Search technology week 8

Group 1

Agenda

- Introduce Assignment D
- Lecture repetition
  - Rocchio vs. Rocchio (1971)
  - kNN for classification
- Weekly shoutout!
- Individual work

Remember science fair groups!

Semester outline

- 23.10
  - My Science fair presentation, recap
- 30.10
  - Assignment E, Naive Bayes, recap
- 06.11
  - Final assignment push, final recap
- 13.11
  - Exam 2023 walkthrough
- 20.11
  - No plan, answer questions

Assignment d-1

- ShingleGenerator: Make shingles to use in approximate matching (n-of-m matching)
- BetterRanker: Ranker using tf-idf-weighting instead of only tf
- SparseDocumentVector: Represent documents as vectors in a sparse space with useful methods

shinglegenerator

- Make overlapping shingles of size self.__width from a buffer
- For example the 3-shingles from `"informatikk"`: `"inf"`, `"nfo"`, `"for"`, `"orm"`, `"rma"`, `"mat"`, `"ati"`, `"tik"`, `"ikk"`
- If the buffer is shorter than the size of a shingle, make a single shorter-than-usual shingle

shinglegenerator

```
def test_strings(self):
    self.assertListEqual(list(self.__tokenizer.strings("")), [])
    self.assertListEqual(list(self.__tokenizer.strings("b")), ["b"])
    self.assertListEqual(list(self.__tokenizer.strings("ba")), ["ba"])
    self.assertListEqual(list(self.__tokenizer.strings("ban")), ["ban"])
    self.assertListEqual(list(self.__tokenizer.strings("bana")), ["ban", "ana"])
    self.assertListEqual(list(self.__tokenizer.strings("banan")), ["ban", "ana", "nan"])
    self.assertListEqual(list(self.__tokenizer.strings("banana")), ["ban", "ana", "nan", "ana"])
```

betterranker

- SimpleRanker
  - A dead simple ranker, based on TF alone.
- BetterRanker
  - A ranker that does traditional TF-IDF ranking, possibly combining it with a static document score (if present).
- If tf-idf is confusing, check out uke06 in the repo
- `def get_document_frequency(self, term: str) -> int`

$$w_{t,d} = \log(1 + tf_{t,d}) \times \log_{10}(N / df_t)$$

sparsedocumentvector

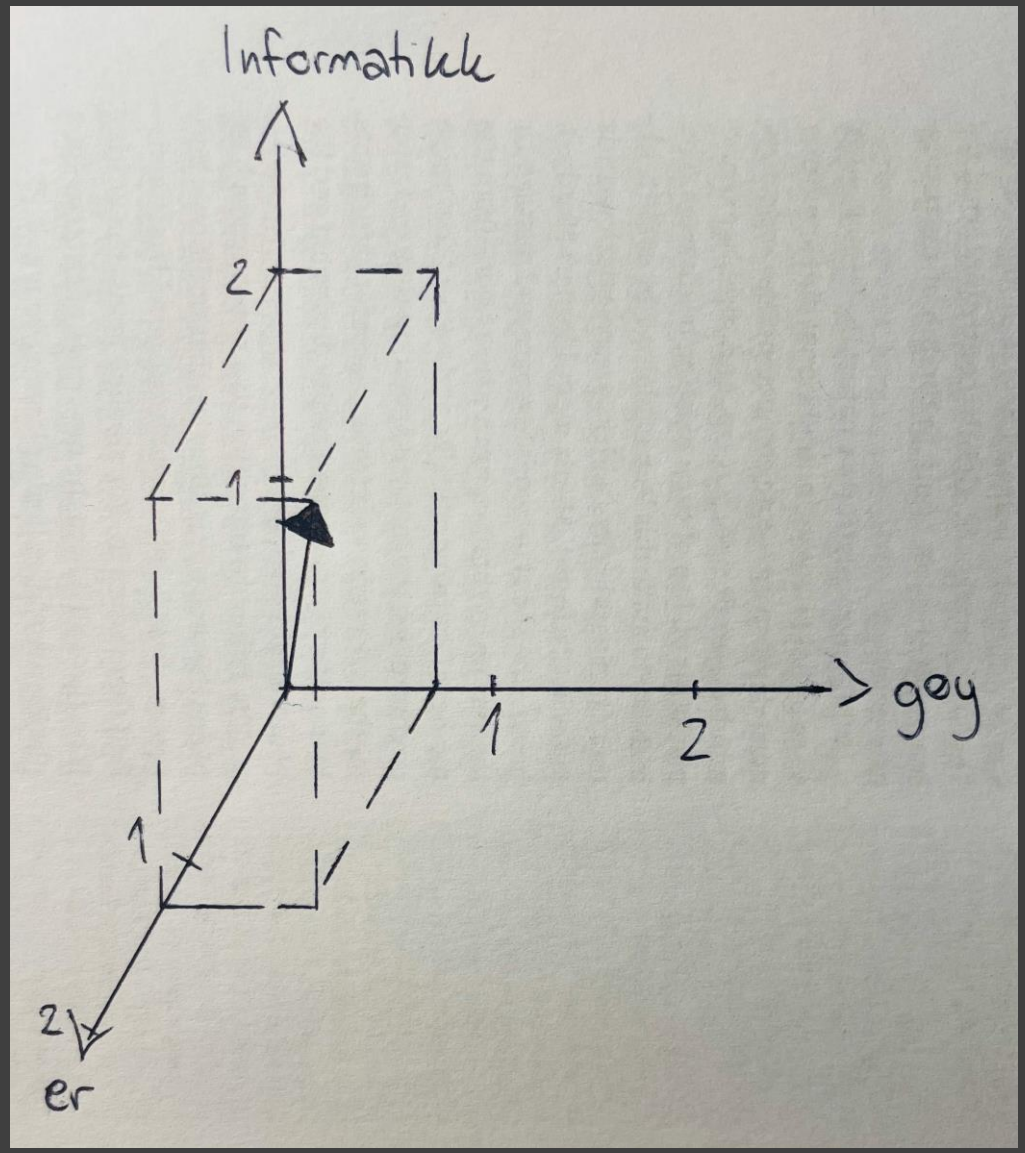
- New this year!
- A dictionary mapping terms of a document to a float value
- A dimension for each term, but only for dimensions not zero
- Tip: remember to update `self._length`
  - What happens with the length of a vector when we normalize, scale, …

sparsedocumentvector

```
def setUp(self):
    self._empty = in3120.SparseDocumentVector({})
    self._vector1 = in3120.SparseDocumentVector({"a": 0.4, "c": 1.2, "b": 0.9})
    self._vector2 = in3120.SparseDocumentVector({"c": 0.5, "a": 0.8, "x": 1.0})
```

Example

```
doc = ({
      “Informatikk": 2.0,
      “er": 1.25,
      “gøy": 0.75
})
```



Assignment d-2

- Implement a compression algorithm of your choice!
- Compare your results to the CompressedInMemoryPostingList from the precode
- No test suite in assignments.py

Agenda

- Introduce Assignment D
- Lecture repetition
  - Rocchio vs. Rocchio (1971)
  - kNN for classification
- Weekly shoutout!
- Individual work

Rocchio 1971 (query expansion)

- Used for document retrieval
- Intuitively: We want to move our query vector towards relevant documents and away from irrelevant documents
- We have a user query and partial knowledge of relevant and nonrelevant documents

Centroid

- The average of vectors
- **Sum of N vectors / N**

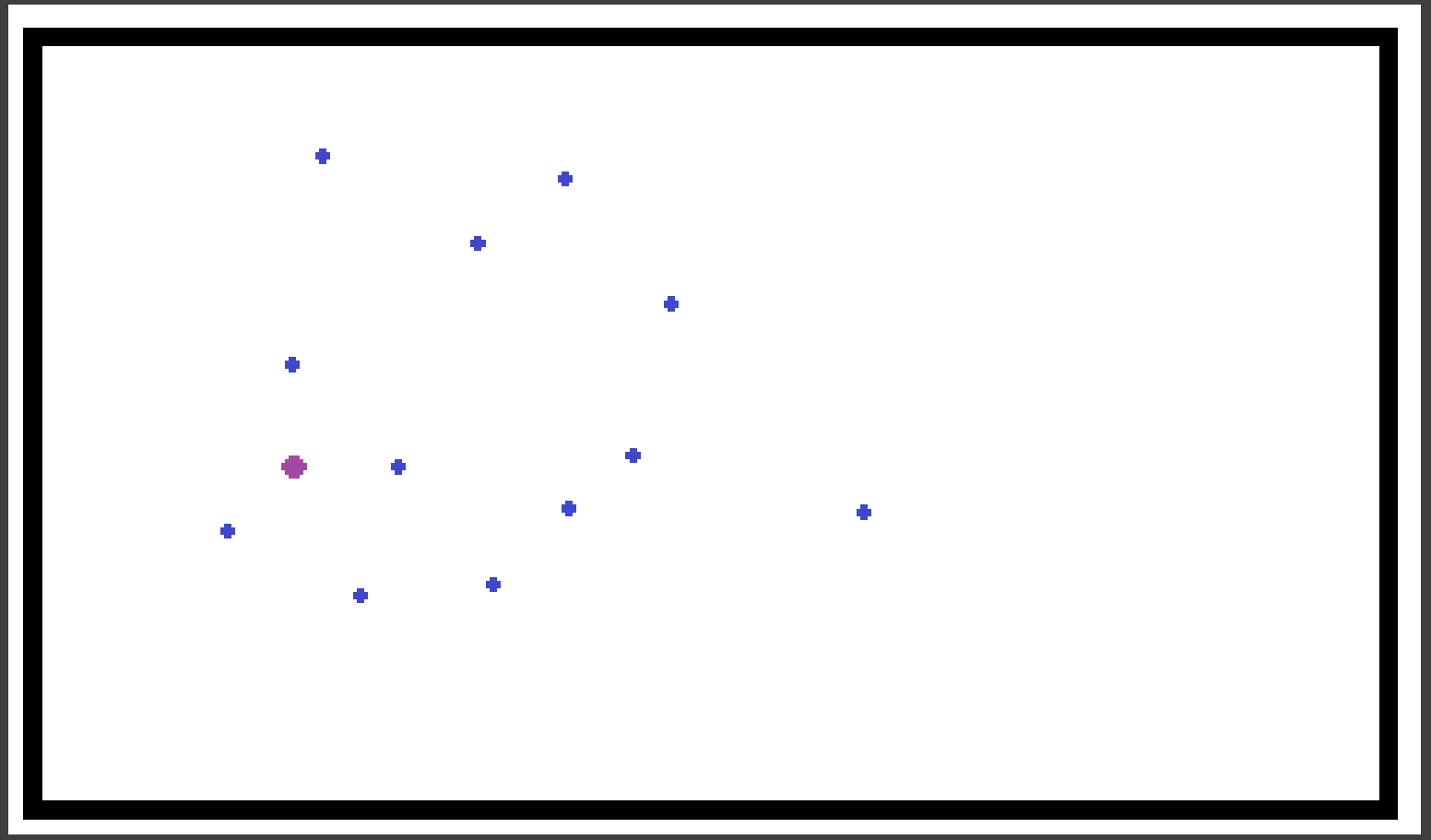
$$\vec{\mu}(C) = \frac{1}{|C|} \sum_{d \in C} \vec{d}$$

Rocchio formula

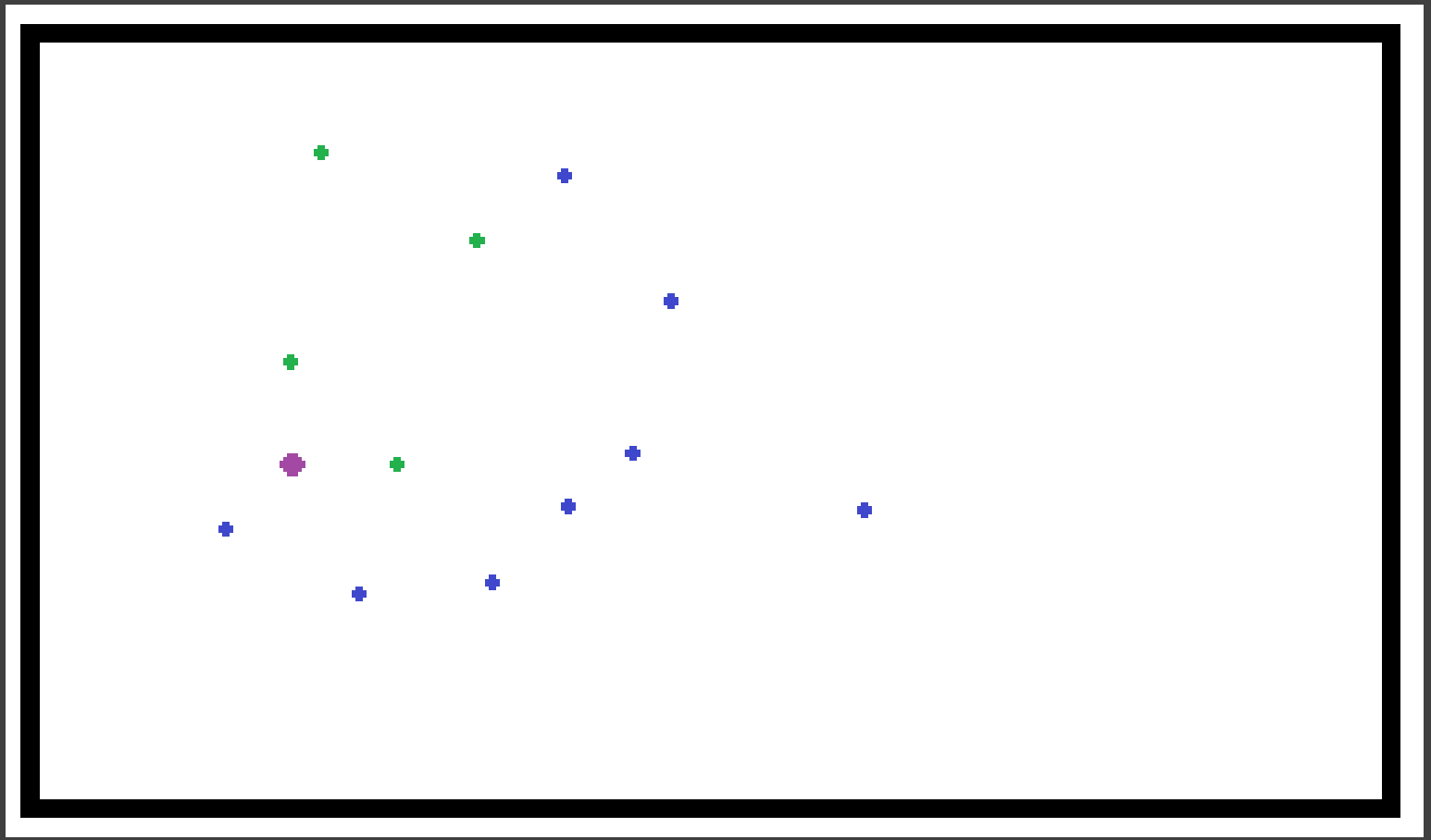
- The centroid of relevant – centroid of nonrelevant documents

$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\vec{d}_j \in C_r} \vec{d}_j - \frac{1}{|C_{nr}|} \sum_{\vec{d}_j \notin C_r} \vec{d}_j$$

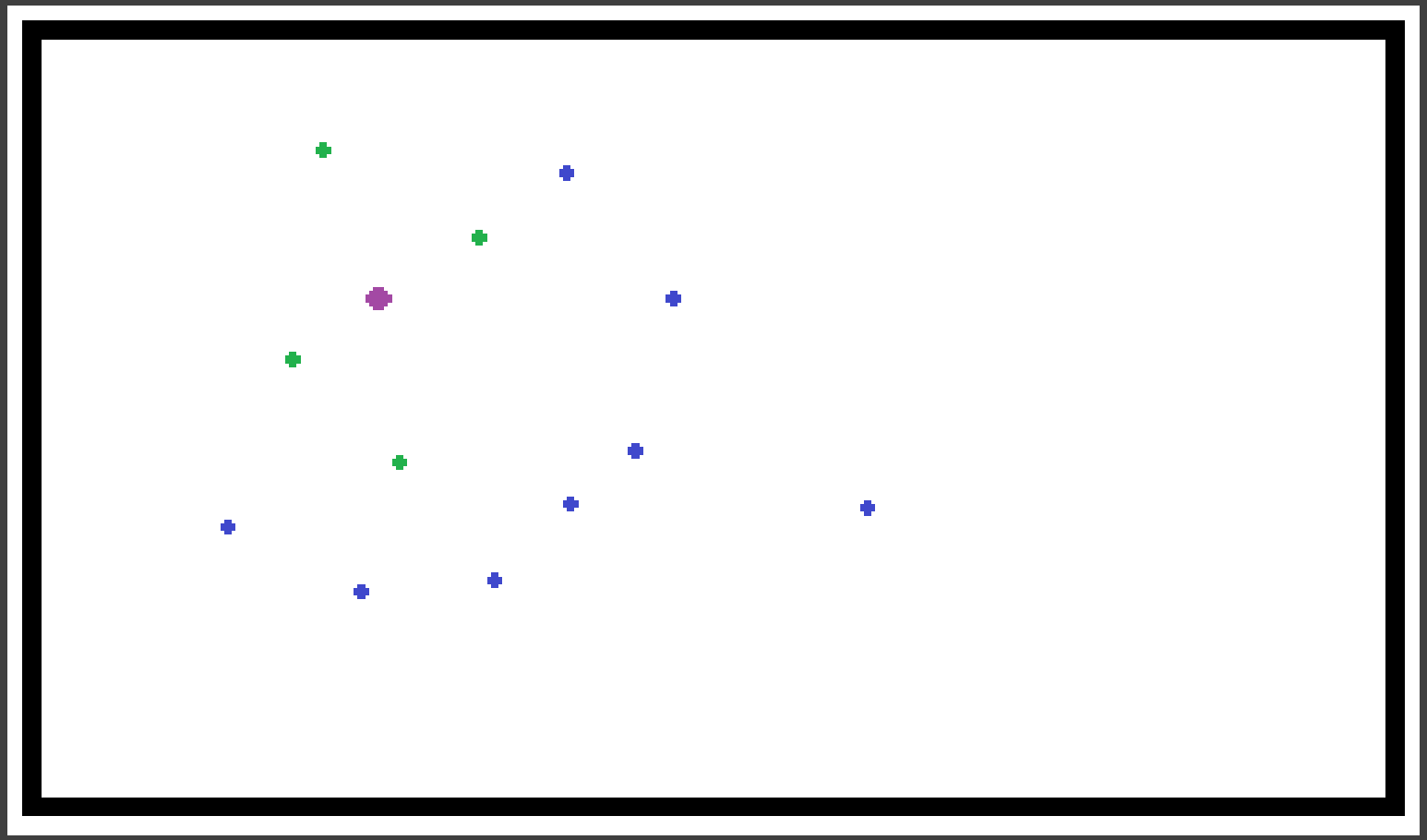
Example



Example



Example

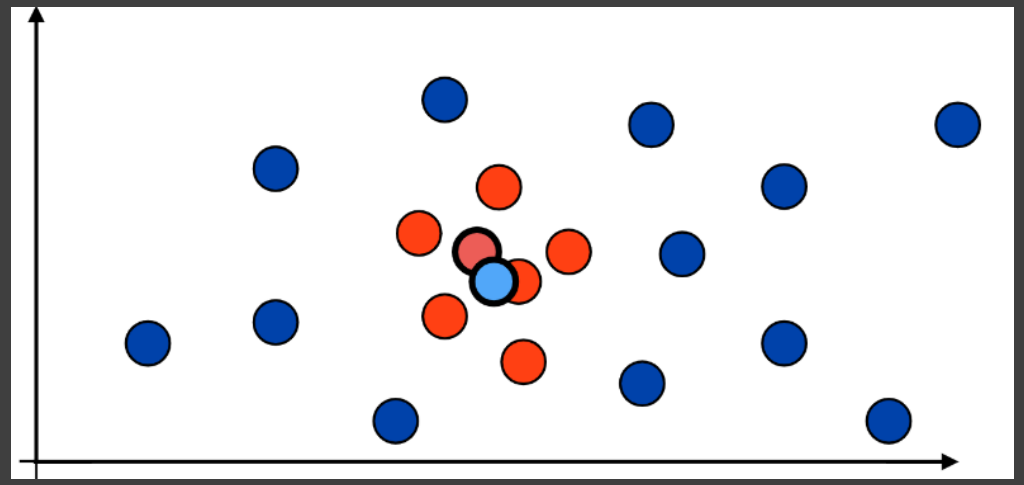


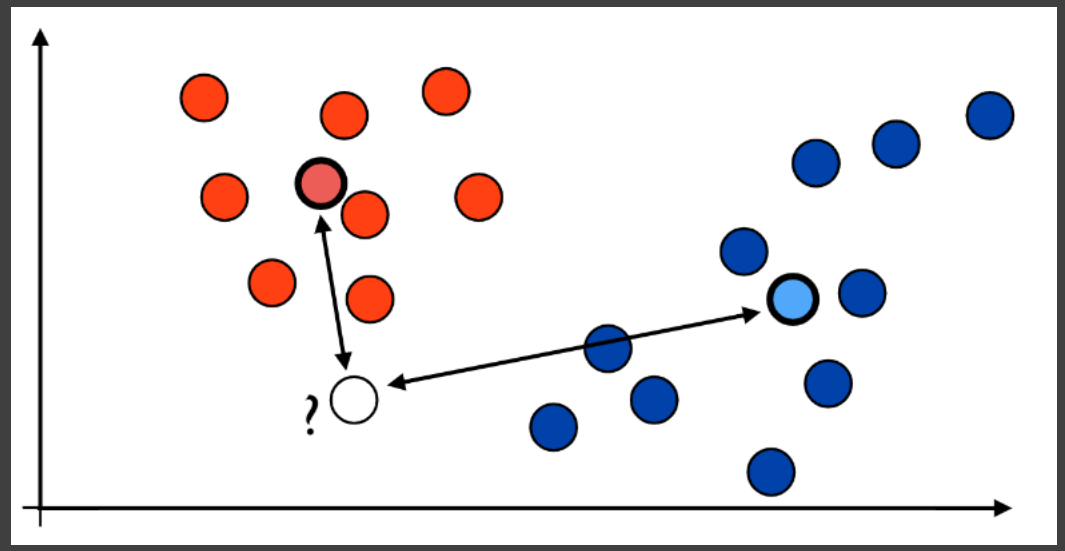
Rocchio 1971 (query expansion)

- The original query is ignored and we are shown documents close to the new vector

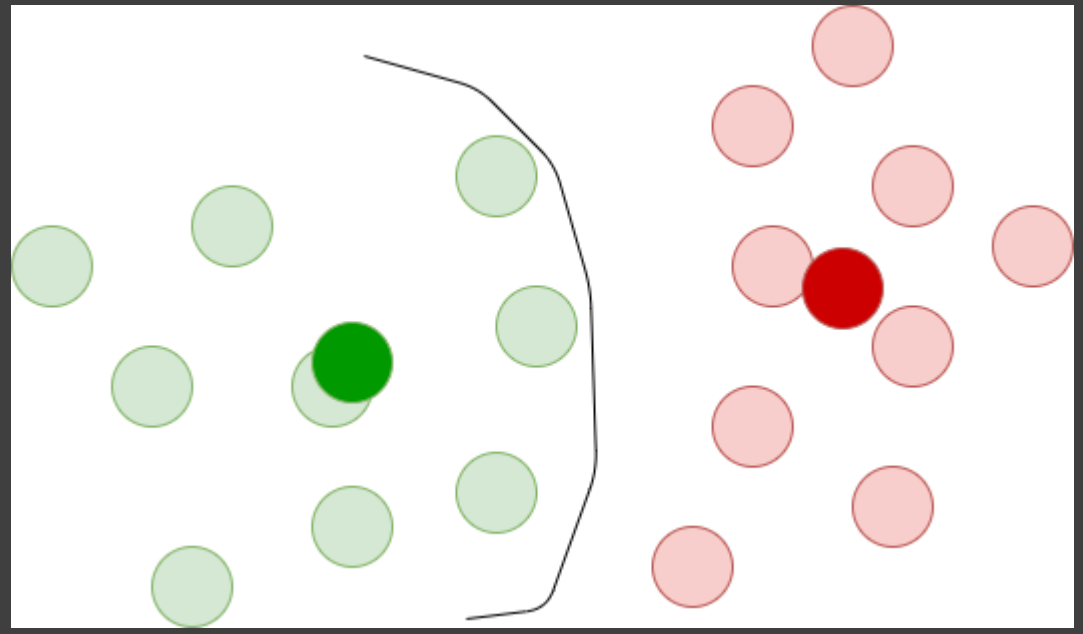
Rocchio (classification)

- Calculate centroid for each class
- New point x will be classified as whatever centroid is closest
- Doesn’t work if classes can’t be separated easily



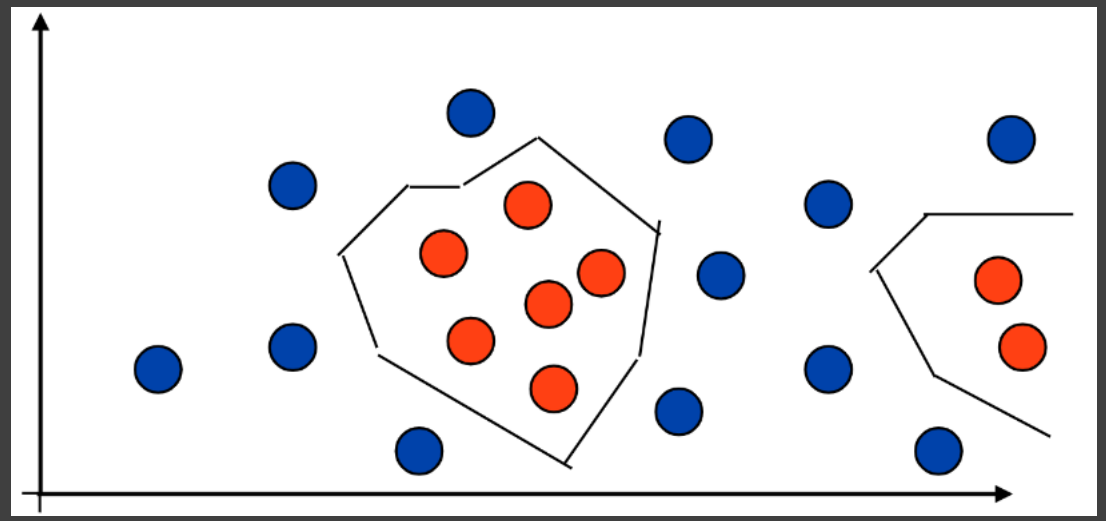


Example - Rocchio



k Nearest Neighbours

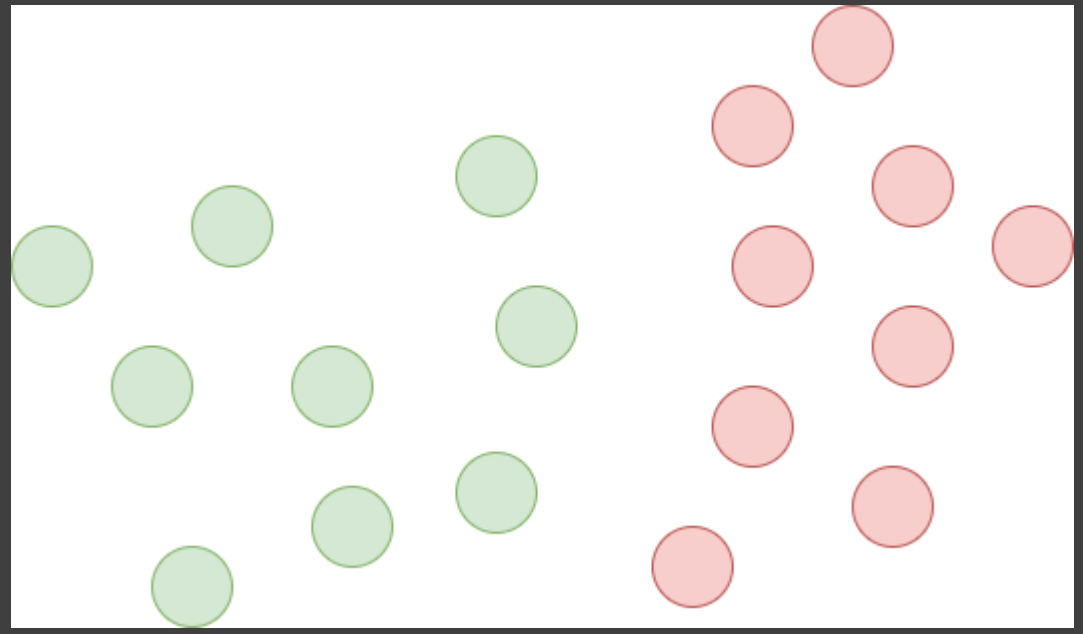
- What is the majority of my k closest neighbours?
- No training required (but need to store data points and calculate many distances)
- Handles complex decision boundaries



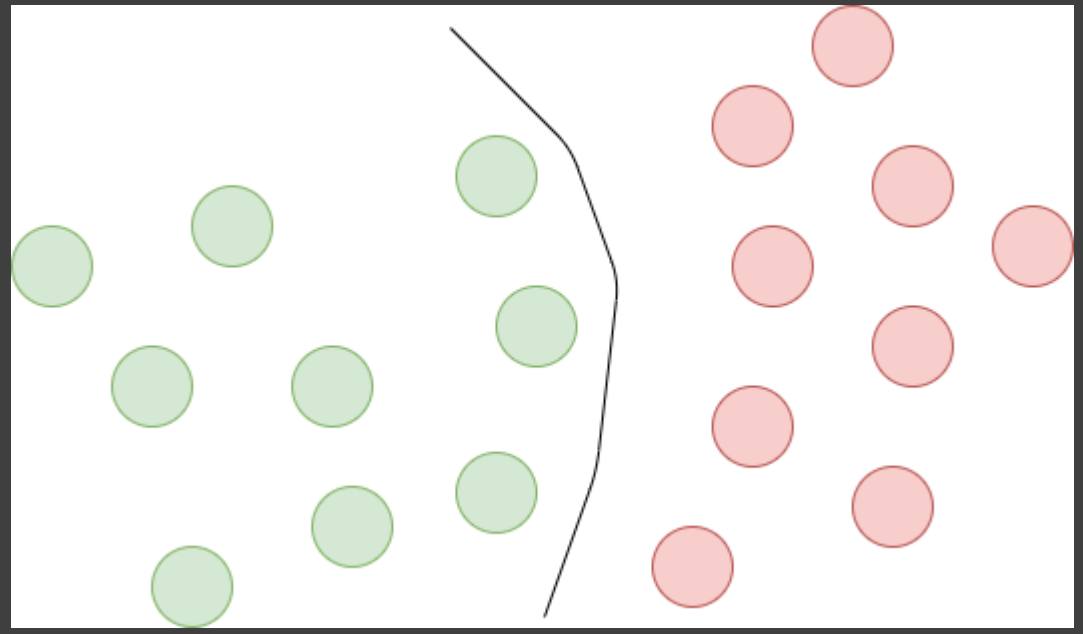
Example - classification

- Since kNN makes local decisions, it’s less affected by nodes far away
- Rocchio uses centroids (can be more affected by outliers)

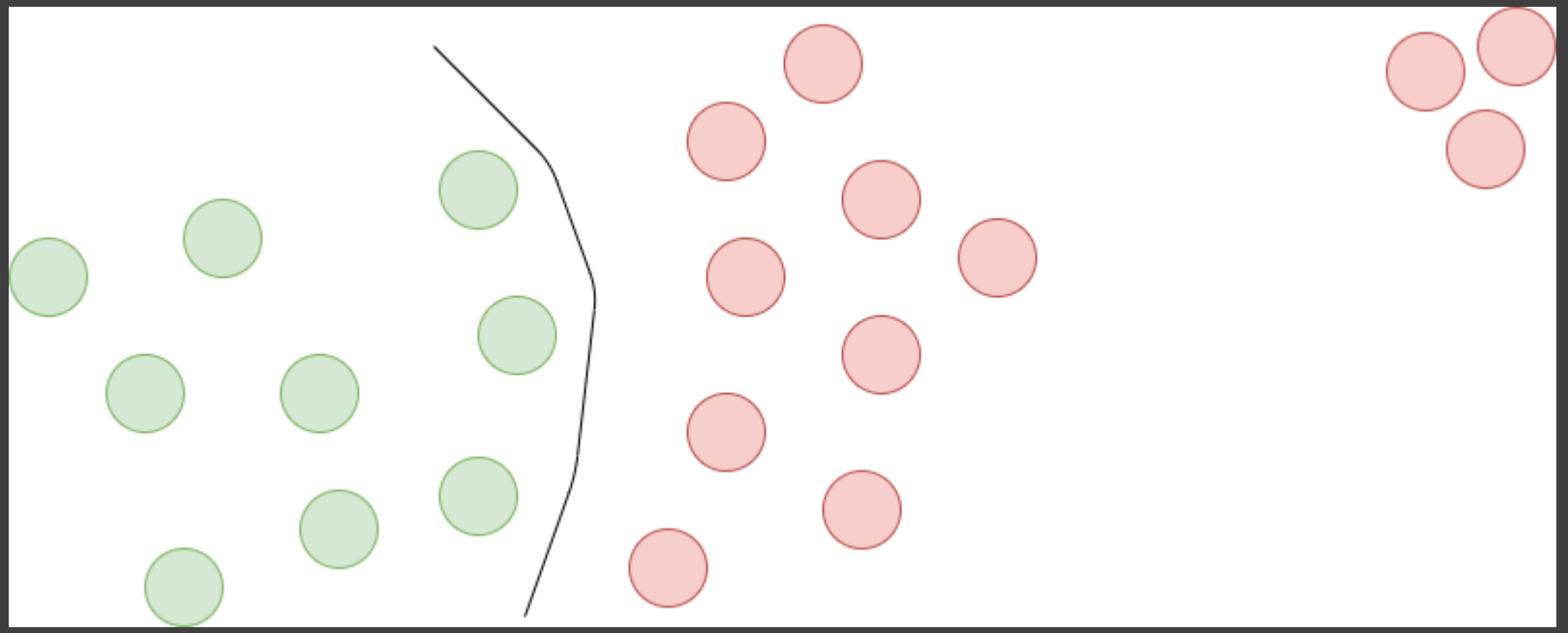
Example - classification



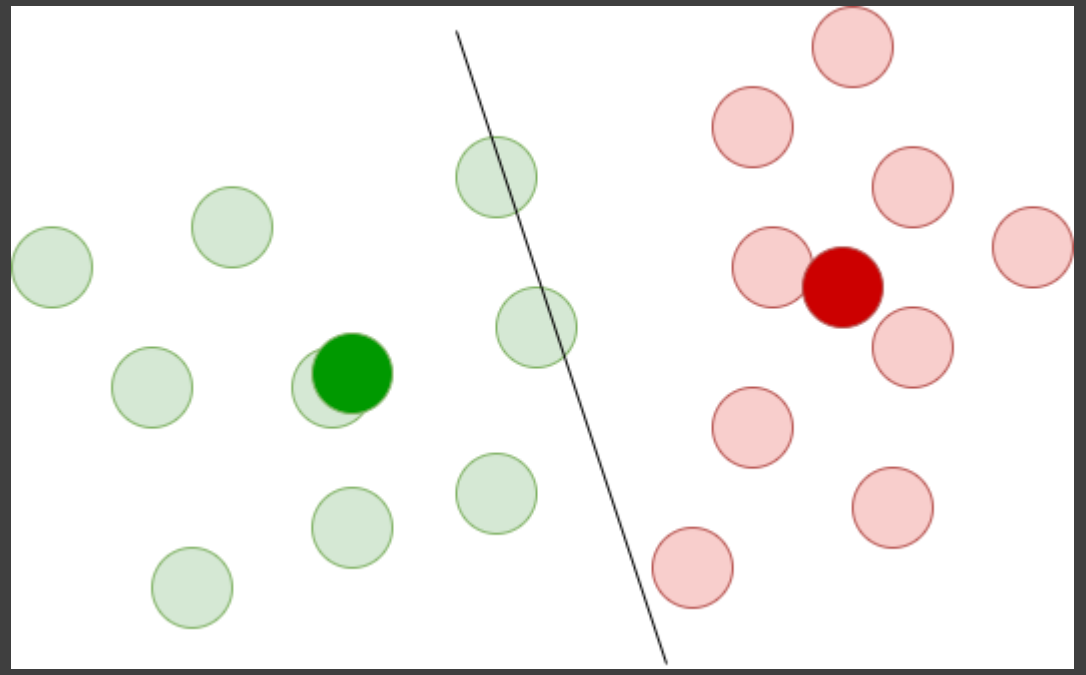
Example - kNN



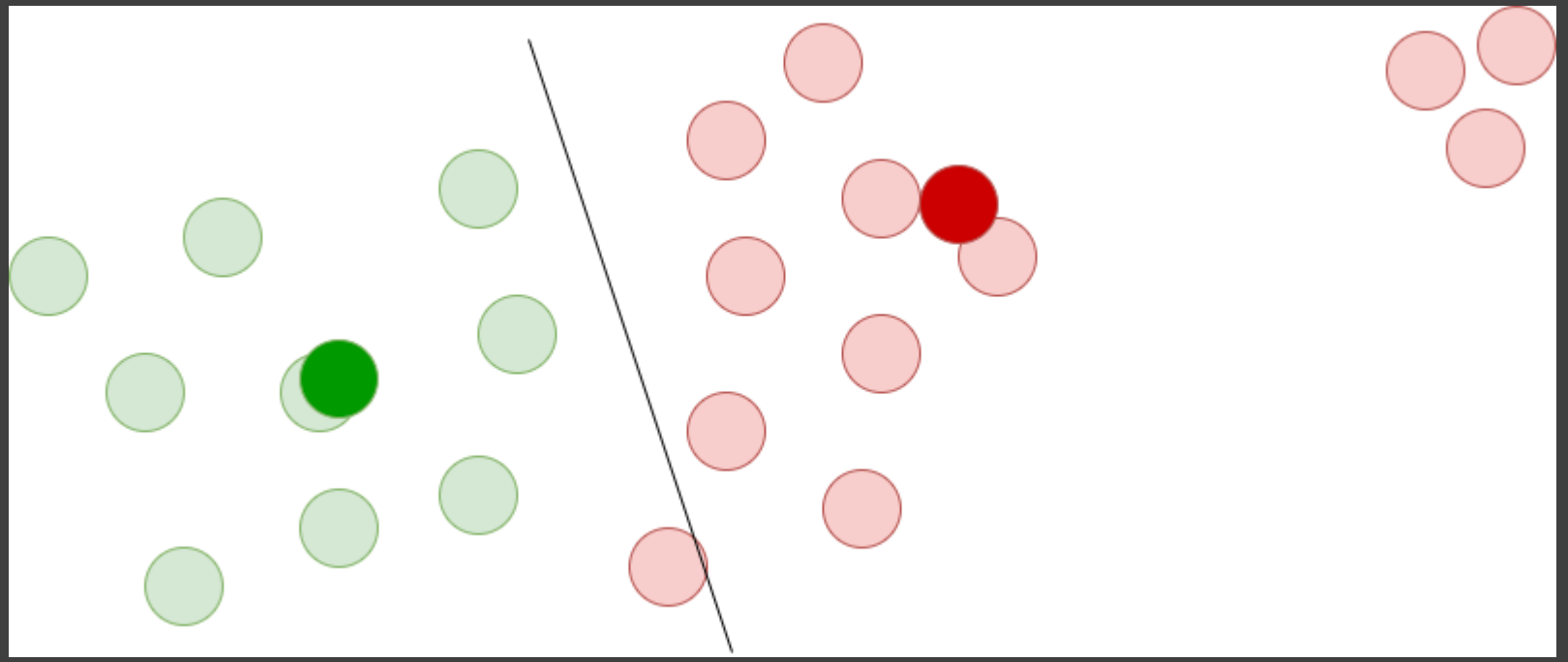
Example - kNN



Example - Rocchio



Example - Rocchio



Agenda

- Introduce Assignment D
- Lecture repetition
  - Rocchio vs. Rocchio (1971)
  - kNN for classification
- Weekly shoutout!
- Individual work

Shoutout of the week

- True Detective S1
- Maybe the best TV series (season) I’ve ever seen

15 min break

Individual work the rest of the session