Image Search with Python

Arif Qodari

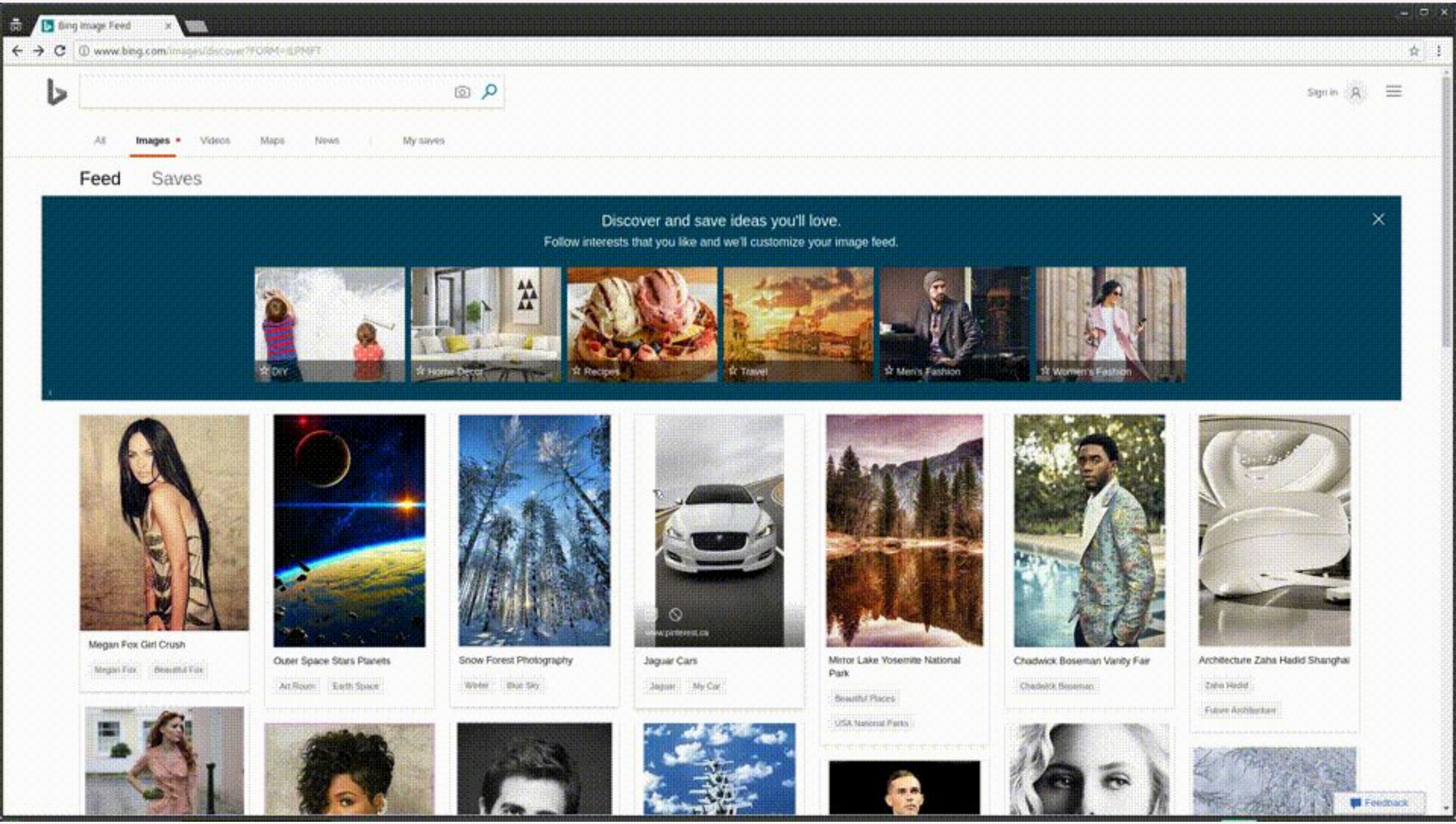
Photo by Toa Heftiba

About me

- Currently Data Scientist at salestock.id
- Previously Ruby on Rails and WordPress dev
- Python and (sometimes) C++
- @arifqodari on Twitter and Github

What is image search engine?

“I have an image, give me similar images.”



How to make images searchable?

Let’s break down into 2 questions:

1. **What kind of features to be used?**
2. **How to search similar features?**

Visual Features

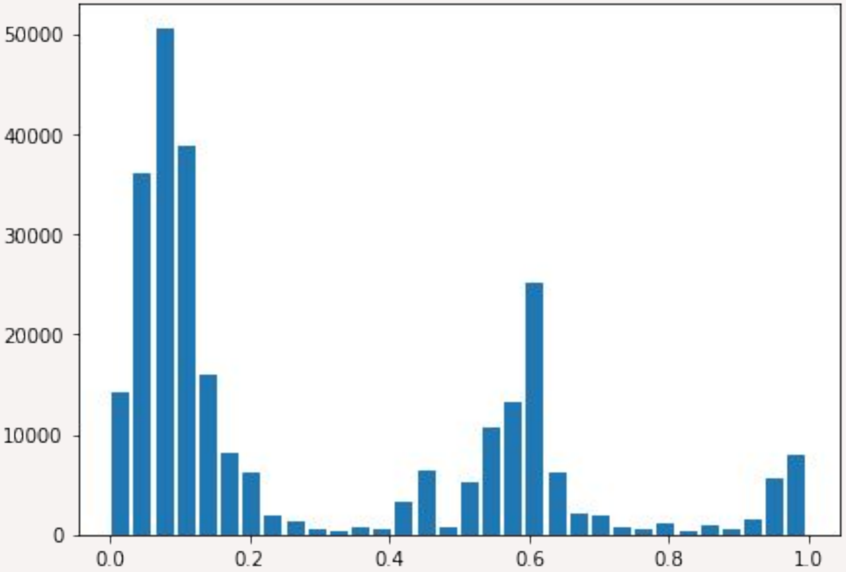
1. **Color composition**, e.g. histogram of color
2. **Shape of objects**, e.g. contour
3. **Texture**, e.g. local binary pattern
4. **Combinations ??**
5. **Deep Learned Features ??**

Basic features extraction pipeline

Features extraction

[0.13, 0.91, 0.62, … ,0.71, 0.42]

1. Histogram of Color (HoC)



How to Construct HoC

- Typically use HSV color instead of RGB
- Define number of bins (or do some experiments!)
- Count number of pixels for each bin in each channel

```
import imageio
import numpy as np
from skimage.color import rgb2hsv

image = imageio.imread("nasipadang.jpg")
hsv_image = rgb2hsv(image)
(hist, bins) = np.histogram(hsv_image[:, :, 0], bins=32)
```

5. FC6 Layer Output as Features

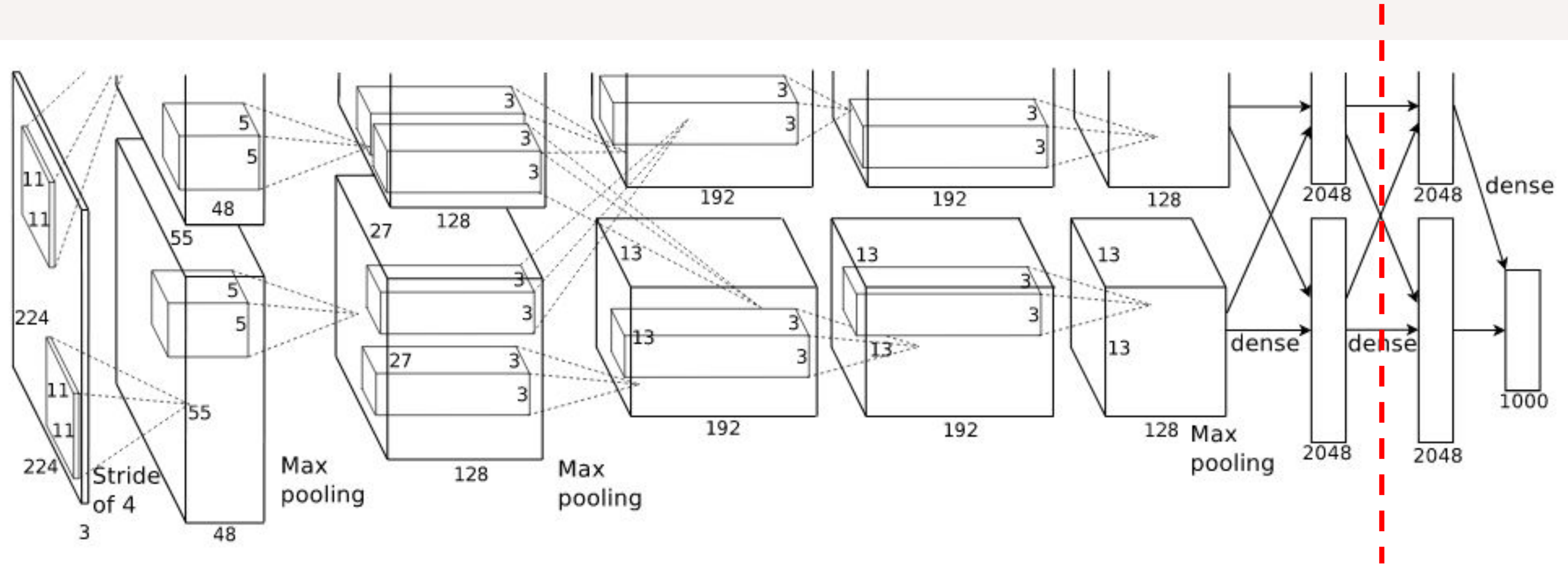


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

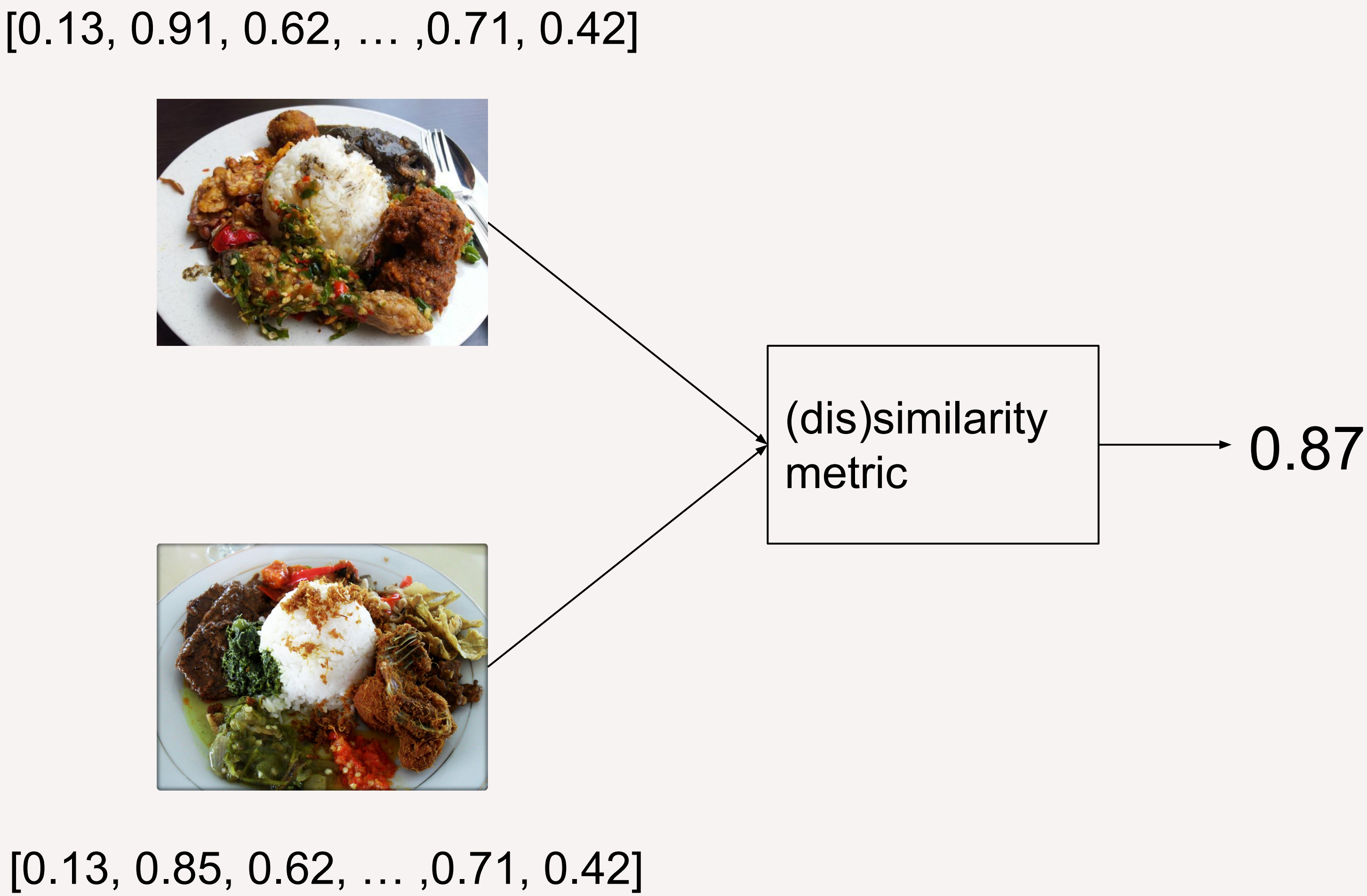
Image from: Krizhevsky et. al.

Let’s break down into 2 questions:

1. **What kind of features to be used?** ✓
2. **How to search similar features?**

(Dis)similarity Metric

- **Dissimilarity,** e.g. Euclidean distance
- **Similarity,** e.g. Cosine similarity


[0.13, 0.91, 0.62, … ,0.71, 0.42]
(dis)similarity metric
0.87
[0.13, 0.85, 0.62, … ,0.71, 0.42]

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

```
import numpy as np

# euclidean distance between vector A and B
# the smaller distance the more similar images
np.linalg.norm(b - a)
```

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

```
import numpy as np

# cosine similarity between vector A and B
# the higher the value the more similar images
np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))
```

… then how to search similar images?

(1) Exhaustive Search

Imagine performing a linear scan (*O(n)*) on your MySQL table every time someone hit the search button.

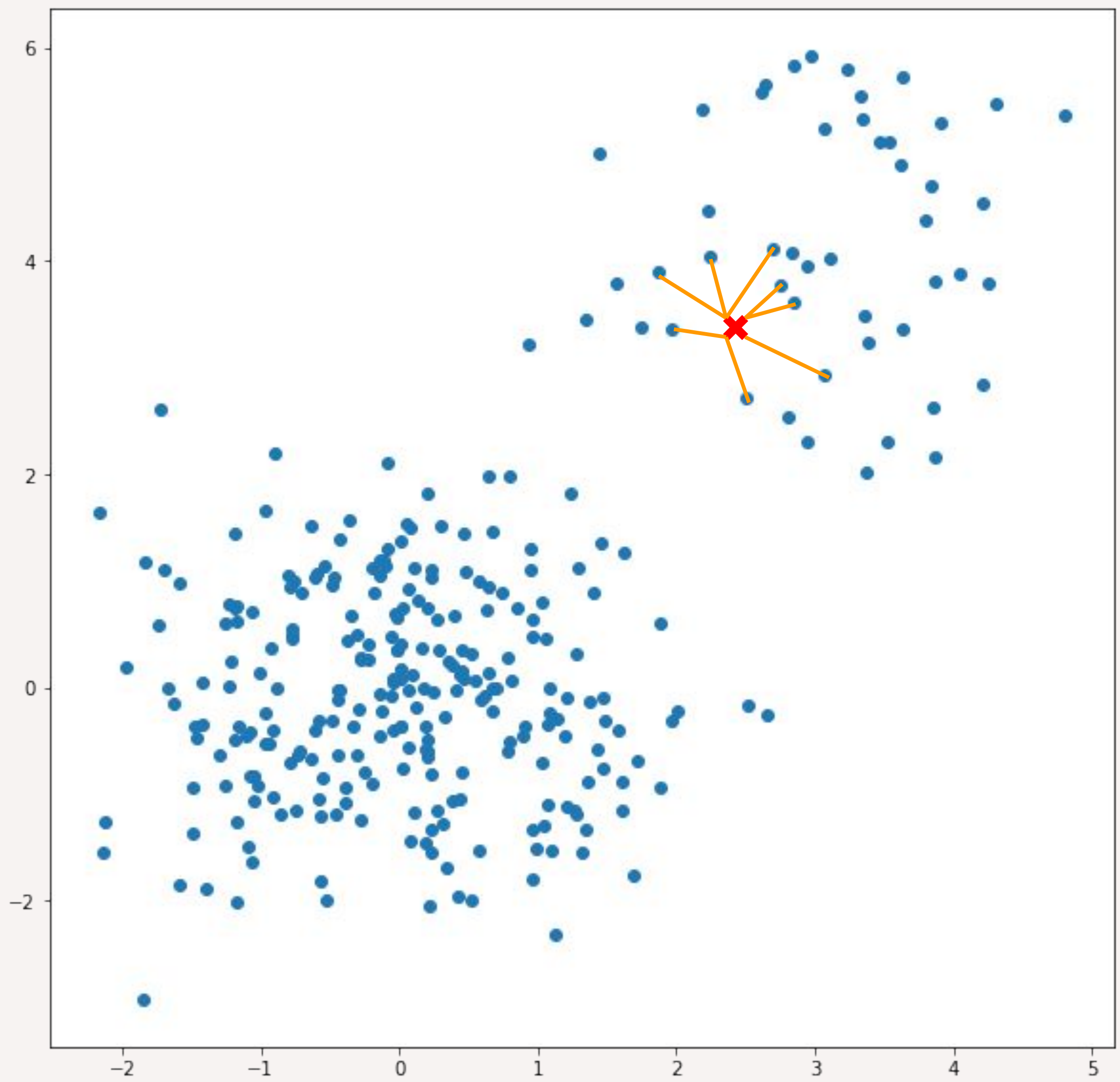
image_id	f1	f2	f3	f4	f5
1	0.340	0.329	0.871	0.261	0.730
2	0.041	0.042	0.570	0.833	0.569
3	0.924	0.966	0.453	0.154	0.846
4	0.534	0.499	0.006	0.555	0.369
5	0.277	0.594	0.753	0.540	0.736
6	0.898	0.554	0.241	0.993	0.353
7	0.019	0.883	0.065	0.795	0.676
8	0.643	0.906	1.000	0.901	0.954
9	0.314	0.399	0.953	0.493	0.851
10	0.517	0.914	0.668	0.101	0.101

```
-- input [0.340 0.329 0.871 0.261 0.730]
SELECT
     image_id,
     SQRT(POWER((0.340 - f1), 2)
          + POWER((0.329 - f2), 2)
          + POWER((0.871 - f3), 2)
          + POWER((0.261 - f4), 2)
          + POWER((0.730 - f5), 2))
     AS distance
FROM db_features
ORDER BY 2
```

It works but ...


TOO SLOW

(2) Better Approach: Nearest Neighbor



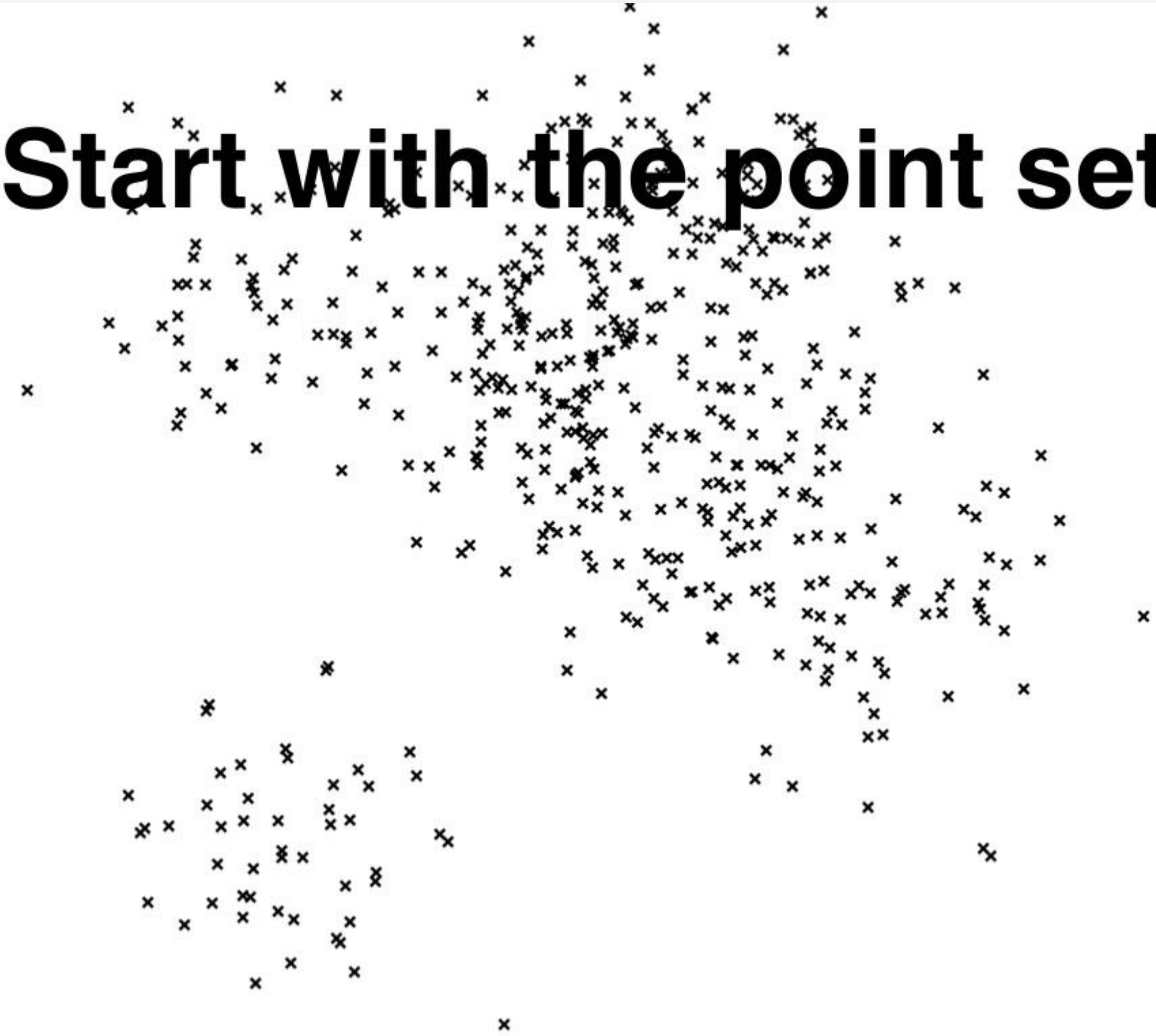
annoy

Library for approximate nearest neighbor. Written in C++ with Python bindings. Based on forest of binary trees.

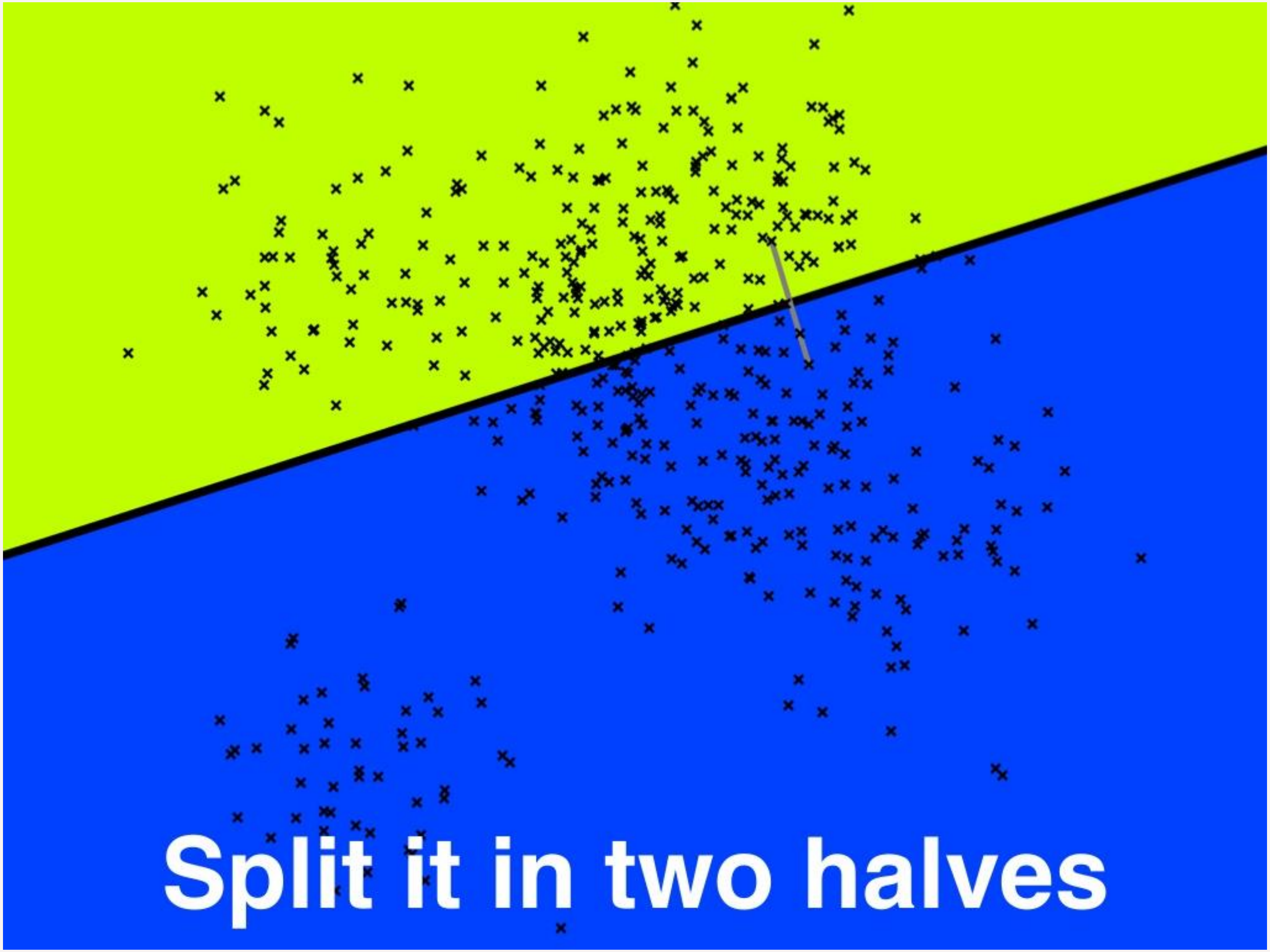
https://github.com/spotify/annoy/

Alternatives: kd-tree, KGraph, Faiss, NMSLib

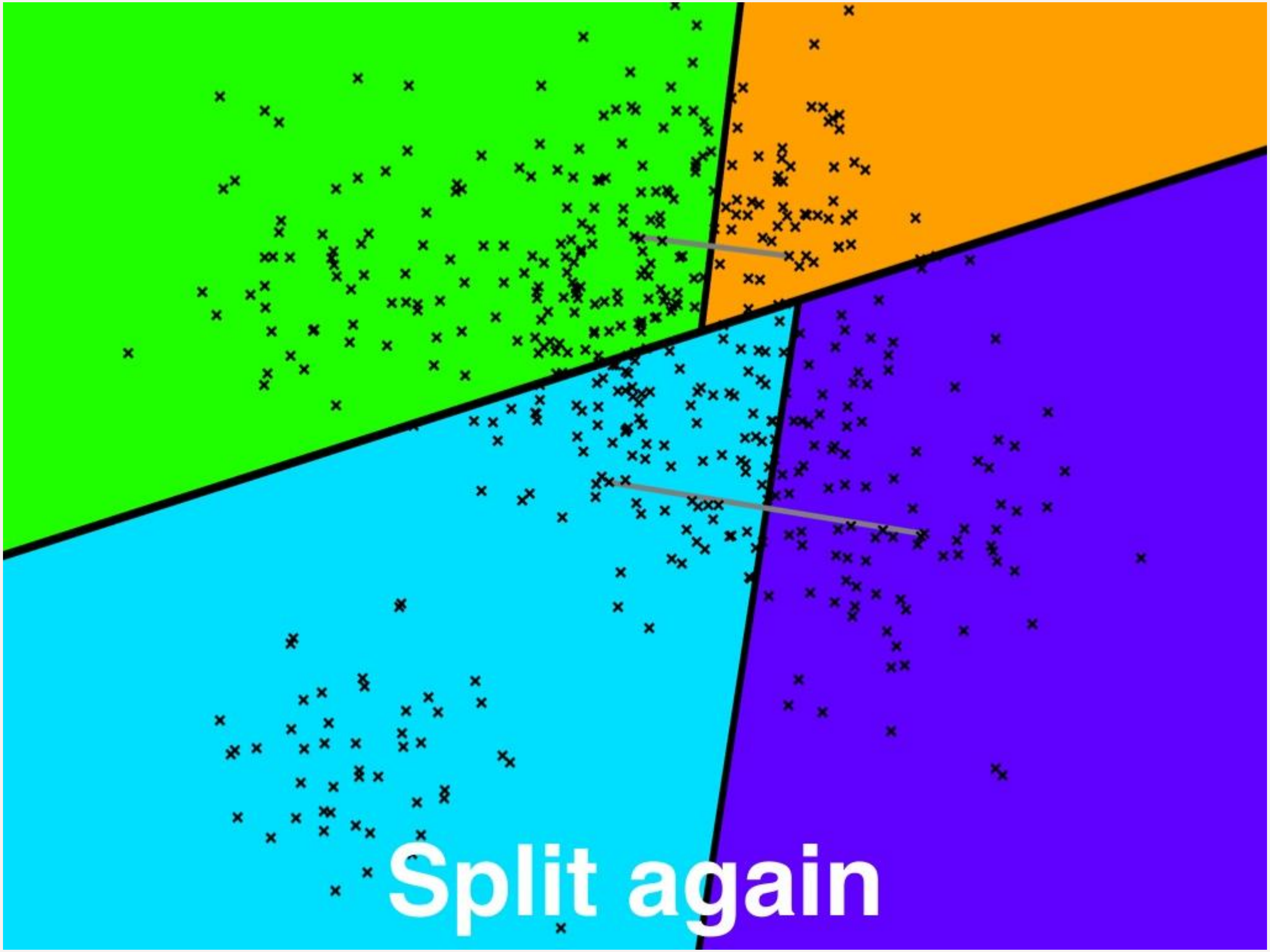
Start with the point set



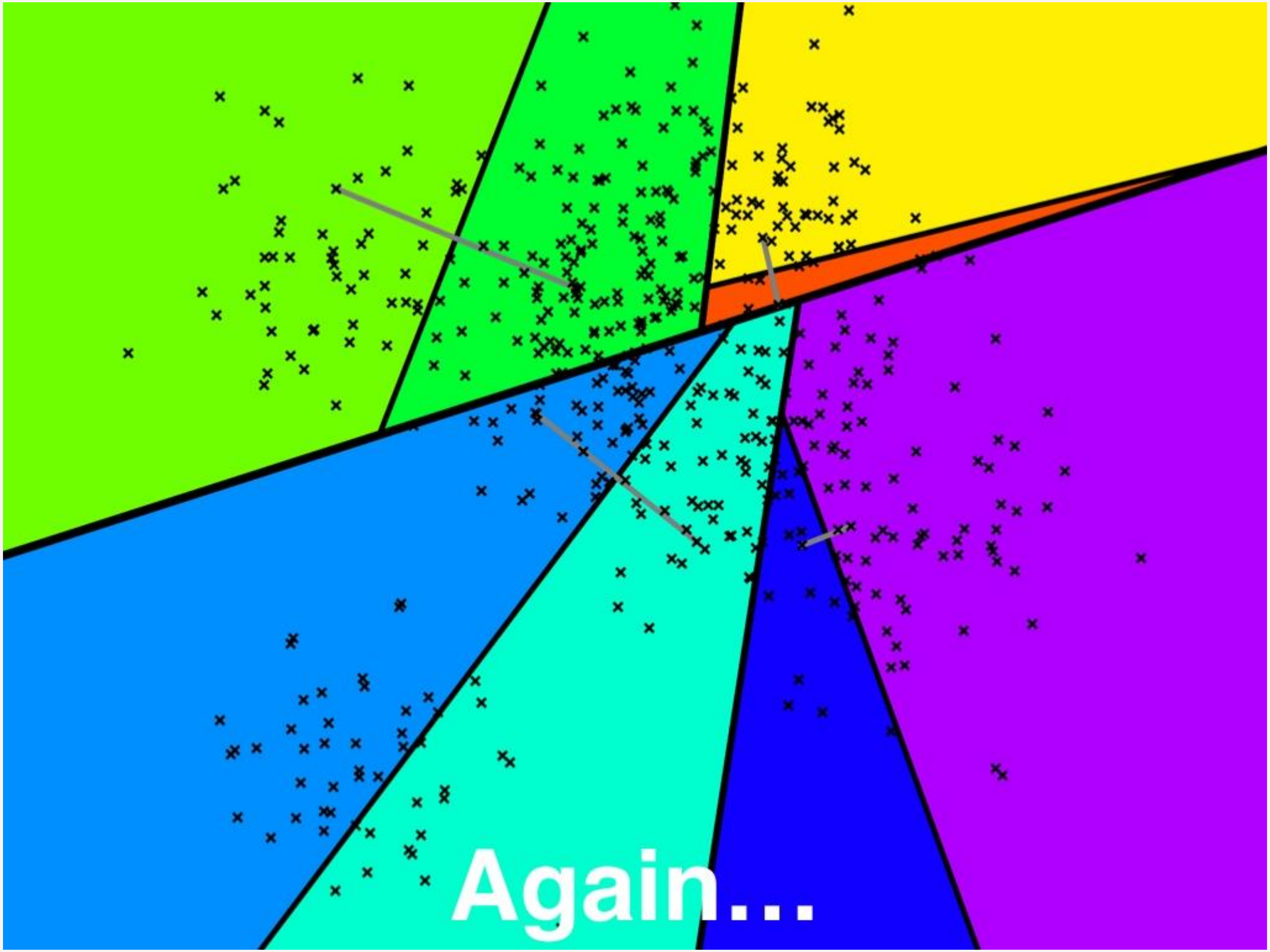
Slide from: Erik Bernhardsson



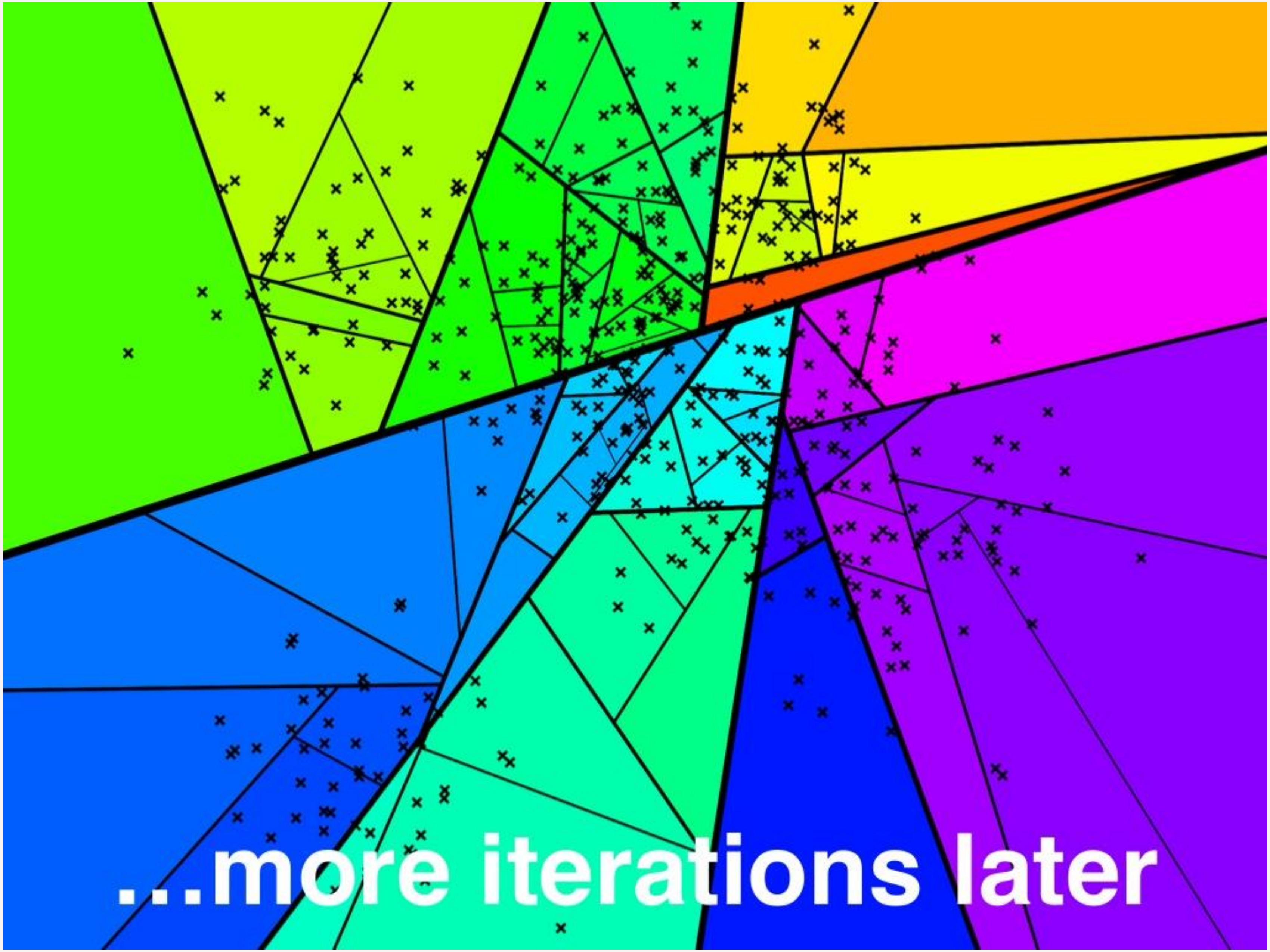
Slide from: Erik Bernhardsson



Slide from: Erik Bernhardsson

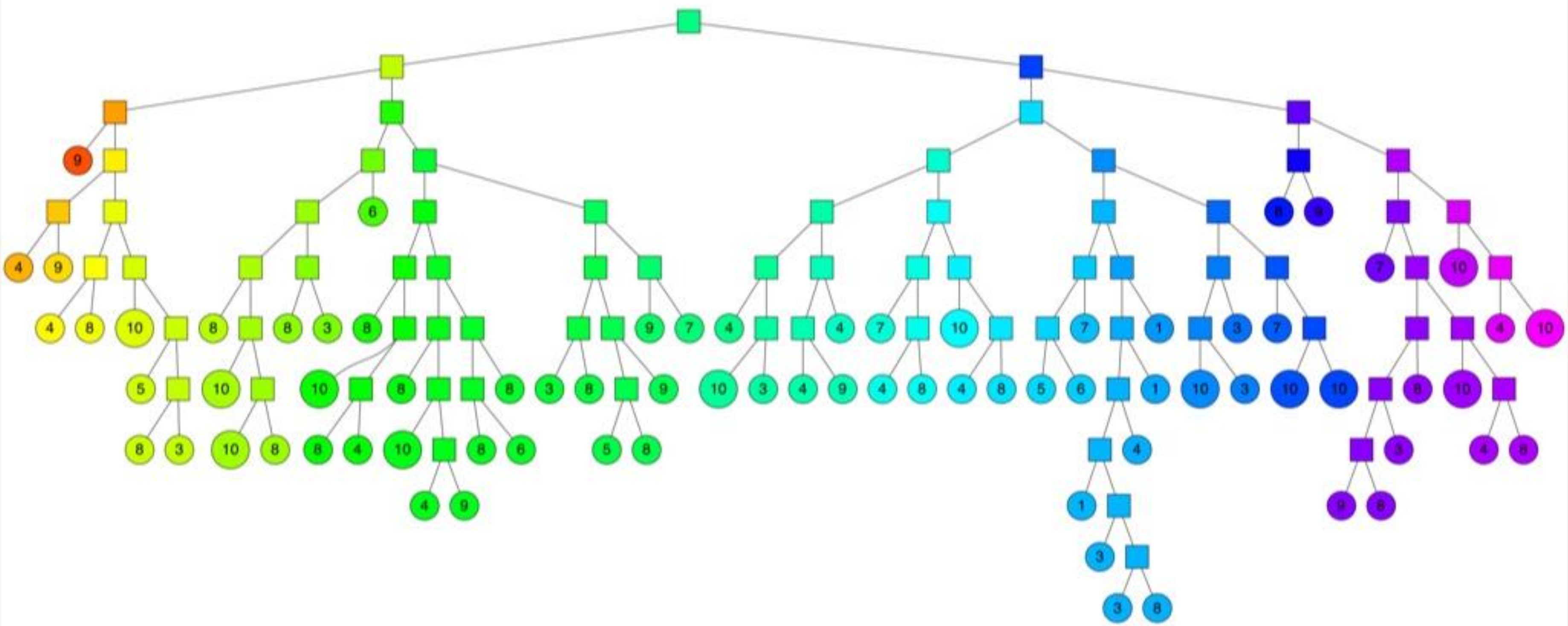


Slide from: Erik Bernhardsson



Slide from: Erik Bernhardsson

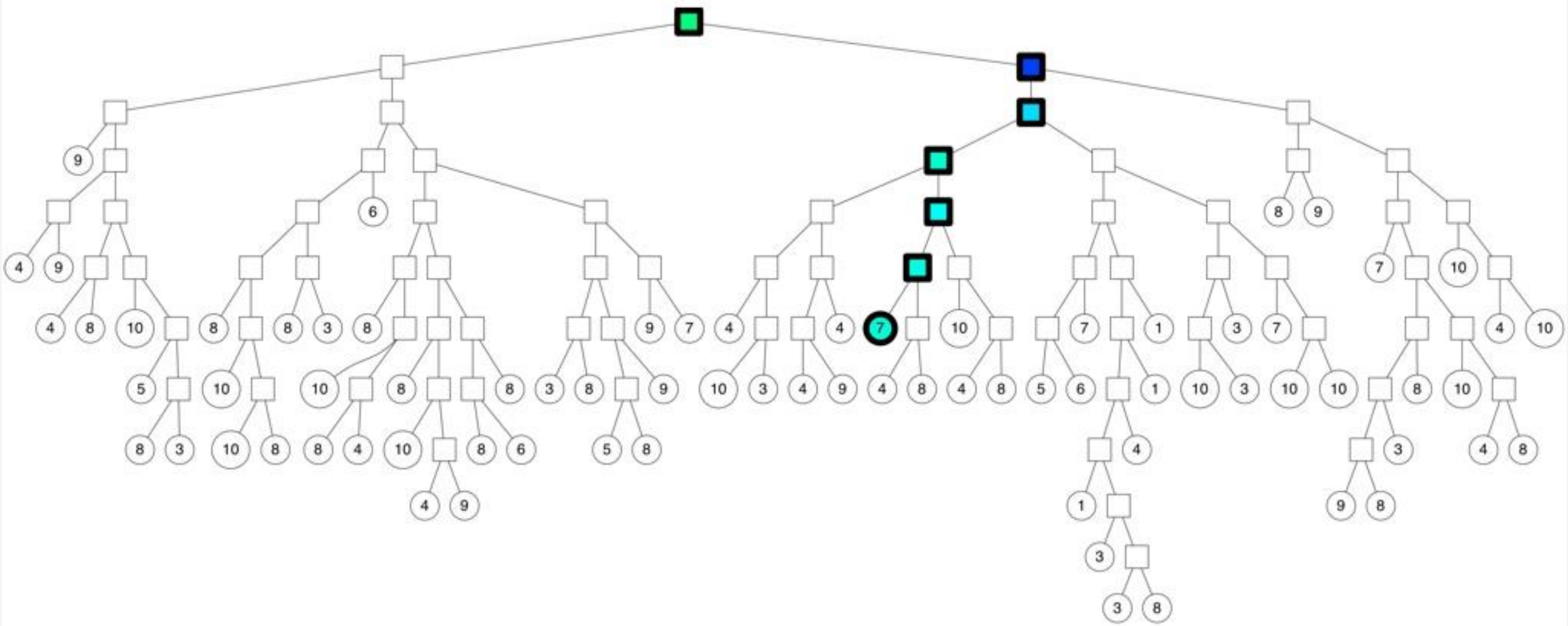
Binary tree



Slide from: Erik Bernhardsson

```
from annoy import AnnoyIndex

# init index with 40 dimension features
ann_index = AnnoyIndex(40)

# add data
for i, features in enumerate(db_features):
    ann_index.add_item(i, features)

# perform indexing with 10 trees
ann_index.build(10)

# save the resulting index into a bin file
ann_index.save('test.ann')
```

Searching the tree



Slide from: Erik Bernhardsson

```
from annoy import AnnoyIndex

ann_index = AnnoyIndex(40)

# load saved index
ann_index.load('test.ann')

# return 10 nearest image indices
ann_index.get_nns_by_vector(query_features, 10)
```

Let’s break down into 2 questions:

1. **What kind of features to be used?** ✓
2. **How to search similar features?** ✓

OK Cool! Then what?

Clone project:

https://github.com/arifqodari/simple-image-search

Download Sample Dataset:

http://bitly.com/holiday-small

Take home questions:

1. **What features that important?**
2. **How to search efficiently and effectively?**
3. **When to use Euclidean distance vs Cosine Similarity?**
4. **How to handle huge dimension?**

“Searching is Easy, Finding is Not”


T. HANKS