Libraries

```
In [ ]: # code courtesy of https://www.kaggle.com/alvations/n-gram-language-model-with
        -nltk

        !pip install -U pip
        !pip install -U dill
        !pip install -U nltk==3.4
```

```
In [ ]: import nltk
        from nltk.util import pad_sequence
        from nltk.util import bigrams
        from nltk.util import ngrams
        from nltk.util import everygrams
        from nltk.lm.preprocessing import pad_both_ends
        from nltk.lm.preprocessing import flatten
        from nltk.corpus import reuters
```

For Reutres

```
In [ ]: text =[]
        for sentence in reuters.sents():
            text.append(sentence)
```

```
In [ ]: from nltk.util import pad_sequence
        list(pad_sequence(text[0],
                          pad_left=True, left_pad_symbol="<s>",
                          pad_right=True, right_pad_symbol="</s>",
                          n=2)) # The n order of n-grams, if it's 2-grams, you pad onc
        e, 3-grams pad twice, etc.
```

```
In [ ]: padded_sent = list(pad_sequence(text[0], pad_left=True, left_pad_symbol="<s>",
                                        pad_right=True, right_pad_symbol="</s>", n=2))
        list(ngrams(padded_sent, n=2))
```

```
In [ ]: list(pad_sequence(text[0],
                          pad_left=True, left_pad_symbol="<s>",
                          pad_right=True, right_pad_symbol="</s>",
                          n=3)) # The n order of n-grams, if it's 2-grams, you pad once
        e, 3-grams pad twice, etc.
```

```
In [ ]: padded_sent = list(pad_sequence(text[0], pad_left=True, left_pad_symbol="<s>",
                                        pad_right=True, right_pad_symbol="</s>", n=3))
        list(ngrams(padded_sent, n=3))
```

Building Language Model

In [ ]:
```
from nltk.lm.preprocessing import pad_both_ends
list(pad_both_ends(text[0], n=2))
```

In [ ]:
```
list(bigrams(pad_both_ends(text[0], n=2)))
```

In [ ]:
```
from nltk.util import everygrams
padded_bigrams = list(pad_both_ends(text[0], n=2))
list(everygrams(padded_bigrams, max_len=2))
```

In [ ]:
```
from nltk.lm.preprocessing import flatten
list(flatten(pad_both_ends(sent, n=2) for sent in text))
```

In [ ]:
```
from nltk.lm.preprocessing import padded_everygram_pipeline
train, vocab = padded_everygram_pipeline(2, text)
```

In [ ]:
```
training_ngrams, padded_sentences = padded_everygram_pipeline(2, text)
for ngramlize_sent in training_ngrams:
    print(list(ngramlize_sent))
    print()
print('#############')
list(padded_sentences)
```

For Corpus

In [ ]:
```
import os
import requests
import io #codecs


# Text version of https://kilgarriff.co.uk/Publications/2005-K-lineer.pdf
if os.path.isfile('language-never-random.txt'):
    with io.open('language-never-random.txt', encoding='utf8') as fin:
        text = fin.read()
else:
    url = "https://github.com/praveenjoshi01/COMP9066_27375-Natural-Language-P
rocessing/blob/main/language-never-random.txt"
    text = requests.get(url).content.decode('utf8')
    with io.open('language-never-random.txt', 'w', encoding='utf8') as fout:
        fout.write(text)
```

In [ ]:
```
try: # Use the default NLTK tokenizer.
    from nltk import word_tokenize, sent_tokenize
    # Testing whether it works.
    # Sometimes it doesn't work on some machines because of setup issues.
    word_tokenize(sent_tokenize("This is a foobar sentence. Yes it is.")[0])
except: # Use a naive sentence tokenizer and toktok.
    import re
    from nltk.tokenize import ToktokTokenizer
    # See https://stackoverflow.com/a/25736515/610569
    sent_tokenize = lambda x: re.split(r'(?<=[^A-Z].[.?]) +(?=[A-Z])', x)
    # Use the toktok tokenizer that requires no dependencies.
    toktok = ToktokTokenizer()
    word_tokenize = word_tokenize = toktok.tokenize
```

In [ ]:
```
# Tokenize the text.
tokenized_text = [list(map(str.lower, word_tokenize(sent)))
                  for sent in sent_tokenize(text)]
```

In [ ]:
```
tokenized_text[0]
```

In [ ]:
```
# Preprocess the tokenized text for 3-grams language modelling
n = 3
train_data, padded_sents = padded_everygram_pipeline(n, tokenized_text)
```

In [ ]:
```
from nltk.lm import MLE
model = MLE(n) # Lets train a 3-grams model, previously we set n=3
```

In [ ]:
```
len(model.vocab)
```

In [ ]:
```
model.fit(train_data, padded_sents)
print(model.vocab)
```

In [ ]:
```
len(model.vocab)
```

In [ ]:
```
model.counts['language']
```

Generation using N-gram Language Model

In [ ]:
```
from nltk.tokenize.treebank import TreebankWordDetokenizer

detokenize = TreebankWordDetokenizer().detokenize

def generate_sent(model, num_words, random_seed=42):
    """
    :param model: An ngram language model from `nltk.lm.model`.
    :param num_words: Max no. of words to generate.
    :param random_seed: Seed value for random.
    """
    content = []
    for token in model.generate(num_words, random_seed=random_seed):
        if token == '<s>':
            continue
        if token == '</s>':
            break
        content.append(token)
    return detokenize(content)
```

In [ ]:
```
print(model.generate(28, random_seed=0))
```

In [ ]:
```
generate_sent(model, 20, random_seed=42)
```

Computing Perplexity

In [ ]:
```
test_sentences = ['he mathematics of the random is well']
tokenized_text = [list(map(str.lower, nltk.tokenize.word_tokenize(sent))) for
sent in test_sentences]

test_data = [nltk.bigrams(t,  pad_right=True, pad_left=True, left_pad_symbol="
<s>", right_pad_symbol="</s>") for t in tokenized_text]
for test in test_data:
    print ("MLE Estimates:", [((ngram[-1], ngram[:-1]),model.score(ngram[-1],
ngram[:-1])) for ngram in test])

test_data = [nltk.bigrams(t,  pad_right=True, pad_left=True, left_pad_symbol="
<s>", right_pad_symbol="</s>") for t in tokenized_text]
for i, test in enumerate(test_data):
  print("PP({0}):{1}".format(test_sentences[i], model.perplexity(test)))
```

In [ ]:
```

```