Linguistic features

NATURAL LANGUAGE PROCESSING WITH SPACY

Azadeh Mobasher
Principal Data Scientist

POS tagging

- POS tags depend on the **context**, surrounding words and their tags

```python
import spacy
nlp = spacy.load("en_core_web_sm")
text = "My cat will fish for a fish tomorrow in a fishy way."
print([(token.text, token.pos_, spacy.explain(token.pos_))
        for token in nlp(text)])
```

```
>>> [('My', 'PRON', 'pronoun'), ('cat', 'NOUN', 'noun'), ('will', 'AUX', 'auxiliary'),
('fish', 'VERB', 'verb'), ('for', 'ADP', 'adposition'), ('a', 'DET', 'determiner'),
('fish', 'NOUN', 'noun'), ('tomorrow', 'NOUN', 'noun'), ('in', 'ADP', 'adposition'),
('a', 'DET', 'determiner'), ('fishy', 'ADJ', 'adjective'), ('way', 'NOUN', 'noun'),
('.', 'PUNCT', 'punctuation')]
```

What is the importance of POS?

- Better accuracy for many NLP tasks

```
I will fish tomorrow.
I ate fish.
```

- Translation system use case

```
verb -> pescaré
noun -> pescado
```

What is the importance of POS?

- **Word-sense disambiguation** (WSD) is the problem of deciding in which **sense** a word is used in a sentence.
- Determining the sense of the word can be crucial in machine translation, etc.

Word	POS tag	Description
Play	VERB	engage in activity for enjoyment and recreation
Play	NOUN	a dramatic work for the stage or to be broadcast

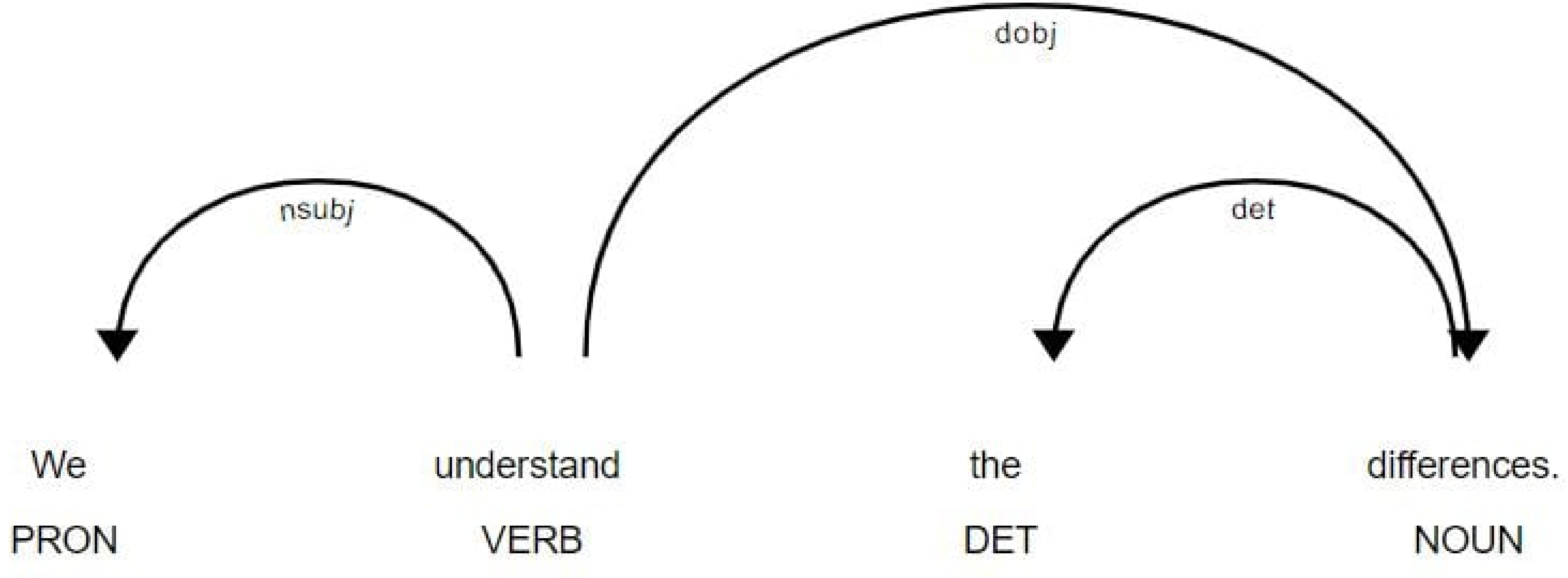
Word-sense disambiguation

```python
import spacy
nlp = spacy.load("en_core_web_sm")

verb_text = "I will fish tomorrow."
noun_text = "I ate fish."

print([(token.text, token.pos_) for token in nlp(verb_text) if "fish" in token.text], "\n")
print([(token.text, token.pos_) for token in nlp(noun_text) if "fish" in token.text])
```

```
[('fish', 'VERB', 'verb')]
[('fish', 'NOUN', 'noun')]
```

Dependency parsing

- Explores a sentence syntax
- Links between two tokens
- Results in a tree



Dependency parsing and spaCy

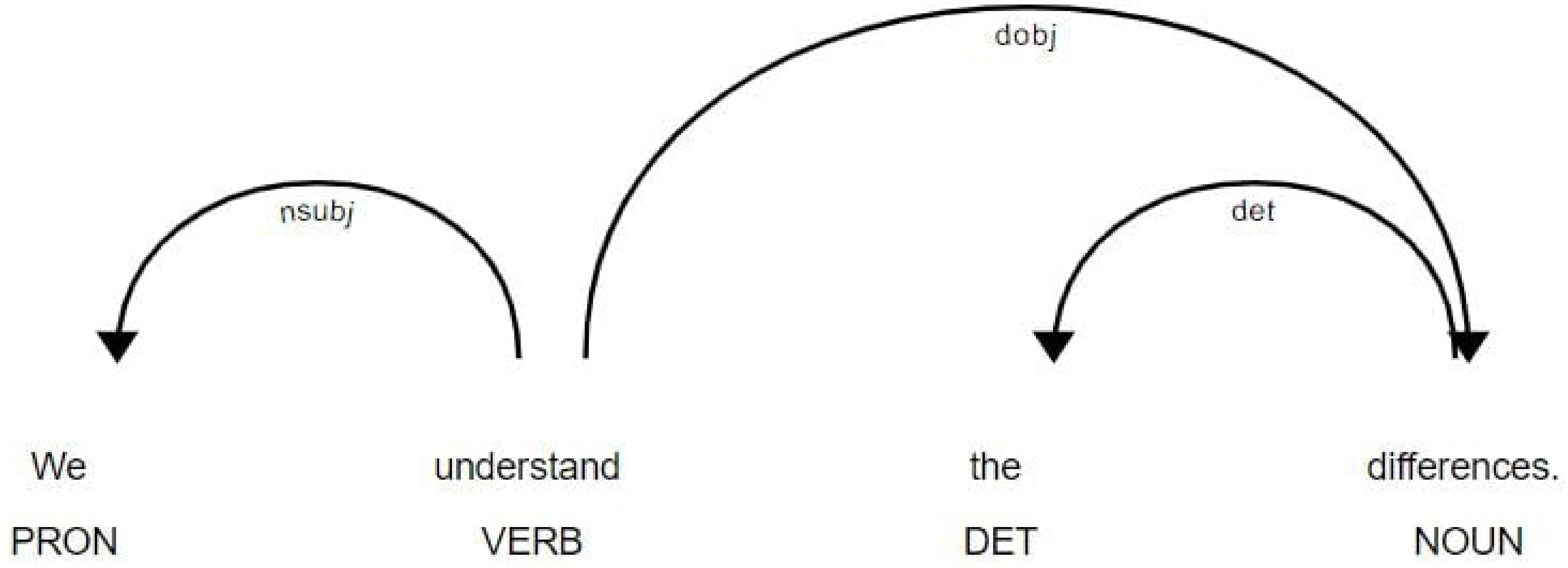
- **Dependency label** describes the type of syntactic relation between two tokens

Dependency label	Description
nsubj	Nominal subject
root	Root
det	Determiner
dobj	Direct object
aux	Auxiliary

Dependency parsing and displaCy

- `displaCy` can draw dependency trees

```
doc = nlp("We understand the differences.")

spacy.displacy.serve(doc, style="dep")
```



Dependency parsing and spaCy

- `.dep_` attribute to access the dependency label of a token

```
doc = nlp("We understand the differences.")
print([(token.text, token.dep_, spacy.explain(token.dep_)) for token in doc])
```

```
[('We', 'nsubj', 'nominal subject'), ('understand', 'ROOT', 'root'),
('the', 'det', 'determiner'), ('differences', 'dobj', 'direct object'),
('.', 'punct', 'punctuation')]
```

Let's practice!

NATURAL LANGUAGE PROCESSING WITH SPACY

Introduction to word vectors

NATURAL LANGUAGE PROCESSING WITH SPACY

Azadeh Mobasher
Principal Data Scientist

Word vectors (embeddings)

- Numerical representations of words
- Bag of words method: `{"I": 1, "got": 2, ...}`

- Older methods do not allow to understand the **meaning**:

Sentences	I	got	covid	coronavirus
I got covid	1	2	3	
I got coronavirus	1	2		4

Word vectors

- A **pre-defined number of dimensions**
- Considers **word frequencies** and the **presence of other words** in **similar contexts**

	living being	feline	human	gender	royalty	verb	plural
cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8

Word vectors

- **Multiple approaches** to produce word vectors:
  - word2vec, Glove, fastText and transformer-based architectures
- An example of a word vector:

```
array([ 2.2407   ,  1.0389   ,  1.3092   , -1.7335   , -0.78466  ,
       -0.29269  , -1.8059   , -2.5223   ,  0.78025  ,  2.4899   ,
       -0.091849 ,  0.28755  , -1.5057   ,  2.6337   ,  2.5252   ,
       -0.22432  , -2.2068   , -0.57895  , -0.56551  , -1.9338   ,
        1.4973   ,  0.85889  ,  3.3559   , -3.7527   ,  0.22585  ,
       -0.16969  ,  0.51389  ,  0.46073  , -0.28248  , -2.6048   ,
       -3.5896   , -1.0971   , -1.5517   , -0.12185  ,  2.8633   ,
       -1.2525   , -1.6924   , -2.2917   ,  0.97793  ,  0.46954  ,
       -3.595    , -0.17357  ,  0.9805   , -1.8044   , -0.72183  ,
       -0.40709  , -3.0943   ,  0.13095  , -2.9015   ,  1.4768   ,
       -1.0588   , -2.8123   ,  1.2936   , -0.0075977,  2.9975   ,
       -2.4438   ,  0.12348  ,  1.8322   ,  0.35869  , -0.018335 ,
        1.9534   ,  1.4417   ,  0.99895  , -2.8209   , -0.75846  ,
       -1.8438   , -3.2658   , -0.46574  ,  0.90322  ,  0.79868  ,
       -1.6134   , -0.33082  ,  1.1541   , -4.7334   ,  1.4964   ,
       -2.4014   , -1.3461   , -0.95551  ,  0.29671  , -1.4506   ,
       -0.87128  , -3.0714   ,  1.3597   , -0.038133 ,  1.6414   ,
       -0.90879  ,  2.7406   ,  2.2951   , -3.1423   , -3.7525   ,
        0.74033  ,  1.4921   ,  0.47422  , -1.8337   , -1.8168   ,
        0.66901  , -1.3612   , -2.2729   , -1.7656   , -0.73968  ],
      dtype=float32)
```

spaCy vocabulary

- A part of many spaCy models.
- `en_core_web_md` has **300**-dimensional vectors for **20,000** words.

```
import spacy
nlp = spacy.load("en_core_web_md")
print(nlp.meta["vectors"])
```

```
>>> {'width': 300, 'vectors': 20000, 'keys': 514157,
'name': 'en_vectors', 'mode': 'default'}
```

Word vectors in spaCy

- `nlp.vocab` : to access vocabulary (`Vocab` class)
- `nlp.vocab.strings` : to access word IDs in a vocabulary

```
import spacy
nlp = spacy.load("en_core_web_md")
like_id = nlp.vocab.strings["like"]
print(like_id)
```

```
>>> 18194338103975822726
```

- `.vocab.vectors` : to access words vectors of a model or a word, given its corresponding ID

```
print(nlp.vocab.vectors[like_id])
```

```
>>> array([-2.3334e+00, -1.3695e+00, -1.1330e+00, -6.8461e-01, ...])
```

Let's practice!

NATURAL LANGUAGE PROCESSING WITH SPACY

Word vectors and spaCy

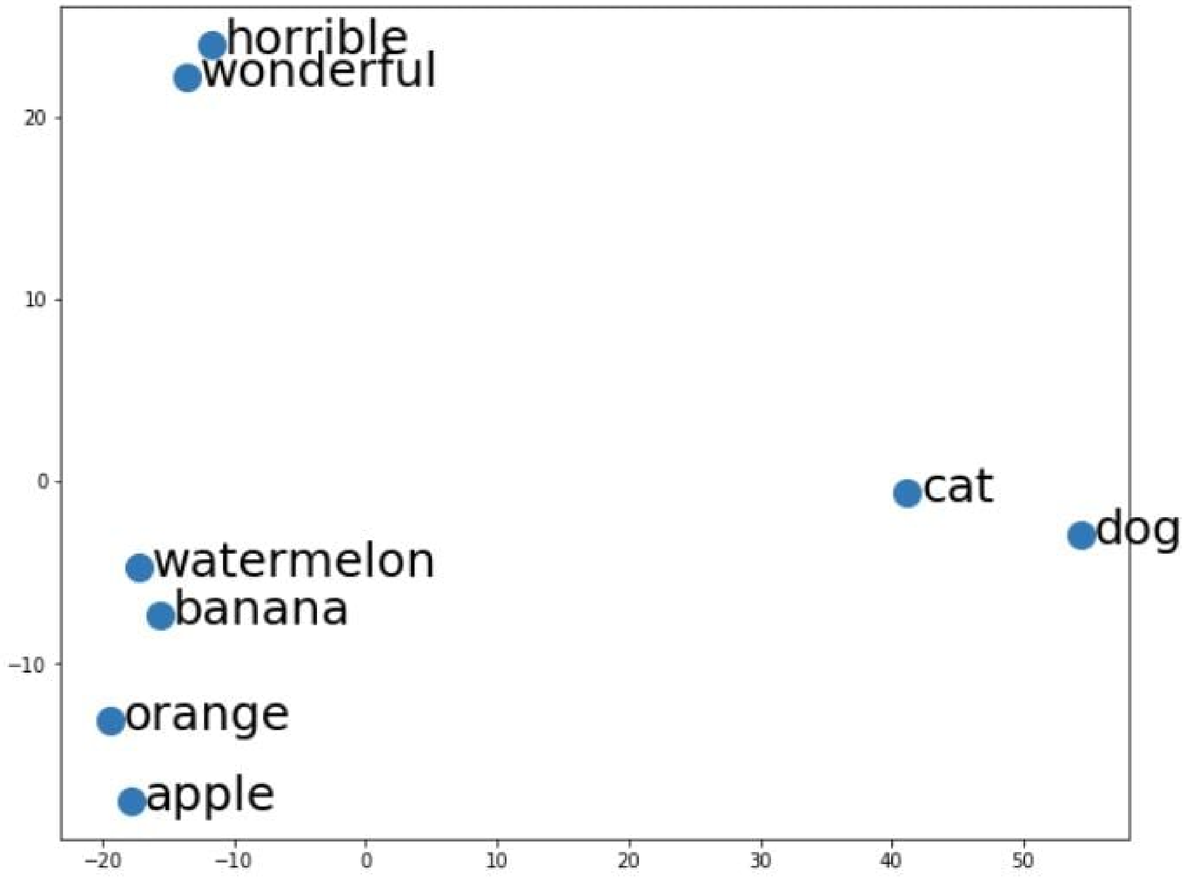
NATURAL LANGUAGE PROCESSING WITH SPACY

Azadeh Mobasher
Principal Data Scientist

Word vectors visualization

- Word vectors allow to understand how words are **grouped**
- Principal Component Analysis projects word vectors into a two-dimensional space

```
array([ 2.2407   ,  1.0389   ,  1.3092   , -1.7335   , -0.78466  ,
       -0.29269  , -1.8059   , -2.5223   ,  0.78025  ,  2.4899   ,
       -0.091849 ,  0.28755  , -1.5057   ,  2.6337   ,  2.5252   ,
       -0.22432  , -2.2068   , -0.57895  , -0.56551  , -1.9338   ,
        1.4973   ,  0.85889  ,  3.3559   , -3.7527   ,  0.22585   ,
       -0.16969  ,  0.51389  ,  0.46073  , -0.28248  , -2.6048   ,
       -3.5896   , -1.0971   , -1.5517   , -0.12185  ,  2.8633   ,
       -1.2525   , -1.6924   , -2.2917   ,  0.97793  ,  0.46954   ,
       -3.595    , -0.17357  ,  0.9805   , -1.8044   , -0.72183   ,
       -0.40709  , -3.0943   ,  0.13095  , -2.9015   ,  1.4768    ,
       -1.0588   , -2.8123   ,  1.2936   , -0.0075977,  2.9975    ,
       -2.4438   ,  0.12348  ,  1.8322   ,  0.35869  , -0.018335  ,
        1.9534   ,  1.4417   ,  0.99895  , -2.8209   , -0.75846   ,
       -1.8438   , -3.2658   , -0.46574  ,  0.90322  ,  0.79868   ,
       -1.6134   , -0.33082  ,  1.1541   , -4.7334   ,  1.4964    ,
       -2.4014   , -1.3461   , -0.95551  ,  0.29671  , -1.4506    ,
       -0.87128  , -3.0714   ,  1.3597   , -0.038133 ,  1.6414    ,
       -0.90879  ,  2.7406   ,  2.2951   , -3.1423   , -3.7525    ,
        0.74033  ,  1.4921   ,  0.47422  , -1.8337   , -1.8168    ,
        0.66901  , -1.3612   , -2.2729   , -1.7656   , -0.73968   ],
      dtype=float32)
```



Word vectors visualization

- Import required libraries and a `spaCy` model.

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import numpy as np
nlp = spacy.load("en_core_web_md")
```

- Extract word vectors for a given list of words and stack them vertically.

```
words = ["wonderful", "horrible",
           "apple", "banana", "orange", "watermelon",
           "dog", "cat"]
word_vectors = np.vstack([nlp.vocab.vectors[nlp.vocab.strings[w]] for w in words])
```

Word vectors visualizations

- Extract two principal components using PCA.

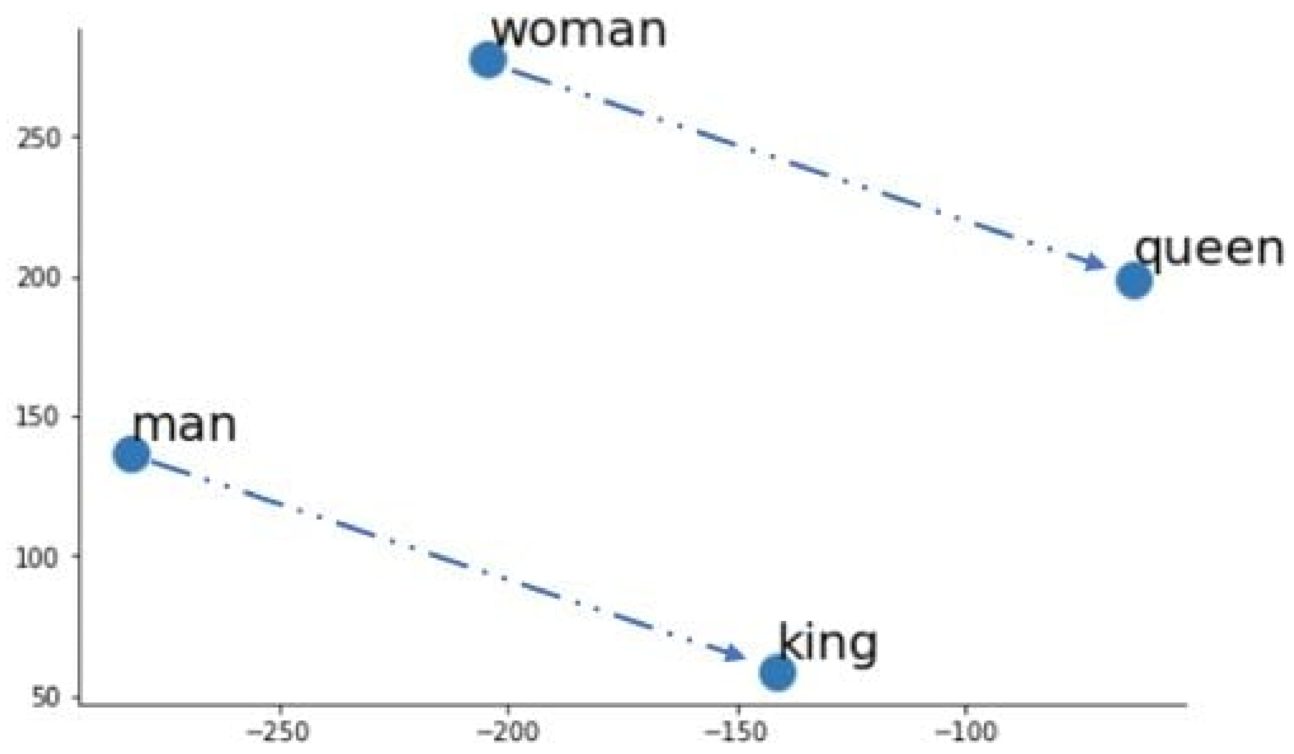
```
pca = PCA(n_components=2)
word_vectors_transformed = pca.fit_transform(word_vectors)
```

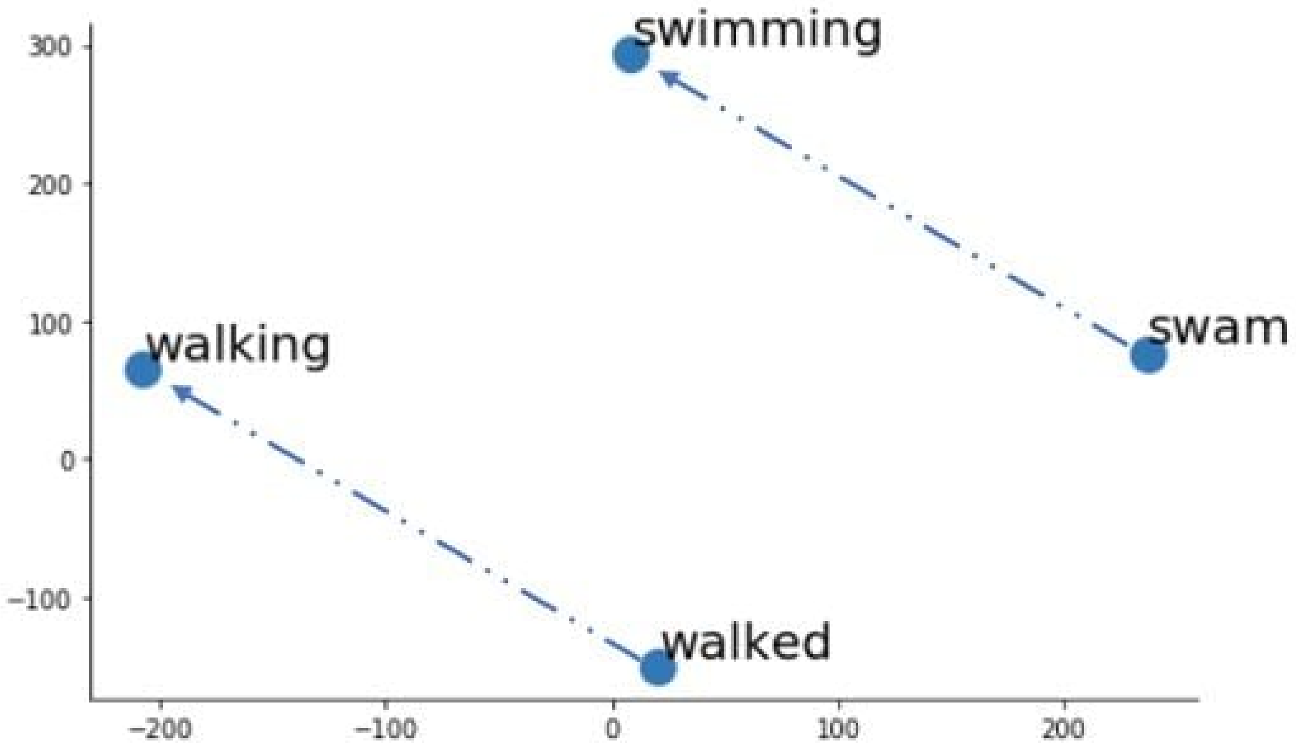
- Visualize the scatter plot of transformed vectors.

```
plt.figure(figsize=(10, 8))
plt.scatter(word_vectors_transformed[:, 0], word_vectors_transformed[:, 1])
for word, coord in zip(words, word_vectors_transformed):
    x, y = coord
    plt.text(x, y, word, size=10)
plt.show()
```

Analogies and vector operations

- A semantic relationship between a pair of words.
- **Word embeddings** generate analogies such as gender and tense:
  - queen - woman + man = king





Similar words in a vocabulary

- `spaCy` find **semantically similar terms** to a given term

```python
import numpy as np
import spacy
nlp = spacy.load("en_core_web_md")

word = "covid"
most_similar_words = nlp.vocab.vectors.most_similar(
    np.asarray([nlp.vocab.vectors[nlp.vocab.strings[word]]]), n=5)

words = [nlp.vocab.strings[w] for w in most_similar_words[0][0]]
print(words)
```

```
>>> ['Covi', 'CoVid', 'Covici', 'COVID-19', 'corona']
```

Let's practice!

NATURAL LANGUAGE PROCESSING WITH SPACY

Measuring semantic similarity with spaCy

NATURAL LANGUAGE PROCESSING WITH SPACY

Azadeh Mobasher
Principal Data Scientist

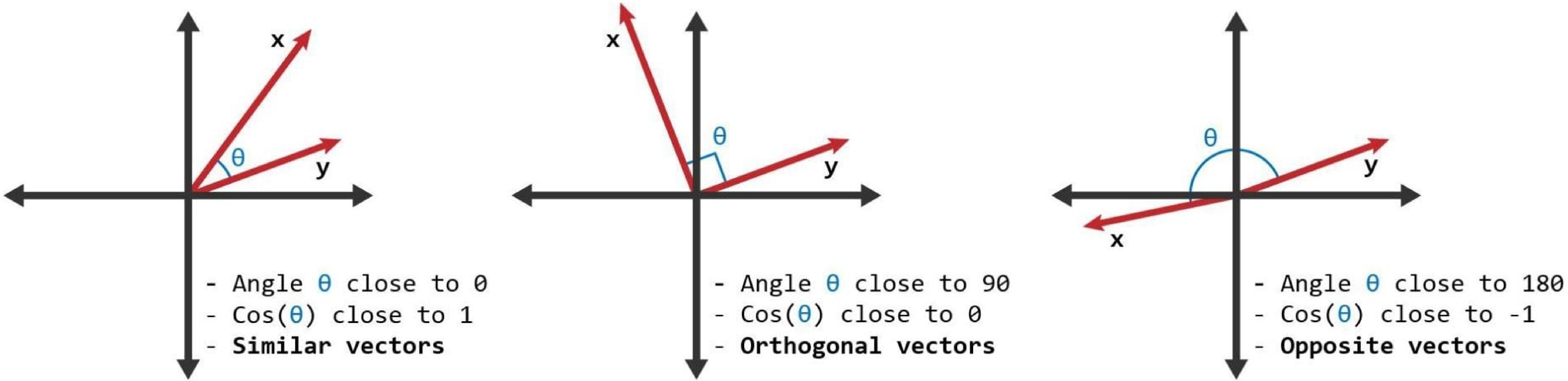
The semantic similarity method

- Process of **analyzing texts** to **identify similarities**
- Categorizes texts into **predefined categories** or detect **relevant texts**
- **Similarity score** measures how similar two pieces of text are

```
What is the cheapest flight from Boston to Seattle?
Which airline serves Denver, Pittsburgh and Atlanta?
What kinds of planes are used by American Airlines?
```

Similarity score

- A **metric** defined over texts
- To measure similarity use **Cosine similarity** and **word vectors**
- **Cosine similarity** is any number between 0 and 1



Token similarity

- `spaCy` calculates similarity scores between Token objects

```
nlp = spacy.load("en_core_web_md")
doc1 = nlp("We eat pizza")
doc2 = nlp("We like to eat pasta")
token1 = doc1[2]
token2 = doc2[4]
print(f"Similarity between {token1} and {token2} = ", round(token1.similarity(token2), 3))
```

```
>>> Similarity between pizza and pasta =  0.685
```

Span similarity

- `spaCy` calculates semantic similarity of two given `Span` objects

```
doc1 = nlp("We eat pizza")
doc2 = nlp("We like to eat pasta")

span1 = doc1[1:]
span2 = doc2[1:]
print(f"Similarity between \"{span1}\" and \"{span2}\" = ",
        round(span1.similarity(span2), 3))
```

```
>>> Similarity between "eat pizza" and "like to eat pasta" =  0.588
```

```
print(f"Similarity between \"{doc1[1:]}\" and \"{doc2[3:]}\" = ",
        round(doc1[1:].similarity(doc2[3:]), 3))
```

```
>>> Similarity between "eat pizza" and "eat pasta" =  0.936
```

Doc similarity

- `spaCy` calculates the similarity scores between two documents

```
nlp = spacy.load("en_core_web_md")

doc1 = nlp("I like to play basketball")
doc2 = nlp("I love to play basketball")
print("Similarity score :", round(doc1.similarity(doc2), 3))
```

```
>>> Similarity score : 0.975
```

- High cosine similarity shows highly semantically similar contents
- `Doc` vectors default to an average of word vectors

Sentence similarity

- `spaCy` finds relevant content to a given keyword
- Finding similar customer questions to the word **price:**

```
sentences = nlp("What is the cheapest flight from Boston to Seattle?
                 Which airline serves Denver, Pittsburgh and Atlanta?
                 What kinds of planes are used by American Airlines?")

keyword = nlp("price")
for i, sentence in enumerate(sentences.sents):
    print(f"Similarity score with sentence {i+1}: ", round(sentence.similarity(keyword), 5))
```

```
>>> Similarity score with sentence 1:  0.26136
Similarity score with sentence 2:  0.14021
Similarity score with sentence 3:  0.13885
```

Let's practice!

NATURAL LANGUAGE PROCESSING WITH SPACY