Report For CS382 Homework 1
Implementation of Smoothing Algorithm for N-gram Language Model

Yanming Liu; ID: 518030910393

October 17, 2021

Declare All the implementation here are finished by myself without the help of open source tools.

1 Smoothing Algorithm

1.1 Hierarchical Dirichlet Language Model[1]

The hierarchical Dirichlet language model is a smoothed n-gram language model proposed by MacKay et al (1995)[1]. The article mainly discusses the prior and posterior distribution of n-gram language model parameters, and points out that the prior distribution can be a Dirichlet distribution.

Denote the language model parameters as $\mathbf{p} = (p_1, p_2, \cdots, p_n)$, (here $p_i = Pr(\mathit{some\ token}|\mathit{some\ context})$) before seeing the training data we have a prior distribution of $\mathbf{p}$:

$$P(\mathbf{p}|\alpha\mathbf{m}) = \frac{1}{Z(\alpha\mathbf{m})}\prod_{i=1}^{n} p^{\alpha m_i - 1}\delta(\sum_{i}^{n} p_i - 1) \equiv \texttt{Dirichlet}^{(n)}(\mathbf{p}|\alpha\mathbf{m}) \tag{1}$$

Here $Z(\alpha\mathbf{m})$ is a normalizing constant (can be calculated with explicit formulas), $\delta(.)$ is Dirac delta function, $\mathbf{m} = \mathbb{E}[\mathbf{p}]$ (so, $\sum_i m_i = 1$) and α is a positive scalar.

As far as I understand it, one of the advantages of using Dirichlet distribution as the prior distribution is that we can easily get a brief posterior distribution, after seeing the training corpus and getting the statistical information $\mathbf{F} = (F_1, F_2, \cdots, F_n)$, where F_i means the number of tokens group i:

$$\begin{aligned} P(\mathbf{p}|\mathbf{F}, \alpha\mathbf{m}) &= \frac{P(\mathbf{p}, \mathbf{F}, \alpha\mathbf{m})}{P(\mathbf{F}, \alpha\mathbf{m})} = \frac{P(\mathbf{F}|\mathbf{p})P(\mathbf{p}|\alpha\mathbf{m})}{P(\mathbf{F}|\alpha\mathbf{m})} \\ &= \frac{1}{P(F|\alpha\mathbf{m})Z(\alpha\mathbf{m})}\textstyle\prod_i^n p_i^{F_i}\prod_i^n p_i^{\alpha m_i - 1}\delta(\sum_i p_i - 1) \\ &= \frac{1}{Z(\mathbf{F}, \alpha\mathbf{m})}\textstyle\prod_i^n p_i^{F_i + \alpha m_i - 1}\delta(\sum_i p_i - 1) = \texttt{Dirichlet}^{(n)}(\mathbf{p}|\mathbf{F} + \alpha\mathbf{m}) \end{aligned} \tag{2}$$

Just change the Dirichlet distribution parameters from $\alpha\mathbf{m}$ to $\mathbf{F} + \alpha\mathbf{m}$. And according to a good property of Dirichlet distribution, $\mathbb{E}[p_i] \propto F_i + \alpha m_i$, so we have:

$$\mathbb{E}[p_i] = \frac{F_i + \alpha m_i}{\sum_{i'} F_{i'} + \alpha m_{i'}} \tag{3}$$

So, when predict, we use the predictive distribution:

$$P(i|j, \mathbf{F}, \alpha\mathbf{m}) = \frac{F_{i|j} + \alpha m_i}{\sum_{i'} F_{i'|j} + \alpha m_{i'}} = \frac{F_{i|j} + \alpha m_i}{F_j + \alpha} = \frac{F_j}{F_j + \alpha}\frac{F_{i|j}}{F_j} + \frac{\alpha}{F_j + \alpha} m_i \tag{4}$$

If we let $\lambda_j = \dfrac{\alpha}{F_j + \alpha}$, and $f_{i|j} = F_{i|j}/F_j$, we get

$$P(i|j, \mathbf{F}, \alpha\mathbf{m}) = (1 - \lambda_j)f_{i|j} + \lambda_j m_i \tag{5}$$

Therefore, when we assume that the prior distribution of the parameters is Dirichlet distribution, we can derive the following:

- Posterior distribution is similar (almost the same) to the one of interpolation method.
- The interpolation coefficient $\lambda_j = \alpha/(F_j + \alpha)$ can be calculated theoretically and not need to be tuned on dev set. And this result confirms a conclusion from experiments: the tokens group with the same frequency is better to use the same lambda.
- Explicit assumption of the prior distribution allows us to even calculate α and $\mathbf{m}$.

It can be derived by Bayes formula that the optimal parameters α and $\mathbf{m}$ maximize $P(F|\alpha\mathbf{m})$. And we can calculate $\alpha, \mathbf{m}$ by analysis method (the deriving process is complicated, we omit it here): Let N_{Fi} be the number of contexts j such that $F_{i|j} \geq F$, compute (when f is big enough, N_{fi}=0, so only need to calculate finite items):

$$G_i = \sum_{f=2}^{\infty} \frac{N_{fi}}{f-1}; \quad H_i = \sum_{f=2}^{\infty} \frac{N_{fi}}{(f-1)^2} \tag{6}$$

We define $K(\alpha) = \sum_j \log\left(\frac{F_j + \alpha}{\alpha}\right) + \frac{F_j}{2\alpha(F_j + \alpha)}$

The optimal parameters $\mathbf{u}^* = \alpha^* \mathbf{m}^*$ should satisfy:

$$u_i^* = \frac{2N_{1i}}{K(\alpha^*) - G_i + \sqrt{(K(\alpha^*) - G_i)^2 + 4H_i N_{1i}}} \tag{7}$$

And we know, $\sum_i m_i = 1$, so $\alpha = \sum_i u_i$. So far, we can calculate $\alpha, \mathbf{u}$ in an iterative way: calcuate $\mathbf{u}$ from α according to Eq.7 and then $\alpha \leftarrow \sum_i u_i$.

1.2 Tuned Interpolation Language Model

This model is an intuitive one. From **section 1.1** we can see that the smoothed distribution should be in such a family of distribution:

$$P(w_i|w_{i-n+1}^{i-1}) = (1 - \lambda_{w_{i-n+1}^{i-1}}) f_{i|j} + \lambda_{w_{i-n+1}^{i-1}} P(w_i|w_{i-n+2}^{i-1})$$

Here $f_{i|j}$ is directly from the statistical frequency, $P(w_i|w_{i-n+2}^{i-1})$ is used as the prior distribution, that is, m_{w_i} in Eq.5. We do not calculate λ explicitly here, instead we learn it from the dev set with maximum likelihood estimation:

$$\log P(F^{dev}; \lambda) = \sum_{j \in \{w_{i-n+1}^{i-1}\}} \sum_{i \in \{w_i\}} F_{i|j}^{dev} \log\Big((1 - \lambda_j) f_{i|j}^{train} + \lambda_j P(i|j^{backoff})\Big) \tag{8}$$

$$\frac{\partial}{\partial \lambda_j} \log P(F^{dev}; \lambda) = \sum_i F_{i|j}^{dev} \frac{P(i|j^{backoff}) - f_{i|j}^{train}}{(1 - \lambda_j) f_{i|j}^{train} + \lambda_j P(i|j^{backoff})} \tag{9}$$

It is obviously that $\partial^2 \log P/\partial \lambda_j^2 < 0$, so $\partial \log P/\partial \lambda_j$ is monotonous about λ_j, so the optimal λ^* can be calculated by solving the equation $\partial \log P/\partial \lambda_j = 0$ with bisection.

Inspired by the method in **section 1.1**, tokens group j with the same frequency F_j uses the same λ_{F_j} as parameter. It solves the following problems:

- Overfitting (good performance on the dev set but poor performance on the test set).
- When a tokens group j does not appear in the dev set, we can not assign a proper value for λ_j with Eq.9.

2 Experiment

2.1 Experiment Settings

- Model: I test the two models mentioned in **Section 1**.
- Model parameters: I test $n = 2$ and $n = 3$.
- Corpus: Use the given corpus train_set.txt, dev_set.txt and test_set.txt.
  - For model in **Section 1.1**, since it uses only train set and no need for dev set, so I use both train_set.txt and dev_set.txt as the train corpus, and use test_set.txt as the test corpus.

- For model in **Section 1.2**, I use train_set.txt as the train corpus, that is, do counting on train_set.txt, use dev_set.txt as the dev corpus, that is, do tuning on dev_set.txt, and use test_set.txt as the test corpus.

- Vocabulary: All the tokens appearing in the train corpus including '$< s >$' and '$< /s >$' and an additional '$< unk >$' token.
  - That is to say, here I did not use low-frequency words as the unknown. Instead, I used words that never appeared in train corpus as the unknown. In order to let the model see '$< unk >$' during counting, I manually insert n '$< unk >$' into the train corpus, which will not affect the correctness of the model, but will only slightly hurt performance.

- Iteration accuracy:
  - For model in **Section 1.1**, in order to ensure that the conditional distribution predicted by model is normalized, we set $\epsilon = 1e - 6$ when calcuate α and $\mathbf{m}$.
  - For model in **Section 1.2**, when I tune the parameter λ, I use $\epsilon = 1e - 3$.

2.2 Experiment Results

Table 1: Perplexity

	2-gram ppl	3-gram ppl
MacKay Method (**Section 1.1**)	571	760
Simpple Interpolation (**Section 1.2**)	527	421

The perplexity of the two models under different parameters n is shown in Table 1. The result generally meets my expectation.

MacKay method, i.e. the model in **Section 1.1**, is beautiful in theory, but in fact its performance is not very good (maybe my implementation is poor). The original paper uses a lot of analysis methods and approximations in order to get a beautiful explicit results, so the actual result may not be as good as theoretically expected.

The simple interpolation method, i.e. the one in **Section 1.2**, is just to maximum the likelihood on dev corpus (equivalent to minimizing ppl). But I found in experiments that the optimization effect is not very significant. Compared with a random λ, the optimal λ can only reduce the ppl by less than 10%.

I have compared my result with that of my classmate who used a similar algorithm, and I found that my ppl is slightly larger than his. However, I double-checked my implementation and did not find any obvious errors. And I have checked that the conditional distribution reported by my model is normalized.

References

[1] David JC MacKay and Linda C Bauman Peto. A hierarchical dirichlet language model. *Natural language engineering*, 1(3):289–308, 1995.