Mikhail Skobov
CSEP 517
HW 1

Problem 1:
No, p is not a valid distribution because the sum of all probabilities will not equal 1. It appears to be an attempt at Katz Backoff, but it's missing the *crucial* subtraction of a Beta value from the probability mass. A simple "proof" by counterexample:
Assume training set: {"the cat has", "my cat has"}

Our trigram ML probability becomes:
p(has | the, cat) = c(the, cat, has) / c(the, cat) = 1 /1 = 1.0

The bigram probability we care about is:
p(has | cat) = c(cat, has) / c(cat) = 1 / 2 = 0.5
(It's multiplied by some factor, but is suffices to say that it does not go to 0)

From this, we see that the sum of all probabilities p('has') is > 1 (1 + something greater than 0), thus it is not a valid probability distribution.

How to fix:
Per the slides, the probability "mass" needs to come from somewhere to supply to the backoff probabilities. An example of this is absolute discounting: where we need to use some c* and B, such that:
Bigram case:
c*(u, v) = c(u, v) - B
and
q*(u | v) = c*(u, v) / c(v)
giving us "excess" probability mass
a(u) = 1 - sum(q*(u | v))

Trigram case:
c**(u, v, w) = c(u, v, w) - B'
and
q**(u | v, w) = c**(u, v, w) / c(v, w)
giving us "excess" probability mass
b(u) = 1 - sum(q**(u | v, w))

Lastly, you would take the probability equations p1, p2, and p3, and write them such that:
p1' => p1 where p_ML = c**(u, v, w) / c(v, w)
p2' => b(u) * p2(u) and where p_ML = c*(u, v) / c(v)
p3' => a(u) * p3(u)

Problem 2:
Language models are essentially classifiers for the likelihood that a sequence of strings is a sentence. Since there is only one class -- sentence -- we can predict the probability p(s) of whether or not s is a sentence.

When working with multiple target classes such as { sports, finance, science } we need to create separate probability distributions that a sentence 's' belongs to a given class, thus we have multiple "language models", each trained on the a corpus for a given target class. This allows each probability distribution to remain independent, so the weight of one class does not overpower another.

Continuing with the example of spam:
y in {spam, not_spam}

Train the LMs on separate corpuses {spam_text, non_spam_text} producing the following probabilities:
p_spam(s) = product(q(x_i | x_i-1, x_i-2))
p_notspam(s) = product(q(x_i | x_i-1, x_i-2))

Although the above can be any LM probability like using linear interpolation, backoff, etc.

Now to classify, we can say that:
        {spam   if p_spam(s) > p_notspam(s)
y(s) = {not_spam   if p_notspam(s) > p_spam(s)

Alternatively, we can use a Bayesian probability, such that:
P(spam|s) = P(spam) * P(s | spam) / P (s)
and
P(not_spam | s) = P(not_spam) * P(s | not_spam) / P(s)

For this to work, we need the probability P(s) which we can get by training a 3rd LM on the joint set of spam and not_spam (essentially a language model on our whole corpus).

Then we also need P(spam) and P(not_spam) which will just be:
P(spam) = c(spam)/total
and
P(not_spam) = c(not_spam)/total
where c() is the count of the documents (or sentences)

Now to classify, we can say that (y is our target class):
        {spam   if P(spam|s) > P(not_spam|s)
y(s) = {not_spam   if P(spam|s) < P(not_spam|s)

This can be generalized to apply to multiple categories, spam was just used as a simple binary example.

Problem 3:

Design choices:

- Text parsing uses lowercase and applies simple rules to numbers, proper nouns, days, etc. for replacing them as a special token (ex: 1234 becomes __NUM__)
- Using a low UNK cutoff value of 2 (if less than 2 words, it becomes an UNK)
- Unigram is used to parse UNKs/get the number of unique words, and the bigram and trigram models consume the unigram for getting word counts/UNKs

Since running a test on the training data ensures that all conditional variables exist, all probabilities and perplexities are computable. The experiment was run on dev data, and it was shown that the vast majority of bigrams and trigrams resulted in an undefined or 0 probability, which in turn made the entire sentence probability the same. Without probability, there is no perplexity, so the data for dev and test data is not shown.

Perplexities on training data:
unigram perplexity: 509.78
bigram perplexity: 73.90
trigram perplexity: 9.80

Problem 4:

3.4.2: add-k smoothing does not work well for language modeling

4.4.1.a
See Table 1 and Table 2 below.

4.4.1.b
See Table 3 and Table 4 below.

4.4.1.c
Hyperparameter tuning - this was achieved through graph/binary search.
K = 0.0026
Lambdas:
l1: 0.1, l2: 0.55, l3: 0.35

Test Data Perplexities:
K-smoothing:
unigram perplexity: 483.18
bigram perplexity: 334.72
trigram perplexity: 1476.21

Linear Interpolation: 214.26

4.4.2
By using only half of the training data, it would increase the sparsity (words seen are become more "rare"). It also decreases the chance of a future sentence having been seen, therefore perplexity becomes higher, since the LM will not give a future sentence as high of a probability.

4.4.3
If the threshold for UNKs was set to 5 instances, this means that words which are branched to all get grouped into a single branch -- the UNK branch. Therefore, the overall branching factor will be reduced, and therefore the perplexity will be reduced as well. Additionally, for the unigram model, the likelihood of an UNK would be higher, so the chance of unknown words will also be higher, once again, lowering the perplexity.

Dev Perplexity k-smoothing:
K: 0.0026
unigram perplexity: 472.8790601221958
bigram perplexity: 331.72911582962087
trigram perplexity: 1039.8282539477616

Dev Perplexity Linear Interpolation:
perplexity: 210.1590120916592

Problem 4 Data:

Table 1: K-smoothing Training:
K: 10
unigram perplexity: 509.77905581738
bigram perplexity: 2708.065550135412
trigram perplexity: 7744.885847474142
K: 1
unigram perplexity: 509.77905581738
bigram perplexity: 851.3809170082309
trigram perplexity: 3013.023402707679
K: 0.1
unigram perplexity: 509.77905581738
bigram perplexity: 281.842566046679
trigram perplexity: 593.5063657480451
K: 0.01
unigram perplexity: 509.77905581738
bigram perplexity: 128.00160287127957
trigram perplexity: 109.64987775173378
K: 0.001

unigram perplexity: 509.77905581738
bigram perplexity: 85.52460390846835
trigram perplexity: 28.860033495369812
K: 0.0001
unigram perplexity: 509.77905581738
bigram perplexity: 75.58539894075348
trigram perplexity: 13.11315983860052
K: 1e-05
unigram perplexity: 509.77905581738
bigram perplexity: 74.07958746178436
trigram perplexity: 10.210856681403826

Table 2: K-Smoothing Dev:
K: 10
unigram perplexity: 472.8790601221958
bigram perplexity: 2674.234114406643
trigram perplexity: 5796.54074732375
K: 1
unigram perplexity: 472.8790601221958
bigram perplexity: 979.8438450500341
trigram perplexity: 3270.4927143355926
K: 0.1
unigram perplexity: 472.8790601221958
bigram perplexity: 469.1463319910539
trigram perplexity: 1728.286053058837
K: 0.01
unigram perplexity: 472.8790601221958
bigram perplexity: 335.15470791194815
trigram perplexity: 1127.6694301484324
K: 0.001
unigram perplexity: 472.8790601221958
bigram perplexity: 354.5212303936207
trigram perplexity: 1073.942646289045
K: 0.0001
unigram perplexity: 472.8790601221958
bigram perplexity: 492.07651369194554
trigram perplexity: 1594.256401500404
K: 1e-05
unigram perplexity: 472.8790601221958
bigram perplexity: 752.9974370802273
trigram perplexity: 3302.8410821620637

Table 3: Linear Interpolation Training:
l1: 0.33, l2: 0.33, l3: 0.33

perplexity: 20.11231870427095
l1: 0.7, l2: 0.15, l3: 0.15
perplexity: 12.53739682787949
l1: 0.15, l2: 0.7, l3: 0.15
perplexity: 27.377321881032277
l1: 0.15, l2: 0.15, l3: 0.7
perplexity: 36.651910777807146
l1: 0.6, l2: 0.3, l3: 0.1
perplexity: 13.519997329488588

Table 3: Linear Interpolation Dev:
l1: 0.3333333333333333, l2: 0.3333333333333333, l3: 0.3333333333333333
perplexity: 221.28983645642882
l1: 0.7, l2: 0.15, l3: 0.15
perplexity: 307.3583933698061
l1: 0.15, l2: 0.7, l3: 0.15
perplexity: 222.8897380626047
l1: 0.15, l2: 0.15, l3: 0.7
perplexity: 248.33711981674057
l1: 0.6, l2: 0.3, l3: 0.1
perplexity: 281.66747678611614