Building a Basic Language Model

Now that we understand what an N-gram is, let's build a basic language model using trigrams of the Reuters corpus.

Reuters corpus is a collection of 10,788 news documents totaling 1.3 million words. We can build a language model in a few lines of code using the NLTK package.

In [1]:
```
from nltk.corpus import reuters
from nltk import bigrams, trigrams
from collections import Counter, defaultdict
import random
```

What is defaultdict?

- A common problem that you can face when working with Python dictionaries is to try to access or modify keys that don't exist in the dictionary. This will raise a KeyError and break up your code execution. To handle these kinds of situations, the standard library provides the Python defaultdict type, a dictionary-like class that's available in collections.
- The Python defaultdict type behaves almost exactly like a regular Python dictionary, but if you try to access or modify a missing key, then defaultdict will automatically create the key and generate a default value for it. This makes defaultdict a valuable option for handling missing keys in dictionaries.

In [2]:
```
# Create a placeholder for model
model = defaultdict(lambda: defaultdict(lambda: 0))
#  it will create a defaultdict with default 0 when a key doesn't exist
```

In [3]:

```
1 print(reuters.sents()[:5])
```

```
[['ASIAN', 'EXPORTERS', 'FEAR', 'DAMAGE', 'FROM', 'U', '.', 'S', '.-', 'JAPAN', 'RIFT', 'Mounting', 'trade', 'fricti
on', 'between', 'the', 'U', '.', 'S', '.', 'And', 'Japan', 'has', 'raised', 'fears', 'among', 'many', 'of', 'Asia',
"'", 's', 'exporting', 'nations', 'that', 'the', 'row', 'could', 'inflict', 'far', '-', 'reaching', 'economic', 'dam
age', ',', 'businessmen', 'and', 'officials', 'said', '.'], ['They', 'told', 'Reuter', 'correspondents', 'in', 'Asia
n', 'capitals', 'a', 'U', '.', 'S', '.', 'Move', 'against', 'Japan', 'might', 'boost', 'protectionist', 'sentiment',
'in', 'the', 'U', '.', 'S', '.', 'And', 'lead', 'to', 'curbs', 'on', 'American', 'imports', 'of', 'their', 'product
s', '.'], ['But', 'some', 'exporters', 'said', 'that', 'while', 'the', 'conflict', 'would', 'hurt', 'them', 'in', 't
he', 'long', '-', 'run', ',', 'in', 'the', 'short', '-', 'term', 'Tokyo', "'", 's', 'loss', 'might', 'be', 'their',
'gain', '.'], ['The', 'U', '.', 'S', '.', 'Has', 'said', 'it', 'will', 'impose', '300', 'mln', 'dlrs', 'of', 'tariff
s', 'on', 'imports', 'of', 'Japanese', 'electronics', 'goods', 'on', 'April', '17', ',', 'in', 'retaliation', 'for',
'Japan', "'", 's', 'alleged', 'failure', 'to', 'stick', 'to', 'a', 'pact', 'not', 'to', 'sell', 'semiconductors', 'o
n', 'world', 'markets', 'at', 'below', 'cost', '.'], ['Unofficial', 'Japanese', 'estimates', 'put', 'the', 'impact',
'of', 'the', 'tariffs', 'at', '10', 'billion', 'dlrs', 'and', 'spokesmen', 'for', 'major', 'electronics', 'firms',
'said', 'they', 'would', 'virtually', 'halt', 'exports', 'of', 'products', 'hit', 'by', 'the', 'new', 'taxes', '.']]
```

In [4]:

```
# Count frequency of co-occurance

for sentence in reuters.sents()[:5]:
    for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
        print(w1,w2,w3)
```

```
None None ASIAN
None ASIAN EXPORTERS
ASIAN EXPORTERS FEAR
EXPORTERS FEAR DAMAGE
FEAR DAMAGE FROM
DAMAGE FROM U
FROM U .
U . S
. S .-
S .- JAPAN
.- JAPAN RIFT
JAPAN RIFT Mounting
RIFT Mounting trade
Mounting trade friction
trade friction between
friction between the
between the U
the U .
U . S
```

- **bi-grams** _T, TE, EX, XT, T_
- **tri-grams** _TE, TEX, EXT, XT_, T__
- **quad-grams** _TEX, TEXT, EXT_, XT__, T___

In [5]:

```
# Count frequency of co-occurance
for sentence in reuters.sents():
    for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
        model[(w1, w2)][w3] += 1
```

In [6]:
```
1 model
```

Out[6]:
```
defaultdict(<function __main__.<lambda>()>,
            {(None,
              None): defaultdict(<function __main__.<lambda>.<locals>.<lambda>()>, {'ASIAN': 4,
                          'They': 446,
                          'But': 1054,
                          'The': 8839,
                          'Unofficial': 1,
                          '"': 3589,
                          'In': 1380,
                          'Threat': 2,
                          'Taiwan': 38,
                          'Retaliation': 3,
                          'A': 764,
                          'Last': 202,
                          'Much': 8,
                          'He': 1586,
                          'Meanwhile': 41,
                          'Japan': 111,
                          'Deputy': 8,
```

In [7]:
```
1 # Let's transform the counts to probabilities
2 for w1_w2 in model:
3     total_count = float(sum(model[w1_w2].values()))
4     for w3 in model[w1_w2]:
5         model[w1_w2][w3] /= total_count
```

In [8]:
```
1 model
```

Out[8]:
```
defaultdict(<function __main__.<lambda>()>,
            {(None,
              None): defaultdict(<function __main__.<lambda>.<locals>.<lambda>()>, {'ASIAN': 7.311144011259162e-0
5,
                                 'They': 0.008151925572553965,
                                 'But': 0.01926486446966789,
                                 'The': 0.16155800478879934,
                                 'Unofficial': 1.8277860028147905e-05,
                                 '"': 0.06559923964102284,
                                 'In': 0.02522344683884411,
                                 'Threat': 3.655572005629581e-05,
                                 'Taiwan': 0.0006945586810696204,
                                 'Retaliation': 5.483358008444371e-05,
                                 'A': 0.013964285061504999,
                                 'Last': 0.0036921277256858768,
                                 'Much': 0.00014622288022518324,
                                 'He': 0.028988686004642578,
                                 'Meanwhile': 0.0007493922611540641,
                                 'Japan': 0.0020288424631244176,
```

We first split our text into trigrams with the help of NLTK and then calculate the frequency in which each combination of the trigrams occurs in the dataset.

We then use it to calculate probabilities of a word, given the previous two words. That's essentially what gives us our Language Model!

Check Language Model

In [9]:
```
1 dict(model["the","news"])
```

Out[9]:
```
{'brought': 0.041666666666666664,
 'about': 0.041666666666666664,
 'with': 0.08333333333333333,
 'of': 0.125,
 'conference': 0.25,
 ',': 0.08333333333333333,
 'broke': 0.041666666666666664,
 '.': 0.125,
 'on': 0.041666666666666664,
 'agency': 0.08333333333333333,
 'that': 0.08333333333333333}
```

In [10]:
```
1 dict(model["today","the"])
```

Out[10]:
```
{'public': 0.05555555555555555,
 'European': 0.05555555555555555,
 'Bank': 0.05555555555555555,
 'price': 0.1111111111111111,
 'emirate': 0.05555555555555555,
 'overseas': 0.05555555555555555,
 'newspaper': 0.05555555555555555,
 'company': 0.16666666666666666,
 'Turkish': 0.05555555555555555,
 'increase': 0.05555555555555555,
 'options': 0.05555555555555555,
 'Higher': 0.05555555555555555,
 'pound': 0.05555555555555555,
 'Italian': 0.05555555555555555,
 'time': 0.05555555555555555}
```

In [11]:
```
1 dict(model["the","price"])
```

Out[11]:
```
{'yesterday': 0.004651162790697674,
 'of': 0.3209302325581395,
 'it': 0.05581395348837209,
 'effect': 0.004651162790697674,
 'cut': 0.009302325581395349,
 'for': 0.05116279069767442,
 'paid': 0.013953488372093023,
 'to': 0.05581395348837209,
 'increases': 0.013953488372093023,
 'used': 0.004651162790697674,
 'climate': 0.004651162790697674,
 '.': 0.023255813953488372,
 'cuts': 0.009302325581395349,
 'reductions': 0.004651162790697674,
 'limit': 0.004651162790697674,
 'now': 0.004651162790697674,
 'moved': 0.004651162790697674,
 'per': 0.013953488372093023,
 'adjustments': 0.004651162790697674,
```

Text Generation Using the Trigram Model

In [21]:

```
# starting words
text = ["the", "news"]
# text = ["today", "the"]
sentence_finished = False

while not sentence_finished:
    # select a random probability threshold
    r = random.random()
    threshold = .0

    for word in model[tuple(text[-2:])].keys():
        threshold += model[tuple(text[-2:])][word]
        # select words that are above the probability threshold
        if threshold >= r:
            text.append(word)
            break

    if text[-2:] == [None, None]:
        sentence_finished = True

print (' '.join([t for t in text if t]))
```

```
the news broke : " I don ' t find him very credible anymore ," said William Sullivan of Dean Witter Reynolds Inc sai
d its board believes that Brazil had raised questions about comments by Rep . Pat Roberts , R . M - 1 / 2 / 87 Prev
Wk 4 / 3 pct in the 1986 / 87 1987 / 88 fiscal year , the U . S . BUDGET DEFICIT NEEDED TO IMPROVE IN 1987
```

In [ ]:

```

```

In [ ]:

```

```