Dall-E

Similarity, Clustering, and Retrieval

Applied Machine Learning
Derek Hoiem

Last class: How to represent data

- Images, text, categories, numerical → vector of numbers
- Dataset: a collection of data points or samples from some distribution
- We can measure entropy, information gain, and other distributional properties

Today’s lecture

- Similarity
- Retrieval
  - “Brute force”
  - Faiss library
  - Approximate: LSH
- Clustering
  - Kmeans
  - Hierarchical Kmeans
  - Agglomerative Clustering

Key principle of machine learning

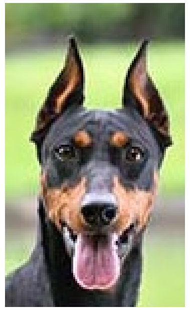
Given feature/target pairs $(X_1, y_1), \dots, (X_n, y_n)$:

if X_i is similar to X_j, then y_i is probably similar to y_j

Fundamentally, learning depends on:

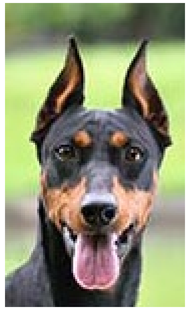
1. Representation of samples
2. Similarity function

Aggressive?



Aggressive

Friendly

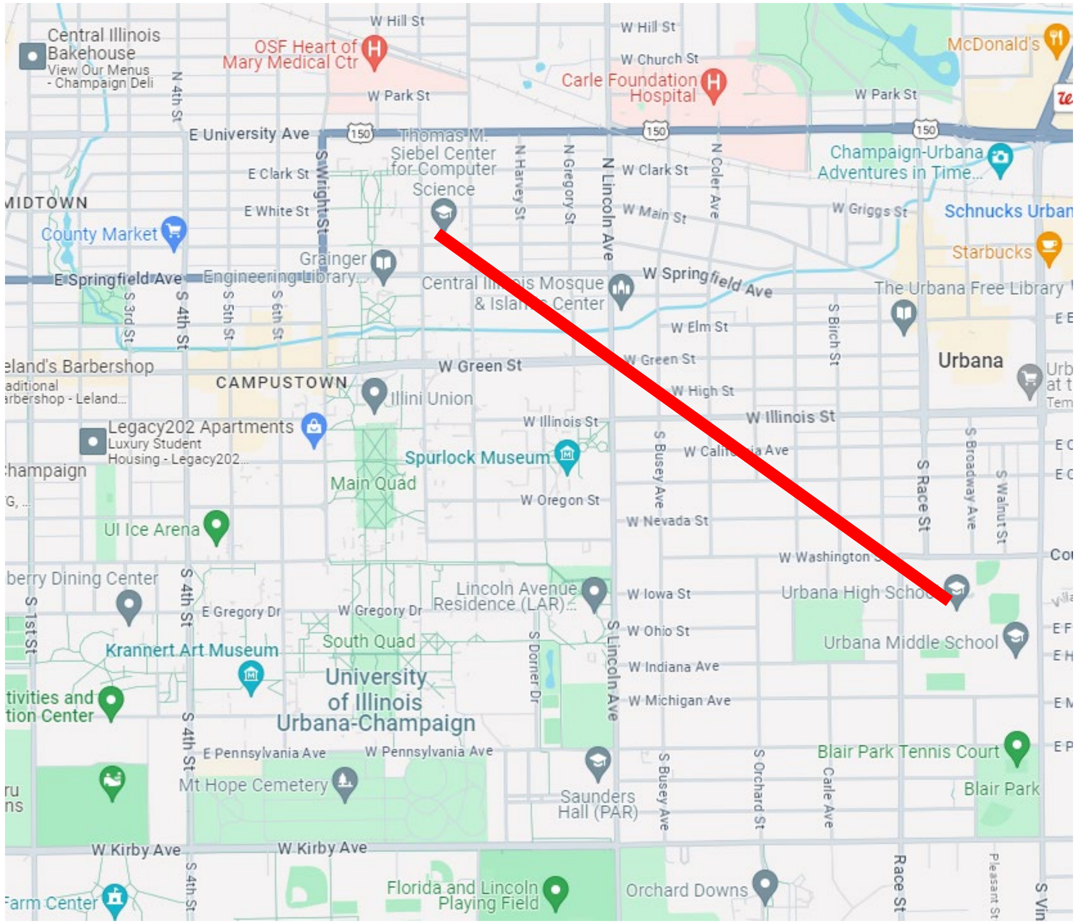


Probably Aggressive

Common Distance/Similarity Measures

- L2: Euclidean

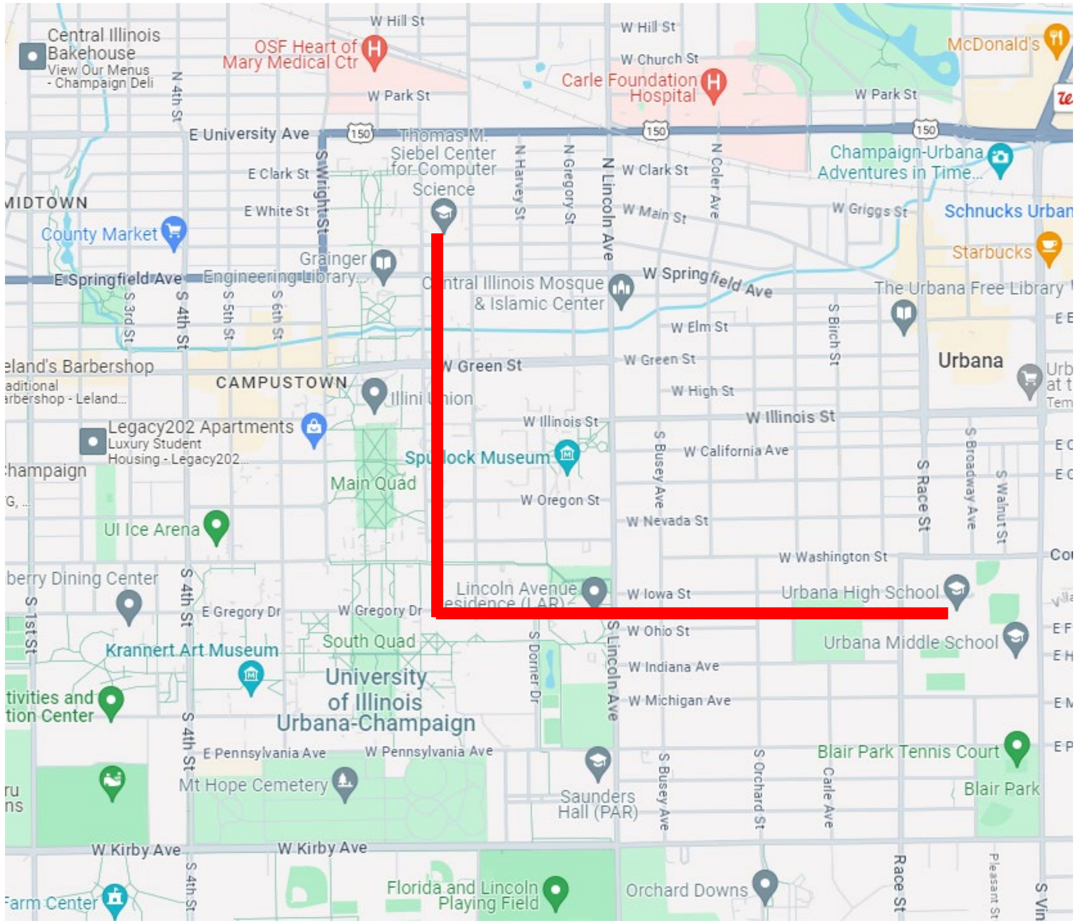
$$d_2(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2$$
$$= \sqrt{\sum_i (x_i - y_i)^2}$$



Common Distance/Similarity Measures

- L1: City-Block

$$d_1(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_1$$
$$= \sum_i |x_i - y_i|$$



Common Distance/Similarity Measures

- Dot product, Cosine

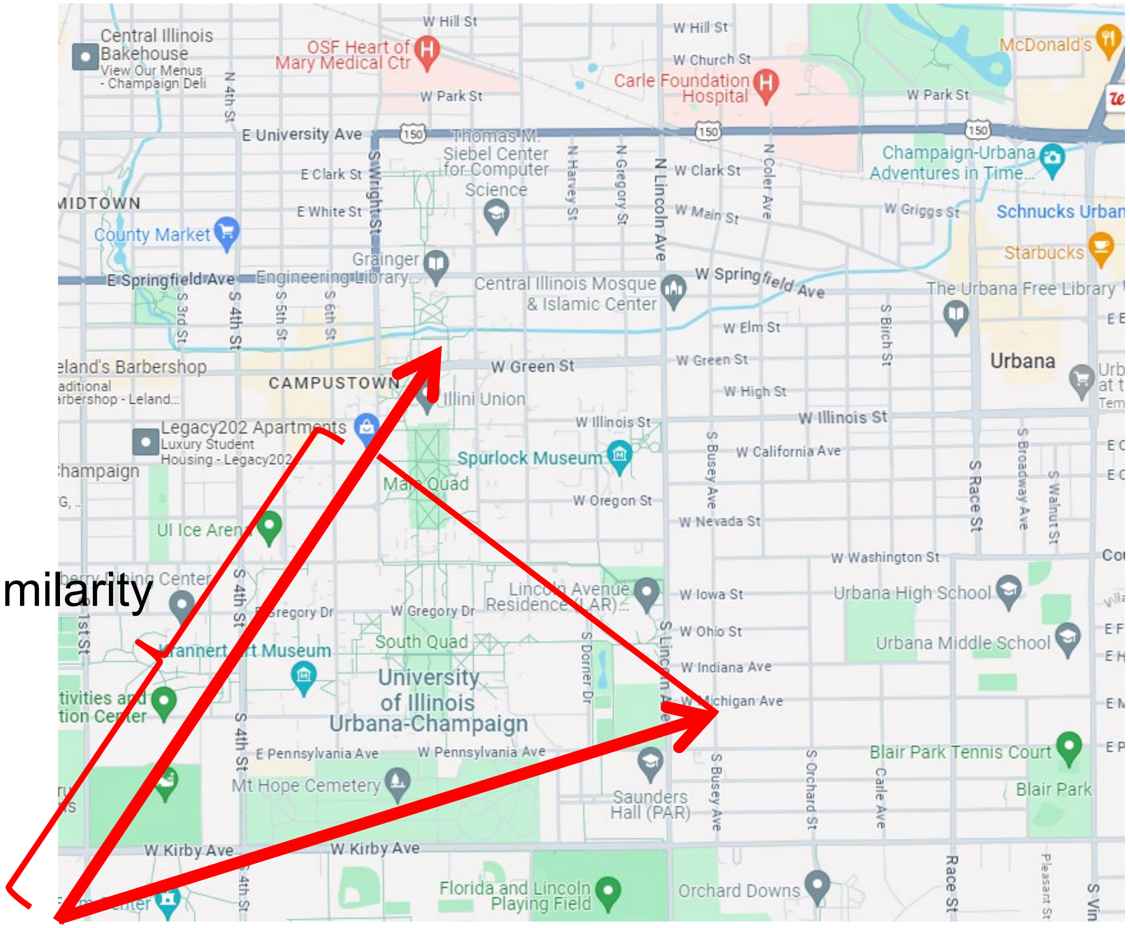
Dot product (or inner product)

$$s_{dot}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y} = \sum_i x_i y_i$$

Cosine similarity

$$s_{cos}(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^T \boldsymbol{y}}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2}$$

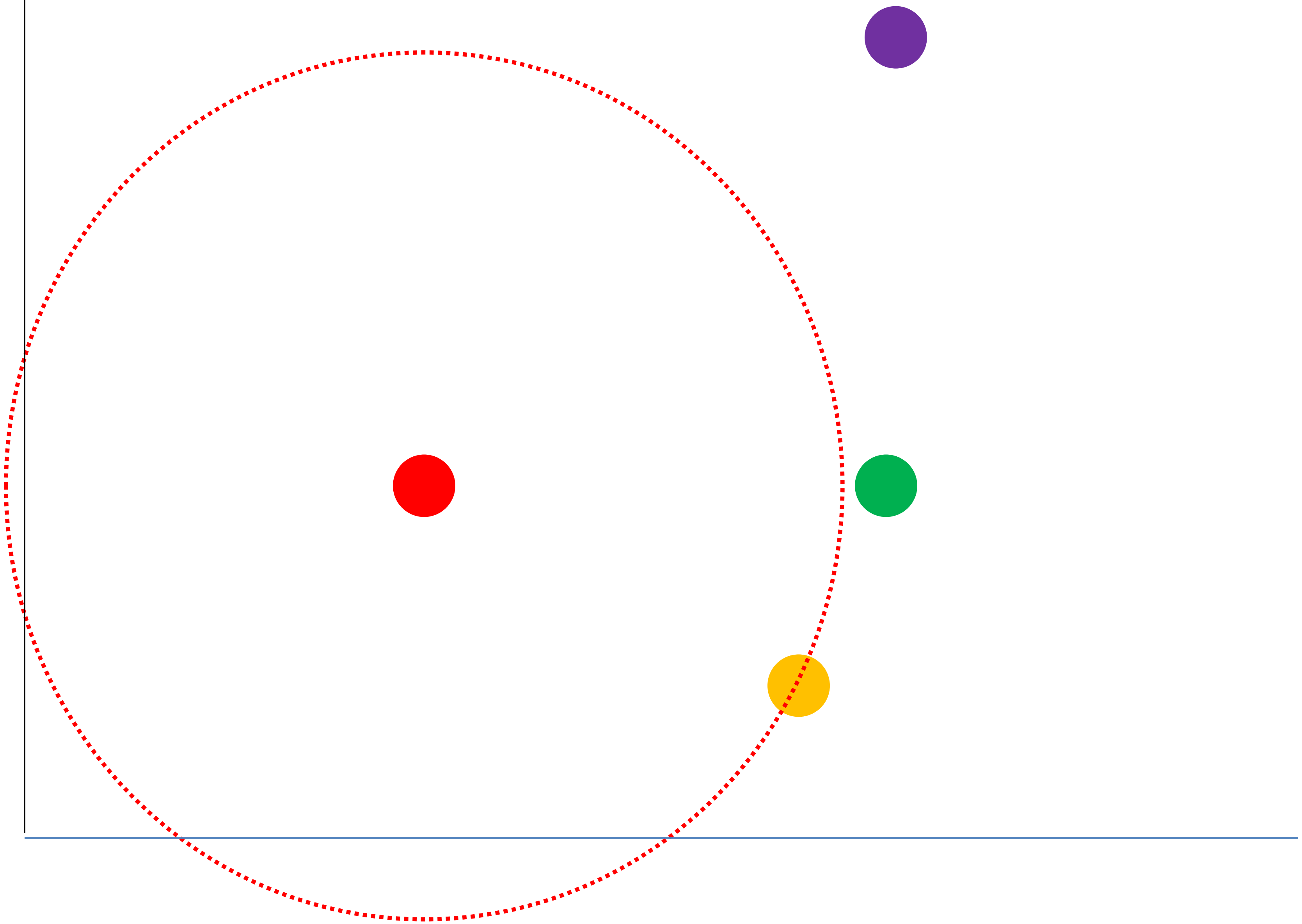
Cosine similarity



Dot product: how far does one vector go in the direction of the other vector

Cosine similarity: how similar are the two directions

Which is closest to the red circle under L1, L2, and cosine distance?



Comparing distance/similarity functions

- L2 depends much more heavily than L1 on the coordinates with the biggest differences

$$d_2([0\ 100],[5\ 1]) = 99.1$$
$$d_1([0\ 100],[5\ 1]) = 104$$

- Cosine and L2 are equivalent if the vectors are unit length

$$\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 = \underbrace{\boldsymbol{x}^T\boldsymbol{x}}_{1} - 2\boldsymbol{x}^T\boldsymbol{y} + \underbrace{\boldsymbol{y}^T\boldsymbol{y}}_{1} = 2(1 - s_{cos}(\boldsymbol{x},\boldsymbol{y}))$$

Retrieval

- Given a new sample, find the closest sample in a dataset

- Applications
  - Finding information (web search)
  - Prediction (e.g. nearest neighbor algorithm)
  - Clustering (kmeans)

“Brute force” search

- Compute distance between query and each dataset point and return closest point

Brute force search pseudo-code

```
getNearest(x_q, X)

dist_min = Inf
idx_min = -1

For each nth sample in X:
   dist = sum((X[n]-x_q)**2) # sum square diff
   if dist < dist_min:
      dist_min = dist
      idx_min = n
return idx_min
```

FAISS library makes even brute force search very fast

- Multi-threading, BLAS libraries, SIMD vectorization, GPU implementations
- KNN for MNIST takes seconds

```
import faiss                    # make faiss available
index = faiss.IndexFlatL2(d)    # build the index, d=size of vectors
# here we assume xb contains a n-by-d numpy matrix of type float32
index.add(xb)                   # add vectors to the index
print index.ntotal
```

```
# xq is a n2-by-d matrix with query vectors
k = 4                           # we want 4 similar vectors
D, I = index.search(xq, k)      # actual search
print I
```

https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

Locality Sensitive Hashing (LSH)

A fast approximate search method to return similar data points to query

Basic LSH process

1. Convert each data point into an array of bits or integers, using the same conversion process/parameters for each

2. Map the arrays into buckets (e.g. with 10 bits, you have 2^10 buckets)
   – Can use subsets of arrays to create multiple sets of buckets

3. On query, return points in the same bucket(s)
   – Can check additional buckets by flipping bits to find points within hash distances greater than 0

Random Projection LSH

Data Preparation

Given data $\{X\}$ with dimension d:

1. Center data on origin (subtract mean)
2. Create b random vectors h_b of length d
3. Convert each X_n to b bits: $\mathrm{X_n} h^\mathrm{T} > 0$

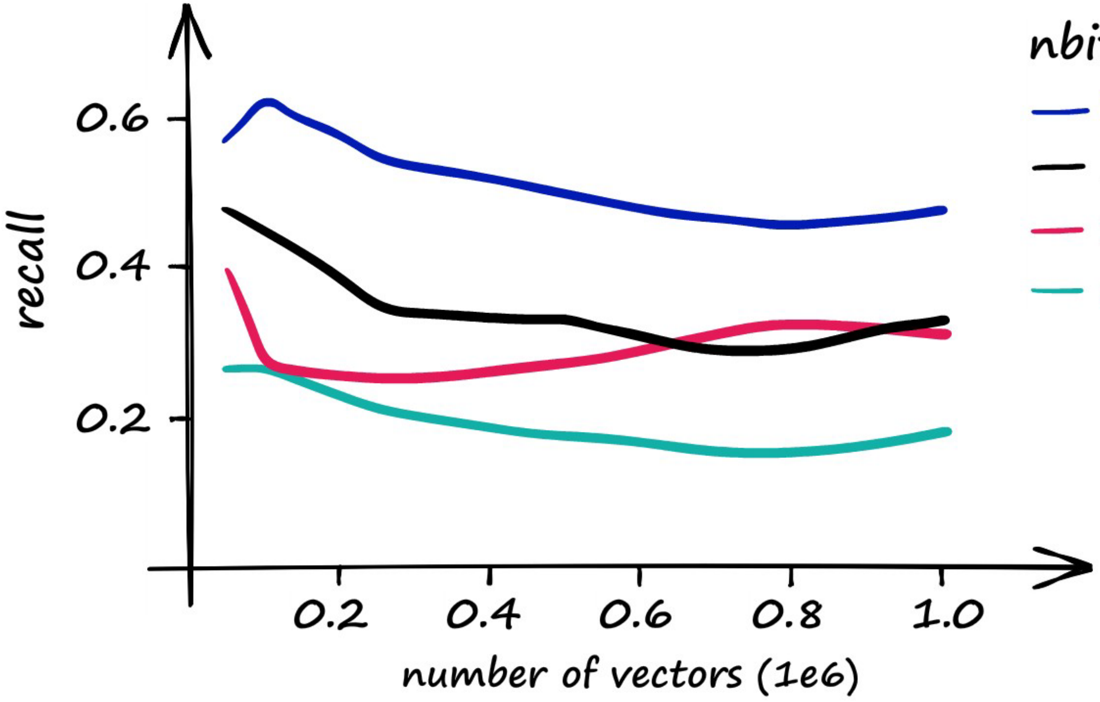
```
h= np.random.rand(nbits, d) - .5
```

Query

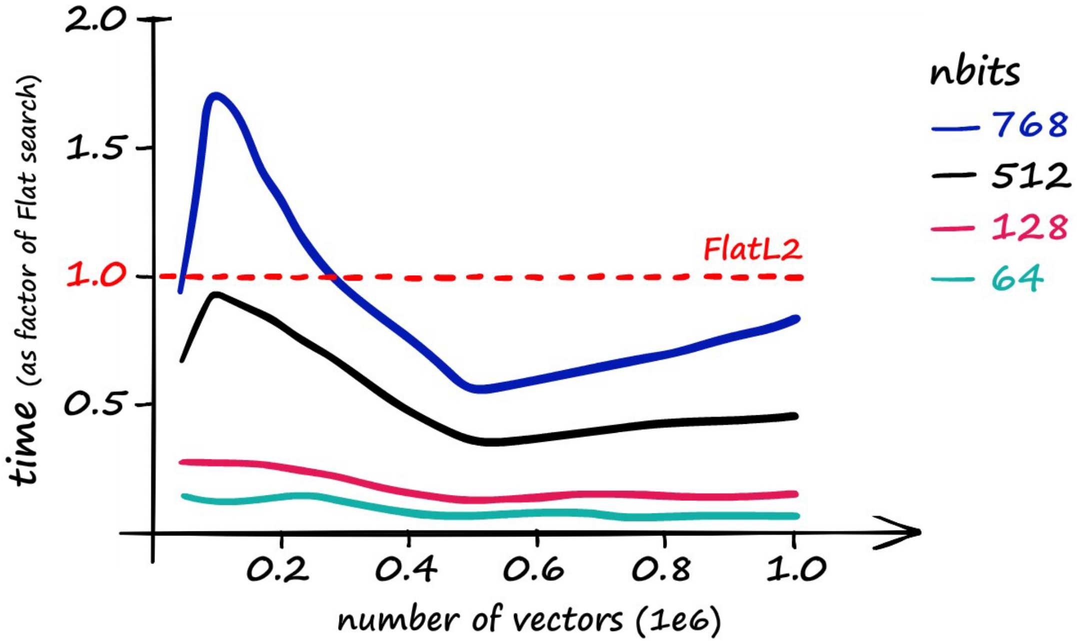
1. Convert X_q to bits using h
2. Check buckets based on bit vector and similar bit vectors to return most similar data points

Key parameter: nbits

- Rule of thumb: nbits = dim is a decent choice (1 bit per feature dimension)
- Optionally, can retrieve K closest data points and then use brute force search on those



Recall vs. exact nearest neighbor



Time compared to brute force search

Nice video about LSH in faiss:
https://youtu.be/ZLfdQq_u7Eo

which is part of this very detailed and helpful post:
https://www.pinecone.io/learn/locality-sensitive-hashing-random-projection/

Inverse document file for retrieval of count-based docs

Applies to text (word counts), images (clustered keypoint counts), and other tokenized representations

- Like a book index: keep a list of all the words (tokens) and all the pages (documents) that contain them.

- Rank database documents based on summed tf-idf measure for each word/token in the query document

tf-idf: Term Frequency – Inverse Document Frequency

times word appears in document

documents

$$t_i = \frac{n_{id}}{n_d} \log \frac{N}{n_i}$$

documents that contain the word

words in document

Clustering

- Assign a label to each data point based on the similarities between points

- Why cluster
  - Represent data point with a single integer instead of a floating point vector
    - Saves space
    - Simple to count and estimate probability
  - Discover trends in the data
  - Make predictions based on groupings

K-means algorithm

1. Randomly select K centers

2. Assign each point to nearest center (L2 dist)

3. Compute new center (mean) for each cluster

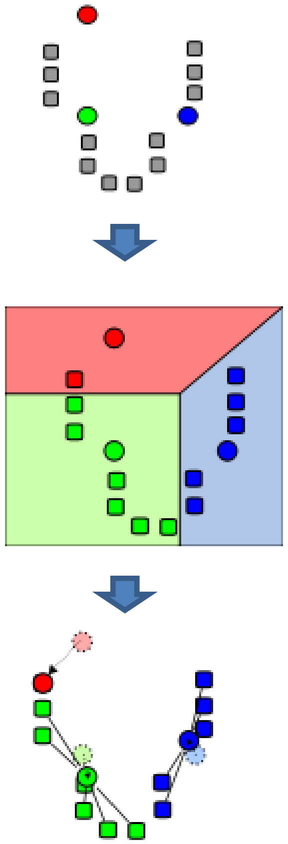


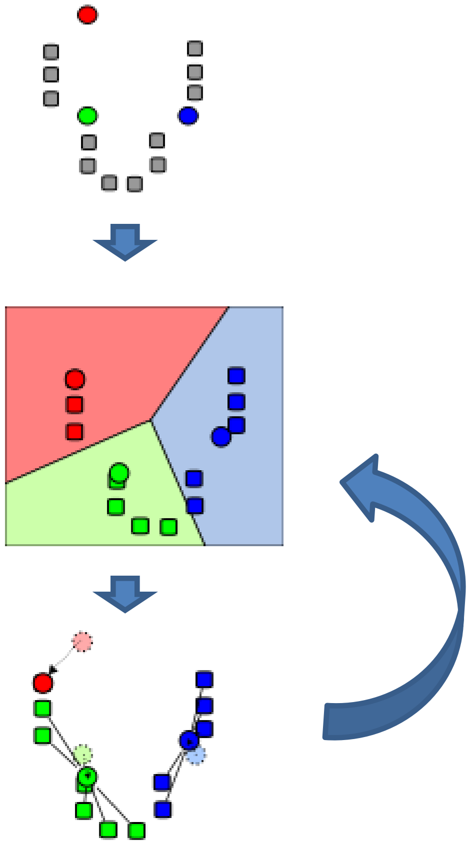
Illustration: http://en.wikipedia.org/wiki/K-means_clustering

K-means algorithm

1. Randomly select K centers

2. Assign each point to nearest center (L2 dist)

3. Compute new center (mean) for each cluster



Back to 2

Illustration: http://en.wikipedia.org/wiki/K-means_clustering

Pseudo-code

```
kmeans(X, K, maxiter)

# Create cluster centers
center = X[:K]

Until maxiter iterations or convergence:

   For each nth sample in X:
       # get index of nearest center
       idx[n]  = get_nearest(X[n], centers)

   For each kth center:
       # get mean of data points assigned to cluster k
       center[k] = X[idx==k].mean(axis=0)

   Convergence is if no idx changed in this iteration

return center, idx
```

What is the cost minimized by K means?

$$id^*, centers^* = argmin_{id,centers} \sum_n \left\| centers_{id_n} - X_n \right\|^2$$

1. Choose ids that minimizes square cost given centers
2. Choose centers that minimize square cost given ids

K-means Demo

https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

2 minute break

What are some disadvantages of K-means in terms of clustering quality?

What are some disadvantages of K-means in terms of clustering quality?

- All feature dimensions equally important
- Tends to break data into clusters of similar numbers of points (can be good or bad)
- Does not take into account any local structure
- Typically, not an easy way to choose K
- Can be slow if the number of data points and clusters is large

Implementation issues

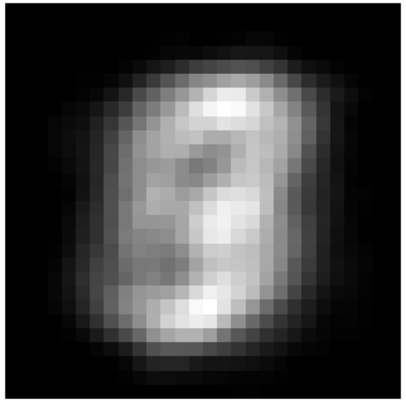
- How to choose K?
  - Typically chosen by hand
  - Often based on the number of clusters you want considering time/space requirements
- How do you initialize
  - Randomly choose points
  - Iterative furthest point

Evaluating clustering with RMSE

- RMSE = root mean squared error
- Measures a kind of compression loss, i.e. how well is each data point represented by the closest cluster center

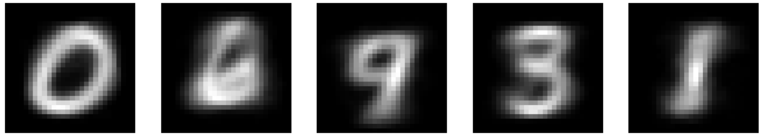
$$RMSE(X, C) = \sqrt{\frac{1}{N} \sum_{n \in \{0..N-1\}} \min_k \|C_k - X_n\|_2{}^2}$$

Example: K-means on MNIST

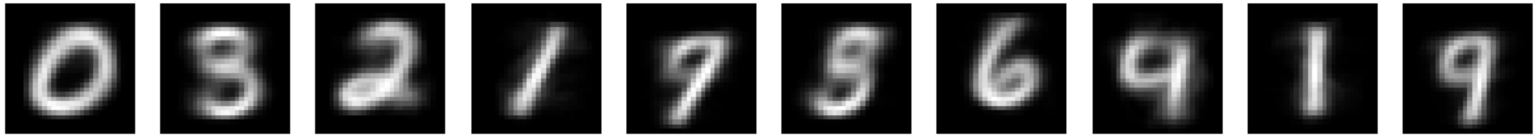


K=1, RMSE = 55.0

K=3, RMSE = 49.3

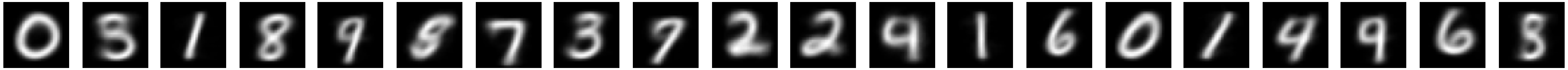


K=5, RMSE = 45.6

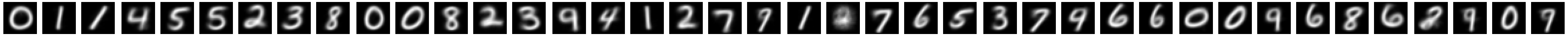


K=10, RMSE = 41.9

K=20, RMSE = 37.82



K=40, RMSE = 34.44



Evaluating clusters with purity

- We often cluster when there is no definitively correct answer, but a *purity* measure can be used to check the consistency with labels

$$purity = \sum_k \left(\max_y \sum_{n:id_n=k} \delta(label_n = y) \right) / N$$

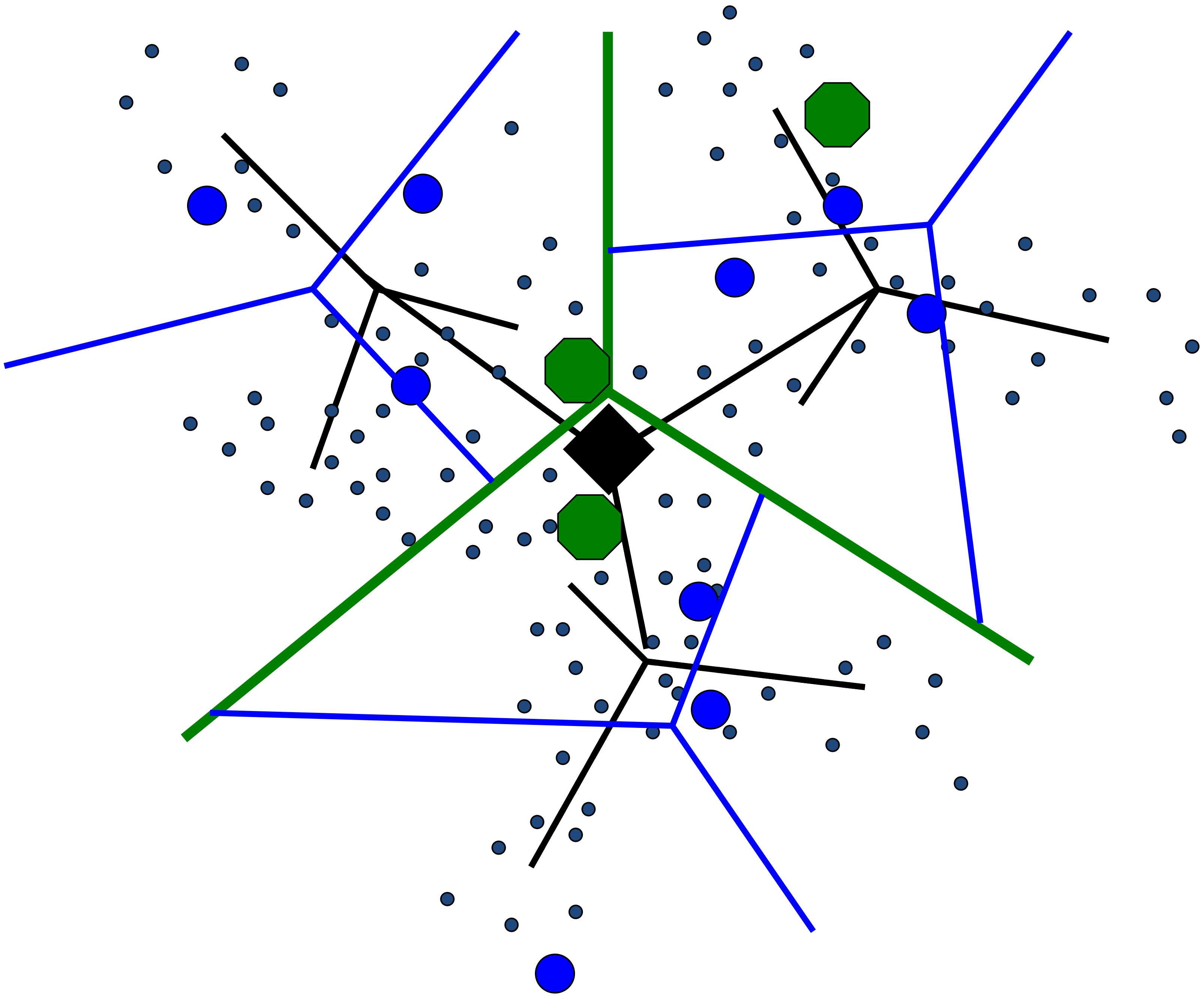
- Purity is the count of data points with the most common label in each cluster, divided by the total number of data points (N)

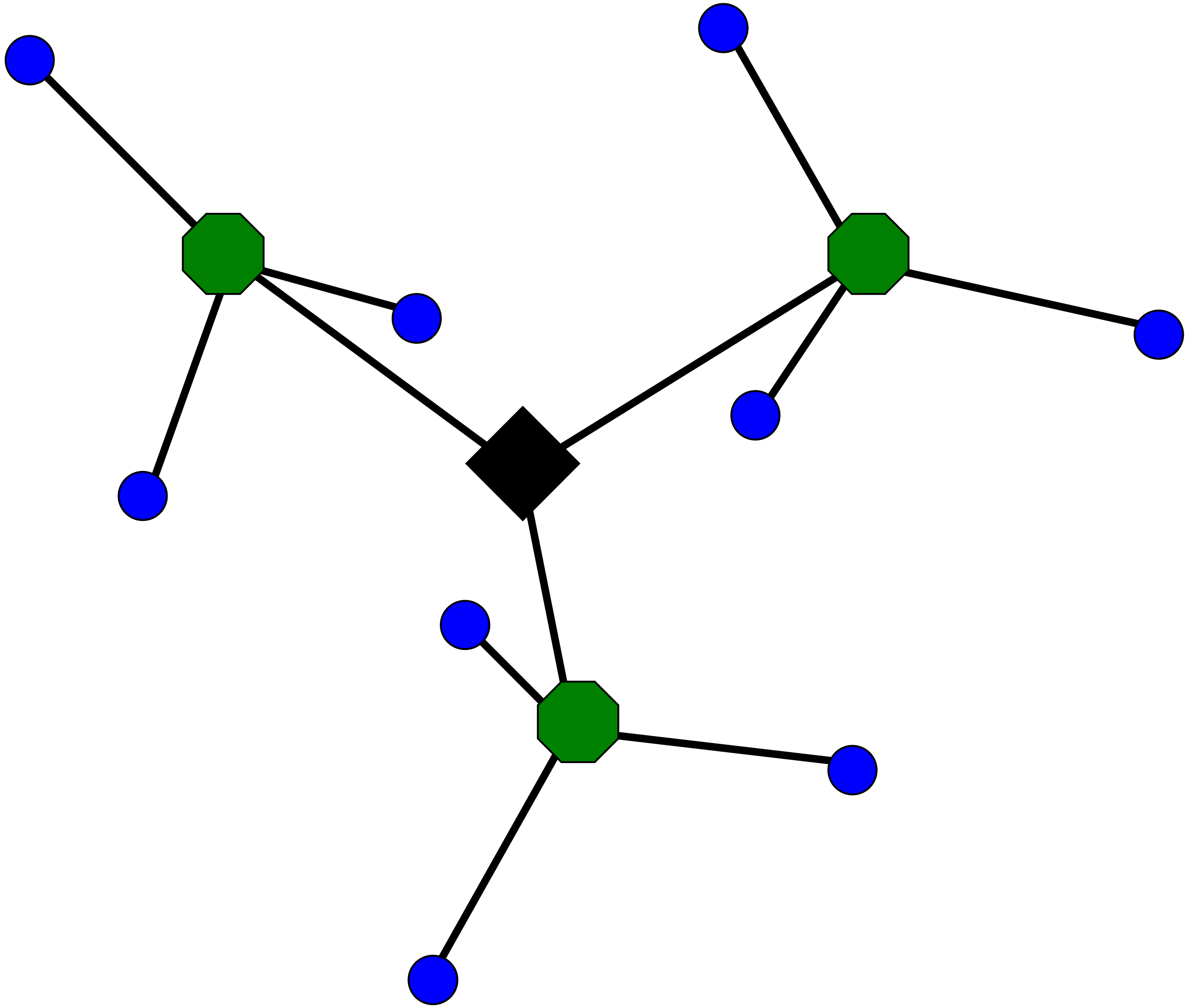
- E.g., labels = {0, 0, 0, 0, 1, 1, 1, 1}, cluster ids = {0, 0, 0, 0, 0, 1, 1, 1}, purity = ?
  purity = 7/8

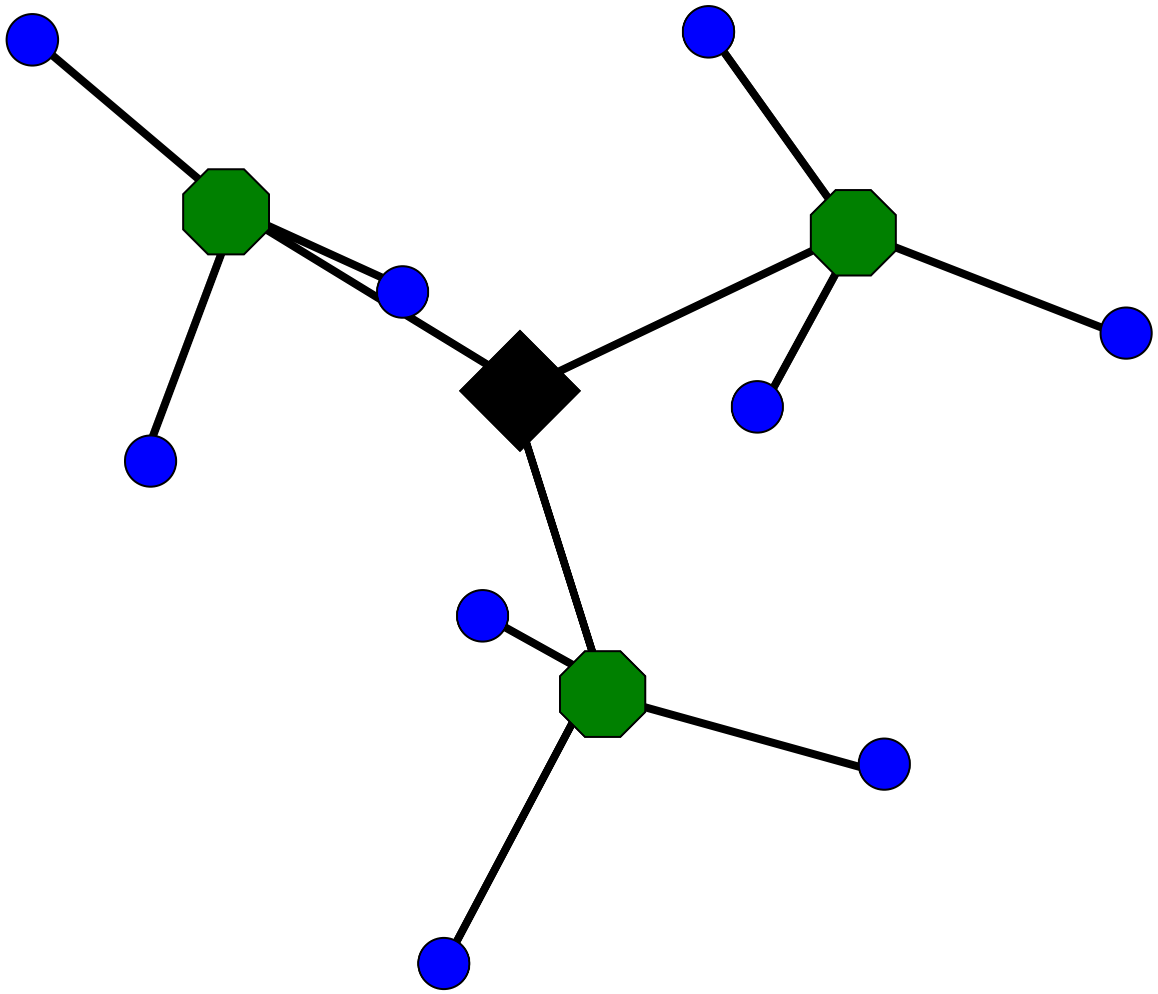
- Purity can be used to select the number of clusters, or to compare approaches with a given number of clusters
  - A relatively small number of labels can be used to estimate purity, even if there are many data points

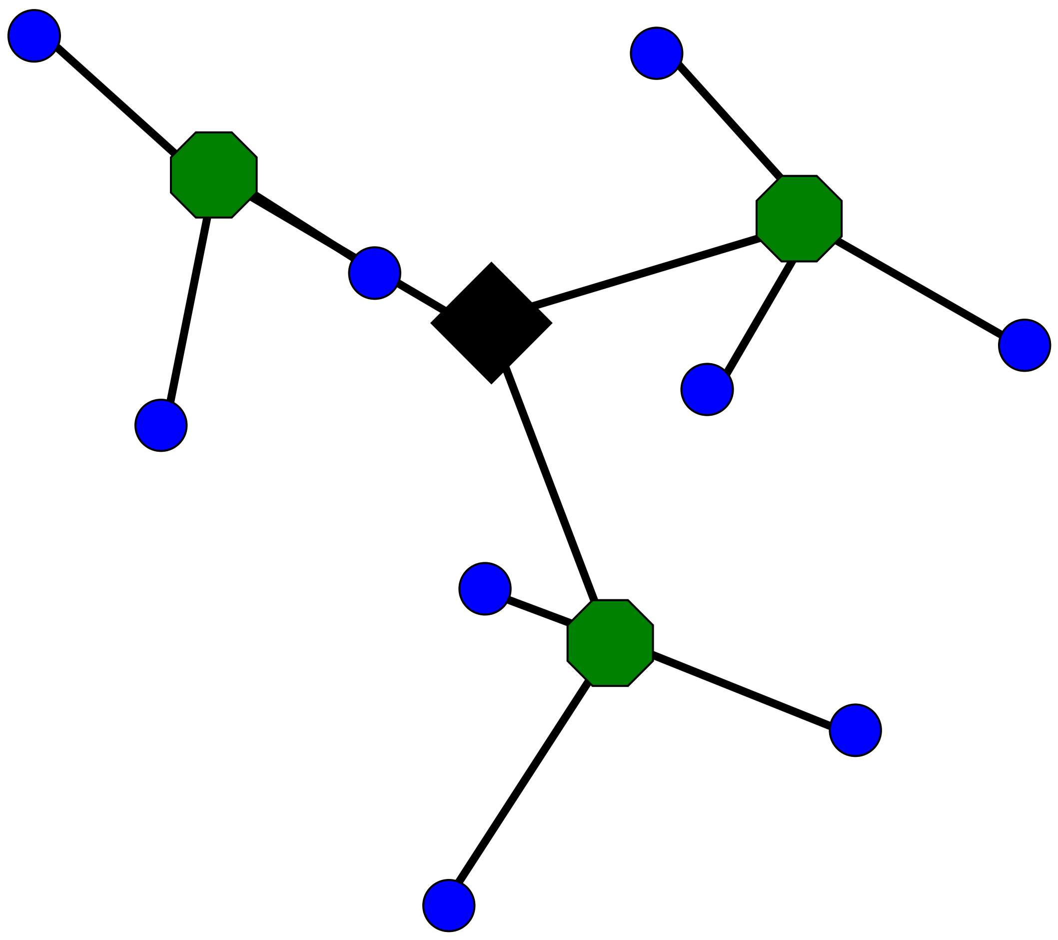
Hierarchical K-means

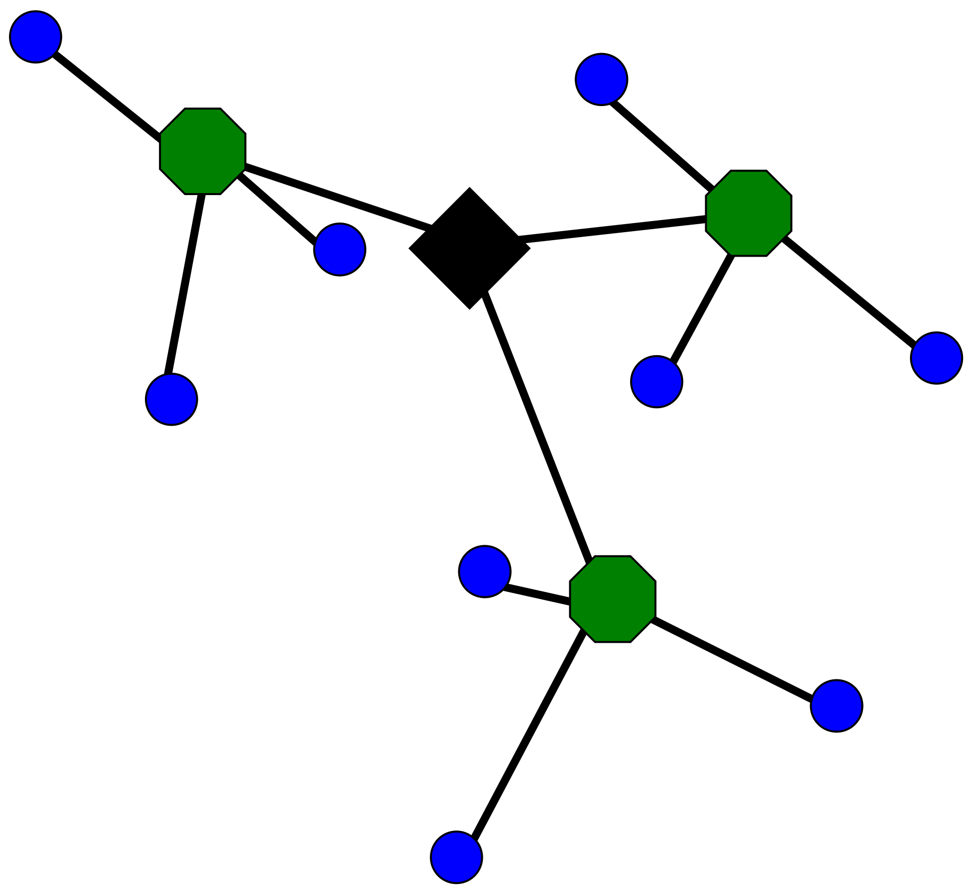
- Iteratively cluster points into K groups, then cluster each group into K groups

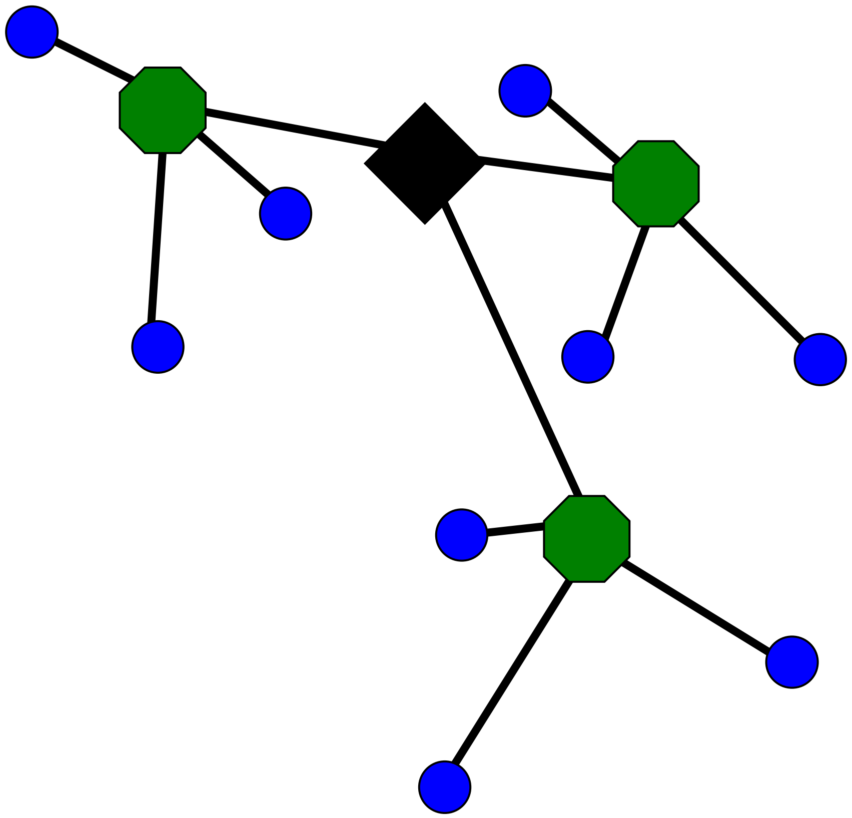


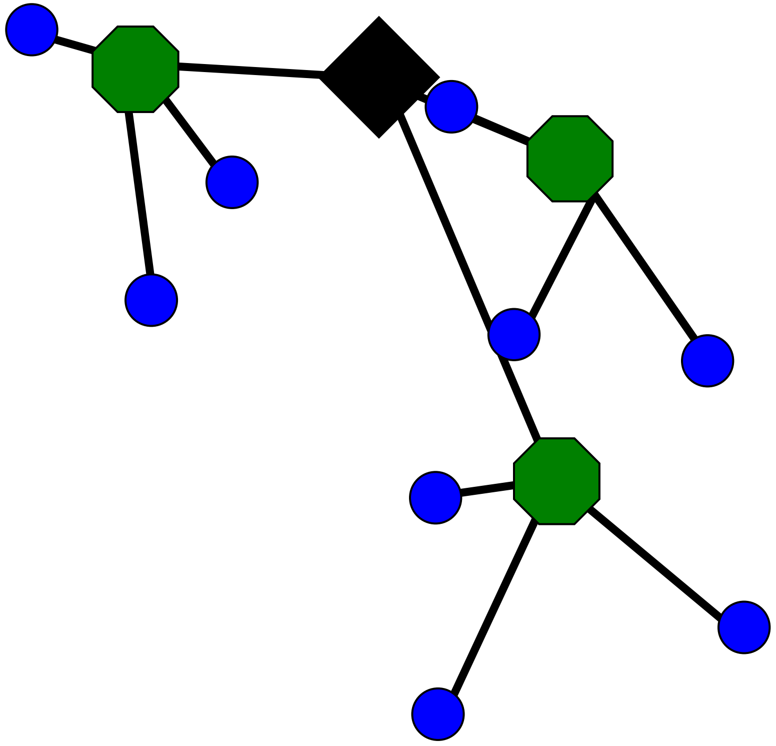


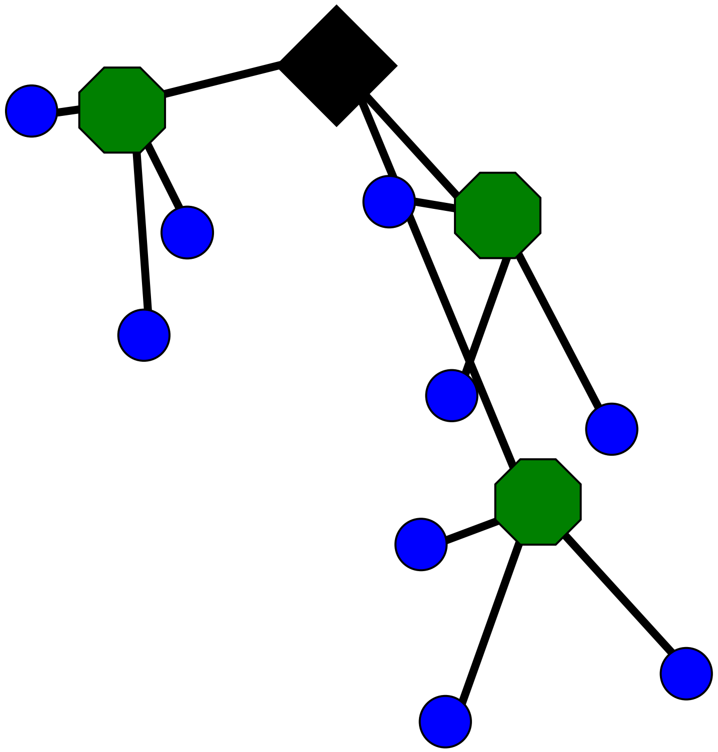


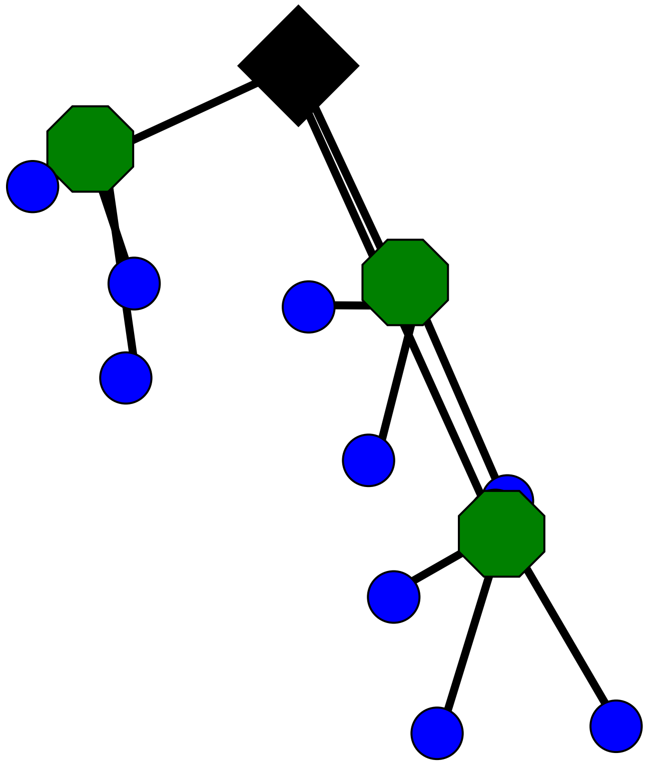


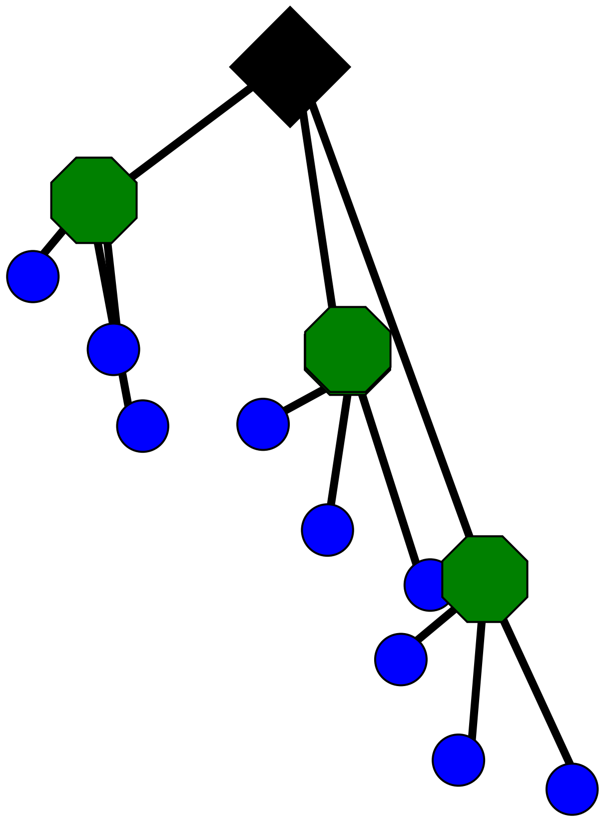


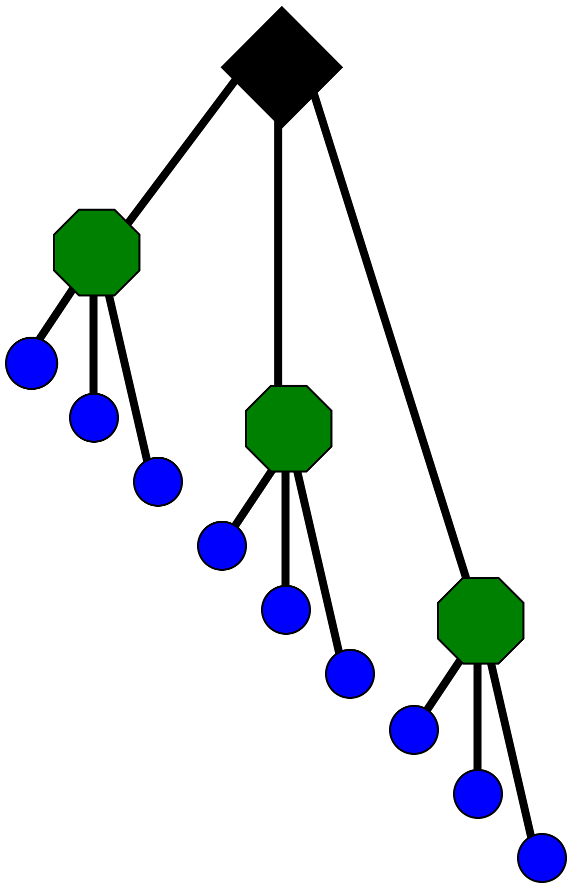












Advantages of Hierarchical K-Means

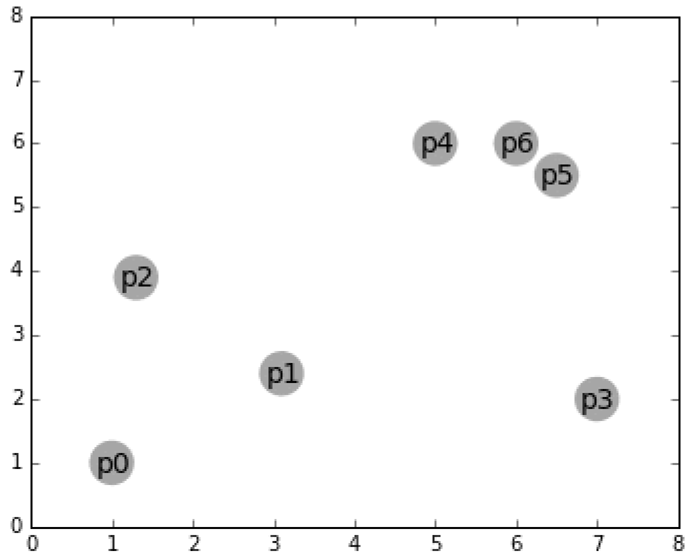
- Fast cluster training
  - With a branching factor of 10, can cluster into 1M clusters by clustering into 10 clusters ~111,111 times, each time using e.g. 10K data points
  - Vs. e.g. clustering 1B data points into 1M clusters
  - Kmeans is O(K*N*D) per iteration so this is a 900,000x speedup!
- Fast lookup
  - Find cluster number in O(log(K)*D) vs. O(K*D)
  - 16,667x speedup in the example above

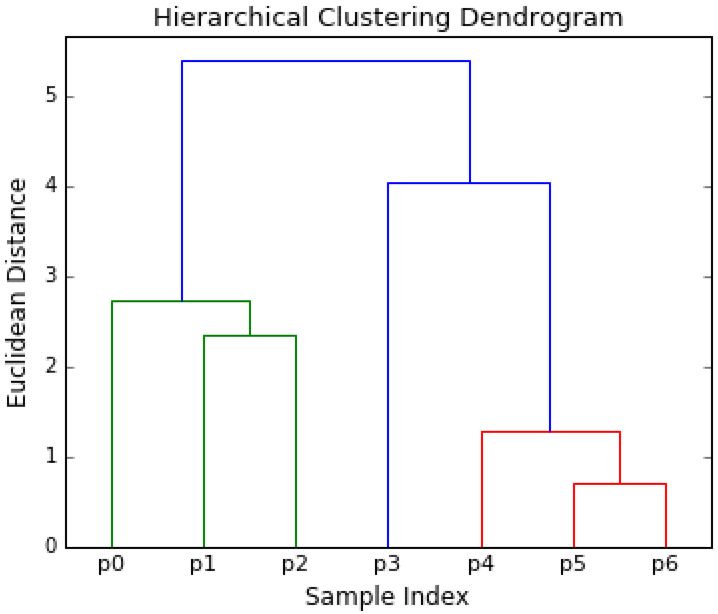
Are there any disadvantages of hierarchical Kmeans?

Yes, the assignment might not be quite as good, but often usually isn’t a huge deal since K means is used to approximate data points with centroid anyway

Agglomerative clustering

- Iteratively merge the two most similar points or clusters
  - Can use various distance measures
  - Can use different “linkages”, e.g. distance of nearest points in two clusters or the cluster averages
  - Ideally the minimum distance between clusters should increase after each merge (e.g. if using the distance between cluster centers)
  - Number of clusters can be set based on when the cost to merge increases suddenly

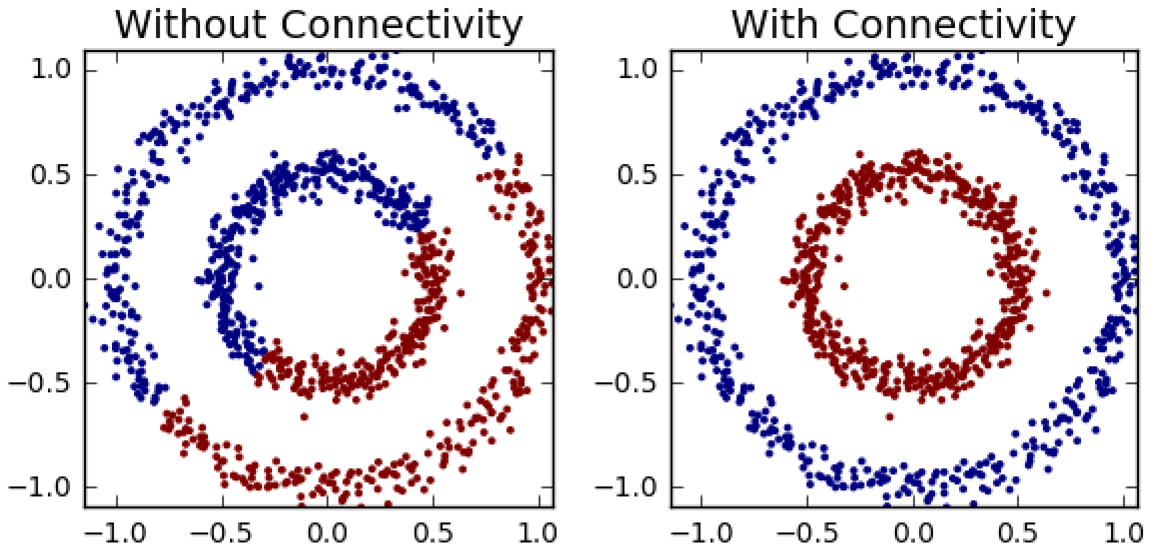




https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/

Agglomerative clustering

- With good choices of linkage, agglomerative clustering can reflect the data connectivity structure ("manifold")



Clustering based on distance of 5 nearest neighbors between clusters

https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/

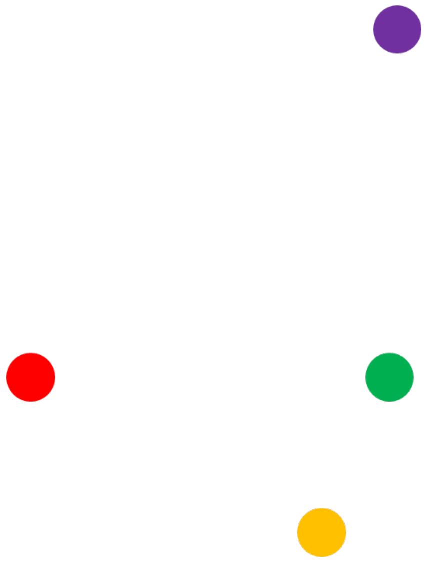
Applications of clustering

- K-means
  - Quantization (codebooks for image generation)
  - Search
  - Data visualization (show the average image of clusters of images)

- Hierarchical K-means
  - Fast search (document / image search)

- Agglomerative clustering
  - Finding structures in the data (image segmentation, grouping camera locations together)

Things to remember



- Similarity is foundational to machine learning
- Use highly optimized libraries like FAISS for search/retrieval
- Approximate search methods like LSH can be used to find similar points quickly
- TF-IDF is used for similarity of tokenized documents and used with index for fast search
- Clustering groups similar data points
- K-means is the must-know method, but there are many others