WEEK 5: TEXT MINING 3

SECU0057

BENNETT KLEINBERG

13 FEB 2020

Applied Data Science

WEEK 5: TEXT MINING 3

TODAY

- text similarity
- intro to word embeddings

What makes two strings similar?

FACTORS IN SIMILARITY

- lexical similarity
- phonetic similarity
- semantic similarity

Underlying challenge: **quantification**

STRING SIMILARITY

- Levenshtein distance
- Damerau-Levenshtein distance
- q-gram distances

Aim: numerical measure of distance

```
string_a = 'crime'
string_b = 'criminal'
```

How much do these two differ?

LEVENSHTEIN'S APPROACH

Distance between A and B:

1. change A to obtain B
2. allowed edits:
    - substituting characters
    - adding characters

LEV(‘CRIME’, ‘CRIMINAL’)

1. crim$_{e}$**i**
2. crimi**n**
3. crimin**a**
4. crimina**l**

4 edits are required to change ‘crime’ into ‘criminal’.

IN R

```r
library(stringdist)

stringdist(a = 'crime'
           , b = 'criminal'
           , method = 'lv')
```

```
## [1] 4
```

$$Lev_{(crime,criminal)} = 4$$

YOUR TURN

Pair 1:

- A = “london”
- B = “amsterdam”

Pair 2:

- C = “london”
- D = “condom”

```
stringdist(a = 'london'
           , b = 'amsterdam'
           , method = 'lv')
```

```
## [1] 8
```

```
stringdist(a = 'london'
           , b = 'condom'
           , method = 'lv')
```

```
## [1] 2
```

CONVERTING DISTANCES TO SIMILARITY

1. calculate distance $dist$ (e.g. $Lev_{(crime,criminal)} = 4$)
2. calculate maximum distance $\max dist$
3. obtain quotient of $\frac{dist}{\max dist}$
4. substract quotion from 1: $sim = 1 - \frac{dist}{\max dist}$

DISTANCE TO SIMILARITY

1. $Lev_{(crime,criminal)} = 4$
2. $\max Lev_{(crime,criminal)} = 8$
3. $\frac{Lev_{(crime,criminal)}}{\max Lev_{(crime,criminal)}} = \frac{4}{8}$
4. $1 - \frac{Lev_{(crime,criminal)}}{\max Lev_{(crime,criminal)}}$

$$1 - \frac{4}{8} = 0.5$$

USING `STRINGSIM(...)`

```
stringsim(a = 'crime'
          , b = 'criminal'
          , method = 'lv')
```

```
## [1] 0.5
```

Q-GRAMS

- essentially: n-grams on character-level

```
qgrams('crime', 'criminal', q = 2)
```

```
##     cr ri im me mi na in al
## V1   1  1  1  1  0  0  0  0
## V2   1  1  1  0  1  1  1  1
```

SETTING Q

Note that q is anaolgous to n in n-grams:

```
qgrams('crime', 'criminal', q = 3)
```

```
##     cri rim ime imi min nal ina
## V1    1   1   1   0   0   0   0
## V2    1   1   0   1   1   1   1
```

Q-GRAMS FOR STRING DISTANCE

Simple approach: absolute difference

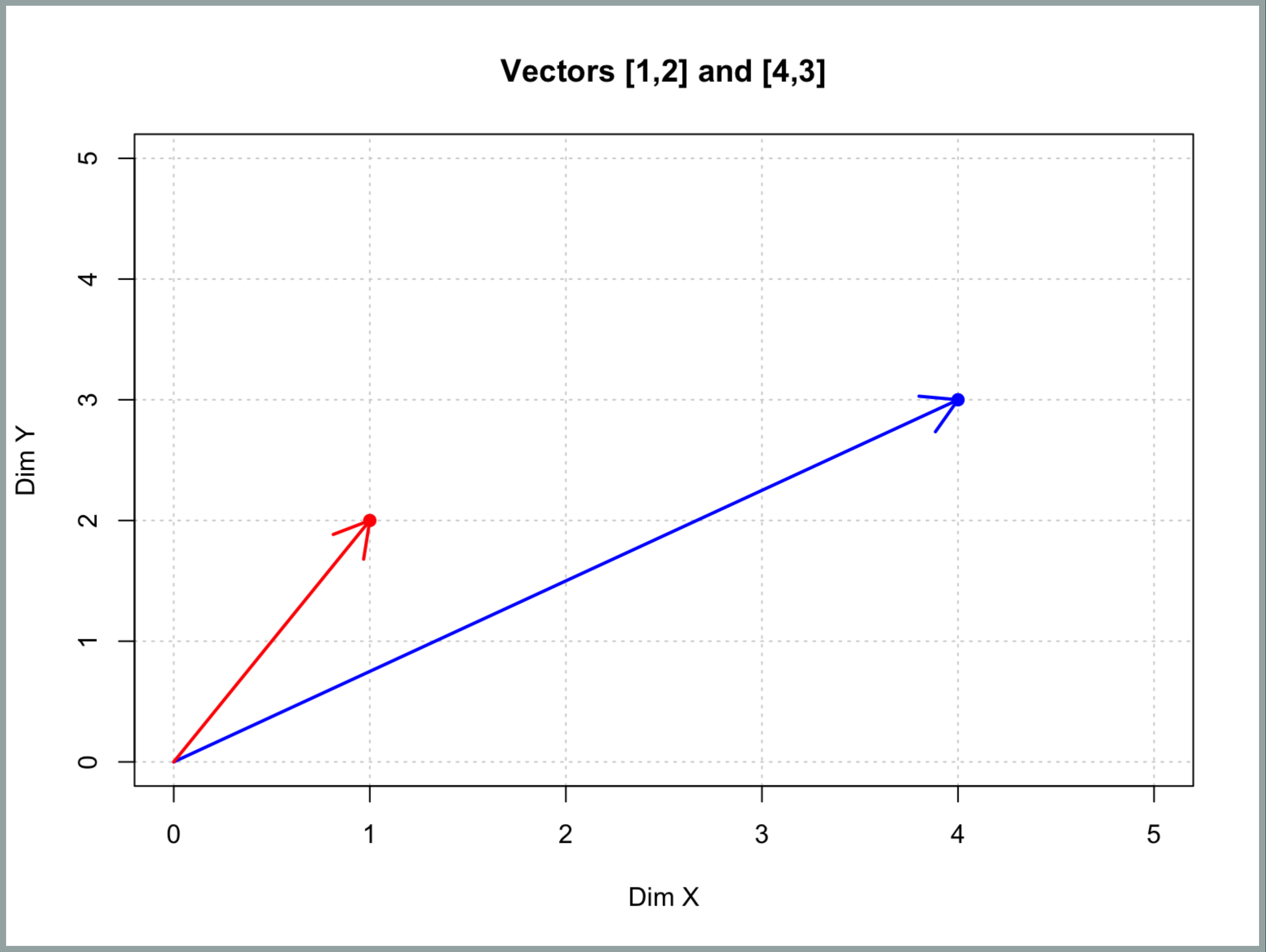
```
stringdist(a = 'crime'
           , b = 'criminal'
           , method = 'qgram'
           , q = 3)
```

```
## [1] 5
```

DISTANCES BETWEEN VECTORS

Suppose we have got two vectors:

- $\overrightarrow{v_1} = [1, 2]$
- $\overrightarrow{v_2} = [4, 3]$



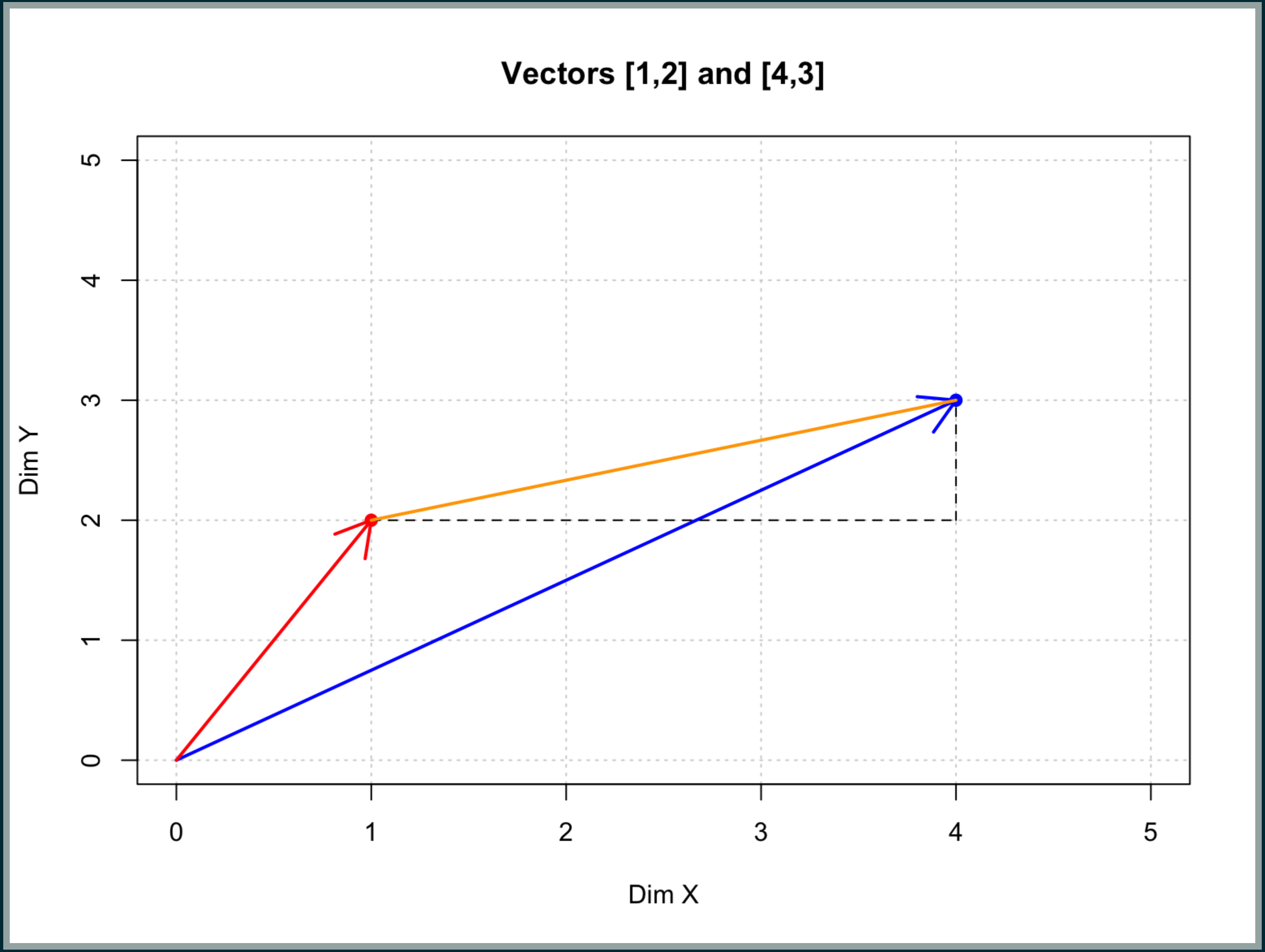
Vectors [1,2] and [4,3]

EUCLIDEAN DISTANCE

Uses Pytharogean theorem.

For two two-dimensional locations:

- build a right triangle
- use $a^2 = b^2 + c^2$ to calculate the length of the hypothenuse c



Vectors [1,2] and [4,3]

BY HAND:

- $a = x_2 - x_1 = 4 - 1 = 3$
- $b = y_2 - y_1 = 3 - 2 = 1$
- $c^2 = a^2 + b^2$

Thus:

$$c = \sqrt{a^2 + b^2} = \sqrt{9 + 1} = 3.16$$

EUCLIDEAN DISTANCE IN 3 DIMENSIONS

$$dist(A, B) = \sqrt{(A_1 - B_1)^2 + (A_2 - B_2)^2 + (A_3 - B_3)^2}$$

ON THE Q-GRAMS

```r
qgrams('london', 'condom', q = 2)
```

```
##    lo on nd om do co
## V1  1  2  1  0  1  0
## V2  0  1  1  1  1  1
```

VECTORS V1 AND V2

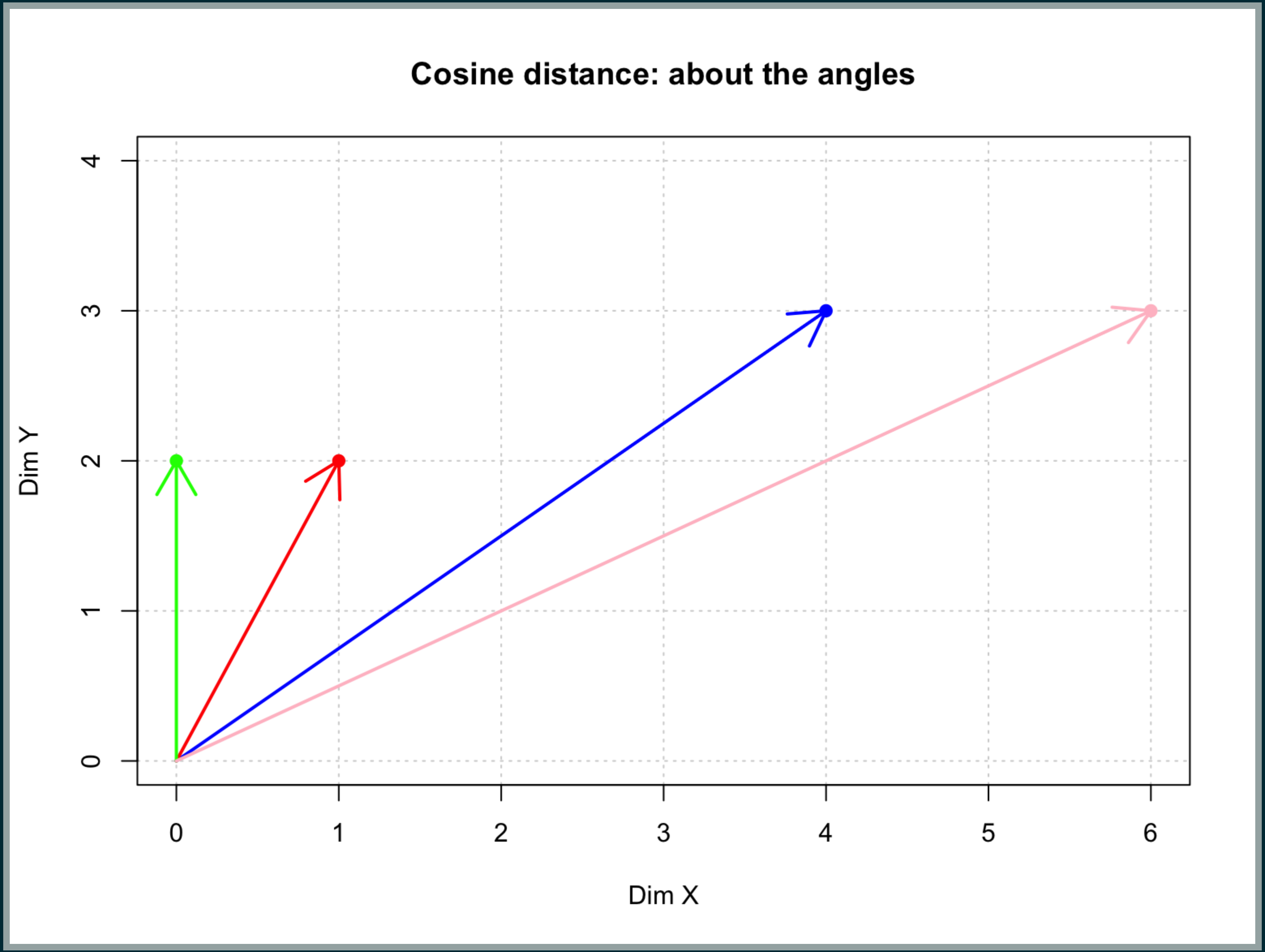
```
matrix_1 = matrix(data = c(c(1,2,1,0,1,0),c(0,1,1,1,1,1))
                  , nrow = 2, byrow = T)

dist(matrix_1, method = 'euclidean')
```

```
##   1
## 2 2
```

EUCLIDEAN DISTANCE AND MAGNITUDE

- takes into account the magnitude of vectors
- but this is not always meaningful
- different metric: *cosine* distance

COSINE DISTANCE



COSINE SIMILARITY

$$csim = \frac{A \times B}{\sqrt{A \times A} * \sqrt{B \times B}}$$

_Note: $A \times B$ is the dot product of the vector.

IN R

```r
V1 = c(4,2,3)
V2 = c(1,3,1)

cossim = function(A, B){
  numerator = sum(A*B)
  denominator = sqrt(sum(A*A))*sqrt(sum(B*B))
  cosine_sim = numerator/denominator
  return(cosine_sim)
}

cossim(V1, V2)
```

```
## [1] 0.7278603
```

COSINE SIMILARITY ON Q-GRAMS

```
qgrams('london', 'condom', q = 2)
```

```
##    lo on nd om do co
## V1  1  2  1  0  1  0
## V2  0  1  1  1  1  1
```

```
stringsim('london', 'condom', q = 2, method = 'cosine')
```

```
## [1] 0.6761234
```

SIMILARITY OF PHRASES

```r
phrase_1 = "I think numbers are great"
phrase_2 = "Numbers are useless"

stringsim(phrase_1, phrase_2, q=3, method = 'cosine')
```

```
## [1] 0.4551496
```

PROBLEMS WITH LEXICAL SIMILARITY?

CONSIDER THESE

```
[sand, beach]
[water, sea]
[bank, money]
[euro, dollar]
```

WHAT ABOUT THE SEMANTIC DIMENSION?

- lexical dimension doable
- But what about the meaning of words?
- “sand” and “beach” are close to one another
- … but lexical metrics fail to uncover this

SEMANTIC SIMILARITY

If a method can do this, then we expect:

- $cossim(sand, beach) > cossim(bleach, beach)$
- $cossim(euro, dollar) > cossim(neuro, euro)$

WORD EMBEDDINGS

- meaning of words determined by information associated with it
- computationally: we express each word as a vector
- that vector contains the information associated with the word

Does that work? When?

VECTORS ARE EVERYWHERE

“these are word embeddings”

```
## Warning: 'as.data.frame.dfm' is deprecated.
## Use 'convert(x, to "data.frame")' instead.
## See help("Deprecated")
```

document	these	are	word	embeddings
text1	1	0	0	0
text2	0	1	0	0
text3	0	0	1	0
text4	0	0	0	1

Every word is a vector (called: **a one-hot vector**)

EVERY DOCUMENT IS A VECTOR

```
## Warning: 'as.data.frame.dfm' is deprecated.
## Use 'convert(x, to "data.frame")' instead.
## See help("Deprecated")
```

document	career	upon	which	i	am	about	to	enter	appear	f
1825-Adams	2	16	27	15	4	1	101	1	1	
1829-Jackson	0	0	12	12	0	1	53	0	1	
1833-Jackson	0	5	14	6	2	1	46	0	0	

VECTOR-BASED WORD SIMILARITY

So can’t we just calculate the vector similarity then? (e.g. cosine similarity)

Ideas?

PROBLEM OF SPARSITY

Remember: dot products!

Dot product of two one-hot vector is always zero.

EMBEDDING A VECTOR

- we want a denser representation of each word
- two approaches:
  - word2vec: $P(word|context)$ and $P(context|word)$
  - glove: corpus co-occurrences
  - Details available here

EMBEDDINGS: AIM

From high-dimensional vector space to lower dimensional space.

E.g. from a one-hot vector of size 10,000 to an embeddeding of size 300.

PROPERTIES OF WORD EMBEDDINGS

- we have an embedding of each word
- that captures some kind of “context”
- each embedding in a given model has the same length

USING WORD EMBEDDINGS

- no need to build the embeddings
- typically we use pre-trained models (e.g. from Google, Facebook)
- e.g. Word vectors for 157 languages and the Glove embeddings
- we can then retrieve the embeddings for a given word

IN R

Note: heavy on your RAM (loads massive file into memory)

1. initialising pre-trained models
2. calculating similarity between terms

Code: https://github.com/ben-aaron188/r_helper_functions/blob/master/init_glove.R

OUR EXPECTATIONS (1)

$$cossim(sand, beach) > cossim(bleach, beach)$$

```
sand_emb = as.vector(glove.pt[row.names(glove.pt) == 'sand', ])
beach_emb = as.vector(glove.pt[row.names(glove.pt) == 'beach', ])
bleach_emb = as.vector(glove.pt[row.names(glove.pt) == 'bleach', ])
```

```
cossim(sand_emb, beach_emb)

#[1] 0.5469368
```

```
cossim(beach_emb, bleach_emb)

#[1] -0.0501422
```

OUR EXPECTATIONS (2)

$$cossim(euro, dollar) > cossim(neuro, euro)$$

```r
euro_emb = as.vector(glove.pt[row.names(glove.pt) == 'euro', ])
dollar_emb = as.vector(glove.pt[row.names(glove.pt) == 'dollar', ])
neuro_emb = as.vector(glove.pt[row.names(glove.pt) == 'neuro', ])
```

```r
cossim(euro_emb, dollar_emb)
#[1] 0.7328042

cossim(euro_emb, neuro_emb)
#[1] -0.08723018
```

ARITHMETICS WITH WORD EMBEDDINGS

What if we could do maths with meaning?

Do semantic relationships hold true in embeddings?

BERLIN - GERMANY + FRANCE

If: $\overrightarrow{BERLIN} \approx BERLIN$

$$\frac{\overrightarrow{GERMANY}}{\overrightarrow{BERLIN}} :: \frac{\overrightarrow{FRANCE}}{\vec{?}}$$

```
berlin = as.vector(glove.pt[row.names(glove.pt) == 'berlin', ])
germany = as.vector(glove.pt[row.names(glove.pt) == 'germany', ])
france = as.vector(glove.pt[row.names(glove.pt) == 'france', ])
```

```
mystery_1 = berlin - germany + france

  [1]  0.5129700 -0.7331210 -0.1134250  0.4532580  0.5920800  0.5046900
  [7]  0.1716500  0.6980830  0.0461335  0.2364360 -0.2775940 -0.0869580
 [13] -0.2696400  0.1602300 -0.9382800  0.3996800  0.1305250 -0.6906400
 [19] -0.6324500 -0.3029900  0.2935000  0.1269000 -0.1562700 -1.1325400
 [25]  ...
```

CLOSTEST NEIGHBOURS TO THE MYSTERY VECTOR?

```
head(sort(cos_sim_vals[,1], decreasing = TRUE), 10)

 mystery_1      paris     france     french prohertrib     berlin    bruss
 1.0000000  0.8827144  0.7558026  0.7075165  0.6943174  0.6665562  0.657
      lyon     london         le
 0.6526201  0.6407975  0.6403628
```

KING - MAN + WOMAN

$$\frac{\overrightarrow{MAN}}{\overrightarrow{KING}} :: \frac{\overrightarrow{WOMAN}}{\vec{?}}$$

```
head(sort(cos_sim_vals[,2], decreasing = TRUE), 10)

  mystery_2      king     queen   monarch    throne  daughter    prince
1.0000000 0.8551837 0.7834414 0.6933802 0.6833110 0.6809082 0.6713142 0.6
   mother elizabeth
0.6579325 0.6563301
```

WHY DOES THIS WORK?

Possible explanation: **the distributional hypothesis** (Harris, 1954)

Words in that occur in the same context have the same meaning.

CAUTIONARY NOTE

Word embeddings are very powerful, but:

- instable
- numerically meaningless
- Limitations of word embeddings, Burdick (2019)

(see the required reading for today)

WHAT'S NEXT?

- Today's tutorial: text similarity, word embeddings
- Homework: more word embeddings, custom functions

Next week: reading week (then: Machine Learning 1)