SMAI-Lecture 5

Making KNN Classifier Faster

Team : HARSHIT PANDEY, APURVI, SHOBHIT

Topics covered so far:

- *Introduction to basic concepts in ML(3-axes of ML/When to Learn/etc)*
- *ML Workflow*
- *Data Sample Representation*
- *Basic data transformations*
- *Data visualization*
- *Classification of Supervised Learning*
- *Performance Measures(Accuracy/Precision/Recall/Utility and Cost)*
- *Multi-Class problems*
- *Introduction to KNN*

Topics covered in Lecture 5:

- Why KNN is slow?
- KNN Using Kd Tree
- Locality Sensitive Hashing
- Inverted List Example
- The curse of dimensionality
- Manifold

Why KNN is Slow?

- K Nearest Neighbour (KNN) algorithms calculates the distance for every query observation with every data point in the train dataset and then finds the K closest observations.
- The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.
- It Becomes computationally expensive and takes high test time.

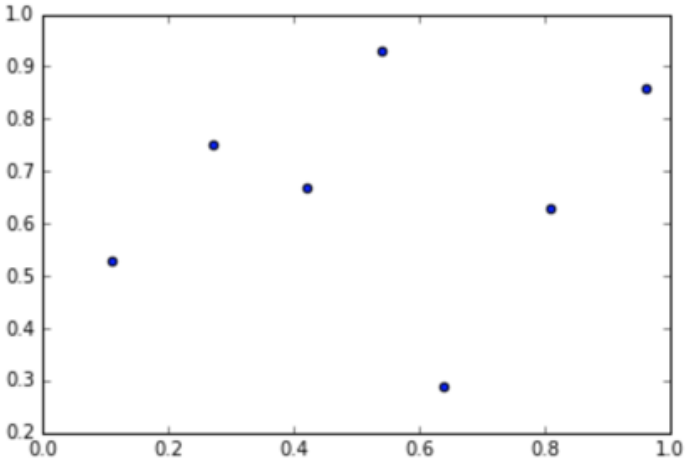
Ways of making KNN Faster

- **Reduce d** (dimensionality reduction) : Reduce the no. of dimensions used in computing the distance.
- **Reduce n** : Reduce the no. of train data from which distance is calculated because to find K nearest neighbour, there might be no need to calculate the distance from all the points.

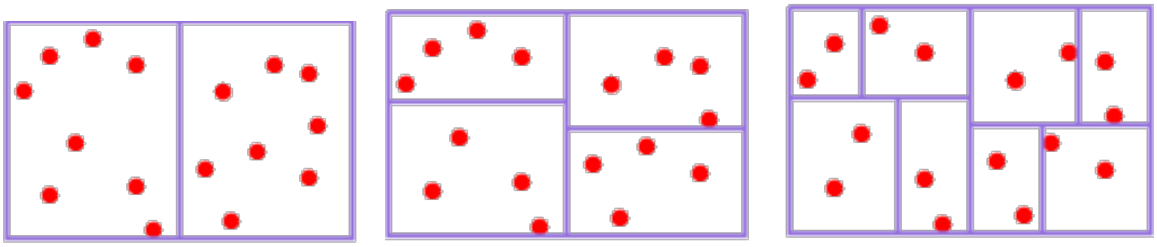
KNN Using Kd Tree (Low Dimension Data)

KD-trees are a specific data structure for efficiently representing our data. In particular, KD-trees helps organize and partition the data points based on specific conditions.

Let's say we have a data set with 2 input features. We can represent our data as-



Now, we're going to be making some axis aligned cuts, and maintaining lists of points that fall into each one of these different bins.



Now the question arises of how to draw these cuts?

- One option is to split at the median value of the observations that are contained in the box.
- You could also split at the centre point of the box, ignoring the spread of data within the box.

Then a question is when do you stop?

There are a couple of choices that we have.

- One is you can stop if there are fewer than a given number of points in the box. Let's say m data points left.
- Or if a minimum width to the box has been achieved.

So again, this second criteria would ignore the actual data in the box whereas the first one uses facts about the data to drive the stocking criterion.

Intuitive Explanation of KD Trees

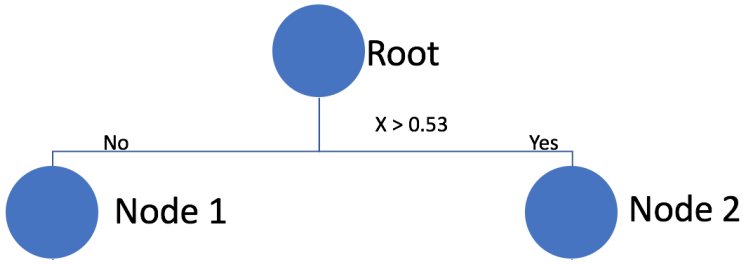
Suppose we have a data set with only two features.

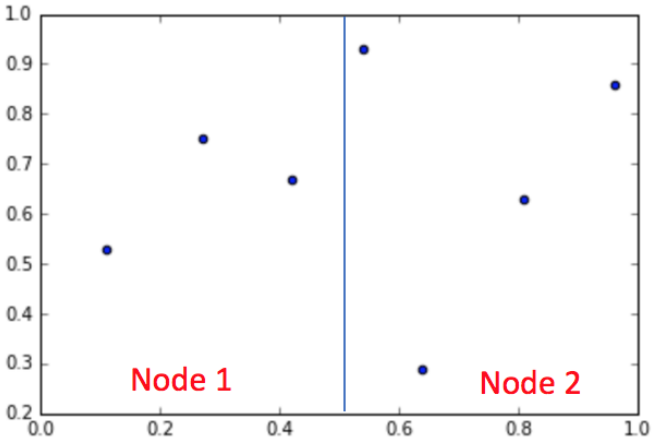
	X	Y
Data point 0	0.54	0.93
Data point 1	0.96	0.86
Data point 2	0.42	0.67
Data point 3	0.11	0.53
Data point 4	0.64	0.29
Data point 5	0.27	0.75
Data point 6	0.81	0.63

Let's split data into two groups.
We do it by comparing x with mean of max and min value.

$Value = (Max + Min)/2$
$= (0.96 + 0.11)/2$

$= 0.53$



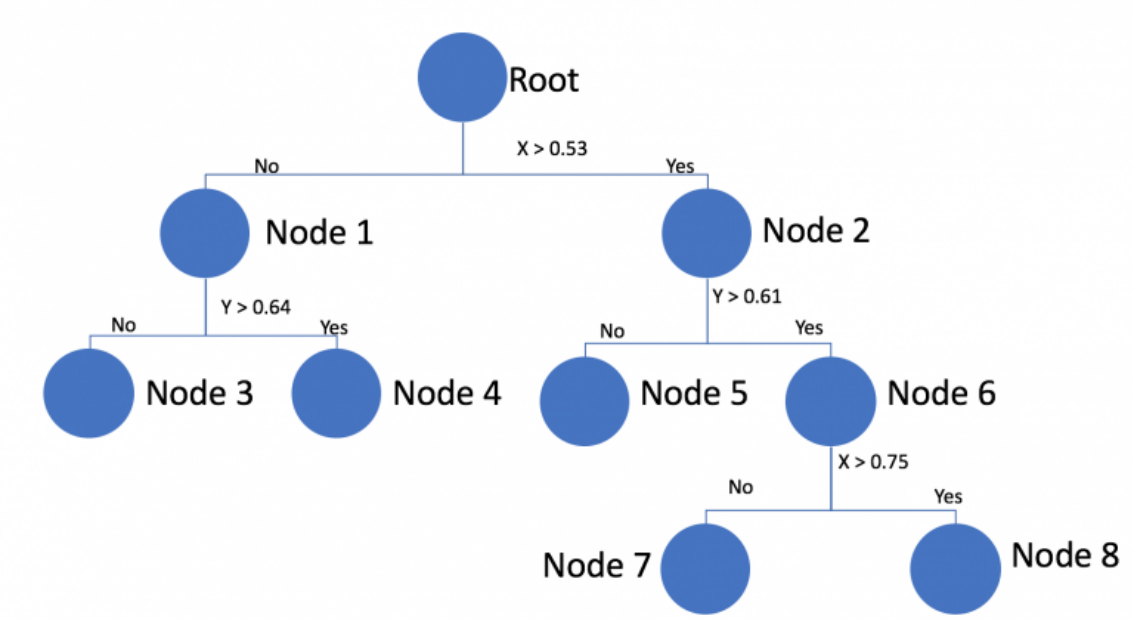


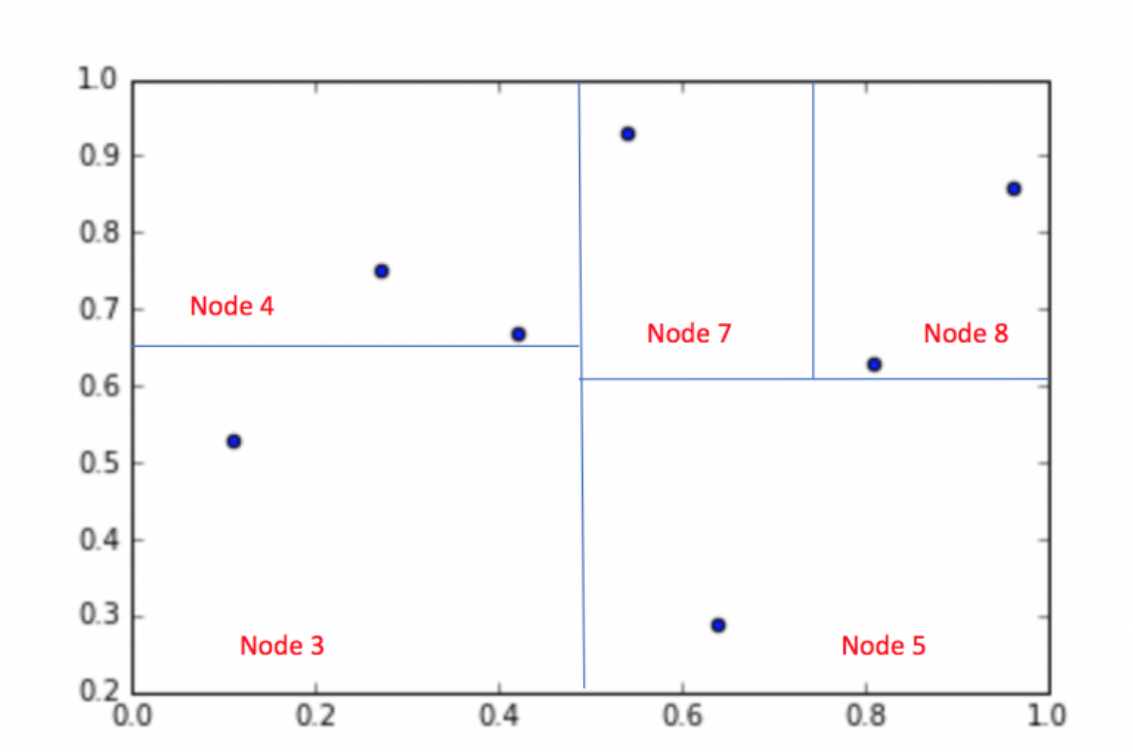
At each node we will save 3 things.

- Dimension we split on
- Value we split on
- Tightest bounding box which contains all the points within that node.

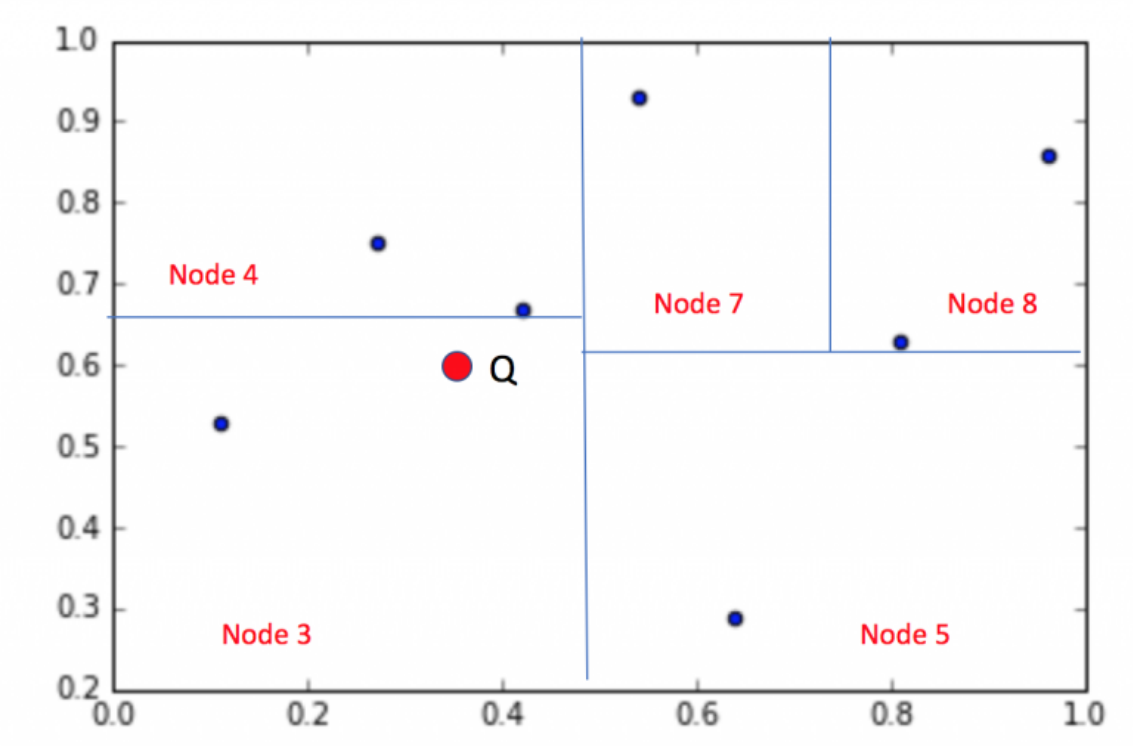
Tight bounds	X	Y
Node 1	0.11 <= X <= 0.42	0.53 <= Y <= 0.75
Node 2	0.54 <= X <= 0.96	0.29 <= Y <= 0.93

Similarly dividing the structure into more parts on the basis of alternate dimensions until we get maximum 2 data points in a Node.

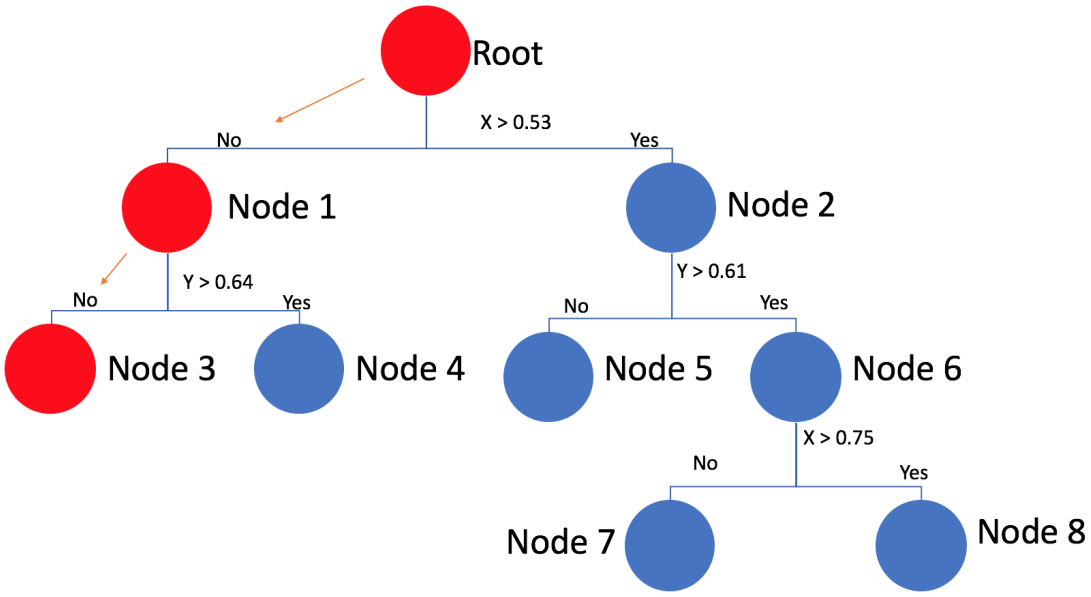




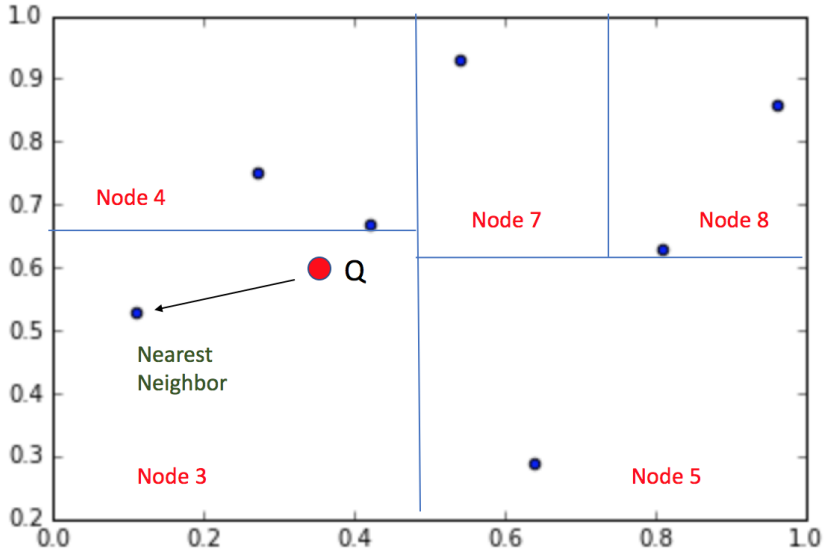
Let's say now we have a query point 'Q' to which we have to find the nearest neighbor.



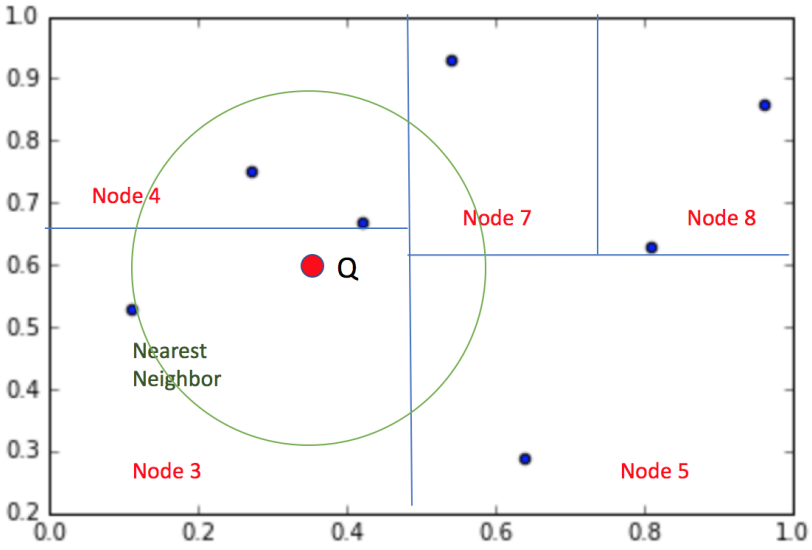
Using the tree we made earlier, we traverse through it to find the correct Node.



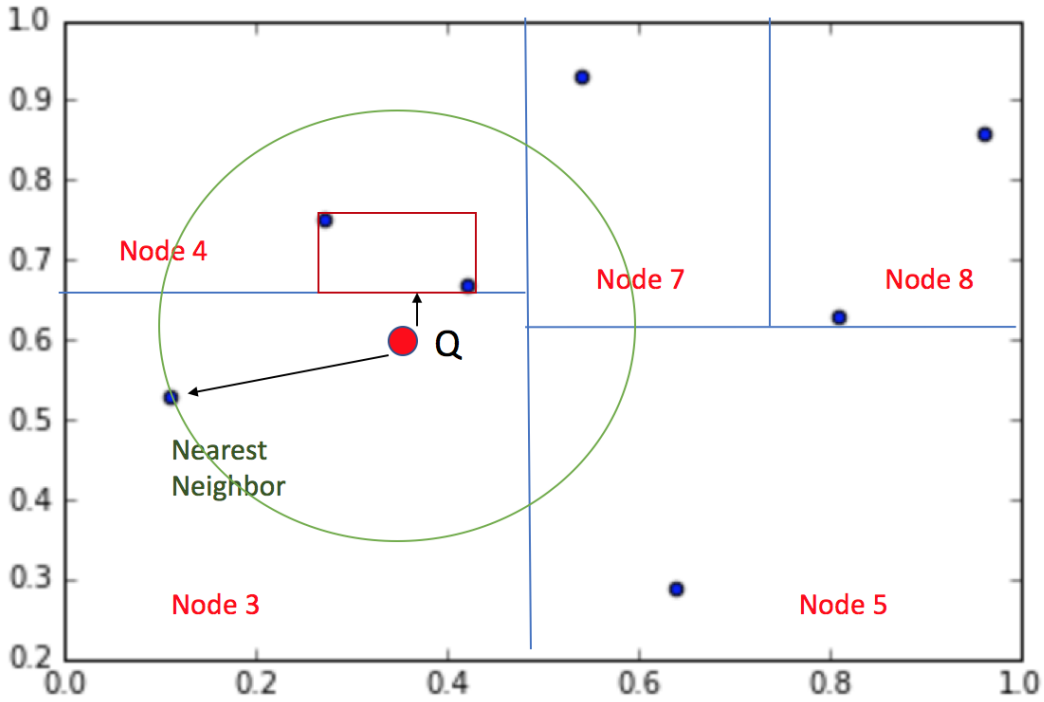
When using Node 3 to find the Nearest Neighbor.



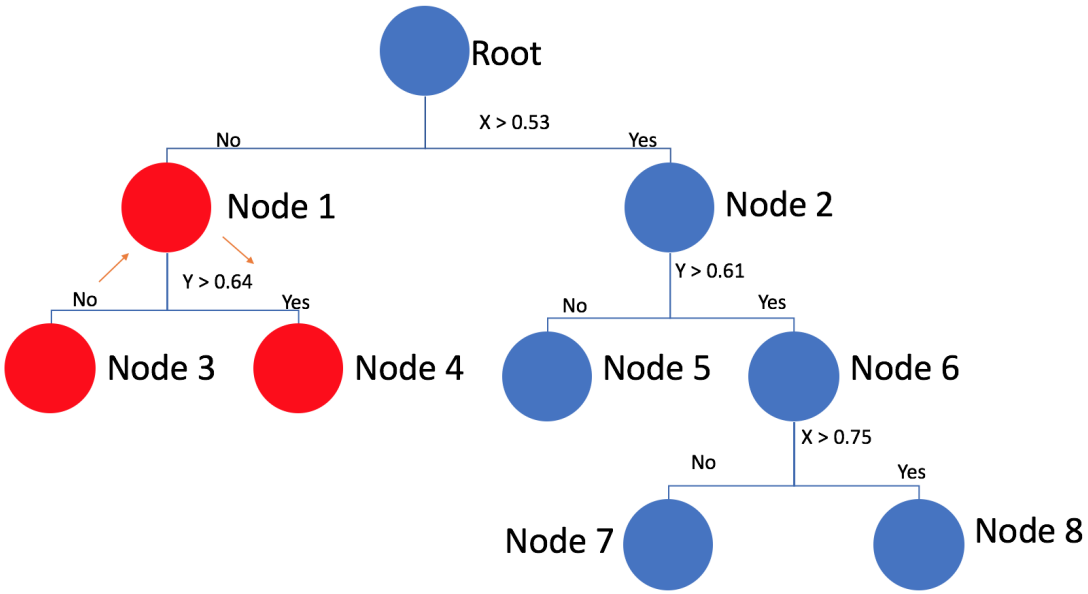
But we can easily see, that it is in fact not the Nearest neighbor to the Query point.

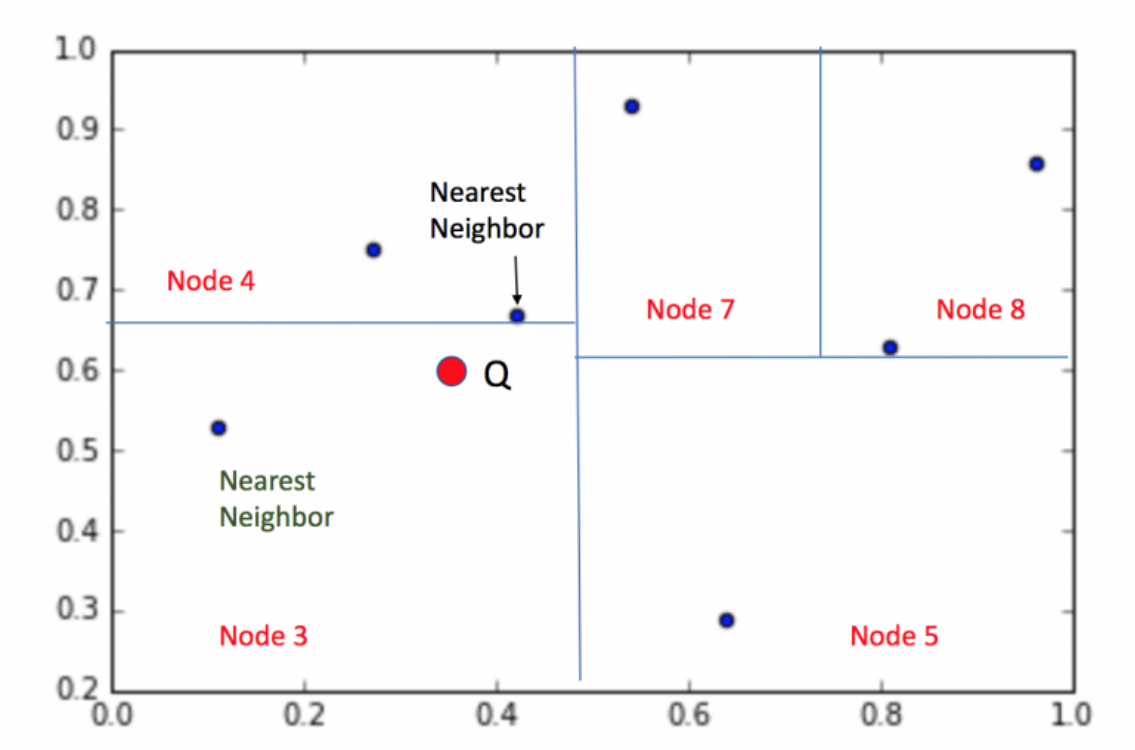


We now traverse one level up, to Node 1. We do this because the nearest neighbor may not necessarily fall into the same node as the query point.
Do we need to inspect all remaining data points in Node 1 ?
We can check this by checking if the tightest box containing all the points of Node 4 is closer than the current near point or not.

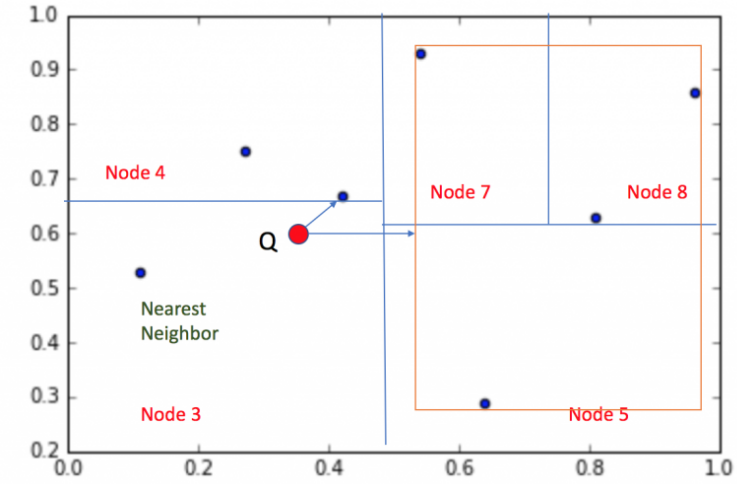


This time, the bounding box for Node 4 lies within the circle, indicating that Node 4 may contain a point that's closer to the query.

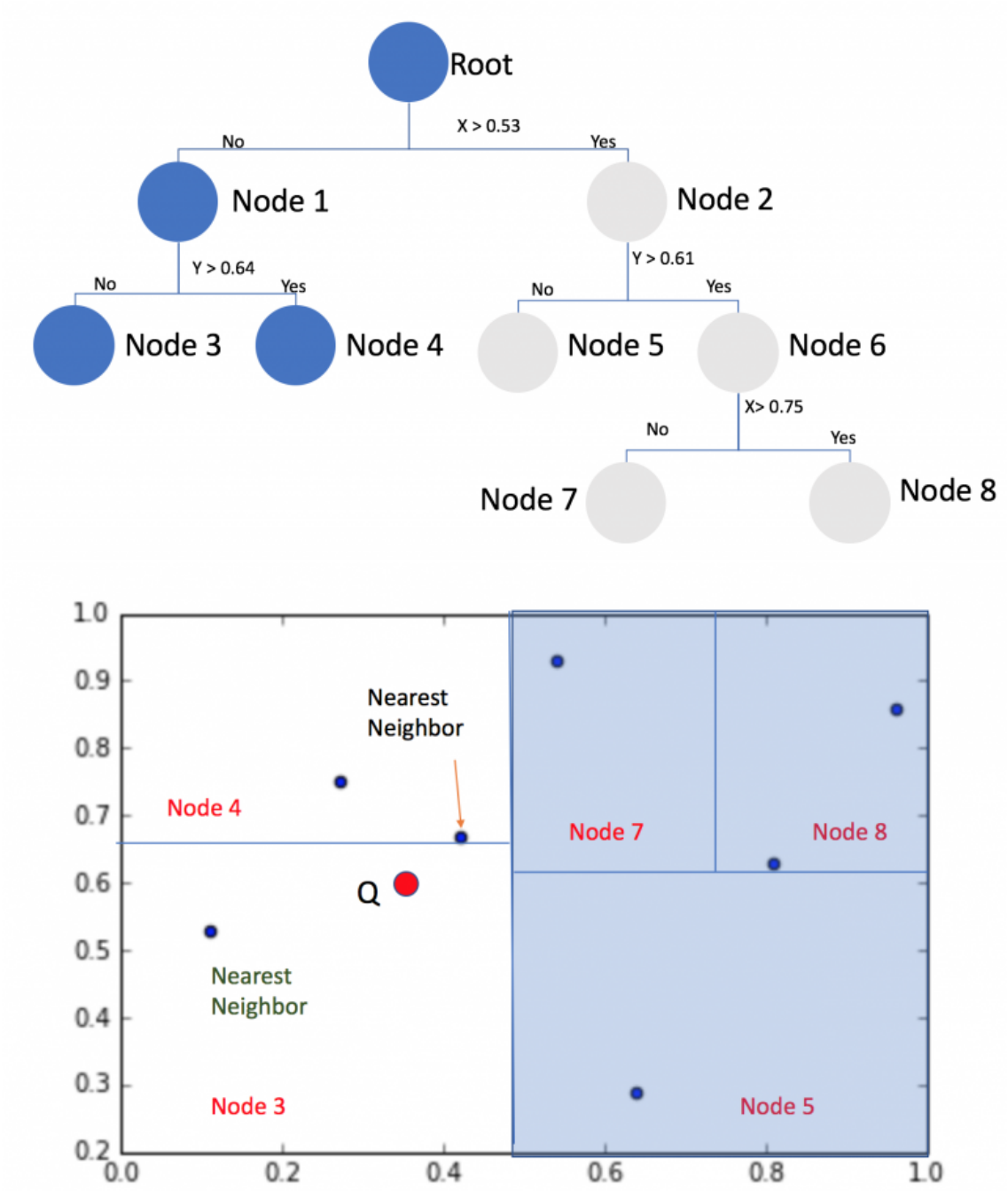




We now traverse one level up, to Root.
Do we need to inspect all remaining data points in Node 2 ?
We can check this by checking if the tightest box containing all the points of Node 2 is closer than the current Near point or not.



We can see that the Tightest box is far from the current Nearest point.
Hence, we can prune that part of the tree.



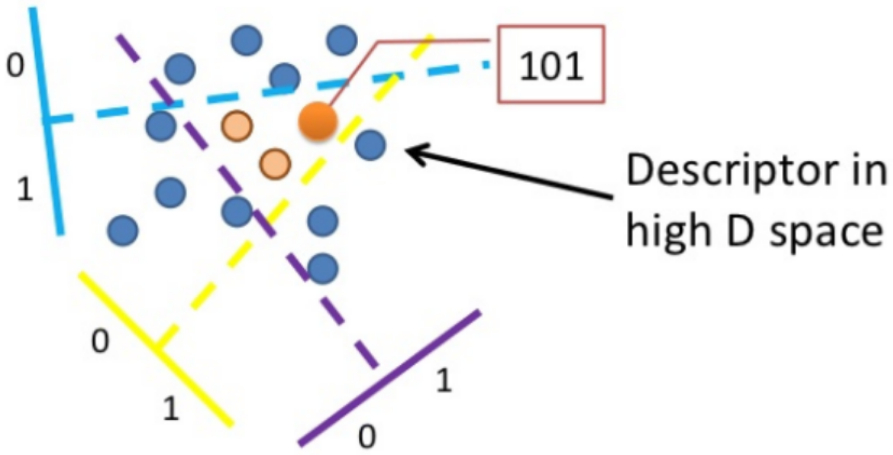
Since we've traversed the whole tree, we are done: data point marked is indeed the true nearest neighbour of the query.

Locality Sensitive Hashing(High Dimension Data)

Locality sensitive hashing (LSH) is an efficient technique for approximate kNN search problem. The core idea of LSH is to hash items in a similarity preserving way, i.e., it tries to store similar items in the same buckets, while keeping dissimilar items in different buckets.

The basic idea is that we generate a hash (or signature) of size k using the following procedure:

- we generate k random hyperplanes
- The i-th coordinate of the hash value for an item x is binary: it is equal to 1 if and only if x is above the i-th hyperplane.



the hash value of the orange dot is 101, because it: 1) above the purple hyperplane; 2) below the blue hyperplane; 3) above the yellow hyperplane

The entire algorithms is just repeating this procedure k times:

Repeat L times:

1. Partition the space using K random hyperplanes parametrized by $w_1, \ldots, w_k$.
2. $[hash(a)]_i = \begin{cases} 1 & w^T a > 0 \\ 0 & w^T a \leq 0 \end{cases}, i = 1 \ldots K$
3. Place a in a hash table using $hash(a)$ as its' key.

an LSH algorithm using random projections with parameters k and L

Random hyper-planes $h_1, h_2, h_3 \ldots\ldots h_k$ slices the space into 2^k regions.

Complexity - O(kd+dn/2^k)

Inverted List Example

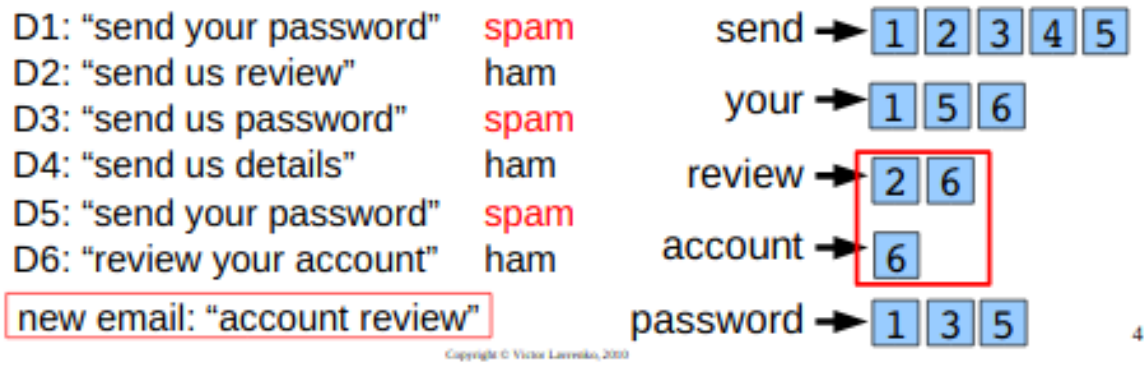
An inverted index of file is an index data structure used for storing content of original file in compressed form, such as digits, to its place in a database file, or in a file or a collection of documents. The main motive of an inverted index structure is to allow fast searching of text with increased processing speed when a document is added to the original database. It is possible that the inverted data may be the database file itself, rather than index of that file. It is the most popular data structure used for document storing and accessing the systems, used on a big scale for example in google.

Data structure used by search engines (Google, etc.)

- list of training instances that contain a particular attribute
- main assumption is most of the attribute values are zero (sparseness).

Given a new testing example :

- fetch and merge inverted lists for attributes present in example
- O($n' \ d'$): d' ... attributes present, n' ... avg. length of inverted list.



The curse of dimensionality

The KNN classifier makes the assumption that similar points share similar labels.
Unfortuately, in High dimensional spaces, points that are drawn from probabilty distribution tends to be never close together.
As the dimension of dimension grows, the amount of data we need to generalise grows exponentially.
For example: If a 1m size coin is dropped in 10m line. The probability is 0.1 to find a coin. If it was 2D and 3D space of same magnitude, The probability drops to 0.01 and 0.001 respectively.

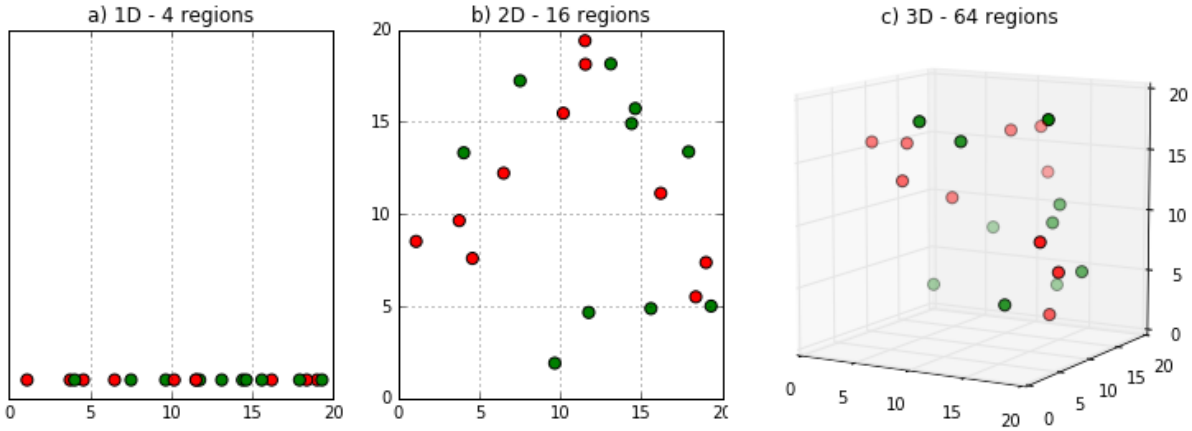
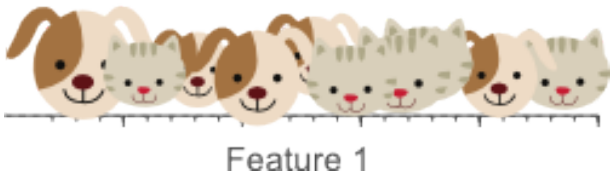
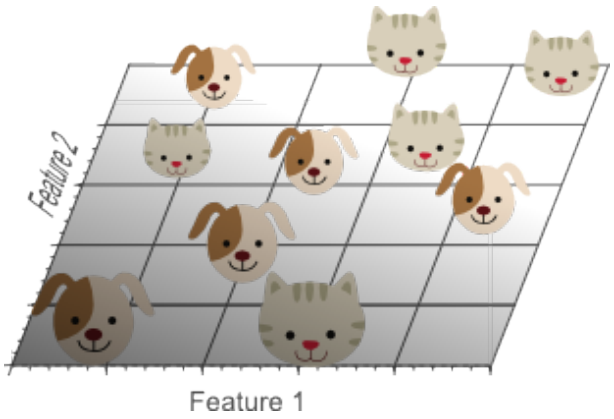


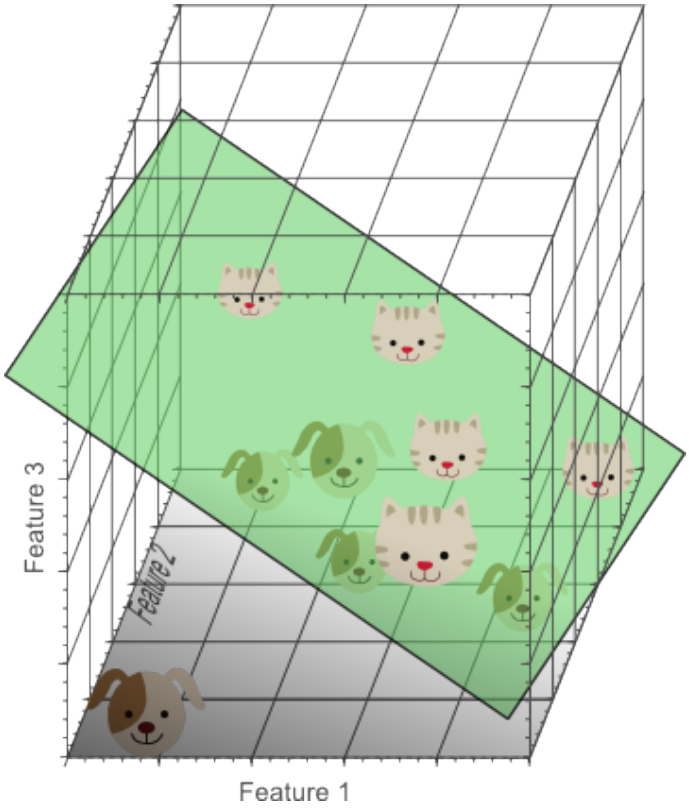
Illustration: Increasing dimensions can result in overfitting.



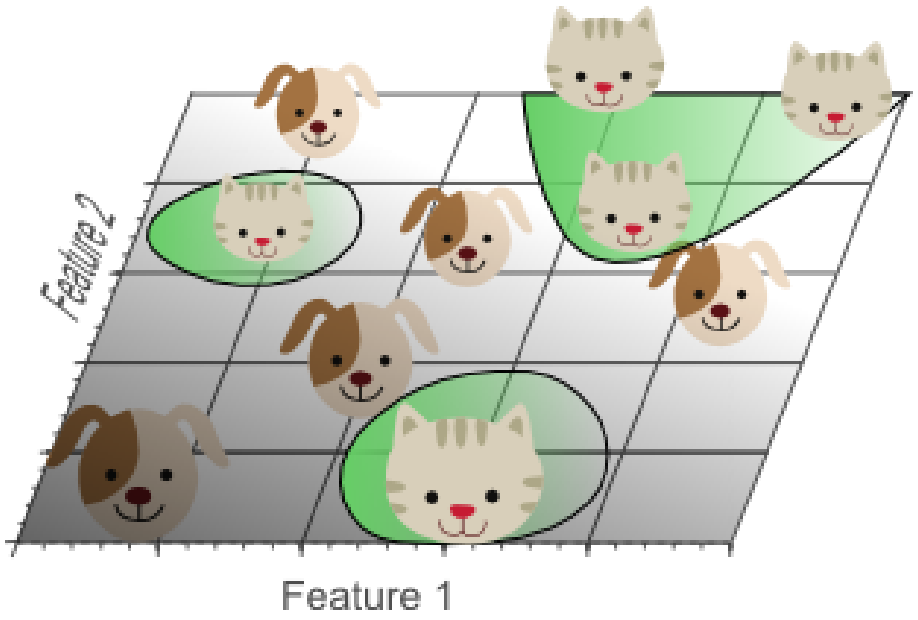
A single feature does not result in a perfect separation of our training data.



A second feature still does not result in a linearly separable classification problem: No single line can separate all cats from all dogs in this example.



The more features we use, the higher the likelihood that we can successfully separate the classes perfectly.



Using too many features results in overfitting. The classifier starts learning exceptions that are specific
to the training data and do not generalize well when new data is encountered.

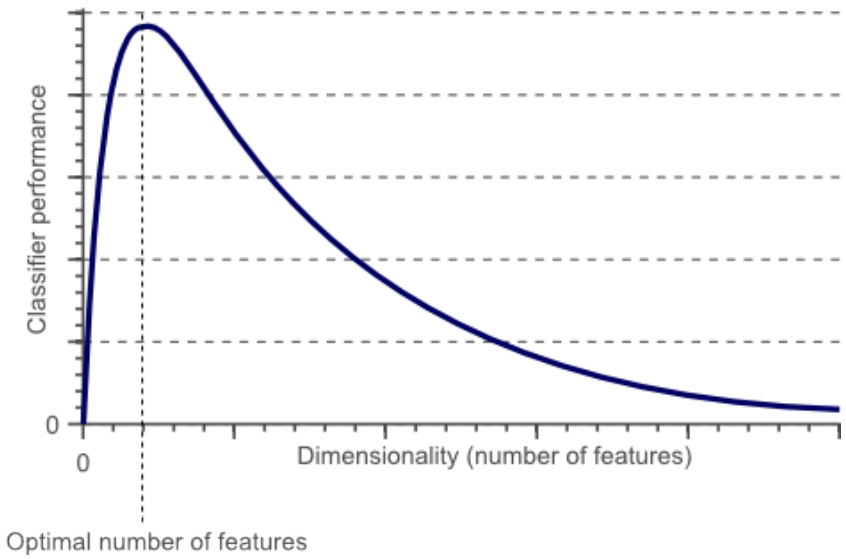


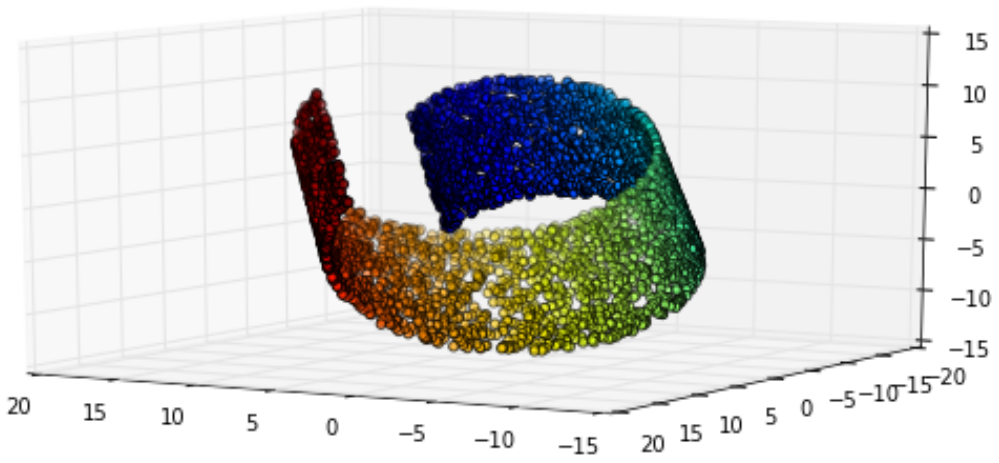
Figure . As the dimensionality increases, the classifier's performance increases until the optimal number of features is reached. Further increasing the dimensionality without increasing the number of training samples results in a decrease in classifier performance.

MANIFOLD

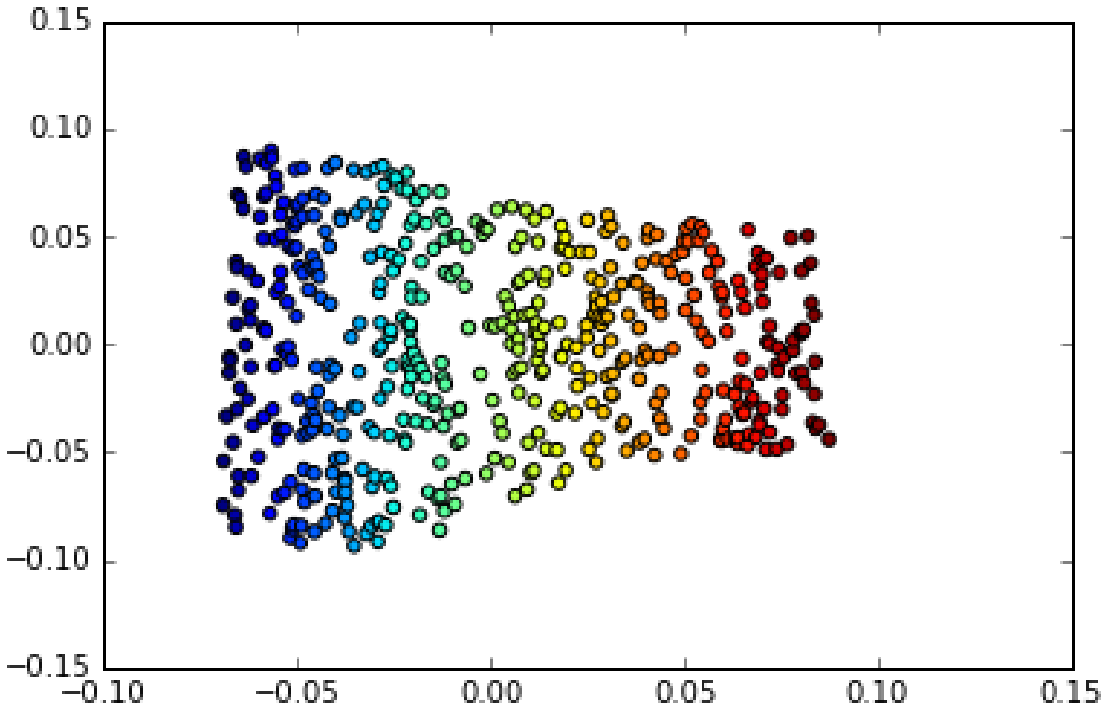
What is a dimension?
To put it simply, if you have a tabular data set with m rows and n columns, then the dimensionality of your data is n:

What is a manifold?
The simplest example is our planet Earth. For us it looks flat, but it really is a sphere. So it's sort of a 2d manifold embedded in the 3d space.

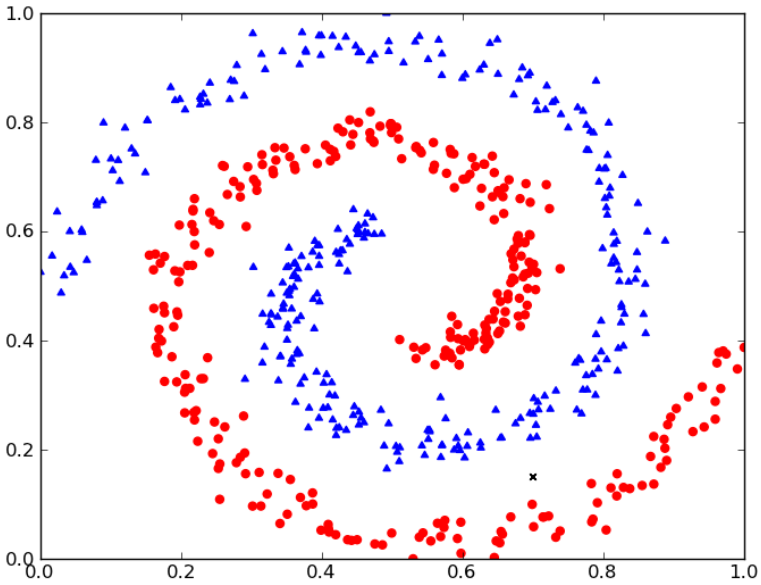
What is the difference?
To answer this question, consider example of a manifold:



This is so-called "swiss roll". The data points are in 3d, but they all lie on 2d manifold, so the dimensionality of the manifold is 2, while the dimensionality of the input space is 3.
There are many techniques to "unwrap" these manifolds. One of them is called Locally Linear Embedding, and this is how it would do that:

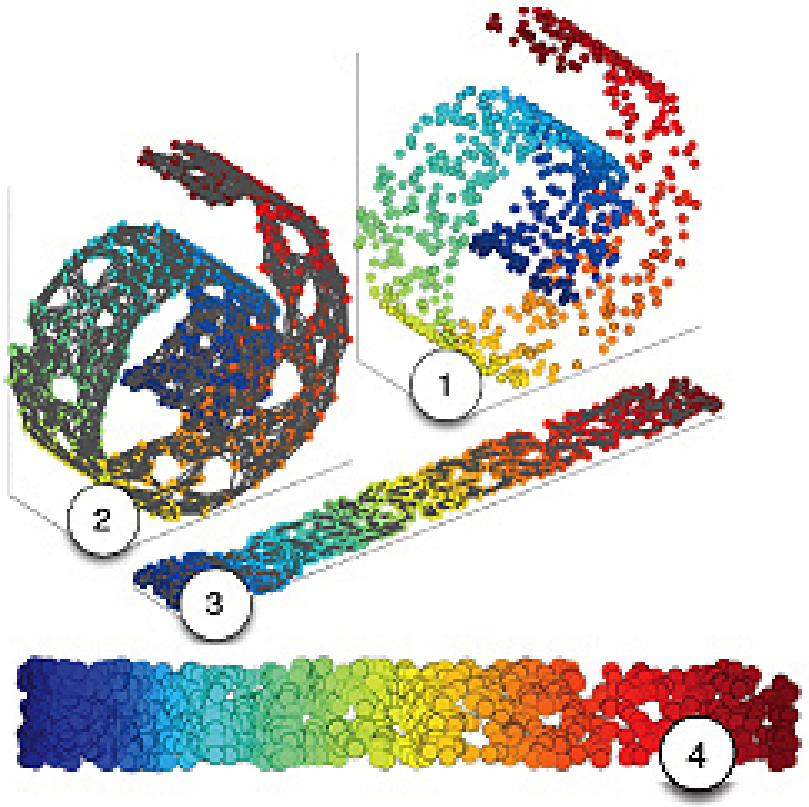


Classification problems are prime examples for manifold learning — where we are specifically looking for manifolds that separate two types of data. For example:



Intuition for thinking about manifolds

The common theme of these examples is that they are somewhat smooth — meaning that there are no sharp spikes or edges. The overall shape of the object can be amorphous, which is nice when describing datasets that don't have rigid boundaries.



References:

- https://en.wikipedia.org/wiki/ManifolD
- https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/
- https://arxiv.org/pdf/1509.06957.pdf