Deep Learning for Time Series Forecasting

2022.08.28.

김동현

Contents

1. 자연어 처리란
2. 시소러스
3. 통계 기반 기법
4. 통계 기반 기법 개선하기
5. 정리

- 목표 : 컴퓨터에게 ‘단어의 의미‘ 이해시키기
- 교재 : Deep Learning from Scratch 2(Chapter. 2)

1. 자연어 처리란

- **자연어(Natural Language)**
- 한국어와 영어 등 평소에 쓰는 말

- **자연어 처리(Natural Language Processing, NLP)**
- 자연어를 처리하는 분야
- 우리의 말을 컴퓨터에게 이해시키기 위한 기술(분야)

- **NLP 예시**
- 검색 엔진, 기계 번역
- 질의응답 시스템, IME(입력기 전환), 문장 자동요약과 감정분석 등

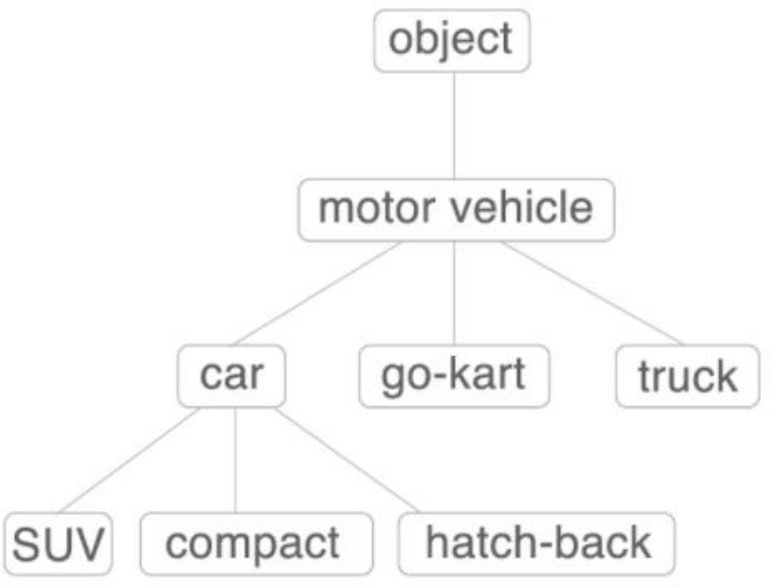
1. 자연어 처리란

- **단어의 의미**
- 의미의 최소 단위
- 자연어를 컴퓨터에게 이해시키는데 '단어의 의미' 를 이해시키는 것이 중요함.

- **기법 종류**
- 시소러스를 활용한 기법
- 통계 기반 기법
- 추론 기반 기법(word2vec)

2. 시소러스

- **시소러스**
- 유의어 사전
- ‘뜻이 같은 단어(동의어)’나 ‘뜻이 비슷한 단어(유의어)’가 한 그룹으로 분류됨.
- 예시 1 : car = auto, automobile, machine, motorcar



- 단어 사이의 ‘상위와 하위’ 혹은 ‘전체와 부분’ 등 더 세세한 관계까지 정의해둔 경우가 있음.
- 예시 2 : 좌측 그림

-> 이러한 단어 네트워크를 이용하여 컴퓨터에 단어 사이의 관계를 가르칠 수 있음.

2. 시소러스

- **WordNet**
- 자연어 처리 분야에서 가장 유명한 시소러스임.
- 프린스턴 대학교에서 1985년부터 구축하기 시작한 전통 있는 시소러스임.

어떻게 활용?

- 유의어를 얻거나 단어 네트워크를 이용할 수 있음.
- 단어 네트워크를 이용해 단어 사이의 유사도를 구할 수 있음.

2. 시소러스

- **시소러스의 문제점**
- 사람이 수작업으로 레이블링하는 방식에는 크나큰 결점이 존재함.

1) 시대 변화에 대응하기 어렵다.

- 신조어 : crowd funding(크라우드 펀딩)
- 영단어 예시 : heavy(무겁다, 심각하다)

2) 사람을 쓰는 비용은 크다.

3) 단어의 미묘한 차이를 표현할 수 없다.

- 영단어 예시 : vintage(낡고 오래된 것), retro(복고)

-> 해결 : 통계 기반 기법, 추론 기반 기법

3. 통계 기반 기법

말뭉치(corpus)

- 대량의 텍스트 데이터
- 다만, 맹목적으로 수집된 텍스트 데이터가 아닌 자연어 처리나 애플리케이션을 염두에 두고 수집된 텍스트 데이터
- 사람이 쓴 글 -> 자연어에 대한 사람의 지식이 충분히 담겨 있다 볼 수 있음.

통계 기반 기법의 목표

사람의 지식으로 가득한 말뭉치에서 자동으로, 효율적으로 핵심을 추출하는 것

3. 통계 기반 기법

- **말뭉치 전처리**
- 전처리 : 텍스트 데이터를 단어로 분할하고 그 분할된 단어들을 단어 ID 목록으로 변환하는 일

```python
def preprocess(text):
    text = text.lower()
    text = text.replace('.', ' .')
    words = text.split(' ')

    word_to_id = {}
    id_to_word = {}
    for word in words:
        if word not in word_to_id:
            new_id = len(word_to_id)
            word_to_id[word] = new_id
            id_to_word[new_id] = word

    corpus = np.array([word_to_id[w] for w in words])

    return corpus, word_to_id, id_to_word
```

```python
text = 'You say goodbye and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)
```

```python
corpus
```

```
array([0, 1, 2, 3, 4, 1, 5, 6])
```

```python
word_to_id
```

```
{'you': 0, 'say': 1, 'goodbye': 2, 'and': 3, 'i': 4, 'hello': 5, '.': 6}
```

```python
id_to_word
```

```
{0: 'you', 1: 'say', 2: 'goodbye', 3: 'and', 4: 'i', 5: 'hello', 6: '.'}
```

3. 통계 기반 기법

- **단어의 분산 표현(distributional representation)**
- '단어의 의미'를 정확하게 파악할 수 있는 벡터 표현

- **분포 가설(distributional hypothesis)**
- 아이디어 : '단어의 의미는 주변 단어에 의해 형성된다.'

-> 단어 자체에는 의미가 없고, 그 단어가 사용된 맥락이 의미를 형성함.

- 예시 : "I drink beer", "We drink wine"

맥락(context) : 주변에 놓인 단어(특정 단어를 중심에 둔 그 주변 단어)

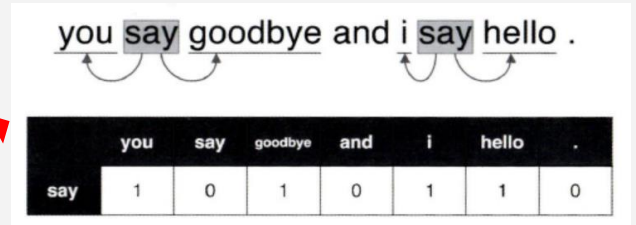
윈도우 크기(window size) : 맥락의 크기(주변 단어를 몇 개 포함할지?)

- 예시(윈도우 크기가 2인 맥락) :

3. 통계 기반 기법

- **동시발생 행렬(co-occurrence matrix)**
- 모든 단어에 대해 동시발생하는 단어를 표에 정리한 것
- 각 행은 해당 단어를 표현한 벡터가 됨(윈도우 크기가 1인 맥락).

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	0	0	0
and	0	0	1	0	1	0	0
i	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0



	you	say	goodbye	and	i	hello	.
say	1	0	1	0	1	1	0

```python
def create_co_matrix(corpus, vocab_size, window_size=1):
    '''동시발생 행렬 생성
    :param corpus: 말뭉치(단어 ID 목록)
    :param vocab_size: 어휘 수
    :param window_size: 윈도우 크기(윈도우 크기가 1이면 타깃 단어 좌우 한 단어씩이 맥락에 포함)
    :return: 동시발생 행렬
    '''
    corpus_size = len(corpus)
    co_matrix = np.zeros((vocab_size, vocab_size), dtype=np.int32)

    for idx, word_id in enumerate(corpus):
        for i in range(1, window_size + 1):
            left_idx = idx - i
            right_idx = idx + i

            if left_idx >= 0:
                left_word_id = corpus[left_idx]
                co_matrix[word_id, left_word_id] += 1

            if right_idx < corpus_size:
                right_word_id = corpus[right_idx]
                co_matrix[word_id, right_word_id] += 1

    return co_matrix
```

3. 통계 기반 기법

- **벡터 간 유사도**
- 코사인 유사도를 자주 이용한다고 함.
- 수식 : $$\text{similarity}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \frac{x_1y_1 + \cdots + x_ny_n}{\sqrt{x_1^2 + \cdots + x_n^2}\sqrt{y_1^2 + \cdots + y_n^2}}$$
- 의미 : 두 벡터가 가리키는 방향이 얼마나 비슷한가?

- 벡터의 방향이 완전히 같으면 1, 완전히 반대라면 -1이 됨.

```
def cos_similarity(x, y, eps=1e-8):
    '''코사인 유사도 산출

    :param x: 벡터
    :param y: 벡터
    :param eps: '0으로 나누기'를 방지하기 위한 작은 값
    :return:
    '''
    nx = x / (np.sqrt(np.sum(x ** 2)) + eps)
    ny = y / (np.sqrt(np.sum(y ** 2)) + eps)
    return np.dot(nx, ny)
```

3. 통계 기반 기법

- **유사 단어의 랭킹 표시**
- 어떤 단어가 검색어로 주어지면, 그 검색어와 비슷한 단어를 유사도 순으로 출력하는 함수

```python
def most_similar(query, word_to_id, id_to_word, word_matrix, top=5):
    '''유사 단어 검색

    :param query: 쿼리(텍스트)
    :param word_to_id: 단어에서 단어 ID로 변환하는 딕셔너리
    :param id_to_word: 단어 ID에서 단어로 변환하는 딕셔너리
    :param word_matrix: 단어 벡터를 정리한 행렬. 각 행에 해당 단어 벡터가 저장되어 있다고 가정한다.
    :param top: 상위 몇 개까지 출력할 지 지정
    '''
    if query not in word_to_id:
        print('%s(을)를 찾을 수 없습니다.' % query)
        return

    print('\n[query] ' + query)
    query_id = word_to_id[query]
    query_vec = word_matrix[query_id]

    # 코사인 유사도 계산
    vocab_size = len(id_to_word)

    similarity = np.zeros(vocab_size)
    for i in range(vocab_size):
        similarity[i] = cos_similarity(word_matrix[i], query_vec)

    # 코사인 유사도를 기준으로 내림차순으로 출력
    count = 0
    for i in (-1 * similarity).argsort():
        if id_to_word[i] == query:
            continue
        print(' %s: %s' % (id_to_word[i], similarity[i]))

        count += 1
        if count >= top:
            return
```

```python
text = 'You say goodbye and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)
vocab_size = len(word_to_id)
C = create_co_matrix(corpus, vocab_size)

most_similar('you', word_to_id, id_to_word, C, top=5)
```

```
[query] you
 goodbye: 0.7071067691154799
 i: 0.7071067691154799
 hello: 0.7071067691154799
 say: 0.0
 and: 0.0
```

4. 통계 기반 기법 개선하기

- **상호정보량**
- 동시발생 행렬의 원소는 두 단어가 동시에 발생한 횟수를 나타냄.
- 발생횟수라는 것은 그리 좋은 특징이 아님.
- 예시 : ‘the’, ‘car’의 동시발생

-> 해결 : 점별 상호정보량(Pointwise Mutual Information, PMI)

4. 통계 기반 기법 개선하기

- **점별 상호정보량(PMI)**
- 수식 : $$\mathrm{PMI}(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$
- 동시발생 행렬을 사용한 수식 : $$\mathrm{PMI}(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)} = \log_2 \frac{\frac{C(x,y)}{N}}{\frac{C(x)}{N}\frac{C(y)}{N}} = \log_2 \frac{C(x,y)\cdot N}{C(x)C(y)}$$
- 예시 : ‘the’, ‘car’의 동시발생

  - ‘the’ = 1000번, ‘car’ = 20번, ‘drive’ = 10번
  - ‘the’ & ‘car’ = 10번, ‘car’ & ‘drive’ = 5번

$$\mathrm{PMI}(\text{“the”},\text{“car”}) = \log_2 \frac{10\cdot 10000}{1000\cdot 20} \approx 2.32 \qquad \mathrm{PMI}(\text{“car”},\text{“drive”}) = \log_2 \frac{5\cdot 10000}{20\cdot 10} \approx 7.97$$

- 문제점 : 두 단어의 동시발생 횟수가 0이면 log(0) = -inf가 됨.

4. 통계 기반 기법 개선하기

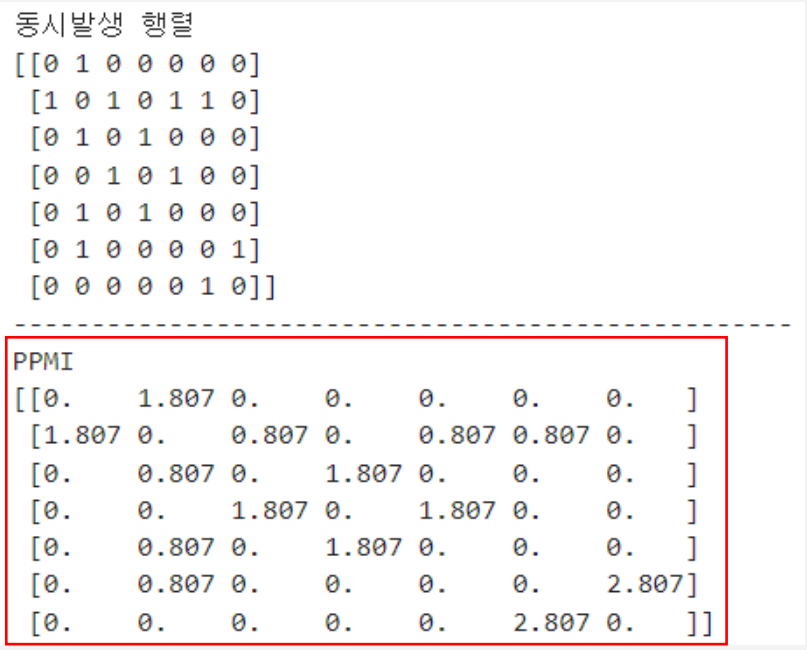
- **양의 상호정보량(Positive PMI, PPMI)**
- 수식 : $\mathrm{PPMI}(x,y) = \max(0, \mathrm{PMI}(x,y))$
- 앞서 나온 PMI의 문제점(log(0) = -inf)을 해결할 수 있음.
- 위 식에 따라 PMI가 음수일 때는 0으로 취급함.

```python
def ppmi(C, verbose=False, eps = 1e-8):
    '''PPMI(점별 상호정보량) 생성

    :param C: 동시발생 행렬
    :param verbose: 진행 상황을 출력할지 여부
    :return:
    '''
    M = np.zeros_like(C, dtype=np.float32)
    N = np.sum(C)
    S = np.sum(C, axis=0)
    total = C.shape[0] * C.shape[1]
    cnt = 0

    for i in range(C.shape[0]):
        for j in range(C.shape[1]):
            pmi = np.log2(C[i, j] * N / (S[j]*S[i]) + eps)
            M[i, j] = max(0, pmi)

            if verbose:
                cnt += 1
                if cnt % (total//100 + 1) == 0:
                    print('%.1f%% 완료' % (100*cnt/total))
    return M
```

```python
text = 'You say goodbye and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)
vocab_size = len(word_to_id)
C = create_co_matrix(corpus, vocab_size)
W = ppmi(C)

np.set_printoptions(precision=3)  # 유효 자릿수를 세 자리로 표시
print('동시발생 행렬')
print(C)
print('-'*50)
print('PPMI')
print(W)
```

```
동시발생 행렬
[[0 1 0 0 0 0 0]
 [1 0 1 0 1 1 0]
 [0 1 0 1 0 0 0]
 [0 0 1 0 1 0 0]
 [0 1 0 1 0 0 0]
 [0 1 0 0 0 0 1]
 [0 0 0 0 0 1 0]]
--------------------------------------------------
PPMI
[[0.    1.807 0.    0.    0.    0.    0.   ]
 [1.807 0.    0.807 0.    0.807 0.807 0.   ]
 [0.    0.807 0.    1.807 0.    0.    0.   ]
 [0.    0.    1.807 0.    1.807 0.    0.   ]
 [0.    0.807 0.    1.807 0.    0.    0.   ]
 [0.    0.807 0.    0.    0.    0.    2.807]
 [0.    0.    0.    0.    0.    2.807 0.   ]]
```

4. 통계 기반 기법 개선하기



동시발생 행렬
```
[[0 1 0 0 0 0 0]
 [1 0 1 0 1 1 0]
 [0 1 0 1 0 0 0]
 [0 0 1 0 1 0 0]
 [0 1 0 1 0 0 0]
 [0 1 0 0 0 0 1]
 [0 0 0 0 0 1 0]]
```

PPMI
```
[[0.    1.807 0.    0.    0.    0.    0.   ]
 [1.807 0.    0.807 0.    0.807 0.807 0.   ]
 [0.    0.807 0.    1.807 0.    0.    0.   ]
 [0.    0.    1.807 0.    1.807 0.    0.   ]
 [0.    0.807 0.    1.807 0.    0.    0.   ]
 [0.    0.807 0.    0.    0.    0.    2.807]
 [0.    0.    0.    0.    0.    2.807 0.   ]]
```

- 문제점 : 말뭉치의 어휘 수가 증가함에 따라 각 단어 벡터의 차원 수도 증가함.
- 예시 : 말뭉치의 어휘 수 = 10만개 = 벡터 차원 수

- 문제점 2 : 원소 대부분이 0임.
  각 원소의 중요도가 낮다는 뜻임.
  노이즈에 약하고 견고하지 못함.

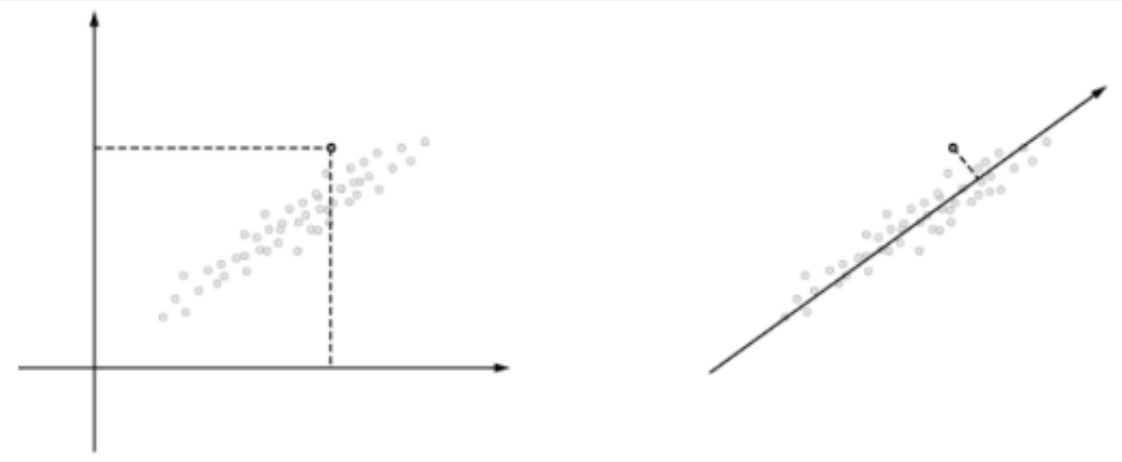
-> 해결 : 벡터의 차원 감소

4. 통계 기반 기법 개선하기

- **차원 감소(dimensionality reduction)**
- 벡터의 차원을 줄이는 방법
- 핵심 : 중요한 정보는 최대한 유지하면서 줄이는 것

그림으로 이해하는 차원 감소

- 2차원 데이터를 1차원으로 표현하기 위해 '데이터를 넓게 분포시키는 축'을 찾아야 함.
- 1차원 값만으로도 데이터의 본질적인 차이를 구별할 수 있어야 함.



4. 통계 기반 기법 개선하기

- **<u>차원 감소 방법</u>** : 특잇값분해(Singular Value Decomposition, SVD)
- 수식 : $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}$
- U, V는 직교행렬(orthogonal matrix)이고 열벡터는 서로 직교함.
- S는 대각행렬(diagonal matrix)임.

<u>수식의 시각적 표현</u> :

- U는 어떠한 공간의 축을 형성함. -> '단어 공간'
- S의 대각성분에는 특잇값(singular value)이 큰 순서로 나열되어 있음.

- 특잇값(singular value) : '해당 축'의 중요도

4. 통계 기반 기법 개선하기

- 중요도가 낮은 원소(특잇값이 작은 원소)를 깎아내는 방법을 생각할 수 있음.

SVD에 의한 차원 감소 :

- 행렬 S에서 특잇값이 작다면 중요도가 낮다는 뜻임.
- 행렬 U에서 여분의 열벡터를 깎아내어 원래의 행렬을 근사할 수 있음.

Python 계산

- Truncated SVD는 특잇값이 작은 것은 버리는(truncated) 방식으로 성능 향상을 꾀함.
- Scikit-learn library의 Truncated SVD를 이용하는 것이 좋음.

4. 통계 기반 기법 개선하기

- **PTB Dataset**
- PTB : Penn Treebank
- 말뭉치 예시 :

```
consumers may want to move their telephones a little closer to the tv set
<unk> <unk> watching abc 's monday night football can now vote during <unk> for the greatest play in N years from among four or five <unk> <unk>
two weeks ago viewers of several nbc <unk> consumer segments started calling a N number for advice on various <unk> issues
and the new syndicated reality show hard copy records viewers ' opinions for possible airing on the next day 's show
interactive telephone technology has taken a new leap in <unk> and television programmers are racing to exploit the possibilities
eventually viewers may grow <unk> with the technology and <unk> the cost
```

- 말뭉치 확인 코드 :

```
from dataset import ptb

corpus, word_to_id, id_to_word = ptb.load_data('train')

print('말뭉치 크기:', len(corpus))
print('corpus[:30]:', corpus[:30])
print()
print('id_to_word[0]:', id_to_word[0])
print('id_to_word[1]:', id_to_word[1])
print('id_to_word[2]:', id_to_word[2])
print()
print("word_to_id['car']:", word_to_id['car'])
print("word_to_id['happy']:", word_to_id['happy'])
print("word_to_id['lexus']:", word_to_id['lexus'])
```

```
말뭉치 크기: 929589
corpus[:30]: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29]

id_to_word[0]: aer
id_to_word[1]: banknote
id_to_word[2]: berlitz

word_to_id['car']: 3856
word_to_id['happy']: 4428
word_to_id['lexus']: 7426
```

4. 통계 기반 기법 개선하기

- **PTB Dataset 평가**

```
window_size = 2
wordvec_size = 100

corpus, word_to_id, id_to_word = ptb.load_data('train')
vocab_size = len(word_to_id)
print('동시발생 수 계산 ...')
C = create_co_matrix(corpus, vocab_size, window_size)
print('PPMI 계산 ...')
W = ppmi(C, verbose=True)

print('calculating SVD ...')
try:
    # truncated SVD (빠르다!)
    from sklearn.utils.extmath import randomized_svd
    U, S, V = randomized_svd(W, n_components=wordvec_size, n_iter=5,
                             random_state=None)
except ImportError:
    # SVD (느리다)
    U, S, V = np.linalg.svd(W)

word_vecs = U[:, :wordvec_size]

querys = ['you', 'year', 'car', 'toyota']
for query in querys:
    most_similar(query, word_to_id, id_to_word, word_vecs, top=5)
```

[query] you
i: 0.702039909619
we: 0.699448543998
've: 0.554828709147
do: 0.534370693098
else: 0.512044146526

[query] car
luxury: 0.620933665528
auto: 0.615559874277
cars: 0.569818364381
vehicle: 0.498166879744
corsica: 0.472616831915

[query] year
month: 0.731561990308
quarter: 0.658233992457
last: 0.622425716735
earlier: 0.607752074689
next: 0.601592506413

[query] toyota
motor: 0.738666107068
nissan: 0.677577542584
motors: 0.647163210589
honda: 0.628862370943
lexus: 0.604740429865

-> 단어의 의미 혹은 문법적인 관점에서 비슷한 단어들이 가까운 벡터로 나타남.

5. 정리

- **시소러스 기반 기법에서는 단어들의 관련성을 사람이 수작업으로 하나씩 정의함.**

-> 매우 힘들고 표현력(미세한 차이를 나타낼 수 없는 등)에도 한계가 있음.

- **통계 기반 기법은 말뭉치로부터 단어의 의미를 자동으로 추출하고,**
  그 의미를 벡터로 표현함.

1) 단어의 동시발생 행렬 구하기
2) PPMI 행렬로 변환
3) SVD를 이용해 차원 감소

-> 문법적인 용법면에서 비슷한 단어들이 벡터 공간에서도
서로 가까이 모여 있음을 확인함.

THANK YOU