Applied probability models for CS (2017-2018)
Exercise 2

General

In this exercise you will explore different methods to estimate the probability of seen and yet unseen events. You will write a computer program that uses a development set to learn different language models and compares them according to their perplexity on a test set. In this exercise you will implement a unigram language model. In the next exercise (exercise 3) you will implement a bigram model with backoff.

Read and follow the instructions below carefully.
Debug your code before submitting.

The input

Your program will accept the following 4 arguments in this exact order:
$< development\_set\_filename > < test\_set\_filename > < INPUT\_WORD > < output\_filename >$
The meaning of these arguments is explained in this section.

Development set (develop.txt) and test set (test.txt) files are included in the data set provided with this exercise. They are derived from a Reuters corpus, known as Reuters-21578 (since it contains 21,578 articles). They contain articles and their topics from a list of 9 topics (see topics.txt for that list). For now you should ignore the topic information (you will use it in exercise 4 in which you will implement a classification algorithm).

You will develop your models using the development set only and report results based on the test set. This exercise is not about text processing, we just use words as representing events, so 'Flower' is a different event from 'flower' and 'flowers' (we also count proper names and punctuation as events). Therefore you should be case sensitive. Consider the text as a sequence of events (words) that are separated by white spaces (usually a single space or a new line). Basically everything between 2 white spaces is an event.

Some preprocessing was applied to the input in order to make its processing easier:

- Words are tokenized, so every sequence of signs between 2 white spaces is a word (event).
- All words were converted to lowercase.
- In addition, the following punctuation tokens were removed (if found alone between two white spaces) ! # , - . : ; ? @ _ ~

The files develop|test_no_preprocess.txt present the same articles as develop|test.txt respectively but no preprocessing was applied there. You can use these files to read the articles in ease but do NOT use them for training or testing your model.

There are 2 lines for each article in the input files. The first one is the article header and the second is the article itself. When developing and testing your model you should only consider the article itself (and NOT its header line).

You should use the supplied develop.txt and test.txt files to generate the required report

as described later, but your program should be able to accept any other input files with the same format.

INPUT_WORD is a simple string representing either a seen or unseen word (such as 'the', 'honduras', '???', etc.).

The output file name is the file where you should write the output of your program as described below.

Let V denote the language vocabulary, i.e. the set of all possible different events in the language. Assume that the vocabulary size $|V|$ is $300,000$.

Implementation requirements - Unigram language model

Your program is to implement two smoothing methods and evaluate their results in a single run as described in the steps below. The output data required from your program is indicated below in 'OutputX' bullets. In the report section following this one you will be instructed how to produce the output file with this data. Do not round your results unless specifically instructed to.

1. Init
   (a) Output1: development set file name
   (b) Output2: test set file name
   (c) Output3: INPUT_WORD
   (d) Output4: output file name
   (e) Output5: language vocabulary size
   (f) Output6: $P_{uniform}(Event = INPUT\_WORD)$; i.e. the probability of this event, assuming all events in the language have an equal probability to occur (uniform distribution).
2. Development set preprocessing

   Let S denote the sequence of events given in the input development set. After reading the development input file compute the following:

   (a) Output7: total number of events in the development set $|S|$
3. Lidstone model training

   In order to choose the best value for λ, a parameter estimation process is needed. Your program should split the development set into a training set with exactly the first 90% of the words in S (should contain the first $(round(0.9 * |S|))$ words and a validation set with the rest 10% of the words.

   (a) Output8: number of events in the validation set
   (b) Output9: number of events in the training set
   (c) Output10: the number of different events in the training set (i.e. observed vocabulary).
   (d) Output11: the number of times the event INPUT_WORD appears in the training set

   Then your program should use the training set to train a Lidstone unigram model with a certain λ parameter choice and validate it with the validation set. As a measure for the model quality your program should calculate the model's perplexity on the validation set (see explanation about perplexity below). Your program should check the range of different values for λ between 0 and 2, and choose the one that minimizes the perplexity on the validation set. λ values should be specified up to two digits after the decimal point (no need to check λ values with three or more digits after the decimal point).

(e) Output12: P(Event = INPUT_WORD) the Maximum Likelihood Estimate (MLE) based on the training set, i.e. no smoothing.

(f) Output13: P(Event = 'unseen-word') the Maximum Likelihood Estimate (MLE) based on the training set, i.e. no smoothing (note that this is the probability that MLE assigns to unseen events if the word 'unseen-word' is not in the training set).

(g) Output14: P(Event = INPUT_WORD) as estimated by your model using $\lambda = 0.10$

(h) Output15: P(Event = 'unseen-word') as estimated by your model using $\lambda = 0.10$

(i) Output16: The perplexity on the validation set using $\lambda = 0.01$

(j) Output17: The perplexity on the validation set using $\lambda = 0.10$

(k) Output18: The perplexity on the validation set using $\lambda = 1.00$

(l) Output 19: The value of λ that you found to minimize the perplexity on the validation set

(m) Output20: The minimized perplexity on the validation set using the best value you found for λ

4. Held out model training

   Split the development set into exactly 2 halves (again, according to the number of words) where the first half is the training set $|S^T|$ and the second half is the held-out set $|S^H|$.

   (a) Output21: number of events in the training set

   (b) Output22: number of events in the held-out set

   Compute your held out model as seen in class.

   (c) Output23: P(Event = INPUT_WORD) as estimated by your model.

   (d) Output24: P(Event = 'unseen-word') as estimated by your model.

5. For both Lidstone and held-out models, debug your code by making sure that

$$p(x^*)n_0 + \sum_{x:count(x)>0} p(x) = 1$$

   where $p(x^*)$ is the probability of any unseen event and n_0 is the number of such events.

6. Models evaluation on test set

   After reading the test set input file, output the following:

   (a) Output25: total number of events in the test set

   Use your best Lidstone model (with the λ you chose) and the held-out model to output the following:

   (b) Output26: The perplexity of the test set according to the Lidstone model with the λ that you chose during development

   (c) Output27: The perplexity of the test set according to your held-out model

   (d) Output28: If your Lidstone model is a better language model for the test set than the held-out model then output the string 'L', otherwise output 'H'

   (e) Output29: Create the following table filled with your results. You should NOT output the first row with the column titles. Numbers in each row should be tab delimited (i.e. separated by a tab character). Round your results in this table to 5 digits after the decimal point.

   r values denote event frequencies in the training corpus. f_λ (referring to the best λ

you found) is r^* from class, i.e. the expected frequency according to the estimated p(x) on the same corpus from which the original r was counted. f_H is the same for the held out estimation. You calculate these fs by multiplying the discounted probability by the size of the training corpus. N_r^T is the number of events of frequency r in the training half of the development set, which is used in the held out estimation, and $t_r = \sum_{x:C^T(x)=r} C^H(x)$ as shown in class.

$r = f_{MLE}$	f_λ	f_H	N_r^T	t_r
0				
1				
2				
..				
9				

The output

Your output file should include exactly the following lines, in the following order. Each line should be tab delimited (i.e. a tab character between every two strings).

#Students <your_names> <your_ids>
#Output1 < *value* >
.
.
#Output28 < *value* >
#Output29
<10 lines of the table described in Output29>

The perplexity measure

As a measure for the quality of a model q we will use the inverse of the mean likelihood of a test corpus w_1^n as given by q: $\frac{1}{\sqrt[n]{q(w_1^n)}}$. Remember to log scale your calculation as taught in class.

The code

Observe the following directions when writing your code:

1. Write your code either in Java or Python.
2. Your code should not depend on packages that are not part of the basic distribution of Java or Python.
3. Write clear code with informative comments, especially in the parts where you implement the smoothing methods and the perplexity calculation.
4. The first line of each code file should include *your_names your_ids*.
5. The name of the main source code file that runs your program should be Ex2.java or ex2.py.
6. Use double precision floating points for real numbers' arithmetics.
7. Your program's run time with the provided development and test sets is expected to be a few minutes at most on an average PC. The formal requirement is for it to complete its run within no more than one hour.

Submission

Submissions should be in pairs. Each pair should submit only one copy (from one user only).

Run your program with the following arguments:
develop.txt test.txt honduras output.txt

Then submit your code files(s), jar file (in case of Java) and output.txt via the 'Submit' web interface. See explanation about Submit in the course web site. The exercise name in Submit is ex2 and your group number is 01.