Question 1 Smoothing

Consider the following back-off scheme. First, we define the sets

$$\begin{aligned} \mathrm{A}(w_{i-1}) &= \{w : c(w_{i-1}, w) > 0\} \\ \mathrm{B}(w_{i-1}) &= \{w : c(w_{i-1}, w) == 0\} \\ \mathrm{A}(w_{i-2}, w_{i-1}) &= \{w : c(w_{i-2}, w_{i-1}, w) > 0\} \\ \mathrm{B}(w_{i-2}, w_{i-1}) &= \{w : c(w_{i-2}, w_{i-1}, w) == 0\} \end{aligned} \tag{1}$$

where c is a function that counts n-grams in the training set. For example, if the bigram "Honey Bunny" appears 22 times in the corpus, we will have *c(Honey, Bunny)* = 22.

Now, we can define a back-off trigram model:

$$p(w_i|w_{i-2}, w_{i-1}) = \begin{cases} p_1(w_i|w_{i-2}, w_{i-1}) & w_i \in \mathrm{A}(w_{i-2}, w_{i-1}) \\ p_2(w_i|w_{i-2}, w_{i-1}) & w_i \in \mathrm{A}(w_{i-1}) \quad and \quad \mathrm{B}(w_{i-2}, w_{i-1}) \\ p_3(w_i|w_{i-2}, w_{i-1}) & w_i \in \mathrm{B}(w_{i-1}) \end{cases}$$

Where:

$$\begin{aligned} p_1(w_i|w_{i-2}, w_{i-1}) &= p_{ML}(w_i|w_{i-2}, w_{i-1}) \\ p_2(w_i|w_{i-2}, w_{i-1}) &= \frac{p_{ML}(w_i|w_{i-1})}{\sum_{w \in \mathrm{B}(w_{i-2}, w_{i-1})} p_{ML}(w_i|w_{i-1})} \\ p_3(w_i|w_{i-2}, w_{i-1}) &= \frac{p_{ML}(w_i)}{\sum_{w \in \mathrm{B}(w_{i-1})} p_{ML}(w_i|w_{i-1})} \end{aligned} \tag{2}$$

Is the above model a valid probability distribution? Justify your answer. If it is not a valid probability distribution, suggest how to make it one by modifying p_1, p_2, and p_3, using p_{ML} (the maximum likelihood estimate) and/or the count function c, and briefly explain why your modification works.

Solution

For the model to be a viable probability distribution,

$$\begin{aligned} \sum_{w \in \mathrm{A}(w_{i-2}, w_{i-1})} p_1(w_i|w_{i-2}, w_{i-1}) \quad &+ \sum_{w \in \mathrm{A}(w_{i-1}), \mathrm{B}(w_{i-2}, w_{i-1})} p_2(w_i|w_{i-2}, w_{i-1}) \\ &+ \sum_{w \in \mathrm{B}(w_{i-1})} p_3(w_i|w_{i-2}, w_{i-1}) = 1 \end{aligned} \tag{3}$$

However, under the current model, p_1, p_2, and p_3 all sum to 1 in their respective disjoint set space. To modify it into a probability distribution, we can apply Katz Backoff where a probability mass is subtracted from p_1 and p_2:

$$\begin{aligned} p_{ML}(w_i|w_{i-2}, w_{i-1}) &= \frac{c(w_i, w_{i-2}, w_{i-1}) - q(w_i, w_{i-1})}{c(w_{i-2}, w_{i-1})} \\ p_{ML}(w_i|w_{i-1}) &= \frac{c(w_i, w_{i-1}) - q(w_i)}{c(w_i)} \end{aligned} \tag{4}$$

where $q()$ is a user-defined function (e.g., a constant or some other smoothing function that takes away probability from observed trigrams and bigrams in the training set).

Finally we re-distribute the probability mass from p_1 to p_2 and from p_2 to p_3:

$$\begin{aligned}\alpha_{p1\to p_2} &= 1 - \sum_{w\in \mathrm{A}(w_{i-2},w_{i-1})} p_{ML}(w_i|w_{i-2},w_{i-1}) \\ \alpha_{p2\to p_3} &= 1 - \sum_{w\in \mathrm{A}(w_{i-1}),\mathrm{B}(w_{i-2},w_{i-1})} p_{ML}(w_i|w_{i-1})\end{aligned} \tag{5}$$

Which results in the probability distribution:

$$p(w_i|w_{i-2},w_{i-1}) = \begin{cases} p_1(w_i|w_{i-2},w_{i-1}) & w_i \in \mathrm{A}(w_{i-2},w_{i-1}) \\ \alpha_{p1\to p_2} \times p_2(w_i|w_{i-2},w_{i-1}) & w_i \in \mathrm{A}(w_{i-1}) \quad and \quad \mathrm{B}(w_{i-2},w_{i-1}) \\ \alpha_{p2\to p_3} \times p_3(w_i|w_{i-2},w_{i-1}) & w_i \in \mathrm{B}(w_{i-1}) \end{cases}$$

Question 2 Language Model Classifier

One of the possible applications of language models is text categorization: given a document x as a sequence of words, assigning a category label y. Examples of text categorization include spam email detection $y \in \{spam, not\_spam\}$, news topic categorization $y \in \{sport, politics, science, ...\}$, authorship attribution $y \in \{shakespeare, woolf, pinker, ...\}$.

Sketch out how you can make use of language models for text categorization. Use equations whenever possible.

Solution

A Naives Bayes classifier can be analogous to a language model when its features are the individual words in the vocabulary. In this model, the features can be N-gram, although most likely unigram or at most bigram due to how sparse and uniformly small the counts of high-order N-gram counts become (high entropy - burying any distinguishing signal words that make a good classifier).

The classification problem can be modeled for a set of classifications $c \in \{c_1, c_2, ..., c_n\}$ for a document d and vocabulary v as:

$$\begin{aligned}\hat{c} &= \arg\max_{c_i \in c} p(c_i|d) \\ &= \arg\max_{c_i \in c} \frac{p(d|c_i)p(c_i)}{p(d)} \\ &\propto \arg\max_{c_i \in c} p(d|c_i)p(c_i)\end{aligned} \tag{6}$$

where with the assumption of "bag of words" (positional agnostic) and independence between words (unigram) and Markovian dependence between words (bigram)

$$\begin{aligned}p(c) &= \frac{count(c_i)}{\sum_{c_i \in c} count(c_i)} \\ p(d|c_i) &= \begin{cases} \Pi_{w_i \in d} p(w_i|c) = \Pi_{w_i \in d} p(\frac{count(w_i,c_i)}{\sum_{v_i \in V} count(v_i,c_i)} & unigram \\ \Pi_{w_i,w_{i-1} \in d} p(w_i|c,w_{i-1}) = \Pi_{w_i,w_{i-1} \in d} p(\frac{count(w_i,w_{i-1},c_i)}{\sum_{v_i,v_{i-1} \in V} count(v_i,v_{i-1},c_i)} & bigram \end{cases}\end{aligned} \tag{7}$$

Here, out-of-vocabulary words or words with probability mass of zero are simply discarded.

Question 3 Language Models [Programming]

For the first part of this assignment, you will implement unigram, bigram, and trigram language models. Recall that probability of seeing a sentence $s = \{x_1, ..., x_n\}$ can be modeled as follows if using the trigram models without the special START symbols:

$$p(s) = p(x_1, ..., x_n) = q(x_1)q(x_2|x_1)\prod_{i=3}^{n} q(x_i|x_{i-2}, x_{i-1}) \tag{8}$$

$q(x_i|x_{i-2}, x_{i-1})$ is the probability of seeing the current word x_i after seeing the bigram (x_{i-2}, x_{i-1}). Always assume that the last word is the special STOP symbol, i.e., x_n = STOP. Alternatively, if you like to include the special START symbols, you can instead do the following:

$$p(s|START_2, START_1) = p(x_1, ..., x_n|START_2, START_1) = \prod_{i=1}^{n} q(x_i|x_{i-2}, x_{i-1}) \tag{9}$$

Dataset

For this part, you are provided with the following files:

- *brown.train.txt*: data to train your language modle with (i.e., to estimate $q(x_i|x_{i-2}, x_{i-1})$)
- *brown.dev.txt*: development data for you to choose the best smoothing parameters (hyper-parameters)
- *brown.test.txt*: test data for evaluating your language model

The data files are formatted with each line containing a tokenized sentence (white spaces mark token boundaries). Note that using the test set for training or finding the best parameters is considered cheating.

Report the perplexity scores of the unigram, bigram, trigram language models for your *training*, *dev*, and *test* sets, and elaborate all your design choices (e.g., if you introduced START symbols in addition to STOP symbols, how you handled out-of-vocabulary words). Briefly discuss the experimental results.

Perplexity

One way to evaluate the language model is computing the perplexity of the model on previously unseen data. Recall that computing the perplexity involves computing the average log probability of the entire corpus:

$$perplexity = 2^{-l} \tag{10}$$

where

$$l = \frac{1}{M}\sum_{i=1}^{m} log_2 p(s_i) = \frac{1}{M}\sum_{i=1}^{m} log_2 p(x_{i1}, ..., x_{in_k}) \tag{11}$$

and M is the total number of words in the text (i.e. the sum of lengths of all sentences).

A word of caution: You will primarily use *brown.dev.txt* as the previously unseen data while (i) developing and testing your code, (ii) trying out different design decisions (e.g., different ways of handling out-of-vocabulary words), (iii) tuning the hyper-parameters, and (iv) performing error analysis. For scientific integrity, it is extremely important that you will use the real test data *brown.test.txt* only once just before you report all the final results. Otherwise, you will start overfitting on the test set indirectly. Please don't be tempted to run the same experiment more than once on the test data.

Perplexity	Training	Dev	Test
Unigram	969	859	860
Bigram	68	184	184
Trigram	7.8	227	228

Table 1: Perplexity by N-gram Model.

Tips

When implementing the language model, there are several things to consider:

- Don't forget to handle the out-of-vocabulary (OOV) words by converting some of the low frequency words into UNK during training.
- You will need to include the special STOP word at the end of each sentence. It is optional if you include the special START words.
- Make sure that $q(x_i|x_{i-2}, x_{i-1})$ is a valid probability distribution. More concretely, each trigram probability conditioning on a different history must sum up to 1, e.g., $\sum_i q(x_i|thecat) = 1$ and $\sum_i q(x_i|thedog) = 1$
- Be mindful of underflow! Cod up in the log space, where $log \prod_i q(x_i) = \sum_i logq(x_i)$

Solution

See code **lm.py**. The results shown in Table 1 were generated using:

./lm.py -n 1 -oov 2
./lm.py -n 2 -oov 2
./lm.py -n 3 -oov 2

I used *nltk* library which provided masking for OOV words, replacing any unigram under a threshold with *<UNK>* token. When calculating perplexity, any unknown words were first substituted (this is done natively by the library's base language model). Further I used the native padding method to pad each sentence with the start *<s>* and stop *</s>* tokens. If a bigram or trigram context does not exist, perplexity is calculated assuming using the probability of its unigram. If the word is OOV, we use the probability that it is unknown. For this we set the cut-off at 2.

Question 4 Smoothing [Programming]

To make your language model work better, you will need to implement the following smoothing techniques. For this part of the homework, you will only work with trigram language models.

1. (a) Report perplexity scores on *training* and *dev* sets for varoius values of K (no interpolation). Try to cover a couple of orders of magnitude (e.g., $K = 10$, $K = 1$, 0.1,...)
   (b) Similarly, repot perplexity scores on *training* and *dev* sets for various values of λ_1, λ_2, λ_3 (no K smoothing). Report no more than 5 different values for your λ's.
   (c) Putting it all together, report perplexity on the *test* set, using different smoothing techniques and the corresponding hyper-parameters that you chose from the *dev* set. Specify those hyper-parameters.

Trigram Perplexity	Training	Dev
k=100	9.16	39.7
k=10	8.1	36.7
k=1	7.5	42.66
k=0.1	7.47	76.3

Table 2: Trigram Perplexity for K-values

2. If you use only half of the training data, would it in general increase or decrease the perplexity on the previously unseen data? Discuss the reason why.

3. If you convert all the training words that appeared less than 5 times to UNK (the special symbol for OOV words), would it in general increase or decrease the perplexity on the previously unseen data compared to an approach that converts only a fraction of the words that appeared just once as UNK? Discuss the reason why.

Add-K Smoothing

$$p(x_i|x_{i-2}, x_{i-1}) = \frac{K + C(x_{i-2}, x_{i-1}, x_i)}{\sum_{x \in V^*}(K + C(x_{i-2}, x_{i-1}, x))} \tag{12}$$

where $C(x_{i-2}, x_{i-1}, x_i)$ is the trigram count, and V^* is the set of your entire vocabulary (including special tokens such as OOV words and sentence boundaries).

Linear Interpolation

$$p'(x_i|x_{i-2}, x_{i-1}) = \lambda_1 \times p(x_i|x_{i-2}, x_{i-1}) + \lambda_2 \times p(x_i|x_{i-1}) + \lambda_3 \times p(x_i) \tag{13}$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

Choosing Hyper-parameters

You can use the development data *brown.dev.txt* to decide the best values of the hypder-parameters(K, λ_1, λ_2, and λ_3). The fancier way of picking the hyper-parameters for linear interpolation is through Expectation-Maximization (EM), but you haven't learned it yet. For this homework, you will isntead do a *"grid search"*, which means you will simply try a few combinations of "reasonable" values.

Solution

See code **lm.py**.

The results for (1a) shown in Table 2 were obtained by running:

./lm.py -n 3 -oov 2 –k 100 | 10 | 1 | 0.1

The results for (1b) show in Table 3 were obtained by running:

./lm.py -n 3 -oov 2 –interpolate 0.1 0.45 0.45
./lm.py -n 3 -oov 2 –interpolate 0.2 0.4 0.4
./lm.py -n 3 -oov 2 –interpolate 0.3 0.35 0.35
./lm.py -n 3 -oov 2 –interpolate 0.4 0.3 0.3
./lm.py -n 3 -oov 2 –interpolate 0.5 0.25 0.25

Trigram Perplexity	**Training**	**Dev**
interpolate=0.1 0.45 0.45	12.2	257
interpolate=0.2 0.40 0.40	13.5	266
interpolate=0.3 0.35 0.35	15	277
interpolate=0.4 0.30 0.30 .1	17.3	291
interpolate=0.5 0.25 0.25 .1	20.1	307

Table 3: Trigram Perplexity with Linear Interpolation

The results for (1c) were obtained by running *./lm.py -n 3 -oov 2 –k 10 –interpolate 0.1 0.45 0.45* which yielded a perplexity of 52.2 for the *test* set.

The answer for (2) I believe is that perplexity would decrease. This is because a large portion of the probability space is actually covered by the unknown token. By decreasing the training set, I would have increased the probability mass of unknown token. This would create an artificially low uncertain even though UNK tokens have infinite uncertain since the decrease in perplexity from UNK tokens will far outweigh the increase in perplexity from my known words.

Then answer for (3) is the same as (2). Increasing the cut-off increases the amount of words assigned to UNK, which artificially deflates the perplexity since the decrease in perplexity from UNK tokens will far outweigh the increase in perplexity from my known words. For the same settings as (1c) except for *-oov 5*, the perplexity was 47.5.