N-граммы — это непрерывные последовательности n-элементов в предложении. N может быть 1, 2 или любым другим положительным целым числом, хотя обычно мы не рассматриваем очень большое N, потому что эти n-граммы редко появляются сразу во многих разных местах.

При выполнении задач машинного обучения, связанных с обработкой естественного языка, мы обычно генерируем n-граммы из входящих предложений. Например, в задачах классификации текста, помимо использования каждого отдельного токена, найденного в корпусе, мы можем захотеть добавить би-граммы или три-граммы в качестве функций для представления наших документов. Этот пост описывает несколько разных способов быстрой генерации n-граммов из входящих предложений в Python.

Код N-граммы на чистом Python

В общем, входное предложение — это просто строка символов. Мы можем использовать встроенные функции в Python для быстрой генерации n-грамм. Давайте возьмем следующее предложение в качестве примера ввода:

```
s = "Natural-language processing (NLP) is an area of computer science " \
    "and artificial intelligence concerned with the interactions " \
    "between computers and human (natural) languages."
```

Если мы хотим сгенерировать список би-грамм из вышеприведенного предложения, ожидаемый результат будет примерно таким, как показано ниже (в зависимости от того, как мы хотим обрабатывать знаки пунктуации, желаемый результат может отличаться):

```
[
    "natural language",
    "language processing",
    "processing nlp",
    "nlp is",
    "is an",
    "an area",
    ...
]
```

Следующая функция может быть использована для достижения этой цели:

```
import re

def generate_ngrams(s, n):
    # Convert to lowercases
    s = s.lower()

    # Replace all none alphanumeric characters with spaces
    s = re.sub(r'[^a-zA-Z0-9\s]', ' ', s)

    # Break sentence in the token, remove empty tokens
    tokens = [token for token in s.split(" ") if token != ""]

    # Use the zip function to help us generate n-grams
    # Concatentate the tokens into ngrams and return
```

```
    ngrams = zip(*[token[:i] for i in range(n)])
    return [" ".join(ngram) for ngram in ngrams]
```

Применение вышеуказанной функции к предложению с n=5 дает следующий результат:

```
>>> generate_ngrams(s, n=5)
['natural language processing nlp is',
 'language processing nlp is an',
 'processing nlp is an area',
 'nlp is an area of',
 'is an area of computer',
 'an area of computer science',
 'area of computer science and',
 'of computer science and artificial',
 'computer science and artificial intelligence',
 'science and artificial intelligence concerned',
 'and artificial intelligence concerned with',
 'artificial intelligence concerned with the',
 'intelligence concerned with the interactions',
 'concerned with the interactions between',
 'with the interactions between computers',
 'the interactions between computers and',
 'interactions between computers and human',
 'between computers and human natural',
 'computers and human natural languages']
```

Вышеупомянутая функция использует функцию zip, которая создает генератор, который агрегирует элементы из множественных списков. Код ниже предлагает более подробное объяснение:

```
# Sample sentence
s = "one two three four five"

tokens = s.split(" ")
# tokens = ["one", "two", "three", "four", "five"]

sequences = [tokens[i:] for i in range(3)]
# The above will generate sequences of tokens starting from different
# elements of the list of tokens.
# The parameter in the range() function controls how many sequences
# to generate.
#
# sequences = [
#   ['one', 'two', 'three', 'four', 'five'],
#   ['two', 'three', 'four', 'five'],
#   ['three', 'four', 'five']]

bigrams = zip(*sequences)
# The zip function takes the sequences as a list of inputs (using the * operator,
# this is equivalent to zip(sequences[0], sequences[1], sequences[2]).
# Each tuple it returns will contain one element from each of the sequences.
#
# To inspect the content of bigrams, try:
# print(list(bigrams))
# which will give the following:
#
# [('one', 'two', 'three'), ('two', 'three', 'four'), ('three', 'four', 'five')]
#
# Note: even though the first sequence has 5 elements, zip will stop after returning
# 3 tuples, because the last sequence only has 3 elements. In other words, the zip
```

```
# function automatically handles the ending of the n-gram generation.
```

Использование NLTK в Python

Вместо использования чистых функций Python, мы также можем получить воспользоваться библиотеками обработки естественного языка, таких как Natural Language Toolkit (NLTK). В частности, в nltk есть функция ngrams которая возвращает генератор n-грамм с заданным предложением.

```
import re
from nltk.util import ngrams

s = s.lower()
s = re.sub(r'[^a-zA-Z0-9\s]', ' ', s)
tokens = [token for token in s.split(" ") if token != ""]
output = list(ngrams(tokens, 5))
```

Приведенный выше блок кода сгенерирует те же данные, что и функция generate_ngrams() как показано выше.