s2118909
s2074755

ANLP-assignment1

1. Task1

In this task, we will removes all characters from the line that are not in the following set: characters in the English alphabet, space, digits, or the '.' character, and lowercase all remaining characters and convert all digits to '0'.

To complete the task, We use regular expression:

```
def preprocess_line(line):
  line = line[:-1] # remove '\n'
  line = re.sub(r'[^a-zA-Z0-9\s.]','',line) # remove all other characters
  line = re.sub(r'[0-9]', '0', line) # replace 0-9 to 0
  line = line.lower()
  return line
```

2. Task2

In this section, we analyse a language model to figure out the kind of estimation method that was used. Here's our analysis.

This *model-br.en* model is trigram model, the prediction depends only on the two previous character. Normally, n-gram is based on Maximum Likelihood Estimation. And we think this model use add-alpha smoothing.

The reasons are:
1) There is no zero probability combination, it means the model must use some smoothing.
2) We can see that '.aa' , '.ab', ... ,'.az' have the same probability, it means this model doesn't involve bigram or unigram, so it dose not use back-off or interpretation. This same probability means do not use other information of 'aa, ab... az'.

Therefore, we think this is add-alpha smoothing, however, the value of alpha is hard to predict.

3. Task3

In this task, we build our own trigram character language model. we introuce some details about the method and implement, Also, we display an excerpt of the language model for English.

3.1 method and implement

We use Maximum Likelihood Estimation and add-alpha smoothing to implement probability estimation in this assignment, which is formula as:

$$p(c_3|c_1c_2) = (C(c_1c_2c_3) + \alpha)/(C(c_1c_2) + \alpha v)$$

where:

v =the type of characters size
α the constant value which can be optimized by development data set

Because we think add-one steals way too much, and it is also hard to improve. To find the best α, we diveded the training.en into training set and development set. When the α is near 0.08, we get the lowerest perplexity in development set. In fact, we find that the best α on development set and test set are totally different, we think it is because the character-related distribution of test document and training document are different, that is to say they may be from two different corpus.

We write a class *Language_Model* to implement this, and use a dictionary to store the probability, the key of the dictionary is the two previous character and the value is also a dictionary whose key is the current character and value stores the distribution. This class has 5 main functions *train_model, read_model, print_model, generator_from_LM,calculate_perplexity* .

3.2 assumption

This model contain only characters in the English alphabet, space, digits, or the '.' character.

We assume that different digits have the same probability and replace all digits with 0, because we think all the digits are the same, and if we do not replace them, the digit will steal 10 times from others by add-alpha.

We also need lowercase all characters. I think the corpus is not so big enough, if we add too many characters in the lexicon, many of the conditional probs are also too sparse. That is also the reason that why we use characters rather than words to build this model.

3.3 ng

The probability with the two-character history ng. (Expect and explain)

1) I think 'ng ' and 'ng.' will have the maximum probability, intuitively, there should be lots of words end -ing. The result is that 'ng ' is large than 0.8, which matches our expectation.

2) There should be some prediction have the same and small probability, like 'ngk, ngg', I think these combination is relative rare in English. So the counts may be zero, and they have very small probability because of the method of smoothing. The result shows that 'ngb,ngc,ngg...', which also meets our expectation.

Here is the excerpt of the language model for English

> nga:0.0027104508730779256
> ngb:0.00010424811050299714
> ngc:0.00010424811050299714
> ngd:0.00401355225436539
> nge:0.0874120406567631
> ngf:0.0014073494917904614
> ngg:0.00010424811050299714
> ngh:0.00010424811050299714
> ngi:0.0014073494917904614
> ngj:0.00010424811050299714
> ngk:0.00010424811050299714
> ngl:0.0027104508730779256
> ngm:0.00010424811050299714
> ngn:0.0014073494917904614
> ngo:0.006619755016940319
> ngp:0.00010424811050299714
> ngq:0.00010424811050299714
> ngr:0.011832160542090174
> ngs:0.020953870211102423
> ngt:0.013135261923377639
> ngu:0.0027104508730779256
> ngv:0.00010424811050299714
> ngw:0.00010424811050299714
> ngx:0.00010424811050299714
> ngy:0.00010424811050299714
> ngz:0.00010424811050299714
> ng :0.8145426114151681
> ng.:0.02616627573625228
> ng0:0.00010424811050299714
> ng#:0.0014073494917904614

Task 4

In this task, we are required to generate 300 characters of random output for each of two different models: (a) the model we estimated from the English training data, and (b) the model in model-br.en.

4.1 Method

In this section, we introduce what we do to generate the sequence.

In the language model, $P(\mathrm{a} \mid \#\#)$ means the probability of a sentence beginning with `a`, and $P(\# \mid ab)$ means the probability of the sentence ending with `ab`.

Firstly, we set the `##` as the first two characters of the random sequence, and next, we use our language model to generate the new character with the last 2 characters of the sequence as the history, and so on. In this process, when the new generated character is `#`, which means it is the end of a sentence, we use `##` as the history characters again to generate the next character, and

then repeat the above process.

4.2 Output

According to the method, we generate two random sequences.
1) from the model we estimated from the English training data:

> ##as and ancy wit ing quall obil exteurove whiss whis porticuldvolithes wor this to this rebad ited of tweve of tive of that as st anced mons.zhsngents witymigh the elsone show bate ourocalloceporctions.#funionsijdrente of thisfectugzmzprobleme ext to wilso dirw0#i suen throper matuapvh#they.#ant.#m

2) from the model in `model-br.en`

> ##say cou got yout i pats thands.#vn.#se.#the ifungs.#ithat plase.#kaby.#no.#gues.#do you dont a sletthis at eah.#jushumpty is wits hats parig throu.#he hats the right.#se.#her.#i smellys ou right.#call onna pas thers.#issies there the paull go.#dook a goones.#the puld the st is the bot pushom ring

4.3 Analysis

The first sequence contains fewer sentences, and the average length of these sentence is longer compared to the second one. Another finding that is that the words in the second model is shorter than first. The reasons are that the training data of the first model includes longer words and sentences than the second one, so the generated sentences and words become longer.

Task 5

In this task, we will calculate the perplexity of the test document under each of the three language models you estimated from the training documents and using the result to guess the language of the test document. Lastly, we discuss an interesting question.

5.1 Result

After caculating, we get the three different perplexitys:

> perplexity under the model from English training data:
> 9.157260842399836
> perplexity under the model from Spanish training data:
> 30.281172643914307
> perplexity under the model from German training data:
> perplexity from LMfromtraining.en:
> 30.863778278270967

5.2 Analysis

According to the result, we can find the least perplexity one is using the model from the English training data, so we think the test document belongs to English. And the perplexity of models traing by Spanish and German are large than 30, which means the perplexity is large than uniform generator.

5.3 Question

Now, we analyse another question:

Suppose we ran our program on a new test document and get the perplexity under my English LM. Would this be enough for us to determine if the document is written in English? Why or why not?

We think it is useful to determine English, but it is not enough, here are the reasons:

1）If the perplexity of the test document is near or slight biger than 6.05, we can determine this is written in English. Because the perplexity of training data under our English LM is near 6.05.

2）If the perplexity is relative large, we can not determine whether is written in English. The high perplexity may cause by written in other language. But the test document may be still written in English, due to the different styles or genres between training data and test data, the distribution of words may be quite different. For example:

> perplexity under the model from English training data:
> 9.157260842399836
> perplexity under the model of model-br.en:
> 22.094457902882443

Both models are trained with English data, but get extremely difference perplexity.

In conclusion, we cannot determine the language only given the perplexity.