M1 Internship:

Making KNN algorithms faster with GoldFinger, a Fast and Precise method for approximating Jaccard similarity

Guilhem Niot[1]
Supervised by Anne-Marie Kermarrec[2], Olivier Ruas[3] and François Taiani[4]

[1]Informatique Fondamentale
ENS de Lyon

[2]EPFL, Lausanne, Switzerland

[3]Inria, Lille

[4]IRISA, Rennes

26. April - 23. July 2021

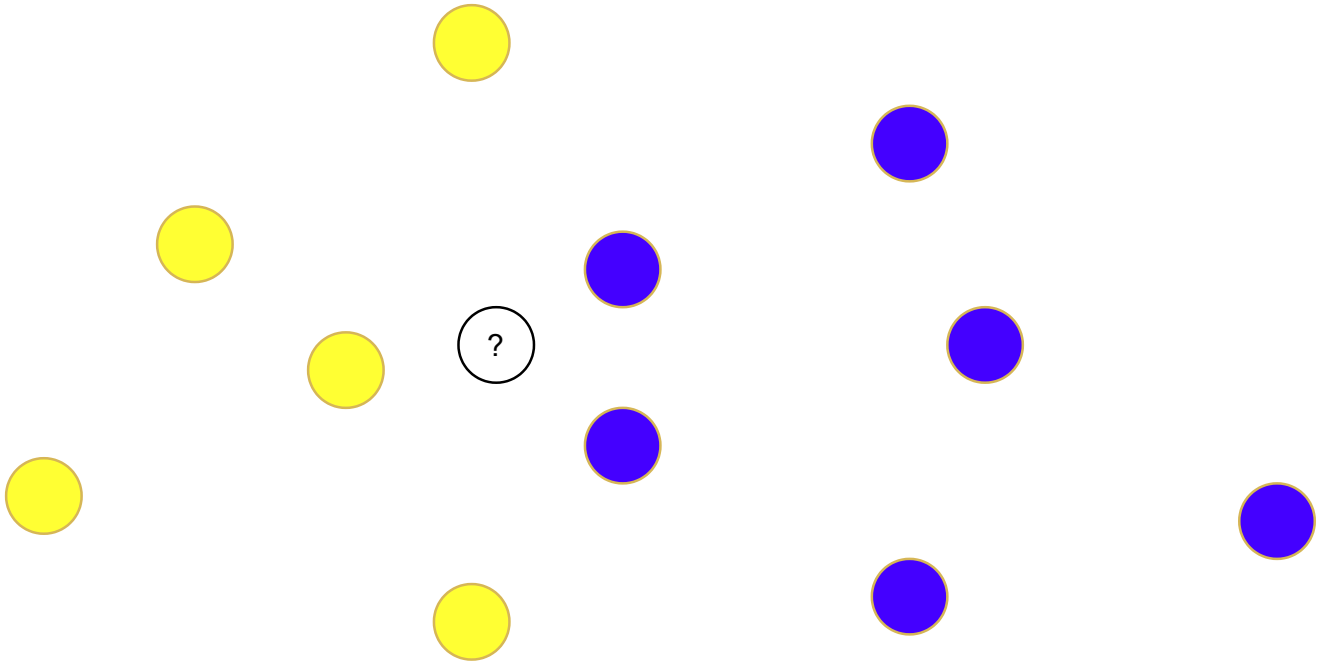
1. K-Nearest Neighbors and Jaccard similarity

K-Nearest Neighbors (KNN)

- invented in the 50's
- consists in retrieving the K-Nearest Neighbors of a point to guess some of its properties
  $\Longrightarrow$ can be used for machine learning models

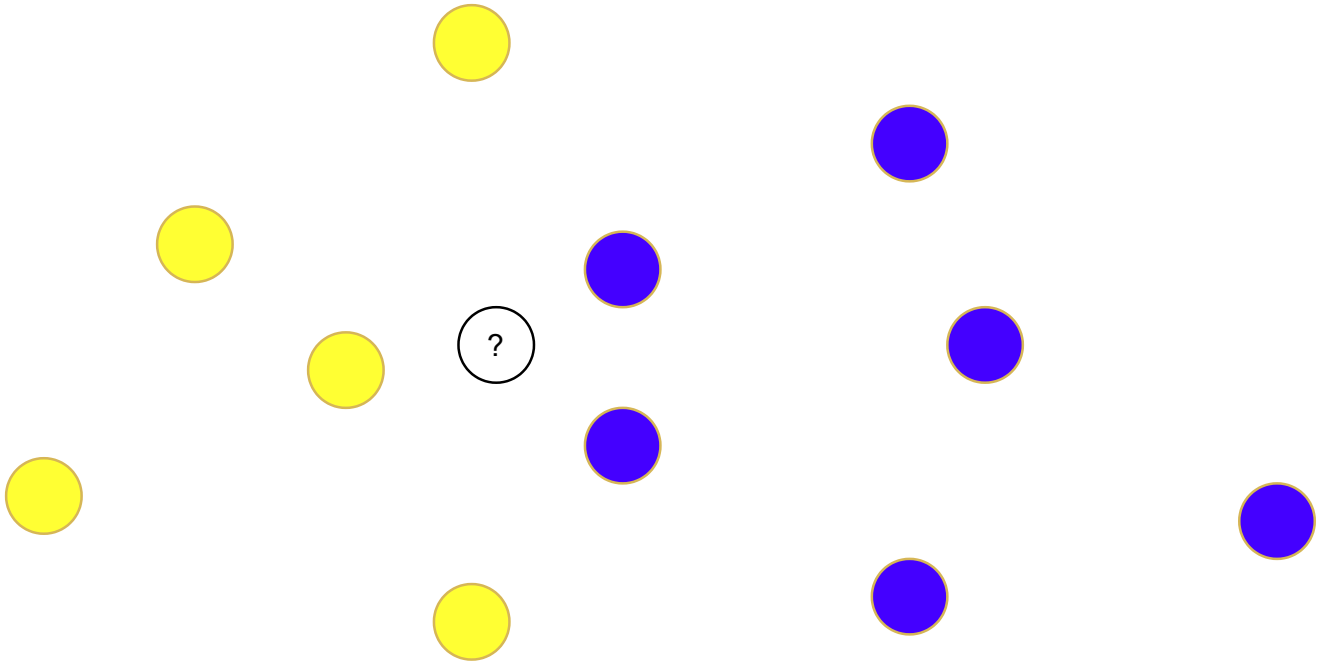
K-Nearest Neighbors (KNN)

Let's assume we want to retrieve the color of the point with an interrogation mark.



K-Nearest Neighbors (KNN)

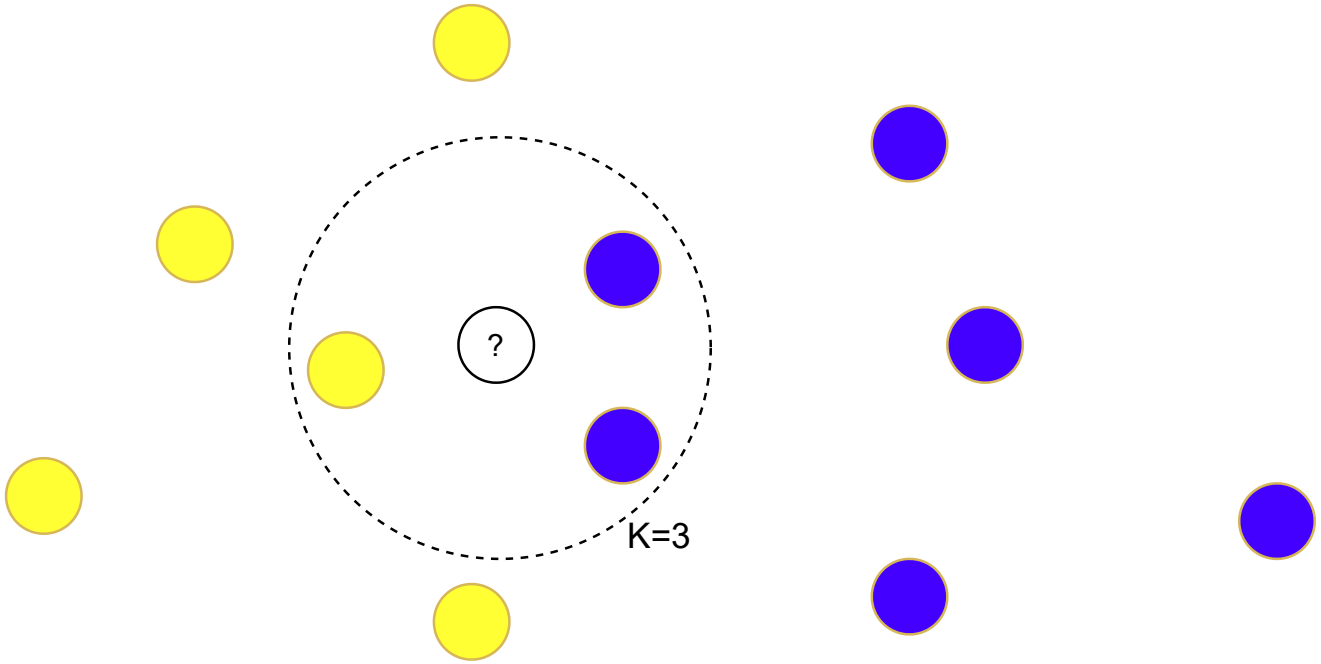
Let's assume we want to retrieve the color of the point with an interrogation mark.



$\Longrightarrow$ Retrieve the K-Nearest Neighbors of the star point, then majority poll.

K-Nearest Neighbors (KNN)

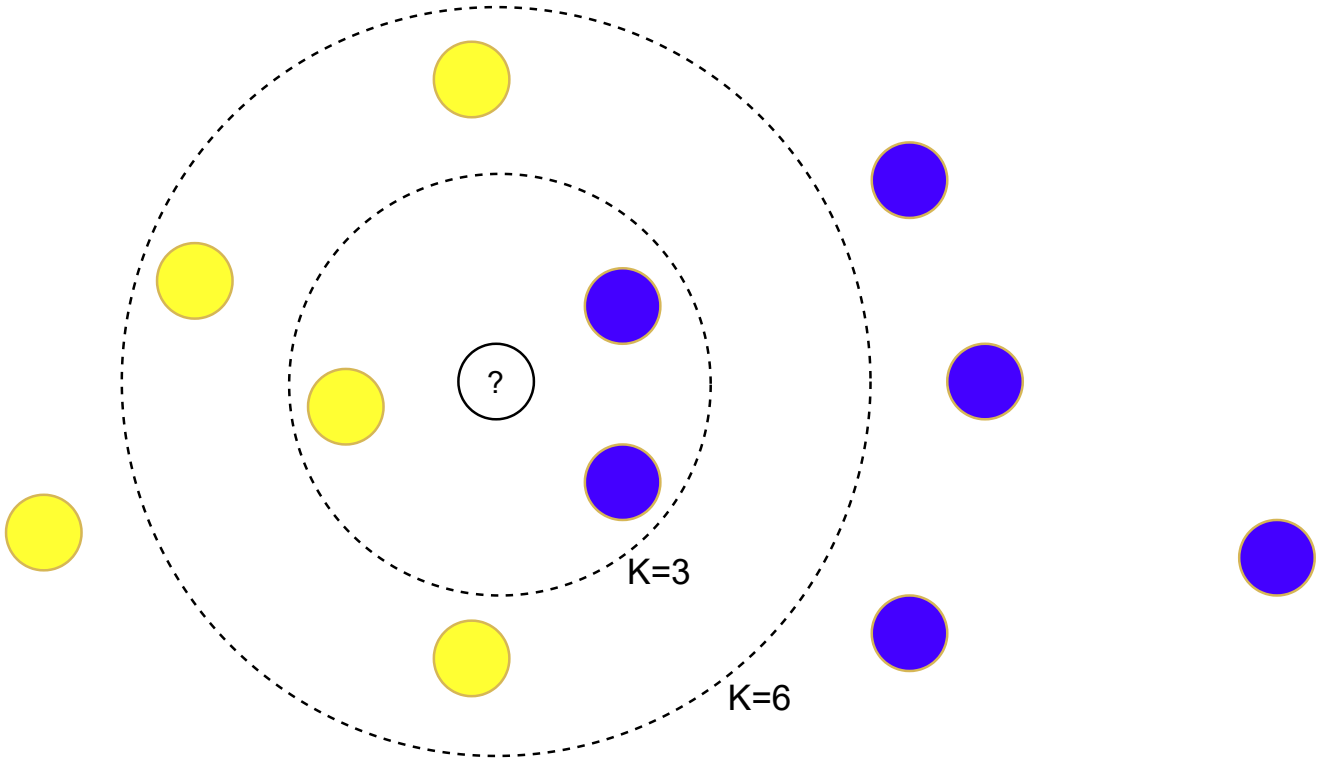
Let's assume we want to retrieve the color of the point with an interrogation mark.



$\Longrightarrow$ Retrieve the K-Nearest Neighbors of the star point, then majority poll. With $K = 3$, we guess *blue*.

K-Nearest Neighbors (KNN)

Let's assume we want to retrieve the color of the point with an interrogation mark.



$\Longrightarrow$ Retrieve the K-Nearest Neighbors of the star point, then majority poll. With $K = 3$, we guess *blue*. With $K = 6$, we guess *yellow*.

Formalization of the KNN problem

How to formalize the "Nearest" property?

Definition (Similarity function)

Given a set of points S, a similarity function is a function $S \times S \to \mathbb{R}^+$.
For dissimilar objects, ≈ 0. For similar objects, takes high values.

Formalization of the KNN problem

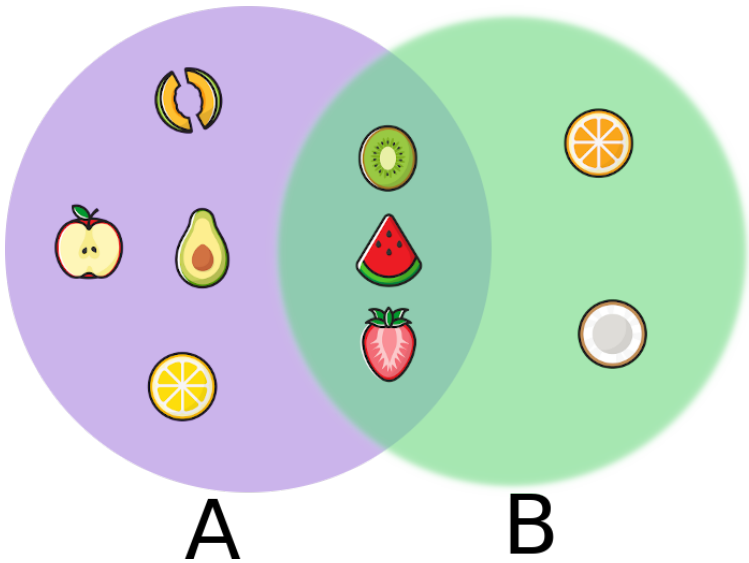
How to formalize the "Nearest" property?

Definition (Similarity function)

Given a set of points S, a similarity function is a function $S \times S \to \mathbb{R}^+$.
For dissimilar objects, ≈ 0. For similar objects, takes high values.

For example, the Jaccard similarity for sets.
Given an items set I and $S = \mathcal{P}(I)$:

$$Jaccard(x_1, x_2) = \frac{|x_1 \cap x_2|}{|x_1 \cup x_2|}$$



Formalization of the KNN problem

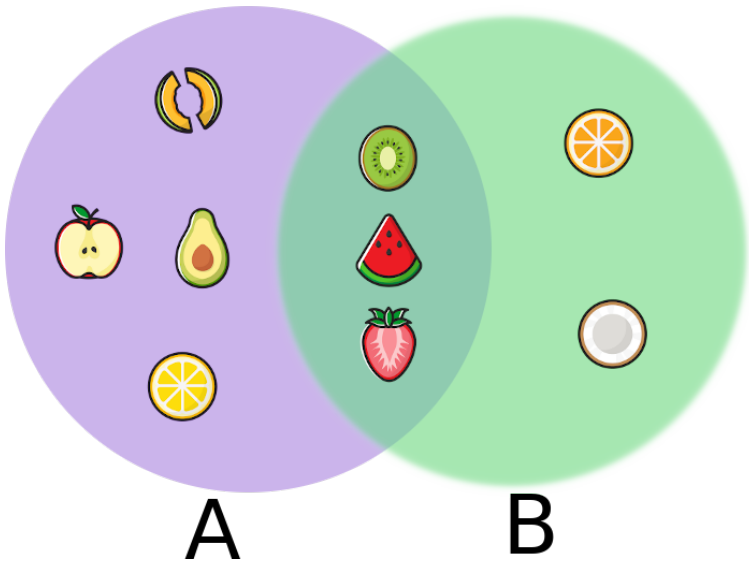
How to formalize the "Nearest" property?

Definition (Similarity function)

Given a set of points S, a similarity function is a function $S \times S \to \mathbb{R}^+$.
For dissimilar objects, ≈ 0. For similar objects, takes high values.

For example, the Jaccard similarity for sets.
Given an items set I and $S = \mathcal{P}(I)$:

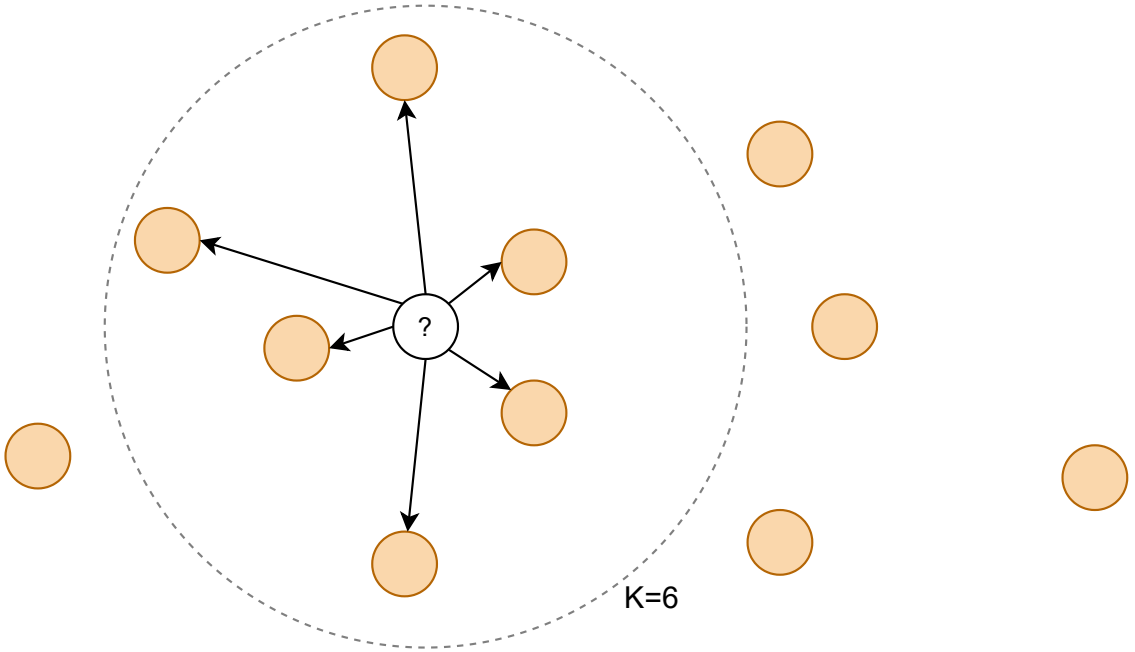
$$\begin{aligned} Jaccard(x_1, x_2) &= \frac{|x_1 \cap x_2|}{|x_1 \cup x_2|} \\ &= \frac{3}{9} \end{aligned}$$



Formalization of the KNN problem

Definition (KNN graph)

Given an integer K, a set of points S and a similarity function sim, we retrieve for **each** point $p \in S$ their K Nearest Neighbors in S, i.e. the K points $q_1, ..., q_K \in S$ with highest values $sim(p, q)$.



Formalization of the KNN problem

Definition (KNN graph)

Given an integer K, a set of points S and a similarity function sim, we retrieve for **each** point $p \in S$ their K Nearest Neighbors in S, i.e. the K points $q_1, ..., q_K \in S$ with highest values $sim(p, q)$.

Definition (KNN queries)

We retrieve the K Nearest Neighbors of **several** points $p \in S$.

2. Approximate K Nearest Neighbors (ANNs)

Exact KNNs are too expensive in practice

Complexity:

- KNN queries: $|S|$ for each query.
- KNN graphs: $|S|^2$.

$\Longrightarrow$ not acceptable in practice ($|S| \gg 10^9$!!)

Exact KNNs are too expensive in practice

Complexity:

- KNN queries: $|S|$ for each query.
- KNN graphs: $|S|^2$.

$\Longrightarrow$ not acceptable in practice ($|S| \gg 10^9$!!)

We only compute Approximate Nearest Neighbors (ANN).
Achieve a trade-off between the "quality" of the neighbors retrieved and the computation time.

Quality of Approximate Nearest Neighbors

What is the "quality" of approximate Nearest Neighbors?

Quality of Approximate Nearest Neighbors

What is the "quality" of approximate Nearest Neighbors?

Definition (Recall)

The **Recall** is the proportion of true nearest neighbors in the output of the approximate algorithm:

$$Recall(ANN) = \frac{|ANN \cap KNN|}{|KNN|}$$

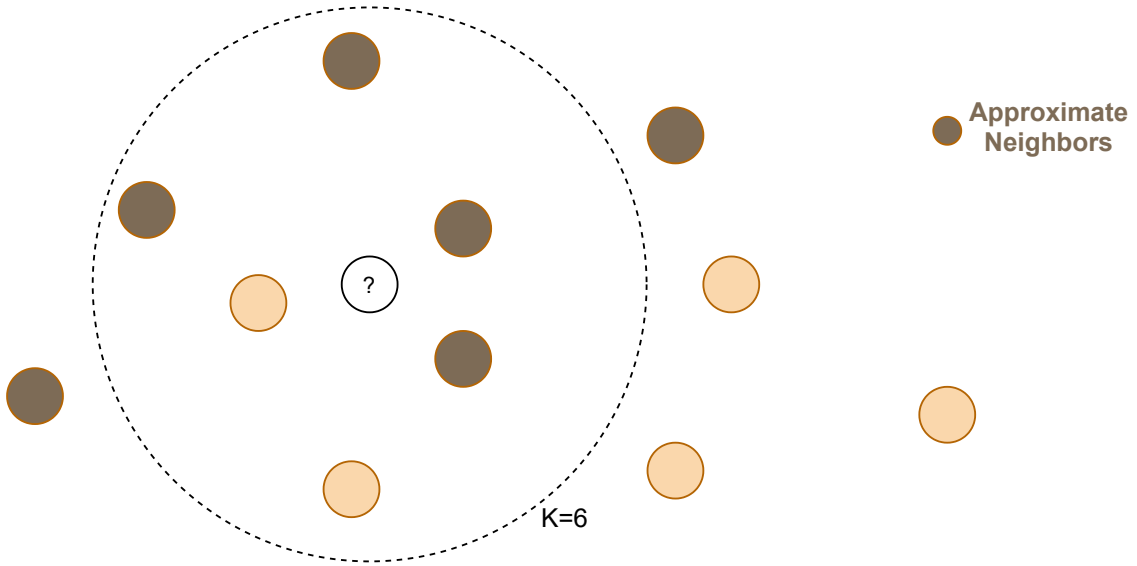
Quality of Approximate Nearest Neighbors

What is the "quality" of approximate Nearest Neighbors?

Definition (Recall)

The **Recall** is the proportion of true nearest neighbors in the output of the approximate algorithm:

$$Recall(ANN) = \frac{|ANN \cap KNN|}{|KNN|}$$



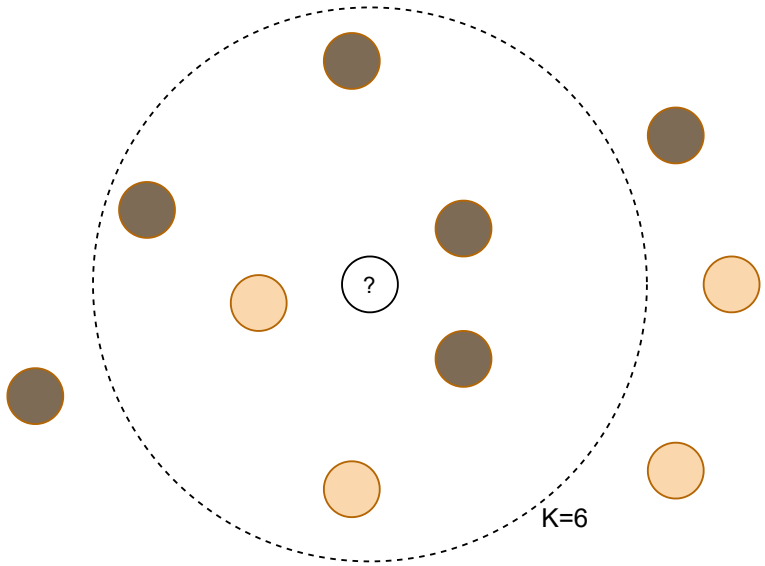
Quality of Approximate Nearest Neighbors

What is the "quality" of approximate Nearest Neighbors?

Definition (Recall)

The **Recall** is the proportion of true nearest neighbors in the output of the approximate algorithm:

$$Recall(ANN) = \frac{|ANN \cap KNN|}{|KNN|}$$



$$Recall = \frac{4}{K} = \frac{4}{6}$$

Quality of Approximate Nearest Neighbors

What is the "quality" of approximate Nearest Neighbors?

Definition (Graph Quality)

The **Quality** of a KNN graph is the ratio of the similarities of the computed neighbors over the similarities of the actual nearest neighbors.

$$Quality(ANN) = \frac{\sum_{(p,q)\in ANN} sim(p,q)}{\sum_{(p,q)\in KNN} sim(p,q)}$$

The closer to 1, the better.

How do we Approximate Nearest Neighbors?

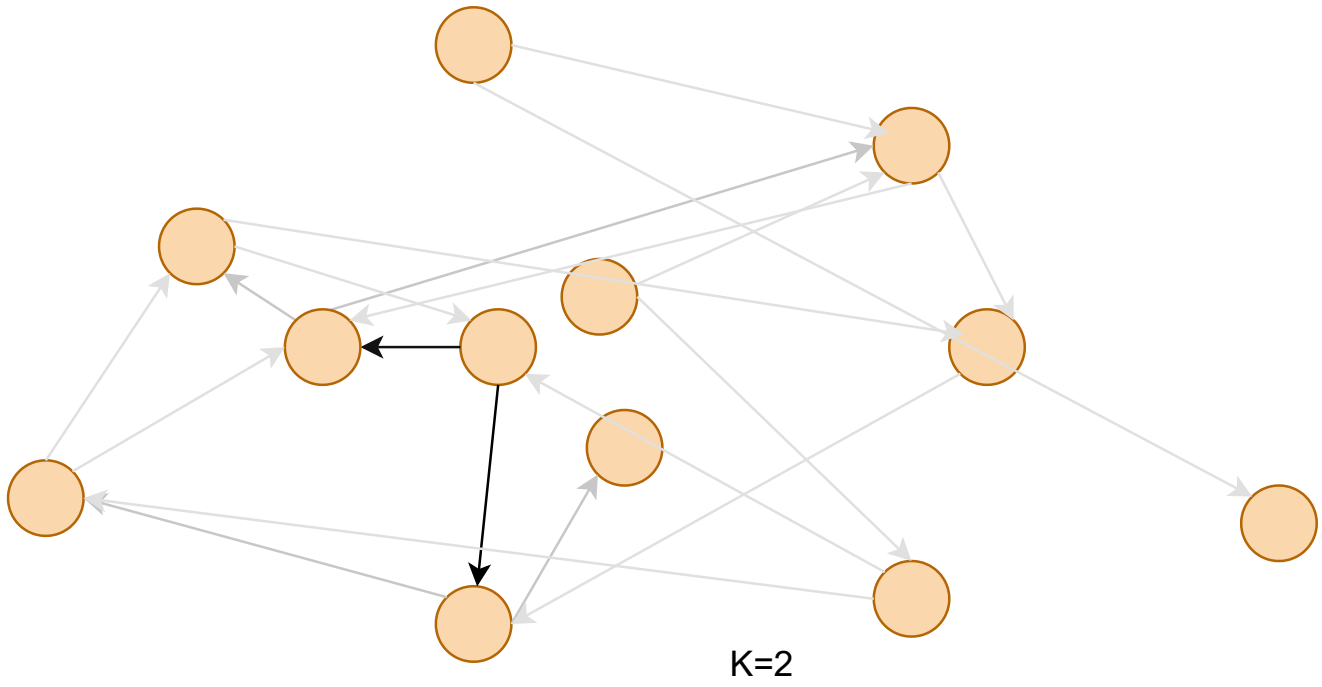
Two main leverages to approximate Nearest Neighbors efficiently:

- Reducing the number of similarities computed.

How do we Approximate Nearest Neighbors?

Two main leverages to approximate Nearest Neighbors efficiently:

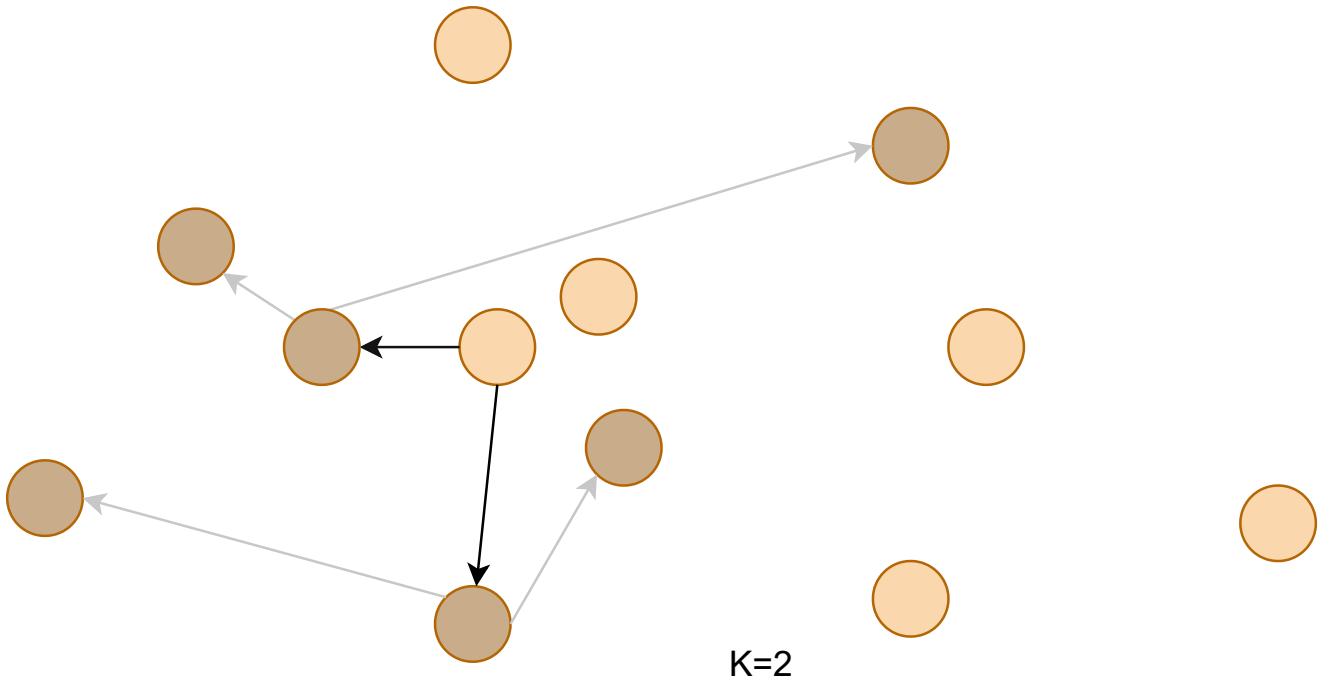
- Reducing the number of similarities computed.
  - For KNN graph computation: using graph heuristics (called NNDescent).
    1. Start with a random graph
    2. Improve it step by step by assuming there likely are closer neighbors at distance at most two in the graph



How do we Approximate Nearest Neighbors?

Two main leverages to approximate Nearest Neighbors efficiently:

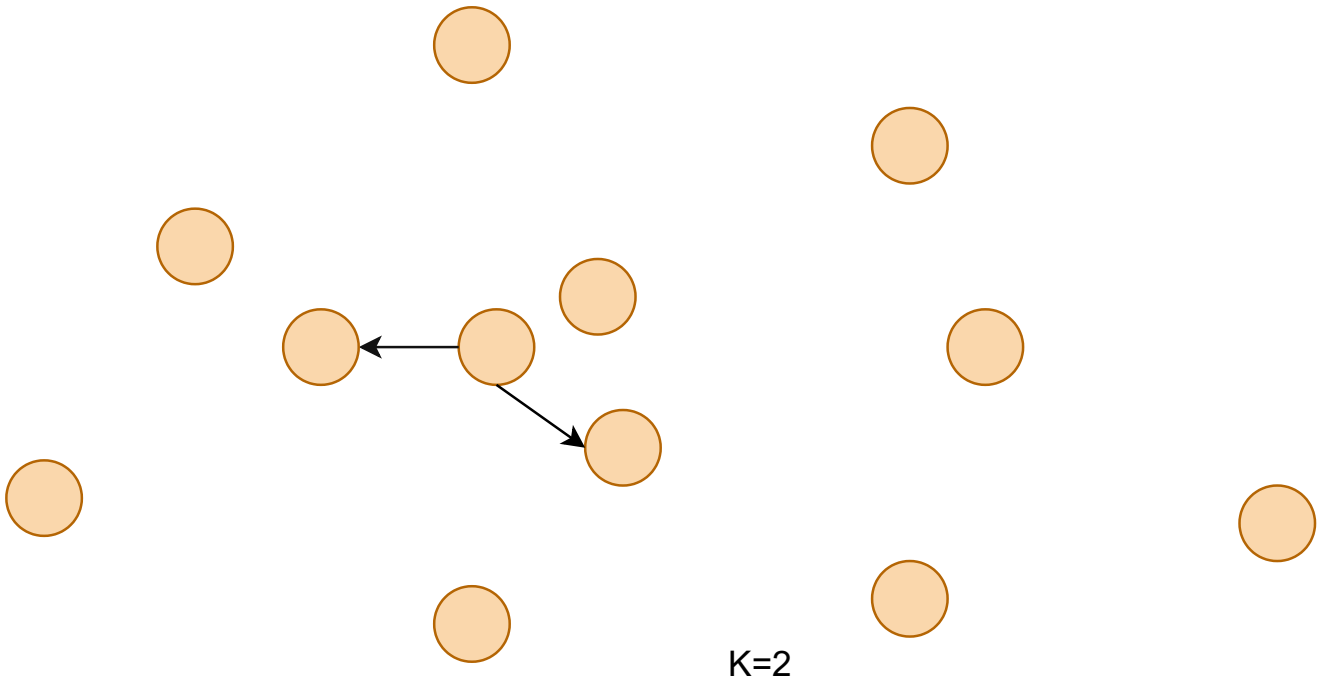
- Reducing the number of similarities computed.
  - For KNN graph computation: using graph heuristics (called NNDescent).
    1. Start with a random graph
    2. Improve it step by step by assuming there likely are closer neighbors at distance at most two in the graph



How do we Approximate Nearest Neighbors?

Two main leverages to approximate Nearest Neighbors efficiently:

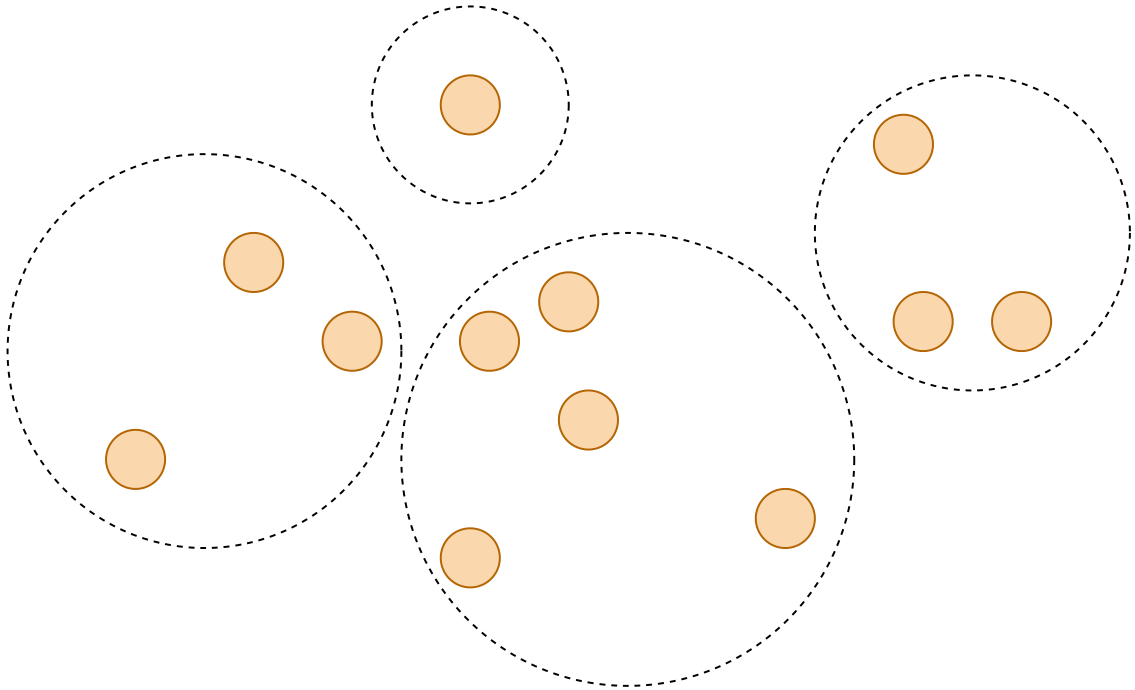
- Reducing the number of similarities computed.
  - For KNN graph computation: using graph heuristics (called NNDescent).
    1. Start with a random graph
    2. Improve it step by step by assuming there likely are closer neighbors at distance at most two in the graph



How do we Approximate Nearest Neighbors?

Two main leverages to approximate Nearest Neighbors efficiently:

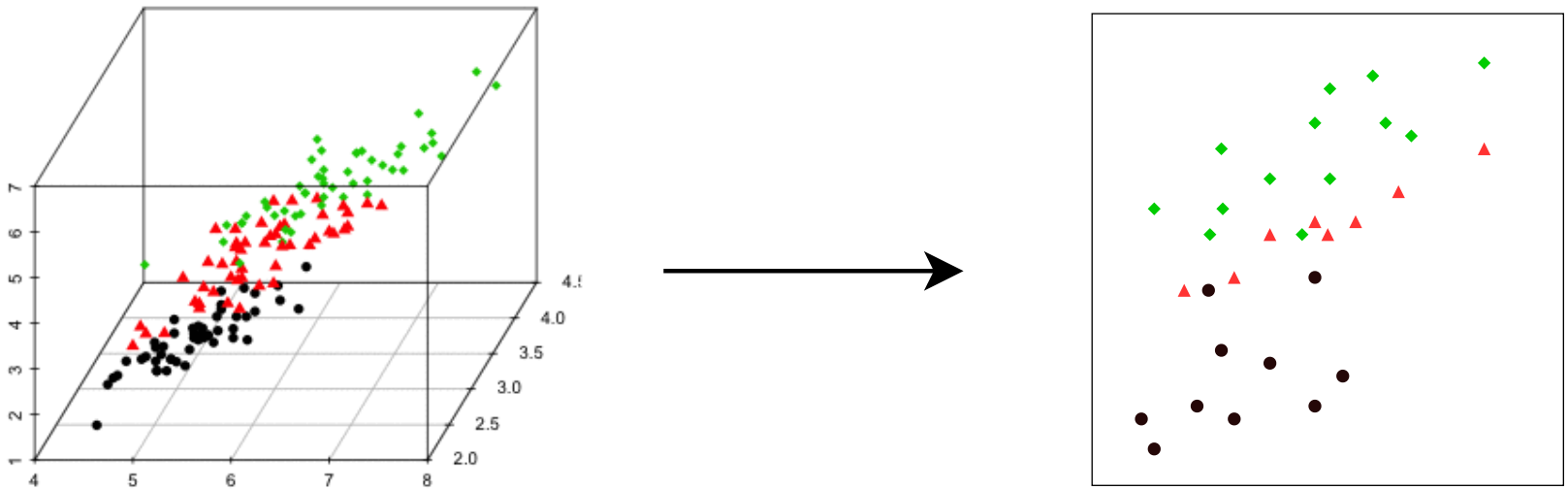
- Reducing the number of similarities computed.
  - For KNN graph computation, we can use graph heuristics.
  - For KNN querying, we can use LSH (Locally-Sensitive Hashing) functions.
    They hash similar items to the same hash with high probability: we can limit our computation to items with the same hash.



How do we Approximate Nearest Neighbors?

Two main leverages to approximate Nearest Neighbors efficiently:

- Reducing the number of similarities computed.
- Approximating the similarity.
  We usually project the points of the dataset onto sketches, i.e. vectors of smaller dimensions with which we can efficiently approximate the similarity of the original objects.



Approximating the Jaccard Similarity

Min-wise hashing: Given a random permutation π of the items set, we have for $X_1, X_2 \in \mathcal{P}(I)$:

$$\mathbb{P}\left(\min_{i \in X_1} \pi(i) = \min_{i \in X_2} \pi(i)\right) = \mathit{Jaccard}(X_1, X_2)$$

Approximating the Jaccard Similarity

Min-wise hashing: Given a random permutation π of the items set, we have for $X_1, X_2 \in \mathcal{P}(I)$:

$$\mathbb{P}\left(\min_{i \in X_1} \pi(i) = \min_{i \in X_2} \pi(i)\right) = Jaccard(X_1, X_2)$$

Approximation of Jaccard by averaging:

$$Jaccard(X_1, X_2) \approx \frac{\sum_{\ell=1}^{n} \mathbb{1}_{\min_{X_1} \pi_\ell = \min_{X_2} \pi_\ell}}{n}$$

With, $sketch(X) = (min_X \pi_\ell)_{1 \leq \ell \leq n}$.

Approximating the Jaccard Similarity

Min-wise hashing: But, random permutations + min computations are costly.
Complexity of $n \times |X|$ for computing the sketch of $|X|$.

$\Longrightarrow$ Fast Similarity Sketching reduces this complexity to $n + |X|$ by filling the sketch on the whole instead of coordinate by coordinate and by stopping early when it is full.

Approximating the Jaccard Similarity

GoldFinger: inspired by the feature hashing technique.
Using a hash function $h : I \to [0, B]$, we produce:

$$sketch(X) = \{h(x), x \in X\}$$

Then,

$$Jaccard(X_1, X_2) \approx GoldFinger(X_1, X_2) = \frac{|sketch(X_1) \cap sketch(X_2)|}{|sketch(X_1) \cup sketch(X_2)|}$$

Approximating the Jaccard Similarity

GoldFinger: inspired by the feature hashing technique.
Using a hash function $h : I \to [0, B]$, we produce:

$$sketch(X) = \{h(x), x \in X\}$$

Then,

$$Jaccard(X_1, X_2) \approx GoldFinger(X_1, X_2) = \frac{|sketch(X_1) \cap sketch(X_2)|}{|sketch(X_1) \cup sketch(X_2)|}$$

Assumes:

- $|X_1 \cap X_2| \approx |sketch(X_1) \cap sketch(X_2)|$
- $|X_1 \cup X_2| \approx |sketch(X_1) \cup sketch(X_2)|$

works well in practice when $|X| \ll B$ (sparse dataset).

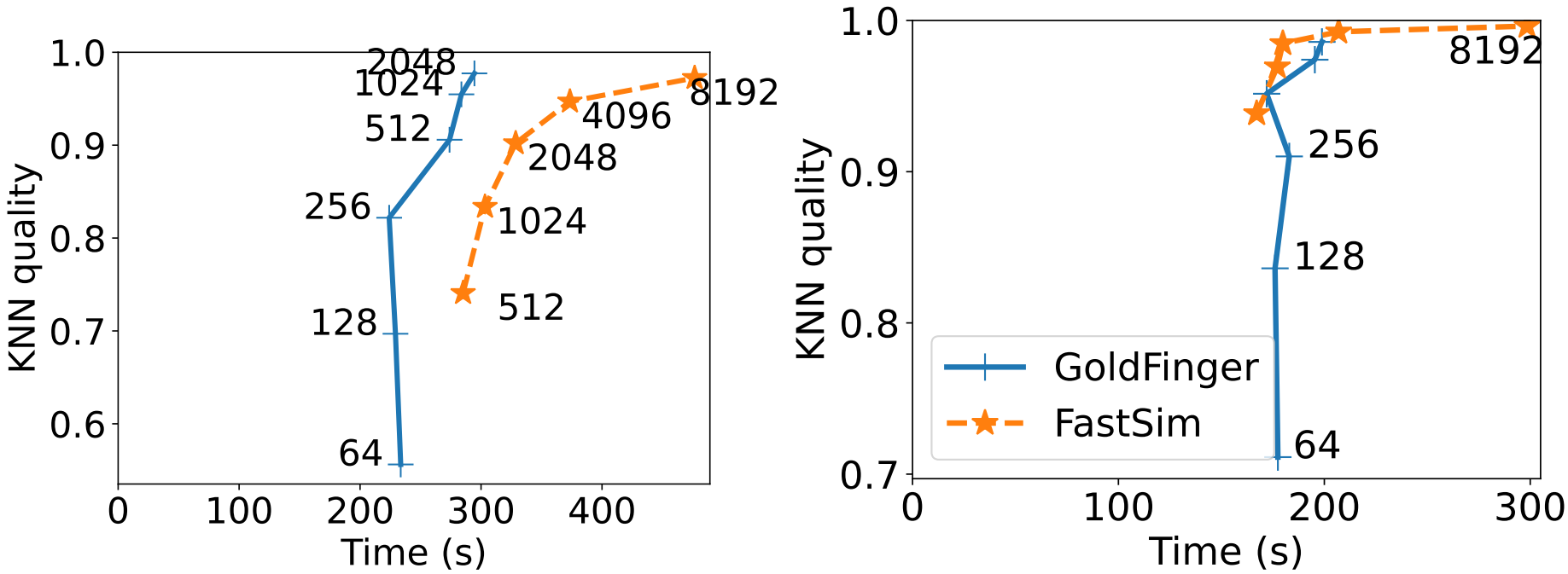
3. GoldFinger against Fast Similarity Sketching for the KNN graph problem

GoldFinger against Fast Similarity Sketching for the KNN graph problem

GoldFinger vs Min-wise hashing: huge advantage for GoldFinger due to the prohibitive preprocessing cost of Min-wise hashing.

GoldFinger against Fast Similarity Sketching for the KNN graph problem

Comparison of GoldFinger performance against state-of-the-art Fast Simarity Sketching for the KNN graph problem.



(a) movielens10M

(b) AmazonMovies

Figure: Relation between the computation time and the KNN quality for different sketch sizes (in number of bits) using Fast Similarity Sketching and GoldFinger.

GoldFinger against Fast Similarity Sketching for the KNN graph problem

- GoldFinger has a clear advantage on Movielens10M.
- Fast Similarity Sketching has a short advantage on Amazon Movies.

GoldFinger is competitive against Fast Similarity Sketching. GoldFinger is more efficient on denser datasets.

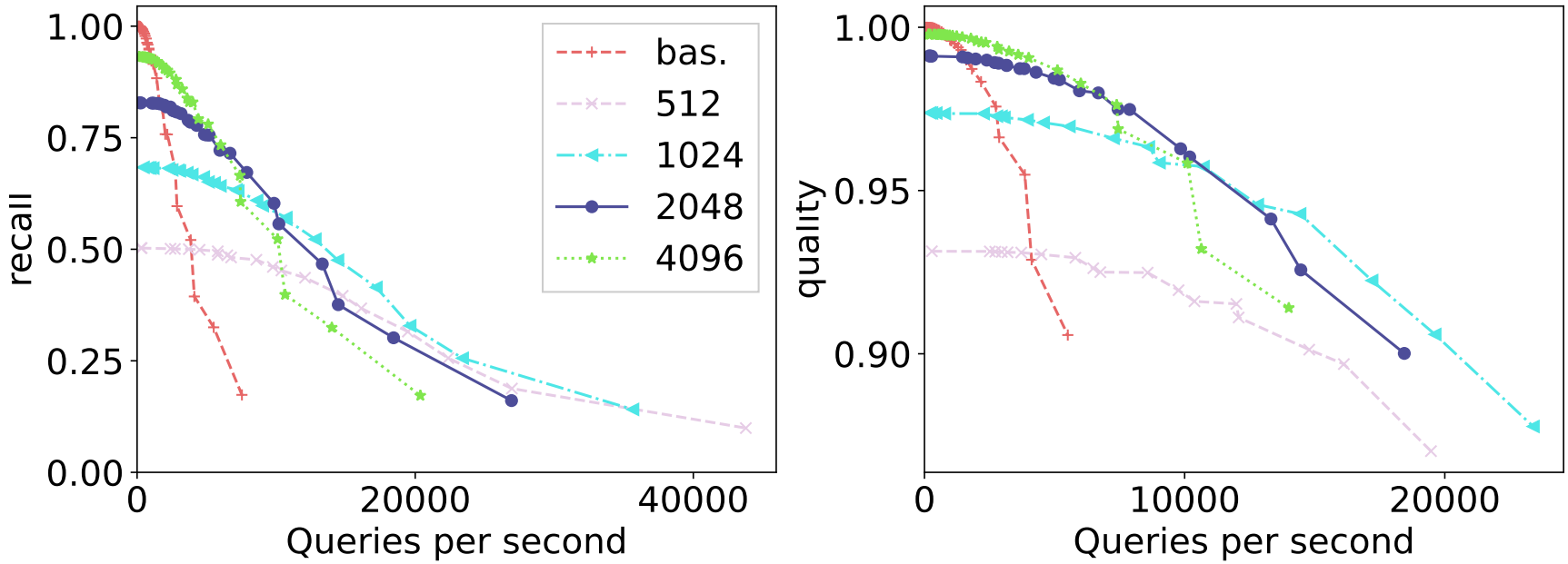
4. Improving KNN querying with GoldFinger

Improving KNN querying with GoldFinger

I also showed that GoldFinger could be used to improve state-of-the-art algorithms for KNN querying.

HNSW is an ANN algorithm used at Amazon and Facebook, that consistently outperforms other algorithms for Jaccard similarity.

Improving KNN querying with GoldFinger



(a) Recall

(b) KNN quality

Figure: Relation between the quality metrics and the number of queries per second on the dataset movielens10M. Using GoldFinger highly increases the number of queries per second at the expense of a slight decrease in quality.

Improving KNN querying with GoldFinger

For a similar quality, GoldFinger provides a significant speedup (up to a factor 4).

5. Conclusion

Conclusion

- GoldFinger is competitive against state-of-the-art sketching technique on the ANN graph problem.
- GoldFinger provides a significant speed-up to state-of-the-art algorithms for answering ANN queries.

Other achievements

- GoldFinger performs better than Fast Similarity Sketching for another problem consisting in retrieving all the pairs (p, q) such that $sim(p, q) \geq threshold$.
- GoldFinger can be significantly improved using adaptative hash functions to adapt to the dataset and avoid harmful collisions.

- Some changes merged to upstream libraries, and usable for future research.
- Great experience with my team, half of the internship on site, and a paper in production.