N-GRAMM TIL MODELLARI VOSITASIDA O`ZBEK TILIDA MATN GENERATSIYA QILISH

Elov Botir Boltayevich,
elov@navoiy-uni.uz
Alisher Navoiy nomidagi Toshkent davlat o‘zbek tili va adabiyoti
Kompyuter lingvistikasi va raqamli texnologiyalar kafedrasi mudiri
texnika fanlari bo‘yicha PhD

Annotatsiya. Ushbu maqola tabiiy tilni qayta ishlash (NLP) vositalari (n-grammalar) orqali o‘zbek tilida matnlarni generatsiya qilish usullari keltiriladi. Matn generatsiya qiluvchi dasturiy ta’minonning xususiyatlari, Python imkoniyatlari, N-gramm model, unigram, bigram, trigram va matn korpusi yordamida matn generatsiya qilish algoritmi yoritilgan.

Kalit so‘zlar: *Tabiiy tilni qayta ishlash, NLP, Python, matn korpusi, N-gramm model, unigram, bigram, trigram.*

Annotation. This article describes the methods of generating texts in Uzbek using natural language processing (NLP) tools (n-grams). Features of text generating software, Python capabilities, N-gram model, unigram, bigram, trigram and text generation algorithm are described.

Keywords: *Natural language processing, NLP, Python, text corpus, N-gram model, unigram, bigram, trigram.*

Аннотация. В данной статье описаны методы генерации текстов на узбекском языке с использованием инструментов обработки естественного языка (NLP) (н-грамм). Описаны особенности программного обеспечения для генерации текста, возможности Python, модель N-грамм, униграмма, биграмма, триграмма и алгоритм генерации текста.

Ключевые слова: *обработка естественного языка, NLP, Python, корпус, модель N-грамм, униграмма, биграмма, триграмма.*

Kirish. Hozirda axborot tizimlari va Internetning turli sohalarga tobora keng qo`llanilishi natijasida katta hajmdagi ma`lumotlar hosil bo`lmoqda. Shuning uchun O`zbek kontekstida ma`lumotlarni avtomatik qayta ishlash va qidirish ehtiyojga aylanib bormoqda. Ushbu maqolada NLP sohasidagi ba`zi asosiy vositalarni ishlab chiqish uchun N-gramm yondashuvini muhokama qilamiz. Ushbu yondashuv

statistik va sun`iy intellekt vositalariga asoslangan. Mashinalar vositasida oddiy statistik va ehtimollik usullaridan foydalangan holda yangi matn yaratish sun`iy intellektning tabiiy tilni qayta ishlash (NLP) sohasidagi ilmiy izlanishlarning asosiy yo`nalishlaridan biri hisoblanadi. Ushbu maqolada matn yaratish modelini yaratishning juda oddiy va intuitiv usullari ko`rib chiqiladi.

Bugungi kunda jahonda bir qator olimlar NLP vositalari orqali matnlarni avtomatik generatsiya qilish ustida ilmiy izlanishlar olib bormoqda. Jumladan, Mehmet Ali Kutlugun va Yahya Shirin tomonidan turk tilidagi matnlarni generatsiya qilish tizimlarini yaratish uchun katta hajmdagi ma`lumotlardagi n-grammalardan foydalangan holda ma`noli yangi turkiy matnlarni yaratish usullari taklif qilingan [1]. Bunda yangi matnni generatsiya qilishda trigramma modelidan foydalanilgan va gapdagi so`z yoki so`z birikmalari asosida unikal matnlar hosil qilingan. A.K.Yadav va S. K.Borgohain berilgan so`zlar to`plamidan yangi matnni generatsiya qilishda n-gramm modeli asosida *matn korpusi*dan foydalanib til modelini yaratish masalasi ko`rib chiqilgan [2] va DFS (Depth First Search) filtrlash texnikasini qo`llanilgan. Shunigdek, ikkita korpus (*matnli korpus* va *POS teglarining izohli korpusi*)dan foydalanib, izohlangan korpusdan barcha yaroqli POS trigramma teglari hosil qilingan. Aniqlangan nomzodlar ketma-ketligining har biri ehtimollik balli hisonlanib, trigram POS yorlig`i va ehtimollik balliga moslashtirish orqali tartiblangan va yangi matn generatsiya qilingan. I.Ashwini va B.M.Sagar tomonidan n-grammalar asosida parafrazalarni generatsia qilish masalalari ko`rib chiqilgan [3]. K.G. Srinivasa va B.N.Shree Devi tomonidan sun`iy intellekt metodlaridan hisoblangan GPUga asoslangan N-gramm satrlarni moslashtirish algoritmi va katta hajmdagi hujjatlarda satrlarni izlash uchun ballar jadvali yondashuviga asoslangan [4]. Katta hajmdagi hujjatlarda satrlarni izlash va uning GPUni amalga oshirish uchun N-gramm modeliga optimallashtirilgan yangi yondashuvni taklif qilingan. Algoritm GPGPU-lardan ko'plab hujjatlardagi satrlarni qidirish uchun N-gramm belgilar darajasidagi *parallel Skorlar jadvali yondashuvi* va *CUDA API* yordamida qidirish uchun foydalanadi. E.Mulyani tominidan polinomial bayes metodidan foydalangan holda, N-grammlar orqali martndagi xususiyatlarni ajratib olish va tanlash usullari tahlil qililgan [5].

Dasturiy ta`minot hujjatlarini ishlab chiqish ko`pincha nusxa ko`chirishni o`z ichiga oladi, bu esa juda ko`p takroriy matnlarni hosil qilinishiga olib keladi. Bunday dublikatlar, ayniqsa, dasturiy ta`minot va uning hujjatlarining ishlash muddati uzoq bo`lgan taqdirda, hujjatlarni saqlashni qiyin va qimmat qiladi. Vaziyat ikki

nusxadagi ma`lumotlarning tez-tez takroriy bo`lishi bilan yanada murakkablashadi, ya`ni bir xil ma`lumotlar turli darajadagi tafsilotlar bilan, turli kontekstlarda va hokazolarda ko`p marta taqdim etilishi mumkin. D.V.Luciv, L.D.Kanteev, Yu.O.Kostyukov, D.V. Koznov, M.N.Smirnovlar tomonidan ishlab chiqilgan N-gramm modeli tabiiy tilni qayta ishlash texnikasidan foydalangan holda dasturiy ta`minot hujjatlarida yaqin dublikatlarni aniqlash imkonini beradi [6].

O‘zbek tilida yangi matnni generatsiya qilish bo`yicha ilmiy amaliy ishlar deyarli olib borilmagan. Shu sababli, yuqorida keltirilgan fikr mulohazalardan foydalanib *o`zbek tilida yangi matnni generatsiya qilish masalasini* ko`rib chiqamiz. Barchaga ma`lumki, tabiiy tillarda matndagi so`zlarning yozilish (qo`llanish) tartibi muhim ahamiyatga ega. Ushu xususiyat gapdagi ba`zi no`malum so`zlarni tushunmasdan ham bizga gapning kontekstini tushunishga imkon beradi. Quyidagi misolni ko`rib chiqamiz:

"Anvar **Gargula** tomonidan chiqarilgan dahshatli shovqin tufayli qo`rqib ketdi."

Oldingi kontekstsiz biz **"Gargula"** nima ekanligini bilmaymiz. Gap tarkibidagi so`zlarning bunday bog`liqligi bizga tushunmayotgan so`zning tabiati haqida ba`zi ma`lumotlar berishi mumkin. Ba`zi hollarda biz butun kontekstni bilishimiz shart emas. Yuqoridagi misolda, biz intuitiv darajada faqat "tashqari shovqin" ga qarab, keyingi so`z nutqning boshqa qismi emas, balki **ot so`z turkumiga mansub so`z** bo`lishi lozimligini aytishimiz mumkin.

Yuqoridagi mulohazalardan **N-grammlar** deb nomlanuvchi berilgan gapni n ta elementning ketma-ketligidan iborat qismlarga ajratish qoyasi kelib chiqadi. Asosiy g`oya shundan iboratki, har qanday matn berilgan bo`lsa, biz **unigramlar** *(1 gramm)*, **bigramlar** *(2 gramm)*, **trigramlar** *(3 gramm)* va boshqalar ro`yxatiga ajratishimiz mumkin [8-10]. Misol uchun:

Matn: "Men kitobni o`qidim"
Unigramlar: *[(Men), (kitobni), (o`qidim)]*
Bigrammalar: *[((Men, kitobni)), (kitobni, o`qidim)]*

Ko`rib turganingizdek, "kitobni" so`zi bigrammda quyidagicha shakllantirilgan: *(men, kitobni)* va *(kitobni, o`qidim)*.

1-rasm. Trigammalar

N-gramm mashinali o`rganish sohasida tushunish uchun eng oson tushunchadir. Tabiiy tilni qayta ishlashda yoki qisqacha NLPda n-grammlar gaplarni avtomatik ravishda yakunlashni, avtomatik imloni tekshirishni va ma`lum darajada, biz berilgan gapni grammatikasini tekshirishimiz mumkin.

N-gramm

N-gramm yordamida matn yaratishning asosiy g`oyasi n-grammning oxirgi so`zini **(x^n)** bir xil n-grammda **(x^{n-1}, x^{n-2}, … x^1)** hosil qilingan boshqa so`zlar ketma ketligidan hosil qilish mumkin. Shunday qilib, modelning asosiy soddalashtirilganligi shundan iboratki, keyingi so`zni shakllantirish uchun butun gapni tahlil qilish shart emas. Faqat **n-1** tokenlardan iborat ketma-ketlikni qidirishimiz lozim [11]:

$$P(x^{(t+1)}| x^{(t)},\dots, x^{(1)})=P(x^{(t+1)}| x^{(t)},\dots, x^{(t-n+2)})$$

$$P(x^{t+1}| x^{t},\dots,x^{t-n+2}) = \frac{P(x^{t+1},x^{t},\dots,x^{t-n+2})}{P(x^{t},\dots,x^{t-n+2})}$$

Masalan: *trigram modelidan foydalanish (n=3)*

Matn: *“Mary was scared because of ___”*

Biz trigramma modelidan foydalanganimiz uchun jumlaning boshini tushiramiz: **"Mary was scared"** va **"because of"** dan mumkin bo`lgan davomini shakllantirishimiz lozim. Ma`lumotlar bazasidan biz quyidagi mumkin bo`lgan davomlar mavjudligini bilamiz deb taxmin qilamiz: **"me"**, **"noise"**.

Shunday qilib, biz hisoblashimiz kerak:

P(noise | because of) va *P(me | because of)*

Ehtimollar hisoblab chiqilgandan so`ng, barcha *nomzodlar* uchun yakuniy so`zni tanlashning bir necha yo`li mavjud. Buning usullaridan **biri** eng yuqori shartli ehtimolga ega bo`lgan so`zni aniqlash bo`lishi mumkin (*bu variant eng yaxshisi*

bo`lmasligi mumkin, chunki u **n** kichik bo`lsa, cheksiz sikl yuzaga kelish ehtimoli mavjud).

Boshqa (va yaxshiroq) variant **"yarim tasodifiy"** so`zni shartli ehtimollik nuqtai nazaridan chiqarishdir. Shunday qilib, ehtimolligi yuqori bo`lgan so`zlarning hosil bo`lish ehtimoli yuqori bo`ladi, ehtimolligi past bo`lgan boshqa so`zlar esa hosil bo`lish imkoniyatiga ega bo`ladi.

Tadbiq qilish

Avvalo, bizga **til modelimizni** o`rgatmoqchi bo`lgan manba (**matn)** kerak. Ushbu amalni bajarish uchun ideal holda katta hajmdagi kitobga (hatto bir nechta kitobga) ega bo`lishni xohlaymiz, chunki biz nafaqat katta lug`atga ega bo`lishni xohlaymiz, balki iloji boricha ko`proq turli xil almashtirish (*n-gramm*) yoki so`zlarni ko`rishni xohlaymiz. Ushbu amalni 2 xil usulda bajarish munkin:

- *mavjud NLP paketlari yordamida [12];*
- *Python tilida yozilgan yangi funksyalar orqali [13-15].*

Ushbu maqolada mavjud NLP paketlari yordamida n-grammlardan foydalanib o`zbek tilida yangi mattni generatsiya qilish masalasini ko`rib chiqamiz. Ko`p hollarda, biz katta hajmdagi matnda mavjud so`zlar statistikasini hisobga olgan holda, berilgan so`zdan keyin qaysi so`z kelishini aniqlovchi til modellarini yaratish uchun **n-grammdan** foydalanishimiz mumkin. *Neyron bo`lmagan tilni* modellashtirish qanday amalga oshirilganligini tushunish uchun biz Python tilida yozilgan **NKTK** *(Natural Language Toolkit)* dagi **lm moduli**dan foydalanamiz.

```
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
```

Berilgan matndagi **bigramma**larni aniqlab olish uchun **NLTK**ning *grams()* funktsiyasidan foydalanish lozim:

```
text = [['Maqola', 'NLPga', 'asoslangan'], ['Ushbu', 'hujjatning', 'istalgan', 'satrini', 'tokenlarga', 'ajratish', 'usullari', 'keltiriladi']]
print(list(bigrams(text[0])))
```

```
[('Maqola', 'NLPga'), ('NLPga', 'asoslangan')]

print(list(ngrams(text[1], n=3)))
_____________________________________
[('Ushbu', 'hujjatning', 'istalgan'), ('hujjatning', 'istalgan', 'satrini'), ('istalgan', 'satrini', 'tokenlarga'), ('satrini', 'tokenlarga', 'ajratish'), ('tokenlarga', 'ajratish', 'usullari'), ('ajratish', 'usullari', 'keltiriladi')]
```

Yuqoridagi birinchi misolda "NLPga" element turli bigrammalarning birinchi va ikkinchi a`zosi sifatida kelgan. Gaplar "Maqola" so`zi bilan boshlanib, "asoslangan" so`zi bilan tugash statistikasini aniqlash lozim bo`lsin. Buni hal qilishning standart usuli – jumlani grammlarga boëlishdan oldin unga maxsus **"to'ldirish"** belgilarini qo`shishdir. **NLTK** paketida ushbu masalani hal iqluvchi funktsiya mavjud.

```
from nltk.util import pad_sequence
print(list(pad_sequence(text[0],
            pad_left=True, left_pad_symbol="<s>",
            pad_right=True, right_pad_symbol="</s>",
            n=2))) # n-grammning n-tartibi, agar u 2 gramm bo'lsa, siz bir marta, 3 grammni ikki marta va hokazo.
_____________________________________
['<s>', 'Maqola', 'NLPga', 'asoslangan', '</s>']

padded_sent = list(pad_sequence(text[0], pad_left=True, left_pad_symbol="<s>",
                    pad_right=True, right_pad_symbol="</s>", n=2))
print(list(ngrams(padded_sent, n=2)))
_____________________________________
[('<s>', 'Maqola'), ('Maqola', 'NLPga'), ('NLPga', 'asoslangan'), ('asoslangan', '</s>')]

print(list(pad_sequence(text[0],
            pad_left=True, left_pad_symbol="<s>",
            pad_right=True, right_pad_symbol="</s>",
            n=3))) # N-grammning n-tartibi, agar u 2 gramm bo'lsa, siz bir marta, 3 grammni ikki marta va hokazo.
_____________________________________
['<s>', '<s>', 'Maqola', 'NLPga', 'asoslangan', '</s>', '</s>']

padded_sent = list(pad_sequence(text[0], pad_left=True, left_pad_symbol="<s>",
                    pad_right=True, right_pad_symbol="</s>", n=3))
print(list(ngrams(padded_sent, n=3)))
_____________________________________
[('<s>', '<s>', 'Maqola'), ('<s>', 'Maqola', 'NLPga'), ('Maqola', 'NLPga', 'asoslangan'), ('NLPga', 'asoslangan', '</s>'), ('asoslangan', '</s>', '</s>')]
```

Dastur kodidagi **n** argumenti funktsiya bigramlar uchun to`ldirish kerakligini ifodalaydi. Ushbu parametrlarni barchasini har safar uzatishda ularni *default* qiymay bilan qabul qilish mumkin. Shunday qilib, **nltk.lm** modulida ushbu amalni

bajaradigan funksiya mavjud bo`lib, boshqa argumentlar esa **pad_sequence** uchun bir xil bo`lib qoladi.

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

```
['<s>', 'Maqola', 'NLPga', 'asoslangan', '</s>']
```

Yuqoridagi fikr mulohazalardan kelib chiqib, bitta gap uchun quyidagi natijaga ega bo`lamiz:

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

```
[('<s>', 'Maqola'), ('Maqola', 'NLPga'), ('NLPga', 'asoslangan'), ('asoslangan', '</s>')]
```

Modelimizni yanada mustahkam qilish uchun **unigramma** *(bitta so`z)* va uning asosiy ma`lumot manbai bo`lgan *bigrammalarga* o`rgatish uchun NLTK paketida **Everygram** deb nomlangan funktsiyadan foydalanish lozim:

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

```
[('<s>',), ('<s>', 'Maqola'), ('Maqola',), ('Maqola', 'NLPga'), ('NLPga',), ('NLPga', 'asoslangan'), ('asoslangan',), ('asoslangan', '</s>'), ('</s>',)]
```

n-grammlarni aniqlashda model uchun **"ma`lum" so`zlarni** aniqlaydigan lug`atga asoslaniladi. Ushbu lug`atni yaratish uchun biz gaplardagi so`zlarni umumiy qilib aniqlab olishimiz lozim.

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

```
['<s>', 'Maqola', 'NLPga', 'asoslangan', '</s>', '<s>', 'Ushbu', 'hujjatning', 'istalgan', 'satrini', 'tokenlarga', 'ajratish', 'usullari', 'keltiriladi', '</s>']
```

Ko`p hollarda *lug`at* va *n-gramlarni* hisoblash uchun manba sifatida bir xil matndan foydalanishga to`g`ri keladi. Matnni xotirada qayta yaratmaslik uchun **train** va

vocab iteratorlaridan foydalanamiz va **padded_everygram_pipeline** natijani iteratorlarni ro`yxatga kiritish orqali **“materiallashtiramiz”**:

```
from nltk.lm.preprocessing import padded_everygram_pipeline
train, vocab = padded_everygram_pipeline(2, text)
training_ngrams, padded_sentences = padded_everygram_pipeline(2, text)
for ngramlize_sent in training_ngrams:
   print(list(ngramlize_sent))
   print()
print(list(padded_sentences))
```

```
[('<s>',), ('<s>', 'Maqola'), ('Maqola',), ('Maqola', 'NLPga'), ('NLPga',), ('NLPga', 'asoslangan'), ('asoslangan',), ('asoslangan', '</s>'), ('</s>',)]
[('<s>',), ('<s>', 'Ushbu'), ('Ushbu',), ('Ushbu', 'hujjatning'), ('hujjatning',), ('hujjatning', 'istalgan'), ('istalgan',), ('istalgan', 'satrini'), ('satrini',), ('satrini', 'tokenlarga'), ('tokenlarga',), ('tokenlarga', 'ajratish'), ('ajratish',), ('ajratish', 'usullari'), ('usullari',), ('usullari', 'keltiriladi'), ('keltiriladi',), ('keltiriladi', '</s>'), ('</s>',)]
['<s>', 'Maqola', 'NLPga', 'asoslangan', '</s>', '<s>', 'Ushbu', 'hujjatning', 'istalgan', 'satrini', 'tokenlarga', 'ajratish', 'usullari', 'keltiriladi', '</s>']
```

Berilgan matnni olamiz va uni tokenlashtiramiz:

```
import re
from nltk.tokenize import ToktokTokenizer
sent_tokenize = lambda x: re.split(r'(?<=[^A-Z].[.!?]) +(?=[A-Z])', x)
toktok = ToktokTokenizer()
word_tokenize = word_tokenize = toktok.tokenize

import os
import requests
import io
# Standart NLTK tokenizerdan foydalanish. Zarur paketlarni yuklab olish
if os.path.isfile('D:\Examples\Examples.txt'):
   with io.open('D:\Examples\Examples.txt', encoding='utf8') as fin:
      text = fin.read()

tokenized_text = [list(map(str.lower, word_tokenize(sent)))
            for sent in sent_tokenize(text)]
print(tokenized_text[0])
```

```
['ushbu', 'maqolada', 'hujjatni', '(', 'matnni', ')', 'uning', 'istalgan', 'satrini', 'alohida', 'qismlarga', '(', 'tokenlarga', ')', 'ajratish', 'usullari', 'keltiriladi', '.']
```

N-gramm modelini o`rgatish

Yuqoridagi kelitirlgan fikr mulohazalardan kelib chiqib, modelni tayyorlashni boshlash mumkin. Oddiy misol sifatida, keling, **Maksimal ehtimollik hisoblagichini** *(Maximum Likelihood Estimator, MLE)* o`rgataylik. N-gramm bilan ishlatiladigan ehtimollikni ko'rib chiqishimiz kerak. Uni yaratish uchun faqat eng yuqori *n-gramm tartibini* belgilashimiz kerak.

```
from nltk.lm.preprocessing import padded_everygram_pipeline
# 3 grammli tilni modellashtirish uchun tokenlashtirilgan matn
n = 3
train_data, padded_sents = padded_everygram_pipeline(n, tokenized_text)

from nltk.lm import MLE
model = MLE(n) # 3-grammli modelni o'rgataylik
```

MLE modeli ishga tushirilgach, bo`sh lug`at yaratadi. Modelga mos keladigan tarzda to`ldiriladi:

```
model.fit(train_data, padded_sents)
print(model.vocab)
```

```
<Vocabulary with cutoff=1 unk_label='<UNK>' and 319 items>
```

Lug`at bizga mashg`ulot paytida uchramagan so`zlarni boshqarishga yordam beradi.

```
print(model.vocab.lookup(tokenized_text[0]))
```

```
('ushbu', 'maqolada', 'hujjatni', '(', 'matnni', ')', 'uning', 'istalgan', 'satrini', 'alohida', 'qismlarga', '(', 'tokenlarga', ')', 'ajratish', 'usullari', 'keltiriladi', '.')
```

Agar biz lug`atni o`quv ma`lumotlaridan emas, balki ko`rinmaydigan gaplardan qidirsak, lug`atda bo`lmagan so`zlarni avtomatik ravishda **<UNK>** bilan almashtiradi.

```
print(model.vocab.lookup('Har doim maqolada yozishda modellarini yaratishdan foydalaniladi.'.split()))
```

```
('<UNK>', 'doim', 'maqolada', '<UNK>', 'modellarini', '<UNK>', '<UNK>')
```

Shuningdek, ba`zi hollarda biz mashg`ulot paytida ko`rgan, lekin tez-tez uchramaydigan so`zlarni e`tiborsiz qoldirmoqchi bo`lsak, bu bizga foydali ma`lumot beradi. **Unk_cutoff** argumentidan foydalanib, lug`atga bunday so`zlarni e`tiborsiz qoldirishni ko`rsatish mumkin. Murakkab n-gram modellari uchun **nltk.lm.models** paketidagi quyidagi ob`ektlarni ko`rib chiqish lozim:

- **Lidstone:** *Lidstone bilan tekislangan ballarni beradi;*
- **Laplas:** *Laplas (bir qo`shing) silliqlashni amalga oshiradi;*
- **InterpolatedLanguageModel:** *Barcha interpolyatsiya qilingan til modellari uchun umumiy mantiq;*
- **WittenBellInterpolated:** *Witten-Bell silliqlashning interpolyatsiya qilingan versiyasi.*

N-gramm til modelidan foydalanish

N-gram modellari haqida gap ketganda, mashg`ulotlar o`quv (til) korpusidagi grammlarni hisoblashgacha tushadi.

```
print(model.counts)
```

```
<NgramCounter with 3 ngram orders and 2262 ngrams>
```

Bu unigramlar sonini aniqlash uchun qulay interfeysni taqdim etadi:

```
print(model.counts['ajratish']) # 'ajratish' lar soni
```

```
6
```

va "ajratish kerak" so`z birikmasi uchun bigrammalar:

```
print(model.counts[['ajratish']]['kerak']) # 'ajratish'|'kerak' lar soni
```

```
2
```

va "kabi ajratish kerak" so`z birikmasi uchun trigrammalar:

```
print(model.counts[['kabi', 'ajratish']]['kerak']) # i.e. Count('kabi ajratish'|'kerak')
```

```
2
```

Biroq, til modelini o`rgatishning asl maqsadi ma`lum kontekstlarda so`zlarning qanchalik ehtimoli mavjudligini aniqlashdir. Bu **MLE** bo`lib, model elementning nisbiy chastotasini ball sifatida qaytaradi:

```
print(model.score('ajratish'))
```

```
print(model.score('lozim','topish'.split()))
```

```
0.008849557522123894
1.0
```

O`qitish davomida ko`rinmaydigan elementlar lug`atning **"noma'lum yorlig'i"** belgisi bilan taqqoslanadi *(default qiymat – "")*.

```
print(model.score("<UNK>") == model.score("oddiy"))
print(model.score("<UNK>") == model.score("doim"))
print(model.score("<UNK>") == model.score("har"))
```

```
True
False
False
```

Qulaylik uchun buni **logscore** usuli yordamida **logarifmini** amalga oshirish mumkin:

```
print(model.logscore("ajratish", "kerak".split()))
```

```
-0.6520766965796932
```

N-gramm til modeli yordamida generatsiya qilish

N-gram modellarining ajoyib xususiyati shundaki, ularni matn yaratish uchun ishlatilishi mumkin. Biz yaratilgan tokenlarni tilde mavjud bo`lgan gaplagra o`xshatish uchun **tozalashni** amalga oshirishimiz lozim.

```
print(model.generate(10, random_seed=5))
```

```
['ma', '`', 'noli', 'so', '`', 'zlarning', 'bigramm', 'va', 'trigrammlarni', 'aniqlash']
```

Yaratilgan tokenlarni biz “o`rgangan” gaplarga o`xshashi uchun ularni “tozalashimiz” mumkin:

```
def generate_sent(model, num_words, random_seed=42):
    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)

print(generate_sent(model, 10, random_seed=5))
```

ma ` noli so ` zlarning bigramm va trigrammlarni aniqlash

Modelni saqlash

Python tilida modelidagi lambda funksiyalar saqlanmasligi mumkin. Shuning uchun biz til modelini saqlash va yuklash uchun **pickle** o`rniga **dill** kutubxonasidan \foydalanishimiz mumkin:

```
import dill as pickle
with open('kilgariff_ngram_model.pkl', 'wb') as fout:
    pickle.dump(model, fout)
with open('kilgariff_ngram_model.pkl', 'rb') as fin:
    model_loaded = pickle.load(fin)
print(generate_sent(model_loaded, 10, random_seed=42))
```

matndan kerakli ma ` noli so ` z borligini tushunasiz

Xulosa

Ushbu maqolada biz n-grammlarning o‘ziga xos xusussiyati va Pythonda berilgan matnli ma’lumotlar to‘plami uchun istalgan miqdordagi n-grammlarni generatsiya qilish usullari tavsiflandi. N-gramm va oddiy ehtimollik qoidalaridan foydalanadigan holda berilgan matn(lar) asosida shakllantiriladigan soddalashtirilgan til modeli taqdim etildi. Shuningdek, ushbu yaratilgan model asosida qandaydir ma’noga ega bo‘lishi mumkin bo‘lgan *yangi matn yaratish* usullari keltirildi. Albatta, bunday sodda model insonga o‘xshash maqolalar yoza olmaydi yoki **GPT-3** kabi ishlay olmaydi. Bunday tizim siklga tushib qolishi mumkin yoki uzoq muddatli kontekstli munosabatlarni kuzatib borishga qodir emas. Ammo bu o‘zbek tili misolida tabiiy tilni qayta ishlash va tilni modellashtirishga qaratilgan boshlang‘ch qadam hisoblanadi. Keyingi qadamda sun’iy intellekt metodlari orqali tabiiy tilni qayta ishlash masalalarichi yechish usullari amalga oshiriladi.

Foydalanilgan adabiyotlar:

1. M.A. Kutlugun, Y. Sirin (2018). Turkish meaningful text generation with class based n-gram model | Sinif tabanli N-gram modeli ile Türkçe anlamli metin üretme. *26th IEEE Signal Processing and Communications Applications Conference, SIU 2018 (2018)*

2. Yadav, A. K., & Borgohain, S. K. (2015). Sentence generation from a bag of words using N-gram model. *Proceedings of 2014 IEEE International Conference on Advanced Communication, Control and Computing Technologies, ICACCCT 2014*. https://doi.org/10.1109/ICACCCT.2014.7019414
3. Gadag, A. I., & Sagar, B. M. (2016). N-gram based paraphrase generator from large text document. *2016 International Conference on Computation System and Information Technology for Sustainable Solutions, CSITSS 2016*. https://doi.org/10.1109/CSITSS.2016.7779447
4. Srinivasa, K. G., & Shree Devi, B. N. (2017). GPU Based N-Gram String Matching Algorithm with Score Table Approach for String Searching in Many Documents. *Journal of The Institution of Engineers (India): Series B*, *98*(5). https://doi.org/10.1007/s40031-017-0295-3
5. Mulyani, E., Muhamad, F. P. B., & Cahyanto, K. A. (2021). Pengaruh N-Gram terhadap Klasifikasi Buku menggunakan Ekstraksi dan Seleksi Fitur pada Multinomial Naïve Bayes. *JURNAL MEDIA INFORMATIKA BUDIDARMA*, *5*(1). https://doi.org/10.30865/mib.v5i1.2672
6. Kanteev, L. D., Kostyukov, Yu. O., Luciv, D. V., Koznov, D. V., & Smirnov, M. N. (2017). Discovering Near Duplicate Text in Software Documentation. *Proceedings of the Institute for System Programming of the RAS*, *29*(4), 303–314. https://doi.org/10.15514/ispras-2017-29(4)-21
7. Breckle, M., & Zinsmeister, H. (2012). A corpus-based contrastive analysis of local coherence in {L}1 and {L}2 {German}. *Discourse and {Dialogue} / {Diskurs} Und {Dialog}*.
8. Jabbar, A., Iqbal, S., Akhunzada, A., & Abbas, Q. (2018). An improved Urdu stemming algorithm for text mining based on multi-step hybrid approach. *Journal of Experimental and Theoretical Artificial Intelligence*, *30*(5). https://doi.org/10.1080/0952813X.2018.1467495
9. Kalaivani, K. S., Kuppuswami, S., & Kanimozhiselvi, C. S. (2019). Use of NLP based combined features for sentiment classification. *International Journal of Engineering and Advanced Technology*, *9*(1). https://doi.org/10.35940/ijeat.F8290.109119
10. Litvinova, T. A., Seredin, P. v., & Litvinova, O. A. (2015). Using part-of-speech sequences frequencies in a text to predict author personality: A corpus study. *Indian Journal of Science and Technology*, *8*, 93–97. https://doi.org/10.17485/ijst/2015/v8iS9/51103
11. Kouremenos, D., Ntalianis, K., & Kollias, S. (2018). A novel rule based machine translation scheme from Greek to Greek Sign Language: Production of different types of large corpora and Language Models evaluation. *Computer Speech and Language*, *51*. https://doi.org/10.1016/j.csl.2018.04.001
12. Thanaki, J. (2017). Python Natural Language Processing. In *Packt Publishing Ltd*.
13. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, *2020-December*.
14. Dale, R. (2021). GPT-3: What’s it good for? In *Natural Language Engineering* (Vol. 27, Issue 1). https://doi.org/10.1017/S1351324920000601
15. Wallace, E., Gardner, M., & Singh, S. (2020). *Interpreting Predictions of NLP Models*. https://doi.org/10.18653/v1/2020.emnlp-tutorials.3