CS 4740 Project 1

1 Unsmoothed n-grams

Preprocess
For preprocessing, first we removed irrelevant text such as the “Re: ”, “From: ” and emails. Then converted all letters to lowercase. Also, we removed most punctuations that were very likely to be sentence separators such as ‘.’, ‘!’, ‘?’ etc . We implemented nltk python package for sentence separation. For each sentence, we created sentence boundary “<S>” and “</S>”, which marked the beginning and the end for random sentence generation.

N-grams
We implemented a general n-gram model, which were not limited to unigram, bigram or trigram. N was the input that dictated the N-gram model to be generated when the method was called. From the training file/directory we generated the n-gram probability table.

2 Random sentence generation

Algorithm
We developed a random sentence generator based on the n-gram model, which could generate sentences with or without giving the beginning of the sentence. For the start of sentence, if there was no preceding word, we set default preceding word “<S>”. For the end of sentence, when getting token “</S>” or the sentence meets the maximum length, the generator stops and outputs the sentence. For n > 1, we use back-off method to process unknown tokens. For example, say we already have preceding words “I have a” and want to use trigram get the next probable word (or token): if we have tokens in trigram probability table starting with “have a”, such as “have a pen” or “have a laptop”, we could choose one of them based on probability; if we do not have such kind of tokens in trigram table, we will try to find tokens starting with “a” in the bigram table; if failed again, we will resort to unigram.

For sentence structures construction, we realized functions such as capitalizing the first character of a sentence, and capitalize the letter “I” when it’s occurrence is an instance of “ I ”, “ I’m ” , or “ I’ve ”

Results and Analysis
Here are results of 1-4 gram:

[1-gram] In I please teens will proof the them for all r than it 96k most violence came car the so .
[2-gram] Does come ; those who can't make the ' open vms 4ever ' em aren't outofline with jackson racing for not as it look at which the change .
[3-gram] Personally , I had my subaru liberty owners out there who are absolutely disgusted at the airport and read the service manual and get your head out of curiosity , how expensive they are rare ?
[4-gram] Does anyone know the law ?

First, look at the sentences generated completely by the algorithm. 1-gram algorithm performed poorly at eliminating broken words from preprocessing. The sentences appeared to consist of random pairings of words without any grammatical or sentence structures. Inadequate study of word sequences in 1-gram algorithm resulted in

the lack of continuous and fluent sentence structures. 2-gram model considers 2 preceding words when computing the probabilities of a word sequences. Therefore, the sentence completely generated by the algorithm showed some improvement on producing intelligible word sequences, but they still lacked sentence structures. However, the sentences generated seem to be random combinations of word sequences. There is still room for improvement by applying a higher "gram" algorithm. 3-gram algorithm considers 3 preceding words for each word when computing the probabilities of word sequences, which enables the algorithm to produce some sentence structures. Although there were some grammatical errors in the sentences generated, sentence structures were clearly shown in the 3-gram output. Also, 3-gram algorithm produced sentences with comprehensible meanings. 4-gram algorithm considers 4 preceding words for each word when computing the probabilities of word sequences. The sentences generated were relatively well constructed and had very specific meanings. However, we believe that this is where signs of overfitting begin to show. Overfitting happens when the model is fixating on the training data(essentially memorizing them) . As a result, it is unable to generalize when applied to the testing set(in this case, generating random sentences), which entails that while the output looks more sophisticated, the inherent lacking of flexibility makes it less desirable.

3 Smoothing and unknown words

Smoothing and unknown word handling is important. Because the training set of data is always different from the testing set, some words that appear in the test set may never have appeared in the training set. Without smoothing and unknown words handling, the model does not have an effective way of processing the "new" words that pop into its radar when applied to the test set. Smoothing also reduces the huge difference in probability between words that only appear a few times and words that appear for a large number of times. This reduction of difference is also a reduction of bias in the training set. Because all training set is in no way comprehensive and representative for all the test set, smoothing is important to make the model perform better when applied to a test set.

4 Perplexity

We produced language models for each topic from n = 1 to n = 5. With each model, we computed the perplexity for each training file and test file. Average perplexity is the highest when n = 1, which is reasonable because unigram has the lowest comprehension of the data. We calculated the perplexities of the training sets with respect to themselves, and that of the test sets with respect to the training sets. We logged the average results of perplexities for training and test sets in the following table.

Topic	motorcycles		religion		space		atheism		autos		graphics		medicine	
ngram	train	test	train	test	train	test	train	test	train	test	train	test	train	test
1	436	177	417	218	393	216	393	216	404	191	450	200	445	212
2	26	31	30	35	30	34	30	34	29	33	32	36	32	35
3	13	186	12	204	12	187	12	187	15	190	19	218	18	240
4	16	663	12	769	11	705	11	705	18	682	23	788	21	887
5	21	1346	15	1596	14	1473	14	1473	26	1357	33	1583	27	1779

The difference between the results on training set and test set are very obvious. Thus the perplexity of training set is expected to be smaller than that of the test set. However, for unigram and bigram, due to the lack of sufficient study of the data, the perplexities generated increased by a large amount. Therefore we concluded that in order to use perplexity in the realization of other functions, we should consider employing a higher-than-3gram model.

5 Topic classification

First, we splitted the labeled data into training and test set by a ratio of 4:1. With training data, we obtained the ngram probability table for different topics then used these tables as the basis for computing the perplexity of the test file. For each test file, we obtained 7 perplexity results for 7 topic models. We then chose the topic with smallest perplexity as the classification result for test file. Finally we used accuracy of classification to evaluate the ngram model.

Accuray = sum of correct classification test files / sum of test files

We implemented three different modifications of ngram algorithms for topic classification: good-turing, linear Interpolation and Katz's Backoff. The estimators for both Backoff and linear interpolation are tested with enumeration and select the best one. The results are compared as follows:

Smoothing with Good-Turing

Results generated with different N models are shown in the table below:

Accuracy of Language Model with Good-Turing

N	1	2	3	4	5	6	7
Accuracy	1.7%	8.0%	43.4%	47.2%	51.1%	54.0%	56.6%

When $n \leq 3$, the accuracy grows in a large scale with n increasing because as discussed above, unigram and bigram models are extremely ineffective in giving an accurate perplexity estimate.
When $3 \leq n \leq 5$, the accuracy increases slowly. The perplexity results shown in the previous section for 3-5 gram models also only have a small amount of improvement. This is likely because for our training and testing data sets are relatively concentrated in a small number of topics. Therefore, without overfitting, 3-5 gram models all produce relatively good results.
When $n \geq 5$, the accuracy decreases because due to the curse of dimensionality, there will be too many unknown tokens in probability table when n is large.

Combining Linear Interpolation Estimators

Based on good-turing ngram, we calculated perplexity with linear interpolation. The linear interpolation was applied for trigram, 4-gram and 5-gram as these grams are proved to have better prediction in previous section. The accuracy increases as N increases. However, all of the accuracy are slightly lower than what we got in previous section. No significant improvement from linear interpolation is found. However, on Kaggle submission, accuracy of models with linear interpolation is slightly higher than those without linear interpolation (52% compared to 47.2%). Therefore, linear interpolation does help improve performance of language models.

The highest accuracy for 5-gram is 83.6%, which is lower than using 5-gram without linear interpolation. However, according to the result on Kaggle, accuracy of model applied with linear interpolation for test data is higher on test data than models without combining estimators. This could be because ngram models with n higher than 3 would result in overfitting.

Accuracy of Model with Linear Interpolation

N	3	4	5
Accuracy	41.2%	46.0%	48.9%

Combining Katz's Backoff Estimator

Backoff is another approach we implemented to improve performance of our language models. The performance is also lower than that of using Good-Turing alone. The result was not submitted to Kaggle, so we cannot compare the result of LM with Katz's Backoff. But we predict similar result as we got with Linear Interpolation

Accuracy of Model with Katz's Backoff

N	3	4	5
Accuracy	43.6%	46.7%	48.9%

Profile Distance Algorithm

Profile Distance Algorithm is also applied to improve the performance of our language model. The accuracy for different N value is shown below. Profile distance algorithm sort the order of n-grams by frequency and calculate the distance between high frequency n-grams in language models with test data (Cavnar, etc.). The model with lowest distance will be selected as our prediction. This proves to be a good solution for classification. The accuracy is highest when N = 1 and dropped slightly when N = 3. This could be due to this method is invulnerable to unknown word. As our sample size is small, the rate of unknown words grows quickly. When N = 3, the average rate of unknown words reaches around ⅓. We submitted our prediction on testing data with unigram model, and the accuracy is 78.4% on Kaggle, which proves that this is a valid method for classification.

Accuracy of Model with Profile Distance Algorithm

N	1	2	3	4
Accuracy	88.9%	88.7%	82.9%	79.8%

6 Context-Aware Spell Checker

We built a context-aware spell checking function based on good-turing ngram model. We are given a file of words that are likely to be confused.

First, we built the good-turing ngram model based on the training data for each topic. Then, after preprocessing the text to be checked, we found the occurrences of the confusing words in the text, and computed the perplexity of the sentence where the word was located with respect to the good turing ngram model we constructed. Then we replaced this word with possible substitutions and computed the perplexity of the alternative sentences. For each alternative sentence, we determined that the one that had the lowest perplexity would be the the one that contained the correct word. The accuracies of this method described for each n-gram model are listed below.

Accuracy of Spell Check on Different Language Model

N	1	2	3	4
Accuracy	69.2%	71.2%	79.6%	75.4%

Apparently, 3-gram model has the highest accuracy. This was expected because unigram and bigram were not particularly effective in computing perplexity as discussed in the previous section. 4-gram model was being affected by overfitting, so the accuracy for 4-gram model decreased. As a result, we determined that trigram model was the best model for the task of context-aware spell checker. Therefore, we used trigram model to perform the task of correcting the test documents. We used the same method as described; after finding the corrected words, we replaced the confusing words in the original file with the correct words.

7 Open-ended extension

General Ngram Model

For training corpora, we got perplexities for each file with different n-gram model. To better comparing the results, a table is shown below.

Perplexity peaked when n = 1, is lowest when getting to trigram and 4-gram

Topic	motorcycle	religion	space	atheism	autos	graphics	medicine
1-gram	383	374	452	353	371	431	435
2-gram	51	61	50	38	54	59	57
3-gram	20	22	22	22	21	26	23
4-gram	22	21	22	21	21	28	24
5-gram	33	31	34	30	33	42	38

Linear Interpolation and Katz's Backoff.

Details described in section 6: Topic classification.

8 WorkFlow

Guanqiao Qian : Implemented context aware spell checker.
Zili Xiang : Implemented topic classification function with perplexity and Katz's Backoff. Implemented linear interpolation model.
Youdan Xu : Implemented the general n-gram model and the Good-Turing n-gram model. Implemented profile distance algorithm for topic classification,

Reference

Cavnar, William B., and John M. Trenkle. "N-gram-based text categorization."Ann Arbor MI 48113.2 (1994): 161-175.