Statistical Language Modeling

Computational linguistics is an emerging field that is widely used in analytics, software applications, and contexts where people communicate with machines. Computational linguistics may be defined as a subfield of artificial intelligence. Applications of computational linguistics include machine translation, speech recognition, intelligent Web searching, information retrieval, and intelligent spelling checkers. It is important to understand the preprocessing tasks or the computations that can be performed on natural language text. In the following chapter, we will discuss ways to calculate word frequencies, the Maximum Likelihood Estimation (MLE) model, interpolation on data, and so on. But first, let's go through the various topics that we will cover in this chapter. They are as follows:

•Calculating word frequencies (1-gram, 2-gram, 3-gram)

•Developing MLE for a given text

•Applying smoothing on the MLE model

•Developing a back-off mechanism for MLE

•Applying interpolation on data to get a mix and match

•Evaluating a language model through perplexity

•Applying Metropolis-Hastings in modeling languages

•Applying Gibbs sampling in language processing

Understanding word frequency

Collocations may be defined as the collection of two or more tokens that tend to exist together. For example, the United States, the United Kingdom, Union of Soviet Socialist Republics, and so on.

In [6]:
```
import nltk
from nltk.util import ngrams
from nltk.corpus import alpino
alpino.words()
```

Out[6]:
```
['De', 'verzekeringsmaatschappijen', 'verhelen', ...]
```

In [18]:

```
import nltk
from nltk.collocations import BigramCollocationFinder
from nltk.corpus import webtext
from nltk.metrics import BigramAssocMeasures
tokens=[t.lower() for t in webtext.words('grail.txt')]
words=BigramCollocationFinder.from_words(tokens)
words.nbest(BigramAssocMeasures.likelihood_ratio, 10)
```

Out[18]:

```
[("'", 's'),
 ('arthur', ':'),
 ('#', '1'),
 ("'", 't'),
 ('villager', '#'),
 ('#', '2'),
 (']', '['),
 ('1', ':'),
 ('oh', ','),
 ('black', 'knight')]
```

In [26]:

```
from nltk.corpus import stopwords
from nltk.corpus import webtext
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
stop = (stopwords.words('english'))
stops_filter = lambda w: len(w) < 3 or w in stop
tokens=[t.lower() for t in webtext.words('grail.txt')]
words=BigramCollocationFinder.from_words(tokens)
words.apply_word_filter(stops_filter)
words.nbest(BigramAssocMeasures.likelihood_ratio, 15)
```

Out[26]:

```
[('black', 'knight'),
 ('clop', 'clop'),
 ('head', 'knight'),
 ('mumble', 'mumble'),
 ('squeak', 'squeak'),
 ('saw', 'saw'),
 ('holy', 'grail'),
 ('run', 'away'),
 ('french', 'guard'),
 ('cartoon', 'character'),
 ('iesu', 'domine'),
 ('pie', 'iesu'),
 ('round', 'table'),
 ('sir', 'robin'),
 ('clap', 'clap')]
```

In [27]:
```
text1="Hardwork is the key to success. Never give up!"
word = nltk.wordpunct_tokenize(text1)
finder = BigramCollocationFinder.from_words(word)
bigram_measures = nltk.collocations.BigramAssocMeasures()
value = finder.score_ngrams(bigram_measures.raw_freq)
sorted(bigram for bigram, score in value)
```

Out[27]:
```
[('.', 'Never'),
 ('Hardwork', 'is'),
 ('Never', 'give'),
 ('give', 'up'),
 ('is', 'the'),
 ('key', 'to'),
 ('success', '.'),
 ('the', 'key'),
 ('to', 'success'),
 ('up', '!')]
```

In [31]:
```
text="Hello how are you doing ? I hope you find the book interesting Hello how are
tokens=nltk.wordpunct_tokenize(text)
fourgrams=nltk.collocations.QuadgramCollocationFinder.from_words(tokens)
for fourgram, freq in fourgrams.ngram_fd.items():
    print(fourgram,freq)
```

```
('Hello', 'how', 'are', 'you') 2
('how', 'are', 'you', 'doing') 1
('are', 'you', 'doing', '?') 1
('you', 'doing', '?', 'I') 1
('doing', '?', 'I', 'hope') 1
('?', 'I', 'hope', 'you') 1
('I', 'hope', 'you', 'find') 1
('hope', 'you', 'find', 'the') 1
('you', 'find', 'the', 'book') 1
('find', 'the', 'book', 'interesting') 1
('the', 'book', 'interesting', 'Hello') 1
('book', 'interesting', 'Hello', 'how') 1
('interesting', 'Hello', 'how', 'are') 1
```

In [51]:
```
sent=" Hello , please read the book thoroughly . If you have anyqueries , then don
n = 5
n_grams=ngrams(sent.split(),n)
for grams in n_grams:
    print(grams)
```

```
('Hello', ',', 'please', 'read', 'the')
(',', 'please', 'read', 'the', 'book')
('please', 'read', 'the', 'book', 'thoroughly')
('read', 'the', 'book', 'thoroughly', '.')
('the', 'book', 'thoroughly', '.', 'If')
('book', 'thoroughly', '.', 'If', 'you')
('thoroughly', '.', 'If', 'you', 'have')
('.', 'If', 'you', 'have', 'anyqueries')
('If', 'you', 'have', 'anyqueries', ',')
('you', 'have', 'anyqueries', ',', 'then')
('have', 'anyqueries', ',', 'then', "don't")
('anyqueries', ',', 'then', "don't", 'hesitate')
(',', 'then', "don't", 'hesitate', 'to')
('then', "don't", 'hesitate', 'to', 'ask')
("don't", 'hesitate', 'to', 'ask', '.')
('hesitate', 'to', 'ask', '.', 'There')
('to', 'ask', '.', 'There', 'is')
('ask', '.', 'There', 'is', 'no')
('.', 'There', 'is', 'no', 'shortcut')
('There', 'is', 'no', 'shortcut', 'to')
('is', 'no', 'shortcut', 'to', 'success.')
```

Hidden Markov Model estimation

In [108]:
```
corpus = nltk.corpus.brown.tagged_sents(categories='adventure')[:700]
```

In [109]:
```
len(corpus)
```

Out[109]:
```
700
```

In [110]:
```
from nltk.util import unique_list
tag_set = unique_list(tag for sent in corpus for (word,tag) in sent)
```

In [111]:
```
print(len(tag_set))
```

```
104
```

In [112]:
```
symbols = unique_list(word for sent in corpus for (word,tag) in sent)
```

In [113]:
```
print(len(symbols))
```

```
1908
```

In [114]:
```
trainer = nltk.tag.HiddenMarkovModelTrainer(tag_set, symbols)
train_corpus = []
test_corpus = []
for i in range(len(corpus)):
    if i % 10:
        train_corpus += [corpus[i]]
    else:
        test_corpus += [corpus[i]]
```

In [115]:
```
len(train_corpus)
```

Out[115]:
```
630
```

In [116]:
```
len(test_corpus)
```

Out[116]:
```
70
```

In [117]:
```
def train_and_test(est):
    hmm = trainer.train_supervised(train_corpus, estimator=est)
    print('%.2f%%' % (100 * hmm.evaluate(test_corpus)))
```

In the preceding code, we have created a 90% training and 10% testing file and we have tested the estimator.

Good Turing

Good Turing was introduced by Alan Turing along with his statistical assistant I.J. Good. It is an efficient smoothing method that increases the performance of statistical techniques performed for linguistic tasks, such as word sense disambiguation (WSD), named entity recognition (NER), spelling correction, machine translation, and so on. This method helps to predict the probability of unseen objects. In this method, binomial distribution is exhibited by our objects of interest. This method is used to compute the mass probability for zero or low count samples on the basis of higher count samples . Simple Good Turing performs approximation from frequency to frequency by linear regression into a linear line in log space.

In [118]:
```
gt = lambda fd, bins: SimpleGoodTuringProbDist(fd, bins=1e5)
```

In [119]:
```
gt
```

Out[119]:
```
<function __main__.<lambda>(fd, bins)>
```

In [120]:
```
train_and_test(gt)
```

```
C:\Users\bdhooda\AppData\Local\conda\conda\envs\dfhdfh\lib\site-packages\nltk
\probability.py:1415: UserWarning: SimpleGoodTuring did not find a proper best
fit line for smoothing probabilities of occurrences. The probability estimates
are likely to be unreliable.
  'SimpleGoodTuring did not find a proper best fit '

88.74%
```

Kneser Ney estimation

Kneser Ney is used with trigrams. Let's see the following code in NLTK for the Kneser Ney estimation:

In [121]:
```
corpus = [[((x[0],y[0],z[0]),(x[1],y[1],z[1])) for x, y, z in nltk.trigrams(sent)]
```

In [123]:
```
tag_set = unique_list(tag for sent in corpus for (word,tag) in sent)
```

In [124]:
```
len(tag_set)
```

Out[124]: 906

In [125]:
```
symbols = unique_list(word for sent in corpus for (word,tag) in
sent)
```

In [126]:
```
len(symbols)
```

Out[126]: 1341

In [127]:
```
trainer = nltk.tag.HiddenMarkovModelTrainer(tag_set, symbols)
train_corpus = []
test_corpus = []
for i in range(len(corpus)):
    if i % 10:
        train_corpus += [corpus[i]]
    else:
        test_corpus += [corpus[i]]
```

In [128]:
```
kn = lambda fd, bins: KneserNeyProbDist(fd)
train_and_test(kn)
```

```
0.86%
```

Witten Bell estimation

Witten Bell is the smoothing algorithm that was designed to deal with unknown words having zero probability. Let's consider the following code for Witten Bell estimation in NLTK:

In [129]:
```
train_and_test(WittenBellProbDist)
```

```
6.90%
```

Develop a back-off mechanism for MLE

In [ ]:
```
```