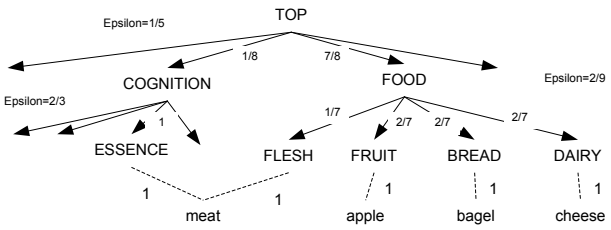


**Figure 1.1.** an example of semantic hierarchy.

### 1.1.2 The Algorithm

Selectional preference can be re-written by a probability $p(v, r, n)$ which is equal to $p(n|v, r) \cdot p(v, r)$. The main part of this expression is the likelihood probability $p(n|v, r)$. To estimate this probability we consider an HMM for each $p(n|v = v_0, r = r_0)$. In other words, for every assignment of values for $v$ and $r$ we assume one HMM. All of these HMM are the same in terms of structure, but different in terms of emission and transition probabilities. The states and transitions for the HMMs are identified with the given semantic hierarchy. The nodes of hierarchy represent classes (concepts) and the arcs are hyponymy ("is-a") relation. We assume that all the terminal nodes in the hierarchy are word senses.

A run of HMM begins at the root of semantic hierarchy, and it generates many child concepts in the internal nodes to reach to the terminal node ($c$) and generate the word sense $w$ (which are shows by leaves in the semantic hierarchy). Therefore, HMM run is the same as finding a path from the root to the leaves. If there is no arc between two nodes, it means that the transition probability between these nodes is zero, otherwise it should be between 0-1. Also in our HMMs the words, $\epsilon$ (empty) can be generated, in other word states are free to generate a word or not. But $\epsilon$ can't be generated in the terminal nodes, so the probability of it is zero.

Now we should estimate the parameters of our HMMs. To do this, for every HMM correspondent to $(v, r)$ the training data is a bunch of words $n$ from the token $(v, r, n)$. Now with these HMMs we can say a given word $n$ is ambiguous or not. If there are multiples of path with similar probabilities leading to generating of $n$, then there is ambiguity. Also using the probabilities we can find the selectional preferences. For example consider figure 1.1. There are two paths leading to the word sense *meat*, therefore it is ambiguous.

**Smoothing.** Assume that the training data is *meat*, *apple*, *bagel*, and *cheese*. *meat* is ambiguous because we have two paths from the root ending to it. Now suppose that we want to find the transition probabilities, and our training data are just these four words, each of them repeated only once. For the word *meat* we have two paths, so for the estimated count of the transition FOOD → FLESH we have $1/2 = 0.5$. For each of the other words we have the transitions FOOD → FRUIT, FOOD → BREAD, and FOOD → DAIRY each of them once. So all the transitions from FOOD is 3.5, and the transition probabilities are $\{1/7, 2/7, 2/7, 2/7\}$. These probabilities are called

empirical probabilities $p(t)$. By smoothing this probability we can compute the pseudo-empirical probability. Let $u(t)$ be the uniform probability then the pseudo-empirical is computed as $\epsilon u(t)+(1-\epsilon)p(t)$. Where $\epsilon$ is the degree of smoothing and is a real number between 0-1.

Because we see the word *meat* with other words located in the category of FOOD, we should give some weight in the hierarchy to sense it as FLESH rather than ESSENCE. Now we show how the smoothing method we explained above helps us to do this. Before smoothing, as we can see in figure 1.1 the probability of reaching to the word *meat* in both directions of FOOD and COGNITION is the same and equal to 1/8. If we take $u(t)$ as $\{1/4, 1/4, 1/4, 1/4\}$ and $\epsilon = 1/(c+1)$, where $c$ is the number of states, then the total count of the FOOD will be 31/2 while for COGNITION is 1/2. In other words, the probability of 1 in transition from COGNITION to ESSENCE has been smoothed, and therefore helps us to categorize *meat* in the FOOD class.

**Sense Balancing.** In the smoothing part we made one bias and categorized *meat* as Food. There are some undesirable cases which arise. For example in figure 1.2 we see a little difference than the figure 1.1. Because there are two paths from FOOD to *meat*, therefore in training 2/3 of the count go down the FOOD side; thus two senses of *meat* under FOOD will be preferred.

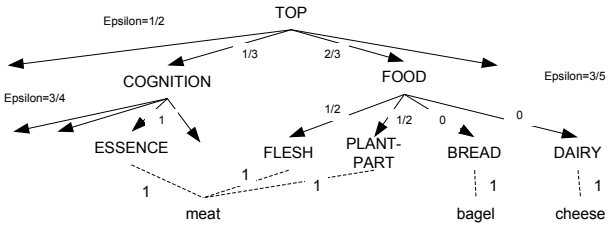


**Figure 1.2.** Imbalanced Senses.

To solve this problem two things should be done. First the count for the transition in the prefix path have to be reduced. So the transition probability from state $i$ to $j$ is as follow $\hat{E}_w(X_{i\to j}) = \frac{E_w(X_{i\to j})}{D(j,w)}$. Where, $D(j,w)$ is the number of distinct paths from state $j$ to generate word $w$. Second, we should reduce the probability of the paths to the sister sense of the phantom sense (e.g., COGNITION → ESSENCE) which is discussed with more details in [1]

**Length and Width Balancing.** It turns out that shorter paths are preferred to longer ones and paths that go through states with few existing transition are preferred to one that go through state with many. We can solve this problem by balancing the hierarchy. All paths that result in generating a symbol should be of the same length and all distributions should contain the same number of members. To do that we modify the uniform distribution for smoothing and we use the largest distribution ( $u_{max}$, which corresponds to the state with largest number of exiting transitions). This modification worsen those paths with few exiting transitions. To do length balancing, extra states and nodes are added to the shorter paths.

**Results.** The results of this algorithm didn't outperform the state-of-art methods for word ambiguity evaluation.

## 1.2 Catching the Drift: Probabilistic Content Models, with Applications to Generation and Summarization [2]

### 1.2.1 General Idea

Articles have many different topics. For example one article discusses about earthquake, and another one about medical diagnosis, and etc. Each of these contexts are called domain. In every domain there are many different concepts that are usually covered. For example in an article about earthquake, casualties, rescue process and geological concepts are usually discussed. In this paper, the goal is that to find this concept transition within an specific article.

Topic and topic transition within an article is called *content* in this paper. To find this topic transition in an article, we should at first divide the text into different documents (clusters) and talk about the topic of each cluster separately. The topic of each cluster is determined by *distributional* view. The idea is based on the fact that "various types of recurrence patters seem to characterize various types of discourse". This method can reduce human effort for recognition of the topics.

For better illustration of the whole process see the algorithm 2.

The rest of this scribe is categorized as follow. Algorithm 2 is explained in the section 1.2.2, and in section 1.2.3 application of this algorithm is studied.

**Algorithm 1** Algorithmic representation of the outline of paper.

1: Initialize clusters $c_1, c_2, ..., c_m$ using the word bigrams and cosine metric for similarity of sentences.
2: Initialize HMM using $c_1, c_2, ..., c_m$, to build states $s_1, s_2, ..., s_m$, estimate emission and transition probabilities.
3: Find Viterbi path for each document.
4: Re-cluster sentences into $c_1, c_2, ..., c_m$ using Viterbi path.
5: go to the line 2

### 1.2.2 Model Construction

In the first line of the algorithm 2, $K$ clusters are created via complete-link clustering, measuring sentence similarity by the cosine metric using word bigrams. Clusters having lower than $T$ sentences are merged into a new cluster called *etcetera*, which can be supposed as a cluster containing all outliers. The final number of clusters is represented by $m$. The $c_m$ is the *etcetera* cluster.

Now in the second line of algorithm 2, we assume an HMM with insertion state $s_1 s_2 ... s_m$, where each state $s_i$ is related to a cluster $c_i$. After determining the structure of the HMM we should estimate the transition and emission probabilities in the HMM.

As we can see in the line 1 of the algorithm 2, we use the bigram language model. As a result, the probability of generating a sequence $w_1 w_2 ... w_n$ will be $p_s(x) = \prod_{i=1}^{n} p_s(w_i|w_{i-1})$. Where $p_s(w_i|w_{i-1})$ is the probability of generating the word $w_i$ in the state $s$ with this condition that the last word generated is $w_{i-1}$. $p_s(w_i|w_{i-1})$ itself can be computed as follow.

$$p_s(w'|w) = \frac{f_c(ww') + \delta_1}{f_c(w) + \delta_1|V|}, \tag{1.1}$$

where $f_c(ww')$ is the frequency of the sequence $ww'$ in the cluster $c$, such that cluster $c$ corresponds to the state $s$. $|V|$ is the size of the vocabulary $V$. Also $\delta_1$ is usually a small value like 0.05. The idea behind using $\delta_1$ in the nominator of the equation (1.1) is that not observing the sequence $ww'$ in the training data doesn't necessarily mean that the probability of $p_s(w'|w)$ is zero. Therefore, to avoid the zero probability and smoothing the probability distribution function we use this idea. The constant $\delta_1|V|$ in the denominator of the equation (1.1) is for normalizing the probability distribution function

over the state $s$.

We want that state *etcetera* ($s_m$) to cover all unseen topics, therefore its distribution should be complement of the other topics. So we can write the following equation.

$$p_{s_m}(w'|w) = \frac{1 - \max_{i:i<m} p_{s_i}(w'|w)}{\sum_{u \in V} 1 - \max_{i:i<m} p_{s_i}(w'|w)}, \tag{1.2}$$

After computing the emission probabilities we should calculate the transition probabilities between states. The following equation was used for computing transition probabilities [2].

$$P(s_j|s_i) = \frac{D(c_i, c_j) + \delta_2}{D(c_j) + \delta_2 m}, \tag{1.3}$$

where $D(c_i, c_j)$ is the frequency of observing a sentence from $c_i$ proceeds by a sentence from $c_j$. $D(c_j)$ is the number of sentences from cluster $c_j$ observed in all the training documents [1].

The initial clustering of this paper ignores the sentence ordering, while in some cases two documents with the same word distribution could have two different topics. As an example usually the words at the end of each document are introducing the next topic, and similarly the first words of each document are the remaining words of the previous topic. One idea which was used in this paper is that to find the topic transition by running Viterbi algorithm on the current learned HMM, and then repartitioning the original article into different clustering. This idea is clearly shown in the line 5 of the algorithm 2. The process is usually finished with predefined number of iterations or reaching to an acceptable accuracy.

### 1.2.3 Task Evaluation

The technique explained in the section 1.2.2 is used in two different applications, information ordering for text generation and information selection for single-document summarization [2].

**Information ordering** is also essential to multi-document text summarization and concept to text generation. In [2] many documents are used

[1] In [2] $D(c_i, c_j)$ is defined as the number of documents in which a sentence from $c_i$ proceed one from $c_j$, which can't be correct unless each document at most contains one sentence from a cluster.

to train the content model, and the most probable ordering (most probable path) is chosen as the appropriate ordering.

**Extractive summarization** is another application of the proposed algorithm. For this purpose the content model of an article is build using the introduced algorithm. We need to learn which topics should appear in the text summary. The algorithm is as follow. At first we use our trained HMM to tag all the sentences in the training summary and original articles with a topic (this tag is called V-topic). Then, for each state $s$ such that at least three full training-set articles contained V-topic $s$, we compute the probability that the state generates sentences that should appear in a summary. This probability is estimated by simply (1) counting the number of document-summary pairs in the parallel training data such that both the originating document and the summary contain sentences assigned V-topic, and then (2) normalizing this count by the number of full articles containing sentences with V-topic $s$.

### 1.2.4 Experiments

For evaluation of this system 5 different domains are chosen. In each domain 100 articles for training, 100 articles for test and 20 articles for development set which is used for parameter tuning. The mentioned algorithm has 4 different parameters, two for probability distributions $(\delta_1, \delta_2)$ and 2 parameters for clustering part. Development set using Powells grid search suggested to choose $\delta < 0.0000001$.

**Ordering Experiment.** The intent behind our ordering experiments is to test whether content models assign high probability to acceptable sentence arrangements. There are two problems in this experiment.

- For a single document with $N$ word, there are $N!$ orderings, and therefore asking for human to test the result of ordering is impossible. But at least we know that the original ordering in the training data is correct which gives us OSO (original sentence order) measure. Using the content model trained we can compute the probability of each ordering and rank all the orderings. 0 stands for the best and $N! - 1$ is the worst ordering.

- Many state-of-the-art methods (e.g. Lapata) don't compute all the ordering of the sentences. Therefore, we use the metric $\tau(\sigma)$ in the equation (1.4) to rank all the orderings and compare them with OSO.

$$\tau(\sigma) = 1 - 2\frac{S(\sigma)}{\binom{N}{2}}, \tag{1.4}$$

where $S(\sigma)$ is the number of swaps of adjacent sentences necessary to rearrange $\sigma$ into the OSO. This metric is from -1 to 1.

**Results.** They compared content model with Lapata and bigram and showed that in ordering experiment it outperforms both of them. Another experiment was that with increasing the training data no over-fitting occurs and always the accuracy of OSO increases.

**Text summarization experiment.** For evaluation of text summarization two questions should be answered. 1) Are the summaries produced of acceptable quality, in terms of selected content? 2) Does the content-model representation provide additional advantages over more locally-focused methods?

To answer the first question we compare our method with lead baseline, which chooses $\ell$ sentences of the text and claims that most of the text summarization techniques can't beat this base line. To answer the second question we introduce other method called sentence classifier where the problem is converted to a two-class classification of sentences. The task is to decide whether each sentence is "in" the summary or "out" of it. The features for classification is the place of the sentence in the document (first, middle, end) and the bigrams within the sentence. Experiments show that the proposed algorithm outperforms both of these methods. Again with increasing the training data the summarization accuracy increases showing there is no over-fitting in the algorithm.

## 1.3 Bayesian Learning of Probabilistic Language Models [3]

### 1.3.1 General Idea

In this paper we mainly discuss on the structure of the HMM rather than parameter estimation (emission and transition probabilities), and the main question is that how many states we should consider for an HMM. We try to find an acceptable answer for this question using Bayesian probabilistic models. Assume our training data is $X$, and the model we are searching for is

$M$. The Bayesian inference says that our model should maximizes $P(M|X)$ in the following equation.

$$P(M|X) = \frac{P(X|M)P(M)}{P(X)}, \tag{1.5}$$

where $P(X|M)$ is likelihood, $P(M)$ is called prior and $P(M|X)$ is posterior. In Bayesian inference we maximize the posterior while in maximum likelihood we maximize the probability $P(X|M)$.

There is a duality between Bayesian inference and minimum description length principle in information theory. The denominator in the equation (1.5) in independent of the model $M$, and we want to find a model that maximizes $P(M|X)$, therefore we can remove the denominator and write the following equation.

$$P(M|X) = P(X|M)P(M), \tag{1.6}$$

which itself is equivalent to minimizing equation (1.7).

$$-\log P(M|X) = -\log P(X|M) - \log P(M), \tag{1.7}$$

In information theory the negative logarithm of a probability of random variable is the number of bits needed to store that random variable. Also in minimum description length (MDL) principle, we are interested in hypothesis which are simpler and we can represent them we fewer number of bits. Therefore MDL and Bayesian inference both suggest we should choose the models which are simpler. There is another philosophical idea which is called "Occam's Razor" saying that usually those hypothesis which are simpler have better generalization than the others. Therefore, we should find an HMM which is both simple and accurate in terms of fitting to the training data.

### 1.3.2 Priors for multinomial parameters

In Bayesian inference the prior distribution plays the role of smoothing in maximum likelihood inference. Let we have multiple of parameters. We can show all the parameters in a vector $\boldsymbol{\theta} = (\theta_1, ..., \theta_n)$. A standard prior for multinomials is Dirichlet distribution which is shown as follow:

$$P(\boldsymbol{\theta}) = \frac{1}{B(\alpha_1, ..., \alpha_n)} \prod_{i=1}^{n} \theta_i^{\alpha_{i-1}}, \tag{1.8}$$

where

$$B(\alpha_1, ..., \alpha_n) = \frac{\Gamma(\alpha_1)...\Gamma(\alpha_n)}{\Gamma(\alpha_1 + ... + \alpha_n)}$$

One interesting feature of Dirichlet distribution is that it is conjugate prior. In other words, when the likelihood probability is multiplied by it, the likelihood probability preserves its shape and template and just the parameters of it are changed. For example the likelihood probability depends on the count of each event $c_1, c_2, ..., c_n$, and is computed as follow.

$$P(c_1, c_2, ..., c_n|c, \boldsymbol{\theta}) = \prod_{i=1}^{n} \theta_i^{c_i} \tag{1.9}$$

Therefore the product of the likelihood in (1.9) and prior in (1.8) yields:

$$P(\boldsymbol{\theta}|c_1, c_2, ..., c_n) = \frac{1}{B(c_1 + \alpha_1, ..., c_n + \alpha_n)} \prod_{i=1}^{n} \theta_i^{c_i+\alpha_i-1} \tag{1.10}$$

And as we can see the posterior distribution in (1.10) has the same format as the likelihood in (1.9). Also it is clear that the Dirichlet has added the values $\alpha_i$ to the original counts $c_i$, and in other words has modified each of $c_i$ according to its prior knowledge.

Now consider an especial case where we have only two parameters $\theta_1$ and $\theta_2$. In this case one parameter can be written by another where $\theta_1 = p, \theta_2 = 1 - p$. The MAP (Maximum Posteriori) is as follow.

$$\hat{\theta}_i = \frac{c_i + \alpha_i - 1}{\sum_j c_j + \alpha_j - 1}$$

It is clear that for values $\alpha_i = 1$ the prior is uniform, for values $\alpha_i < 1$ the MAP estimation suggests extreme values (e.g. $\theta_1 = 1, \theta_2 = 0$), and for values $\alpha_i > 1$ it suggests that these parameters to be considered equal [3].

### 1.3.3 HMMs Structural vs. Parameter Priors

A global prior model for a hidden Markov model can be written as follow.

$$P(M) = P(M_G) \prod_{q \in Q} P(M_S^{(q)}|M_G) P(\theta_M^{(q)}|M_G, M_S^{(q)}),$$

where $P(M_G)$ is a prior for global aspects of the model structure (including, e.g., the number of states), $P(M_S^{(q)})$ is a prior contribution for the

structure associated with state $q$, and $P(M_S^{(q)}|M_G)$ is a prior on the parameters (transition and emission probabilities) associated with state $q$. Also $P(\theta_M^{(q)}|M_G, M_S^{(q)})$ is the probability of transition and emission out of the state.

### 1.3.4 HMM Structure Determination

After choosing a set of priors and prior parameters, it is conceptually straightforward to modify the simple likelihood-based algorithm to accommodate the Bayesian approach. The best-first HMM merging algorithm takes on the following generic form.

**Algorithm 2** Best-First Merging (Batch Version).

1: Build the initial, maximum-likelihood model $M_0$ from the dataset $X$.
2: **for** $i = 0$ **do**
3: Compute a set of candidate merges $K$ among the states of model $M_i$.
4: For each candidate $k \in K$ compute the merged model $k(M_i)$, and its posterior probability $P(k(M_i)|X)$.
5: $k^* = \arg\max_k P(k(M_i)|X)$
6: **if** $P(M_{i+1}|X) < P(M_i|X)$ **then**
7: **return** $M_i$
8: **end if**
9: $i \leftarrow i + 1$
10: **end for**

Algorithm 2 suggests that we should continue merging until the posterior probability can't be improved.

The complexity of the line 3 is $O(|Q|^2)$, because each of two states should be chosen to test whether we can merge them or not. One approach for reducing this complexity is clustering the states and try to not merge those states which are in different clusters, because obviously there are so different. Another strategy is the online version of merging. The idea of online-merging is that by small amount of training data HMM structure is built and states are merged. In this way, the number of states is reduced so the complexity $O(|Q|^2)$ which is proportional to $|Q|$ automatically decreased.

# Bibliography

[1] Abney, S., and Light, M. Hiding a semantic hierarchy in a markov model.

[2] Barzilay, R., and Lee, L. Catching the drift: Probabilistic content models, with applications to generation and summarization. In *HLT-NAACL* (2004), pp. 113–120.

[3] Stolcke, A. *Bayesian Learning of Probabilistic Language Models*. PhD thesis, Berkeley, CA, 1994.