# Document Recovery from Bag-of-Word Indices

**Abstract**

## 1 Problem formulation

We seek a log-linear exponential distribution

$$p(w_1^l) = \frac{1}{Z} p_0(w_1^l) \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(w_1^l)\right\}$$

where $p_0(w_1^l)$ is the prior (discussed in §2), each $f_k(w_1^l)$ is a feature function (§3), and each $\lambda_k$ is a weight that needs to be learned (§4) from a training corpus. The normalizer

$$Z = \sum_{v_1^{l'}} p_0(v_1^{l'}) \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(v_1^{l'})\right\}$$

is impossible to compute but is not required for some tasks, e.g. classification or optimization.

## 2 Prior

We have two choices for the prior $p_0(w_1^l)$: (a) a uniform prior, or (b) an n-gram language model as the prior. If we use a uniform prior, then we would use the log of the n-gram language model as a feature function.

Using a uniform prior is theoretically appealing, because it is neutral. In contrast, it is unclear in what sense an n-gram language model is prior to other language models we may incorporate as feature functions. However, there is no practical difference, since

$$p_0(w_1^l) \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(w_1^l)\right\}$$
$$= \exp\left\{\lambda'_0 \log p_0(w_1^l) + \sum_{k=1}^{K} \lambda'_k f_k(w_1^l)\right\}$$

if

$$\frac{1}{\sum_{k=1}^{K} \lambda_k} = \frac{\lambda'_0}{\sum_{k=1}^{K} \lambda'_k}$$

## 3 Feature functions

In addition to an n-gram LM score, we can try the following feature functions.

### 3.1 Class-based n-gram LM

First, we can try a class-based n-gram LM score:

$$f_k(w_1^l) \equiv p(C(w_l)|C(w_1^{k-1}))$$
$$= \sum_{i=1}^{l} p(C(w_i)|C(w_{i-n+1}^{i-1}))$$

where $C(w)$ is the class of $w$. But how should we define $C(\cdot)$? (a) We can use part-of-speech (POS) classes. (b) We can infer classes by clustering based on co-occurrence, as in (Brown et al., 1992). (c) We can define classes according to the Google 1T unigram count; in other words, a word $w$ is in class $C_i$ if

$$\beta^{i-1} \leq c(w) < \beta^i$$

where $c(\cdot)$ is the unigram count and $\beta$ controls the class boundaries ($\beta = 10$ might be reasonable). These three class definitions are not mutually exclusive: we could use all three classes in the same model, but in different feature functions.

Once we have defined $C(\cdot)$, we need to learn the probability distribution $p(C(\cdot))$. One question is what training corpus to use: (a) the Google 1T n-gram corpus, or (b) some other corpus that contains complete documents or sentences. If we use the Google 1T corpus, we will use a stupid-backoff LM; if we use another corpus, we may use modified Kneser-Ney smoothing instead. Google's 1T corpus is much larger than any other corpus currently available, but it may be more difficult to use than a corpus of complete documents. For example, state-of-the-art POS taggers such as the Stanford Log-linear POS Tagger (Toutanova and Manning, 2000; Toutanova et al., 2003) are designed to tag whole sentences, not individual n-grams, and may not have high accuracy on the Google 1T corpus. Similarly, it may cause problem to infer clusters based on co-occurrence within n-grams, and then use the same n-grams to estimate the probabilities: this may bias the probabilities against sequences containing a diversity of classes. One solution could be to use a smaller corpus to learn the clusters and the Google 1T corpus to learn the probabilities. This will leave some words in the Google 1T corpus without classes; these words could be marked as UNK, or their n-grams could be ignored. Another solution, for either POS or clustered classes, would be to use a sampling procedure (e.g., (Rosenfeld, 1997), §3.1) to generate a large corpus of complete sentences from the Google 1T n-grams using a word-based LM. Then this corpus can be tagged or clustered and class-based n-grams generated from it. Unfortunately, it may take a prohibitive amount of time to sample a sufficiently large number of sentences.

### 3.2 Fraction of words in a class

Instead of using classes in an n-gram language model, it may be more useful to consider the fraction of words in each class in a given document. We define a separate feature function for each class $C$:

$$f_k(w_1^l) = p\left(\frac{1}{l}\sum_{i=1}^{l} 1(C(w_i) = C)\right) = p(r(w_1^l))$$

where $1(\cdot)$ is 1 if its argument is true and 0 otherwise, and $r(w_1^l)$ is the fraction of class-$C$ words in $w_1^l$.

We could estimate $p(\cdot)$ using the binomial MLE based on some training corpus $\mathcal{D}$ (success $\equiv$ a word is in class $C$; failure $\equiv$ a word is in some other class); that is, we can let the probability of a given document having a given fraction of class-$C$ words be the percentage of documents in the training corpus that have exactly that fraction of class-$C$ words:

$$p(r(w_1^l)) = \frac{1}{|\mathcal{D}|}\sum_{v_1^{l'} \in \mathcal{D}} 1(r(v_1^{l'}) = r(w_1^l))$$

However, this estimate will suffer from data sparseness. For example, in a given test document, $6/50$ of the words may have class $C$; in our training set, perhaps no documents will have exactly $6/50$ class-$C$ words, but several documents may have $6/51$ or $11/100$ class-$C$ words. The binomial estimate will improperly assign probability of 0 to the test document.

A better estimate will use smoothing to avoid this problem. For example, we can use the normal approximation to the binomial as follows. Let the MLE mean and variance be

$$\bar{r} = \frac{1}{|\mathcal{D}|}\sum_{v_1^{l'} \in \mathcal{D}} r(v_1^{l'})$$

$$s^2 = \frac{1}{|\mathcal{D}|}\sum_{v_1^{l'} \in \mathcal{D}} (r(v_1^{l'}) - \bar{r})^2$$

(To obtain an unbiased estimator for the variance, we can use $\frac{1}{|\mathcal{D}|-1}$ instead of $\frac{1}{|\mathcal{D}|}$.) Then the normal

$$p(r(w_1^l)) = \mathcal{N}(r(w_1^l)|\bar{r}, s^2)$$

is the MLE.

For this type of feature function, I conjecture that POS and count-based classes will be

useful, but co-occurrence cluster classes will not, because the distribution of co-occurrence classes will exhibit too much variation among sentences.

### 3.3 Spacing of words in a class

We define a separate feature function for each class $C$:

$$f_k(w_1^l) = p(\bar{d}(w_1^l), s^2(w_1^l))$$

where $\bar{d}(w_1^l)$ and $s^2(w_1^l)$ are the sample mean and variance of the distance between consecutive elements of class-$C$ words in $w_1^l$:

$$d(j) = \mathsf{i}_{j+1} - \mathsf{i}_j, \quad j = 1, \ldots, |\mathsf{i}| - 1$$

and $\mathsf{i} = \{i : C(w_i) = C, i = 1, \ldots, l\}$. As before, the probability distributions can be estimated using the bivariate normal MLE on a training corpus. To calculate the MLE, we let

$$\boldsymbol{\theta}(v_1^{l'}) = \left(\bar{d}(v_1^{l'}) \;\; s^2(v_1^{l'})\right)^\top$$

Then the MLE mean and covariance of $\boldsymbol{\theta}$

$$\bar{\boldsymbol{\theta}} = \frac{1}{|\mathcal{D}|} \sum_{v_1^{l'} \in \mathcal{D}} \boldsymbol{\theta}(v_1^{l'})$$

$$\hat{\Sigma} = \frac{1}{|\mathcal{D}|} \sum_{v_1^{l'} \in \mathcal{D}} (\boldsymbol{\theta}(v_1^{l'}) - \bar{\boldsymbol{\theta}})(\boldsymbol{\theta}(v_1^{l'}) - \bar{\boldsymbol{\theta}})^\top$$

(Again, we could use the sample covariance instead of the MLE.) Finally let

$$p(\bar{d}(w_1^l), s^2(w_1^l)) = \mathcal{N}(\boldsymbol{\theta}(w_1^l) | \bar{\boldsymbol{\theta}}, \hat{\Sigma})$$

Detailed classes may prove ineffective for this type of feature function. Instead, we could use, for example, the following three classes: sentence boundaries (containing "<S>", "</S>", and the bigram "</S> <S>"), stopwords, and non-stopwords. It may prove useful to include 1 and $l$ in the set i for all classes.

### 3.4 Parser

We could use an off-the-shelf parser as a feature function. Depending on the parser, the feature could be $0 =$ parses, $1 =$ doesn't parse, or it could be more fine-grained. We only parse up to the end of the last complete sentence:

$$f_k(w_1^l) = \text{score}(w_1^j)$$

where $j$ is the index of the last </S> in the document.

### 3.5 Document length

For stopwords and indicator BOWs, we do not know the true document length. We define a feature function that scores the current length of the document

$$f_k(w_1^l) = p(l|\mathbf{x}) = \mathcal{N}(l|\mathbf{1}^\top\mathbf{x}; \bar{l}, s^2)$$

where $\mathbf{x}$ is the BOW, $\mathbf{1}$ is the ones vector, and $\bar{l}$ and $s^2$ are the MLE mean and variance of document length, given a BOW of length $\mathbf{1}^\top\mathbf{x}$, on a training corpus.

This approach may suffer from data sparseness, since we need to estimate the mean and variance separately for all possible BOW lengths. A better solution is as follows. First we use SVM regression on a training corpus to estimate document length $\hat{l}(\mathbf{1}^\top\mathbf{x})$ given BOW length. Then we estimate the variance of the ratio of the estimated length to the true length, assuming a unit mean:

$$s^2 = \frac{1}{|\mathcal{D}| - 1} \sum_{v_1^{l'} \in \mathcal{D}} \left( \frac{\hat{l}(\mathbf{1}^\top\mathbf{x}(v_1^{l'}))}{l} - 1 \right)^2$$

We use as our estimate the normal whose mean is the estimated document length $\hat{l} = \hat{l}(\mathbf{1}^\top\mathbf{x})$, and whose variance is $s^2$ scaled by the estimated document length:

$$p(l|\mathbf{x}) = \mathcal{N}(l|\mathbf{1}^\top\mathbf{x}; \hat{l}, \hat{l}\, s^2)$$

## 4 Feature weights

We present several different ways to learn the feature weights $\lambda_1^K = \lambda_1, \ldots, \lambda_K$.

### 4.1 Convex methods

We choose the feature weights so as to maximize the probability of a training corpus $\mathcal{D}$:

$$\hat{\lambda}_1^K = \arg\max_{\lambda_1^K} \left\{ \sum_{\mathbf{d} \in \mathcal{D}} p(\mathbf{d}|\lambda_1^K) \right\}$$

where $p(\mathbf{d}|\lambda_1^K)$ is our exponential distribution using the feature weights $\lambda_1^K$, and $\mathbf{d} = v_1^{l'}$. As (Och and Ney, 2004) point out, this is a convex optimization problem and can be solved using standard methods such as gradient descent.

### 4.2 Generalized iterative scaling

Generalized iterative scaling ((Rosenfeld et al., 2001), §2.2, and originally (Darroch and Ratcliff, 1972)) is an algorithm to learn the feature weights that takes advantage of the specific structure of our problem. We initialize each $\lambda_i$ arbitrarily, then iteratively set

$$\lambda_i \leftarrow \lambda_i + \frac{1}{F_i} \log \frac{E_{\tilde{p}}[f_i]}{E_p[f_i]}$$

where

$$E_p[f_i] = \sum_{\mathbf{d} \in \mathcal{D}} p(\mathbf{d}) f_i(\mathbf{d})$$

is the expected value of feature $f_i$ under the current model,

$$E_{\tilde{p}}[f_i] = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{d} \in \mathcal{D}} f_i(\mathbf{d})$$

is the value of feature $f_i$ observed in the training corpus, and $F_i$ is a parameter that controls the rate of convergence. Each $\lambda_i$ will converge when $E_{\tilde{p}}[f_i] = E_p[f_i]$, since in that case $\log \frac{E_{\tilde{p}}[f_i]}{E_p[f_i]} = 0$.

Unfortunately, it is impossible to compute $E_p[f_i]$, since it is impossible to compute the normalization factor $Z$ of $p(w_1^l)$ (§1). Instead we estimate $E_p[f_i]$ by sampling. (Rosenfeld et al., 2001), §3.1, test Gibbs sampling, Metropolis sampling, independence sampling, and importance sampling for this purpose; they find that independence and importance sampling work well, but Gibbs sampling is too slow and Metropolis sampling gives poor results.

### 4.3 Minimum error rate training

Instead of setting the feature weights to maximize probability with respect to a training corpus $\mathcal{D}$, we can set them to minimize error rate $L$ with respect to $\mathcal{D}$ (Och, 2003):

$$\hat{\lambda}_1^K = \arg\min_{\lambda_1^K} \left\{ \sum_{\mathbf{d} \in \mathcal{D}} \sum_{i=1}^{N} L(\mathbf{d}, \hat{\mathbf{d}}_i) \right\}$$

where $\{\hat{\mathbf{d}}_i\}$ are the $N$-best documents recovered from $\mathbf{d}$'s BOW using A* search with $p(w_1^l|\lambda_1^K)$ as the score function. This optimization problem presents two difficulties: (a) it is not convex—in fact, it has a very large number of local minima (see (Och, 2003), figure 1), and (b) even one iteration over the $\mathbf{d} \in \mathcal{D}$ may require hours or days for a large training corpus, because of the A* search. The first problem can be mitigated by smoothing:

$$\hat{\lambda}_1^K = \arg\min_{\lambda_1^K} \left\{ \sum_{\mathbf{d} \in \mathcal{D}} \sum_{i=1}^{N} L(\mathbf{d}, \hat{\mathbf{d}}_i) \frac{p(\hat{\mathbf{d}}_i)^\gamma}{\sum_{i=1}^{N} p(\hat{\mathbf{d}}_i)^\gamma} \right\}$$

for some smoothing paramater $\gamma$.

However, even this smoothed optimization problem is not convex. (Och, 2003), §5, following (Papineni, 1999), proposes a procedure also used and elucidated by (Smith and Eisner, 2006), §6. Let $\mathcal{B}(\mathbf{d})$ be a list of $(c, l')$-tuples, where $c = c(\mathbf{d}, \hat{\mathbf{d}})$ is the precision and $l' = l'(\hat{\mathbf{d}})$ is the length of some document $\hat{\mathbf{d}}$ recovered from $\mathbf{d}$'s BOW, and all $\mathcal{B}(\mathbf{d})$ are initially empty. (Precision and length are used to compute BLEU.) We iterate as follows.

- For each $\mathbf{d} \in \mathcal{D}$, and for each document $\hat{\mathbf{d}}_i$ recovered from $\mathbf{d}$, if $(c(\mathbf{d}, \hat{\mathbf{d}}_i), l'(\hat{\mathbf{d}}_i)) \notin \mathcal{B}(\mathbf{d})$, add it.
- If no new tuples have been added to any $\mathcal{B}(\mathbf{d})$, we have found locally optimal feature weights $\lambda_1^K$.
- Otherwise, we update the weights and go to the first step.[1]

The feature weights can be initialized according to the maximum entropy estimate. To ameliorate the problem (b) above, iteration can be stopped after a fixed number of steps even if convergence has not been reached.

## References

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992.

[1] How? (Och, 2003)'s exposition is somewhat confusing. It requires finding the minimum of a piecewise linear function.

Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479.

J. N. Darroch and D. Ratcliff. 1972. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480.

Franz Josef Och and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Comput. Linguist.*, 30(4):417–449.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 160–167, Morristown, NJ, USA. Association for Computational Linguistics.

K. A. Papineni. 1999. Discriminative training via linear programming. In *ICASSP '99: Proceedings of the Acoustics, Speech, and Signal Processing, 1999. on 1999 IEEE International Conference*, pages 561–564, Washington, DC, USA. IEEE Computer Society.

Ronald Rosenfeld, Stanley F. Chen, and Xiaojin Zhu. 2001. Whole-sentence exponential language models: a vehicle for linguistic-statistical integration. *Computer Speech Language*, 15(1):55–73.

R. Rosenfeld. 1997. A whole sentence maximum entropy language model. *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on*, pages 230–237, Dec.

David A. Smith and Jason Eisner. 2006. Minimum risk annealing for training log-linear models. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 787–794, Morristown, NJ, USA. Association for Computational Linguistics.

Kristina Toutanova and Christopher D. Manning. 2000. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora*, pages 63–70, Morristown, NJ, USA. Association for Computational Linguistics.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 173–180, Morristown, NJ, USA. Association for Computational Linguistics.