ქართული ენის მოდელირება

თორნიკე ჯიჯელავა ცოტნე ბაბუნაშვილი

ბესიკი კაპანაძე ელენე ცირამუა

{tjije19, tbabu19, etsir19, bkapa18}@freeuni.edu.ge

მიმოხილვა

დავაიმპლემენტირეთ ქართული ენის მოდელი, რომელსაც შეუძლია წინადადების გენერაცია შესაბამისი კონტექსტის მეშვეობით. რამდენიმე მოდელზე ჩატარებული ცდების შემდგომ საუკეთესო შედეგი ორშრიანმა LSTM მოდელმა მოგვცა. შეზღუდული გამოთვლითი რესურსების არსებობის პირობებშიც, მოდელი ახდენს გამართულ აზრთან ახლოს მყოფი წინადადებების გენერაციას.

მოდელის გამოყენება შესაძლებელია როგორც ქართული წინადადების დაგენერირებისთვის, ასევე შესაბამისი რესურსების არსებობის პირობებში მოდელის დატრენინგება შესაძლებელია შედარებით დიდი მოცულობის ტექსტებზე უკეთესი შედეგების მისაღწევად.

1 შესავალი

ქართული ენის მოდელირების ამოცანა არ არის ისე დეტალურად დამუშავებული როგორც სხვა ფართომაშტაბიანი ენების. შესაბამისად მიმდინარე პროცესში რამდენიმე სირთულის გადალახვა მოგვიწია, რათა სასურველ შედეგამდე მივსულიყავით.

პირველ და ალბათ ერთ-ერთი ყველაზე მნიშვნელოვან გამოწვევას ქართული მონაცემების შეგროვება წარმოადგენდა. ქართული ენის კორპუსი არ არის საკმარისად დიდი დასამუშავებლად. პრობლემის გადასაჭრელად მოგვიხდა ვიკიპედიის ქართული dump-ების შეგროვება და დამუშავებაც. ამასთან ერთად ვცდილობდით საუკეთესო კორპუსი გვეპოვნა და შეგვერჩია მოდელისთვის.

ასევე აღსანიშნია, რომ ქართული სიტყვების word vector-ების გამოსადეგი და დამაკმაყოფილებელი რესურსი არ არსებობს და შესაბამისად მოგვიწია word2vec-ის დასწავლა შედარებით დიდ მონაცემებზე. ვექტორების ხარისხის შესაფასებლად გამოვიყენეთ სხვადასხვა ხერხები: სინონიმების, ანტონიმების, ანალოგიებისა და კლასტერების პოვნა.

რადგანაც ქართული ენის გენერირების ხარისხის შესამოწმებლად არ არის მეტრიკა შემუშავებული, ავაგეთ n-gram მოდელი რომელსაც ვიყენებთ როგორც benchmark-ს, ძირითადი მოდელის ევალუაციის დროს. ეს საშუალებას გვაძლევს გავიგოთ რამდენად კარგია ჩვენი მოდელი baseline-თან შედარებით.

ბევრის სხვადასხვა ვარიანტის ცდის შემდგომ, მთავარ მოდელად ორშრიანი LSTM ავარჩიეთ. ტექსტის გენერაციისთვის კი ვიყენებთ როგორც ხარბ ასევე არა ხარბ - beam search ალგორითმს.

2 მონაცემების შეგროვება

მონაცემების შეგროვების პირველ ეპატზე ვიკიპედიის dump-ები გამოვიყენეთ. შევაგროვეთ მონაცემების საკმაოდ დიდი მასალა, მაგრამ სამწუხაროდ ვიკიპედიას ქართული ენაზე არ აქვს საკმარისი მონაცემები. შემდგომ სხვადასხვა კორპუსების ძიება დავიწყეთ, რომლებიც ქართული ენის მოდელირებაში გამოგვადგებოდა. არჩევანი CC100 კორპუსზე შევაჯერეთ, რომელიც მილიონზე მეტ ქართულ ფორმალურ თუ არაფორმალურ წინადადებას შეიცავს. ამ მიმართულებით გუნდის წევრები, ბესო კაპანაძე და თორნიკე ჯიჯელავა მუშაობდნენ.

3. მონაცემთა ანალიზი და დამუშავება

მონაცემები, რომელიც შევაგროვეთ, თავდაპირველად არ იყო ისეთ მდგომარეობაში, რომ პირდაპირ დაგვეწყო მათზე მუშაობა, ემბედინგების და მოდელის აგება. შესაბამისად დაგვჭირდა, რომ დაგვემუშავებინა ჩვენ მიერ შეგროვებული მონაცემები, შემდეგ გაგვეანალიზებინა და მხოლოდ ამის შემდეგ დაგვეწყო უშუალოდ მოდელზე მუშაობა. ამ ყველაფერზე ელენე ცირამუამ და თორნიკე ჯიჯელავამ იმუშავეს.

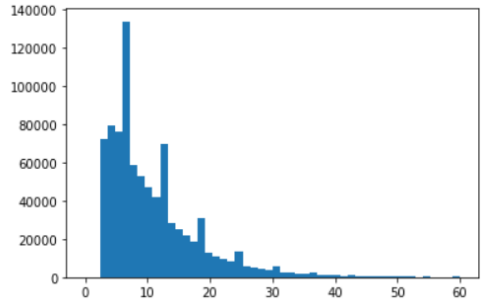
დამუშავება პირველ რიგში დავიწყეთ არაქართული სიმბოლოების ამოკლებით, ჩვენ მიერ შეგროვებულ სტატიებში არც თუ ისე იშვიათად გვხვდებოდა ლათინური სიმბოლოები, თუნდაც ემოჯიები და ა.შ. ამის გამო, ამოვიღეთ ისეთი ზედმეტი სიმბოლოები, რომლებიც ხელს შეგვიშლიდა და ქართული ენის მოდელისთვის არ იქნებოდა რელევანტური.

ამის შემდეგ, ასევე ამოვაკელით ციფრები, რიცხვები, რადგან როგორც ვიცით ენის მოდელში რიცხვების გამოყენება არ არის მიზანშეწონილი, სირტვებს შორის ემბედინგები შეიძლება დაარღვიოს მონაცემებში რიცხვების გამოყენებამ და არასასურველ შედეგამდე მიგვიყვანოს - სწორედ ამიტომ ამოვაკელით რიცხვებიც.

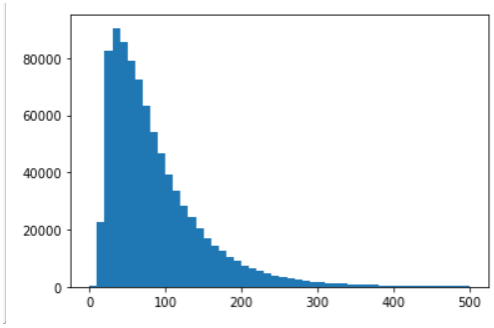
და ბოლოს, ამოვაკელით ისეთი წინადადებები, რომლების

სიგრძეც ზედმეტად პატარა იყო და კარგი მოდელის აგებაში ვერ დაგვეხმარებოდა. ამან ჩვენ მიერ შეგროვებული წინადადებების რაოდენობა დაახლოებით 20%ით შეამცირა, თუმცა ნამდვილად ერთ-ერთი აუცილებელი ეტაპი იყო რადგან წინადადებები რომელშიც ორი სიტყვა გვხვდებოდა, მოდელის ტრენინგში არ გამოგვადგებოდა.

უკვე გასუფთავებული და დამუშავებული მონაცემების ხარისხი და განაწილება შეგვიძლია სხვადსხვა გრაფიკის სახით ვნახოთ და დავრწმუნდეთ, რომ ჩვენთვის მისაღებ მდგომარეობამდე მივიყვანეთ.



ამ გრაფიკზე ვხედავთ, რომ წინადადებების სიგრძეები უკვე მისაღებია და საშუალოდ არის 10ის ტოლი.



ხოლო აქ ჩანს თუ რამდენი სიმბოლოა საშუალოდ ერთ წინადადებაში.

ასევე საინტერესოა თუ რომელი სიტყვები გვხვდება ჩვენ მიერ შეგროვებულ წინადადებებში ყველაზე ხშირად:

	Word	Word Count
0	ეს	48780
1	ამ	45270
2	ის	43746
3	არის	35337
4	იყო	30242
5	საქართველოს	29029
6	წლის	27792
7	რა	27336
8	მე	25129
9	მისი	22438
10	ერთი	22021
11	ახალი	16419
12	დიდი	16315
13	ერთ	16147
14	ს	15987
15	სხვა	15841
16	მათ	15575
17	ძალიან	15273
18	მას	14974
19	იმ	14388

4. მოდელები

4.1 Pre-trained word2vec

Word2vec-ის დასატრენინგებლად გამოვიყენეთ genism.models.Word2Vec, ხოლო მონაცემები შედგებოდა დაახლოებით 850 000 წინადადებისგან. ვექტორების შესაფასებლად კი სხვადასხვა ანალოგიებისა და კლასტერების მაგალითები გამოვიყენეთ.

სინონიმის მაგალითი

```
In [3]: trained_model.wv.most_similar('მანქანა')
Out[3]: [('ავტომობილი', 0.7522153258323669),
         ('მანქანით', 0.6987831592559814),
         ('მანქანის', 0.6920327544212341),
         ('ავტომანქანა', 0.645092785358429),
         ('სარეცხი', 0.6296919584274292),
         ('თვითმფრინავი', 0.6293297410011292),
         ('ყუთი', 0.6292032599449158),
         ('გემი', 0.6288377046585083),
         ('საყიდლად', 0.624182939529419),
         ('ტრაქტორი', 0.6215299367904663)]
```

მანქანის სინონიმები ჩვენმა მოდელმა იპოვა. მასთან ერთად იპოვა მასთან კონტექსტში მყოფი სიტყვებიც: სარეცხი, საყიდლად.

ანტონიმის მაგალითი

```
In [5]: trained_model.wv.most_similar('ღარიბი')
Out[5]: [('მდიდარი', 0.7816330790519714),
         ('შეძლებული', 0.7591644525527954),
         ('გაჭირვებული', 0.7345767021179199),
         ('კეთილშობილი', 0.7162554860115051),
         ('ხანდაზმული', 0.6996287703514099),
         ('გონიერი', 0.6984087228775024),
         ('მშრომელი', 0.6934995055198669),
         ('განათლებული', 0.688368558883667),
         ('ცნობისმოყვარე', 0.6784718036651611),
         ('ბრძენი', 0.6778247356414795)]
```

ღარიბი ხშირად გვხვდება წინადადებაში თავის ანტონიმთან ერთად. შესაბამისად მოდელმა იპოვა მდიდარი და შეძლებული.

საინტერესო შედეგი მივიღეთ ანალოგიების ამოცანის ამოხსნისას.

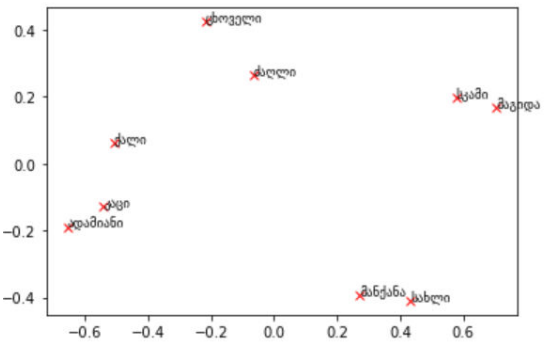
```
In [8]: trained_model.wv.most_similar(
            positive=['ქალი', 'ბაბუა'], negative=['კაცი'])
Out[8]: [('ბებია', 0.7487496137619019),
         ('ქალიშვილი', 0.727142333984375),
         ('შეყვარებული', 0.7100287675857544),
         ('რძალი', 0.7087268233299255),
         ('უსაყვარლესი', 0.6974164247512817),
         ('ტრანსგენდერი', 0.6903185844421387),
         ('ქმარი', 0.6896628141403198),
         ('ცოლი', 0.6744663715362549),
         ('ორსული', 0.6640662550926208),
         ('მოახლე', 0.662255048751831)]
```

მოდელი ანალოგიას კაცი - ბაბუა. ქალი - ? მარტივად გაუმკლავდა და იპოვა სწორი პასუხი ბებია.

ხოლო ანალოგიაზე ანალოგიაზე ქალი - დიასახლისი, კაცი - ? მოდელმა დაგვიბრუნა პასუხები მათხოვარი და ლოთი. ამის მთავარი მიზეზი არის ის, რომ მოდელი არის დატრენინგებული ქართულ „კონტექსტზე“, შესაბამისად არის Georgian-biased კაცები მიიჩნია ლოთები და მათხოვრები.

```
In [13]: trained_model.wv.most_similar(
             positive=['კაცი', 'დიასახლისი'], negative =['ქალი'])
Out[13]: [('მათხოვარი', 0.7229001522064209),
          ('ლოთი', 0.7155242562294006),
          ('მამიდა', 0.6963827610015869),
          ('ვაჟკაცი', 0.6925016641616821),
          ('მარტოსული', 0.685178279876709),
          ('მთვრალი', 0.6822971701622009),
          ('დამპალი', 0.6822889447212219),
          ('მოწადირე', 0.6811962127685547),
          ('მესაიდუმლე', 0.6804308295249939),
          ('ნათლია', 0.6794002056121826)]
```

მოდელმა კლასტერიზაციასაც გაართვა თავი და მარტივ მაგალითზე სიტყვები სწორ კლასტერებში მოაქცია.



გამოიკვეთა ოთხი ლოგიკური კლასტერი: 1) ქალი, კაცი, ადამიანი, 2) ცხოველი ძაღლი, 3) სკამი, მაგიდა, 4) მანქანა, სახლი. Word2vec-ის მოდელი თორნიკე ჯიჯელავამ და ბესო კაპანაძემ დაატრენინგეს.

4.2 Baseline (N-Gram) მოდელი

ენის მოდელის აგებისას აუცილებელია გვქონდეს baseline მოდელი და ამ შემთხვევაში გამოვიყენეთ N-Gram მოდელი, რომელიც ცოტნე ბაბუნაშვილმა და ელენე ცირამუამ ააგეს.

N-Gram მოდელის გამოყენების ძირითადი იდეა არის, რომ წინადადების კონტექსტის მიხედვით გავიგოთ შემდეგი სიტყვა. თუმცა, n-gram მოდელის უპირატესობა და სიმარტივე იმაში მდგომარეობს, რომ კონტექსტის დასადგენად არ არის საჭირო მთლიანი წინადადების შესწავლა. მთავარია შევისწავლოთ n-1 ტოკენი(N არის ჩვენ მიერ წინასწარ შერჩეული რიცხვი და N-gram მოდელის სიზუსტესაც განსაზღვრავს ეს).

სხვანაირად რომ ვთქვათ, N-Gram მოდელი დაფუძნებულია პირობით ალბათობაზე და სწორედ ამაზე დაყრდნობით ხდება იმის დადგენა თუ რა იქნება შემდეგი სიტყვა.

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

ჩვენ მიერ შეგროვებული მონაცემებით, დავატრენინგეთ N-Gram მოდელი, შევადგინეთ Vocabulary, რომელიც საკმაოდ ბევრ უნიკალურ სიტყვას შეიცავს - 483 820 სიტყვას.

```
print(len(model.vocab))
```

```
483820
```

ამ მოდელის დაგენერირებისთვის გამოვიყენეთ nltk.lm-ის Laplace მოდელი, რომელსაც ჩვენ მიერ შედგენილი მონაცემები და წინადადებები გადავეცით და დავატრენინგეთ.

ჩვენ გადავწყვიტეთ, რომ გამოგვეყენებინა 3-Gram მოდელი ანუ დაგვეთვალა 3 სიტყვას შორის ერთმანეთის მიმდევრობით ყოფნის ალბათობები და ამის მიხედვით შეგვედგინა მოდელი.

```
generate_sentence(model, num_words=20, random_seed=0)
```

```
'სფეროში სადაც ეს კომეწტარია გაკეთებული'
```

ვხედავთ, რომ მოდელი გვიგენერირებს წინადადებებს, რომლის სიტყვებიც ასე თუ ისე იდეურად დაკავშირებულია ერთმანეთთან.

ამის შემდეგ, იმის კარგად შესამოწმებლად თუ როგორი იყო დატრენინგებული n-gram მოდელი, დავთვალეთ Perplexity – test dataზე.

Perplexity on test data

```
test = flatten(list(ngrams(sent,n=3) for sent in testcorpus))
```

```
model.perplexity(test)
```

```
170344.31387370508
```

4.3 LSTM მოდელი

ჩვენი ქართული ენის მოდელის მთავარ მოდელად გამოვიყენეთ LTSM მოდელი - რეკურენტული ნეირონული ქსელი. ამისთვის პირველ რიგში საჭირო იყო სიტყვების ინდექსებში გადაქცევა. ამის შემდეგ დავაიმპორტეთ შედგენილი ემბედინგები, რაც არის მთავარი ნაწილი მოდელის დატრენინგებაში. ემბედინგების, DataLoaderის აღწერის შემდეგ გადავედით თვითონ LTSMის ნეირონული ქსელის გამოყენებაზე და მის დატრენინგებაზე. დატრენინგებული მოდელი კი შევინახეთ, რათა გამოგვეყენებინა მისი predictionები შემდგომში ევალუაციისას და ტექსტის გენერაციისას. დროისა და რესურსების შეზღუდულობის გამო, ამ ნაწილზე ფაქტობრივად პარალელურ რეჟიმში ვმუშაობდით გუნდის ოთხივე წევრი.

5. მოდელის ევალუაცია

5.1 ტექსტის გენერირება

ტექსტის გენერაციისთვის გამოვიყენეთ ორი ალგორითმი. პირველი ალგორითმი რენდომულად ირჩევს სიტყვას ალბათობების მიხედვით. ამ მეთოდით მიღებული წინადადებების მაგალითებია:

```
predict(model, 'მერე რა რომ ზამთარია', 7)
```

```
'მერე რა რომ ზამთარია რომელშიც ითქვა შუახნის მწუხრი ველთა სანუგეშოდ
```

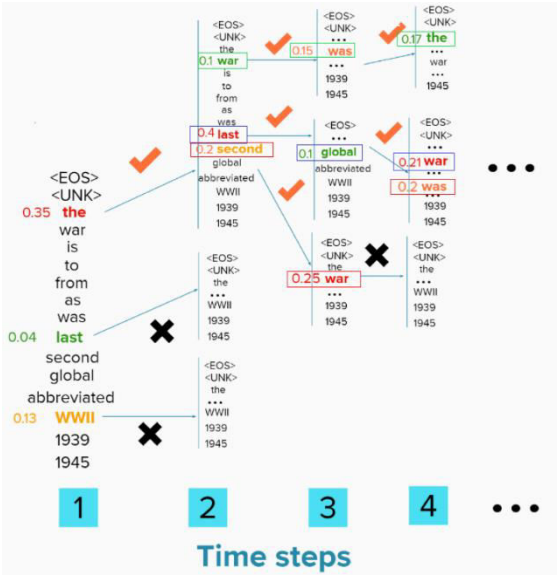
```
predict(model, 'ხვალ', 7)
```

```
'ხვალ სადაც ვუჩივლებდი ასევე ნახსენები უწდა ფორმალობად ბოლომდე'
```

პირველზე შესაძლოა ითქვას, რომ საკმაოდ გამართული წინადადებაა, ხოლო მეორე - შედარებით აზრსმოკლებული.

Beam search

ალგორითმი პირობითი ალბათობების საფუძველზე არჩევს მომავალ N სიტყვას. თუ ხარბი ალგორითმი ყოველ ჯერზე მიმდინარე საუკეთესო სიტყვაზე შეაჯერებს არჩევანს, beam search ითვალისწინებს n-1 სიტყვას, რომლებმაც შესაძლოა განავრცონ წინადადება. ხარგ ალგორითმთან შედარებით მეტი გამოთვლების ჩატარება არის საჭირო, მაგრამ გარანტიას გვაძლევს, რომ შემდგომი N სიტყვა შედარებით უკეთ იქნება დაკავშირებული ერთმანეთთან.



მაგალითზე the გვაქვს სამი შესაძლო მომდევნო სამი სიტყვის არჩევის საშუალება.

The last global war, The second war was, The war was the. ალბათობების გამრავლების შემდგომ აირჩევს Beam search საუკეთესო ვარიანტს. ჩვენს მიერ იმპლემენტირებული beam search-ის num_beams არის 3, რის გამოც წინადადების სამეული ერთმანეთთან კარგად არის შეკავშირებული.

```
generate_with_beamsearch(model, 'გუშინ', 8)
```

```
'გუშინ საქართველოს პრეზიდენტის და სხვა დროს ამ უფლება იმის
```

მიუხედავად ძებნის ალგორითმის გაუმჯობესებისა, წინადადების გამართულობის მხრივ სიტუაცია მკვეთრად არ შეცვლილა.

5.2 Word2vecs მოდელის ტრენინგის შემდეგ

მოდელის ტრენინგის შემდეგ ემბედინგები კვლავ კარგად ართმევენ ანტონიმების, სინონიმების და ანალოგიების ამოცანებს.

სინონიმის მაგალითი:

```
In [19]: lstm_trained_embedding_word2vec.most_similar('კაცი')
```

```
Out[19]: [('ბიჭი', 0.7427014112472534),
          ('ქალი', 0.7313578128814697),
          ('ადამიანი', 0.7084077596664429),
          ('გოგო', 0.7048305869102478),
          ('პოლიციელი', 0.6688650250434875),
          ('კაცს', 0.6588519811630249),
          ('კაციც', 0.6536153554916382),
          ('ყმაწვილი', 0.6511042714118958),
          ('ქალია', 0.6386712789535522),
          ('გოგონა', 0.6384297609329224)]
```

ანალოგიის მაგალითი:

```
In [25]: lstm_trained_embedding_word2vec.most_similar(
             positive=['კაცი', 'დედა'], negative =['ქალი'])
```

```
Out[25]: [('მამა', 0.6966497898101807),
          ('ძმა', 0.6835110783576965),
          ('მღვდელი', 0.6759638786315918),
          ('ყრმა', 0.6744189262390137),
          ('მწყემსი', 0.6697568893432617),
          ('დედ', 0.6549163460731506),
          ('კაციც', 0.6480010151863098),
          ('დეიდა', 0.646030068397522),
          ('დედაკაცი', 0.6448473930358887),
          ('დავითი', 0.644156813621521)]
```

5.3 LSTM და N-gram-ის შედარება

როგორც მოსალოდნელი იყო, LSTM მოდელი უფრო ეფექტური აღმოჩნდა ვიდრე N-gram. ამის გადასამოწმებლად perplexity დავთვალეთ ორივე მოდელის. N-gram-ზე მივიღეთ 170344, მაშინ როცა LSTM-მოდელის perplexity მხოლოდ 812 აღმოჩნდა.

N-gram ის მიერ შედგენილ წინადადებას რომ დავაკვირდეთ, მას თითქოს კონტექსტი აკლია განსხვავებით LSTM-ისა.

```
generate_sentence(model, num_words=20, random_seed=5)
```

```
'მაგრამ რომელი სარეკლამო საშუალება ნაქარგი სურათის სახით'
```

მიუხედავად ამისა LSTM მოდელსაც უჭირს გამართული ქართული წინადადების შედგენა. ასევე გასათვალისწინებელია, რომ სიტყვებზე, რომლებიც არ არის მონაცემებში, LSTM მაინც ახერხებს სხვა სიტყვების ხარჯზე

კონტექსტის დაჭერას, რაც უჭირს N-gram-ს. უკანასკნელ ორ ნაწილზე (მოდელის ევალუაცია და LSTM-ისა და N-gram-ის შედარება) ელენე ცირამუამ და ცოტნე ბაბუნაშვილმა იმუშავეს.

6 დასკვნა

შესაძლოა ითქვას, რომ დასახული მიზანი, ქართული ენის მოდელირება, მიღწეული იქნა. ქართული ენა ერთ-ერთი ყველაზე რთული ენაა და რთულია მოდელმა გამართული, უშეცდომო წინადადებები დააგენერიროს. მიუხედავად ამისა, ჩვენს მიერ აგებული მოდელი წარმატებით იჭერს გადაცემული სიტყვების კონტექსტს და გამართულ წინადადებასთან ახლოს მყოფ წინადადებამდე ავსებს მას.

გამოყენებული ლიტერატურა:

LTSM

https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial

CrossEntropyLoss

https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

Perplexity

https://towardsdatascience.com/evaluation-of-language-models-through-perplexity-and-shannon-visualization-method-9148fbe10bd0

Beam Search

https://towardsdatascience.com/foundations-of-nlp-explained-visually-beam-search-how-it-works-1586b9849a24

N-Gram

https://www.nltk.org/api/nltk.lm.html