Natural Language Processing (NLP)

Lecture 3

Word Embedding

Word Forms

- **Lemma Form**: root word after removing any suffixes

Ex: book, sing, play,….

- **Word form:** word with any additions

Ex: booking, singer, played,….

Word Relations

- **Homographs:** same typing, different meaning

Ex: bank, book, drive, like,……

- **Homophones** : same pronouncing with different meanings

Cause problems in voice to text applications

Ex: piece & peace, write & right ,….

Word relations

- **Synonyms:** Replaceable words in **most** of contexts
  Different correlations according to the context

Ex: Automobile/car, big/large, carbon dioxide/co2,….

car/vehicle➔ low correlation ……..(not all contexts can be replaced)

I bought a ***car*** (**ok**)

I bought a ***vehicle*** (**No**)

convince/persuade ➔ high correlation ……(can be used interchangeably)

I ***convinced*** him to buy a car

I ***persuaded*** him to buy a car

- **Antonyms** : opposite meaning words from specific perspective

(age) big **X** young

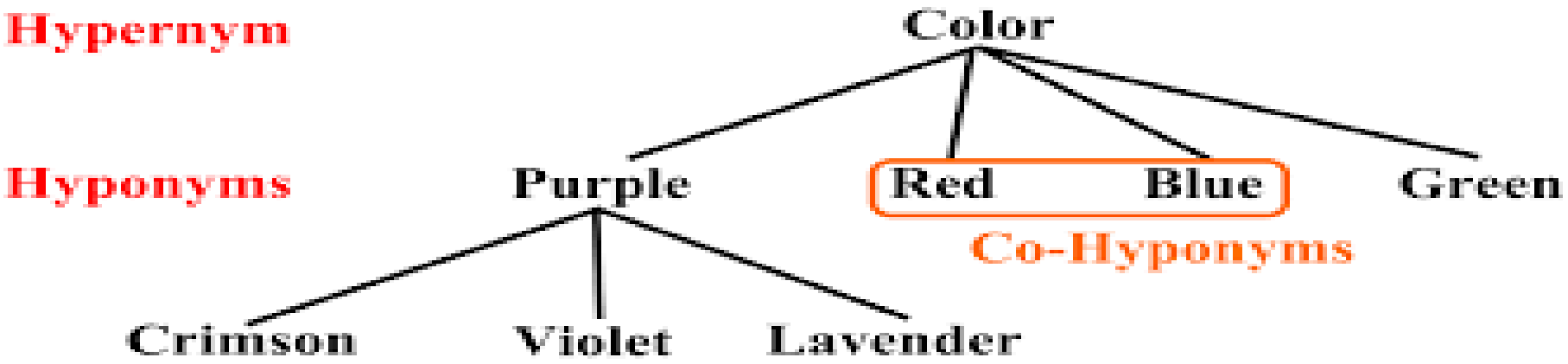
my big sister..My young sister

(size) big **X** small

my big purse….my small purse

Word relations

- **Hypernym:** Parents
  Ex: Vehicles, Vegetables
- **Hyponym :** Childs
  Ex: cars, cucumber



Determining the relations according to the words meaning are used widely in lexicons building, sentimental analysis, chatbot applications and so on ..

Zeugma Test

- It is used for testing the meaning
- Can be defined as Using a word to govern two or more words though appropriate to only one.

Ex: I run in the track daily

I run my business

I run my business in the track daily. **Different meanings**

Ex: I write my diaries

I write a message

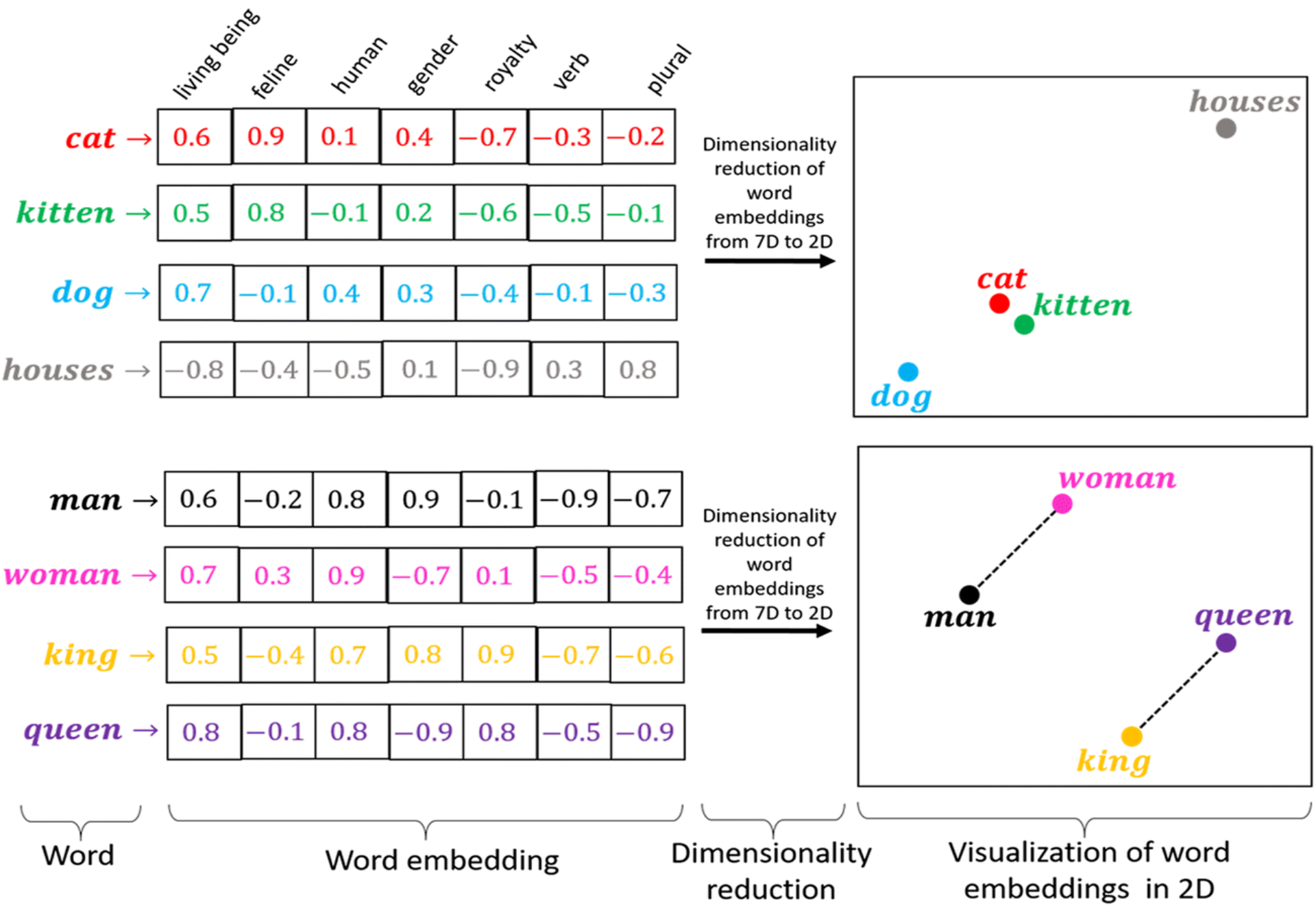
I write my diaries and a message. **Same meaning**

Word embedding/Word vector

- A numeric vector input that represents a word in a lower-dimensional space.
- It allows words with similar/approximated meaning to have a similar representation/encoding.
- A word vector with 50 values can represent 50 unique features.

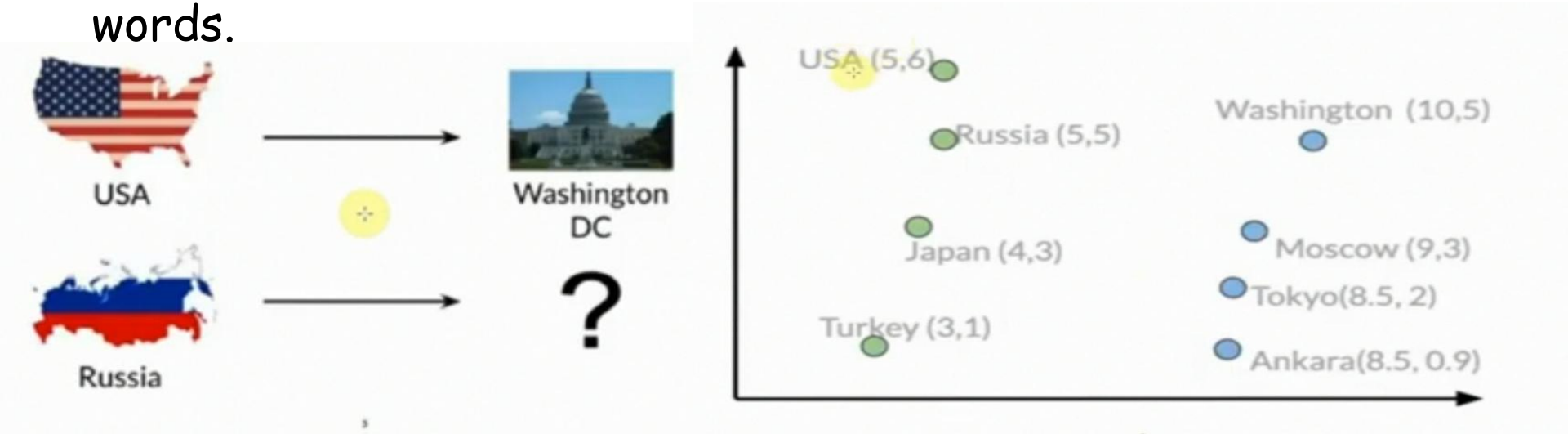
- **Features:** Anything that relates words to one another.

Word embedding Ex



Text vectors

- Mathematical operations applied based on the word embedding of the words to obtain requested words based on other existing words.



- Suppose each country/city name is represented by only two features (x,y)
- So: Washington - USA- = (10,5)- (5,6) = (5,-1)
- To obtain the most correlated word based on the text vector operations for Russia (5,5) ➔ ? (5,5) + (5,-1) = (10,4)➔(closest) "Moscow"
- What is applied on two features, applied on any number of features

Word Analogy

How words are close or distant from each other

- Man - Woman ➔ King – Queen
- Cairo – Egypt ➔ London - England
- King – Queen ➔ Prince – Princess
- Japan – Japanese ➔ China – Chinese
- Eat – Eating ➔ Run - Running

Mathematics on vectors

- Use Scipy lib and spatial functions

```
king = nlp.vocab['king'].vector
man = nlp.vocab['man'].vector
woman = nlp.vocab['woman'].vector

# Now we find the closest vector in the vocabulary to the result of "man" - "woman" + "queen"
new_vector = king - man + woman
computed_similarities = []

words = ['cat','apple','queen','castle','sea','shell','orange','phone'
         ,'angry','book','white','land','study','crown','prince','dog',
         'great','princess','elizabeth','wow','eat','dead','horrible']

for word in words:
    similarity = cosine_similarity(new_vector,nlp.vocab[word].vector)
    computed_similarities.append((word, similarity))

computed_similarities = sorted(computed_similarities, key=lambda item: -item[1])

for a,b in computed_similarities[:10] :
    print(f'Word {a} , has similarity {b}')
```

```
Word queen , has similarity 0.7880843877792358
Word prince , has similarity 0.6401076912879944
Word princess , has similarity 0.6125636100769043
Word elizabeth , has similarity 0.4564860165119171
Word crown , has similarity 0.4247634112834930 4
Word castle , has similarity 0.3814162015914917
Word white , has similarity 0.3274742960929870 6
Word angry , has similarity 0.2994381785392761
Word sea , has similarity 0.2897905111312866
Word cat , has similarity 0.2748688161373138 4
```

Remember in Lab to ..

1. Use Spacy library and en_core_web_lg file to calculate the word embedding of given words.

2. Use word embedding to get the similarity between two words

```
In [1]: nlp(u'man').similarity(nlp(u'woman'))
Out[1]: 0.7401743668099329

In [2]: nlp(u'moon').similarity(nlp(u'woman'))
Out[2]: 0.3194349100251551

In [3]: nlp(u'love').similarity(nlp(u'hate'))
Out[3]: 0.6393099852963738
```

3. Use Scipy lib to perform the mathematical operations on vectors

Word embedding key features

1. **Dimensionality Reduction**:

   Reducing the high-dimensional space of vocabulary (which could be millions of words) into dense vectors in a lower-dimensional space (e.g., 100, 300 dimensions) makes computations more efficient.

2. **Semantic Similarity**:

   In the vector space, words that are contextually or semantically similar (e.g., "king" and "queen") tend to have similar vectors, often close to each other in terms of cosine similarity or Euclidean distance.

3. **Contextual Information**:

   Some of recent embeddings are context-aware introducing different embeddings for the same word depending on the context (BERT).

Word embedding types

- **Prediction word embeddings:**
  1. **Word2Vec**
  2. **GloVe (Global Vectors for Word Representation):**
  3. **FastText**

- **Contextual Embeddings:**
  1. **ELMO**
  2. **BERT**
  3. **GPT**

Prediction word embeddings

1. Word2Vec

- Developed by Google
- 2 Models: CBOW and Skip-gram
- CBOW: predicts the target word from its context.
- Skip-gram: Learns word vectors by predicting surrounding words (context) from the given target word.

Prediction word embeddings

2. GloVe (Global Vectors for Word Representation)

- Created by Stanford University teams.
- Uses a matrix factorization approach based on word co-occurrence statistics.
- Captures global relationships between words rather than just local context.

Prediction word embeddings

3. Fasttext

- Developed by Facebook
- Extends Word2Vec by representing words as a bag of character n-grams.
- Handles rare words or morphologically rich languages, improving the embedding for unseen or misspelled words.

4. Contextual embeddings

- **ELMo (Embeddings from Language Models)**:
  Provides word representations that change depending on the context in which they are used.

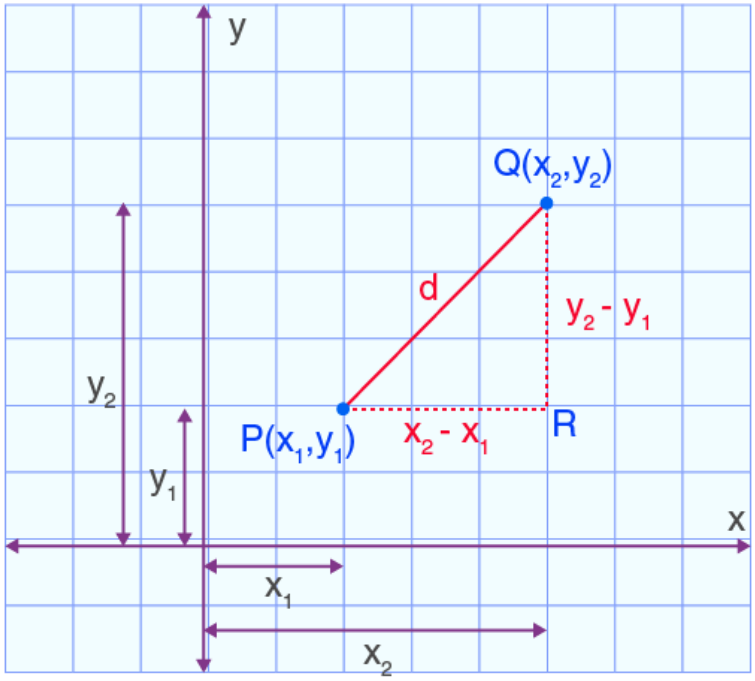
- **BERT (Bidirectional Encoder Representations from Transformers):**
  A deep learning model that produces contextual embeddings for words by considering both left and right contexts.

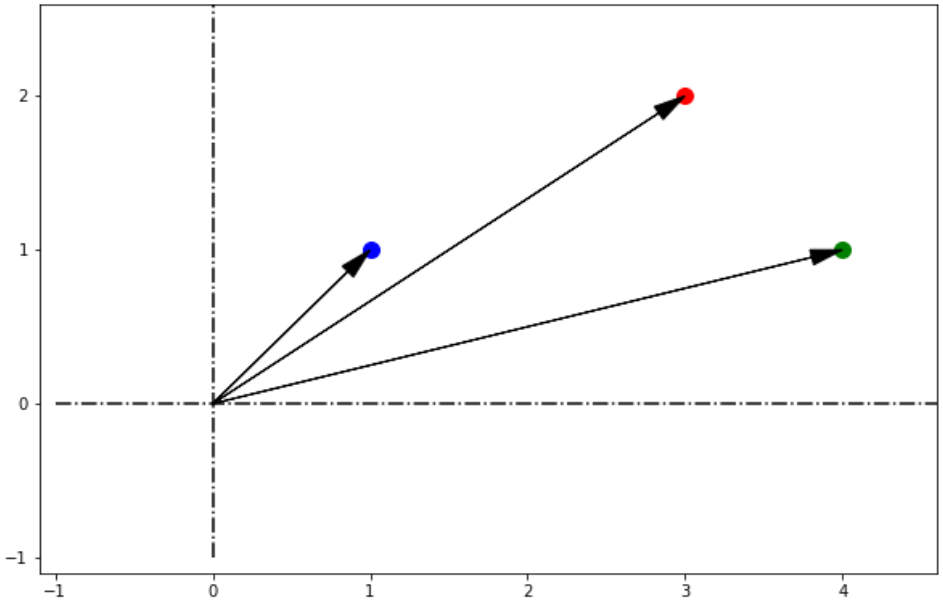
- **GPT (Generative Pre-trained Transformer):**
  A transformer-based model that generates contextual embeddings by predicting the next word in a sentence.

Similarity Calculations

1. Euclidean Distance = distance between two points in 2D space

$$\text{SQRT } (x_2-x_1)^2 + (y_2-y_1)^2$$





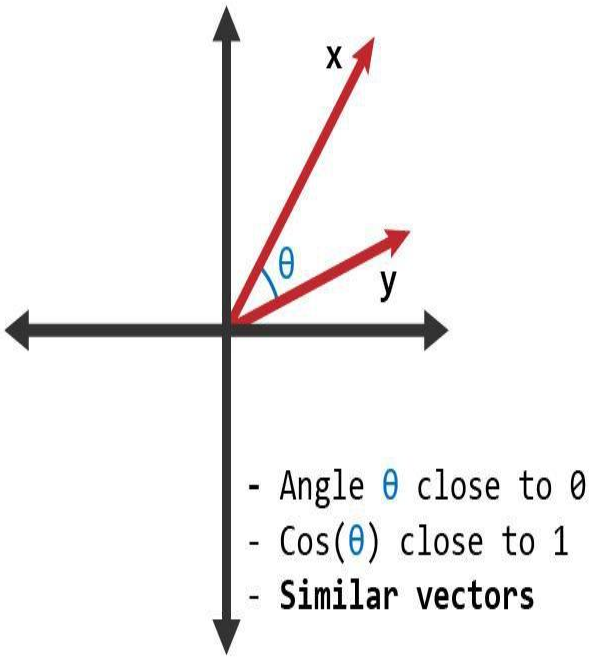
Not very accurate as it may consider closer meaning words as farther entities due to the longer distance

Similarity Calculations

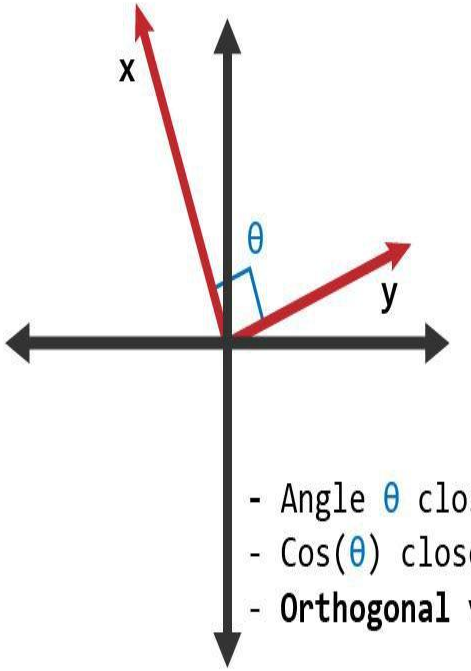
2. Cosine Similarity: A measure of similarity between two vectors, each vector is defined as a sequence of numbers

Cos 0 = 1
Cos 90 = 0
Cos 180 = -1
Cos 270 = 0
Cos 360 = 0

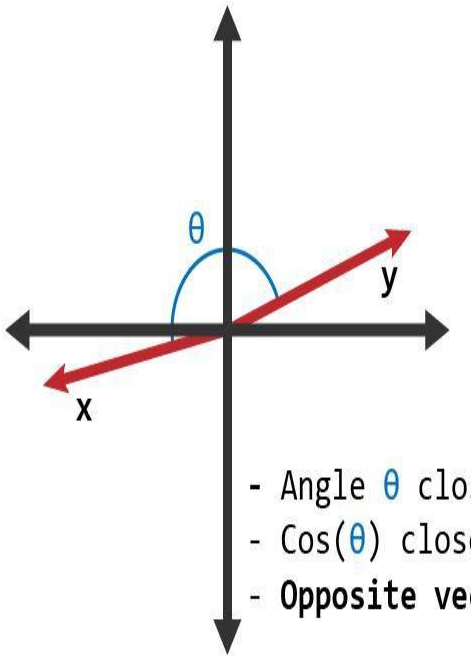
$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$



- Angle θ close to 0
- Cos(θ) close to 1
- **Similar vectors**



- Angle θ close to 90
- Cos(θ) close to 0
- **Orthogonal vectors**



- Angle θ close to 180
- Cos(θ) close to -1
- **Opposite vectors**

THANK YOU … ☺