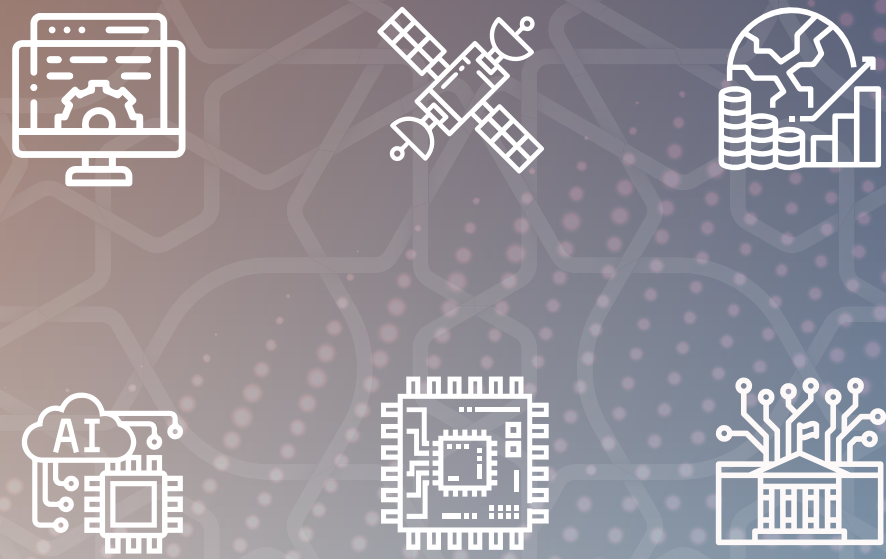
MUHAMMAD AL-XORAZMIY AVLODLARI
ILMIY-AMALIY VA AXBOROT-TAHLILIY JURNAL

DESCENDANTS OF MUHAMMAD AL-KHWARIZMI
SCIENTIFIC-PRACTICAL AND INFORMATION-ANALYTICAL JOURNAL

2(24)/2023

ISSN-2181-9211

MUHAMMAD AL-XORAZMIY NOMIDAGI TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI



MUHAMMAD AL-XORAZMIY AVLODLARI

Ilmiy-amaliy va axborot-tahliliy jurnal 2017 yilda ta’sis etilgan

2(24)/2023

MUNDARIJA

Elov B., Xusainova Z., Yodgorov U.

O‘zbek tili matnlari uchun tokenizayorni ishlab chiqish

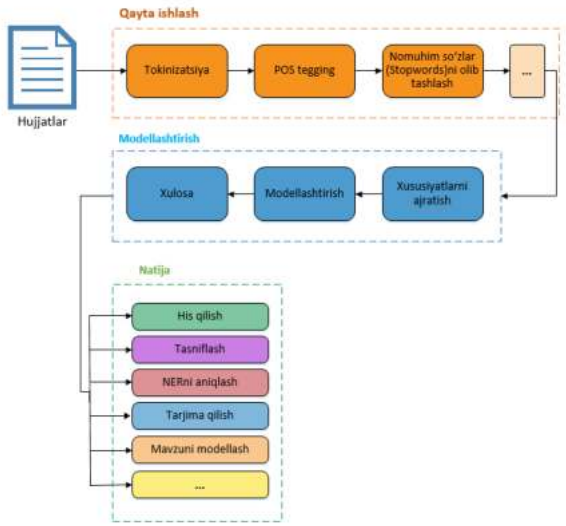
Tokenizatsiya dastlabki ishlov berish bosqichida strukturlanmagan tabiiy til matni yaxshiroq strukturlangan ko`rinishga (kompyuter nuqtai nazaridan) aylantirishni anglatadi. Tokenizatsiya jarayonida matn tokenlar deb ataladigan foydali ma'lumotlarni o'z ichiga olgan qismlarga ajratiladi. Ushbu maqolada maqsadi tokenizatsiya jarayoni tavsifi, uning qanday ishlashini va tokenizatorni ishlab chiqish usullari keltiriladi.

Kalit so‘zlar: Tokenizatsiya, UzbSentencizer, document, sentence, token, ajratuvchilar, regular ifodalar, morfologik analizator.

Kirish.

Tabiiy tilni qayta ishlash (NLP) – mashinali o'rganish algoritmlariga inson tilini tartibga solish va tushunish imkonini beradi. NLP mashinalarga nafaqat matn va nutqni to'plash, balki javob berishi kerak bo'lgan asosiy ma'noni aniqlash imkonini beradi. Inson tili murakkab va doimo rivojlanib boradi. Shu sababali tabiiy tilni qayta ishlash juda qiyin. Tokenizatsiya jarayoni NLP vazifasini hal qilish bosqichlarining dastlabkilaridan biridir (1-rasm). Tabiiy tilni qayta ishlash inson tillarini kompyuterlar tili bilan bog'lash uchun ham tilshunoslik, ham matematika, ham dasturlash texnologiyalaridan foydalanadi. NLP algoritmlari orqali tabiiy aloqa shakllari (yozma/og`zaki) mashina tomonidan tushunilishi mumkin bo'lgan ma'lumotlarga bo'linadi [1].

Tabiiy til bilan ishlashda, ayniqsa, nutqini algoritmlarga moslashtirishga odatlanmagan odamlarda juda ko'p qiyinchiliklarni yuzaga keltiradi. Nutq va yozma matn uchun biz dasturlar yaratishimiz mumkin bo'lgan qoidalar mavjud bo'lsa-da, odamlar har doim ham bu qoidalarga rioya qilmaydi [2].



1-rasm. Matnni qayta ishlashing dastlabki bosqichlari

Tokenizatsiya oddiy jarayon bo'lib, qayta ishlanmagan ma'lumotlarni oladi va uni foydali ma'lumotlarga aylantiradi. Tokenizatsiya kiberxavfsizlikda va NFTlarni yaratishda qo'llanilishi bilan mashhur bo'lsa-da, tokenizatsiya NLP jarayonining muhim qismidir. Tokenizatsiya tabiiy tilni qayta ishlashda paragraflar va gaplar ma'nosini osonroq belgilash mumkin bo'lgan kichikroq birliklarga bo'lish uchun ishlatiladi. NLP jarayonining birinchi bosqichi ma'lumotlarni yig'ish (gap) va uni tushunarli qismlarga (so'zlarga) ajratishdir. Gapni mashina tushunib olishi uchun satrda uni alohida qismlarga ajratish uchun tokenizatsiya amalga oshiriladi. Tokenizatsiya jarayoni matnni morfologik tahlil qilish NLP vazifasining muhim bosqichi hisoblanadi [3,4].

Tokenizatsiya jarayoni oddiydek tuyulishi mumkin. Lekin gapni uning qismlariga ajratish mashinaga qismlarni ham, butunni ham tushunish imkonini beradi. Bu dasturga har bir so'zni o'z-o'zidan tushunishga yordam beradi, shuningdek, ularning katta hajmli matnda qanday ishlashini tushunishga yordam beradi. Tokenizatsiya jarayoni ayniqsa katta hajmdagi matnlar uchun juda muhim bo`lib, u mashinaga ma'lum so'zlarning chastotalarini hamda ularning turli statistikalarini hisoblash imkonini beradi [5]. Tokenizatsiya jarayoni tabiiy tilni qayta ishlashning keyingi bosqichlari uchun muhimdir.

Garchi gaplarni ajratish oddiy bo'lib tuyulsa-da, biz har doim so'zlardan gaplar tuzamiz, bu mashinalar uchun biroz murakkabroq bo'lishi mumkin. Agar bo'shliqlar yoki tinish belgilari so'z chegaralarini aniqlamasa, so'zlarni segmentlarga ajratish katta qiyinchilikdir. Bu, ayniqsa, *xitoy, yapon, koreys* va *tay* tillari kabi belgilarga asoslangan tillarda o`z ifodasini topgan.

Yana bir qiyinchilik – bu so'zning ma'nosini sezilarli darajada o'zgartiradigan belgilar. Masalan, son davomida kelgan "$" belgisi (100$) sonning o'zidan (100) farq qiladi (…ro`yxat ^{0}C,…).

"va h.z." va "mas.", "**Т.**", "б.", "ш.к.", " фл. рвдш.", " q.", … kabi qisqarmalarni ham tegishli tartibda mos tokenlarga ajratish kerak. Gapning har bir qismini to'g'ri belgilamaslik NLP jarayonida keyinchalik tushunmovchiliklarga olib kelishi mumkin.

2-rasm. Tokenizatsiya jarayoni ierarsixasi

Bugungi kunda matndagi tokenizatsiya jarayonini amalga oshirishning bir nechta turli usullar mavjud va

bu usullar NLP jarayonining keyingi bosqichlarini tubdan o'zgartiradi [6,7]:

Ushbu maqolada o`zbek tilidagi matnlariga *Word Tokenization* usulini qo`llash namoyish qilinadi.

Tokenizatsiya orqali tokenlar orasidagi chegaralar belgilanadi. Eng oddiy ajratuvchi sifatida bo'shliq orqali ajratishdan iborat. Ammo bu har doim ham kerakli natijani bermaydi. Masalan, yapon va xitoy tillarida *boshqa turdagi bo'shliqdan* foydalanadilar. Shuningdek, ba`zan bo'shliqlardan tashqari maxsus belgilar orqali so'zni ikkita tokenda ajratish mumkin bo'lgan holatlar ham bo'lishi mumkin. Bigrammalar va idiomalar (masalan, "kundan kunga" yoki "ega bo`lgan") kabi maxsus birikmalarni alohida tokenlarga bo'lish yoki bo'lmaslik to'g'risida **qaror qabul qilish holatlari** mavjud.

Odatda tokenizatsiya jarayoni bilan birgalikda amalga oshiriladigan yana bir qadam **"Sentencizing"** (matnni gaplarga ajratish) hisoblanadi. Bu jarayon hujjatda bir nechta gaplar mavjud bo'lgan hol uchun juda muhimdir. Buning muhimligining asosiy sababi *matnni sintaktik tahlil qilish* bilan bog'liq. *Nutq qismlarini va bog'liqlik munosabatlarni* aniqlash uchun gap (butun matn emas) strukturasini hosil qilish lozim.

NLP ilovalarini ishlab chiqishda, ba'zi vositalar tokenizatsiya va sintaktik tahlildan so'ng matndagi gaplar aniqlanadi. Shunday qilib, agar ikkita tinish belgisi orasidagi so'zlar bir nechta gaplarni o'z ichiga olsa ham, ular ushlanib qoladi.

Boshlag`ich yondashuv

Birinchi navbatda Python standartlari va ba'zi yaxshi amaliyotlardan foydalangan holda **UzbSentencizer**ni yaratamiz. *tokenization.py* faylida **UzbSentencizer** sinfini yaratamiz. Barcha qayta ishlash vazifalarini konstruktorning o'zida jamlaymiz, ya'ni sinfni yaratish bizga gaplar ro'yxatini beradi. Konstruktor parametrlari sifatida matnni, bo‘lish uchun ishlatiladigan belgilarni va ajratuvchi tokenni qabul qilamiz.

```
class UzbSentencizer:
 def __init__(self, input_text,
split_characters=['.','?','!',':'],
delimiter_token='<SPLIT>'):
 self.sentences = []
 self.raw = str(input_text)
 self._split_characters=split_characters
 self._delimiter_token=delimiter_token
 self._index=0
 self._sentencize()
```

UzbSentencizer orqali matndagi har bir *nuqta, undov belgisi, so`roq belgisi* yoki *ikki nuqtaga* mos yangi gap yaratiladi. Foydalanuvchi belgilarning aniq ro'yxatini taqdim etish orqali buni o'zgartirishi mumkin, ammo bizda standart mavjud.

Keyingi qadamda, matnni gaplarga ajratish uchun maxsus *"teg" (delimiter_token)* ishlab chiqilgan. Ushbu teg tinish belgilari ro`yxatiga qo'shiladi. Buning uchun matnda uchramaydigan maxsus "teg"ni yaratamiz. Bu foydalanuvchi tomonidan ham almashtirilishi ham mumkin.

Tokenizatorni ishlab chiqisjda yana muhim tafsilotlar mavjud: foydalanuvchi foydalanishi mumkin bo'lgan **atributlar** va **qayta ishlanmahan qism** (agar foydalanuvchi kerak bo'lsa, asl matnni saqlaydi). Shuningdek yopiq **_index** atributi mavjud bo'lib, u ob'ektni takrorlanuvchiga aylantirish uchun ishlatiladi. Nihoyat, **_sentencize()** funksiyasini chaqirish orqali, dasturni ishga tushirish mumkin.

```
def _sentencize(self):
 work_sentence = self.raw
 for character in self._split_characters:
 work_sentence = work_sentence.replace(character,
character+""+self._delimiter_token)
 self.sentences = [x.strip() for x in
work_sentence.split(self._delimiter_token) if x !='']
```

Dasturning ishlash tamoyili: *dastlabki ma'lumotlarning nusxasi yaratiladi*, *barcha tinish belgilar aniqlanadi* va ularga maxsus ajratilgan tokenni qo'shiladi. Nihoyat, hosil bo'lgan satr maxsus ajratilgan token va bo'shliqlar asosida ajratiladi (x.strip() for x in). Shuningdek, matn ortiqcha bo'linishlardan tozalanishi lozim. Yakunda sinfni takrorlanadigan qilish uchun bir nechta sehrli usullarni bekor qilamiz:

```
def __iter__(self):
 return self

 def __next__(self):
 if self._index < len(self.sentences):
 result = self.sentences[self._index]
 self._index+=1
 return result
 raise StopIteration
```

UzbSentencizer matnni gaplarga ajratuvchi tokenizatorimiz tayyor! Endilikda berilgan martnni gaplarga ajratishimiz mumkin va keying qadamga (UzbTokenizerga) o'tishimiz mumkin.

```
class UzbTokenizer:
 def __init__(self, sentence, token_boundaries=[' ',
'-'],
 punctuations=string.punctuation,
delimiter_token='<SPLIT>'):
 self.tokens = []
 self.raw = str(sentence)
 self._token_boundaries = token_boundaries
 self._delimiter_token = delimiter_token
 self._punctuations = punctuations
 self._index = 0
 self._tokenize()
```

Bizga **matn** (*gap*), **ajratuvchilar** (*token_boundaries, bo'sh joy va defis*) va **maxsus "teg"** (delimiter_token) mavjud. Tinish belgilarini *inobatga olish*, *ularni token sifatida hisoblash* va *ulardan o`zaro bog'langan so'zlardan ajratish* uchun foydalanish mumkin.

```
def _tokenize(self):
 work_sentence = self.raw
 for punctuation in self._punctuations:
```

```
        work_sentence =
work_sentence.replace(punctuation, " "+punctuation+"
")
        for delimiter in self._token_boundaries:
        work_sentence = work_sentence.replace(delimiter,
self._delimiter_token)
        self.tokens = [x.strip() for x in
work_sentence.split(self._delimiter_token) if x != '']
```

Ushbu sinf va undagi metodlar **UzbSentencizer** sinfiga juda o'xshash. Bu ikki sinfning asosiy farqi shundaki, maxsus ajratilgan teglarni qo'shishdan oldin tinish belgilarini bog'langan satrlardan ajratamiz:

```
       def __iter__(self):
        return self

        def __next__(self):
        if self._index < len(self.tokens):
        result = self.tokens[self._index]
        self._index+=1
        return result
        raise StopIteration
```

Biz ishlab chiqqan tokenizator barcha hollarni qamrab olmaydi. Misol uchun, agar biz o'nlik sonni kiritsak (aytaylik, **19,3**), u [**'19'**, **'.'**, **'3'**] ga ajratadi. Buning uchun UzbTokenzatorga bir qator o`zgarishlarni amalga oshirish lozim.

Yakuniy kod quyidagi ko`rinishga ega:

```
       import sys, re
       PUNCTUATIONS = ['(?<=[0-9]|[^0-9.])(\.)(?=[^0-
9.]|[^0-9.]|[\s]|$)','\.{2,}','\!+','\:+','\?+','\,+',
r'\(|\)|\[|\]|\{|\}|\<|\>']

       class Token:
        def __init__(self, start_position, end_position,
raw_sentence_reference, SOS = False, EOS = False):

        self.start_pos = int(start_position)
        self.end_pos = int(end_position)
        self._sentence_string = raw_sentence_reference
        self.next_token = None
        self.previous_token = None
        self.SOS = SOS
        self.EOS = EOS

        def get(self):
        if self.SOS:
        return '<B>'
        elif self.EOS:
        return '<E>'
        else:
        return
self._sentence_string[self.start_pos:self.end_pos]

        def __repr__(self):
        return self.get()

        def __str__(self):
        return self.get()

        def __eq__(self, other):
        return self.get() == other

       def UzbTokenize(raw_input_sentence,
join_split_text = True, split_text_char = '\-',
punctuation_patterns= PUNCTUATIONS,
split_characters = r'\s|\t|\n|\r',
delimiter_token='<SPLIT>'):
        working_sentence = raw_input_sentence
        # Birinchi navbatda mumkin bo'lgan so'z
bo'linishlari:
        if join_split_text:
        working_sentence = re.sub('[a-
z]+('+split_text_char+'[\n])[a-z]+',", working_sentence)
        # Tinish belgilarini qayta ishlash
        for punct in punctuation_patterns:
        working_sentence = re.sub(punct, " \g<0> ",
working_sentence)
        # Tokenlarga ajratish
        working_sentence = re.sub(split_characters,
delimiter_token, working_sentence)
        list_of_token_strings = [x.strip() for x in
working_sentence.split(delimiter_token) if x.strip()
!='']
        previous = Token(0,0,raw_input_sentence,
SOS=True)
        list_of_tokens = [previous]
        for token in list_of_token_strings:
        start_pos = raw_input_sentence.find(token)
        end_pos = start_pos+len(token)
        new_token =
Token(start_pos,end_pos,raw_input_sentence)
        list_of_tokens.append(new_token)
        previous.next_token=new_token
        new_token.previous_token=previous
        previous=new_token
        if previous.SOS != True:
        eos = Token(len(raw_input_sentence),
len(raw_input_sentence), raw_input_sentence,
EOS=True)
        previous.next_token=eos
        eos.previous_token = previous
        list_of_tokens.append(eos)
        return list_of_tokens

       def Test_Uzbtokenize():
        sentence = '«Rossiya bosqinchilik urushiga misli
ko‘rilmagan muvofiqlashtirilgan sanksiyalar javobiga
sodiq qolamiz. Rossiyaga hamda cheklovchi
choralardan qochgan va ularga putur yetkazadiganlarga
iqtisodiy bosimni saqlab qolamiz va kuchaytiramiz». Bu
haqda G7 yetakchilari va Ukraina prezidenti Volodimir
Zelenskiyning bo‘lib o‘tgan muzokaralar yakunlari
bo‘yicha qo‘shma bayonotida aytiladi.'
        tokens = UzbTokenize(sentence)
        print(tokens)

       Test_Uzbtokenize()
```

_____

[<B>, «Rossiya, bosqinchilik, urushiga, misli, ko‘rilmagan, muvofiqlashtirilgan, sanksiyalar, javobiga, sodiq, qolamiz, ., Rossiyaga, hamda, cheklovchi, choralardan, qochgan, va, ularga, putur, yetkazadiganlarga, iqtisodiy, bosimni, saqlab, qolamiz, va, kuchaytiramiz», ., Bu, haqda, G7, yetakchilari, va, Ukraina, prezidenti, Volodimir, Zelenskiyning, bo‘lib, o‘tgan, muzokaralar, yakunlari, bo‘yicha, qo‘shma, bayonotida, aytiladi, ., <E>]

Yuqoridagi natijalardan ko`rinib turibdiki, matnni tokenlashda *“‘” belgili o`zbek tilidagi so`zlar, o`nli formatdagi sonlar (misol: 12,3)* va shunga o`xshash lug`atda mavjud bo`lmagan tokenlarni aniqlashda quyidagi regular ifodadan foydalanildi:

```
PUNCTUATIONS = ['(?<=[0-9]|[^0-9.])(\.)(?=[^0-9.]|[^0-9.]|[\s]|$)','\.{2,}','\!+','\:+','\?+','\,+',
r'\(|\)|\[|\]|\{|\}|\<|\>']
```

NLP vazifaga mos tokenizatorni ishlab chiqish uchun quyida keltirilgan usullardan foydalanish lozim:

Zamonaviy yondashuv

Yuqorida keltirilgan usullarni inobatga olgan hoda yanada ishonchli yondashuvni keltirlamiz. Bu yondashuv asosida keyingi qadamlar uchun qayta ishlatilishi mumkin bo'lgan **bardoshli strukturaga** ega elementlarni hosil qilish nazarda tutiladi. Ushbu yondashuvda biz uchta sinf yaratiladi: **Document, Sentence** va **Token**. Shuningdek, ikkita yangi statik funktsiyani yaratish kerak: **UzTokenizator** va **UzSentenciz**. Ushbu funksiyalar avvalgilaridan qiymatlarni qaytarish usuli bilan farq qiladi: satrlar ro‘yxati o‘rniga tokenlar yoki gaplar qaytariladi.

Matnni tokenlarga ajratish jarayoni foydalanuvchining **Document** sinfini yaratish va unga berilgan satrni uzatish orqali boshlanadi. Bu jarayon quyidagu ketma-ketlikda amalga oshiriladi: *hujjatni sozlash*, *hujjatni gaplarga ajratish*; *gapni tokenlarga ajratish*. Hujjatni yaratishda foydalanuvchi aslida to`liq matnni tokenizatsiya qilishni quyidagi operator baharadi:

doc = nlp(*“String”*))

Ushbu qismda dasturdagi ishlab chiqilgan strukturalar va asosiy o'zgaruvchilar keltiriladi:

```
# Hujjat sinfi
class Document:
 def __init__(self, document_text):
 self.raw = document_text
 self.sentences = sentencize(self.raw)
 self._index = 0

# Gap sinfi
class Sentence:
 def __init__(self, start_position, end_position, raw_document_reference):
 self.start_pos = int(start_position)
 self.end_pos = int(end_position)
 self._document_string = raw_document_reference
 self.next_sentence = None
 self.previous_sentence = None
 self.tokens = tokenize(self._document_string[self.start_pos:self.end_pos])
 self._index = 0

# Token sinfi
class Token:
 def __init__(self, start_position, end_position, raw_sentence_reference, SOS = False, EOS = False):
 self.start_pos = int(start_position)
 self.end_pos = int(end_position)
 self._sentence_string = raw_sentence_reference
 self.next_token = None
 self.previous_token = None
 self.SOS = SOS
 self.EOS = EOS
```

Usbu yondashuvdagi asosiy tafsilotlar shundan iboratki, gaplar berilgan hujjatning (**Document**) boshlang'ich/oxirgi pozitsiyasi bo'yicha aniqlanadi. Xuddi shu jarayon **Token**lar bilan ham sodir bo'ladi.

Document: Bizning hujjatimiz oddiygina qayta ishlanmagan satrni kiritish orqali yaratiladi. Keyin esa **sentencize** chaqiriladi.

Sentence: Sentence hujjatda joylashgan joyning "koordinatalari", qayerda qidirish kerakligi haqidagi dastlabki hujjatlar satri va ba'zi bo'sh jumlalarga havolalar bilan yaratilgan. Ular keyinchalik gaplar orasida harakat qilish uchun ishlatilishi mumkin. Keyin, u **tokenize**ni chaqiradi.

Token: Tokenni **SOS** (gap boshlanishi uchun) yoki **EOS** (gap oxiri uchun) sifatida yaratilib, keyinchalik, **lemma** yoki **POS** kabi boshqa atributlarni ham yaratishga xizmat qiladi.

Token sinfiga misol:

```
def get(self):
 if self.SOS:
 return '<SOS>'
 elif self.EOS:
 return '<EOS>'
 else:
 return self._sentence_string[self.start_pos:self.end_pos]

# Agar o'zgaruvchi chaqirilsa, terminalda Token qiymatini ko'rsatish
def __repr__(self):
 return self.get()

# Token qiymatini stdout uchun ko'rsatish
def __str__(self):
 return self.get()

# Bir tokenni boshqasi bilan oddiy satr asosida taqqoslash
def __eq__(self, other):
 return self.get() == other
```

Keyingi qadamda dastur uchun zarur strukturalarni ishlab chiqamiz: **qaror qabul qiluvchi** va **tokenizatsiya**

funktsiyalari. Ushbu amallarni bajarish uchun **regulyar ifodalar**dan foydalanishga to`g`ri keladi.

Regulyar ifodalar (Regex) – belgilar satridagi aniq ifodalarni moslashtirish usuli. Regex mexanizmi orqali satrda *qidirish* va *shablonni aniqlash* uchun maxsus belgilar to'plamidan foydalanadi. Regulyar ifodalar NLP vazifalarini hal qilishda juda foydali bo`lib, ular chuqur tahlil qilmasdan ko'p muammolarni hal qilishga yordam beradi. Regulyar ifodalardan foydalanib hujjatlardagi ma'lumotlarni topish va qayta ishlash mumkin. Boshlang`ich yondashuvdan biznig zamonaviy yondashuvimiz asosan regulyar ifodalardan foydalanib tokenlarga ajratadi. Joriy yondashuv asosida ikkita shablonlar seriyasini yaratildi:

- *gap chegaralarini aniqlash uchun;*
- *tinish belgilaridan "qochish" uchun.*

Bu ikki turdagi shablonlar paket o'zgaruvchisi sifatida taqdim etiladi va tokenizer va sentencizerga default iqymat bo'yicha o'tkaziladi:

```
DEFAULT_SENTENCE_BOUNDARIES =
['(?<=[0-9][^0-9.])(\.)(?=[^0-9.][^0-
9.][\s]|$)','\.{2,}','\!+','\:+','\?+']
"""
(?<=[0-9][^0-9.])(\.)(?=[^0-9.][^0-9.][\s]|$) ->
```

Raqam yoki boshqa davrdan oldin yoki keyin bo'lmagan davrni izlaydi. Bu gaplardagi o'nlik sonlar yoki default bo'yicha ajratish algoritmidan qochadi.

```
\.{2,} ->
\!+ ->
\:+ ->
\?+ ->
"""

DEFAULT_PUNCTUATIONS = ['(?<=[0-9][^0-
9.])(\.)(?=[^0-9.][^0-9.][\s]|$)','\.{2,}','\!+','\:+','\?+','\,+',
r'\(|\)|\[|\]|\{|\}|\<|\>']
"""
\,+ ->
\(|\)|\[|\]|\{|\}|\<|\> ->
"""
```

Keyingi qadamda yuqorida keltirilgan vositalar asosida ularni birgalikda qo`llashni ko'rib chiqamiz:

```
import re
def sentencize(raw_input_document,
sentence_boundaries =
DEFAULT_SENTENCE_BOUNDARIES,
delimiter_token='<SPLIT>'):
    working_document = raw_input_document
    punctuation_patterns = sentence_boundaries
    for punct in punctuation_patterns:
    working_document = re.sub(punct,
'\g<0>'+delimiter_token, working_document,
flags=re.UNICODE)
    list_of_string_sentences = [x.strip() for x in
working_document.split(delimiter_token) if x.strip() !=
""]
    list_of_sentences = []
    previous = None
    for sent in list_of_string_sentences:
    start_pos = raw_input_document.find(sent)
    end_pos = start_pos+len(sent)
    new_sentence = Sentence(start_pos, end_pos,
raw_input_document)
    list_of_sentences.append(new_sentence)
    if previous == None:
    previous = new_sentence
    else:
    previous.next_sentence = new_sentence
    new_sentence.previous_sentence = previous
    previous = new_sentence
    return list_of_sentences
```

Foydalanuvchi **stencize()** funksiyasiga faqat *raw_input_document* satrini uzatadi. Qolgan amallar bizning parametrlarimiz va regexlar ro'yxati bilan to'ldiriladi. Birinchi muhim qadam – gaplar chegaralari uchun *barcha regexlarimizni takrorlash* va *ularga ajratuvchi tokenni qo'shish*. Shuning uchun biz hujjatni gaplarga ajratganimizda tinish belgilarini saqlab qolishimiz mumkin.

Bu import qilinishi kerak bo'lgan **re** (*RegularExpression uchun*) maxsus python moduli yordamida amalga oshiriladi. Tokenizatsiya funksiyamizni quyidagicha shakllantiramiz:

```
def tokenize(raw_input_sentence, join_split_text =
True, split_text_char = '\-', punctuation_patterns=
DEFAULT_PUNCTUATIONS, split_characters =
r'\s|\t|\n|\r', delimiter_token='<SPLIT>'):
    working_sentence = raw_input_sentence
    # Mumkin bo'lgan so'z bo'linishlari bilan birinchi
navbatda:
    if join_split_text:
    working_sentence = re.sub('[a-
z]+('+split_text_char+'[\n])[a-z]+',", working_sentence)
    # Tinish belgilaridan qochish
    for punct in punctuation_patterns:
    working_sentence = re.sub(punct, " \g<0> ",
working_sentence)
    # Har qanday split_belgiga ajratiladi
    working_sentence = re.sub(split_characters,
delimiter_token, working_sentence)
    list_of_token_strings = [x.strip() for x in
working_sentence.split(delimiter_token) if x.strip()
!=""]
    previous = Token(0,0,raw_input_sentence,
SOS=True)
    list_of_tokens = [previous]
    for token in list_of_token_strings:
    start_pos = raw_input_sentence.find(token)
    end_pos = start_pos+len(token)
    new_token =
Token(start_pos,end_pos,raw_input_sentence)
    list_of_tokens.append(new_token)
    previous.next_token=new_token
    new_token.previous_token=previous
    previous=new_token
    if previous.SOS != True:
```

```
        eos = Token(len(raw_input_sentence),
len(raw_input_sentence), raw_input_sentence,
EOS=True)
        previous.next_token=eos
        eos.previous_token = previous
        list_of_tokens.append(eos)
        return list_of_tokens
```

Bizda o`zbek tili korpusidan olingan **"data.txt"** fayli mavjudligini hisobga olsak, biz oddiy skriptni sinab ko'rishimiz mumkin:

```
import pprint
declaration = []
with open('data.txt','r') as file:
 declaration+=file.readlines()

pp = pprint.PrettyPrinter(indent=2, compact=True)

for line in declaration:
 res = {'Document': line, 'Sentences':[]}
 document = Document(line)
 for sentence in document:
 res['Sentences'].append({'Sentence':sentence,
'Tokens':sentence.tokens})
 pp.pprint(res)
```

_____

```
{ 'Document': 'B«Rossiya bosqinchilik urushiga
misli ko`rilmagan '
 'muvofiqlashtirilgan sanksiyalar javobiga sodiq
qolamiz. '
 'Rossiyaga hamda cheklovchi choralardan qochgan
va ularga putur '
 'yetkazadiganlarga iqtisodiy bosimni saqlab
qolamiz va '
 'kuchaytiramizB». \n',
 'Sentences': [ { 'Sentence': B«Rossiya bosqinchilik
urushiga misli ko`rilmagan muvofiqlashtirilgan
sanksiyalar javobiga sodiq qolamiz.,
 'Tokens': [ <SOS>, B«Rossiya, bosqinchilik,
urushiga, misli,
 ko`rilmagan, muvofiqlashtirilgan, sanksiyalar,
 javobiga, sodiq, qolamiz, ., <EOS>]},
 {'Sentence': Rossiyaga hamda cheklovchi
choralardan qochgan va ularga putur yetkazadiganlarga
iqtisodiy bosimni saqlab qolamiz va kuchaytiramizB».,
 'Tokens': [ <SOS>, Rossiyaga, hamda, cheklovchi,
choralardan,
 qochgan, va, ularga, putur, yetkazadiganlarga,
 iqtisodiy, bosimni, saqlab, qolamiz, va,
 kuchaytiramizB», ., <EOS>]}]}
{ 'Document': 'Bu haqda G7 yetakchilari va Ukraina
prezidenti Volodimir '
 'Zelenskiyning bo`lib o`tgan muzokaralar yakunlari
bo`yicha '
 'qo`shma bayonotida aytiladi.',
 'Sentences': [ { 'Sentence': Bu haqda G7
yetakchilari va Ukraina prezidenti Volodimir
Zelenskiyning bo`lib o`tgan muzokaralar yakunlari
bo`yicha qo`shma bayonotida aytiladi.,
 'Tokens': [ <SOS>, Bu, haqda, G7, yetakchilari, va,
Ukraina,
 prezidenti, Volodimir, Zelenskiyning, bo`lib,
 o`tgan, muzokaralar, yakunlari, bo`yicha,
 qo`shma, bayonotida, aytiladi, ., <EOS>]}]}
```

Yuqorida keltilgan fikr-mulohazalar asosida ishlab chiqilgan algortimlar o`zbek tili morfologik analizatoriga qo`llandi va dasturiy ta`minot ishlab chiqildi. Quyida ushbu dasturiy ta`minotda lavhalar keltirilgan:

3-rasm. O`zbek tili morfologik analizatoridagi UzTokenizator

Xulosa.

Tokenizatsiya – bu matnli ma'lumotlar bilan ishlashning zamonaviy usulidir. Ushbu maqolada o`zbek tili korpusi matnlari uchun tokenizatsiya jarayonini ishlab chiqish bosqichlari turli yondashuvlar asosida keltirildi va Python yordamida tokenizatsiyani amalga oshirdik. Maqolada keltirilgan usullar algortimlar o`zbek tili morfoligik analizatori (http://uznatcorpara.uz/)ga qo`llandi.

Foydalanilgan adabiyotlar ro'yxati:

1. Rai, A., & Borah, S. (2021). Study of various methods for tokenization. In Lecture Notes in Networks and Systems (Vol. 137). https://doi.org/10.1007/978-981-15-6198-6_18

2. ARAVIND PAI. (2020). What is Tokenization in NLP? Here's All You Need To Know.

3. B.Elov, Sh.Hamroyeva, X.Axmedova. (2022). Methods for creating a morphological analyzer, 14th International Conference on Intellegent Human Computer Interaction, IHCI 2022, 19-23 October 2022, Tashkent.

4. B.Elov, Sh.Hamroyeva, D.Elova. (2022). Morfologik analizatorni yaratish usullari, O`zbekiston: til va madaniyat (Amaliy filologiya), 2022, 5(1).

5. Rai, A., & Borah, S. (2021). Study of various methods for tokenization. In Lecture Notes in Networks and Systems (Vol. 137). https://doi.org/10.1007/978-981-15-6198-6_18

6. Park, C., Eo, S., Moon, H., & Lim, H. (2021). Should we find another model?: Improving Neural Machine Translation Performance with ONE-Piece Tokenization Method without Model Modification. NAACL-HLT 2021 - 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Industry Papers. https://doi.org/10.18653/v1/2021.naacl-industry.13

7. Li, X., & Fourches, D. (2021). SMILES Pair Encoding: A Data-Driven Substructure Tokenization Algorithm for Deep Learning. Journal of Chemical Information and Modeling, 61(4). https://doi.org/10.1021/acs.jcim.0c01127

Elov Botir Boltayevich – texnika fanlari falsafa doktori, dotsent. Alisher Navoiy nomidagi Toshkent davlat o‘zbek tili va adabiyoti universiteti

E-mail: *elov@navoiy-uni.uz*

Xusainova Zilola Yuldashevna –Alisher Navoiy nomidagi Toshkent davlat o‘zbek tili va adabiyoti universiteti stajor-o‘qituvchi

E-mail: *xusainovazilola@navoiy-uni.uz*

Yodgorov Umidjov Saydilla o‘g‘li–Alisher Navoiy nomidagi Toshkent davlat o‘zbek tili va adabiyoti universiteti, o‘qituvchi

E-mail: yodgorov@navoiy-uni.uz

B. Elov, Z. Khusainova, U. Yodgorov.
Production of tokenizer for Uzbek language texts

Tokenization refers to the manipulation of unstructured language text into a well-structured (from a computer's point of view) representation in production. In the process of tokenization, the text is divided into parts that contain useful information called tokens. The purpose of the three articles is to describe the tokenization process, how it works, and how to produce a tokenizer.

Keywords: Tokenizatsiya, UzbSentencizer, document, sentence, token, ajratuvchilar, regular ifodalar, morfologik analizator.