Home Work 1 – Language Models Report

Smoothing

No, the back-off model is not a valid probability distribution, because of the following reasons:

- Not use the MLE estimates with discount. So, there would be no probability mass to distribute to the lower order models.
- The normalizing factor for normalize the probability from N-Gram to the lower-order (N-1)-gram is not correct.
- For the second case, should also consider if pair w_{i-1} and w_{i-2} existed, when calculate the probability.
- For the third case, it's no meaning to have the normalizing factor since it

By fixing all those issues, the model should look like:

$$p_1(w_i|w_{i-2},w_{i-1}) = P^*_{ML}(w_i|w_{i-2},w_{i-1}) \qquad \text{if } w_i \in \mathcal{A}(w_{i-2},w_{i-1})$$

$$p_2(w_i|w_{i-2},w_{i-1}) = \begin{cases} \alpha(w_{i-1},w_{i-2})P^*_{ML}(w_i|w_{i-1}) & \text{if } w_i \in \mathcal{B}(w_{i-2},w_{i-1}) \text{ and } w_i \in \mathcal{A}(w_{i-1}) \text{ and } c(w_{i-2},w_{i-1})>0 \\ \alpha(w_{i-1},w_{i-2})\alpha(w_{i-1})P^*_{ML}(w_i) & \text{if } w_i \in \mathcal{B}(w_{i-2},w_{i-1}) \text{ and } w_i \in \mathcal{A}(w_{i-1}) \text{ and } c(w_{i-2},w_{i-1})=0 \end{cases}$$

$$p_3(w_i|w_{i-2},w_{i-1}) = P^*_{ML}(w_i) \qquad \text{if } w_i \in \mathcal{B}(w_{i-1})$$

In the formula, the discounted MLE probability P^*_{ML} is defined as following:

$$P^*_{ML}(w_n|w^{n-1}_{n-N+1}) = \frac{c^*(w^n_{n-N+1})}{c(w^{n-1}_{n-N+1})}$$

While the discounted (c^*) should be, where β should be the differentiate between the held-out data and future data:

$$c^*(v,w) = c(v,w) - \beta$$

Or the can use Good-Turning Discounting:

$$c^*(x) = (r+1)\frac{n_{r+1}}{n_r}$$

And the normalizing factor α should be:

$$\alpha(w^{n-1}_{n-N+1}) = \frac{1 - \sum_{w_n:c(w^{n-1}_{n-N+1})>0} P^*_{ML}(w_n|w^{n-1}_{n-N+1})}{1 - \sum_{w_n:c(w^{n-1}_{n-N+1})>0} P^*_{ML}(w_n|w^{n-1}_{n-N+2})}$$

So, the model will finally look like:

when $w_i \in \mathcal{A}(w_{i-2},w_{i-1})$:

$$p_1(w_i|w_{i-2},w_{i-1}) = P^*_{ML}(w_i|w_{i-2},w_{i-1}) = \frac{c^*(w_{i-2},w_{i-1},w)}{c(w_{i-2},w_{i-1})}$$

When $w_i \in \mathcal{B}(w_{i-2},w_{i-1})$ and $w_i \in \mathcal{A}(w_{i-1})$:

If $c(w_{i-2}, w_{i-1}) > 0$:

$$p_2(w_i|w_{i-2}, w_{i-1}) = \frac{1 - \sum_{w \in \mathcal{A}(w_{i-2}, w_{i-1})} P^*_{ML}(w|w_{i-2}, w_{i-1})}{\sum_{w \in \mathcal{B}(w_{i-2}, w_{i-1})} P^*_{ML}(w|w_{i-1})} P^*_{ML}(w_i|w_{i-1})$$

Else if $c(w_{i-2}, w_{i-1}) = 0$:

$$p_2(w_i|w_{i-2}, w_{i-1}) = \frac{1 - \sum_{w \in \mathcal{A}(w_{i-2}, w_{i-1})} P^*_{ML}(w|w_{i-2}, w_{i-1})}{\sum_{w \in \mathcal{B}(w_{i-2}, w_{i-1})} P^*_{ML}(w|w_{i-1})} \frac{1 - \sum_{w \in \mathcal{A}(w_{i-1})} P^*_{ML}(w|w_{i-1})}{\sum_{w \in \mathcal{B}(w_{i-1})} P^*_{ML}(w)} P^*_{ML}(w_i)$$

When $w_i \in \mathcal{B}(w_{i-1})$

$$p_3(w_i|w_{i-2}, w_{i-1}) = P^*_{ML}(w_i)$$

Language Model Classier

For the given document *x* and category label *y*, we can compute the posterior probability of one document's category $P(y|x)$, Where $y \in Categorization\ Set\ (C)$. The posterior probability can be computed by applying the Bayes rule:

$$P(y|x) = \frac{P(x|y)P(y)}{p(x)}$$

Because the documents for the Language Model should be independent from the category. Therefore, it can be ignored. So, the formula can be simplified to:

$$P(y|x) \propto P(x|y)P(y)$$

We can assume that all the words in the documents are independent (unigram model) from the category. So, we have:

$$P(x|y) = \prod_{i=1}^{n} P(w_i|y)$$

So, the probability of the category of one document can be write as:

$$P(y|x) \propto \prod_{i=1}^{n} P(w_i|y)\, P(y)$$

And the $P(y)$ can be estimated by the percentage of the training examples:

$$P(y) = \frac{N_y}{N}$$

where N is the total number of training documents and N_y is the number of training document in category *y*. Also, the $P(w_i|y)$ can be determined by:

$$P(w_i|y) = \frac{count(w_i, y)}{Ny}$$

Where $count(w_i, y)$ is the number of times that word w_i occurs within the training documents of category *y*.

To handle the zero-probability problem we can add the smoothing algorithm for the probability formula, such as, if we use Laplace smoothing (Add-one):

$$P(w_i|y) = \frac{1 + count(w_i, y)}{|V| + Ny}$$

Where the $|V|$ is the total number of vocabulary.

So, as a conclusion, we can use the unigram model as the Language Model to handle the text categorization problems.

Language Models

After running the program, I get the following data for perplexity scores of the unigram, bigram, trigram language models for training, dev, and test sets. And the following of this chapter will briefly describe the way I do it, and the results.

Unigram

During the implementation of the *Unigram LM*, I

- Add the **STOP** symbols for every sentence
- Mark all the words that count is less or equal to the threshold will be replace to **UNK**.

Table 1 Perplexity of Unigram

	Threshold of unknown words							
	0	1	3	5	10	20	30	50
Training Dataset	1106.34	868.10	646.75	528.90	377.33	242.71	179.74	113.66
Dev Dataset	6404.90[1]	765.90	590.53	491.96	361.52	236.63	175.45	111.60
Test Dataset	6473.15[2]	769.37	599.03	499.18	365.09	240.78	179.40	113.71

1. 33% (1930) of Zero Prob Sentences, the perplexity score is for other sentences
2. 32% (1850) of Zero Prob Sentences, the perplexity score is for other sentences

From the result, can find out:

- Without handle the out-of-vocabulary (OOV) words, there have one 3rd of the data from *Dev Set* and *Test Set* are zero prob sentences. Which prove the logic to handle OOV words is crucial
- By increase the threshold of unknown words, the perplexity score become lower and lower
- But this doesn't mean the Language Model is very good, because for the results form threshold as 30 or 50, although the perplexity is quite low compare to others, since most of the words will be consider as the **UNK**, it just generally become a **UNK** sentence and perplexity is just dependents on the average sentence length of the dataset.

Bigram

During the implementation of the *Bigram LM*, I:

- Implement it dependent on the *Unigram LM*
- After built the *Unigram LM* internally when record the count of the word pair, marked all the words that had been replaced in *Unigram LM* to **UNK** as the **UNK** word pair, such as (**UNK**, word), (word, **UNK**), (**UNK**, **UNK**)
- Add the **START** symbol in the start of the sentences
- The calculation of perplexity is ignored the zero prob sentence

Table 2 Perplexity of Bigram

	Threshold of unknown words			
	0	1	3	5
Training Dataset	44.04	45.39	44.71	43.35
Dev Dataset	1051848764911.70 94% (5450)	23042616306.69 92% (5326)	1253193042.48 90% (5225)	60404856.23 89% (5123)
Test Dataset	33219192266.87 93% (5351)	302000568.55 91% (5197)	28909914.55 89% (5086)	3421153.26 87% (4965)
	Threshold of unknown words			
	10	20	30	50
Training Dataset	40.05	34.44	30.35	24.26
Dev Dataset	1024662.01 85% (4897)	12829.45 77% (4438)	1872.42 70% (4069)	232.50 58% (3355)
Test Dataset	177388.18 83% (4745)	8906.52 75% (4324)	1787.41 69% (3975)	239.38 58% (3297)

From the result, can find out:

- By increase the threshold of unknown words, the perplexity score become lower and lower
- The perplexity of *Bigram LM* is much better than *Unigram LM* for *Training Set*, because the Bigram LM know more information of how to organize the sentence
- Even handled the OOV words, their still have a lot portion of the sentence from *Dev Set* and *Test Set* is the zero prob sentences. Because there may still has lots of the word pairs are not in the Training Set

Trigram

During the implementation of the *Trigram LM*, I:

- Implement it dependent on the *Bigram LM*
- After built the *Bigram LM* internally, marked all the words that had been replaced in *Unigram LM* to **UNK** as the **UNK** word pair also, such as: (word1, word2, **UNK**), (**UNK**, word2, **UNK**), (**UNK**, **UNK**, **UNK**)
- Add the **START1** and **START2** symbol in the start of the sentences

- The calculation of perplexity is ignored the zero prob sentence

Table 3 Perplexity of Trigram

	Threshold of unknown words			
	0	1	3	5
Training Dataset	4.45	4.85	5.37	5.75
Dev Dataset	6675263113237140.0 98% (5651)	20037563167434.20 97% (5610)	296955613133.07 96% (5578)	38929357752.57 96% (5554)
Test Dataset	1.23 98% (5618)	27501106372112.61 97% (5569)	76898089303.67 96% (5527)	3085444711.73 96% (5491)
	Threshold of unknown words			
	10	20	30	50
Training Dataset	6.39	7.12	7.51	7.80
Dev Dataset	820296229.76 95% (5503)	5993651.21 93% (5390)	255608.97 91% (5277)	7569.01 87% (5039)
Test Dataset	285962043.78 95% (5451)	2033587.25 93% (5335)	151626.39 91% (5222)	4893.16 87% (4971)

From the result, can find out:

- By increase the threshold of unknown words, the perplexity scores of *Training Set* is increase, which should mean the probability of **UNK** word pairs are affect the perplexity more
- Perplexity scores of *Dev Set* and *Test Set* is decreasing while increase the threshold of unknown words.
- The perplexity of *Trigram LM* is much better than *Bigram LM* for *Training Set*
- Even handled the OOV words, their still have even more sentence from *Dev Set* and *Test Set* is the zero prob sentences. Because there may even more word pairs are not in the Training Set. So, the perplexity of Trigram LM on *Dev Set* and *Test Set* is worse than the Bigram LM

Smoothing

Add-K Smoothing

By add the add-k smoothing in the *Trigram LM*, the perplexity is different as before.

Table 4 Perplexity of Add-K Smoothing Trigram for Training Set

		K						
		0.00001	0.0001	0.001	0.01	0.1	1	10
Threshold of unknown words	0	10.33	32.83	162.13	1024.52	7063.09	36930.22	83963.74
	1	10.41	30.43	140.97	860.77	5880.82	31076.33	72991.78
	3	10.59	27.67	116.61	667.81	4455.08	23883.01	59056.85
	5	10.71	25.83	101.29	550.23	3582.37	19385.34	49912.69

For *Training Set*:

- The perplexity is larger than non-smoothing version, since after adding the smoothing each probability of the word pairs are more uniformly.
- While the K is larger, the perplexity is increased because the larger the K, the probability are more uniformly.

Table 5 Perplexity of Add-K Smoothing Trigram for Dev Set

		K						
		0.00001	0.0001	0.001	0.01	0.1	1	10
Threshold	0	11871.32	10102.08	12355.41	19168.76	33802.68	60518.68	91313.29
of	1	7307.63	6083.90	7513.55	12313.37	23784.26	46902.61	76599.79
unknown	3	4814.84	3942.34	4860.34	8139.42	16643.60	35383.49	61871.45
words	5	3481.00	2823.34	3478.74	5915.30	12579.15	28313.14	52272.40

For *Test Set*:

- The perplexity is larger than non-smoothing version.
- While the K is larger, the perplexity is increased.
- No more sentence become zero prob sentence because of the smoothing.

Linear Interpolation

Here is the result from 5 pairs by using in the smoothing validation: (0.5, 0.25, 0.25), (0.333, 0.333, 0.334), (0.667, 0.1, 0.233), (0.75, 0.10, 0.15), (0.25, 0.5, 0.25).

Table 6 Perplexity of Linear Interpolation Smoothing Trigram for Training Set

		Lambda				
		(0.5, 0.25, 0.25)	(0.333, 0.333, 0.334)	(0.667, 0.1, 0.233)	(0.75, 0.10, 0.15)	(0.25, 0.5, 0.25)
Threshold	0	7.34	9.28	6.32	5.75	10.05
of	1	7.95	10.03	6.85	6.24	10.87
unknown	3	8.69	10.88	7.53	6.88	11.74
words	5	9.18	11.41	8.00	7.32	12.25

For *Training Set*:

- The perplexity is larger than non-smoothing version, but smaller than Add-K smoothing version

Table 7 Perplexity of Linear Interpolation Smoothing Trigram for Dev Set

		Lambda				
		(0.5, 0.25, 0.25)	(0.333, 0.333, 0.334)	(0.667, 0.1, 0.233)	(0.75, 0.10, 0.15)	(0.25, 0.5, 0.25)
Threshold	0	478.17	431.98	609.91	682.27	409.29
of	1	123.10	113.32	149.89	163.99	108.30
unknown	3	97.13	90.51	117.07	126.47	86.06
words	5	82.69	77.76	98.72	105.71	73.77

For *Dev Set*:

- The perplexity is also larger than non-smoothing version, but smaller than Add-K smoothing version

Choosing Hyper-parameters

From the above validations, let's choose the K from 1, 3 or 5, and the lambda pair should be (0.25, 0.5, 0.25) since it performs best in the above validation running. Here is the result:

Table 8 Perplexity of Add-K & Linear Interpolation Smoothing Trigram for Test Set

	Threshold of unknown words		
	1	3	5
Lambda Pair (0.25, 0.5, 0.25)	72.53	59.56	51.65

Training Half Data?

If just training the model with half of the training data, the perplexity should increase, since it will have more chance to meet the unknown words or words pairs, and this will increase the perplexity. Unless the *Training Set* is very huge, so even half of it still in million level of data, in this case, it might be not affecting a lot.

And here is the result by running the same *Trigram LM* with only half of the *Training Set*, and the perplexity scores:

Table 9 Perplexity of Add-K & Linear Interpolation Smoothing Trigram for Test Set with half Training Set

	Threshold of unknown words		
	1	3	5
Lambda Pair (0.25, 0.5, 0.25)	179.07	128.42	103.55

Increase Unknown Threshold

From those validations run above, while the threshold of **UNK** is increase, normally the perplexity should be decreased. Because:

- it's will increase the chance for the word pairs had been seen from the *Training Set*, since more pairs will become the **UNK** word pairs, so less chance to make the sentence become zero probability.

But for the *Trigram LM* without any smoothing algorithm, we can notice, by add the threshold, the perplexity will increase. I think this might because:

- There have so many zero prob sentence, which make the real Test Set size quite small
- In the real Test Set, there have so many pairs become the **UNK** word pairs since the threshold is increasing
- The other word pair probability is not that high and **UNK** word pairs' probability handle the major of the computation of perplexity.

For an extreme case: the word pairs after training from *Training Set* are all (**UNK, UNK, UNK**), so $\mathrm{p}(\mathbf{UNK}, \mathbf{UNK}, \mathbf{UNK}) = 1$, so the perplexity will become positive correlation with the sentence length in the *Test Set*.