Assignment 2

CS4442B

Albirawi, Zaid
250626065

March 16th, 2015

1. This problem investigates language sparseness, using the word language model. Do not read end of sentences for this problem.

    a. Write a program P1 that takes as the command line arguments the names of two text files, the size n for the nGram, and the last parameter which indicates whether to print out common nGrams or not. The program should count and print to the standard output the percentage of nGrams in the second text file that do not occur in the first text file. If the last input parameter is 0, the program should not print out the common nGrams between the two files. If the last parameter is bigger than 0, your program should print out common nGrams, one nGram per line.

    For example, if we want to count 5-Grams without printing, the program should be executed with:

    ```
    P1 text1.txt text2.txt 5 0
    ```

    The output format should be:
    65.001

    If we want to count 6-Grams with the printing option, the program should be executed with:

    ```
    P1 text1.txt text2.txt 6 1
    ```

    The output format should be:
    75.001
    he thought much better of it
    I can play piano in the

b. Take two parts of a novels by the same writer, “DostoevskyPart1.txt” (as the first input file) and “DostoevskyPart2.txt”(as the second input file), use your program to compute and write down the percentages of zero count nGrams for n = 1, 2, 3, ..., 6. What is the value of n that gives no common nGrams between the two files? What is (are) the largest common nGram for these files?

n	Zero Count Percentage
1	0.0346069
2	0.345976
3	0.765187
4	0.949733
5	0.990671
6	0.99802

The n values that gives no common nGrams between the two files are n > 17. The largest nGram for these files is “repulsion that s what i m afraid of that s what may be too much for me”.

c. Repeat part (b) for different writers, “Dickens.txt” (as first) and “KafkaTrial.txt” (as second).

n	Zero Count Percentage
1	0.0711615
2	0.50375
3	0.894378
4	0.98773
5	0.998449
6	0.999699

The n values that gives no common nGrams between the two files are n > 7. The largest nGram for these files are “there is no such thing as a”, and “in the middle of the table and”.

d. Repeat part (b) for the“opposite” writers, “MarxEngelsManifest.txt” (as first) and “SmithWealthNations.txt” (as second).

n	Zero Count Percentage
1	0.2198
2	0.748103
3	0.966955
4	0.996844
5	0.999713
6	0.999979

The n values that gives no common nGrams between the two files are n > 6. The largest nGram for these files are “of nature and of reason the”, “of a man s own labour”, “from them what they have not”, “in order to keep up the”, “to keep up the rate of”, and “is the same as that of”.

e. Discuss the difference in results between parts (b,c,d).

By looking at the n = 1 values in the b, c, and d parts, we notice that b has the the smallest zero count, while c comes in second, and d has the largest zero count. This means that the vocabularies are very similar. As the n increases the zero count increases as the texts do not contain the same nGrams.

2. This problem continues investigating data sparseness. Use string language model and read EOS markers.

    a. Write a program P2 which takes as the command line arguments the names of two text files and the size n for the nGram. For example, if using 5-Grams, the program should be executed with:

    P2 text1.txt text2.txt 5

    The program should count and print to the standard output the percentage of the sentences in the second file that have zero probability under ML (maximum likelihood) nGram model constructed from the first text file. Note that you do not have to compute the probability of a sentence in this problem, just to find if the probability of a sentence is 0. This is the case if the count of any nGram in the sentence is 0. Also note that you need to compute counts of all nGrams for m = 1, ..., n. For example, if n = 3, you need to compute counts of 1-gram, 2-gram, 3-gram. You can store them all in the same hash-table, or you can have a vector of hash tables, each index for separate value of n. Do not store anything under index 0 in this case, since we have no 0-Grams.

    Function read tokens automatically puts end of sentence maker EOS at the end of the file if you set input parameter eos = true. Suppose file trainP2.txt contains:

    xx ab ab.
    bc bc bc

    When you use my function read tokens you will get a vector of tokens:

    xx ab ab <EOS> bc bc bc <EOS>

    You should treat the contents of trainP2.txt as two sentences:

    xx ab ab
    bc bc bc

    Suppose file testP2.txt contains:

    ab.
    bc.
    xx.

When you use my function read tokens with eos = true, it will read the file as:

ab <EOS> bc <EOS> xx <EOS>.

You should treat the contents of testP2.txt as three sentences:

ab <EOS>
bc <EOS>
xx <EOS>

Your program invoked as

P2 trainP2.txt testP2.txt 2

should output
33.33

The first two sentences have non-zero probability, and the last sentence xx <EOS> has zero probability because bigram xx <EOS> never occurs in the first file trainP2.txt.

When invoked as

P2 trainP2.txt testP2.txt 1

your program should output
0

This is because all three sentences under unigram model (based on counts of individual words) have non-zero probability.

b. Write down the outputs of your program for input files “DostoevskyKaramazov.txt” as the first file, “Dickens.txt” as the second file, for n = 1, 2, 3, 4, 5, 6.

n	Zero Percentage
1	0.742582
2	0.949176
3	0.967033
4	0.96978
5	0.96978
6	0.96978

c. For for n = 6, what is (are) the sentence of non-zero probability of length at least 6?

“i don t know what <END>”

3. You will develop a random text generator according to a learned language model. Use string language model with reading of EOS marker.

a. Write a program P3 which takes as command line arguments the name of a text file and the size n of the nGram model. For example, to generate a sentence using 3-gram word based model learned from from file text.txt, the program should be invoked with

```
P3 text.txt 3
```

First your program should construct an ML (maximum likelihood) language model from text.txt, and then generate a random sentence according to the following procedure.

Random Sentence Generation:

Let the vocabulary V be the set of all unique words in the input text file, that is all the unigrams. Denote the size of V , that is the number of distinct unigrams, as |V|.

The first word w_1 in the sentence has no previous context and should be generated from $P(v)$, $v \in V$. Compute $P(v)$ for all words v in your dictionary V, according to the ML unigram model. Store the generated probabilities in a vector<double> probs. To be specific, assume that the vocabulary words are indexed from 0 to $|V|-1$, that is we list them as $v^1, v^2, ..., v^{|V|-1}$. Then $probs[0] = P(v^0), probs[1] = P(v^1), ..., probs[|V|-1] = P(v^{|V|-1})$.

To generate the first word, use the function for sampling that I provided you with, int drawIndex(vector<double> &probs, int $|V|$). The output of drawIndex() is the index of the word chosen (sampled). For example, if the output is 10, this means that the word with index 10 in your vocabulary is chosen, namely v^{10}. Thus you set the first word in the sentence w_1 to v^{10}. Words with higher probabilities are more likely to be chosen, as discussed in class. If $n = 1$, then you generate the second word exactly as the first one. If $n >= 1$, then to generate a second word, you now have context. Use ML to estimate $P(v|w_1)$ for all $v \in V$ as discussed in class, where w_1 is the first word you have already generated. Store these probabilities in the vector of probs and generate the second word using my sampling function drawIndex().

Continue this procedure until you generate the EOS marker. Note that if $n > 2$, then as more context becomes available, you should use it. For example, if $n = 3$, then to generate the third (and forth, and so on) word, you should use two previously generated words for context. To be more precise, to generate the kth word in the sentence wk, for an nGram model, sample from $P(v|w_{k-n+1}, ..., w_{k-1})$, where $w_{k-n+1}, ..., w_{k-1}$ are $n-1$ previously generated words, and $v \in V$.

Optional: If you wish, you can generate the first word in a slightly better way. Notice that some words are more likely to be at the beginning of a sentence, namely those which occur at the beginning of the sentences in the training text file. These words follow the end of sentence marker $< EOS >$. Thus you can sample the first word v from $P(v| < EOS >)$.

b. Run your program on “KafkaTrial.txt” with $n = 1, 2, 3, 4, 5, 6$. Discuss your results, pay particular attention to the case of $n = 1$ and $n = 6$. You might want to look inside “KafkaTrial.txt” to interpret your results for $n = 6$.

n	Sentence
1	i was only four days we had been made my fixed to be supposed to be near carthage between half round one of the building is but at it even as a fair legible square aperture in an irish colony <END>
2	many of the engine is supplied with fuel the negro wench myra <END>
3	by means of stationary engines the comparatively level spaces between being traversed sometimes by horse and sometimes by engine power as the case demands <END>
4	i omitted to ask the question but i should think it must have stopped and wonder what has become of the faithful secretary whom i brought along with me from boston <END>
5	it could not stand on more appropriate ground and any ground more beautiful can hardly be <END>
6	she chooses for her friends and companions those children who are intelligent and can talk best with her and she evidently dislikes to be with those who are deficient in intellect unless indeed she can make them serve her purposes which she is evidently inclined to do <END>

c. Set $n = 3$ and generate a random sentence from “MarxEngelsManifest.txt”. Compare them with the results generated from “KafkaTrial.txt”.

   in countries like france where the peasants constitute far more than half of the population of the means of communication that are created by modern industry and that place the workers of different localities in contact with one another <END>

d. Just for fun: hand in the funniest sentence from your program generated with either $n = 2$ or 3 from any text file you wish. I will run a poll for the funniest sentence, and the winner will get an edible present.

   she chooses for her friends and companions those children who are intelligent and can talk best with her and she evidently dislikes to be with those who are deficient in intellect unless indeed she can make them serve her purposes which she is evidently inclined to do <END>

4. In this problem you will implement Add-Delta and Good-Turing language models. The model should be built from the first input text file. Using this model, log of probability of each sentence in the second file should be estimated and printed out to the standard output, one line per sentence.

   Recall that Add-Delta has parameter delta, and Good-Turing has parameter threshold. All parameters and text files will be specified as a command line arguments. The last command line argument specifies which language model to use. If it is 1, then Add-Delta should be used. If it is 0, Good-Turing should be used.

   file sentences.txt, build a 5-Gram Good-Turing language model with threshold = 3, use:

   P4 textModel.txt sentences.txt 5 3 0

   To model language from file textModel.txt, estimate log probability of sentences in file sentences.txt, build a 2-Gram Add-Delta language model with delta = 0.01, use:

   P4 textModel.txt sentences.txt 2 0.01 1

   Use word language model for this problem. We need to choose the vocabulary size. Unlike problem 3, vocabulary cannot be the number of unique words in the file textModel.txt, since we have to account for unseen unigrams (words). In a somewhat ad-hoc manner, let us take the size of the vocabulary to be the number of unique words in file textModel.txt multiplied by 2.
   For this problem, we will not include EOS markers as words in our language model. Therefore, you should build language model from textModel without EOS marker. However, since you need to parse the second file textFile.txt, into sentences, I recommend that you read it with EOS markers. Then use these markers to break textFile.txt into sentences, but do not include EOS into your sentences.
   For example, suppose file sentences.txt contains:

   Dogs like to bite.
   Cats like to nap.

   When you use my function read tokens with eos = true, it will read the file as:

   dogs like to bite <EOS> cats like to nap <EOS> .

   You should treat the contents of sentences.txt as two sentences without EOS markers:

   Dogs like to bite.
   Cats like to nap.

Your program should output log probability for each sentence on a separate line. For the example above, the output should be formatted as:

```
-35.08
-55.09
```

a. Write a program P4 specified as above.

   Implementation notes:

   - I suggest to use double data type for probabilities to avoid underflows.
   - Also be careful to avoid integer overflows. When multiplying numbers that could be large, use double instead of int.
   - Be careful if your count variables are of integer type. With integer division, count of 2 divided by count of 5 is 0, but the correct answer is 0.4. Cast integers to double type before dividing.
   - The output of your program is log probabilities (to avoid underflow), therefore your output should be a negative number, since log(x) <= 0 for x <= 1.
   - Make sure that your program works for the special case of unigrams (n = 1).
   - Recall that the threshold threshold for Good-Turing is used as follows. If an nGram has rate r < threshold, use Good-Turing estimate of probability. If r >= threshold use ML estimate of probabilities. Do not forget to renormalize so that probabilities add up to 1. Use the first re-normalization method we covered in class, namely the total weight of all unseen nGrams should stay as estimated by Good-Turing.
   - For Good-Turing, if the user sets threshold so high that Nr=0 for some r ≤ threshold, then your estimated GT probabilities will be 0. Before computing probabilities, loop over Nr to check that they are not zero for all r <= threshold. You have to do this separately for 1-grams, 2-grams, ..., n-grams. Terminate program with an error message if this condition is not satisfied.
   - If delta = 0, then Add-Delta model is equivalent to ML model and some sentences will have probability 0. In this case, your program should not crash but output the maximum negative number in double precision, which is pre-defined in the compiler as -DBL MAX.
   - Good-Turing and Add-Delta each worth 10%. If you do not implement either one of them, your program should output “Not implemented” to the standard output if the user tries to invoke the non-implemented model.

b. Run your program and report the output of your program for the following cases:

- P4 KafkaTrial.txt testFile.txt 1 1 1

  -80.2317
  -104.029

- P4 KafkaTrial.txt testFile.txt 2 1 1

  -191.48
  -203.684

- P4 KafkaTrial.txt testFile.txt 2 0.001 1

  -120.564
  -196.921

- P4 KafkaTrial.txt testFile.txt 3 0.001 1

  -216.961
  -288.532

c. Run your program and report the output of your program for the following cases:

- P4 KafkaTrial.txt testFile.txt 1 1 0

  -101.941
  -124.598

- P4 KafkaTrial.txt testFile.txt 2 5 0

  -130.761
  -215.76

- P4 KafkaTrial.txt testFile.txt 3 5 0

  -150.228
  -300.313

5. In this problem we will use Add-Delta language model to classify which human languages (i.e. English, French, etc.) a given sentence is written in. Use the character based language model that reads all the characters, i.e. latin only = false. Set vocabulary size to 256.
   Folder Languages contains training and test files for six languages. Each language file has the name corresponding to the language. Training files have endings 1.txt (english1.txt, danish1.txt, etc), and test files have ending 2.txt (english2.txt, danish2.txt, etc). Assume that all the text files are stored in the same directory where the program is run, so you do not have to specify their location.
   Train the language models, separately, for each language on training files, i.e. those ending in 1. Language classification is performed as follows. Given a sentence, compute its probability under French, English, etc. language models and classify the sentence with the language giving the maximum probability. For this problem, a sentence is a sequence of characters of fixed length senLen, given as an input parameter. You need to parse an input file into consecutive chunks of characters of length senLen, and classify each chunk. If the last chunk is of length less than senLen, it should be omitted from classification.
   Your program should output the total error rate (in percents), and the confusion matrix. The total percentage error rate for all languages is the overall number of misclassified sentences in all language files divided by the overall number of sentences in all language files, multiplied by 100 to express as percentage.
   The confusion matrix is a 6 by 6 array, where C(i,j) is the number of sentences of language i that were classified as language j. That is diagonal has the correct classifications, and off-diagonal wrong classifications.

   a. Write program P5 for language identification. Your program is invoked with

      P5 n delta senLength

      Where n is the size of the nGram, delta is the parameter for Add-Delta, and senLength is the sentence length.

      The output of your program should be formatted as:
      16.79
      134 3 0 0 0 1
      24 341 1 0 0 0
      38 2 85 0 0 0
      23 1 0 213 0 0
      26 9 1 3 221 0
      77 2 0 0 0 50

Where the first line is the percentage error rate, and the next size lines is the confusion matrix.

b. Run your program and report the error rate on the following cases. Notice that in all these cases delta = 0, so this is equivalent to ML estimation.a

   - P5 1 0 50: 06.67%
   - P5 2 0 50: 44.28%
   - P5 3 0 50: 05.16%

c. Run your program and report the error rate on the following cases:

   - P5 1 0.05 50: 05.97%
   - P5 2 0.05 50: 27.01%
   - P5 3 0.05 50: 37.29%

d. Run your program and report the error rate on the following cases:

   - P5 3 0.05 50 : 37.29%
   - P5 3 0.005 50 : 30.51%
   - P5 3 0.0005 50: 23.51%

e. Compare and discuss the performance between (b,c,d).

   As n increases the error rate increases, this is because not the program will need to match sequences of words not just words. Helps make the selection more accurate. Also, as the delta value reaches 0, the error rate decreases because the population is being smoothed.

f. Explore and discuss how classification performance changes with the sentence length by running your program on the following cases:

   - P5 2 0.05 10 : 58.93%
   - P5 2 0.05 50 : 37.29%
   - P5 2 0.05 100: 31.31%

   The error rate decreases as the sentence length increases because every sentences will require more n-gram matches, and if the sentence has a value of 0 then it is discarded. Therefore, only valid matchable sentences will be tested.

g. Optional, for 5% extra credit. Repeat (b-d) with a character model that reads only 26 Latin characters. Use option latin only = true to read character tokens and set vocabulary size to 26. Discuss performance as compared with the 256 character model.

   i. Run your program and report the error rate on the following cases. Notice that in all these cases delta = 0, so this is equivalent to ML estimation.a

   - P5 1 0 50: 17.24%
   - P5 2 0 50: 44.28%
   - P5 3 0 50: 23.58%

   ii. Run your program and report the error rate on the following cases:

   - P5 1 0.05 50: 17.24%
   - P5 2 0.05 50: 09.38%
   - P5 3 0.05 50: 27.56%

   iii. Run your program and report the error rate on the following cases:

   - P5 3 0.05 50 : 27.56%
   - P5 3 0.005 50 : 08.88%
   - P5 3 0.0005 50: 06.85%

   iv. Compare and discuss the performance between (i,ii,iii).

   Same as above, the difference is that the change in n is less significant because the vocabulary is much smaller. Also, delta is more influential.

   v. Explore and discuss how classification performance changes with the sentence length by running your program on the following cases:

   - P5 2 0.05 10 : 61.85%
   - P5 2 0.05 50 : 09.38%
   - P5 2 0.05 100: 03.22%

   Same reason as the one above.

The results are more jumpy because the vocabulary is smaller.

6. In this problem you will develop a simple spell-checker.

    a. Write a spelling program P6 that is invoked with:

        P6 textTrain.txt textCheck.txt dictionary.txt n t delta model

        The input parameters are as follows: name of text file for model training, name of text file to check spelling, name of text file with dictionary, n for the nGram model, threshold for GoodTuring, delta for Add-One smoothing, model to use. As before, if model = 0 use Good-Turing, and if model = 1 use Add-Delta.
        You have to implement Add-Delta smoothing, but Good-Turing is optional. You will get +5 extra credit if you implement both. If you do not implement Good-Turing, your program should terminate with an error message and terminates if user sets model to 0.
        Use word language model without reading EOS markers to read textTrain.txt and dictionary.txt. File textCheck will contain several sentences to spell checking of. Check each sentence separately. It is convenient to read textCheck.txt with EOS marker to make spell checking easier. Output the result of checking separately on new line. For example, if textCheck.txt has sentences:

        I lke to eat cereal in the morning.
        Pla nicely!

        The format of the output should look like:

        i like to eat cereal in the morning
        play nicely

        You will implement a simpler version than what was discussed in class. Assume that there is only one misspelled word per sentence. Given the next sentence S to check, first compute its probability. Next iterate over all words in the sentence. Suppose current word is w. Find all words in the dictionary that are within edit distance of one from w, using the function for edit distance that I provide in utilsToStudents.h. Let C(w) be the set of all words within edit distance of 1 from w. Replace w in the sentence with a word from C(w). Compute probability of this new sentence. If you get a larger probability, store the sentence. Repeat for all words in C(w), updating the current best sentence (i.e. the highest probability sentence). Repeat this process for all words in sentence S. Print to the output the best sentence. Note that the highest probability sentence can be the input sentence S unchanged.

b. Report and discuss the results of your program for the following cases:

- P6 trainHuge.txt textCheck.txt dictionary.txt 2 3 1 1

  it would love to her the story
  you will read in the garden
  hello from the top of the world
  i will drink milk in the morning
  i will read the story

- P6 trainHuge.txt textCheck.txt dictionary.txt 2 3 0.1 1

  it would love to her the story
  you will read in the garden
  hello from the top of the world
  i will drink milk in the morning
  i will read the story

- P6 trainHuge.txt textCheck.txt dictionary.txt 2 3 0.01 1

  it would love to her the story
  you will read in the garden
  hello from the top of the world
  i will drink milk in the morning
  i will read the story

  No change in results, the delta value did not have a big impact on the result.

c. Report and discuss the results of your program for the following cases:

- P6 trainHuge.txt textCheck.txt dictionary.txt 1 3 0.01 1

  i would love to he the story
  you will re in the garden
  hello from the to of the world
  i will drink milk in the morning
  i will read the story

- P6 trainHuge.txt textCheck.txt dictionary.txt 2 3 0.01 1

  it would love to her the story
  you will read in the garden
  hello from the top of the world
  i will drink milk in the morning
  i will read the story

- P6 trainHuge.txt textCheck.txt dictionary.txt 3 3 0.01 1

  it would love to her the story
  you will read in the garden
  hello from the top of the world
  i will drink milk in the morning
  i will read the story

  The value of n had a big factor on the result. This is because some the words are not tested in groups vs singletons, which would've affected the probability of the sentence.

d. If you implemented Good-Turing, report and discuss the results of your program for the following cases:

   - P6 trainHuge.txt textCheck.txt dictionary.txt 1 3 1 0

     i would cove to her the story
     you wilt red in the garden
     hello from the op of the world
     i will drink milk in the morning
     i wilt read they story

   - P6 trainHuge.txt textCheck.txt dictionary.txt 2 3 1 0

     i would love so her the story
     you will reb in the garden
     hello from the tp on the world
     i will drink ilk in the morning
     i will rad they story

- P6 trainHuge.txt textCheck.txt dictionary.txt 3 3 1 0

  it would love to her the story
  you will read in the garden
  hello from the top of the world
  i will drink milk in the morning
  it will read they story

  The higher the n value the more accurate. Also relative to the answer in part c.

7. Implement program P7 that given two text files, builds a word language model from the first file and tests how good this model is on the second file for Add-One, Add-Delta, Good-Turing models. Your program should compare the rates predicted by the first model on the actual empirical rates found in the second file. You should make sure two files are of the same size in the number of tokens. You can simply clip the longer file to have the same number of tokens as the shorter file.

   Not implemented.

8. Implement program P8 that improves your spelling correction program in problem 6 in any way. You can build a better language model, implement the noisy channel model discussed in class, implement a better edit distance between strings. Hand in your improved program, and report/discuss the cases where your new program does something better compared to the old program.

   The new program will implement some of the errors that still existed in question 6. For example, it will change “it will read they story” to “i will read the story”. The program uses add_one, add_delta, and good_turing to compute the best possible probability for a sentence and chooses the best out of these probabilities.