

# QUERY EMBEDDINGS:

## WEB SCALE SEARCH POWERED BY DEEP LEARNING AND PYTHON

---

**Ankit Bahuguna**

**Software Engineer (R&D), Cliqz GmbH**

[ankit@cliqz.com](mailto:ankit@cliqz.com)

## ABOUT ME

- ▶ Software Engineer (R&D), **CLIQZ GmbH**.
- ▶ Building a **web scale search engine**, optimized for *German* speaking community.
- ▶ **Areas:** Large scale Information Retrieval, Machine Learning, Deep Learning and Natural Language Processing.
- ▶ **Mozilla Representative (2012 - Present)**



**Ankit Bahuguna**  
(@codekee)

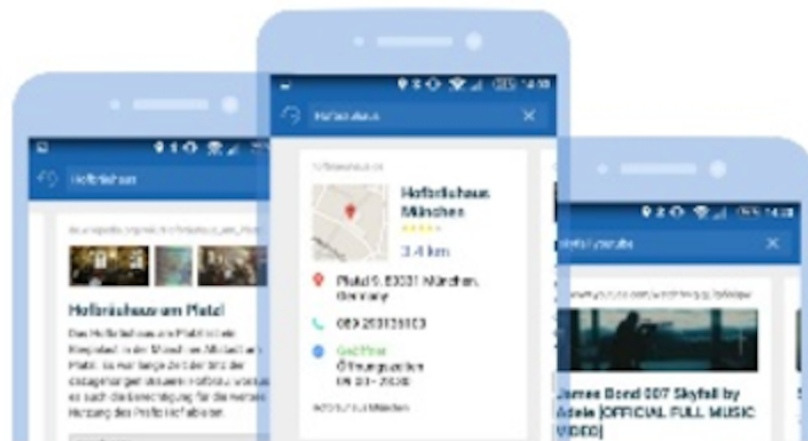
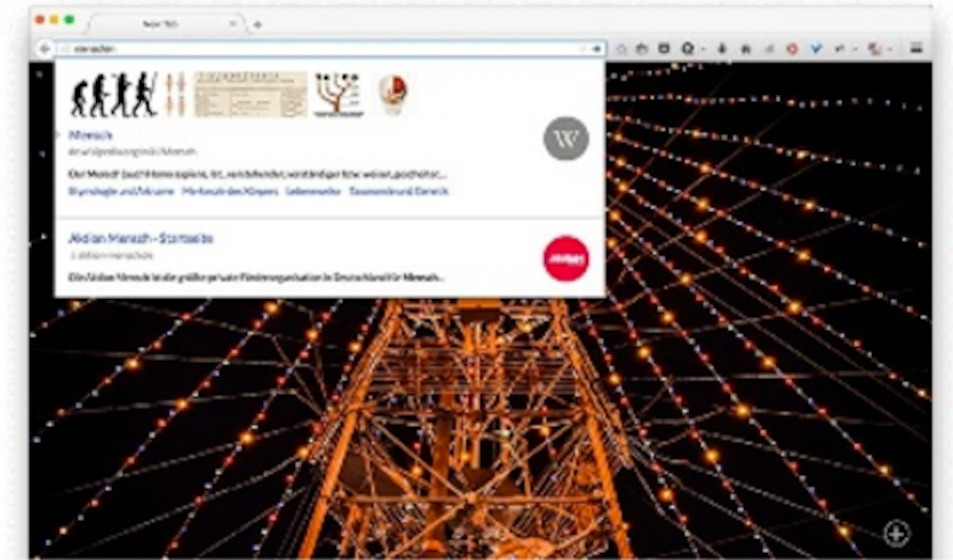


WE REDESIGN THE INTERNET

BASED IN MUNICH

MAJORITY-OWNED  
BY HUBERT BURDA MEDIA

INTERNATIONAL TEAM OF 90  
EXPERTS



WE COMBINE THE POWER  
OF DATA, SEARCH, AND  
BROWSERS  
TO REDESIGN THE INTERNET  
FOR THE USER

**WE'RE  
HIRING!**


<http://cliqz.com/>

# SEARCH@CLIQZ: IN-BROWSER SEARCH


CLIQZFileEditViewHistoryBookmarksToolsWindowHelp

New Tab

python programming wiki



python™



Python (programming language)

en.wikipedia.org/wiki/Python\_(programming\_language)

Python is a widely used high-level, general-purpose, interpreted, dynamic programming ...

HistoryFeatures and philosophySyntax and semanticsLibrariesDevelopment environments

BeginnersGuide - Python Wiki

wiki.python.org/moin/BeginnersGuide

New to Python?

Es

spncricinfo.com

W

weather in bilbao

Bilbao, Spanien


Heute


Montag


Dienstag


Mittwoch


Donnerstag











34° 19°

41° 26°

38° 19°

27° 18°

25° 17°

Extended Forecast

Wetter Bilbao

wetter.com/spanien/bilbao/ES0VZ0028.html


Wie wird das Wetter heute in Bilbao? Temperatur-, Wind- und Regenvorhersage, sowie aktuelle Wetterwarn...

wu


focus

FOCUS Online - Nachrichten


focus.de



Wie viel Taschengeld braucht mein Kind?  
vor 6 Stunden



Panama Papers: Panama will Steuerinformationen mit anderen L...  
vor 9 Stunden



Ihr habt einen Türkei-Urlaub geplant? Das müsst ihr jetzt unbedin...  
vor 10 Stunden

Brutto-Netto-Rec...

Handy-Vergleich

Gehaltsrechner

Fahrtkostenrechn...



## TRADITIONAL SEARCH

- ▶ Traditional Search is based on creating a vector model of the document [TF-IDF etc.] and searching for relevant terms of the query within the same.
- ▶ **Aim:** To give the most accurate document ranked in an order based on several parameters.

## OUR SEARCH STORY

- ▶ Search @ Cliqz based on matching a user query to a query in our index.
- ▶ Construct alternate queries and search them simultaneously.  
Query Similarity based on the words matched and ratio of match.
- ▶ Broadly, our Index:
  - ▶ query: [<url\_id1>, <url\_id2>, <url\_id3>, <url\_id4>]
  - ▶ url\_id1 = "+0LhKNS4LViH\WxbXOTdOQ=="  
{ "url": "http://www.uefa.com/trainingground/skills/video/videoid=871801.html" }

## SEARCH PROBLEM – OVERVIEW

- ▶ Once a user queries search system, two steps happen for an effective search result:
  - ▶ **RECALL:** Get best candidate pages from index which closely represents query.
    - ▶ @Cliqz: Come up with (~10k+) pages using all techniques from index (1.8+ B pages) that are most appropriate pages w.r.t query.
  - ▶ **RANKING:** Rank the candidate pages based on different ranking signals.
    - ▶ @Cliqz: Several steps. After first recall of ~10,000 pages, **pre\_rank** prunes this list down to 100 good candidate pages.
    - ▶ Final Ranking prunes this list of 100 to Top 3 Results.
- ▶ **Given a user Query, find 3 good pages out of ~2 Billion Pages in Index!**

## ENTERS DEEP LEARNING

- ▶ Queries defined as a fixed dimensional vector of floating point values. Ex. 100 dimensions
- ▶ **Distributed Representation:** Words that appear in the same contexts share semantic meaning. The meaning of the Query is defined by the floating point numbers distributed in the vector.
- ▶ Query Vectors are **learned** in an unsupervised manner. Where we focus on the context of words in sentences or queries and learn the same. For learning word representations, we employ a **Neural Probabilistic Language Model (NP-LM)**.
- ▶ Similarity between queries are measured as cosine or vector distance between pair of query vectors We then get “closest queries” to a user query and fetch pages (Recall).



## EXAMPLE QUERY: “SIMS GAME PC DOWNLOAD”

- ▶ "closest\_queries": [
  - ▶ [ "2 download game pc sims", 0.10792562365531921],
  - ▶ [ "download full game pc sims", 0.16451804339885712],
  - ▶ [ "download free game pc sims", 0.1690218299627304],
  - ▶ [ "game pc sims the", 0.17319737374782562],
  - ▶ [ "2 game pc sims", 0.17632317543029785],
  - ▶ ["3 all download game on pc sims", 0.19127938151359558]
  - ▶ ["download pc sims the", 0.19307053089141846],
  - ▶ ["3 download free game pc sims", 0.19705575704574585],
  - ▶ ["2 download free game pc sims", 0.19757266342639923],
  - ▶ ["game original pc sims", 0.1987953931093216],
  - ▶ ["download for free game pc sims", 0.20123696327209473]
- ▶ .....]

## LEARNING DISTRIBUTED REPRESENTATION OF WORDS

- ▶ We use un-supervised deep learning techniques, to learn a **word representation  $C(w)$**  which is a **continuous vector** and is both syntactically and semantically similar.
- ▶ More precisely, we learn a continuous representation of words and would like the distance  $\| C(w) - C(w') \|$  to reflect meaningful similarity between words  $w$  and  $w'$ .
- ▶  *$\text{vector}('king') - \text{vector}('man') + \text{vector}('woman')$  is close to  $\text{vector}('queen')$*
- ▶ We use Word2Vec to learn word and their corresponding vectors.

## WORD2VEC DEMYSTIFIED

- ▶ Mikolov T. et al. 2013, proposes two novel model architectures for computing continuous vector representations of words from very large datasets. They are:
  - ▶ Continuous Bag of Words (cbow)
  - ▶ Continuous Skip Gram (skip)
- ▶ Word2Vec focuses on distributed representations learned by neural networks. Both models are trained using stochastic gradient descent and back propagation.

# WORD2VEC DEMYSTIFIED

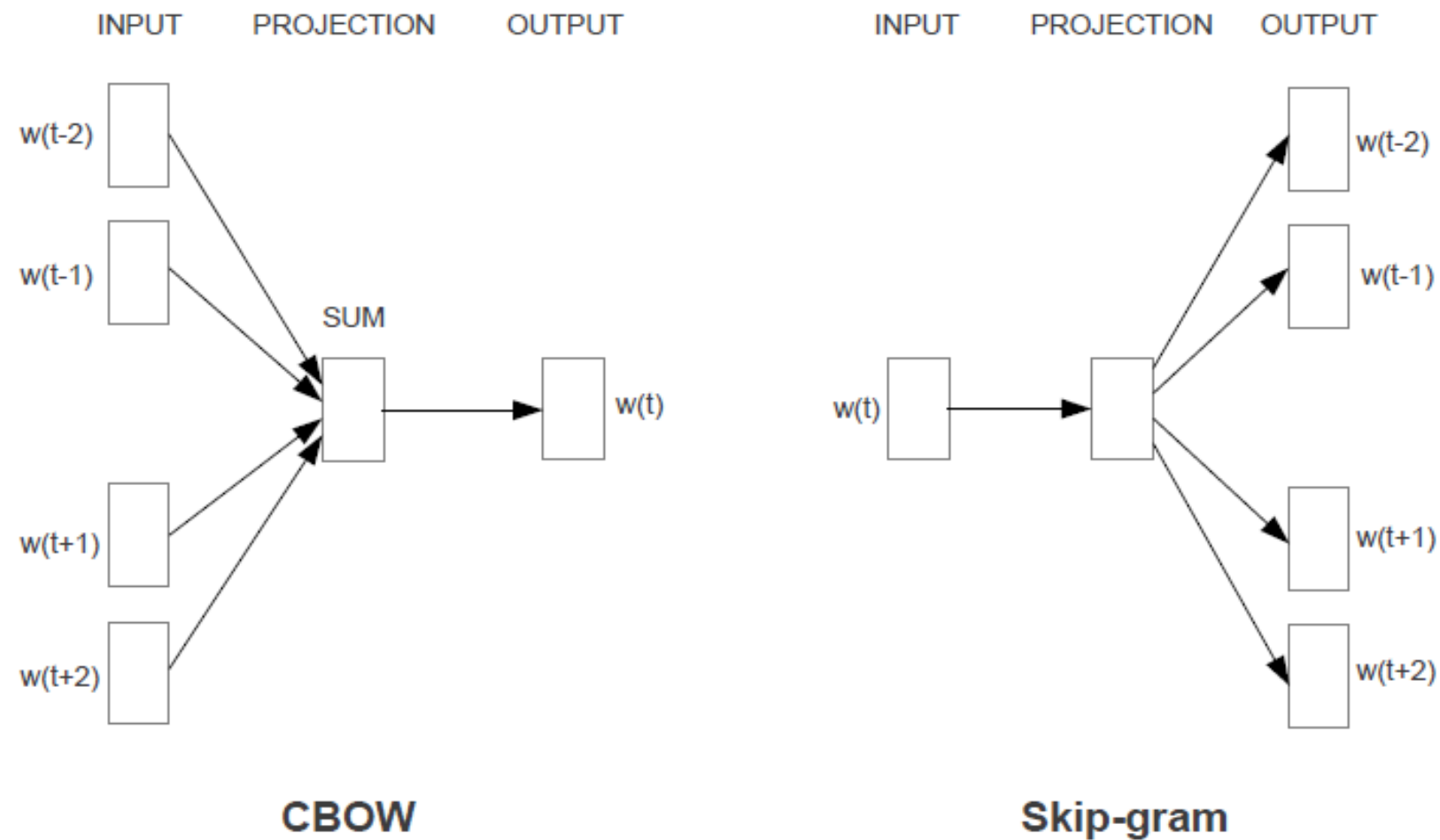


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.



# NEURAL PROBABILISTIC LANGUAGE MODELS

- ▶ NP-LM use Maximum Likelihood principle to maximize the probability of the next word  $w_t$  (for "target") given the previous words  $h$  (for "history") in terms of a soft-max function:

$$\begin{aligned} P(w_t|h) &= \text{softmax}(\text{score}(w_t, h)) \\ &= \frac{\exp\{\text{score}(w_t, h)\}}{\sum_{\text{Word } w' \text{ in Vocab}} \exp\{\text{score}(w', h)\}}. \end{aligned}$$

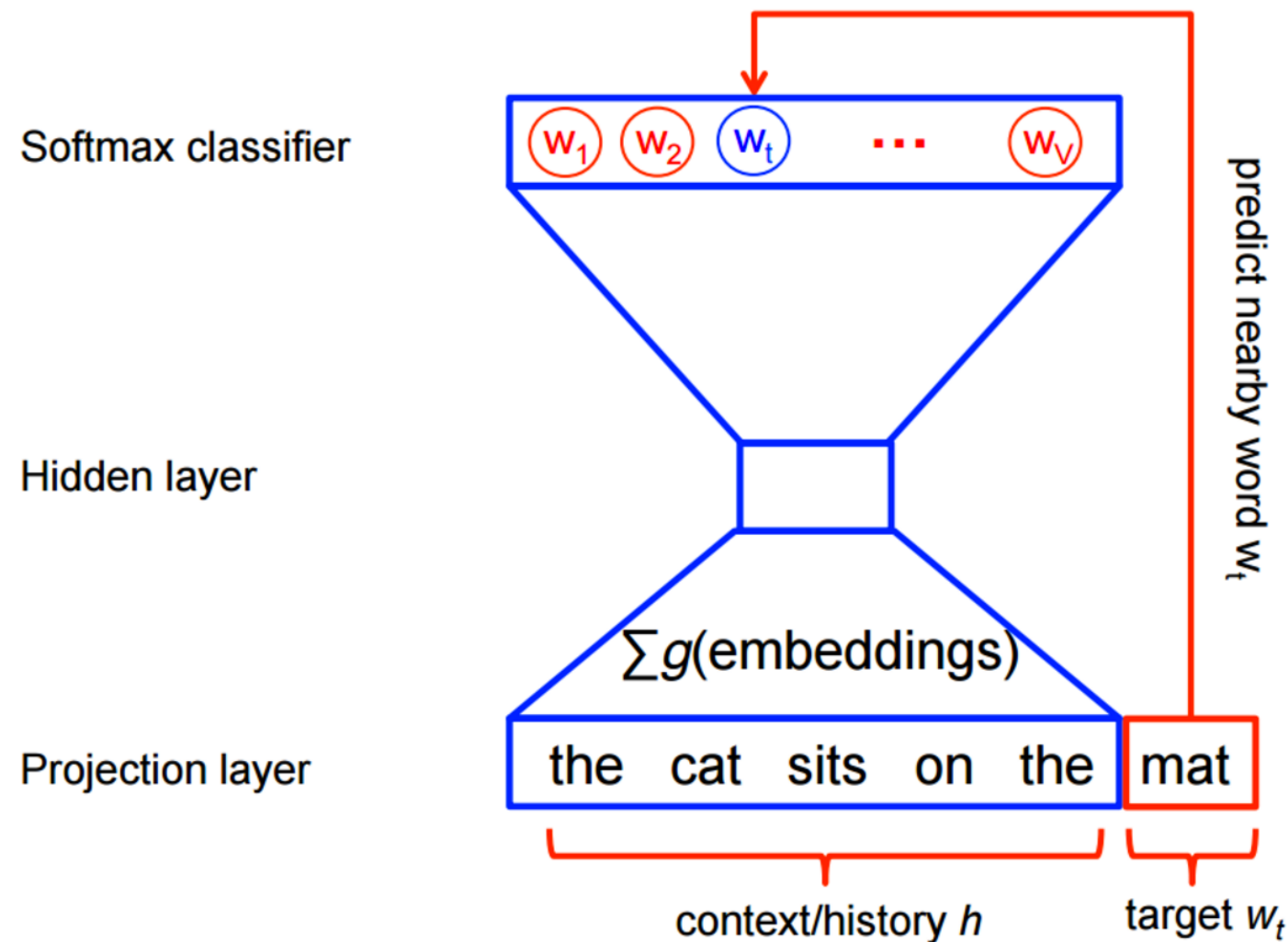
$\text{score}(w_t, h)$  computes the compatibility of word  $w_t$  with the context  $h$  (a dot product). We train this model by maximizing its log-likelihood on the training set, i.e. by maximizing:

$$\begin{aligned} J_{\text{ML}} &= \log P(w_t|h) \\ &= \text{score}(w_t, h) - \log \left( \sum_{\text{Word } w' \text{ in Vocab}} \exp\{\text{score}(w', h)\} \right) \end{aligned}$$

- ▶ **Pros:** Yields a properly normalized probabilistic model for language modeling.
- ▶ **Cons:** Very expensive, because we need to compute and normalize each probability using the score for all other  $V$  words  $w'$  in the current context  $h$ , at every training step.

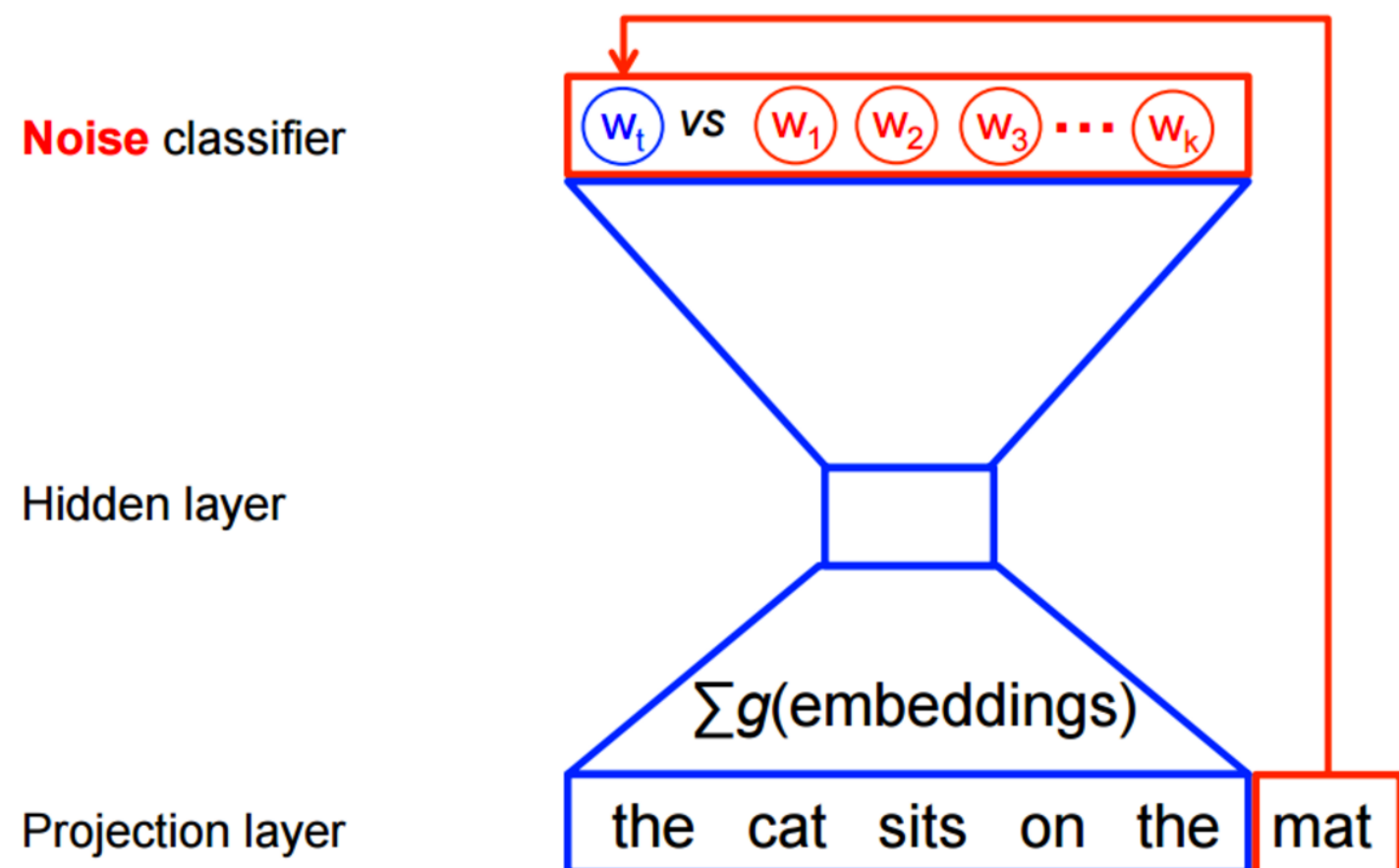
# NEURAL PROBABILISTIC LANGUAGE MODELS

- ▶ A properly normalized probabilistic model for language modeling.



## WORD2VEC DEMYSTIFIED

- ▶ Word2Vec models are trained using binary classification objective (logistic regression) to discriminate the real target words  $w_t$  from  $k$  imaginary (noise) words  $w^{\sim}$ , in the same context.
- ▶ For CBOW:



## WORD2VEC DEMYSTIFIED

- ▶ The objective for each example is to maximize:

$$J_{\text{NEG}} = \log Q_{\theta}(D = 1|w_t, h) + k \mathbb{E}_{\tilde{w} \sim P_{\text{noise}}} [\log Q_{\theta}(D = 0|\tilde{w}, h)]$$

- ▶ Where  $Q_{\theta}(D=1|w, h)$  is the binary logistic regression probability under the model of seeing the word  $w$  in the context  $h$  in the dataset  $\mathbf{D}$ , calculated in terms of the learned embedding vectors  $\theta$ .
- ▶ In practice, we approximate the expectation by drawing  $k$  contrastive words from the noise distribution.
- ▶ This objective is maximized when the model assigns high probabilities to the real words, and low probabilities to noise words (Negative Sampling).
- ▶ **Performance: Way more faster.** Computing loss function scales to only the number of noise words that we select " $k$ " and not to entire Vocabulary " $V$ ".



## EXAMPLE: SKIP-GRAM MODEL

- ▶ d: "the quick brown fox jumped over the lazy dog"
- ▶ Define context window size: 1. Dataset of (context, target):
  - ▶ ([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox), ...
- ▶ Recall, skip-gram inverts contexts and targets, and tries to predict each context word from its target word. So, task becomes to predict 'the' and 'brown' from 'quick', 'quick' and 'fox' from 'brown', etc. Dataset of (input, output) pairs becomes:
  - ▶ (quick, the), (quick, brown), (brown, quick), (brown, fox), ...
- ▶ **Objective function** defined over entire dataset. We optimize this with SGD using one example at a time. (or, using a mini-batch ( $16 \leq \text{batch\_size} \leq 512$ ))

## EXAMPLE: SKIP-GRAM MODEL

- ▶ Say, at training time  $t$ , we see training case: (quick, the)
- ▶ Goal: Predict "the" from "quick"
- ▶ Next, we select "num\_noise" number of noisy (contrastive) examples by drawing from some noise distribution, typically the unigram distribution,  $P(w)$ . For simplicity let's say num\_noise=1 and we select "sheep" as a noisy example.
- ▶ Next, we compute "loss" for this pair of observers and noisy examples. i.e. **Objective** at time step "t" becomes:

$$J_{\text{NEG}}^{(t)} = \log Q_{\theta}(D = 1 | \text{the, quick}) + \log(Q_{\theta}(D = 0 | \text{sheep, quick})).$$

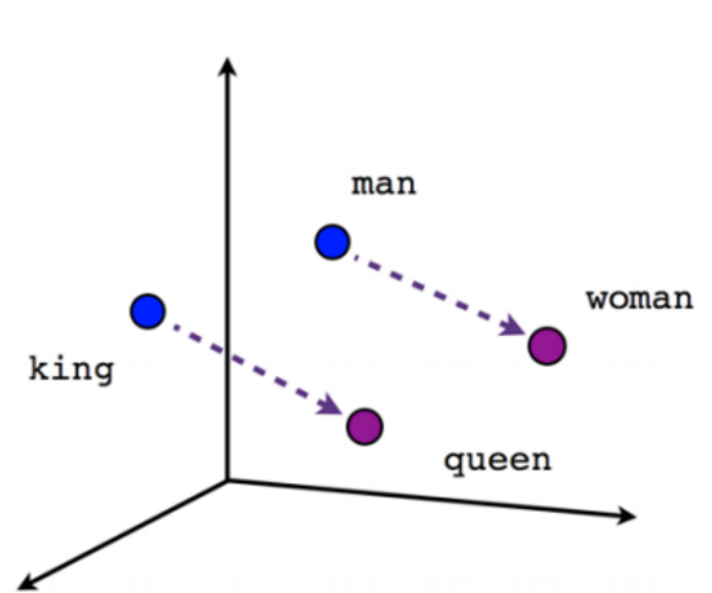
- ▶ Goal: Update  $\theta$  (embedding parameters), to maximize this objective function.

## EXAMPLE: SKIP-GRAM MODEL

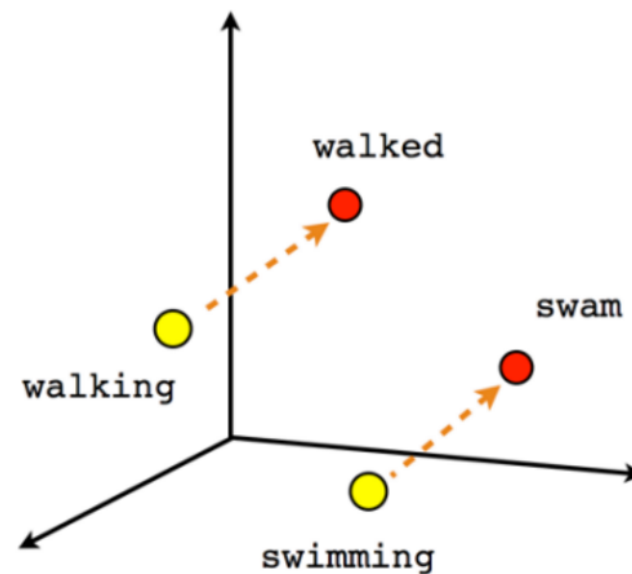
- ▶ For maximizing this loss function we obtain a **gradient** or **derivative** w.r.t embedding parameter  $\theta$ . i.e.  $\frac{\partial}{\partial \theta} J_{\text{NEG}}$
- ▶ We then **perform an update to the embeddings** by taking a small step in the direction of the gradient.
- ▶ We **repeat this process** over the entire training set, this has the effect of 'moving' the **embedding vectors** around for each word until the model is successful at discriminating real words from noise words.

# VISUALIZING WORD EMBEDDINGS

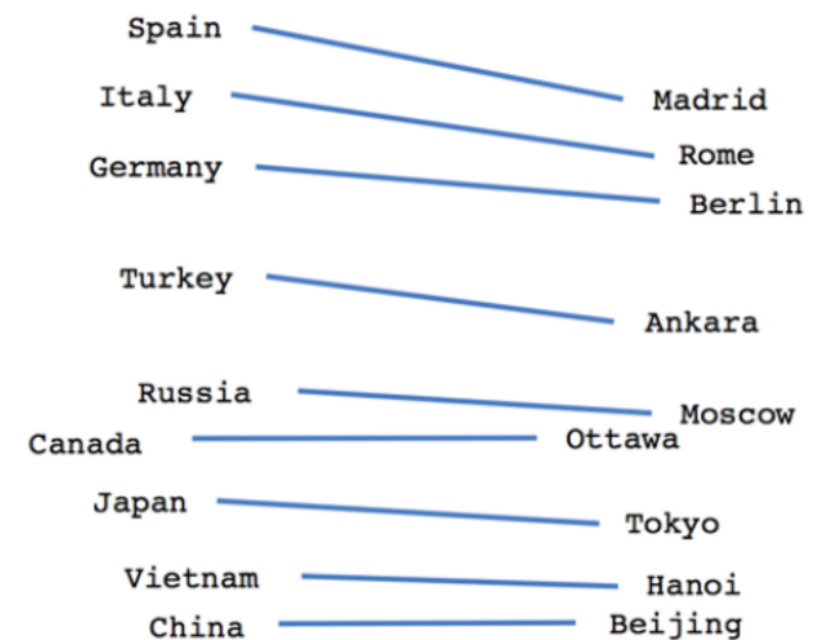
# WORD VECTORS CAPTURING SEMANTIC INFORMATION



Male-Female

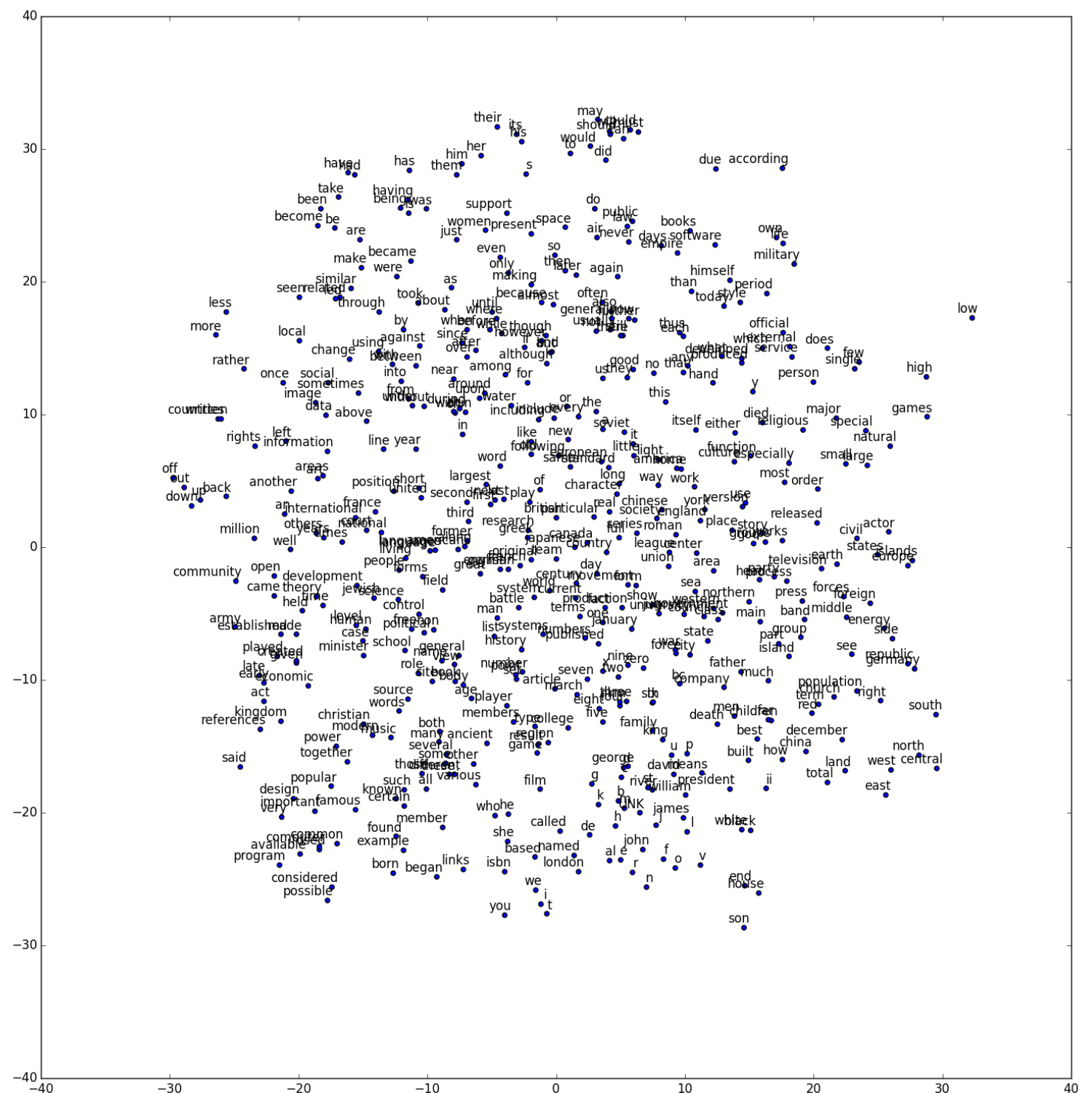


Verb tense



Country-Capital

## WORD VECTORS IN 2D



## QUERY VECTOR FORMATION – “SIMS GAME PC DOWNLOAD”

- ▶ **STEP 1: Word2Vec** training gives unique individual vectors for each word. [dimensionality = 100]
  - ▶ sims: [0.01 ,0.2, ....., 0.23]
  - ▶ game : [0.21 ,0.12, ....., 0.123]
  - ▶ pc: [ -0.71 ,0.52, ....., -0.253]
  - ▶ download: [0.31 ,-0.62, ....., 0.923]
- ▶ **STEP 2: Get the term relevance** for each word in the query.
  - ▶ 'terms\_relevance': {'sims': 0.9015615463502331, 'pc': 0.4762325748412917, 'game': 0.6077838963329699, 'download': 0.5236977938865315}

## QUERY VECTOR FORMATION – “SIMS GAME PC DOWNLOAD”

- ▶ **STEP 3:** Next, we calculate a centroid (or Average) of the vectors (relevance-based) for each of the words in query. This resulting vector represents our Query. Simple, Weighted Average Example:
  - ▶ In [5]: `w_vectors = [[1,1,1],[2,2,2]]`
  - ▶ In [6]: `weights= [1, 0.5]`
  - ▶ In [7]: `numpy.average(w_vectors, axis=0, weights=weights)`  
`array([ 1.33333333, 1.33333333, 1.33333333])`
- ▶ In the end,
  - ▶ **sims game pc download:** `[-0.171 ,0.252, ..... , -0.653]`  
{dimensionality remains 100}



# TERMS RELEVANCE

- ▶ Two modes to compute Term Relevance:
  - ▶ **Absolute:**  $\text{tr\_abs}(\text{word}) = \text{word\_stats}(\text{'tf5df'}) / \text{word\_stats}(\text{'df'})$
  - ▶ **Relative:**  $\text{tr\_rel}(\text{word}) = \log(N/n) * \text{absolute}$ ,
    - ▶ where,  $N$  is the number of page models in the index and  $n = \text{df}$
- ▶  $\text{tf5df}$ ,  $\text{df}$ ,  $N$  are all data dependent, which we compute for each data refresh.
- ▶ For our example, *word\_stats* look like this:
  - ▶ 

```
({'sims': {'f': 3734417, 'df': 481702, 'uqf': 1921554, 'tf1df': 288718, 'tf2df': 369960, 'tf3df': 403840, 'tf5df': 434284}, 'pc': {'f': 20885669, 'df': 3297244, 'uqf': 11216714, 'tf1df': 288899, 'tf2df': 604095, 'tf3df': 967704, 'tf5df': 1570255}, 'game': {'f': 11431488, 'df': 2412879, 'uqf': 5354115, 'tf1df': 253090, 'tf2df': 597603, 'tf3df': 979049, 'tf5df': 1466509}, 'download': {'f': 50131109, 'df': 11402496, 'uqf': 26644950, 'tf1df': 430566, 'tf2df': 1147760, 'tf3df': 2584554, 'tf5df': 5971462}})
```

## QUERY VECTOR INDEX

- ▶ We perform this vector generation for top five queries leading to all pages in our data.
- ▶ We collect, **Top Queries** for each page from PageModels
- ▶ ~465 Million+ Queries representing all pages in our index
- ▶ Learn Query Vectors for them. Size: ~700 GB on disk.
- ▶ **How do we get similar queries: User query vs 465 Million Queries?**

## FINDING CLOSEST QUERIES

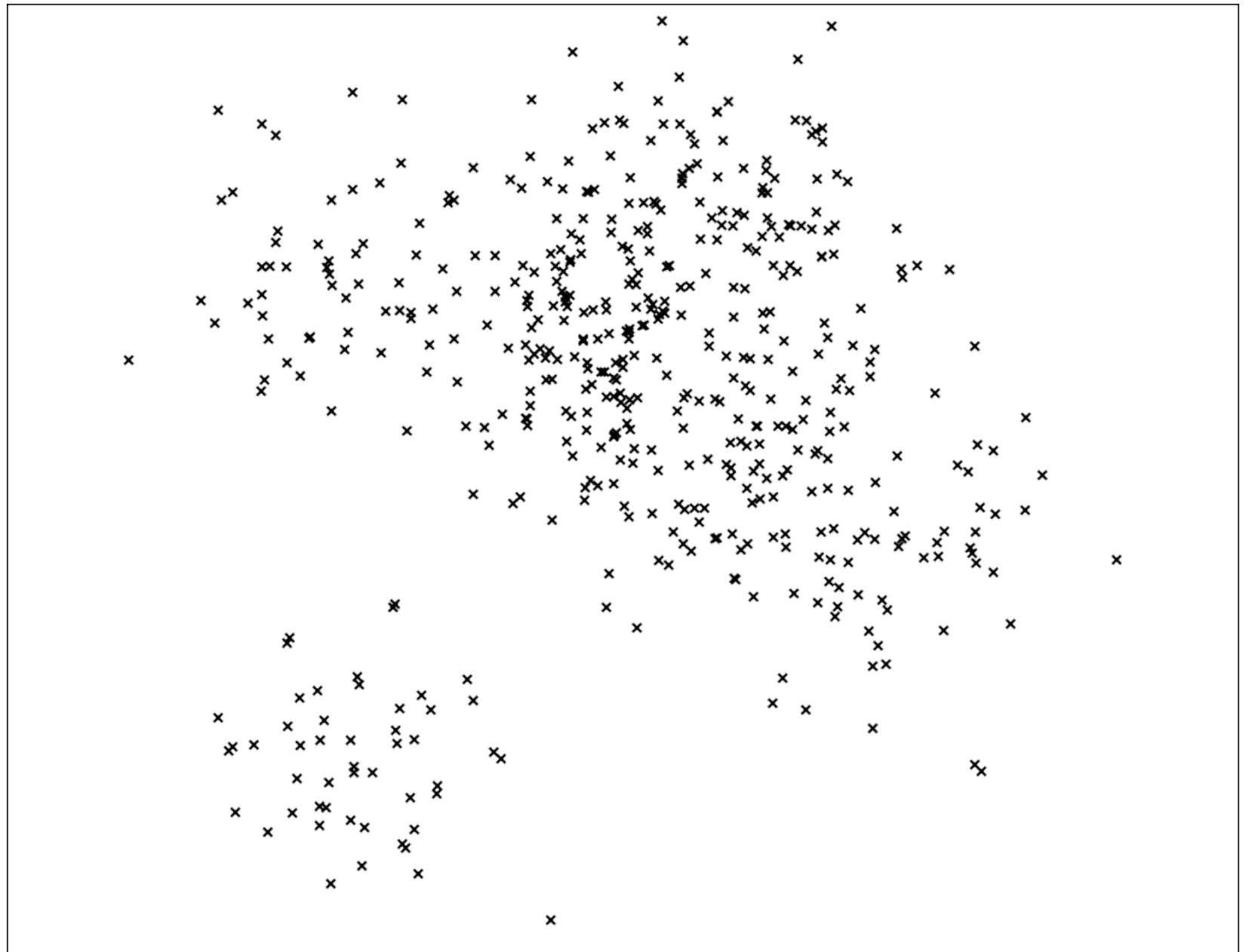
- ▶ Brute Force: User Query vs 465M Queries – Too Too Slow!
- ▶ Hashing Techniques - Not very accurate for vectors. – Vectors are semantic!
- ▶ The solution required:
  - ▶ Application of cosine similarity metric.
  - ▶ Scale to 465 million Query Vectors.
  - ▶ Takes ~10 milli-seconds or less!
- ▶ **Approximate Nearest Neighbor Vector Model to the rescue!**

## ANNOY (APPROXIMATE NEAREST NEIGHBOR MODEL)

- ▶ We use "Annoy" library (C++ & python wrapper) to build the Approximate nearest neighbor models. Annoy is used in production at Spotify.
- ▶ We can't train on all 465M queries at once, too slow.
- ▶ Train: 10 models or 46+ M queries each
- ▶ Number of **Trees**: 10 (explained next)
- ▶ Size of Models: 27 GB per shard [10 models - 270 GB+] [stored in RAM]
- ▶ Query all 10 shards of the cluster at runtime. Sort them based on cos. similarity.
- ▶ Get top 55 nearest queries to user query and fetch pages related to nearest queries.

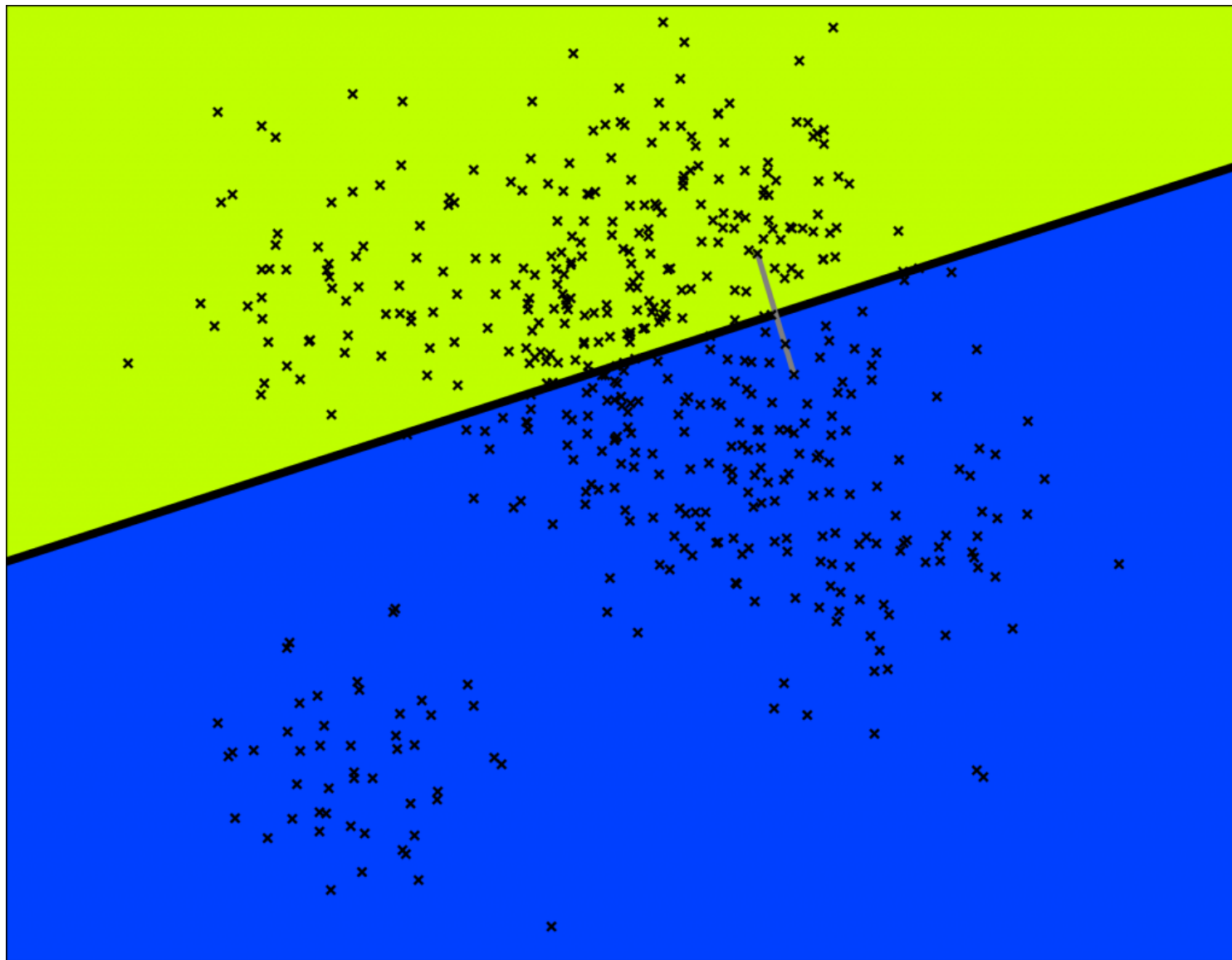
## ANATOMY OF ANNOY

- ▶ **Goal:** Find the nearest points to any query point in sub-linear time.
- ▶ **Build a Tree,**
- ▶ **queries in  $O(\log n)$**



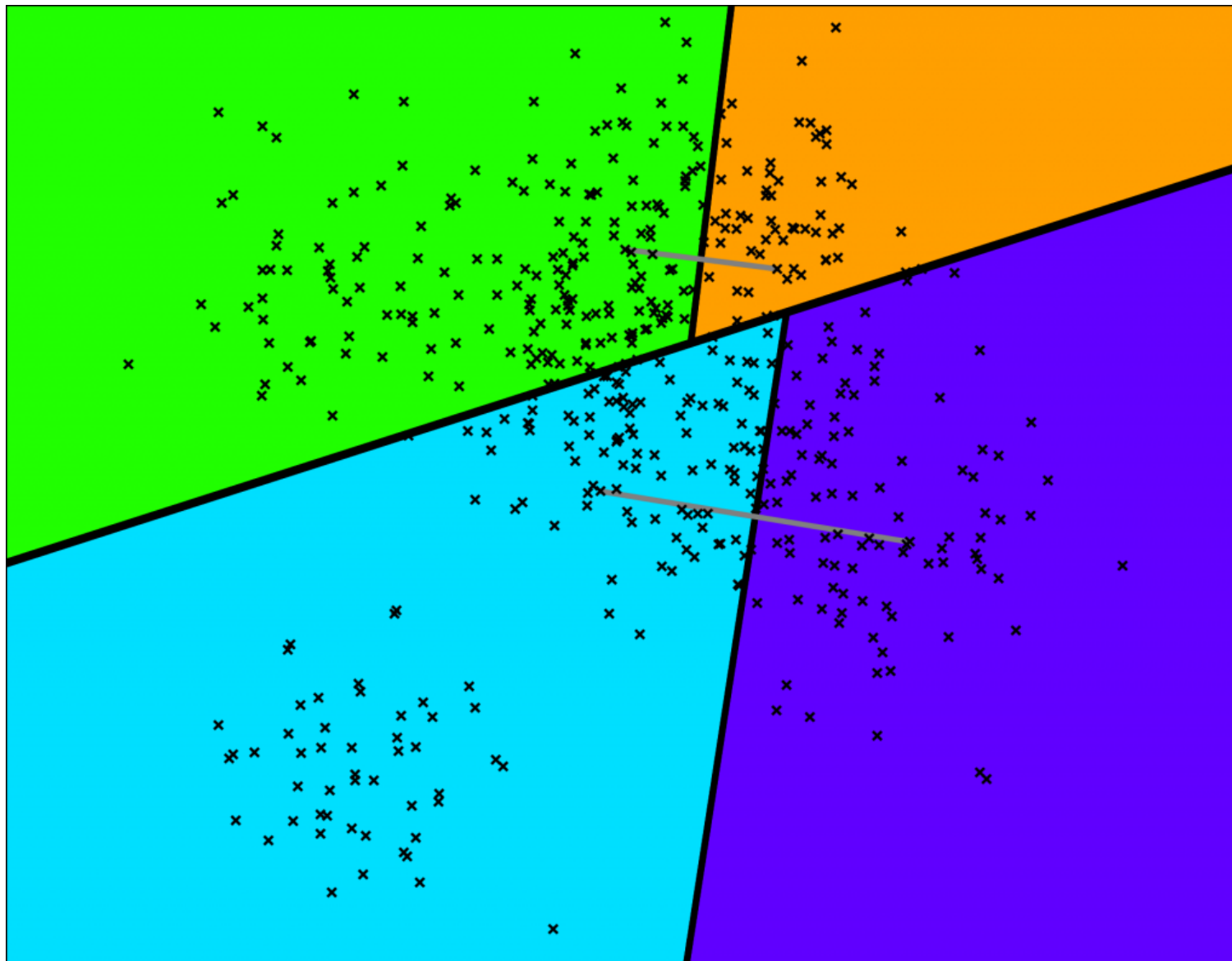
## ANATOMY OF ANNOY

- ▶ Pick two points randomly, split the hyper-space.



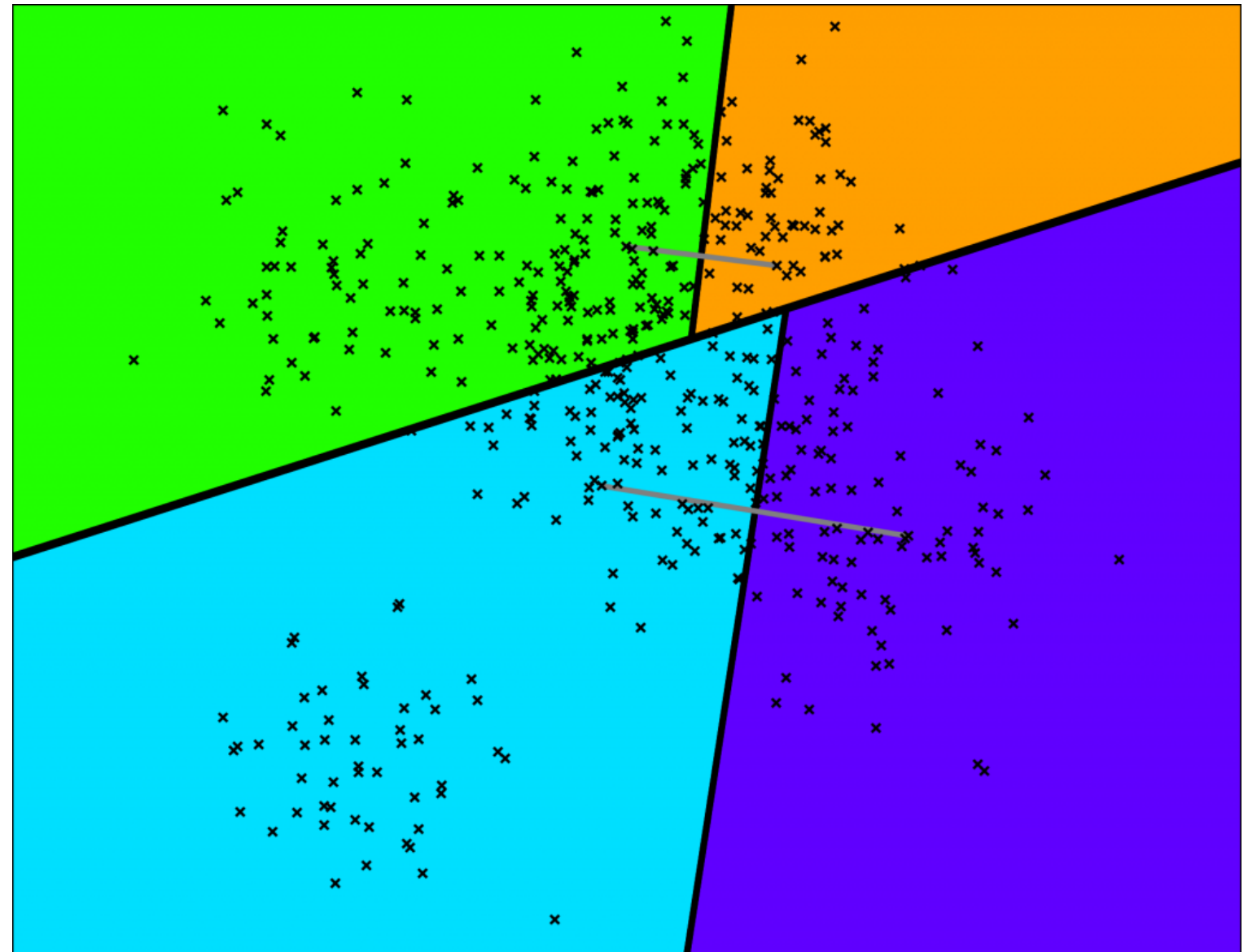
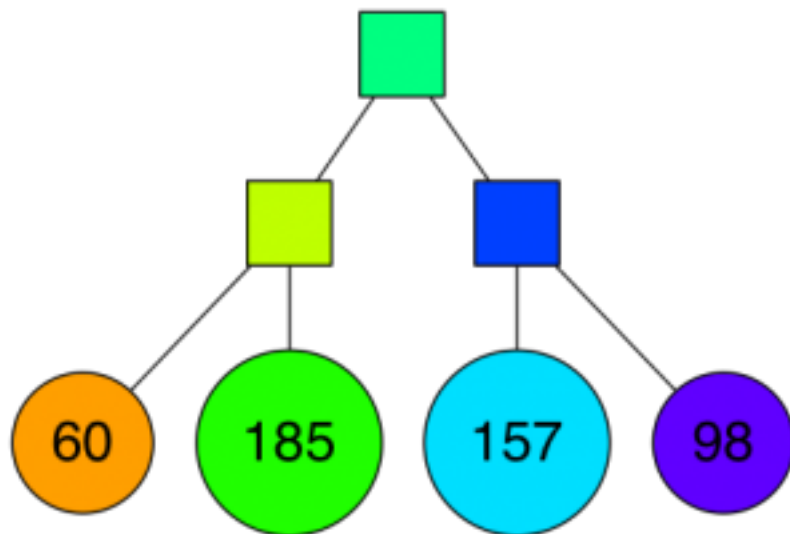
# ANATOMY OF ANNOY

## ► Split Recursively



# ANATOMY OF ANNOY

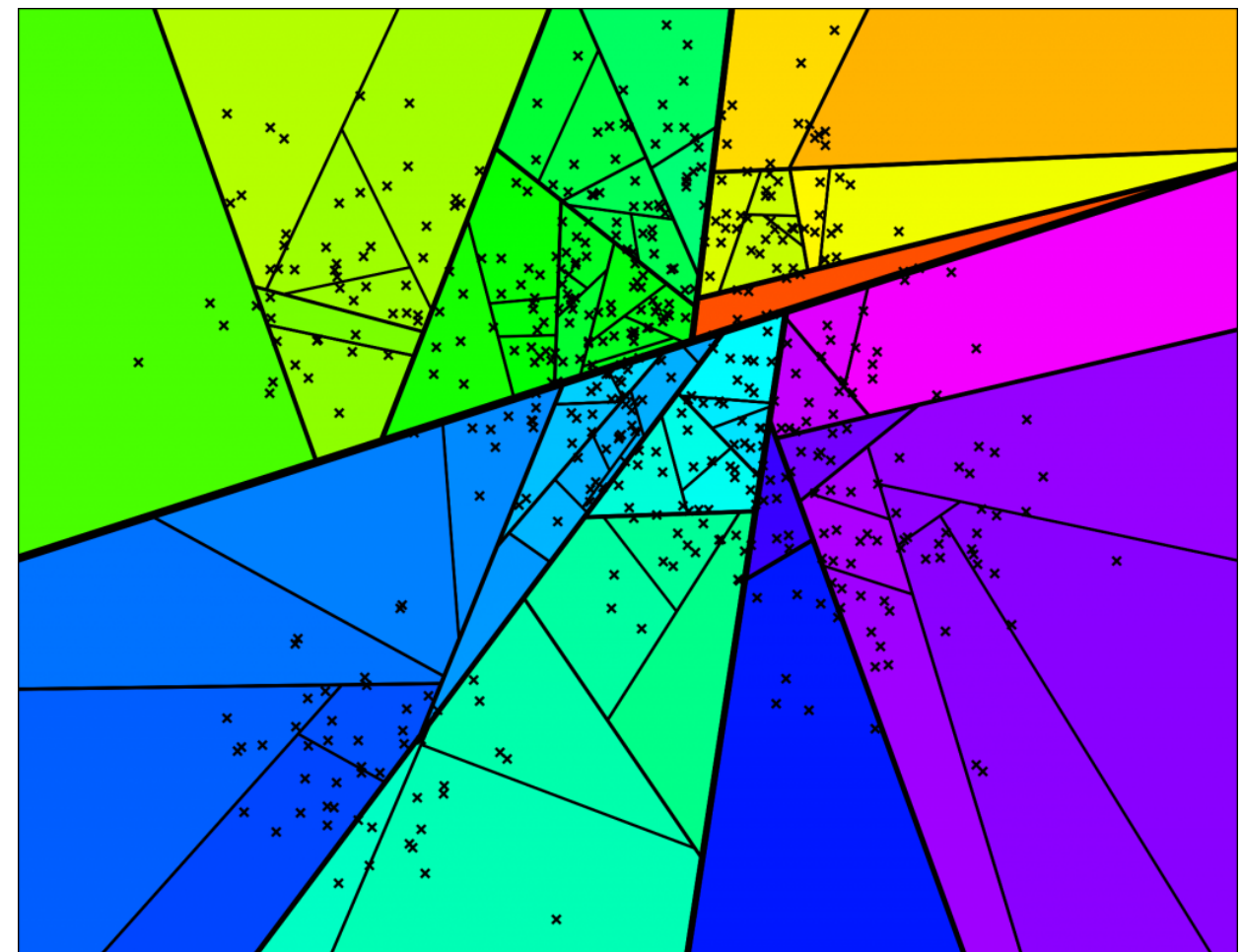
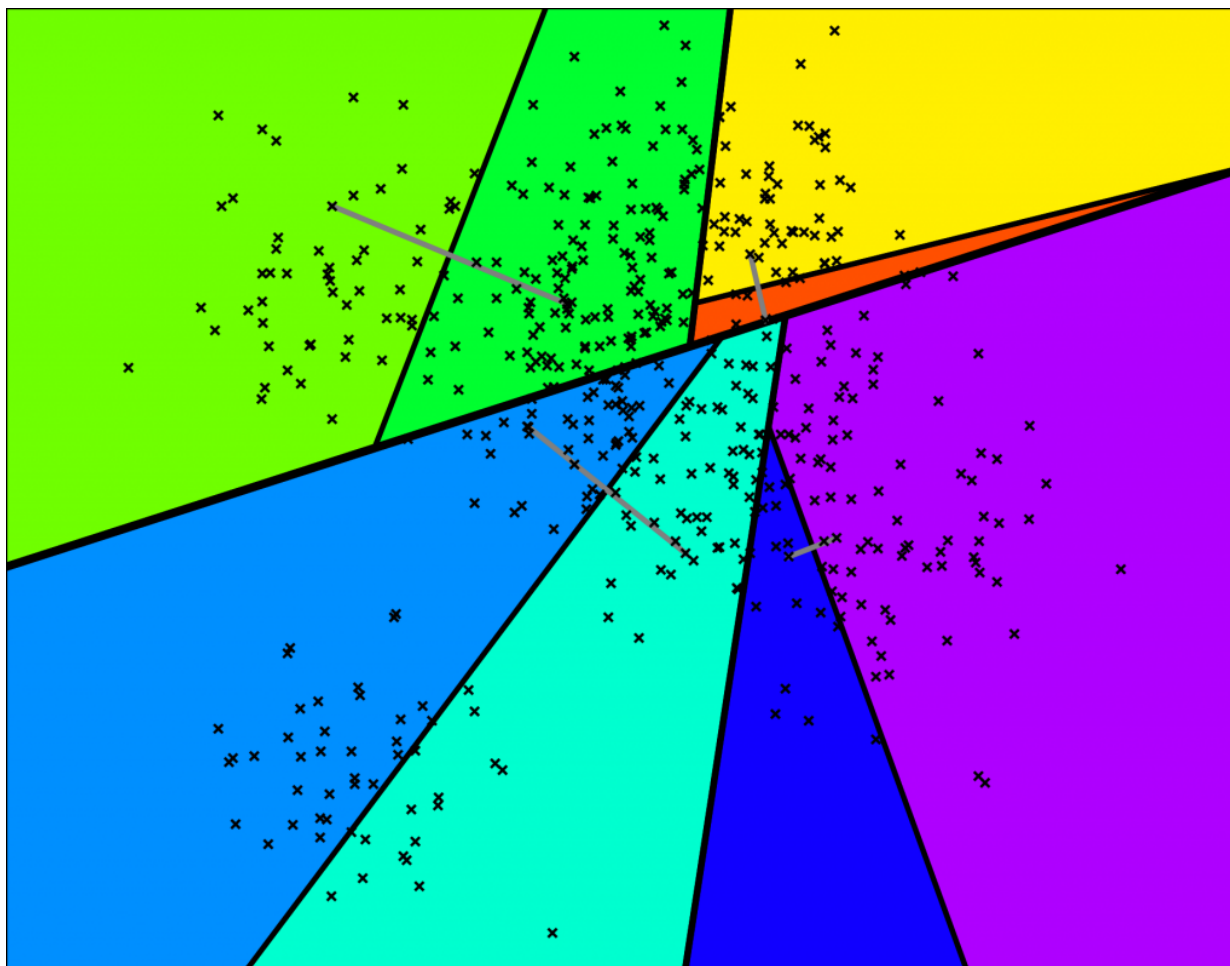
- ▶ Split Recursively
- ▶ Tiny Binary Tree appears.





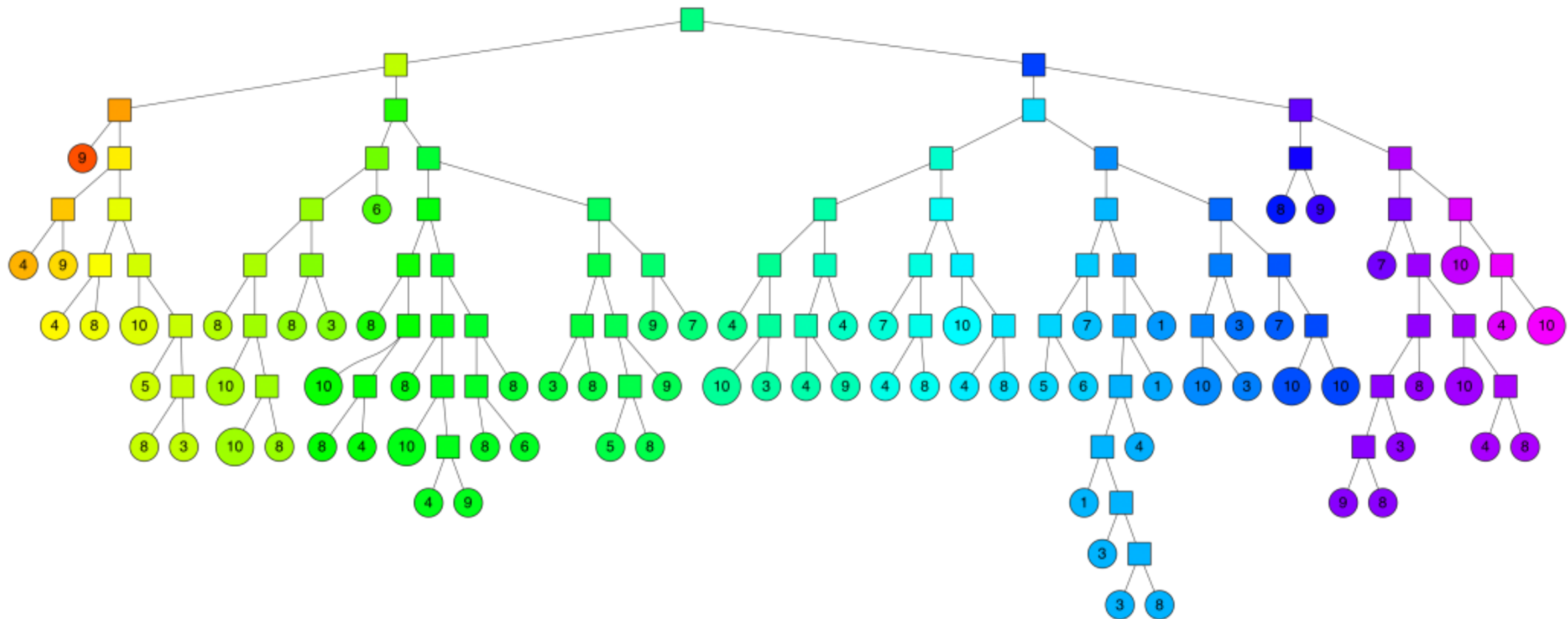
# ANATOMY OF ANNOY

## ► Keep Splitting



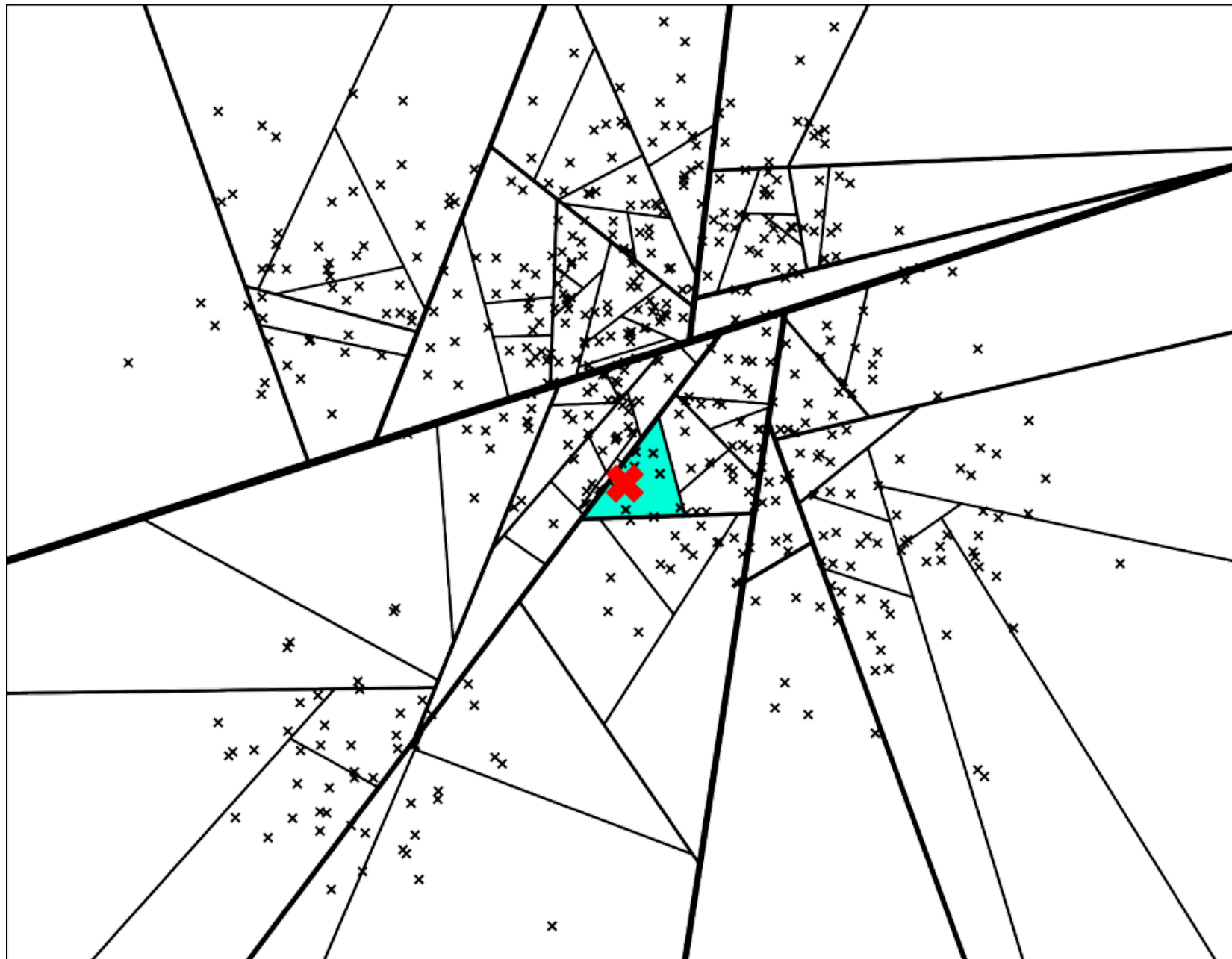
## ANATOMY OF ANNOY

- ▶ End up with Binary Tree Partitioning the Space.
- ▶ Nice thing : Points that are close to each other in the space are more likely to be close to each other in the tree



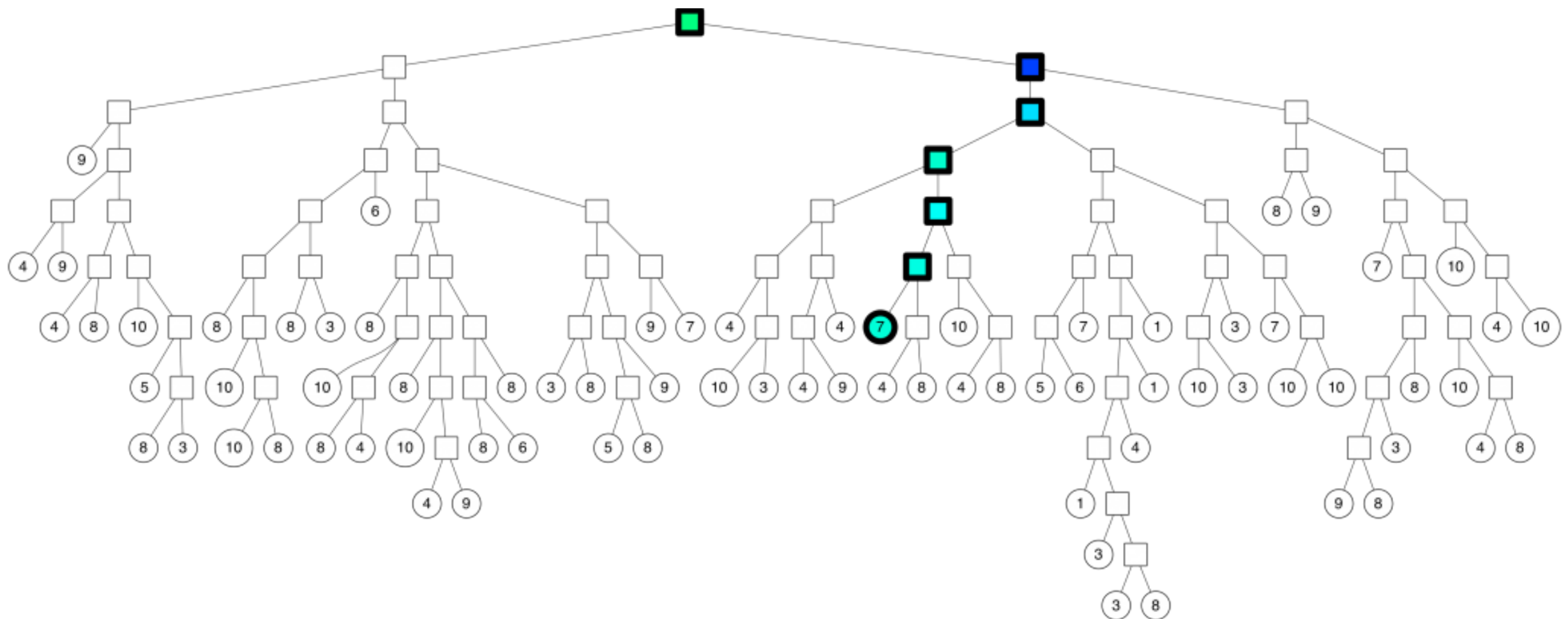
# ANATOMY OF ANNOY

## ► Searching for a point



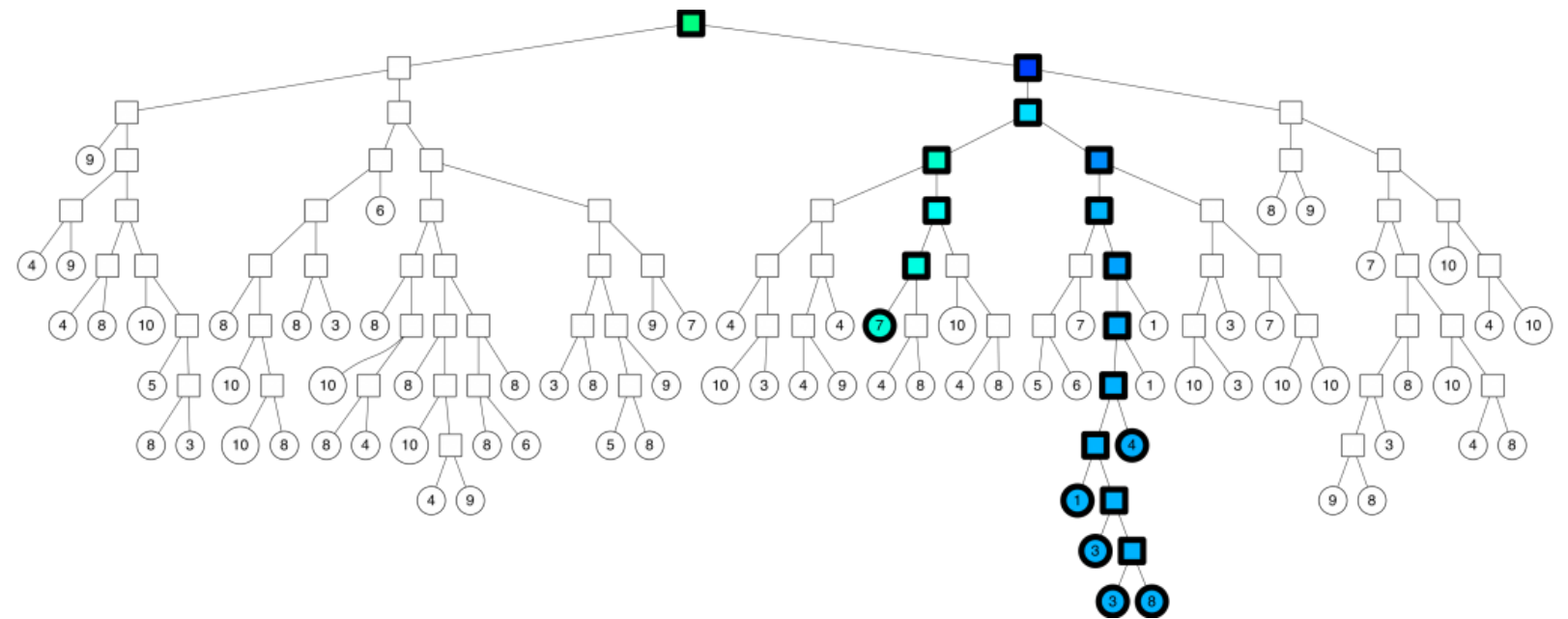
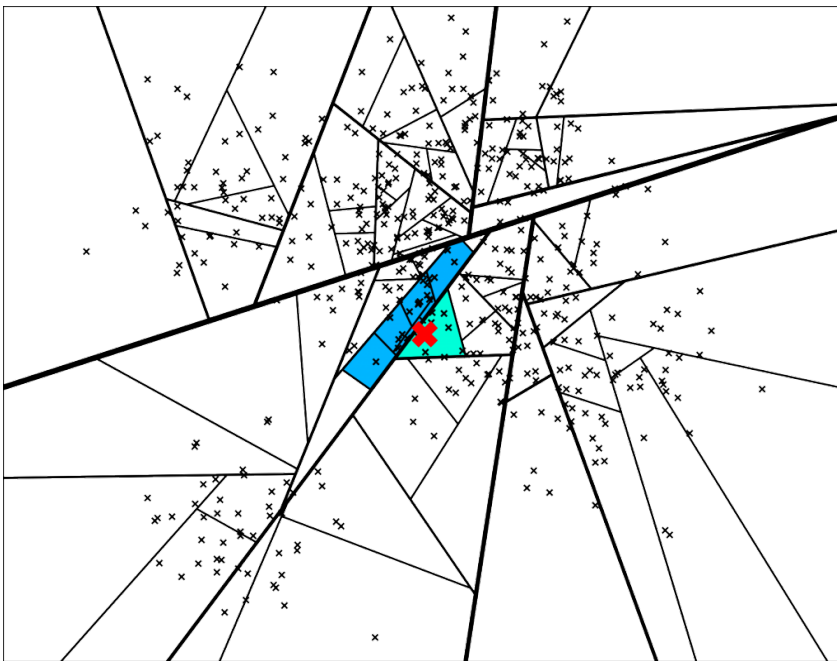
## ANATOMY OF ANNOY

- ▶ Searching for a point: Path down the binary tree.
- ▶ We end up with: 7 neighbors..... Cool!



# ANATOMY OF ANNOY

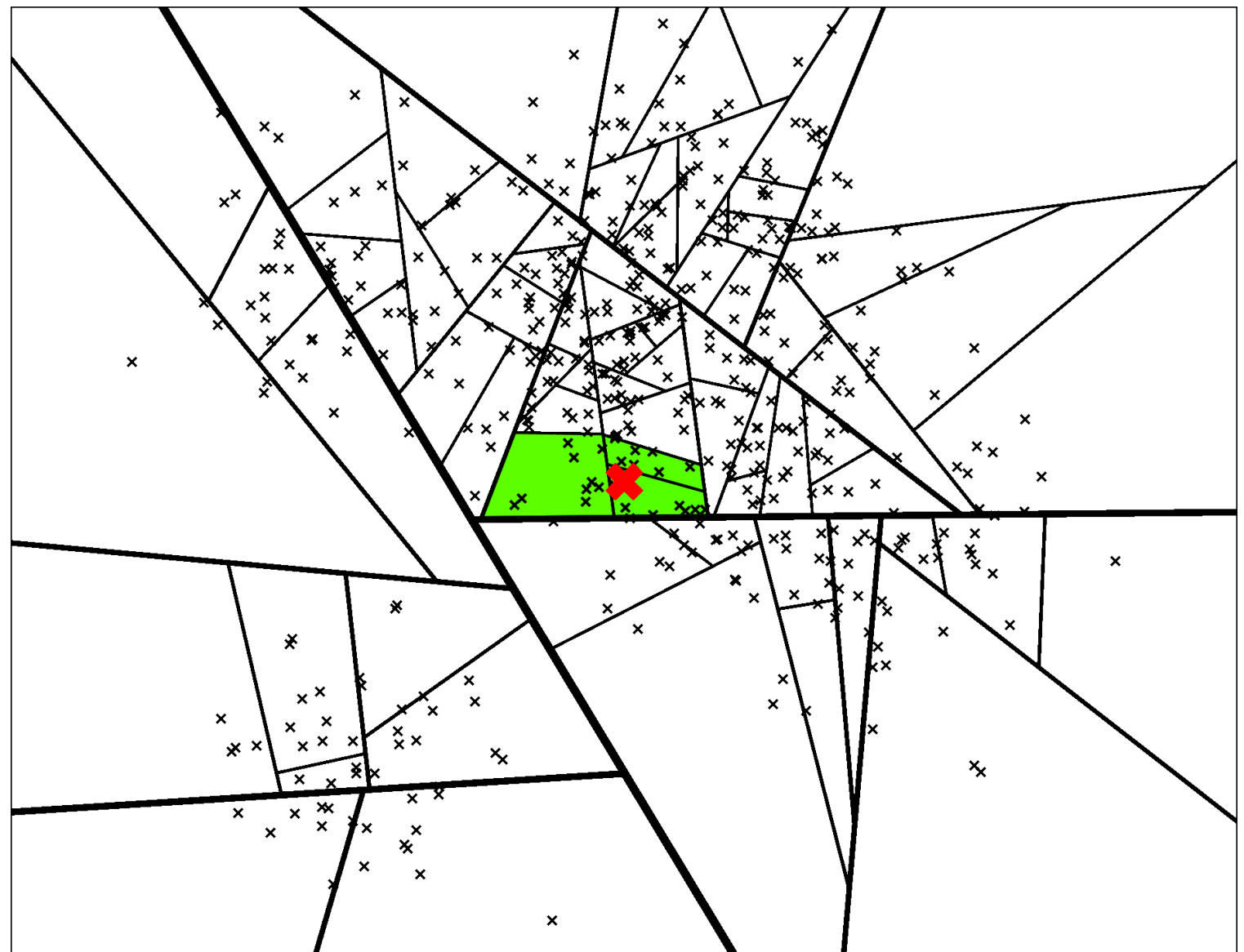
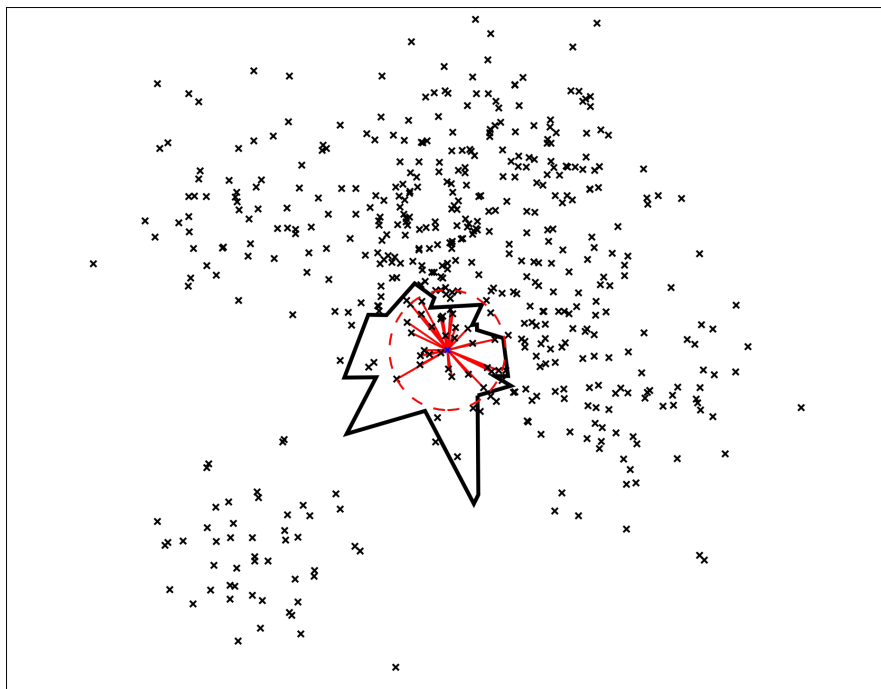
- ▶ What if: We want more than 7 neighbors?
- ▶ Use: Priority Queue [Traverse both sides of split - threshold based]





## ANATOMY OF ANNOY

- ▶ Some of the nearest neighbors are actually outside of this leaf polygon!
- ▶ Use: Forest of Trees



## STORING WORD EMBEDDINGS & QUERY-INTEGER MAPPINGS

- ▶ Word2Vec gives a word - vector pair and Annoy stores query as integer index in its model.
- ▶ These mappings are stored in our key-value index “keyvi”, developed in-house @ CLIQZ, which also takes care of our entire search index.



the key value index  
optimized for size and speed

## RESULTS

- ▶ Much richer set of candidate pages after first fetching step from index, with higher possibility of expected page(s) being among them.
- ▶ The queries are now matched (in real-time) using a cosine vector similarity between query vectors as well as using classical Cliqz - IR techniques.
- ▶ Overall, the recall improvement from previous release is ~ 5% to 7%
- ▶ The translated improvement in precision-value scores is between: ~ 0.5% to 1%



# CONCLUSION

- ▶ Query embeddings is a unique way to improve recall, which is different from conventional web search techniques.
- ▶ Current work:
  - ▶ Ranking changes to include: Query/Page Similarity Metric.
  - ▶ Query to Page Similarity using Document Vectors
  - ▶ Improving search system for pages which are not linked to queries.
  - ▶ And lots more ...

THANK YOU

---

**YOU SHALL KNOW A WORD BY  
THE COMPANY IT KEEPS.**

**John Rupert Firth(1957)**