



COMP3204/COMP6223: Computer Vision

# Image search and Bags of Visual Words

Jonathon Hare  
[jsh2@ecs.soton.ac.uk](mailto:jsh2@ecs.soton.ac.uk)

# Text Information Retrieval

# The *bag* data structure

- ❖ A bag is an **unordered** data structure like a *set*, but which unlike a set allows elements to be **inserted multiple times**.
- ❖ sometimes called a *multiset* or a *counted set*



# Bag of Words

A document



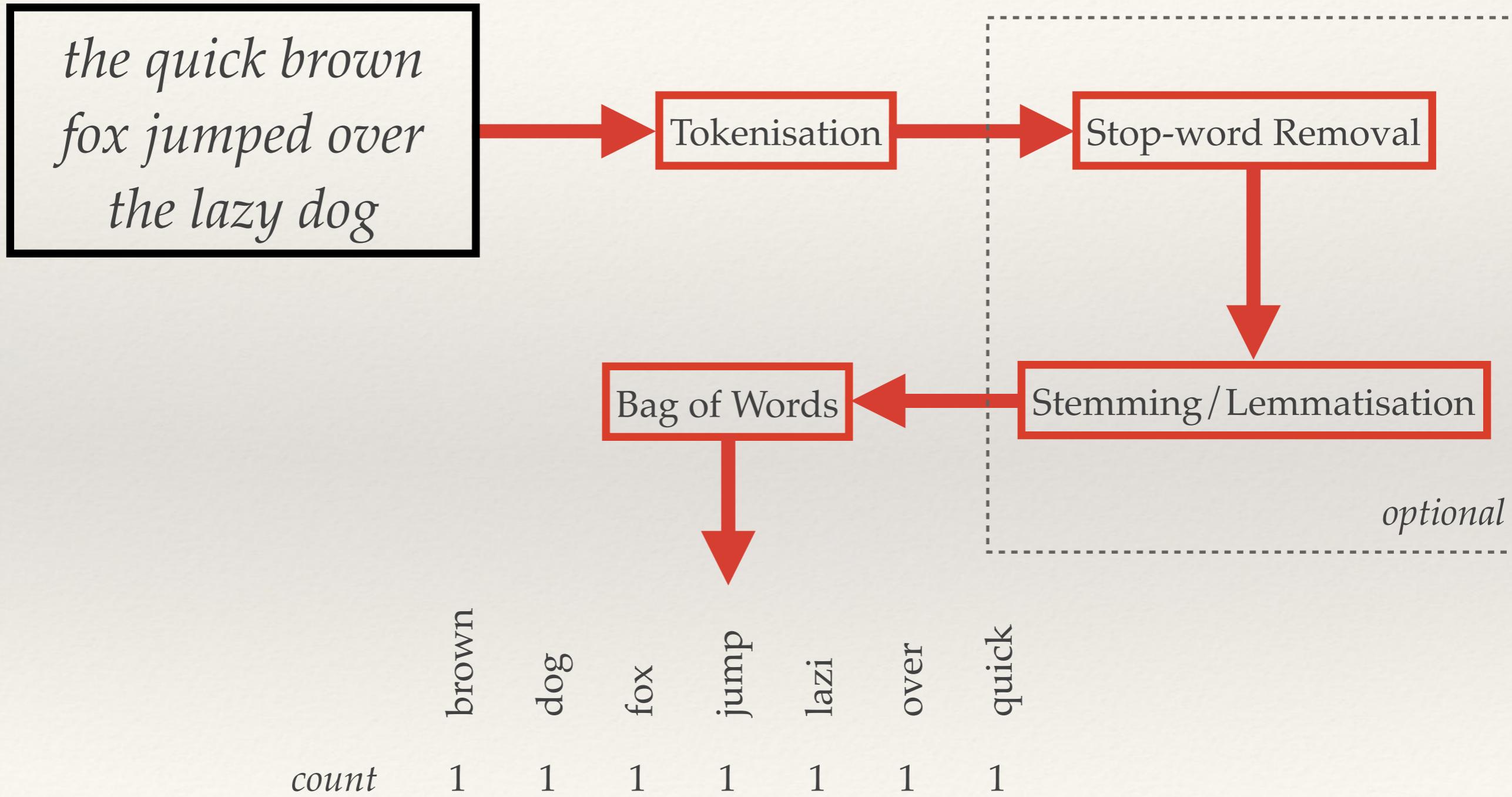
*the quick brown  
fox jumped over  
the lazy dog*



The bag of words  
describing the  
document



# Text processing (feature extraction)



---

# The Vector-Space Model

---

- ❖ Conceptually simple:
  - ❖ Model each document by a vector
  - ❖ Model each query by a vector
  - ❖ Assumption: documents that are “close together” in space are similar in meaning.
  - ❖ Use standard similarity measures to rank each document to a query in terms of decreasing similarity

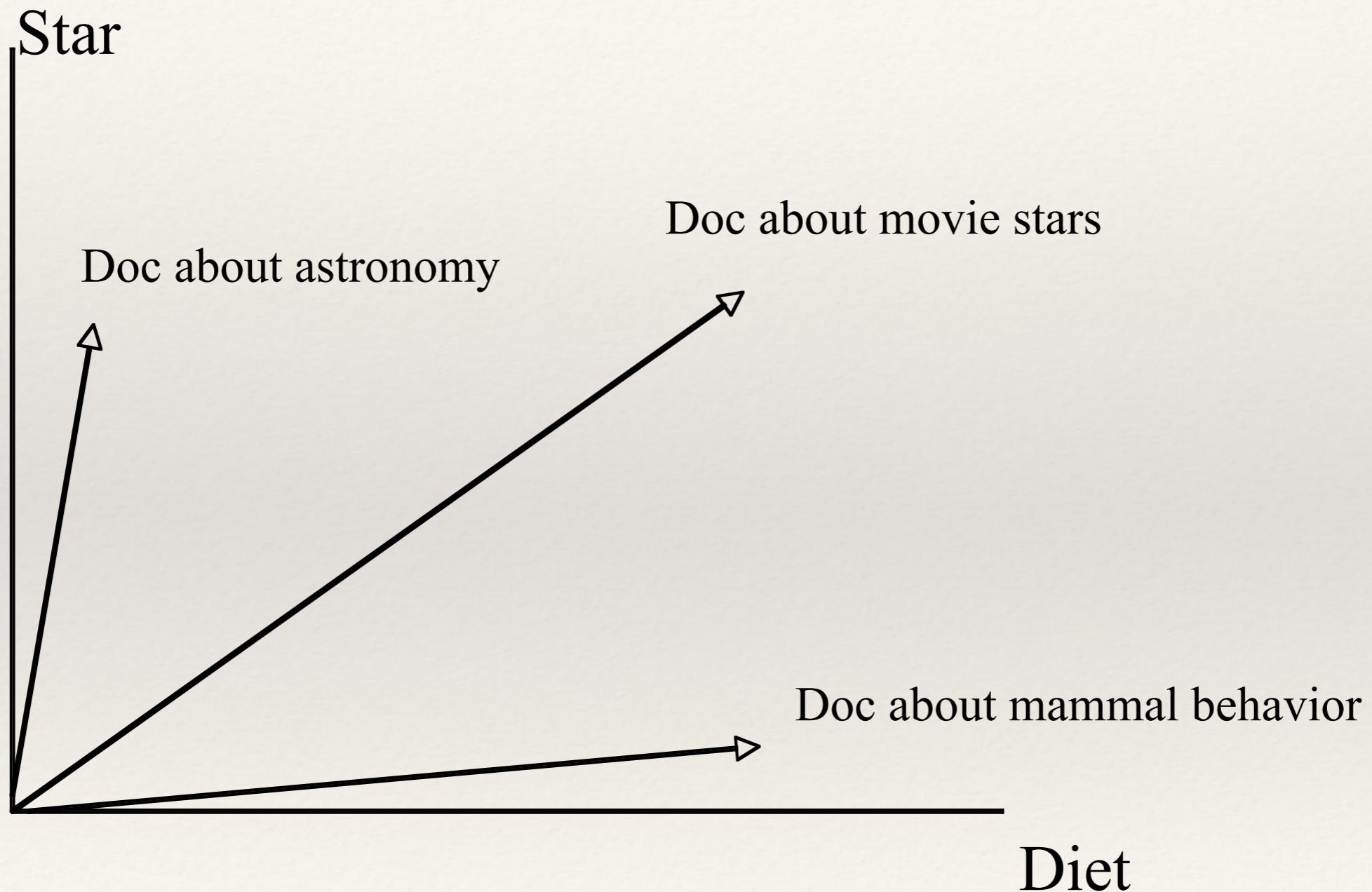
---

# Bag of Words Vectors

---

- ❖ The lexicon or vocabulary is the **set** of all (processed) words across all documents known to the system.
- ❖ We can create vectors for each document with as many dimensions as there are words in the lexicon.
  - ❖ Each word in the document's bag of words contributes a count to the corresponding element of the vector for that word.
    - ❖ In essence, each vector is a histogram of the word occurrences in the respective document.
    - ❖ **Vectors will have very high number of dimensions, but will be very sparse.**

# The Vector-space Model



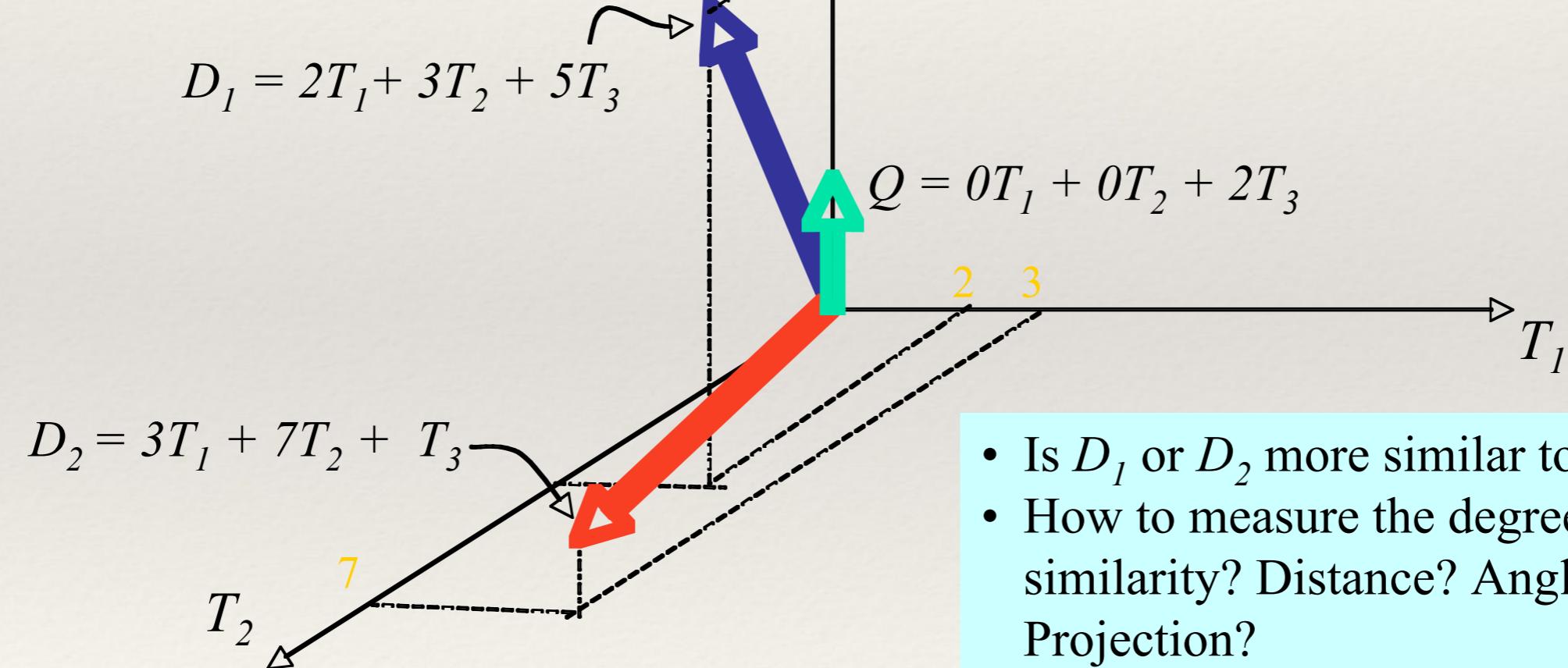
# Searching the VSM

Example:

$$D_1 = 2T_1 + 3T_2 + 5T_3$$

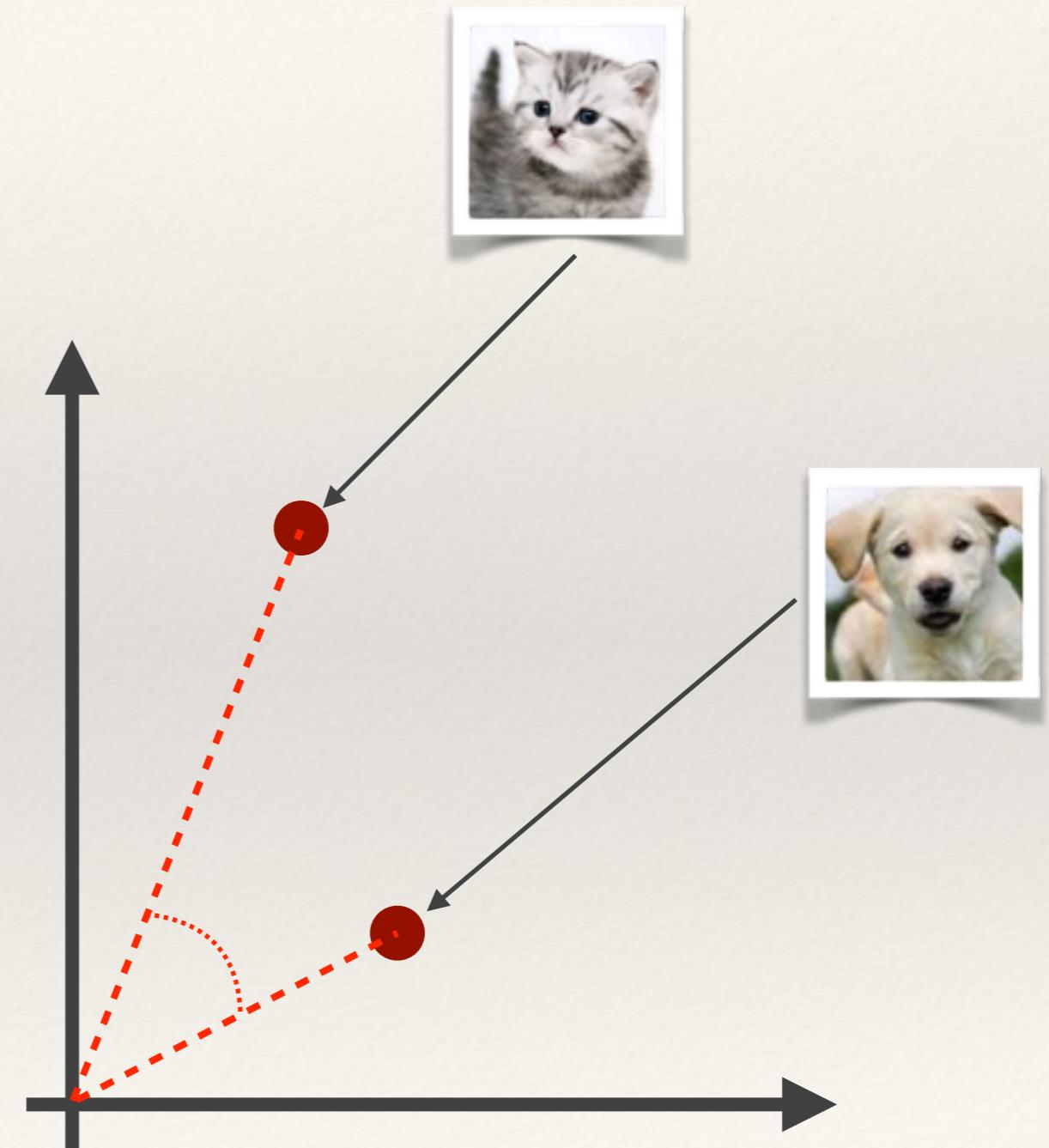
$$D_2 = 3T_1 + 7T_2 + T_3$$

$$Q = 0T_1 + 0T_2 + 2T_3$$



# Recap: Cosine Similarity

$$\cos(\theta) = \frac{p \cdot q}{\|p\| \|q\|} = \frac{\sum_{i=1}^n p_i q_i}{\sqrt{\sum_{i=1}^n p_i^2} \sqrt{\sum_{i=1}^n q_i^2}}$$



# Recap: Cosine Similarity

If  $p$  and  $q$  are both high dimensional and sparse,  
then you're going to spend a lot of time multiplying 0  
by 0 and adding 0 to the accumulator

$$\cos(\theta) = \frac{p \cdot q}{\|p\| \|q\|} = \frac{\sum_{i=1}^n p_i q_i}{\sqrt{\sum_{i=1}^n p_i^2} \sqrt{\sum_{i=1}^n q_i^2}}$$

These can be pre-computed and stored!

# Inverted Indexes

|           |                   |
|-----------|-------------------|
| Aardvark  | [doc3:4]          |
| Astronomy | [doc1:2]          |
| Diet      | [doc2:9; doc3:8]  |
| ...       |                   |
| Movie     | [doc2:10]         |
| Star      | [doc1:13; doc2:4] |
| Telescope | [doc1:15]         |

...A map of words to lists of postings...

# Inverted Indexes

|           |                   |
|-----------|-------------------|
| Aardvark  | [doc3:4]          |
| Astronomy | [doc1:2]          |
| Diet      | [doc2:9; doc3:8]  |
| ...       |                   |
| Movie     | [doc2:10]         |
| Star      | [doc1:13, doc2:4] |
| Telescope | [doc1:15]         |



A **posting** is a pair formed by a **document ID** and the **number of times** the specific word appeared in that document

# Computing the Cosine Similarity

---

- ❖ For each word in the query, lookup the relevant postings list and accumulate similarities for only the documents seen in those postings lists
  - ❖ much more efficient than fully comparing vectors...

Query: “Movie Star”

|           |                   |
|-----------|-------------------|
| Aardvark  | [doc3:4]          |
| Astronomy | [doc1:2]          |
| Diet      | [doc2:9; doc3:8]  |
| ...       |                   |
| Movie     | [doc2:10]         |
| Star      | [doc1:13; doc2:4] |
| Telescope | [doc1:15]         |

Query: “Movie Star”

Accumulation table:

|      |      |
|------|------|
| doc2 | 10x1 |
|------|------|

|           |                   |
|-----------|-------------------|
| Aardvark  | [doc3:4]          |
| Astronomy | [doc1:2]          |
| Diet      | [doc2:9; doc3:8]  |
| ...       |                   |
| Movie     | [doc2:10]         |
| Star      | [doc1:13; doc2:4] |
| Telescope | [doc1:15]         |

Query: “Movie Star”

Accumulation table:

|      |                            |
|------|----------------------------|
| doc2 | $10 \times 1 + 4 \times 1$ |
| doc1 | $13 \times 1$              |

|           |                   |
|-----------|-------------------|
| Aardvark  | [doc3:4]          |
| Astronomy | [doc1:2]          |
| Diet      | [doc2:9; doc3:8]  |
| ...       |                   |
| Movie     | [doc2:10]         |
| Star      | [doc1:13; doc2:4] |
| Telescope | [doc1:15]         |

Query: “Movie Star”

Accumulation table:

|             |   |
|-------------|---|
| doc2        | $(10 \times 1 + 4 \times 1) / 14.04 = \mathbf{0.997}$ |
| doc1        | $13 \times 1 / 19.95 = \mathbf{0.652}$                |
| <i>doc3</i> | <b>0</b>  |

|           |                   |
|-----------|-------------------|
| Aardvark  | [doc3:4]          |
| Astronomy | [doc1:2]          |
| Diet      | [doc2:9; doc3:8]  |
| ...       |                   |
| Movie     | [doc2:10]         |
| Star      | [doc1:13; doc2:4] |
| Telescope | [doc1:15]         |

---

# Weighting the vectors

---

- ❖ The number of times a word occurs in a document reflects the importance of that word in the document.
- ❖ Intuitions:
  - ❖ A term that appears in many documents is not important: e.g., the, going, come, ...
  - ❖ If a term is frequent in a document and rare across other documents, it is probably important in that document.

---

# Possible weighting schemes

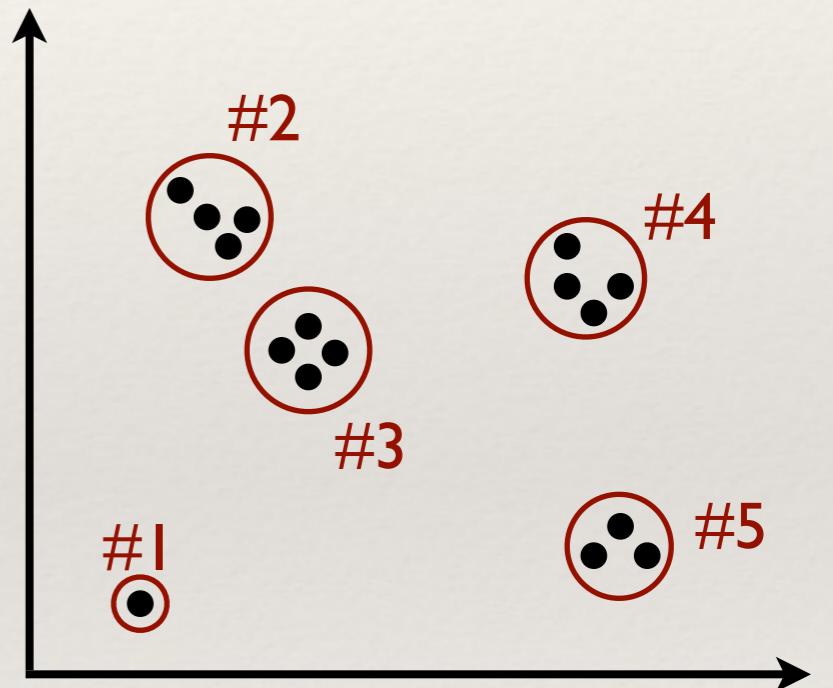
---

- ❖ Binary weights
  - ❖ Only presence (1) or absence (0) of a term recorded in vector.
- ❖ Raw frequency
  - ❖ Frequency of occurrence of term in document included in vector.
- ❖ TF-IDF
  - ❖ Term frequency is the frequency count of a term in a document.
  - ❖ Inverse document frequency (idf) provides high values for rare words and low values for common words.

# Vector Quantisation

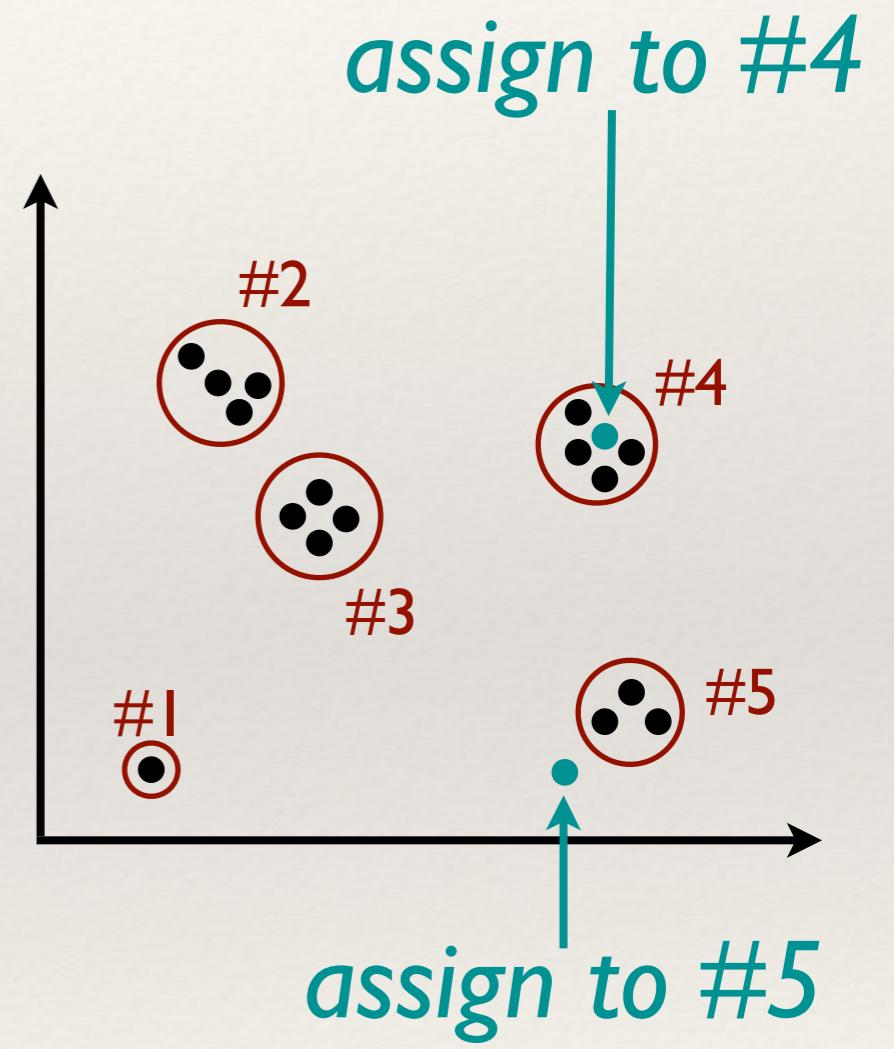
# Learning a Vector Quantiser

- ❖ Vector quantisation is a lossy data compression technique.
- ❖ Given a set of vectors, a technique like K-Means clustering can be used to learn a fixed size set of representative vectors.
  - ❖ The representatives are the mean vector of each cluster in k-means
  - ❖ The set of representation vectors is called a **codebook**



# Vector Quantisation

- ❖ Vector quantisation is achieved by representing a vector by another approximate vector, which is drawn from a pool of representative vectors.
- ❖ Each input vector is assigned to the “closest” vector from the pool.



Visual Words

---

# SIFT Visual Words

---

- ❖ We can vector quantise SIFT descriptors (or any other local feature)
  - ❖ Each descriptor is replaced by a representative vector known as a **visual word**
  - ❖ In essence the *visual word* describes a small image patch with a certain pattern of pixels
  - ❖ In many ways the process of applying vector quantisation to local features is analogous to the process of stemming words.
  - ❖ The codebook is the visual equivalent of a lexicon or vocabulary.

---

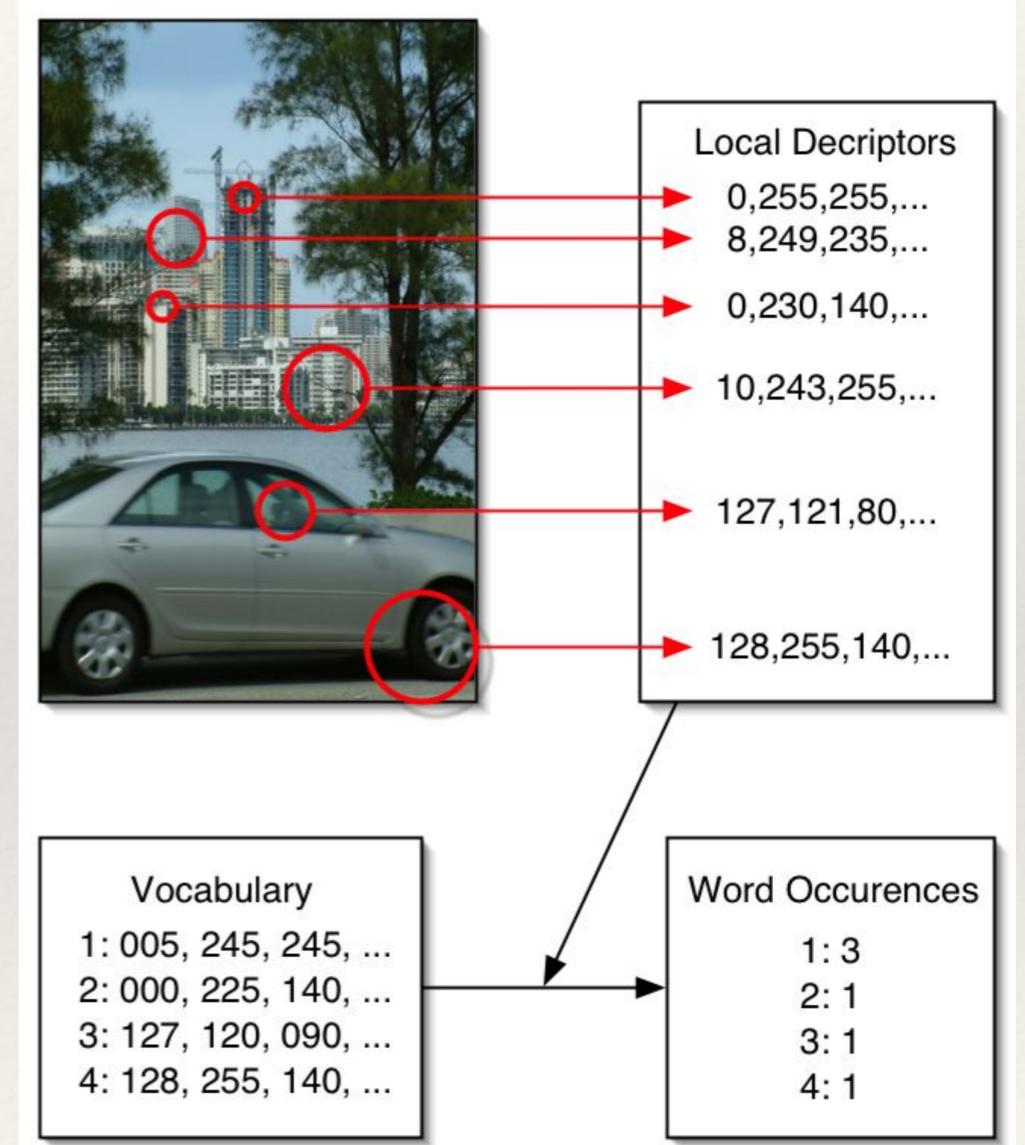
# Bags of Visual Words

---

- ❖ Once we've quantised the local features into visual words, they can be put into a bag.
  - ❖ This is a **Bag of Visual Words (BoVW)**
  - ❖ We're basically ignoring where in the image the local features came from (including ignoring scale)

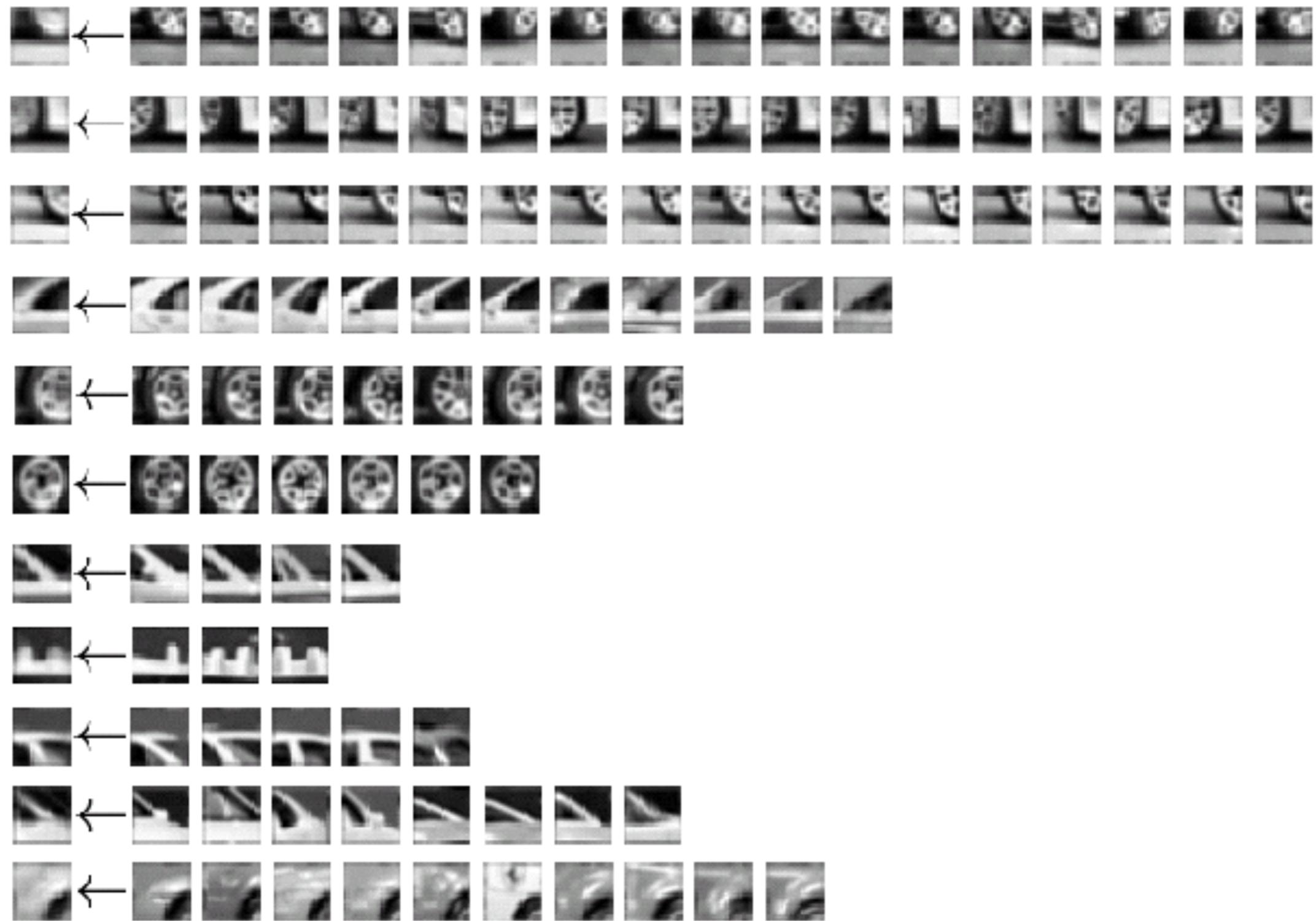
# Histograms of Bags of Visual Words

- ❖ Like in the case of text, once we have a BoVW and knowledge of the complete vocabulary (the codebook) we can build histograms of visual word occurrences!
- ❖ This is rather nice... it gives us a way of aggregating a variable number of local descriptors into a fixed length vector.
  - ❖ Useful for machine learning
  - ❖ But also allows us to apply techniques for text retrieval to images



*Demo: SIFT visual word histogram*

# Visualising Visual Words



# The effect of codebook size

---

- ❖ There is one **key parameter** in building visual words representations - **the size of the vocabulary.**
  - ❖ Too small, and all vectors look the same
    - ❖ Not distinctive
  - ❖ Too big, and the same visual words might never appear across images
    - ❖ Too distinctive

# Content-based Image Retrieval

search  
statement:

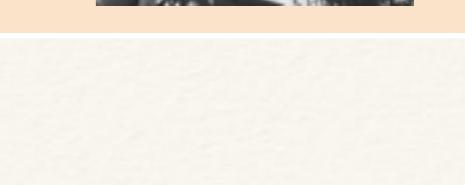


descriptor  
extraction



similarity  
matcher



| Descriptors   | Image   |
|---|---|
|    |   |
|  |  |
|  |  |

---

# BoVW Retrieval

---

- ❖ With the visual word representation, everything used for text retrieval can be applied directly to images
  - ❖ vector space model
  - ❖ cosine similarity
  - ❖ weighting schemes
  - ❖ inverted index

# Optimal codebook size

---

- ❖ Inverted index only gives a performance gain if the vectors are sparse (you don't want to end up explicitly scoring all documents)
- ❖ Visual words also need to sufficiently distinctive to minimise mismatching
  - ❖ Implies a very big codebook
    - ❖ Modern research systems often use 1 Million or more visual words for SIFT vectors

---

# Problems with big codebooks

---

- ❖ There's a slight problem...
  - ❖ Need to use k-means to learn 1 million clusters in 128 dimensions from 10's of millions of features
    - ❖ Non-trivial!
    - ❖ Vector quantisation has the same problems
      - ❖ Have to use approximate methods, like approximate k-d trees

---

# Overall process for building a BoVW retrieval system

---

- ❖ Collect the corpus of images that are to be indexed and made searchable
- ❖ Extract local features from each image
- ❖ Learn a *large* codebook from (a sample of) the features
- ❖ Vector quantise the features, and build BoVW representations for each image
- ❖ Construct an inverted index with the BoVW representations

*Demo: A BoVW retrieval system for  
geo-location estimation*

---

# Current research

---

- ❖ Lot of interest in content-based search for *massive* datasets
  - ❖ Two directions:
    - ❖ Hashing of local features
    - ❖ Tiny features (~16 bytes per image!)
      - ❖ Local features still used as the basis, but encoded in a different way to make dense features
        - ❖ Still uses k-means, but much smaller k
        - ❖ known as *VLAD*: *Vector of Locally Aggregated Descriptors*
        - ❖ VLAD descriptors then vector quantised using a “product quantiser”

---

# Summary

---

- ❖ Effective and efficient text search can be achieved with bags of words, the vector-space model and inverted indexes.
- ❖ Vector-quantisation can be applied to local features, making them into visual words.
  - ❖ Then you can apply all the same techniques used for text to make efficient retrieval systems!
    - ❖ This is a good way of making highly scalable, effective and efficient content-based image retrieval systems