O‘ZBEK TILI KORPUSI MATNLARI ASOSIDA TIL MODELLARINI YARATISH

Elov Botir
Texnika fanlari falsafa doktori, dotsent.
Alisher Navoiy nomidagi
Toshkent davlat o‘zbek tili va adabiyoti universiteti.
E-mail: elov@navoiy-uni.uz
ORCID: 0000-0001-5032-6648

Abdullayev Abdulla
O‘qituvchi. “Urganch innovatsion university”
NTM ta’limni kredit tizimini boshqarish bo‘lim boshlig‘i.
E-mail: abdulla_abdullayev9270@mail.ru

Xudayberganov Nizomaddin
O‘qituvchi. Alisher Navoiy nomidagi
Toshkent davlat o‘zbek tili va adabiyoti universiteti.
E-mail: nizomaddin@navoiy-uni.uz
ORCID: 0000-0002-6213-3015

Annotatsiya: Nutqni tanib olish, mashina tarjimasi, POS teglash, matnni intellektual tahlil qilish, yangi matnni yaratish va so‘zlarni bashorat qilish kabi NLP vazifalarini hal qilish uchun til modellaridan foydalaniladi. Til modellari har qanday NLP vazifasining muhim komponent hisoblanadi. Ushbu maqolada til modellarini ishlab chiqish bosqichlari va o‘zbek tili korpusi matnlari asosida til modellarini yaratishning n-gramm metodi keltiriladi.

Kalit so‘zlar: til modellari, NLP, til korpusi, UzbNlp paketi, unigram, bigram, trigram, n-gramm, so'zni bashorat qilish.

Kirish

Tabiiy tilni qayta ishlash (Natural Language Processing, NLP) dagi til modeli – oldingi so‘zlar asosida gapda berilgan so‘zlar ketma-ketligining hosil bo‘lish ehtimolini aniqlaydigan statistik model. Ushbu model asosida matn konteksti asosida mos so‘zni bashorat qilish imkonini beradi. Til modellaridan nutqni aniqlash, mashina tarjimasi, imloni tuzatish va boshqa NLP vazifalarida keng foydalaniladi. Til modelini ishlab chiqish odatda katta hajmdagi namunaviy gaplarini mashinali o‘qitish (ML) qadamidan boshlanadi. O‘qitish jarayonida model so‘zlar ketma-ketligi bo‘yicha ehtimollik taqsimotini hisoblaydi. Kontekstdagi so‘zni qo‘yilgan NLP vazifasiga muvofiq bashorat qilish uchun *unigram, bigram, trigram* yoki *n-gram* usullaridan foydalanish mumkin.

Til modellari

Til modellari tabiiy tilni qayta ishlashning asosini tashkil qiladi. Til modeli matn haqidagi ma’lumotni mashinalar tushunadigan miqdoriy ma’lumotga aylantiradi [1,2]. Bugungi kunda til modellari asosida moliya, sog‘liqni saqlash, harbiy va shu kabi turli sohalardagi NLP ilovalari ishlab chiqilmoqda [3]. Kundalik hayotda mobil telefonda bashoratli matn kiritish yoki Google

qidiruv tizimida til modellaridan foydalaniladi. Demak, til modellari har qanday tabiiy tilni qayta ishlash dasturining ajralmas qismini tashkil qiladi. Ushbu maqolada til korpusidagi strukturlanmagan matn asosida unigram, bigram va trigram til modellarini yaratish va ular yordamida keyingi so‘zlarni bashorat qilish usullari keltiriladi [1,2,4].

Til korpusidan matnni o‘qish

Til modelini ishlab chiqish uchun o‘zbek tili milliy korpusidagi 2024 yil mart oyida kun.uz va qalampir.uz onlayn nashrlaridagi post/maqolalarni yuklab olamiz. Strukturlanmagan matnlar yuklab olinganidan so‘ng, matn faylidagi belgilarning umumiy sonini aniqlash lozim.

```
file = open("UzbCorpus.txt", "r")
rawReadCorpus = file.read()
print ("O'qitiladigan ma'lumotlar to'plamidagi belgilar soni: {}".format(len(rawReadCorpus)))
```

Toshkent davlat o‘zbek tili va adabiyoti universiteti kompyuter lingvistikasi va raqamli texnologiyalar kafedrasi ilmiy tadqiqotchilari tomonidan ishlab chiqilgan UzbNlp paketida o‘zbek tilidagi matnlarni tahlil qiluvchi bir qator funksiya va metodlar mavjud [5]. Ushbu paketni import qilish lozim:

```
import UzbNlp
from UzbNlp.tokenize import uzb_word_tokenize, uzb_sent_tokenize
```

Strukturlanmagan matnni boshlang‘ich qayta ishlash

Birinchi navbatda, matnlardagi barcha ortiqcha qatorlarni va maxsus belgilarni olib tashlash kerak [6]. Buni quyidagi kod bilan amalga oshiramiz:

```
import string
string.punctuation = string.punctuation +'“'+'”'+'-'+'’'+'‘'+'—'
string.punctuation = string.punctuation.replace('.', '')
file = open('UzbCorpus.txt').read()
# yangi qatorlar va maxsus belgilarni olib tashlash uchun ma'lumotlarni boshlang`ich qayta ishlash
file_new = ""
for line in file:
   line_new = line.replace("n", " ")
   file_new += line_new
preprocessedCorpus = "".join([char for char in file_new if char not in string.punctuation])
```

Yangi qatorlar va maxsus belgilarni olib tashlaganimizdan so'ng, **UzbNlp.tokenize** dan **uzb_sent_tokenize** va **uzb_word_tokenize** metodlari yordamida so‘zlar va gaplarni ajratib olish uchun korpus matnlarini parchalash mumkin [7,8]. Quyidagi dastur kodida korpusdan olingan dastlabki 5 ta gaplani va birinchi 5 ta so‘z chop qilingan:

```
sentences = sent_tokenize(preprocessedCorpus)
print("Korpusdagi birinchi 5 ta gap: ")
print(sentences[0:5])
words = word_tokenize(preprocessedCorpus)
print("Qayta ishlangan korpusdagi birinchi 5 ta so`z/token: ")
print(words[0:5])
```

[Bilgil va ogoh bo‘lg‘ilki dunyoda ilm o‘qimoq va ilm istamoqdin g‘araz-u maqsud ikki dunyoning saodatiga yetmoqdur.]
[Lekin qancha mulla va qancha olim bo‘lsang ham dilingdan axloqi zamima (buzuq xulqlar) va yomon fe’llarni chiqarib ul dilingni axloqi hamida va axloqi pisandidalar (yaxshi va maqtovga sazovor xulqlar) bilan ziynat-u oro bermasang hanuz ikki dunyoni saodatiga yeta olmassen, nechuk?]
[Zeroki ko‘ngul bir podshoh va o‘zga a’zolar fuqaro manzilindadur.]
[Agar ko‘ngul durust bo‘lsa, boshqa a’zolar ham durust bo‘lur.]
[Va agar ko‘ngul buzuq bo‘lsa, o‘zga a’zolar ham buzuq bo‘lur.]

[Bilgil], [va], [ogoh], [bo‘lg‘ilki], [dunyoda]

Keyingi qadamda, korpusdagi **nomuhim so'zlar (stopwords)**ni olib tashlash kerak. Stopwords – bu gapga hech qanday maxsus ma’no yoki ahamiyatga ega bo‘lmagan **"va", "biroq", "ammo"** kabi keng qo`llaniladigan so‘zlar. O‘zbek tili uchun nomuhim so‘zlar ro‘yxatiga namuna quyida keltirilgan:

```
UzbNlp.download('uzb_stopwords')
from UzbNlp.corpus import stopwords
stop_words = set(stopwords.words('uzbek'))
filtered_tokens = [w for w in words if not w.lower() in stop_words]
```

Unigram, bigram va trigram til modellarini yaratish

O‘zbek tilidagi matnlarga mos n-gramm qiymatlarni UzbNlp paketidagi metodlar yordamida yaratish mumkin [5]. N-grammalar korpusda uchraydigan n ta ketma-ket so‘zdan iborat ketma-ketlikdir. Masalan, **"Men kitob o‘qishni yaxshi ko‘raman"** gapida **"men", "kitob", "o‘qishni", "yaxshi"** va **"ko‘raman"**, so‘zlari unigramlar, **"men kitob", "kitob o‘qishni", "o‘qishni yaxshi"** va **"yaxshi ko‘raman"** esa bigrammalardir. Quyidagi kod yordamida korpusdagi unigramlar, bigramlar va trigramlar aniqlanadi:

```
from collections import Counter
from UzbNlp.util import ngrams
unigrams=[]
bigrams=[]
trigrams=[]
for content in (sentences):
    content = content.lower()
    content = word_tokenize(content)
    for word in content:
        if (word =='.'):
            content.remove(word)
        else:
            unigrams.append(word)
    bigrams.extend(ngrams(content,2))
    trigrams.extend(ngrams(content,3))
```

```
print ("n-gramlar:n" + "-------------------------")
print ("--> UNIGRAMLAR: n" + str(unigrams[:5]) + " ...n")
print ("--> BIGRAMLAR: n" + str(bigrams[:5]) + " ...n")
print ("--> TRIGRAMLAR: n" + str(trigrams[:5]) + " ...n")
```

_____

UNIGRAMLAR:

[‘Bilgil’, ‘va’, ‘ogoh’, ‘bo‘lg‘ilki’, ‘dunyoda’, ‘ilm’, ‘o‘qimoq’, ‘va’, ‘ilm’, ‘istamoqdin’, ‘g‘araz-u’, ‘maqsud’, ‘ikki’, ‘dunyoning’, ‘saodatiga’, ‘yetmoqdur’]

BIGRAMLAR:

[(‘Bilgil’, ‘va’), (‘va’, ‘ogoh’), (‘ogoh’, ‘bo‘lg‘ilki’), (‘bo‘lg‘ilki’, ‘dunyoda’), (‘dunyoda’, ‘ilm’), (‘ilm’, ‘o‘qimoq’), (‘o‘qimoq’, ‘va’), (‘va’, ‘ilm’), (‘ilm’, ‘istamoqdin’), (‘istamoqdin’, ‘g‘araz-u’), (‘g‘araz-u’ , ‘maqsud’), (‘maqsud’, ‘ikki’), (‘ikki’, ‘dunyoning’), (‘dunyoning’, ‘saodatiga’), (‘saodatiga’, ‘yetmoqdur’)]

TRIGRAMLAR:

[(‘Bilgil’, ‘va’, ‘ogoh’), (‘va’, ‘ogoh’, ‘bo‘lg‘ilki’), … (‘dunyoning’, ‘saodatiga’, ‘yetmoqdur’)]

Keyingi qadamda, korpusdan **ko‘makchi, bog‘lovchi** yoki **olmosh** kabi nomuhim so‘zlarga ega bo‘lmagan unigramma, bigramma va trigrammalarni shakllantirish lozim. N-gramlardan nomuhim so'zlarni olib tashlash uchun quyidagi koddan foydalanamiz.

```
def stopwords_removal(n, a):
    b = []
    if n == 1:
        for word in a:
            count = 0
            if word in stop_words:
                count = 0
            else:
                count = 1
            if (count==1):
                b.append(word)
        return(b)
    else:
        for pair in a:
            count = 0
            for word in pair:
                if word in stop_words:
                    count = count or 0
                else:
                    count = count or 1
            if (count==1):
                b.append(pair)
        return(b)

unigrams_Processed = stopwords_removal(1,unigrams)
```

```
bigrams_Processed = stopwords_removal(2,bigrams)
trigrams_Processed = stopwords_removal(3,trigrams)
print ("Korpusni boshlang`ich qayta ishlangandan keyingi n-gramlar ro`yxati:")
print ("--> UNIGRAMLAR: n" + str(unigrams_Processed[:5]) + " ...n")
print ("--> BIGRAMLAR: n" + str(bigrams_Processed[:5]) + " ...n")
print ("--> TRIGRAMLAR: n" + str(trigrams_Processed[:5]) + " ...n")
```

_____

UNIGRAMLAR:

['Bilgil', 'ogoh', 'bo'lg'ilki', 'dunyoda', 'ilm', 'o'qimoq', 'ilm', 'istamoqdin', 'g'araz-u', 'maqsud', 'ikki', 'dunyoning', 'saodatiga', 'yetmoqdur']

BIGRAMLAR:

[('Bilgil', 'ogoh'), ('ogoh', 'bo'lg'ilki'), ('bo'lg'ilki', 'dunyoda'), ('dunyoda', 'ilm'), ('ilm', 'o'qimoq'), ('o'qimoq', 'ilm'), ('ilm', 'istamoqdin'), ('istamoqdin', 'g'araz-u'), ('g'araz-u', 'maqsud'), ('maqsud', 'ikki'), ('ikki', 'dunyoning'), ('dunyoning', 'saodatiga'), ('saodatiga', 'yetmoqdur')]

TRIGRAMLAR:

[('Bilgil', 'ogoh', 'bo'lg'ilki'), ... ('dunyoning', 'saodatiga', 'yetmoqdur')]

Yuqoridagi natijalar asosida korpusda hosil bo'ladigan har bir n-grammning sonini yoki chastotasini aniqlash mumkin. Ushbu natija n-grammlar asosida kontekstdagi keyingi mumkin bo'lgan so'zning ehtimolini hisoblash uchun kerak bo'ladi. Quyida keltirilgan **get_ngrams_freq_dist** funksiyasi orqali har bir n-gramga mos keladigan chastota aniqlanadi [9]. Ushbu funksiya orqali korpusdagi barcha unigramlar, bigramlar va trigramlarning chastotalari hisoblanadi.

```
def get_ngrams_freq_dist(n, ngramList):
    ngram_freq_dict = {}
    for ngram in ngramList:
        if ngram in ngram_freq_dict:
            ngram_freq_dict[ngram] += 1
        else:
            ngram_freq_dict[ngram] = 1
    return ngram_freq_dict
unigrams_freqDist = get_ngrams_freqDist(1, unigrams)
unigrams_Processed_freqDist = get_ngrams_freqDist(1, unigrams_Processed)
bigrams_freqDist = get_ngrams_freqDist(2, bigrams)
bigrams_Processed_freqDist = get_ngrams_freqDist(2, bigrams_Processed)
trigrams_freqDist = get_ngrams_freqDist(3, trigrams)
trigrams_Processed_freqDist = get_ngrams_freqDist(3, trigrams_Processed)
```

Bigram va trigram til modellari yordamida keyingi uchta so'zni bashorat qilish

Til modelida gapning ehtimolini hisoblash uchun zanjir qoidasi qo'llaniladi. $w_1 w_2 \ldots w_n$ gap, undagi w_1, w_2, w_n alohida so'zlar/tokenlar bo'lsin. Ushbu gapning hosil bo'lish ehtimoli quyidagi formula bilan aniqlanadi:

$$P(w_1, w_2 \dots w_n) = \prod_i P(w_i | w_1, w_2 \dots w_{i-1})$$

Masalan, **"Men kitob o`qishni yaxshi ko'raman"** gapining ehtimoli quyidagicha aniqlanadi:

$P("Men\ kitob\ o`qishni\ yaxshi\ ko'raman"=$
$P("men")P("kitob"|"men")P("o`qishni"|"men\ kitob")$
$P("yaxshi"|"men\ kitob\ o`qishni")$
$P("ko'raman"|"men\ kitob\ o`qishni\ yaxshi")$

Yuqoridagi formuladagi individual ehtimolliklar quyidagi usul orqali aniqlanadi:

$P(men) = Count('men') \ / \ umumiy\ so`zlar\ soni$
$P("kitob"|"men") = Count('men\ kitob') \ / \ Count('men')$
$P("o`qishni"|"men\ kitob") = Count('men\ kitob\ o`qishni') \ / \ Count('men\ kitob')$
$P("yaxshi"|"men\ kitob\ o`qishni")$
$= Count('men\ kitob\ o`qishni\ yaxshi') \ / \ Count('men\ kitob\ o`qishni')$
$P("ko'raman"|"men\ kitob\ o`qishni\ yaxshi")$
$= Count('men\ kitob\ o`qishni\ yaxshi\ ko'raman')$
$/ \ Count('men\ kitob\ o`qishni'\ yaxshi')$

Bunda,
$Count('men')$,$Count('men\ kitob')$,
$Count('men\ kitob\ o`qishni')$,
$Count('men\ kitob\ o`qishni\ yaxshi')$ va $Count('men\ kitob\ o`qishni\ yaxshi\ ko'raman')$ qiymatlar yuqoridagi get_ngrams_freq_dist(n, ngramList) funksiyasidan foydalangan holda hisoblagan tegishli unigram, bigram va trigramma chastotalaridir.

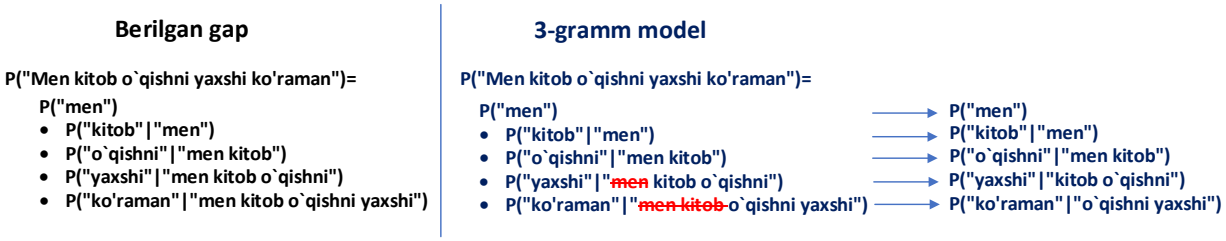
Yuqordagi formulalrdagi ehtimolliklarni hisoblash uchun bigram modelidan foydalanganda, har bir yangi so'zning ehtimolligi faqat oldingi so'zga bog'liqligini qayd etish mumkin. Ya'ni, oldingi misol uchun, gaplaning ehtimoli quyidagicha hisoblanadi:

$P("Men\ kitob\ o`qishni\ yaxshi\ ko'raman"=$
$P("men")P("kitob"|"men")P("o`qishni"|"kitob")$
$P("yaxshi"|"o`qishni")P("ko'raman"|"yaxshi")$

Xuddi shunday, trigramm modeli uchun ehtimollik quyidagicha hisoblanadi:

$P("Men\ kitob\ o`qishni\ yaxshi\ ko'raman"=$
$P("men")P("kitob"|"men")P("o`qishni"|"men\ kitob")$
$P("yaxshi"|"kitob\ o`qishni")$
$P("ko'raman"|"o`qishni\ yaxshi")$

Trigramm modellashtirishni quyidagi diagramma bilan yaxshiroq tushuntirish mumkin:



Biroq, bunday modellashtirishning o‘ziga xos jihati bor. O‘quv namunalarida (dataset) mavjud bo‘lmagan, ammo joriy matn/korpusda mavjud qandaydir bigramma bor. Yuqoridagi

hisob-kitoblar asosida ushbu bigrammaga **0** ehtimol belgilanadi va gapdagi umumiy ehtimollik qiymati **0** ga teng bo‘ladi. Ushbu muammoni bartaraf etish uchun **“silliqlash” (smoothing)** amalga oshiriladi. Ba‘zi ehtimollik qiymatlarini hisobga olimagan (ko‘rib chiqilmagan) hodisalarga qayta belgilash uchun parametrlar tartibga solinadi. Bunday yumshatish usullaridan biri **Add-one** yoki **Laplace smoothing** bo‘lib, yuqoridagi til modelini ishlab chiqishga joriy qilamiz ushbu maqolada foydalanamiz. Buning uchun barcha bigramm qiymatlariga **1** va barcha unigramma qiymatlarga **V** (korpusdagi unikal so‘zlar soni)ni qo‘shish lozim.

Avvalgi baho:

$$P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Add-one baho:

$$P_{Add-1}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Silliqlangan bigram va trigram modellari haqida ma’lumot hosil bo‘lganidan so‘ng, modelda mos o‘zgarishlarni amalga oshirish kerak. Bashorat qilish uchun qayta ishlanmagan bigramm va trigram elementlaridan (nomuhim so`zlarga ega matn) foydalanamiz.

```
smoothed_bigrams_probDist = {}
V = len(unigrams_freqDist)
for i in bigrams_freqDist:
   smoothed_bigrams_probDist[i] = (bigrams_freqDist[i] + 1)/(unigrams_freqDist[i[0]]+V)
smoothed_trigrams_probDist = {}
for i in trigrams_freqDist:
   smoothed_trigrams_probDist[i] = (trigrams_freqDist[i] + 1)/(bigrams_freqDist[i[0:2]]+V)
```

Keyingi qadamda, hisoblangan bigram va trigram tili modellari yordamida keyingi uchta so‘z bashorat qilinadi.

```
testSent1 = "Qolaversa, durustgina "
testSent2 = "Har bir odam uchun"
testSent3 = "Aytmat bobom qo‘li gul usta edi"
```

Berilgan namunaviy gaplarni tokenlarga ajratilib, ulardagi oxirgi unigramlar va bigramlar aniqlanadi.

```
token_1 = word_tokenize(testSent1)
token_2 = word_tokenize(testSent2)
token_3 = word_tokenize(testSent3)
ngram_1 = {1:[], 2:[]}
ngram_2 = {1:[], 2:[]}
ngram_3 = {1:[], 2:[]}
for i in range(2):
   ngram_1[i+1] = list(ngrams(token_1, i+1))[-1]
   ngram_2[i+1] = list(ngrams(token_2, i+1))[-1]
   ngram_3[i+1] = list(ngrams(token_3, i+1))[-1]
print("1-gap: ", ngram_1,"2-gap: ",ngram_2,"3-gap: ",ngram_3)
```

So‘ngra, yangilangan bigram modelidan foydalanib, berilgan namunaviy gaplarga mos keyingi 3 ta so‘zni bashorat qilish funksiyalarini shakllantiramiz.

```
def predict_next_word(last_word,probDist):
    next_word = {}
    for k in probDist:
        if k[0] == last_word[0]:
            next_word[k[1]] = probDist[k]
    k = Counter(next_word)
    high = k.most_common(1)
    return high[0]
def predict_next_3_words(token,probDist):
    pred1 = []
    pred2 = []
    next_word = {}
    for i in probDist:
        if i[0] == token:
            next_word[i[1]] = probDist[i]
    k = Counter(next_word)
    high = k.most_common(2)
    w1a = high[0]
    w1b = high[1]
    w2a = predict_next_word(w1a,probDist)
    w3a = predict_next_word(w2a,probDist)
    w2b = predict_next_word(w1b,probDist)
    w3b = predict_next_word(w2b,probDist)
    pred1.append(w1a)
    pred1.append(w2a)
    pred1.append(w3a)
    pred2.append(w1b)
    pred2.append(w2b)
    pred2.append(w3b)
    return pred1,pred2
print("Silliqlashtirilgan bigram modeli yordamida keyingi 3 ta mumkin bo‘lgan so‘zlar ketma-ketligini bashorat qilish:")
pred1,pred2 = predict_next_3_words(ngram_1[1][0],smoothed_bigrams_probDist)
print("1a)" +testSent1 +" "+ '33[1m' + pred1[0][0]+" "+pred1[1][0]+" "+pred1[2][0] + '33[0m')
print("1b)" +testSent1 +" "+ '33[1m' + pred2[0][0]+" "+pred2[1][0]+" "+pred2[2][0] + '33[0m')
pred1,pred2 = predict_next_3_words(ngram_2[1][0],smoothed_bigrams_probDist)
print("2a)" +testSent2 +" "+ '33[1m' + pred1[0][0]+" "+pred1[1][0]+" "+pred1[2][0] + '33[0m')
print("2b)" +testSent2 +" "+ '33[1m' + pred2[0][0]+" "+pred2[1][0]+" "+pred2[2][0] + '33[0m')
pred1,pred2 = predict_next_3_words(ngram_3[1][0],smoothed_bigrams_probDist)
print("3a)" +testSent3 +" "+ '33[1m' + pred1[0][0]+" "+pred1[1][0]+" "+pred1[2][0] + '33[0m')
print("3b)" +testSent3 +" "+ '33[1m' + pred2[0][0]+" "+pred2[1][0]+" "+pred2[2][0] + '33[0m')
```

Yangilangan bigram modelining bashoratlari:

1a) Qolaversa, durustgina baxshi ham edi
1b) Qolaversa, durustgina rassom ham edi
2a) Har bir odam uchun dunyo ostonadan boshlanadi
2b) Har bir odam uchun hayot juda go`zal
3a) Ahmad bobom qo‘li gul usta edi
3b) Ahmad bobom qo‘li juda chiroyli edi

Xuddi shunday uslubda yangilangan trigram modelidan bashorat olamiz.

```
def predict_next_word(last_word,probDist):
    next_word = {}
    for k in probDist:
        if k[0:2] == last_word:
            next_word[k[2]] = probDist[k]
    k = Counter(next_word)
    high = k.most_common(1)
    return high[0]
def predict_next_3_words(token,probDist):
    pred = []
    next_word = {}
    for i in probDist:
        if i[0:2] == token:
            next_word[i[2]] = probDist[i]
    k = Counter(next_word)
    high = k.most_common(2)
    w1a = high[0]
    tup = (token[1],w1a[0])
    w2a = predict_next_word(tup,probDist)
    tup = (w1a[0],w2a[0])
    w3a = predict_next_word(tup,probDist)
    pred.append(w1a)
    pred.append(w2a)
    pred.append(w3a)
    return pred
print("Silliqlashtirilgan trigram modeli yordamida keyingi 3 ta mumkin bo‘lgan so‘zlar ketma-ketligini bashorat qilish:")
pred = predict_next_3_words(ngram_1[2],smoothed_trigrams_probDist)
print("1)" +testSent1 +" "+ '33[1m' + pred[0][0]+" "+pred[1][0]+" "+pred[2][0] + '33[0m')
pred = predict_next_3_words(ngram_2[2],smoothed_trigrams_probDist)
print("2)" +testSent2 +" "+ '33[1m' + pred[0][0]+" "+pred[1][0]+" "+pred[2][0] + '33[0m')
pred = predict_next_3_words(ngram_3[2],smoothed_trigrams_probDist)
print("3)" +testSent3 +" "+ '33[1m' + pred[0][0]+" "+pred[1][0]+" "+pred[2][0] + '33[0m')
```

---

1) Qolaversa, durustgina omadli biznesmenga aylanishdi
2) Har bir odam uchun eng yaxshi kun

3) Ahmad bobom qo‘li yarador holda qaytdi

Xulosa.

Tilni modellashtirish (Language modeling, LM) kontekstda so‘zlarni bashorat qilish uchun korpus matnlarini tahlil qiladi va tabiiy til modelini shakllantiradi. Ushbu modellar gapda ma’lum bir so‘z ketma-ketligining yuzaga kelish ehtimolini aniqlash uchun statistik va ehtimollik usullaridan foydalanadi. NLP kontseptsiyalariga asoslangan ilovalar va dasturlar audiodan matnga o‘tkazish, hissiyotlarni tahlil qilish, nutqni aniqlash va imlo tuzatish kabi vazifalarda til modellaridan foydalaniladi. Til modellari matn ma’lumotlarining tahlil qilingan qismidagi so‘z ehtimolini aniqlash orqali ishlaydi. Ma’lumotlar ushbu tabiiy tilda kontekst qoidalarini qidiradigan mashinali o‘rganish algoritmiga berilgandan so‘ng qo‘llaniladi. Ushbu maqolada o‘zbek tili kospusi matnlarini n-gramm usuliga asoslangan til modeli ishlab chiqildi va namunaviy gaplarga qo‘llash usullari keltirildi.

Foydalanilgan adabiyotlar:

1. Jurafsky, D., & Martin, J. H. (2019). Chapter 3: N-Gram Language Models N-Gram Language Models. *Speech and Language Processing*.
2. Chen, M., Suresh, A. T., Mathews, R., Wong, A., Allauzen, C., Beaufays, F., & Riley, M. (2019). Federated learning of N-gram language models. *CoNLL 2019 - 23rd Conference on Computational Natural Language Learning, Proceedings of the Conference*. https://doi.org/10.18653/v1/k19-1012
3. Republic, C., & Mikolov, T. (2012). Statistical Language Models Based on Neural Networks. *Wall Street Journal*, *April*. https://doi.org/10.1016/j.csl.2015.07.001
4. P.~Brown, V.~Della Pietra, de Souza, P., J.~Lai, & R.~Mercer. (1992). Class-based n-gram models of natural language. *Computational Linguistics*, *18*.
5. Boltayevich, E. B., Mirdjonovna, H. S., & Ilxomovna, A. X. (2023). Methods for Creating a Morphological Analyzer. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *13741 LNCS*. https://doi.org/10.1007/978-3-031-27199-1_4
6. Bommasani, R., Liang, P., & Lee, T. (2023). Holistic Evaluation of Language Models. *Annals of the New York Academy of Sciences*, *1525*(1). https://doi.org/10.1111/nyas.15007
7. Konstantopoulos, S. (2010). Learning language identification models: A comparative analysis of the distinctive features of names and common words. *Proceedings of the 7th International Conference on Language Resources and Evaluation, LREC 2010*.
8. Wallace, E., Gardner, M., & Singh, S. (2020). *Interpreting Predictions of NLP Models*. https://doi.org/10.18653/v1/2020.emnlp-tutorials.3
9. Roh, J., Park, S., Kim, B. K., Oh, S. H., & Lee, S. Y. (2021). Unsupervised multi-sense language models for natural language processing tasks. *Neural Networks*, *142*. https://doi.org/10.1016/j.neunet.2021.05.023