1 Anaconda: Available for Windows, Linux, macOS

It includes hundreds of popular data science packages and the conda package and virtual environment manager for Windows, Linux, and MacOS. Conda makes it quick and easy to install, run, and upgrade complex data science and machine learning environments like scikit-learn, TensorFlow, and SciPy.

- Download link : **https://www.anaconda.com/download/**.
- 1000+ data science packages available.
- Manage packages, dependencies and environments with conda.
- Uncover insights in your data and create interactive visualizations.

2 NLTK: Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with natural language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

2.1 Installing NLTK in Anaconda:

Open Anaconda Prompt and type:

```
conda install -c anaconda nltk
```

2.2 Installing NLTK Data:

NLTK comes with many corpora, toy grammars, trained models, etc.
To install nltk data first install nltk, then Type in Anaconda console:

```
import nltk
nltk.download()
```

3 Corpus management in NLTK:

Corpus available with nltk are: gutenberg, brown, webtext, reuters, inaugural, abc etc.
For finding all the corpus available with nltk. Type below command in Anaconda editor or console:

```
import os
import nltk
print( os.listdir( nltk.data.find("corpora") ) )
```

Table 1: These are some commands to play with the corpus.

Example	Description
fileids()	the files of the corpus
fileids([categories])	the files of the corpus corresponding to these categories
categories()	the categories of the corpus
categories([fileids])	the categories of the corpus corresponding to these files
raw()	the raw content of the corpus
raw(fileids=[f1,f2,f3])	the raw content of the specified files
raw(categories=[c1,c2])	the raw content of the specified categories
words()	the words of the whole corpus
words(fileids=[f1,f2,f3])	the words of the specified fileids
words(categories=[c1,c2])	the words of the specified categories
sents()	the sentences of the whole corpus
sents(fileids=[f1,f2,f3])	the sentences of the specified fileids
sents(categories=[c1,c2])	the sentences of the specified categories
readme()	the contents of the README file of the corpus

For selecting the brown corpus and using the above command type in below code in anaconda editor or console:

```
from nltk.corpus import brown    #selecting brown corpus
brown.fileids()  # provide all the file ids associated with the
    brown corpus
brown.categories() # provide all the categories associated with
    the brown corpus
```

```
brown.raw() # provide raw content of the corpus brown
```

4 Computing the frequency distribution of ngram:

4.1 Code for finding unigram, bigram, trigram from brown corpus:

ngram: This function is available in nltk.utility function which requires two argument. First argument is list, which contains all the brown corpus words and second argument is ngram order(i.e 2 for bigram, 3 for trigram) and returns list generator.
OrderedDict: The dictionary stores the data in key value pairs and maintain the order of the key value pair.
FreqDist: Takes argument as list or list generator and returns dictionary with key as ngrams(i.e bigrams) and value as frequency.

```
from nltk.util import ngrams
import string
corpus=nltk.corpus.brown  # Selecting brown corpus
unigram=[w.strip(string.punctuation).lower() for w in corpus.
    words(categories='news')] # collecting corpus words from news
     category and removing punctuation.
bigram = [a for a in ngrams(unigram,2)] # finding bigrams from
    corpus
trigram = [a for a in ngrams(unigram,3)] # finding trigrams from
     corpus
print(bigram) # prints all the bigrams
```

4.2 Code for finding sorted frequency distribution of ngram:

```
from nltk.corpus import brown # importing brown corpus
from operator import itemgetter
from collections import OrderedDict # importing special
    dictionary which maintains order called OrderedDict
frequency=nltk.FreqDist(ngrams(brown.words(),2)) # Finding
    frequency of bigrams
```

```
sorted_freq=OrderedDict(reversed(sorted(frequency.items(), key =
    itemgetter(1)))) #Sorting bigrams in descending order based
    on the frequency.
```

5 Maximum likelihood estimation of bigram probability:

ConditionalFreqDist: Conditional frequency distributions are used to record the number of times each word occurred, given the context word and it takes list of bigrams or list generator as argument and returns dictionary of dictionary.
ConditionalProbDist:A ConditionalProbDist is constructed from two arguments i.e "ConditionalFreqDist" and a "ProbDist" factory.

```
import nltk
from nltk.corpus import brown
cfreq_2gram = nltk.ConditionalFreqDist(nltk.bigrams(brown.words
    ())) # Calculate frequency of words given context word, For
    example freq(grand|the): Frequency of how many times 'the' is
     followed by 'grand'.
cprob_2gram = nltk.ConditionalProbDist(cfreq_2gram, nltk.
    MLEProbDist) # Calculate probability of words given context
    word, For example prob(grand|the): Probability of how many
    times 'the' is followed by 'grand'.
```

6 Smoothing with Simple Good-Turing and Laplacian(add-1 smoothing):

6.1 Good-Turing:

$$c^* = (c+1) * \frac{N_c + 1}{N_c} \tag{1}$$

c^* =smoothed count of N-grams that occur c times.
$N_c = N_c$ be the number of N-grams that occur c times
Code to implement Good-Turing:

```
import nltk
from nltk.corpus import brown
cfreq_2gram = nltk.ConditionalFreqDist(nltk.bigrams(brown.words
    ()))
```

```
cprob_2gram = nltk.ConditionalProbDist(cfreq_2gram, nltk.
    SimpleGoodTuringProbDist)# simple Good Turing smoothing is
    implemented.
```

6.2 Laplacian:

$$P(w_i|w_(i-1)) = \frac{1 + c(w_{i-1}, w_i)}{c(w_i, *) + |v|} \tag{2}$$

V = vocabulary size. c(x)=count of x in corpus.

```
import nltk
from nltk.corpus import brown
cfreq_2gram = nltk.ConditionalFreqDist(nltk.bigrams(brown.words
    ()))
cprob_2gram = nltk.ConditionalProbDist(cfreq_2gram, nltk.
    LaplaceProbDist) # Laplacian smoothing technique is
    implemented
```

7 Evaluation with perplexity and cross entropy:

The selected corpus is split into training corpus and test corpus. The ngram model is trained on the training corpus and then entropy and perplexity evaluation measure is obtained with the test corpus.

7.1 Entropy:

H(ngrams)= $-\Sigma_{x \in ngrams} p(x) \log(p(x))$

The below function will give information that is produced on the average for each ngrams in the test corpus.

```
def entropy(n, text):                    # n=ngram order,
    text=list of ngrams
        e = 0.0
      text = ["<s>"] + text + ["</s>"]
       for i in range(n - 1, len(text)):
            context = text[i - n + 1:i]
            token = text[i]
            e +=logprob(token, context)
```

```
        return e / float(len(text) - (n - 1))

def logprob(word, context):
    if len(context)==0:
        p=unigram_prob(word)
    elif len(context)==1:
        p=bigram_prob_with_add1smoothing(context[0], word)
    else:
        p=trigram_prob_with_add1smoothing(context[0], context
            [1], word)
    return  - p * log(p , 2)

def perplexity(n, text): # Perplexity function defination
    return pow(2.0, entropy(n,text))

entropy_value_2gram=entropy(2, test_corpus) # Calling entropy
    function with n=2 i.e bigram and test corpus(list of bigrams)
    .

perp_value_2gram=perplexity(2, test_corpus) # Calling perplexity
     function.
```