Dall-E

Clustering and Retrieval

Applied Machine Learning
Derek Hoiem

Recall from last lecture: What are three ways to mitigate problems in bias from AI algorithms?

- Model cards
- Data sheets
- Intersectional performance analysis
- Adversarial algorithm to prevent using features that predict sensitive attributes such as race or gender
- Multi-task learning to facilitate learning of less common classes

We’ve learned a lot about supervised prediction algorithms – now we will learn about methods to organize and analyze data without labels

- Clustering and retrieval
- EM and missing data
- Estimating probabilities for continuous variables
- Projections for compression and visualization
- Topic modeling
- Anomaly detection

Today’s lecture

- Clustering
  - Kmeans
  - Hierarchical Kmeans
  - Agglomerative Clustering
- Retrieval
  - Using Hierarchical Kmeans
  - LSH
  - Faiss library

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

3. Compute new center (mean) for each cluster

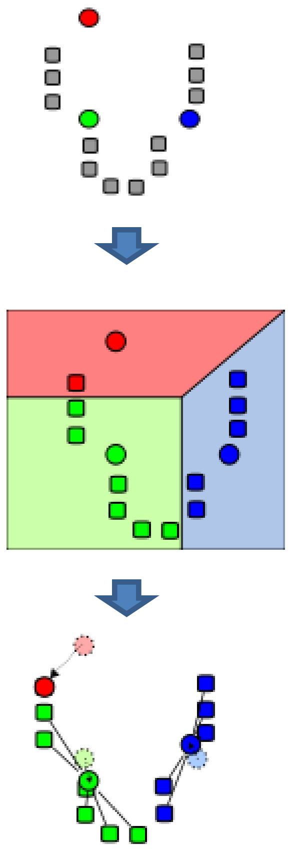
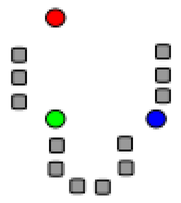


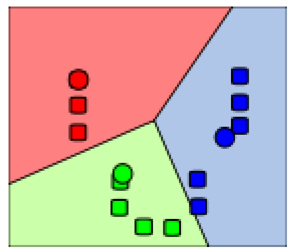
Illustration: http://en.wikipedia.org/wiki/K-means_clustering

K-means algorithm

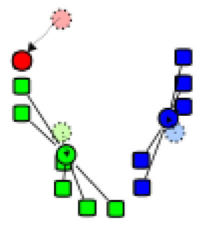
1. Randomly select K centers



2. Assign each point to nearest center



3. Compute new center (mean) for each cluster



Back to 2

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

K-means Demo

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

What is the cost minimized by K means?

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

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

Implementation issues

- How to choose K?
  - Can use MDL (minimum description length) principle that minimizes a cost of parameters plus cost of negative data log likelihood, but in practice K is almost always hand chosen
  - Often based on the number of clusters you want considering time/space requirements
- How do you initialize
  - Randomly choose points
  - Iterative furthest point

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

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

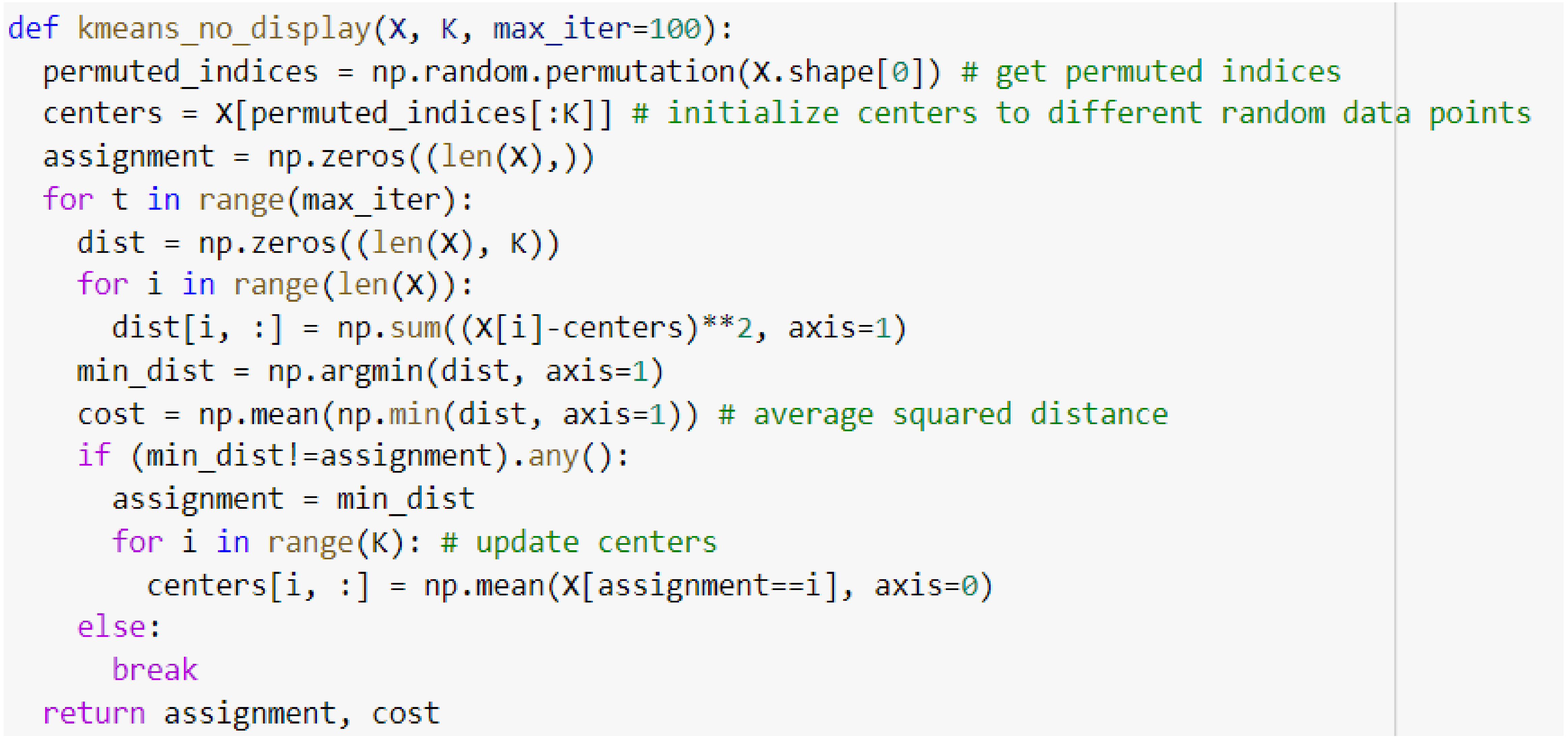
- Purity is the count of data points with the most common label in each cluster, divided by the total number of data points (N)

- 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

Kmeans code



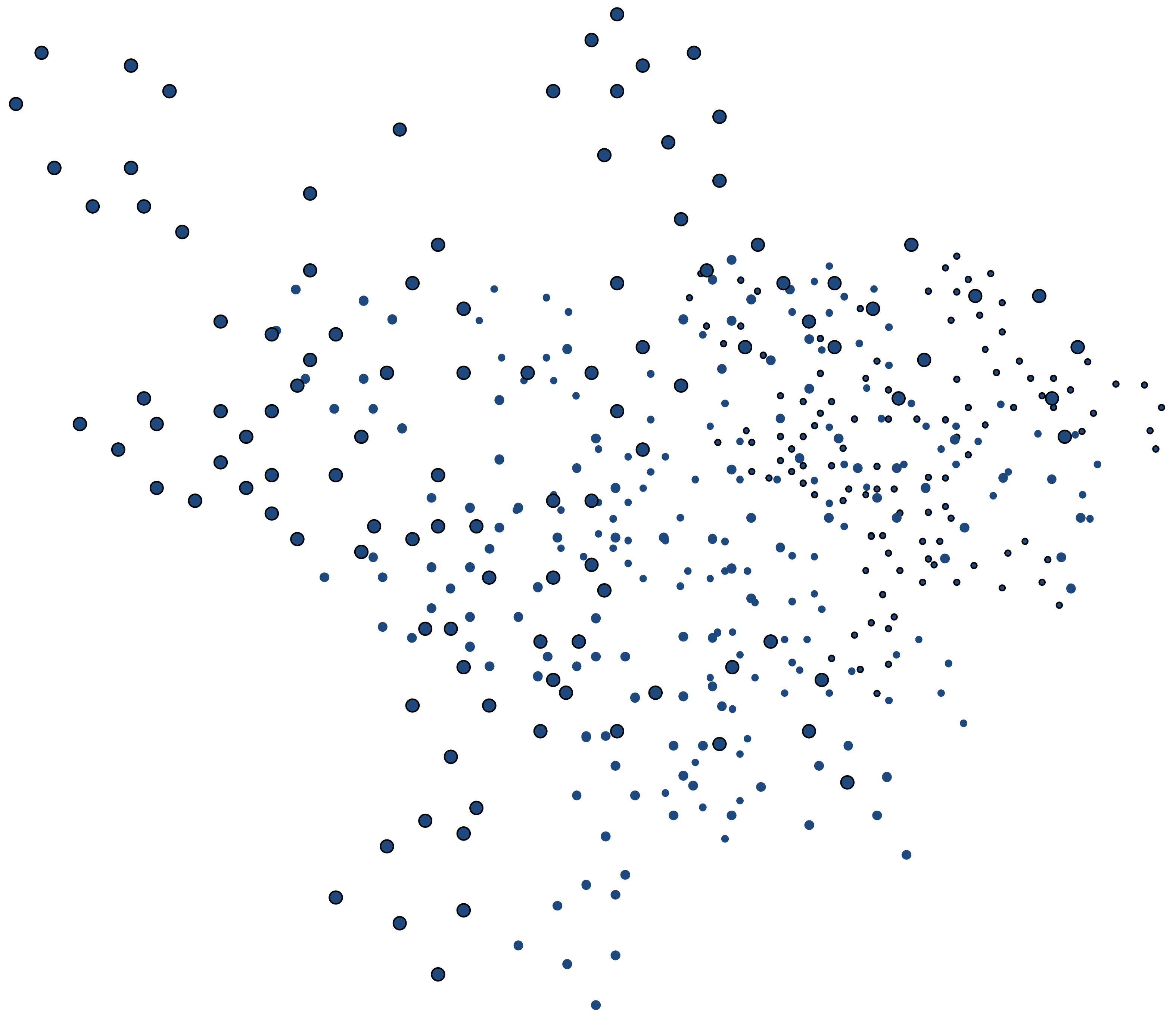
```
def kmeans_no_display(X, K, max_iter=100):
  permuted_indices = np.random.permutation(X.shape[0]) # get permuted indices
  centers = X[permuted_indices[:K]] # initialize centers to different random data points
  assignment = np.zeros((len(X),))
  for t in range(max_iter):
    dist = np.zeros((len(X), K))
    for i in range(len(X)):
      dist[i, :] = np.sum((X[i]-centers)**2, axis=1)
    min_dist = np.argmin(dist, axis=1)
    cost = np.mean(np.min(dist, axis=1)) # average squared distance
    if (min_dist!=assignment).any():
      assignment = min_dist
      for i in range(K): # update centers
        centers[i, :] = np.mean(X[assignment==i], axis=0)
    else:
      break
  return assignment, cost
```

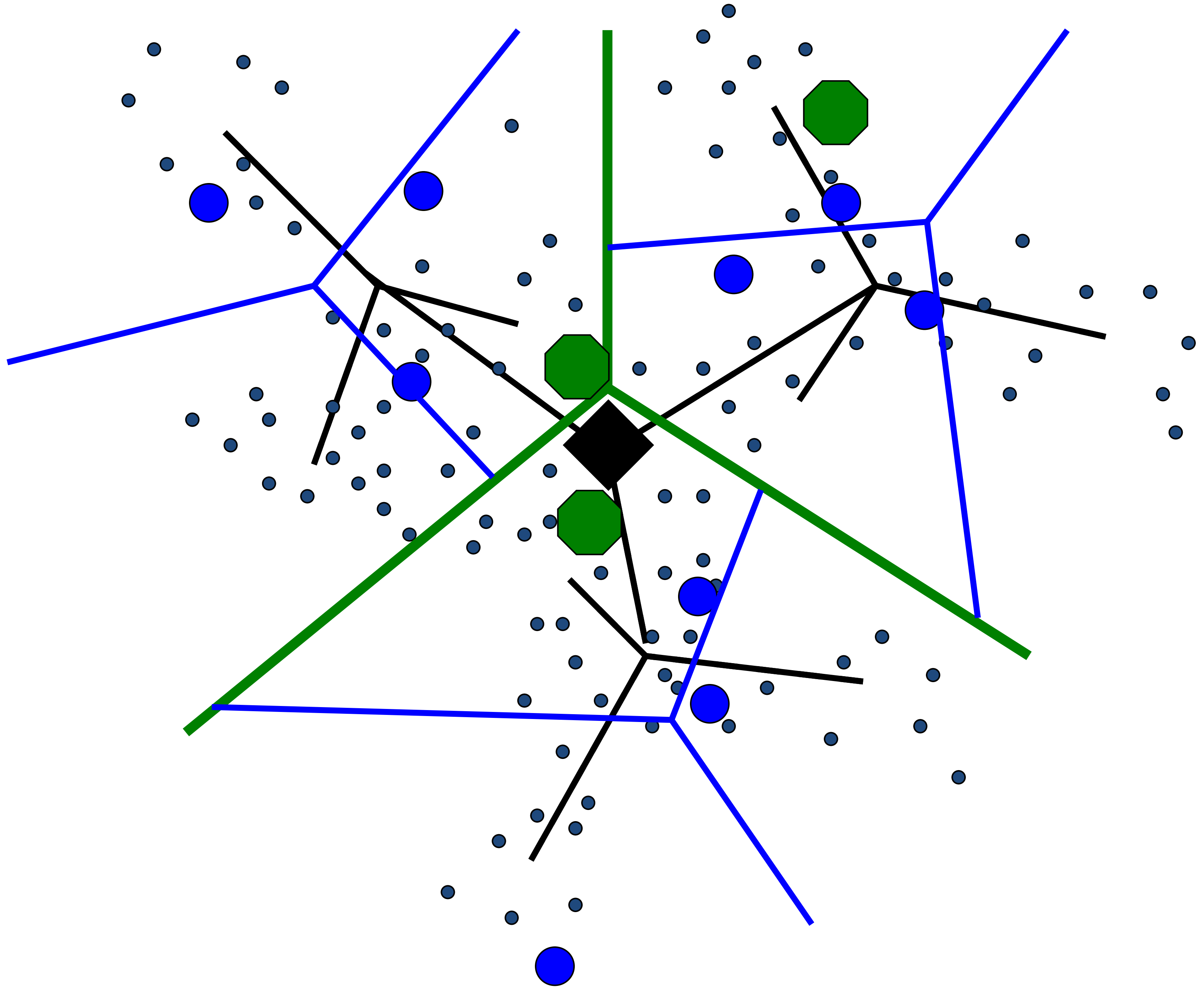
What are some disadvantages of K-means in terms of clustering quality?

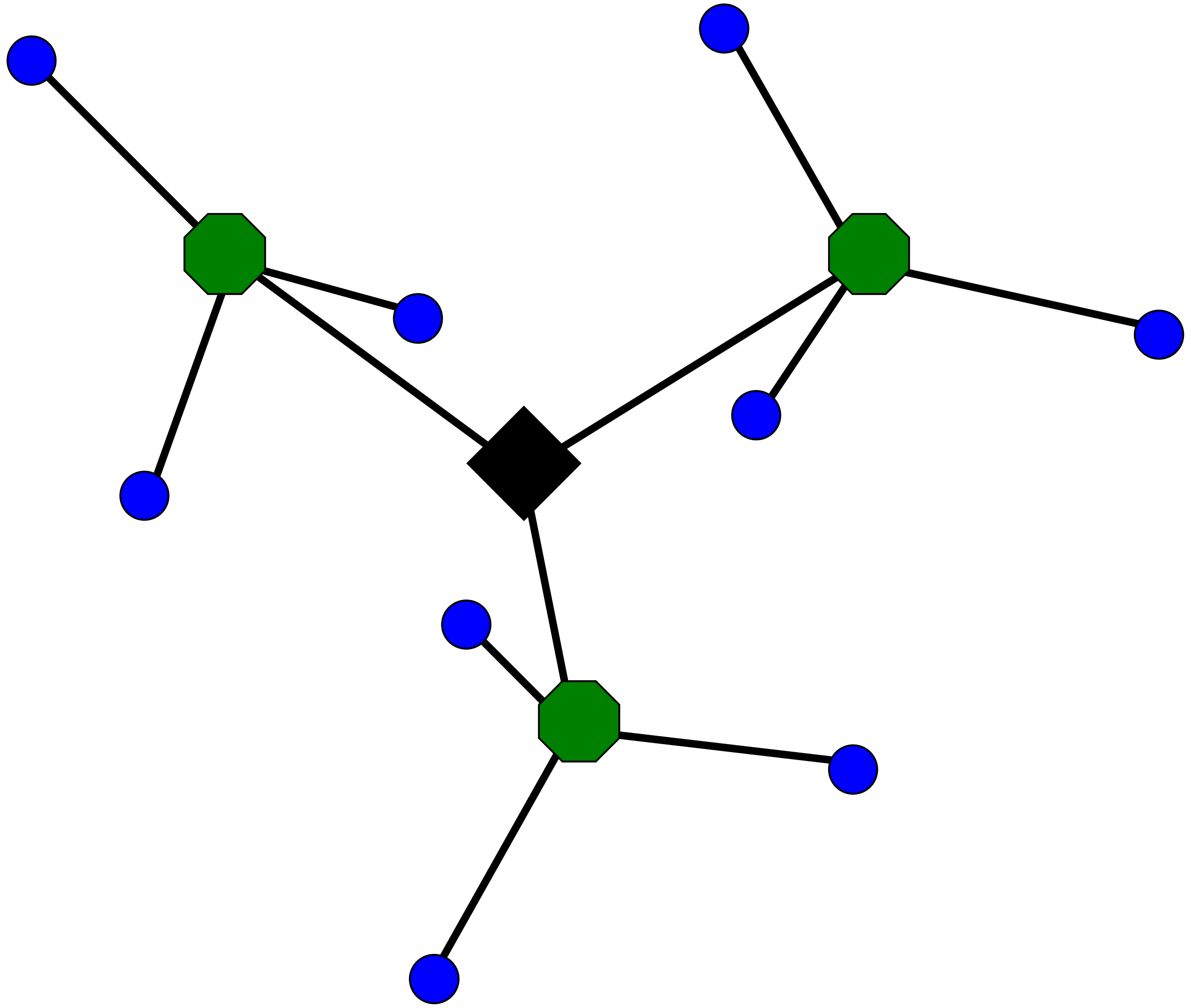
- All feature dimensions equally important
- Tends to break data into clusters of similar numbers of points (can be good or bad)
- Does not take into account any local structure
- Typically, not an easy way to choose K
- Can be very slow if the number of data points and clusters is large

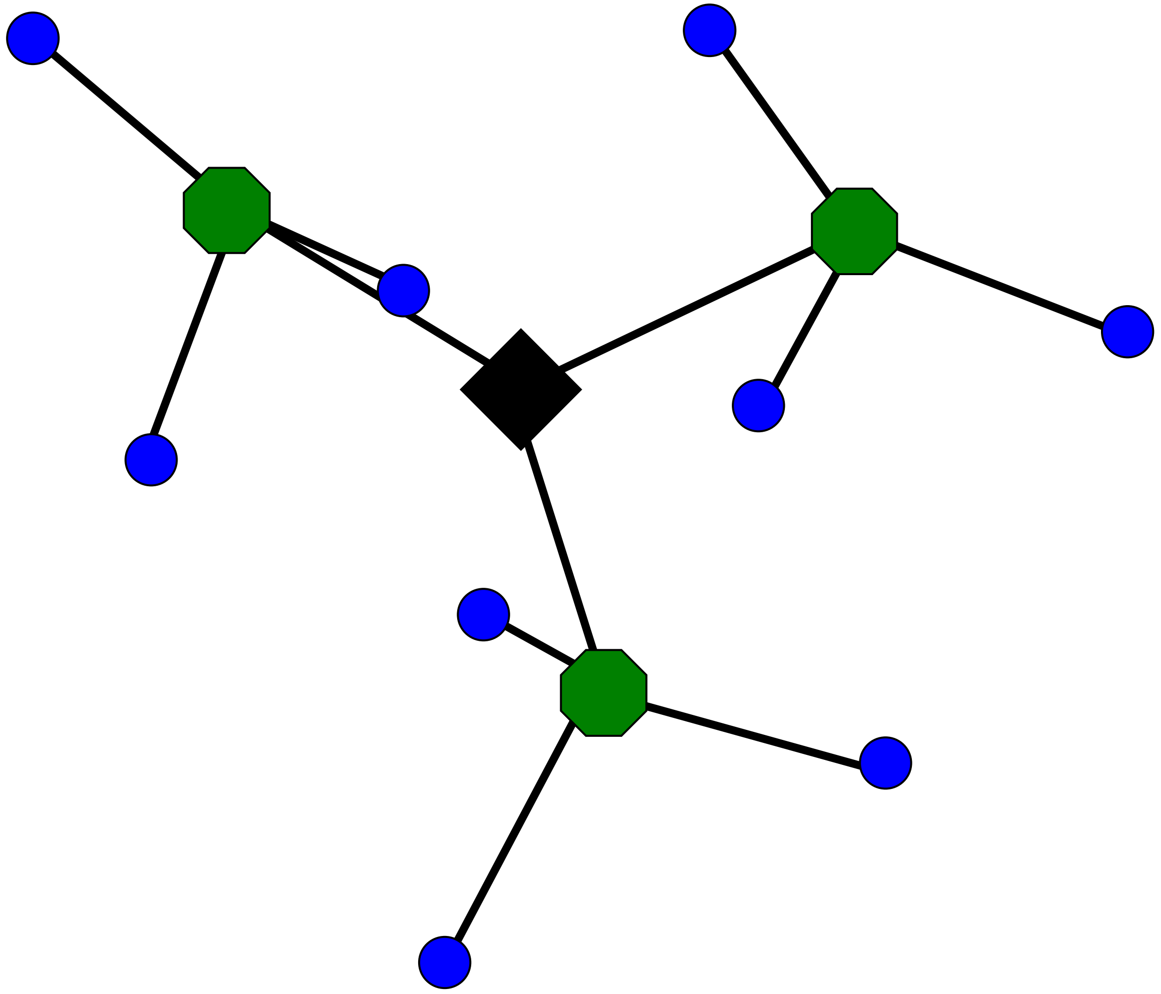
Hierarchical K-means

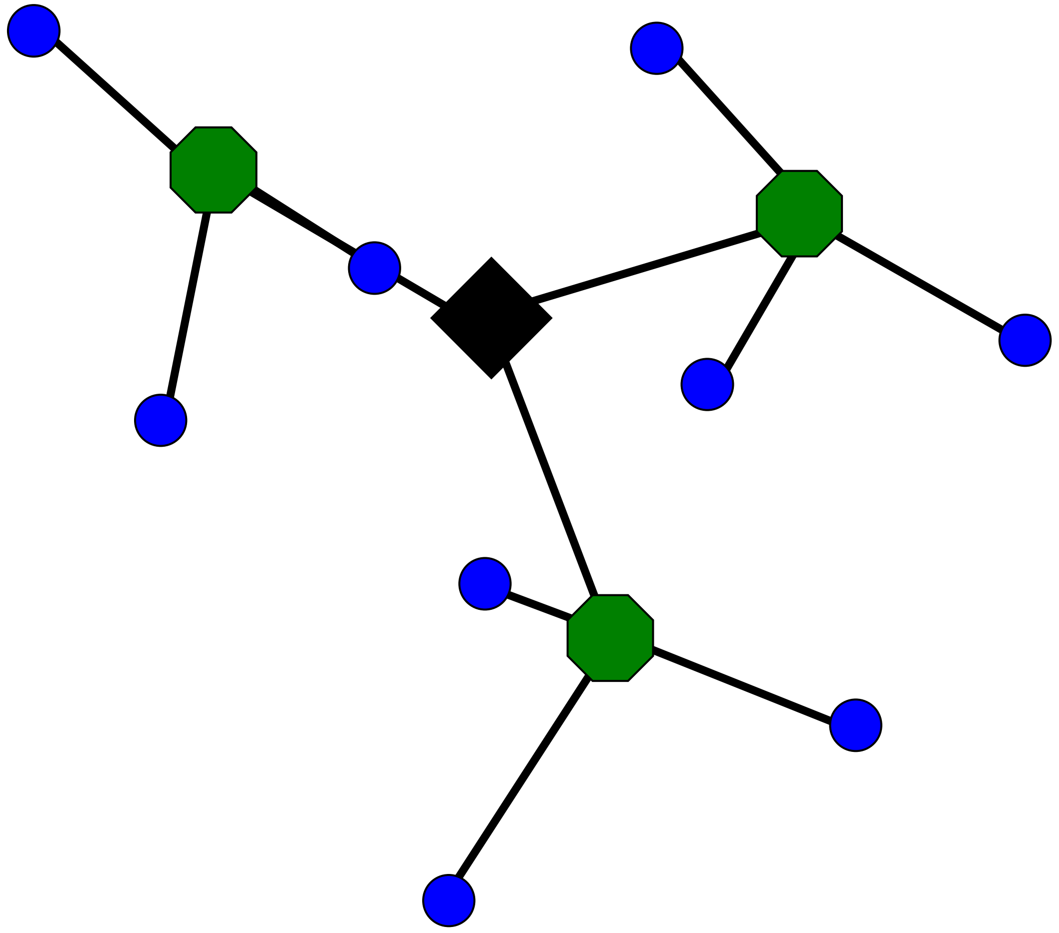
- Iteratively cluster points into K groups, then cluster each group into K groups

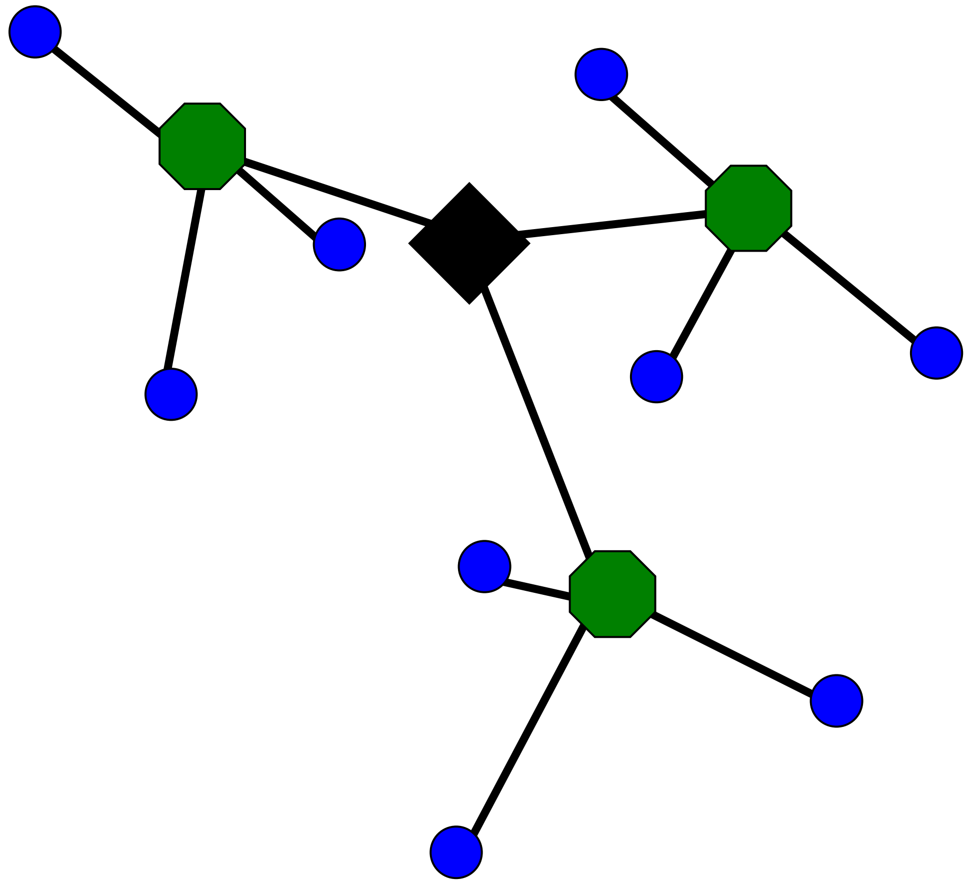


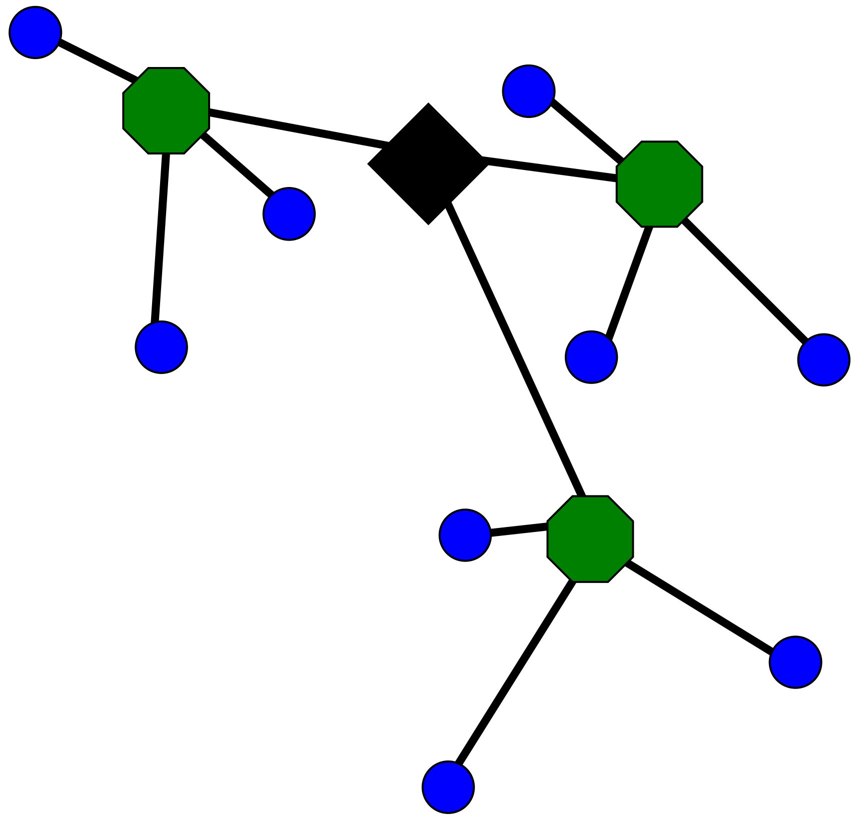


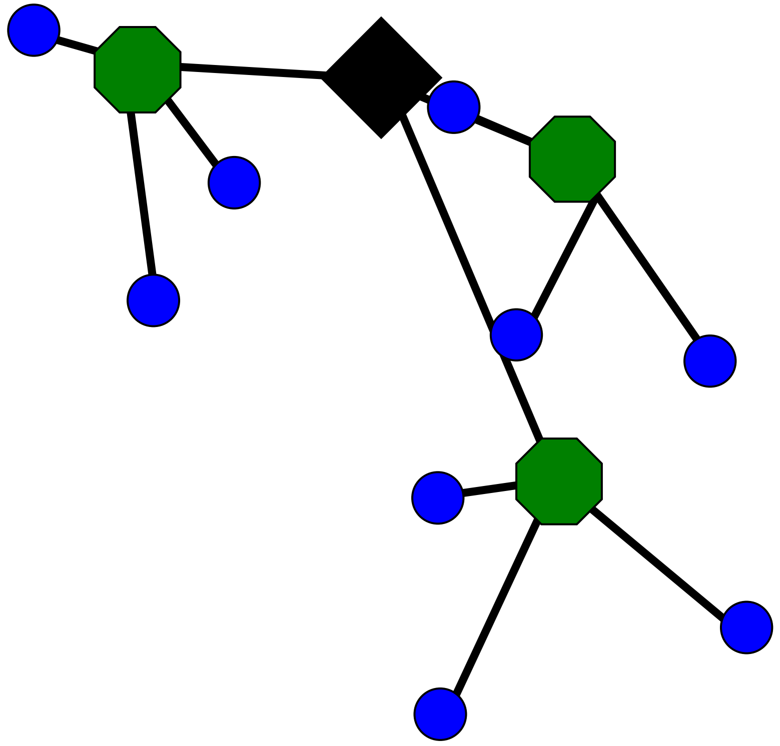


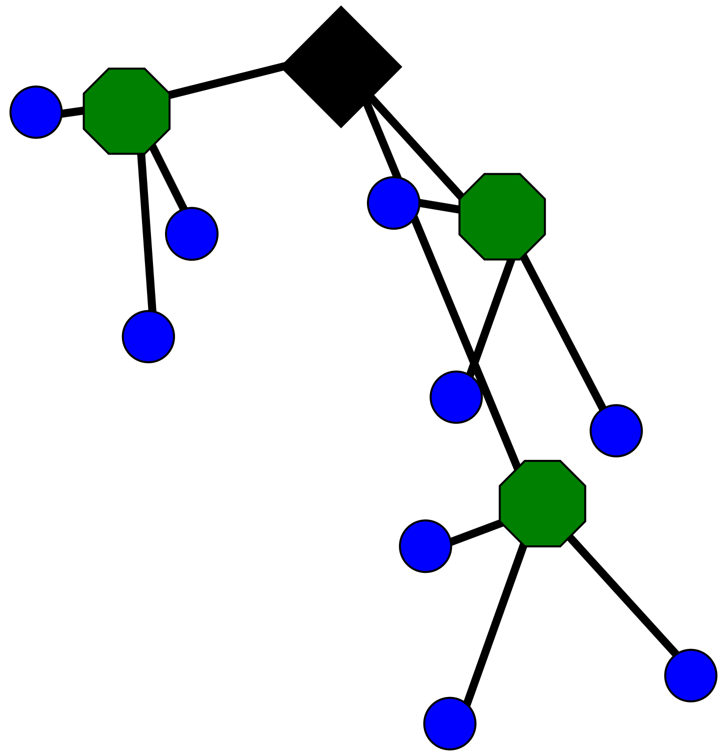


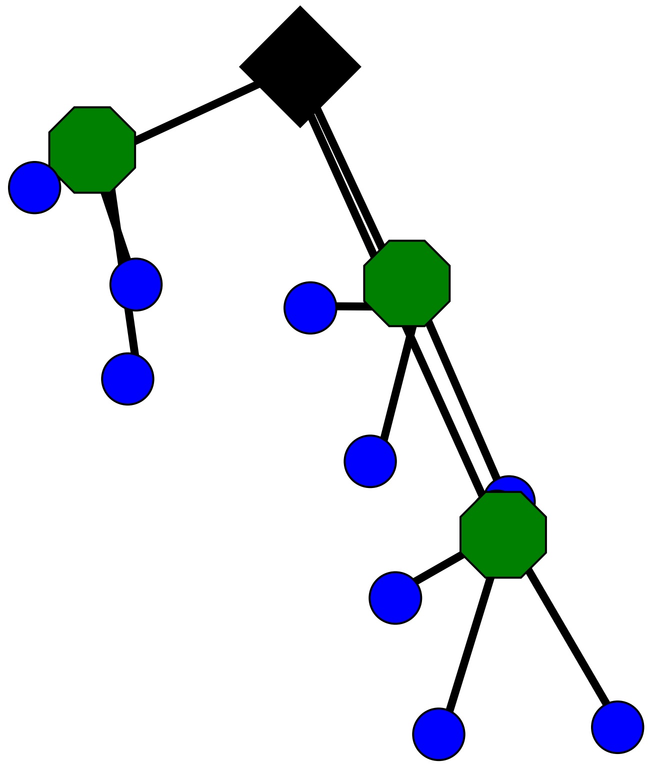


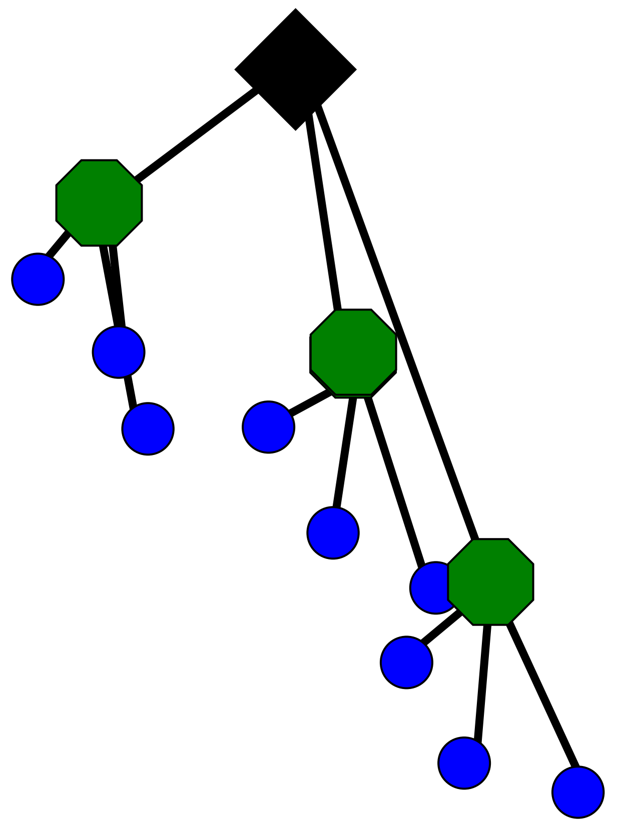


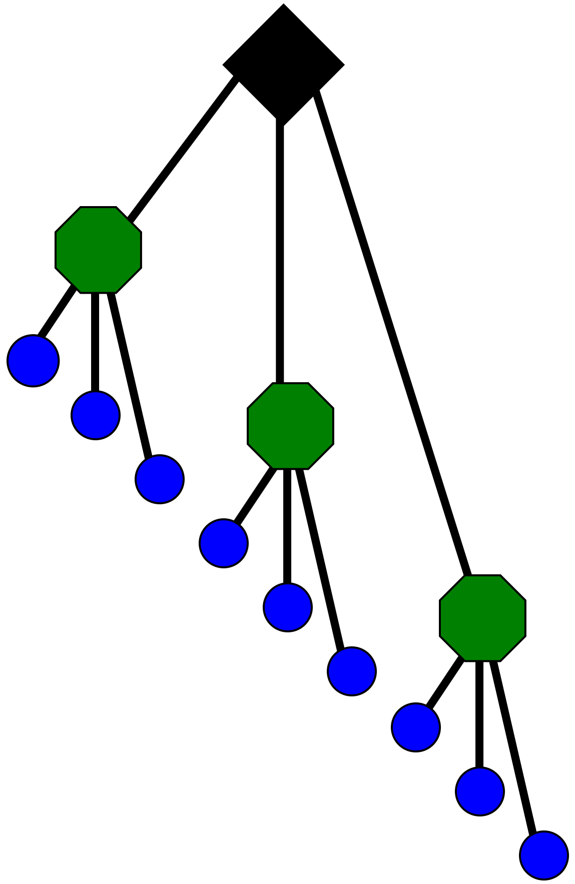












Advantages of Hierarchical K-Means

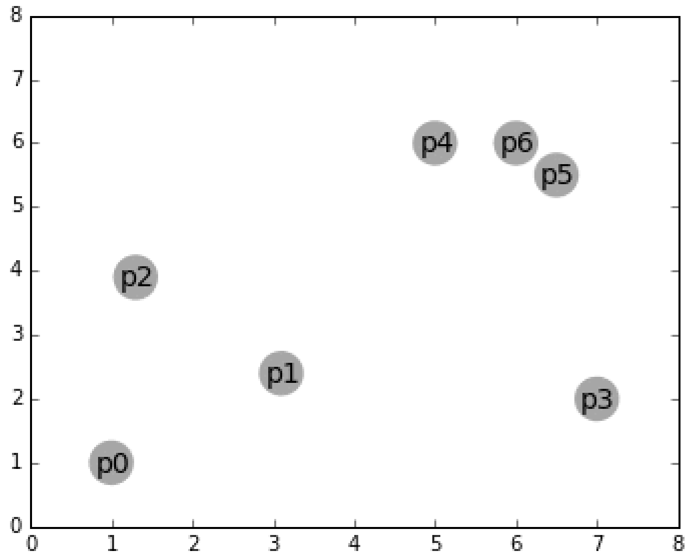
- Fast cluster training
  - With a branching factor of 10, can cluster into 1M clusters by clustering into 10 clusters ~111,111 times, each time using e.g. 10K data points
  - Vs. e.g. clustering 1B data points into 1M clusters
  - Kmeans is O(K*N*D) per iteration so this is a 900,000x speedup!
- Fast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

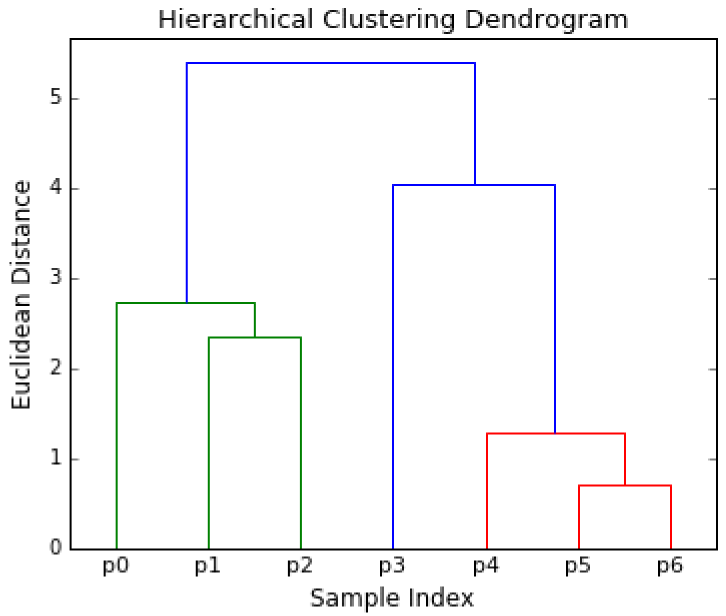
Are there any disadvantages of hierarchical Kmeans?

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

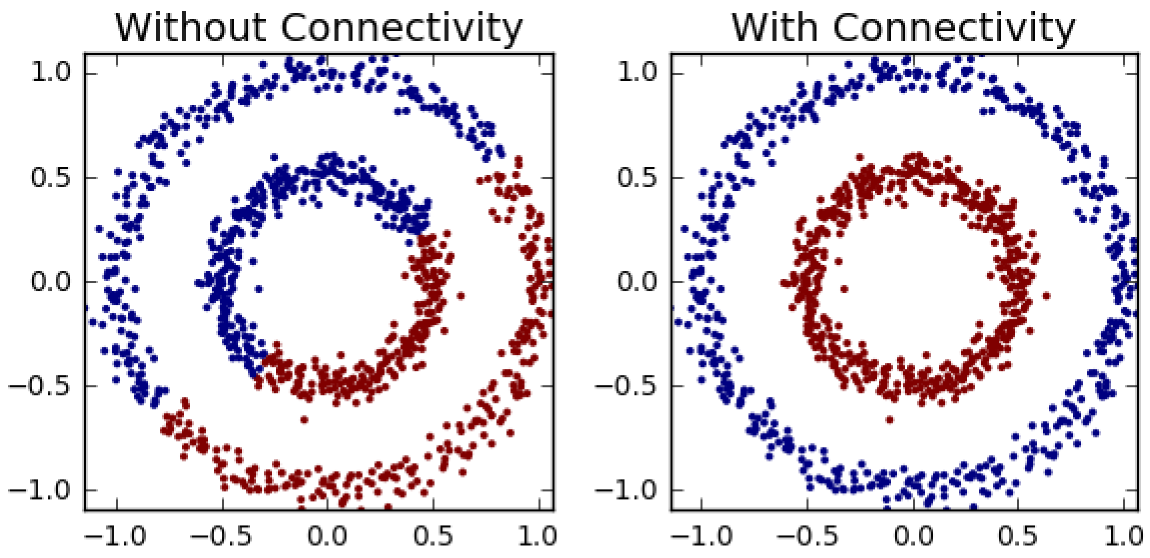




https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/

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)

2 minute break

Are there any times that you’ve used clustering, or that it would be useful?

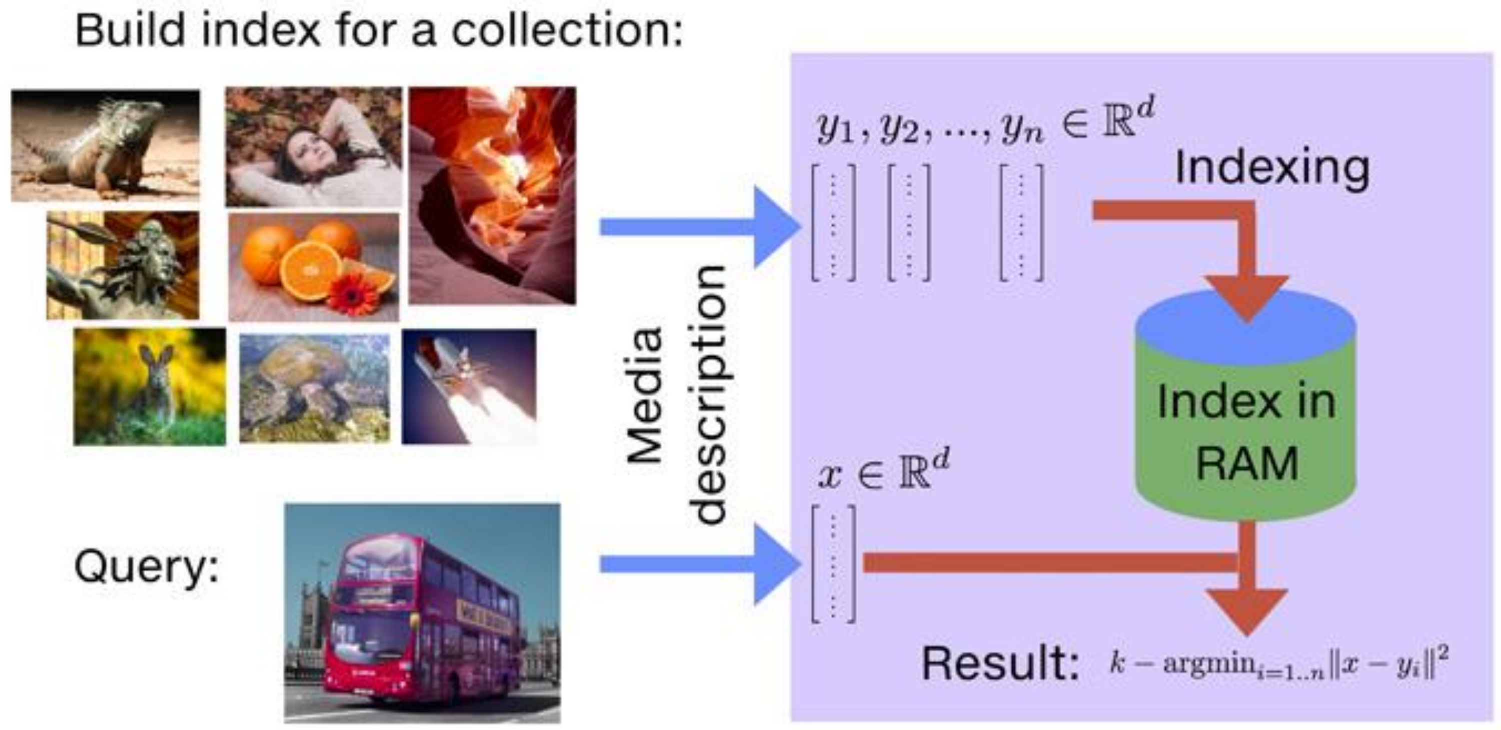
Retrieval

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

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

Vanilla Search

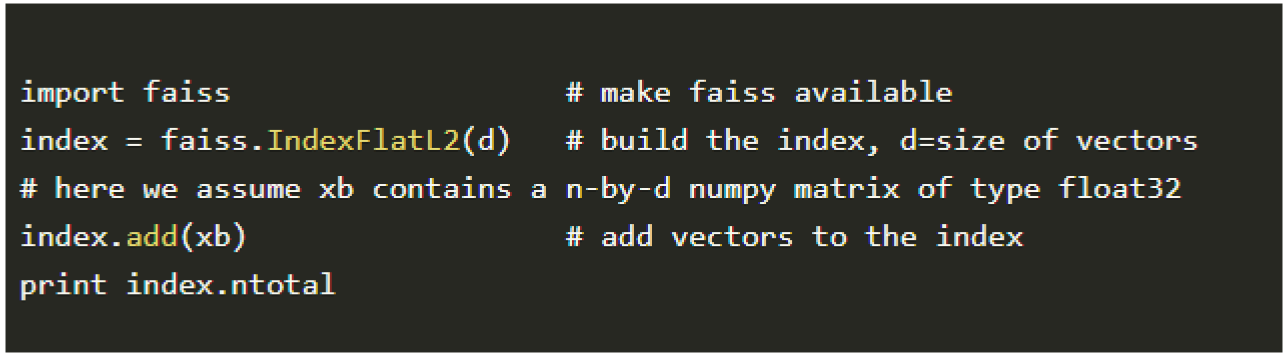
- Compute distance between query and each dataset point and return closest point


Build index for a collection:
Query:
Media description
$y_1, y_2, \ldots, y_n \in \mathbb{R}^d$
Indexing
Index in RAM
$x \in \mathbb{R}^d$
Result: $k-\mathrm{argmin}_{i=1..n} \|x - y_i\|^2$

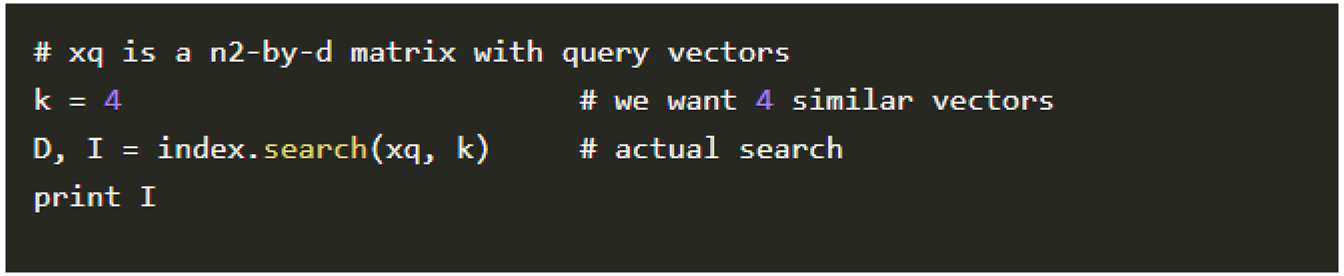
https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

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/

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

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

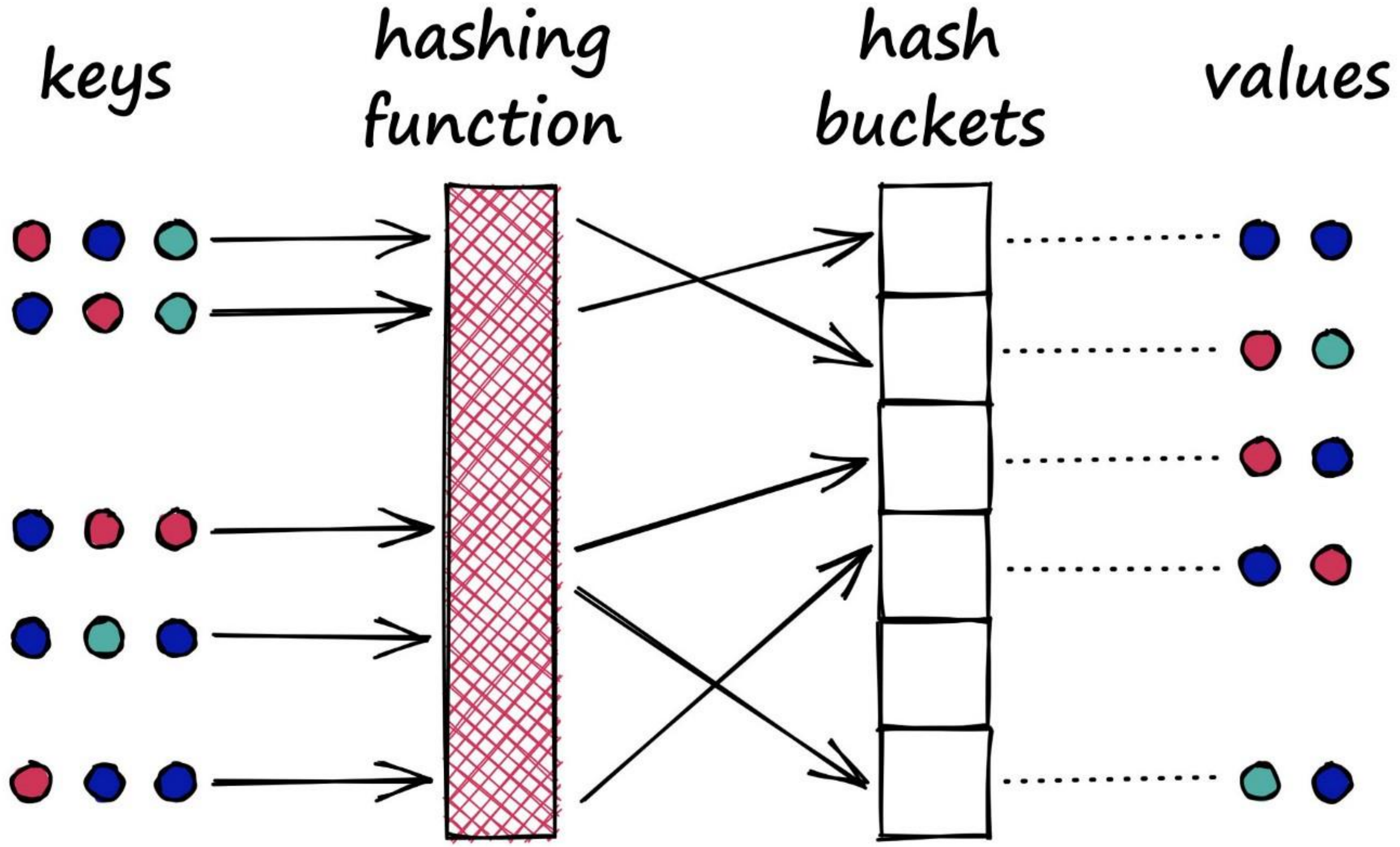
# documents that contain the word

# words in document

Locality Sensitive Hashing (LSH)

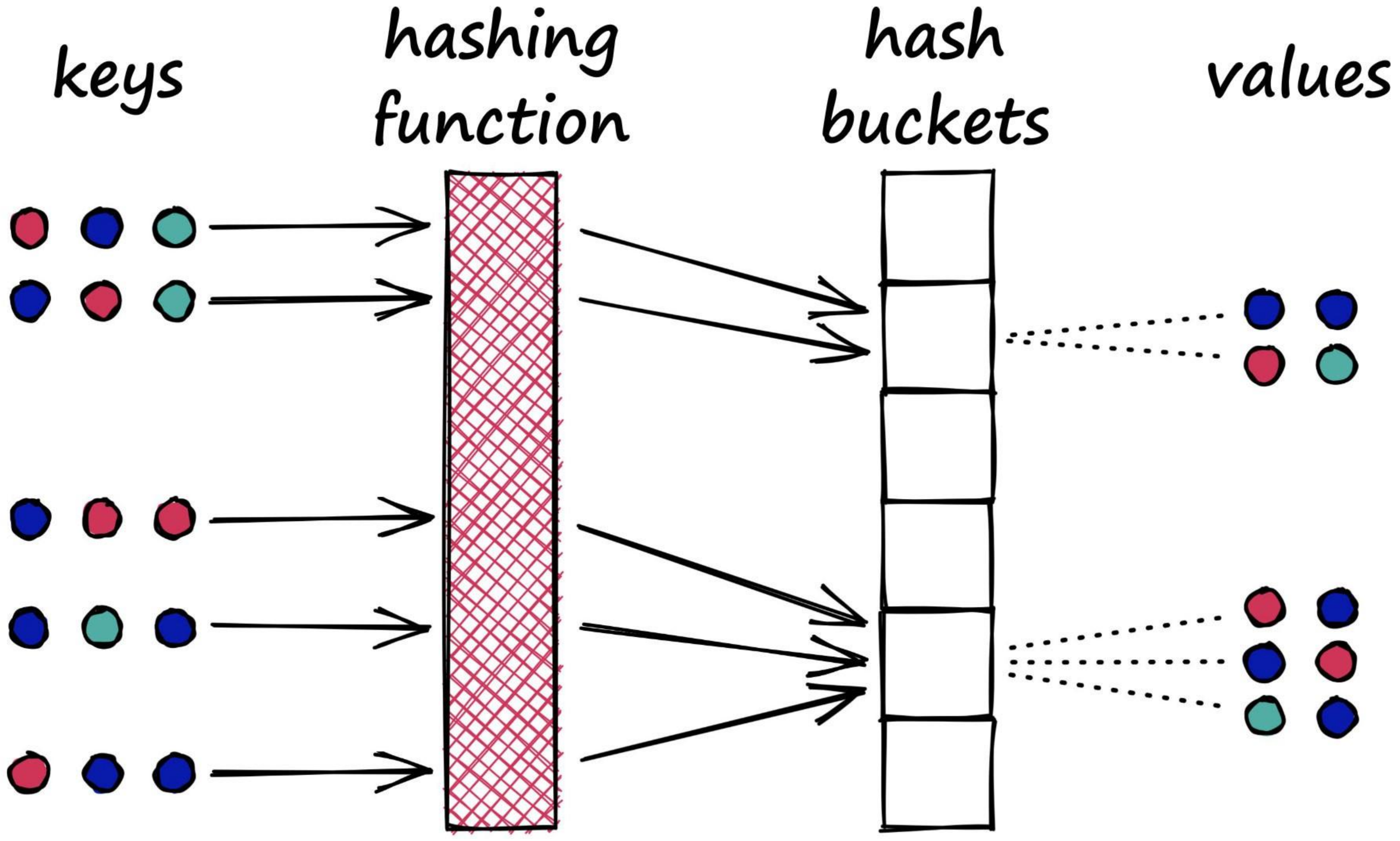
A fast approximate search method to return similar data points to query

A typical hash function aims to place different values in separate buckets



https://www.pinecone.io/learn/locality-sensitive-hashing-random-projection/

LSH aims to put similar keys in the *same* bucket



https://www.pinecone.io/learn/locality-sensitive-hashing-random-projection/

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: $X_n h^T > 0$

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

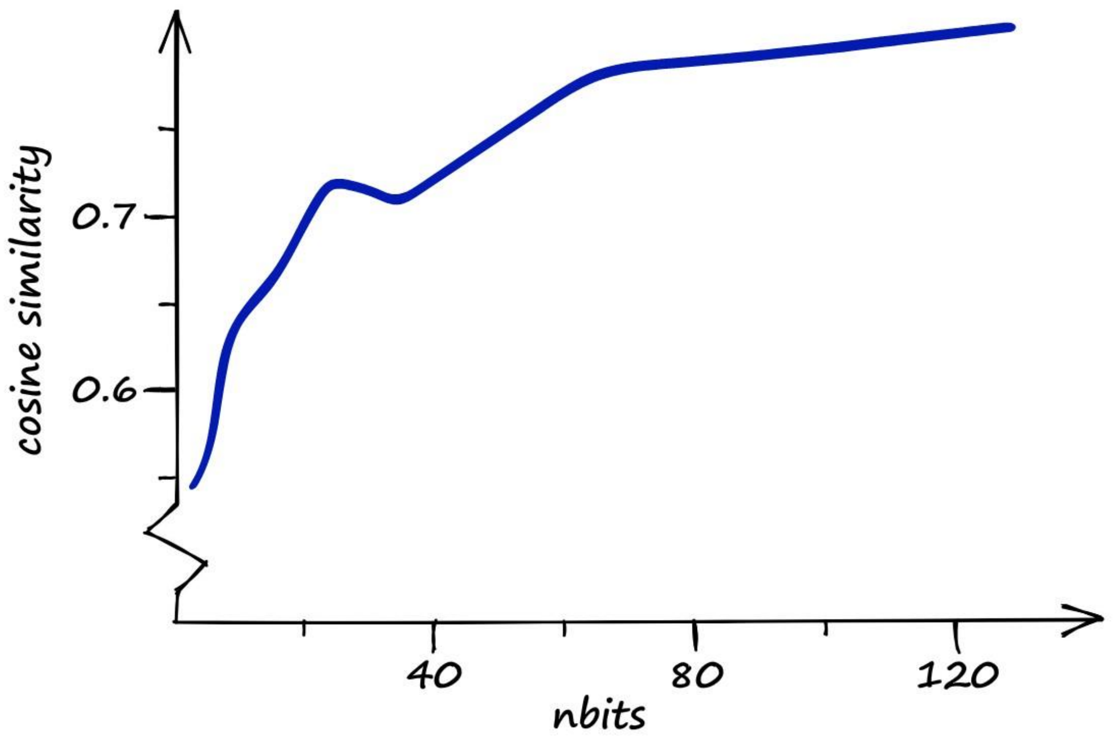
Query

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

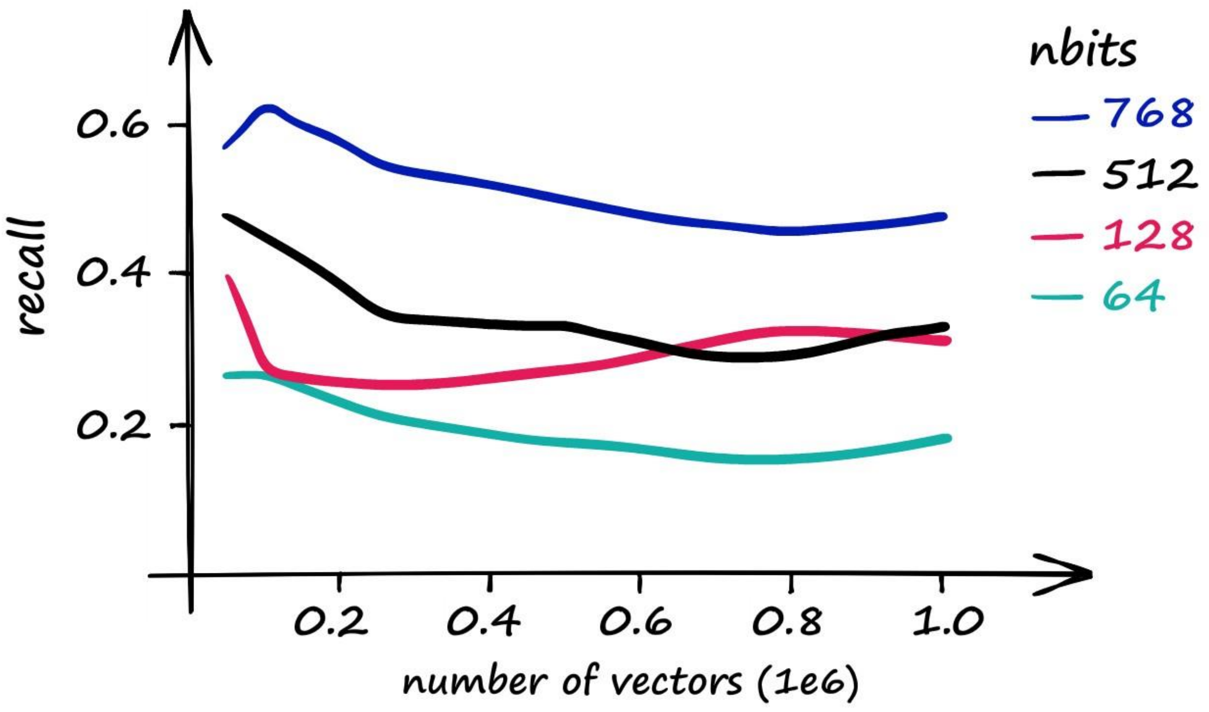
Key parameter: nbits

- Example with 1M 128-bit SIFT vectors

More bits returns more similar vectors



Similarity of LSH-returned vector

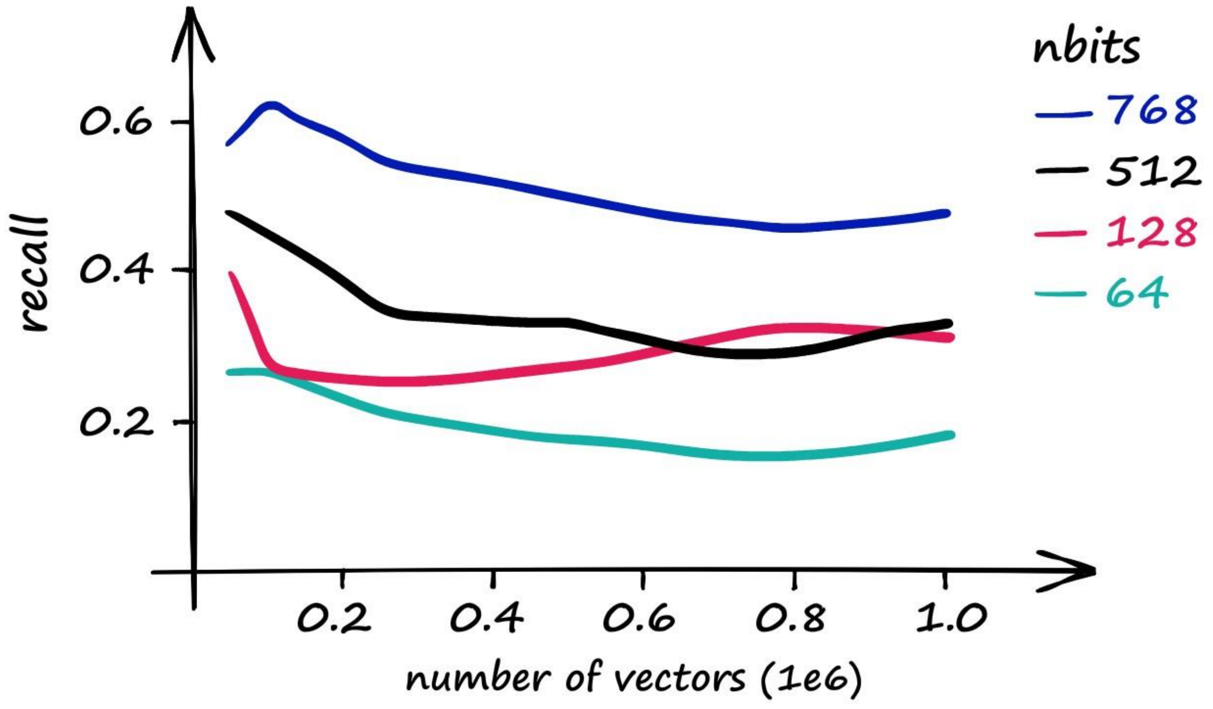


Recall vs. exact nearest neighbor

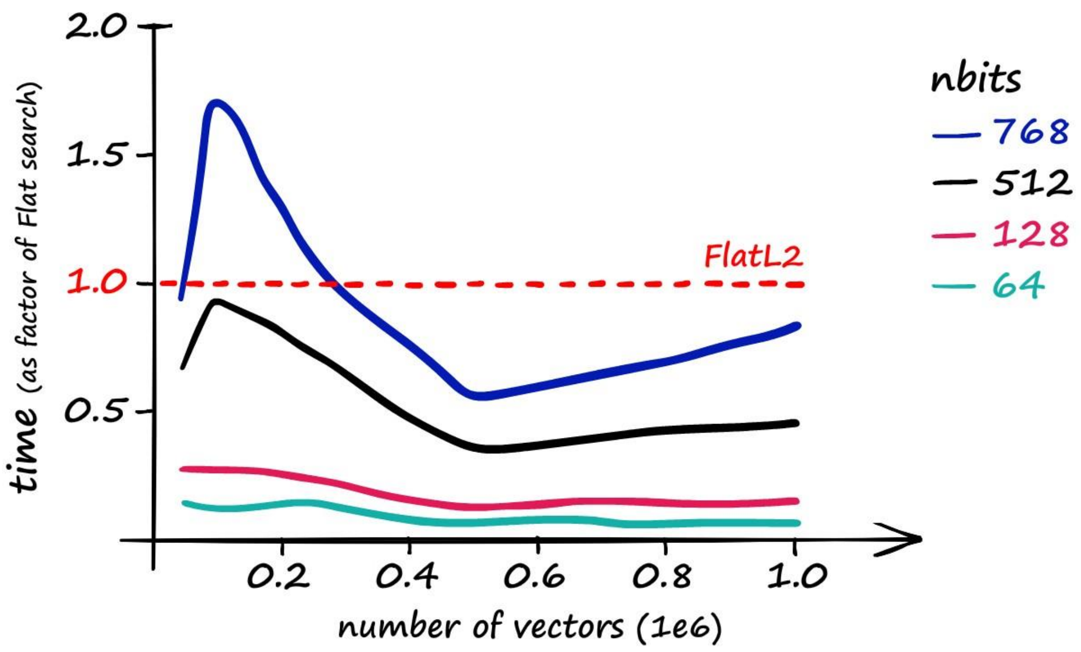
Key parameter: nbits

- Example with 1M 128-bit SIFT vectors

But more bits takes more time to query because it needs to search more buckets to find a collision



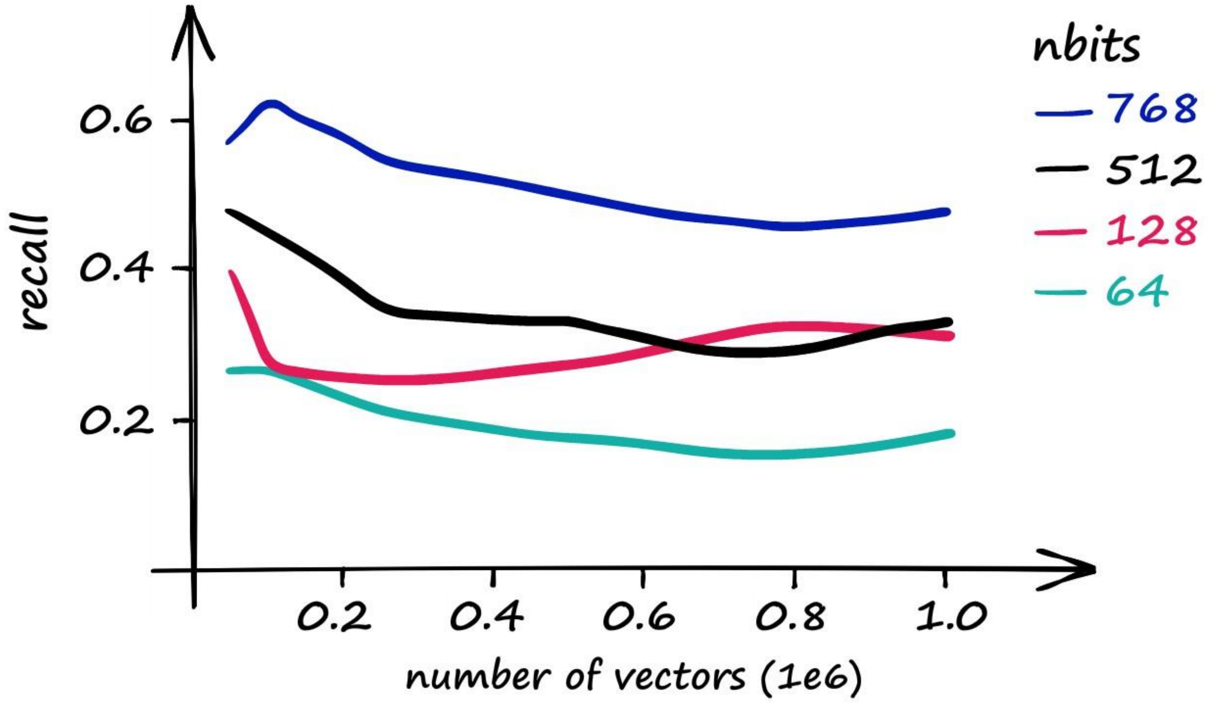
Recall vs. exact nearest neighbor



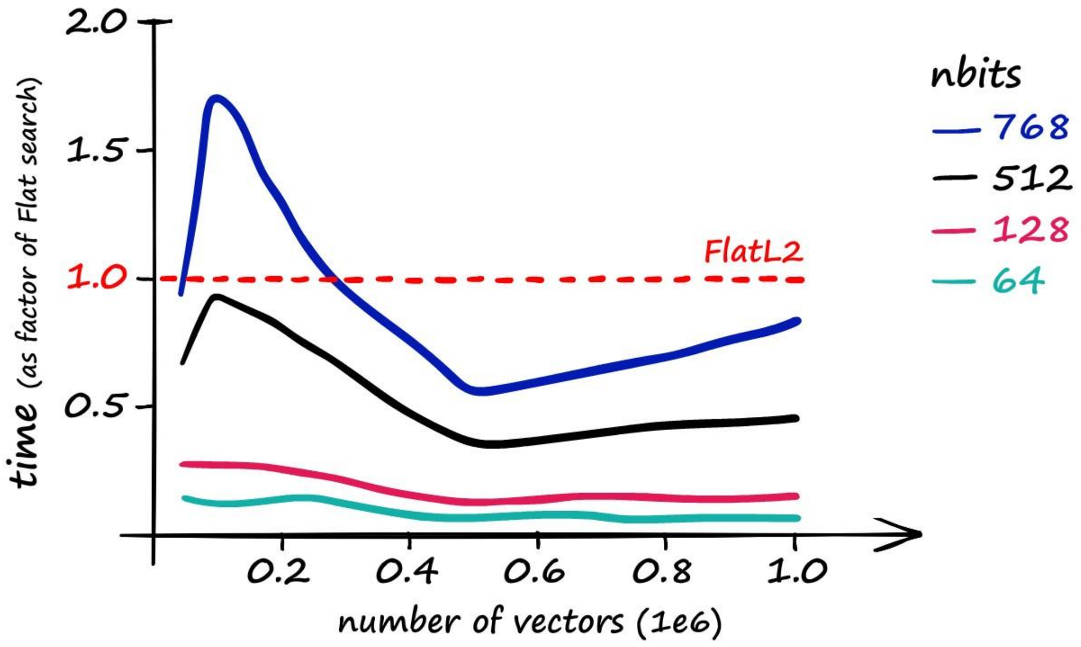
Time compared to brute force search

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/

1. Clustering and Fast Retrieval [40 points]

Efficient querying of high-dimensional data is important for many applications, including retrieval, clustering, and K nearest-neighbor classification. In this part, we will use the Faiss (Facebook AI Similarity Search) library to explore kmeans and KNN on the MNIST data.

K-means: Use faiss to "train" Kmeans (K=10) on `x_train`. Then, get the cluster indices `cluster_idx` for the samples in `x_test`. Display the average sample distance to its centroid, purity, and centroids with the following code.

```
print(dist.mean())

purity, counts = get_purity(y_test, cluster_idx)

idx = get_cluster_order(counts)

display_mnist(kmeans.centroids[idx, :],1,K)
```

Repeat this for K=10, 20, 30, 40, 50, ... 100 and display the cluster center visualization and record mean distance and purity for each. You can experiment with `niter` and `nredo` (see documentation). Include line plots of K vs. purity and K vs. mean distance in your report. Also include the displays of centroids for K=10,20,30.

Also, answer a few questions in your report:

1. As you increase K, do you expect the purity to increase? Why or why not?
2. In a given run, is the average distance of a sample to centroid guaranteed to monotonically decrease (or not change) with each iteration? Why or why not?
3. If you do enough iterations, is Kmeans guaranteed to give you the optimal clustering that minimizes the sum of distances between each sample and its center? Why or why not?
4. Does improving the Kmeans objective necessarily improve expected purity? Why or why not?

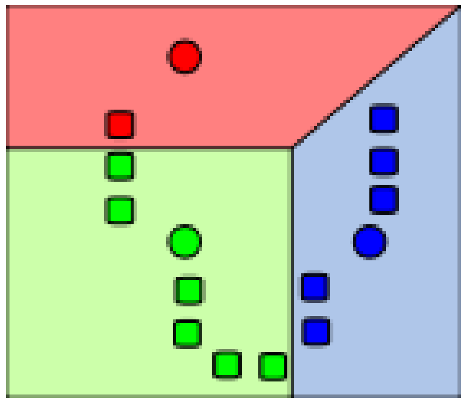
1-NN Revisited: Remember how this took tens of minutes for HW 1? Use the library for more efficient nearest neighbor classification. Don't worry, it will be much faster. You can try two methods, initialized by

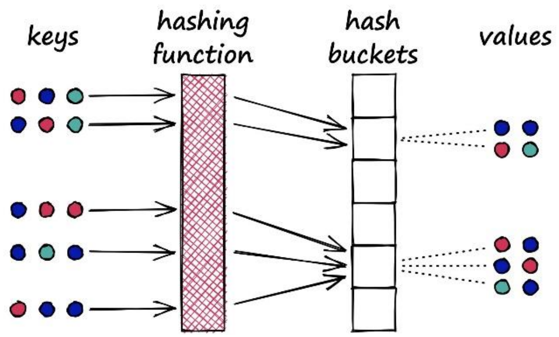
```
index_lsh = faiss.IndexLSH(dim, nbits) # LSH

index_flatl2 = faiss.IndexFlatL2(dim)  # Brute Force
```

In either case, you add the data to search for with `index.add(X_data)` and perform the search with `index.search(X_query,1)`. Try using both methods for 1-NN using `x_train` and `x_test`. Try varying `nbits` to see how it affects time and accuracy (e.g. `ndim/2`, `ndim`, `ndim*2`). If your accuracy is roughly 10% (chance performance), check the shapes of `y_test` and the returned indices to make sure they are the same. For each method, report test error and the time to add and to search, using `time.time()`. See documentation of the different methods available, and feel free to try out more.

Things to remember

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




keys
hashing function
hash buckets
values