Natural Language Processing: Assignment 1

Shifei Chen

1 Tokenization

The first thing I tackled from the sample text is punctuations. Simple punctuations like `%`, `$` and `&` can be easily captured by putting them inside square backets. En dash (`--`) is also rather easy to tackle but since it takes two letters' space I need to put it after a bar, which applies to double single quotation marks as well.

For hyphens, e.g. "30-days", I considered them as one word than three standalone words. Hence I used `(\w+-)+\w+` to tokenize them as they usually consist of $n(n > 1)$ standalone words and $n - 1$ hyphens. A similar approach can be applied on decimals by using the regular expression `\d+\.\d+`.

Apostrophes are a bit more tricky to capture. We can identify possesive apostrophes by `\'s` or simply `\'` for nouns ending with "s". Apostrophes showing omission can have just one letter after it or two letters around it and I used `\'d` and `n\'t` to catch them, respectively. Now the priority of these regular expressions is the thing to take care of. Words like "doesn't" will always be splitted into "doesn" and "'t" and fails our expection. When the computer scan the string from left to right, `\w+` is the first regular expression that matches and it will have a longer result, even if I put `nt́` before `\w+`. We can fix this by introducing another regular expression `\w+(?=n\'t)` to tell the computer to stop scanning right before "n't". So far our expression looks like this: (Note `\'` and `\'` have been combined, so as `\'s` and `\'d`)

```
(\d+\.\d+|n\'t|\'[sd]|(\w+-)+\w+|\'\'?|[,;:.!?\"%$&]|--|\w+(?=n\'t)|\w+)
```

Abbreviations is the most difficult part to be honest. They usually follow a pattern of `([A-Z]\w*\.)+` but this conflicts with captialized words at the end of the sentence, e.g `Auguest.` I added `(?!    [A-Z])` to tell the computer to not look at those abbreviations at the end of the sentence. After discussing with classmates, I realized putting a quantifier is a better idea since abbreviations are meant to be short, like 3 to 4 letter every semgent. `([A-Z]\w0,3\.)+` is what I got in the end.

Finally I used regular expression `(?<=\.|\"|`|\?|\!)\s+(?=[A-Z]|\"|`|$))` to catch those spaces between two sentences and took these spaces as marks of sentence segmentation. Some times the end of the sentence can also means the end of the input string, hence I always added one more space to the end of the input string.

The completed regular expression looks like this: (Note I splitted it into multiple lines for easy understanding.)

```
r'''
    (
        (?<=\.|\"|`|\?|\!)\s+(?=[A-Z]|\"|`|$)|
        \d+\.\d+|
        n\'t|
        \'[sd]|
        (\w+-)+\w+|
        ([A-Z]\w{0,3}\.)+|
        [,;:.!?\"%$&]|
        --|
        \'\'?|
        \w+(?=n\'t)|
        \w+
    )
'''
```

I achieved 0.9895/0.9957 for precision/recall on `dev-raw1.txt`. Most of the incorrect sentence segmentations happened in people's names and titles since my regular expression cannot distinguish "Mr. Smith" from "Mr. *the end of the sentence* Smith". Another problem was in sentence `vice president of human resources for Hollingsworth & Vose.`, my program mistook part of the company's name as an abbervation. Other problems like the variety of quotation marks and sentence-final abbervations can be temporarily ignored and I had made more false positive answers than false negative ones so I reckoned I could achieve about 0.97/0.98 for precison and recall on `dev-raw2.txt`, respectively, which turned out to be 0.9777/0.9903.

2 Language Modelling

MLE in simple is tring to use the most possbile situation that we have observed from the sample data to estimate the real situation. For example suppose we have two people said ”I like dogs.” and we now try to guess what would the third person like. The most reasonable answer would be ”dogs” as well since 2/3 of the 3-people-group told us so. But in reality that third guy will very likely say another animal like cats as his favorite, or even anything other than an animal. This is exactly the limitation of MLE: We could never know what will happen in the real data by our assumption even though we tried our best to make our assumption to be the most reasonable one. In language modelling, there could be two kinds of unseen events: the word we didn't see in the corpus or unknow combinations of words (n-grams).

Take an example from the novel Sherlock Holmes. Suppose we happened to have no idea whether the famous detective is a gentleman or a lady, should we call him ”Mr.”, ”Mrs.” or ”Miss”? Maybe ”Mr.” is a better title as the probability of ”Mr.” appears before ”Holmes” is 0.0026 while ”Mrs.” or ”Miss” never appears before ”Holmes”, judging from the corpus we have. But we cannnot gurantee that the name ”Holmes” is always a male's name. It is very likely that his wife, Mrs. Holmes, also appears in the work.

In this case we have met exactly the second type of unseen events: we have the word ”Mrs.” and ”Holmes” in the sample text but we have no bigrams of ”(Mrs., Holmes)”. To solve this people introduced ”smoothing”, that is to redistribute the probability mass, for example lowering the weight of all seen events and assign some weight to all unseen events. Then we have eliminated all n-grams that didn't appear in the corpus. As the example above, we may assign a probability of 0.0001 to the bigram ”(Mrs., Holmes)” since this combination of words can happen in the book. But because Mr. Holmes is the main character it should always have a lower value.

In practice, some smoothing methods like additive smoothing are just as simple as the one we have described. The formula $\frac{f(w1,w2)+k}{N+k*V^2}$ says we just add k to all n-grams counts and increase the dominator with $k * V^2$ as well. Here N stands for the total count of all n-grams and V stands for the total count of words appeared in the corpus. The smoothing result varys with different k values as these experiments in the table below show.

Table 1: Perplexities of the same bigram model with additive smoothing

Text	$k = 0.001$	$k = 0.01$	$k = 0.1$	$k = 1$
`his-last-bow.txt`	129.967	130.829	204.439	537.455
`lost-world.txt`	209.194	196.53	281.133	652.107
`other-authors.txt`	204.256	187.307	261.827	604.603

So how do we decide which k to take? My assumption is that it depends on the context. Usually k goes lower than 1 and from these experiements we can see that the lower the k value the lower the perplexity. If we are going to model words from a 3-years-old child, than we would better to have a larger k, let's say 1, as his vocabulary should be rather small comparing with an adult's.

Besides additive smoothing, some of the other more advanced smoothing methods are backoff and interpolation. Backoff means we use the simplier model to estimate a rare event, e.g. to use the possibility of unigram”(Mrs.)” to estimate the unseen bigram ”(Mrs., Holmes)”. Interpolation on the other hands combines both the infomation from higher grade models and lower grade models to all events. For example we need to adjust the possibility of bigram ”(Mr., Holmes)” to include both its original possibility and the possibility of the unigram ”(Mr.)”. Here we take two different versions of KN discount and WB discount as examples of interpolation methods to see why they are more advanced smoothing methods compared with simple additive smoothing.

Table 2: Perplexities of bigrams using different smoothing method

Text	**Additive smoothing**	**KN discount**	**KN discount**	**WB discount**
	$k = 0.001$	Original	modified by Chen & Goodman	
`his-last-bow.txt`	129.967	103.774	105.637	111.518
`lost-world.txt`	209.194	153.097	154.41	167.924
`other-authors.txt`	204.256	150.671	151.438	162.632