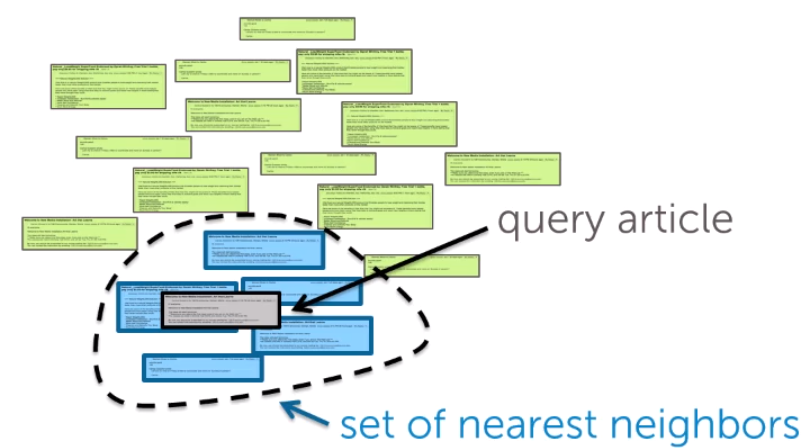
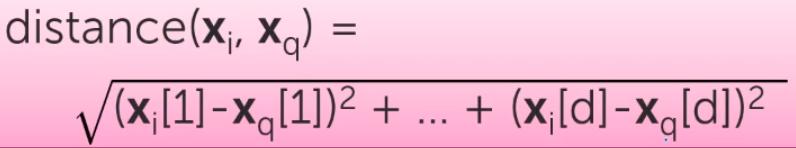
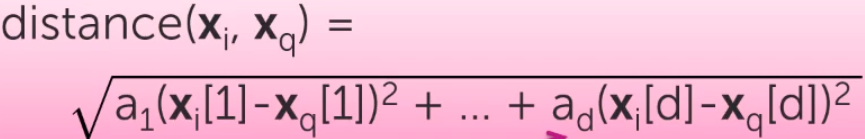
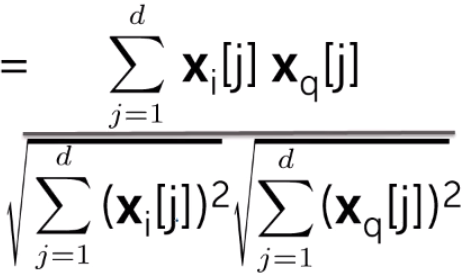
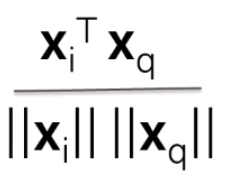
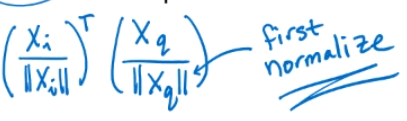
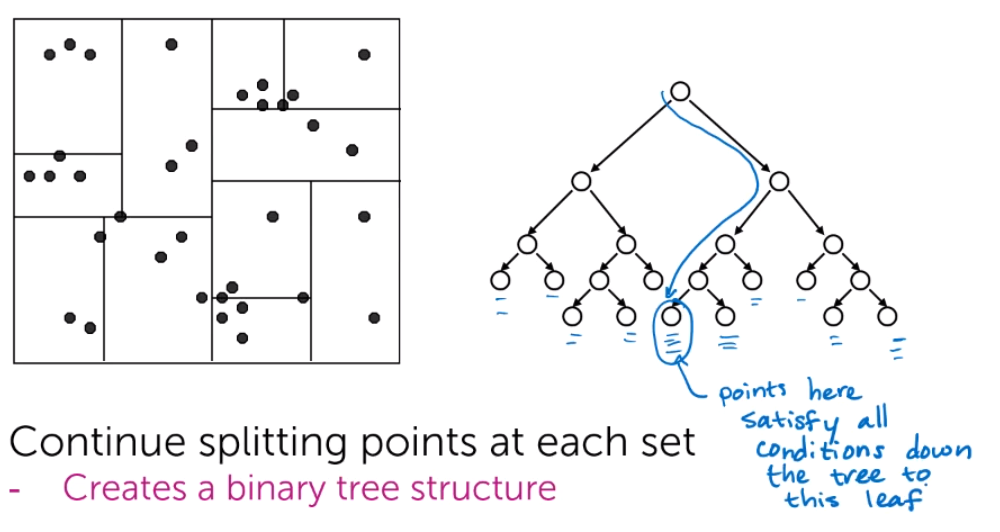
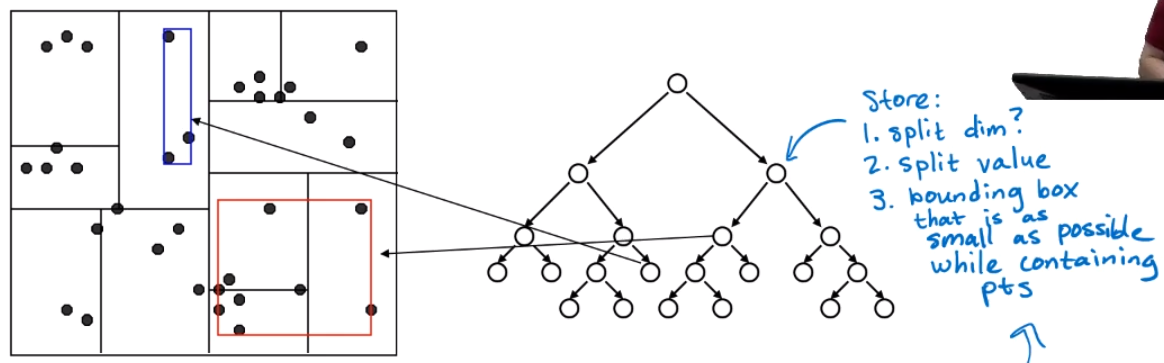
# Intro

* Retrieval – get similar products/documents to the one chosen.
* 
* Clustering – group products/documents in groups.
  + Often it could be an unsupervised learning in contrast to regression and clustering

# Nearest Neighbour Search (k-Nearest Neighbours (k-NN algrorithm))

* We define similarity across articales. We compute distances between the article of interest and other articles. We pick the one (ones) that have the smallest distance. In k-NN, we pick k documents with lowest distances
* Document representation
  + Bag of words model – count all word. Problem is that focus will be on common words like ‘the’, ‘in’ and rare words like ‘volcano’ would be lost
  + TF-IDF – term frequency – inverse document frequency. Local frequency (word count in a document) / global frequency (word count in all documents). Thus, unique words will become more important.
* Distances:
  + Euclidean distance
    - Both magnitude and direction are taken into account
  + Weigth features differently -> e.g. abstract and title are more important than body
  + Scaled euclidean distance 
  + Cosine similarity = cos(theta) =  =  = 
    - Vectors are normalized – their length is made to 1
    - Cosine similarity looks only on direction (co-direction) of vectors, but ignores their magnitude (ignores length of documents)
    - Cosine similarity can make 1 tweet similar to 1 long article. Thus, in practice it is common to compare items (documents) of similar length and not normalizing them
    - In general cosine similarity = [-1,0,1] where 1 is maximum similarity, where -1 means opposite directions and 0 means orthogonality. For only positive features (like tf-idf) cosine similarity = (0,1)
  + Different metrics should be used for different purposes. For the contents of a short text, can use cosine similarity. But for numberical features use euclidean distance
* k-NN complexity:
  + brute force is very computationally expensive, especially if we often send querries
* KD-trees:
  + KD-trees allow splitting data in groups that would make search much faster
  + KD-trees work fine when we have low or medium dimensionality (#of features < #of observations)
  + 
  + 
  + Which dimension to split on?
    - The widest
  + What value split at?
    - Median or center of the box
  + When do we stop?
    - Fewer thatn m points left OR box hits minimum width