Natural Language Processing - Assigment 3

N-gram Language Models

In the textbook, language modeling was defined as the task of predicting the next word in a sequence given the previous words. In this assignment, we will focus on the related problem of predicting the next *character* or *word* in a sequence given the previous characters.

The learning goals of this assignment are to:

- Understand how to compute language model probabilities using maximum likelihood estimation.
- Implement basic smoothing, back-off and interpolation.
- Have fun using a language model to probabilistically generate texts.
- Compare word-level langauage models and character-level language models

Character-level N-gram Language Models [10 pts]

You should complete functions in the script `hw3_skeleton_char.py` in this part. After submitting `hw3_skeleton_char.py` to Gradescope and passing all the test cases for this part, you can get full score.

Part 1: Generation

In [ ]:
```
from ngram_skeleton.hw3_skeleton_char import ngrams,NgramModel,create_ngram_model,NgramModelWithInterpolation
import random
```

Write a function `ngrams(n, text)` that produces a list of all n-grams of the specified size from the input text. Each n-gram should consist of a 2-element tuple `(context, char)`, where the context is itself an n-length string comprised of the n characters preceding the current character. The sentence should be padded with n ~ characters at the beginning (we've provided you with `start_pad(n)` for this purpose). If $n = 0$, all contexts should be empty strings. You may assume that $n \geq 0$.

In [ ]:
```
print(ngrams(1, 'abc'))
```

```
[('~', 'a'), ('a', 'b'), ('b', 'c')]
```

We've also given you the function `create_ngram_model(model_class, path, n, k)` that will create and return an n-gram model trained on the entire file path provided. You should use it.

You will build a simple n-gram language model that can be used to generate random text resembling a source document. Your use of external code should be limited to built-in Python modules, which excludes, for example, NumPy and NLTK.

1. In the `NgramModel` class, write an initialization method `__init__(self, n, k)` which stores the order n of the model and initializes any necessary internal variables. Then write a method `get_vocab(self)` that returns the vocab (this is the set of all characters used by this model).
2. Write a method `update(self, text)` which computes the n-grams for the input sentence and updates the internal counts. Also write a method `prob(self, context, char)` which accepts an n-length string representing a context and a character, and returns the probability of that character occuring, given the preceding context. If you encounter a novel `context`, the probability of any given `char` should be $1/V$ where V is the size of the vocab.
3. Write a method `random_char(self, context)` which returns a random character according to the probability distribution determined by the given context. Specifically, let $V = \langle v_1, v_2, \cdots, v_n \rangle$ be the vocab, sorted according to Python's natural lexicographic ordering, and let $0 \leq r < 1$ be a random number between 0 and 1. Your method should return the character v_i such that

$$\sum_{j=1}^{i-1} P(v_j \mid \text{context}) \leq r < \sum_{j=1}^{i} P(v_j \mid \text{context}).$$

   You should use a single call to the `random.random()` function to generate r.

```
In [ ]: m = NgramModel(1, 0)
```

```
In [ ]: m.update('abab')
        a = m.get_vocab()
        print(a)

        {'a', 'b'}
```

```
In [ ]: m.update('abcd')
        m.get_vocab()

Out[ ]: {'a', 'b', 'c', 'd'}
```

```
In [ ]: m.prob('a', 'b')

Out[ ]: 1.0
```

```
In [ ]: m.prob('~', 'c')

Out[ ]: 0
```

```
In [ ]: m.prob('b', 'c')

Out[ ]: 0.5
```

```
In [ ]: m.update('abab')
        m.update('abcd')
```

```
Out[ ]: [('~', 'a'),
         ('a', 'b'),
         ('b', 'a'),
         ('a', 'b'),
         ('~', 'a'),
         ('a', 'b'),
         ('b', 'c'),
         ('c', 'd'),
         ('~', 'a'),
         ('a', 'b'),
         ('b', 'a'),
         ('a', 'b'),
         ('~', 'a'),
         ('a', 'b'),
         ('b', 'c'),
         ('c', 'd')]
```

```
In [ ]: random.seed(1)
        [m.random_char('') for i in range(25)]
```

```
Out[ ]: ['a',
         'd',
         'd',
         'b',
         'b',
         'b',
         'c',
         'd',
         'a',
         'a',
         'd',
         'b',
         'd',
         'a',
         'b',
         'c',
         'a',
         'd',
         'd',
         'a',
         'a',
         'c',
         'd',
         'b',
         'a']
```

1. In the `NgramModel` class, write a method `random_text(self, length)` which returns a string of characters chosen at random using the `random_char(self, context)` method. Your starting context should always be n ~ characters, and the context should be updated as characters are generated. If $n = 0$, your context should always be the empty string. You should continue generating characters until you've produced the specified number of random characters, then return the full string.

```
In [ ]: m = NgramModel(1, 0)
        m.update('abab')
        m.update('abcd')
        random.seed(1)
        m.random_text(25)

Out[ ]: 'abcdbabcdababababcdddabcdba'
```

Writing Shakespeare

Now you can train a language model. First, let's look at the corpus of data in `shakespeare_data`.

Try generating some Shakespeare with different order n-gram models. You should try running the following commands:

```
In [ ]: m = create_ngram_model(NgramModel, './shakespeare_data/shakespear
        e_input.txt', 2)
        m.random_text(250)

Out[ ]: "Fir, ace,\nAs ofte, spritheiting execore's theaccow:\nBiseer my
        leet\nits, likere me heme,\nBIROSTOLUS A sh'd Save sine in ase ot
        I sell dur'd he paptres, sil terwillseeck thall meou chus,\nTomer
        e? I a verer, butur mad tood, for frow'd, Bre unds a foorbut"
```

```
In [ ]: m = create_ngram_model(NgramModel, './shakespeare_data/shakespear
        e_input.txt', 3)
        m.random_text(250)

Out[ ]: "First Cupid'st. Blow; not darust ver grospect Lond Timontain wel
        ver my thosed, the in that's the ear, I be thee Duke speak nown r
        ageonation therer:\nFare speach you wondire, whold by,\nRebell\nI
        n sore nightertainst eased with of hide.\n\nPRINCE HENRY:\nWho"
```

```
In [ ]: m = create_ngram_model(NgramModel, './shakespeare_data/shakespear
        e_input.txt', 4)
        m.random_text(250)

Out[ ]: "First Apollows-monstands. The foundly and highness' friend.\n\nC
        OSTARD:\nThe would first Guard:\nJoin while he days I true:\nI ha
        ve the reputation, good:\nBut a tradian a prithers.\n\nTOUCHSTON
        E:\nSoft,\nthough they lief talend mud indeed, double your treaso
        n "
```

What do you think? Is it as good as 1000 monkeys working at 1000 typewriters (https://www.youtube.com/watch?v=no_elVGGgW8)?

After generating a bunch of short passages, do you notice anything? *They all start with F!* In fact, after we hit a certain order, the first word is always *First*? Why is that? Is the model trying to be clever? First, generate the word *First*. Explain what is going on in your writeup.

Part 2: Perplexity, Smoothing, and Interpolation

In this part of the assignment, you'll adapt your code in order to implement several of the techniques described in Section 3 of the Jurafsky and Martin textbook (https://web.stanford.edu/~jurafsky/slp3/3.pdf).

Perplexity

How do we know whether a language model is good? There are two basic approaches:

1. Task-based evaluation (also known as extrinsic evaluation), where we use the language model as part of some other task, like automatic speech recognition, or spelling correcktion, or an OCR system that tries to covert a professor's messy handwriting into text.
2. Intrinsic evaluation. Intrinsic evaluation tries to directly evalute the goodness of the language model by seeing how well the probability distributions that it estimates are able to explain some previously unseen test set.

Here's what the textbook says:

> For an intrinsic evaluation of a language model we need a test set. As with many of the statistical models in our field, the probabilities of an N-gram model come from the corpus it is trained on, the training set or training corpus. We can then measure the quality of an N-gram model by its performance on some unseen data called the test set or test corpus. We will also sometimes call test sets and other datasets that are not in our training sets held out corpora because we hold them out from the training data.
>
> So if we are given a corpus of text and want to compare two different N-gram models, we divide the data into training and test sets, train the parameters of both models on the training set, and then compare how well the two trained models fit the test set.
>
> But what does it mean to "fit the test set"? The answer is simple: whichever model assigns a higher probability to the test set is a better model.

We'll implement the most common method for intrinsic metric of language models: *perplexity*. The perplexity of a language model on a test set is the inverse probability of the test set, normalized by the number of characters. For a test set

$$W = w_1 w_2 \ldots w_N$$

:

$$\begin{aligned} Perplexity(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \\ &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_1 \ldots w_{i-1})}} \end{aligned}$$

Now implement the `perplexity(self, text)` function in `NgramModel`. A couple of things to keep in mind:

1. Numeric underflow is going to be a problem, so consider using logs.
2. Perplexity is undefined if the language model assigns any zero probabilities to the test set. In that case your code should return positive infinity - `float('inf')`.
3. On your unsmoothed models, you'll definitely get some zero probabilities for the test set. To test you code, you should try computing perplexity on the training set, and you should compute perplexity for your language models that use smoothing and interpolation.

```
In [ ]: m = NgramModel(1, 0)
        m.update('abab')
        m.update('abcd')
        m.perplexity('abcd')
```

```
Out[ ]: 1.189207115002721
```

```
In [ ]: m.perplexity('abca')
```

```
Out[ ]: inf
```

```
In [ ]: m = create_ngram_model(NgramModel, './shakespeare_data/shakespear
        e_input.txt', 2, k=0)
        with open('./shakespeare_data/shakespeare_sonnets.txt', encoding=
        'utf-8', errors='ignore') as f:
            print(m.perplexity(f.read()))
```

```
inf
```

Note: you may want to create a smoothed language model before calculating perplexity on real data.

Smoothing

Laplace Smoothing is described in section 4.4.1. Laplace smoothing adds one to each count (hence its alternate name *add-one smoothing*). Since there are V characters in the vocabulary and each one was incremented, we also need to adjust the denominator to take into account the extra V observations.

$$P_{Laplace}(w_i) = \frac{count_i + 1}{N + |V|}$$

A variant of Laplace smoothing is called *Add-k smoothing* or *Add-epsilon smoothing*. This is described in section Add-k 4.4.2. Update your `NgramModel` code from Part 1 to implement add-k smoothing.

```
In [ ]: m = NgramModel(1, 1)
        m.update('abab')
        m.update('abcd')
        m.prob('a', 'a')
```

```
Out[ ]: 0.14285714285714285
```

```
In [ ]: m.perplexity('abca')
```

```
Out[ ]: 2.691781635477648
```

```
In [ ]: m = create_ngram_model(NgramModel, './shakespeare_data/shakespear
        e_input.txt', 2, k=0.1)
        print(len(m.get_vocab()))
        with open('./shakespeare_data/shakespeare_sonnets.txt', encoding=
        'utf-8', errors='ignore') as f:
            print(m.perplexity(f.read()))
```

```
67
7.996946415762477
```

Interpolation

The idea of interpolation is to calculate the higher order n-gram probabilities also combining the probabilities for lower-order n-gram models. Like smoothing, this helps us avoid the problem of zeros if we haven't observed the longer sequence in our training data. Here's the math:

$$P_{interpolation}(w_i|w_{i-2}w_{i-1}) = \lambda_1 P(w_i|w_{i-2}w_{i-1}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i)$$

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

We've provided you with another class definition `NgramModelWithInterpolation` that extends `NgramModel` for you to implement interpolation. If you've written your code robustly, you should only need to override the `get_vocab(self)` , `update(self, text)` , and `prob(self, context, char)` methods, along with the initializer.

The value of n passed into `__init__(self, n, k)` is the highest order n-gram to be considered by the model (e.g. $n = 2$ will consider 3 different length n-grams). Add-k smoothing should take place only when calculating the individual order n-gram probabilities, not when calculating the overall interpolation probability.

By default set the lambdas to be equal weights, but you should also write a helper function that can be called to overwrite this default. Setting the lambdas in the helper function can either be done heuristically or by using a development set, but in the example code below, we've used the default.

```
In [ ]: m = NgramModelWithInterpolation(1, 0)
        m.update('abab')
        m.update('abcd')
        m.prob('a', 'a')

Out[ ]: 0.1875
```

```
In [ ]: m.perplexity('abca')

Out[ ]: 2.4513246099214725
```

```
In [ ]: m = NgramModelWithInterpolation(2, 1)
        m.update('abab')
        m.update('abcd')
        m.prob('~a', 'b')

Out[ ]: 0.46825396825396826
```

```
In [ ]: m.perplexity('abca')

Out[ ]: 2.9003863011784152
```

```
In [ ]: m = create_ngram_model(NgramModelWithInterpolation, './shakespear
        e_data/shakespeare_input.txt', 2, k=0.1)
        print(len(m.get_vocab()))
        # m = create_ngram_model(NgramModelWithInterpolation, './shakespe
        are_data/shakespeare_input.txt', 5, k=0.1)
        # # print(len(m.get_vocab()))
        # # m.set_lambdas([0.1, 0.1, 0.1, 0.2, 0.2, 0.3])
        with open('./shakespeare_data/shakespeare_sonnets.txt', encoding=
        'utf-8', errors='ignore') as f:
            print(m.perplexity(f.read()))
```

```
201
10.288650009339714
```

In your report, experiment with a few different lambdas and values of k and discuss their effects.

Word-level N-gram Language Models [Bonus] [5 pts]

You should complete functions in the script `hw2_skeleton_word.py` in this part. After submitting `hw2_skeleton_word.py` to Gradescope and passing all the test cases for this part, you can get full score. Instructions are similar to the instructions above. It is convenient to first use `text.strip().split()` to convert a string of word sequence to a list of words. In some functions, we provide `text.strip().split()`. You can use it optionally.

Besides the corpus above, we also provide you [training data for word-level langauge models] `(word_data/train_e.txt)` and [dev data for word-level langauge models] `(word_data/val_e.txt)` in which each sentence has been processed with word tokenizer and `[EOS]` token has been appended to the end of each sentences. `[EOS]` can be regarded as the sentence boundary when generating a paragraph or evaluating the perplexity of a paragraph.

Part 1: Generation

```
In [ ]: from ngram_skeleton.hw3_skeleton_word import ngrams,NgramModel,cr
        eate_ngram_model,NgramModelWithInterpolation
        import random
```

```
In [ ]: ngrams(1, 'I love Natural Language Processing')
```

```
Out[ ]: [('~', 'I'),
         ('I', 'love'),
         ('love', 'Natural'),
         ('Natural', 'Language'),
         ('Language', 'Processing')]
```

```
In [ ]: m = NgramModel(1, 0)
```

In [ ]:
```
m.update('I love natural language processing')
m.get_vocab()
```

Out[ ]:
```
{'I', 'language', 'love', 'natural', 'processing'}
```

In [ ]:
```
m.update('I love machine learning')
m.get_vocab()
```

Out[ ]:
```
{'I', 'language', 'learning', 'love', 'machine', 'natural', 'proc
essing'}
```

In [ ]:
```
m.prob('I', 'love')
```

Out[ ]:
```
1.0
```

In [ ]:
```
m.prob('~', 'You')
```

Out[ ]:
```
0
```

In [ ]:
```
m.prob('love', 'natural')
```

Out[ ]:
```
0.5
```

In [ ]:
```
m.update('You love computer vision')
m.update('I was late today')
```

Out[ ]:
```
[('~', 'I'),
 ('I', 'love'),
 ('love', 'natural'),
 ('natural', 'language'),
 ('language', 'processing'),
 ('~', 'I'),
 ('I', 'love'),
 ('love', 'machine'),
 ('machine', 'learning'),
 ('~', 'You'),
 ('You', 'love'),
 ('love', 'computer'),
 ('computer', 'vision'),
 ('~', 'I'),
 ('I', 'was'),
 ('was', 'late'),
 ('late', 'today')]
```

In [ ]:
```
random.seed(1)
[m.random_word('~') for i in range(25)]
```

Out[ ]:
```
['I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I',
 'I']
```

In [ ]:
```
m = NgramModel(1, 0)
m.update('You are welcome')
m.update('We are friends')
random.seed(1)
m.random_text(250)
```

Out[ ]:
```
'We are friends We are friends We are friends We are friends We a
re friends We are friends We are friends We are friends We are fr
iends We are friends We are friends We are friends We are friends
We are friends We are friends We are friends We are friends We ar
e friends We are friends We are friends We are friends We are fri
ends We are friends We are friends We are friends We are friends
We are friends We are friends We are friends We are friends We ar
e friends We are friends We are friends We are friends We are fri
ends We are friends We are friends We are friends We are friends
We are friends We are friends We are friends We are friends We ar
e friends We are friends We are friends We are friends We are fri
ends We are friends We are friends We are friends We are friends
We are friends We are friends We are friends We are friends We ar
e friends We are friends We are friends We are friends We are fri
ends We are friends We are friends We are friends We are friends
We are friends We are friends We are friends We are friends We ar
e friends We are friends We are friends We are friends We are fri
ends We are friends We are friends We are friends We are friends
We are friends We are friends We are friends We are friends We ar
e friends We'
```

```
In [ ]: m = create_ngram_model(NgramModel, './word_data/train_e.txt', 2)
        m.random_text(250)
```

```
Out[ ]: 'Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beac
        house Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse'
```

In [ ]:
```
m = create_ngram_model(NgramModel, './word_data/train_e.txt', 3)
m.random_text(250)
```

Out[ ]:
```
'Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beac
house Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
Beachouse Beachouse Beachouse'
```

```
In [ ]: m = create_ngram_model(NgramModel, './word_data/train_e.txt', 4)
        m.random_text(250)
```

```
Out[ ]: 'Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beac
        house Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse Beach
        ouse Beachouse Beachouse Beachouse Beachouse Beachouse Beachouse
        Beachouse Beachouse Beachouse'
```

Do you think these outputs are more reasonable than character-level language models?

After generating a bunch of short passages, do you notice anything? *They all start with In!* Why is that? Is the model trying to be clever? First, generate the word *In*. Explain what is going on in your writeup.

Part 2: Perplexity, Smoothing, and Interpolation

Perplexity

```
In [ ]: m = NgramModel(1, 0)
        m.update('I love natural language processing')
        m.update('You love machine learning')
        m.perplexity('I love machine learning')

Out[ ]: 1.062127176862691

In [ ]: m.perplexity('I love python')

Out[ ]: inf

In [ ]: m = create_ngram_model(NgramModel, './word_data/train_e.txt', 2,
        k=0)
        with open('./word_data/val_e.txt', encoding='utf-8', errors='igno
        re') as f:
            print(m.perplexity(f.read()))

        9.07048454777657
```

Smoothing

```
In [ ]: m = NgramModel(1, 1)
        m.update('I love natural language processing')
        m.update('You love machine learning')
        m.perplexity('I love machine learning')

Out[ ]: 1.3109347884032811

In [ ]: m.perplexity('I love python')

Out[ ]: 1.5168307236158116

In [ ]: m = create_ngram_model(NgramModel, './shakespeare_data/shakespear
        e_input.txt', 2, k=0.1)
        print(len(m.get_vocab()))
        with open('./shakespeare_data/shakespeare_sonnets.txt', encoding=
        'utf-8', errors='ignore') as f:
            print(m.perplexity(f.read()))

        62983
        7.182191266139794

In [ ]: m = create_ngram_model(NgramModel, './word_data/train_e.txt', 2,
        k=0.1)
        print(len(m.get_vocab()))
        with open('./word_data/val_e.txt', encoding='utf-8', errors='igno
        re') as f:
            print(m.perplexity(f.read()))

        129555
        9.07048454777657
```

Interpolation

```
In [ ]: m = NgramModelWithInterpolation(1, 0)
        m.update('I love natural language processing')
        m.update('You love machine learning')
        m.prob('love','machine')

Out[ ]: 0.3125
```

```
In [ ]: m.perplexity('I love machine learning')

Out[ ]: 1.1601490437387003
```

```
In [ ]: m = NgramModelWithInterpolation(2, 1)
        m.update('I love natural language processing')
        m.update('You love machine learning')
        m.prob('~ I','love')

Out[ ]: 0.15277777777777776
```

```
In [ ]: m.perplexity('I love machine learning')

Out[ ]: 1.3937297392804804
```

```
In [ ]: # m = create_ngram_model(NgramModelWithInterpolation, './shakespe
        are_data/shakespeare_input.txt', 2, k=0.1)
        # print(len(m.get_vocab()))
        m = create_ngram_model(NgramModelWithInterpolation, './word_data/
        train_e.txt', 5, k=0.1)
        m.set_lambdas([0.1,0.1,0.1,0.2,0.3,0.4])
        print(len(m.get_vocab()))

        with open('./shakespeare_data/shakespeare_sonnets.txt', encoding=
        'utf-8', errors='ignore') as f:
            print(m.perplexity(f.read()))
```

Running the following code could take about 10 minutes. This should be finished within 15 minutes.

```
In [ ]: m = create_ngram_model(NgramModelWithInterpolation, './word_data/
        train_e.txt', 2, k=0.1)
        print(len(m.get_vocab()))
        with open('./word_data/val_e.txt', encoding='utf-8', errors='igno
        re') as f:
            print(m.perplexity(f.read()))
```

Please compare the perplexity of `shakespeare_sonnets.txt` when using word-level language model and character-level language model. In your writeup, explain why they are different .

Acknowledgement:

This assigment is adapted from Chris Callison-Burch's course CIS 530 - Computational Linguistics (http://computational-linguistics-class.org/index.html).