Vector Semantics

Dr.Furkan Göz

Aralık 4, 2023

Problem

- Sıklık önemlidir ancak kelimeler arası ilişkiyi ölçmek için yeterli olmayabilir.
- 'the' veya 'a' gibi çok sık kullanılan kelimeler önemsizdir.
- Bu problem nasıl çözülebilir?

TF-IDF

Q: The cat.
D1: The cat is on the mat.
D2: My dog and cat are the best.
D3: The locals are playing.

TF(“the”, D1) = 2/6 = 0.33

TF(“the”, D2) = 1/7 = 0.14

TF(“the”, D3) = 1/4 = 0.25

TF(“cat”, D1) = 1/6 = 0.17

TF(“cat”, D2) = 1/7 = 0.14

TF(“cat”, D3) = 0/4 = 0

IDF(“the”) = log(3/3) = log(1) = 0

IDF(“cat”) = log(3/2) = 0.18

TF-IDF(“the”, D1) = 0.33 * 0 = 0

TF-IDF(“the, D2) = 0.14 * 0 = 0

TF-IDF(“the”, D3) = 0.25 * 0 = 0

TF-IDF(“cat”, D1) = 0.17 * 0.18= 0.0306

TF-IDF(“cat, D2) = 0.14 * 0.18= 0.0252

TF-IDF(“cat”, D3) = 0 * 0 = 0

Örnek

Q: The cat.
D1: The cat is on the mat.
D2: My dog and cat are the best.
D3: The locals are playing.

TF-IDF("the", D1) = 0.33 * 0 = 0

TF-IDF("the, D2) = 0.14 * 0 = 0

TF-IDF("the", D3) = 0.25 * 0 = 0

TF-IDF("cat", D1) = 0.17 * 0.18= 0.0306

TF-IDF("cat, D2) = 0.14 * 0.18= 0.0252

TF-IDF("cat", D3) = 0 * 0 = 0

TF-IDF score for D1 = 0 + 0.0306 = 0.0306

TF-IDF score for D2 = 0 + 0.0252 = 0.0252

TF-IDF score for D3 = 0 + 0 = 0

Pointwise Mutual Information (PMI)

- Anternatif bir yöntem
- x ve y olayının, bağımsız olmaları durumunda ne sıklıkla meydana geldiğinin ölçüsüdür.
- PMI -∞ile $+\infty$arasında değişir.
- Büyük derlemlerde daha mantıklıdır
- Negatif PMI değerleri 0 ile değiştirilir

$$\text{PPMI}(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0)$$

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

Co-occurrence counts for four words in 5 contexts in the Wikipedia corpus

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}, \quad p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}, \quad p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$P(\text{w=information, c=data}) = \frac{3982}{11716} = .3399$$

$$P(\text{w=information}) = \frac{7703}{11716} = .6575$$

$$P(\text{c=data}) = \frac{5673}{11716} = .4842$$

$$\text{PPMI(information,data)} = \log 2(.3399/(.6575 * .4842)) = .0944$$

Örnek

```
import nltk
from nltk.corpus import wordnet as wn


nltk.download('wordnet')

def find_synonyms(word):
    synonyms = []
    for syn in wn.synsets(word):
        for lemma in syn.lemmas():
            synonyms.append(lemma.name())
    return set(synonyms)

word = "technology"
synonyms = find_synonyms(word)
print(synonyms)
```

```
{'engineering_science', 'engineering', 'applied_science', 'technology'}
```

Örnek

```
from collections import defaultdict
import math
texts = [
    "kedi ve köpek bahçede oynuyor",
    "köpek ve kuş parkta",
    "kedi ve kuş bahçede",
    "bahçede bir kedi var"
]
word_freq = defaultdict(int)
pair_freq = defaultdict(int)
for text in texts:
    words = text.split()
    unique_words = set(words)
    for word in unique_words:
        word_freq[word] += 1
        for other_word in unique_words:
            if word != other_word:
                pair = tuple(sorted([word, other_word]))
                pair_freq[pair] += 1

def pmi(word1, word2, pair_freq, word_freq, total_docs):
    pair = tuple(sorted([word1, word2]))
    prob_pair = pair_freq[pair] / total_docs
    prob_word1 = word_freq[word1] / total_docs
    prob_word2 = word_freq[word2] / total_docs
    pmi = math.log(prob_pair / (prob_word1 * prob_word2))
    return pmi
example_pairs = [("kedi", "köpek"), ("kedi", "kuş"), ("bahçede", "kedi")]
pmi_values = {pair: pmi(pair[0], pair[1], pair_freq, word_freq, len(texts)) for pair in example_pairs}
pmi_values
```

```
{('kedi', 'köpek'): 0.28768207245178085,
 ('kedi', 'kuş'): 0.28768207245178085,
 ('bahçede', 'kedi'): 0.9808292530117262}
```

Örnek


```python
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

categories = ['comp.graphics', 'sci.space']
newsgroups = fetch_20newsgroups(subset='all', categories=categories, remove=('headers', 'footers', 'quotes'))

vectorizer = CountVectorizer(stop_words='english', max_features=500)
vectors = vectorizer.fit_transform(newsgroups.data)

cosine_similarities = cosine_similarity(vectors)

print("Kosinüs Benzerlikleri (İlk Beş Metin Arasında):")
print(cosine_similarities[:5, :5])
```

```
Kosinüs Benzerlikleri (İlk Beş Metin Arasında):
[[1.         0.05256486 0.3407771  0.18866665 0.01971424]
 [0.05256486 1.         0.01088823 0.04973203 0.04535234]
 [0.3407771  0.01088823 1.         0.         0.        ]
 [0.18866665 0.04973203 0.         1.         0.01441275]
 [0.01971424 0.04535234 0.         0.01441275 1.        ]]
```

Word Embeddings

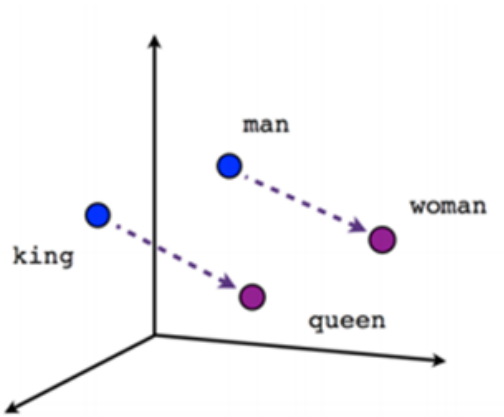
Geleneksel yöntemler:

- one hot encoding kullanılabilir.
  - Sözlükteki her kelime bir vektörde bir bit pozisyonu ile temsil edilir.
  - Örneğin, 10000 kelimelik bir sözlüğümüz varsa ve "Merhaba" sözlükteki 4. kelime ise şöyle temsil edilir: 0 0 0 1 0 0 ... 0 0 0
- Doküman temsili kullanır.
  - Sözlükteki her kelime, dokümanlardaki varlığı ile temsil edilir.
  - Örneğin, 1M dokümanlık bir derlemimiz varsa ve "Merhaba" sadece 1., 3. ve 5. dokümanlarda yer alıyorsa şöyle temsil edilir: 1 0 1 0 1 0 ... 0 0 0

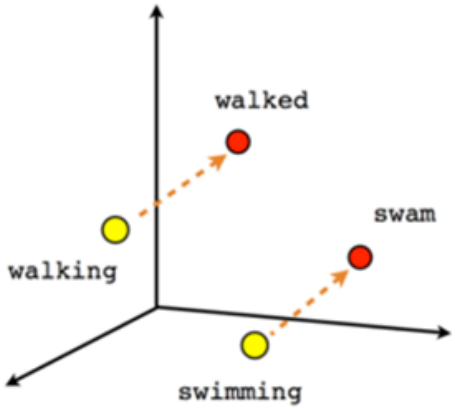
word embeddings

- Her kelimeyi sabit sayıda boyutta bir vektör olarak temsil edilen bir nokta olarak saklar.
- Örneğin, "Merhaba" şöyle temsil edilebilir: [0.4, -0.11, 0.55, 0.3 ... 0.1, 0.02].

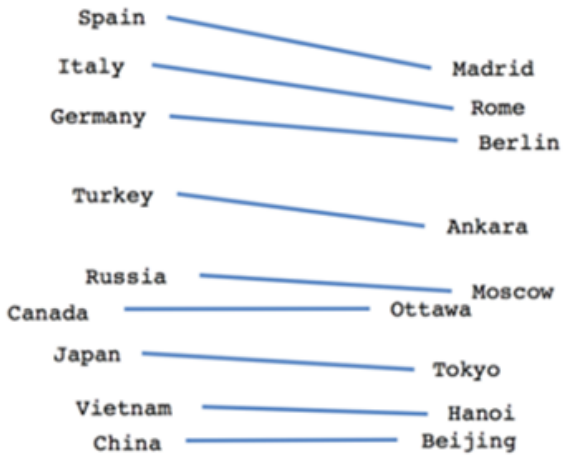
Örnek



Male-Female



Verb tense



Country-Capital

- vector[Queen] ≈ vector[King] - vector[Man] + vector[Woman]
- vector[Paris] ≈ vector[France] - vector[Italy] + vector[Rome]
  - This can be interpreted as “France is to Paris as Italy is to Rome”.

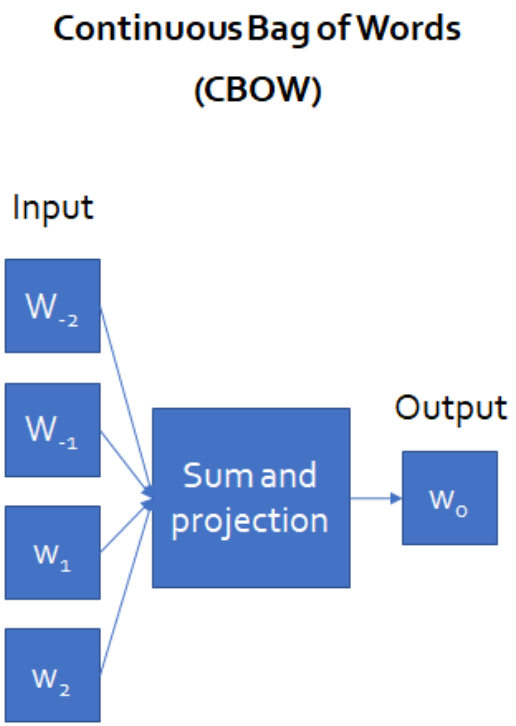
- Word2Vec, kelime gömme yöntemleri arasında en popülerleri arasındadır (Mikolov ve ark. 2013).
- Word2Vec yöntemi hızlıdır, eğitilmesi kolaydır ve kodları ile önceden eğitilmiş gömmeler çevrimiçi olarak kolayca bulunabilir.
- Saymak yerine tahmin etmeye dayalıdır.

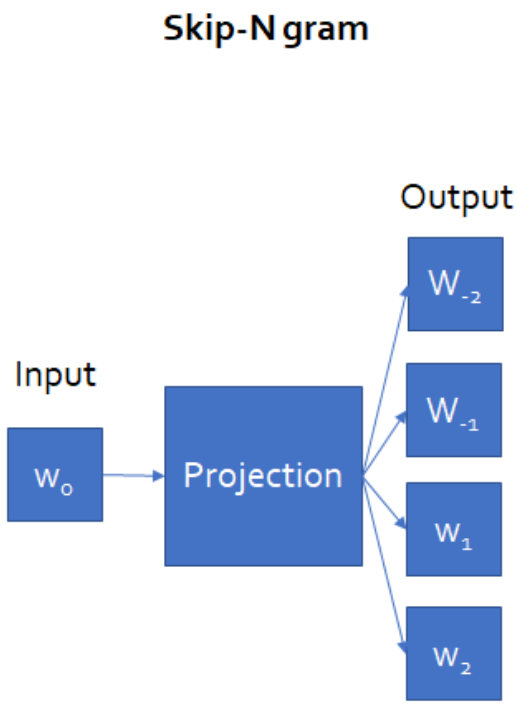
- Word2Vec, tüm belge, her merkez kelimenin etrafındaki k pozisyondaki kelimeleri kullanır.
- Bu kelimelere bağlam kelimeleri denir.
- Örnek (k = 3 için): "It was a bright cold day in April, and the clocks were striking"
  - Merkez kelime: day
  - Bağlam kelimeleri: a bright cold ve in April, and
- Word2Vec, tüm kelimeleri merkez kelime olarak ve tüm bağlam kelimelerini dikkate alır.

- Örnek: window size=2 d1 = "king brave man" , d2 = "queen beautiful women"

word	Word one hot encoding	neighbor	Neighbor one hot encoding
king	[1,0,0,0,0,0]	brave	[0,1,1,0,0,0]
		man	
brave	[0,1,0,0,0,0]	king	[1,0,1,0,0,0]
		man	
man	[0,0,1,0,0,0]	king	[1,1,0,0,0,0]
		brave	
queen	[0,0,0,1,0,0]	beautiful	[0,0,0,0,1,1]
		women	
beautiful	[0,0,0,0,1,0]	queen	[0,0,0,1,0,1]
		women	
woman	[0,0,0,0,0,1]	queen	[0,0,0,1,1,0]
		beautiful	

- **Skip-Gram Modelinin Temeli:**
  1. Hedef kelime ve ona yakın bağlam kelimesini pozitif örnekler olarak ele alınır.
  2. Sözlükteki diğer kelimelerden rastgele örnekler alarak negatif örnekler elde edilir.
  3. İki durumu ayırt etmek için lojistik regresyon kullanarak bir sınıflandırıcı ile eğitilir.
  4. Gömme olarak regresyon ağırlıkları kullanılır.


Continuous Bag of Words
(CBOW)
Input
W-2
W-1
W1
W2
Sum and projection
Output
Wo


Skip-N gram
Input
Wo
Projection
Output
W-2
W-1
W1
W2

- **Giriş** Büyük metin kümelerinden öğrenir. Her kelime, çevresindeki kelime bağlamında değerlendirilir.
- **Model**
  - CBOW (Continuous Bag of Words): Bağlamdaki kelimeler kullanılarak bir hedef kelime tahmin edilir.
  - Skip-Gram: Bir kelime verilir, bağlamındaki kelimeler tahmin edilmeye çalışılır.
- **Vektör** Her kelime için rastgele başlayan bir vektör oluşturulur.
- **Eğitim**
  - CBOW'da, bağlam kelimelerinin vektörleri toplanıp hedef kelimenin vektörüne benzetilmeye çalışılır.
  - Skip-Gram'da, hedef kelimenin vektörü ile bağlam kelimelerinin vektörleri tahmin edilir.
- **Kayıp Fonksiyonu ve Optimizasyon** Gerçek ve tahmin edilen kelimeler arasındaki fark minimize edilir.
- **Sonuç** Her kelime benzersiz bir vektörle temsil edilir.

- **Skip-Gram Modeli:**
  - **Metin Örneği:** "Kedi evde uyuyor."
  - **Vektör İnşası:** "Kedi", "evde", "uyuyor" kelimeleri için başlangıç vektörleri.
  - **Eğitim Süreci**
    - "Kedi" kelimesi verildiğinde, "evde", "uyuyor" kelimelerinin vektörlerini tahmin etmeye çalışır.
    - "Kedi" kelimesinin vektörü, bu kelimelerle ilişkilendirilerek ayarlanır.
  - **Sonuç** "Kedi" kelimesinin vektörü, metindeki bağlamına uygun şekilde ayarlanır.

- **CBOW (Continuous Bag of Words) Modeli:**
  - **Metin Örneği:** "Kedi evde uyuyor."
  - **Vektör İnşası:** "Kedi", "evde", "uyuyor" kelimeleri için başlangıç vektörleri.
  - **Eğitim Süreci:**
    - "kedi", "uyuyor" kelimelerinin vektörleri verilerek "evde" kelimesinin vektörü tahmin edilmeye çalışılır.
    - Bu bağlam kelimelerinin vektörleri toplanır ve "evde" kelimesinin vektörüne benzetilmeye çalışılır.
  - **Sonuç:** "evde" kelimesi, çevresindeki kelimelerin vektörleri ile ilişkilendirilerek anlam kazanır.

GloVe

- Stanford Üniversitesi tarafından geliştirilen bir kelime gömme yöntemi.
- Metinlerdeki kelimelerin co-occurrence istatistiklerini analiz eder.
- Kelimelerin sayısal vektörel temsillerini oluşturur.

GloVe

- Co-occurrence Matrisi: Kelime çiftlerinin bir arada geçme sıklığını içeren matris.
- Matris Faktorizasyonu: Kelimelerin semantik ilişkilerini yansıtan vektörler elde edilir.
- Optimizasyon: Kayıp fonksiyonu minimize edilerek vektörler optimize edilir.

GloVe

- Temel Yaklaşım: Word2Vec bağlam odaklı iken, GloVe genel kelime co-occurrence odaklanır.
- Eğitim Yöntemi: Word2Vec lokal bilgiyi, GloVe ise global istatistikleri kullanır.
- Sonuçlar: Word2Vec bağlam ilişkilerinde, GloVe genel kelime ilişkilerinde iyi performans gösterir.

- **Metin Verisi:** "Kedi bahçede oynuyor." ve "Köpek bahçede oynuyor."
- **Yaklaşım:** "bahçede" kelimesi verildiğinde, çevresindeki "kedi" ve "köpek" gibi kelimeleri tahmin etmeye çalışır.
- **Odak Noktası:** "bahçede" kelimesinin doğrudan bağlamına dayanarak "kedi" ve "köpek" kelimeleriyle olan ilişkisini öğrenir.
- **Sonuç:** "bahçede" kelimesinin vektörü, bu doğrudan bağlamların etkisiyle şekillenir.

- **Metin Verisi:** "Kedi bahçede oynuyor." ve "Köpek bahçede oynuyor."
- **Yaklaşım:** "kedi" ve "bahçede", "köpek" ve "bahçede" gibi kelime çiftlerinin birlikte geçme sıklığını analiz eder.
- **Odak Noktası:** "bahçede" kelimesinin hem "kedi" hem de "köpek" kelimeleriyle olan genel ilişkisini tüm metin veri setindeki co-occurrence frekanslarına dayanarak inceler.
- **Sonuç:** Kelimeler arasındaki geniş çaplı ilişkileri yansıtan vektörler elde edilir.

Fasttext

- Facebook AI Research tarafından geliştirilen bir kelime gömme aracı.
- Kelimeleri bütün olarak değil, alt diziler (n-gramlar) olarak ele alır.
- Her kelimenin n-gramları üzerinden gömme vektörleri oluşturur.

Fasttext

- N-Gram: Kelimelerin içerdiği n-gramları kullanır.
- Kelime Gömmelerinin Birleştirilmesi: N-gramların vektörlerinin toplamı veya ortalaması alınır.
- Eğitim Yaklaşımı: Eğitim, n-gramlar üzerinden gerçekleştirilir

Fasttext

- N-Gram Yaklaşımı: Fasttext, kelimeleri n-gramlarına ayırarak işler.
- Nadir kelimeler için daha etkili gömmeler üretir.
- Dilbilimsel Esneklik: Morfolojik yapının daha iyi yakalanması, özellikle eklemeli dillerde etkili.
- Hesaplama Yükü ve Hız: Ekstra hesaplama gerektirir, model boyutu ve eğitim süresi artabilir.

- **"Kedi" Kelimesi için Fasttext 3-Gram Hesaplaması**
  - **Özel İşaretleme** Kelimeye başlangıç ve bitiş işaretleri eklenir: '<kedi>'.
  - **3-Gram Alt Dizileri**
    - '<ke'
    - 'ked'
    - 'edi'
    - 'di>'
  - **Gömme Vektörlerinin Oluşturulması** Her bir 3-gram için gömme vektörleri hesaplanır ve bu vektörler birleştirilerek "kedi" kelimesinin tamamı için bir gömme vektörü üretilir.

Hazır Modeller

- Word2Vec: Google News Vectors, Yaklaşık 100 milyar kelimelik veri seti üzerinde eğitilmiş model
- Fasttext: Facebook's Pre-trained Fasttext Models: Birçok dil için hazır modeller
- Stanford's Pre-trained GloVe Models: Farklı boyutlarda ve veri setlerinde eğitilmiş modeller

cc.en.300.bin

- **"cc"**: "Common Crawl" anlamına gelir. Bu, web'den toplanan büyük ve çeşitli bir metin veri setini ifade eder.
- **"en"**: Bu modelin İngilizce diline özgü olduğunu belirtir. Fasttext, birçok dil için ayrı model sunmaktador.
- **"300"**: Modeldeki her kelimenin 300 boyutlu bir vektörle temsil edildiğini göstermektedir
- **".bin"**: Dosya formatını belirtir (binary)
- Kelime benzerlikleri, duygu analizi, metin sınıflandırması gibi çeşitli görevlerde kullanılabilir.

```python
import fasttext
import fasttext.util
import numpy as np

kelime1 = 'good'
kelime2 = 'bad'
kelime3 = 'excellent'
vektor1 = model.get_word_vector(kelime1)
vektor2 = model.get_word_vector(kelime2)
vektor3 = model.get_word_vector(kelime3)

def cos_sim(vektor1, vektor2):
    return np.dot(vektor1, vektor2) / (np.linalg.norm(vektor1) * np.linalg.norm(vektor2))

benzerlik = cos_sim(vektor1, vektor2)
print(f"'{kelime1}' ve '{kelime2}' kelimeleri arasındaki benzerlik: {benzerlik}")
benzerlik = cos_sim(vektor1, vektor3)
print(f"'{kelime1}' ve '{kelime3}' kelimeleri arasındaki benzerlik: {benzerlik}")
```

```
'good' ve 'bad' kelimeleri arasındaki benzerlik: 0.7517589330673218
'good' ve 'excellent' kelimeleri arasındaki benzerlik: 0.6737029552459717
```

```python
import fasttext
import fasttext.util
import numpy as np

kelime1 = 'tomatoes'
kelime2 = 'pepper'
kelime3 = 'eggplant'
vektor1 = model.get_word_vector(kelime1)
vektor2 = model.get_word_vector(kelime2)
vektor3 = model.get_word_vector(kelime3)

def cos_sim(vektor1, vektor2):
    return np.dot(vektor1, vektor2) / (np.linalg.norm(vektor1) * np.linalg.norm(vektor2))

benzerlik = cos_sim(vektor1, vektor2)
print(f"'{kelime1}' ve '{kelime2}' kelimeleri arasındaki benzerlik: {benzerlik}")
benzerlik = cos_sim(vektor1, vektor3)
print(f"'{kelime1}' ve '{kelime3}' kelimeleri arasındaki benzerlik: {benzerlik}")
benzerlik = cos_sim(vektor2, vektor3)
print(f"'{kelime2}' ve '{kelime3}' kelimeleri arasındaki benzerlik: {benzerlik}")
```

```
'tomatoes' ve 'pepper' kelimeleri arasındaki benzerlik: 0.49019330739974976
'tomatoes' ve 'eggplant' kelimeleri arasındaki benzerlik: 0.6276549696922302
'pepper' ve 'eggplant' kelimeleri arasındaki benzerlik: 0.4802670478820801
```