Machine Learning

Lecture 7

Unsupervised Learning with a bit of Supervised Learning

Clustering and nearest neighbour classifiers.

Justin Pearson[1]

mailto:it-1dl034@lists.uu.se

[1] http://user.it.uu.se/~justin/Hugo/courses/machinelearning/

Today's Plan

One supervised and a few unsupervised algorithms.

- Revisions and odds and Ends
- k-nearest neighbours
- Hierarchical Clustering
- k-means clustering
- DBSCAN another clustering method.
- Some general remarks about clustering.

It is important not to get k-nearest neighbours and k-means clustering mixed up.

Supervised Learning

The paradigm so far

- Our data set consists of data

$$(x^{(1)}, y^{(1)}), \ldots, (x^{(i)}, y^{(i)}), \ldots, (x^{(m)}, y^{(m)})$$

- Where $x^{(i)}$ is the data and $y^{(i)}$ is a label. Data is in general multi-dimensional and can consist of many variables.

Supervised Learning

The paradigm so far

- Our data set consists of data

$$(x^{(1)}, y^{(1)}), \ldots, (x^{(i)}, y^{(i)}), \ldots, (x^{(m)}, y^{(m)})$$

- Data can
  - Categorical — Data comes from a discrete set of labels. For example you might have 'Passenger class' , 'first', 'second' or 'third'
  - Everything else, sometimes called continuous, or numeric. Even though you cannot have 0.5 of a person, if you are doing population counts it is easier to treat the data as continuous.

Supervised Learning

Two paradigms

Classification Your labels $y^{(i)}$ are categorical data: "Cat", "Dog", "Hippo".

Regression Your value $y^{(i)}$ is (possibly) multi-dimensional non-categorical data.

Don't forget one-hot encoding for categorical data.

Supervised Learning — Multi class Classification

In general it is hard to train a multi-class classifier. Training a classifier that outputs "Dog", "Hippo", "Cat" from an image is quite hard.
Two strategies:

One-vs-Rest Train individual classifiers "Dog" vs "Not Dog", "Hippo" vs "Not Hippo", "Cat" vs "Not Cat" Pick the classifier with the best confidence.

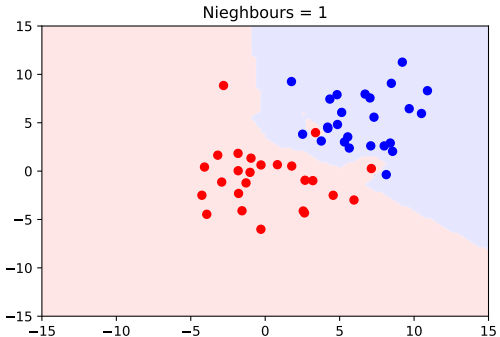
One-vs-One Given n classes train $n(n-1)/2$ binary classifiers: "Dog" vs "Hippo", "Dog" vs "Cat", "Hippo" vs "Cat". Given an unknown image let each classifier run and pick the class that gets the most votes.

k-nearest neighbour classifier

- Very simple classifier.
- Memory based, no model is learned you just have to remember the training data.
- To classify a point, look at the k-closest points look at their classes and take a vote.
- No need to do One-vs-Rest or One-vs-One.

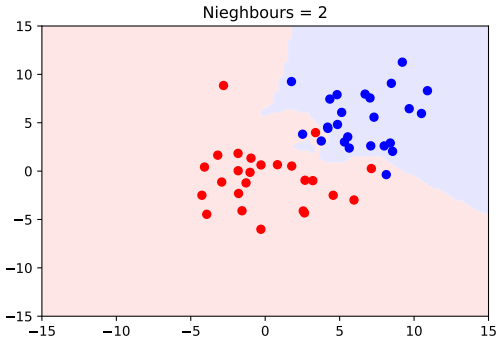
k-nearest neighbour classifier

Example two classes and k=1



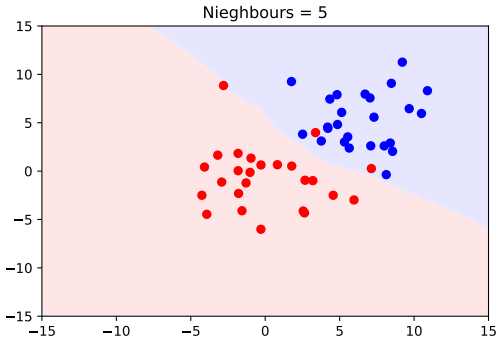
k-nearest neighbour classifier

Example two classes and k=2



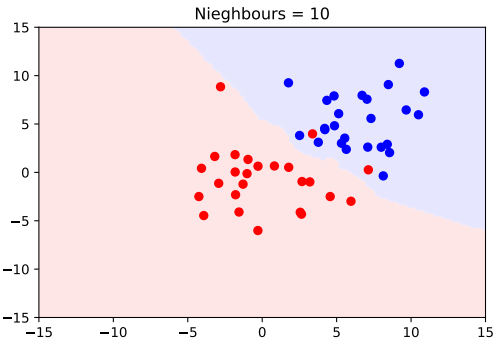
k-nearest neighbour classifier

Example two classes and k=5



k-nearest neighbour classifier

Example two classes and k=10



Why does it seem that points are miss-classified? How can this happen with the algorithm.

Other Metrics

All that we need for k-nearest neighbours is some function $d(x, y)$ that gives you the distance between two points x and y. As long as your function obeys the following axioms:

$$\forall x.d(x, x) = 0$$

$$\forall x, y.d(x, y) = d(y, x)$$

$$\forall x, y, z.d(x, y) + d(y, z) \leq d(x, z)$$

The k-nearest neighbour algorithm will work.

k-nearest neighbour classifier

Picking the metric lets you decide what you mean for two points to be close together.

k-nearest neighbour makes no assumptions on the shape of the data. In a sense you can learn almost anything

It is often a good first algorithm to try.

Other Metrics

For example you could use the edit (Levenshtein) distance metric that tells you the minimum number of insertions, deletions and substitutions to get from one string to another. For example $d(\text{kitten}, \text{sitting}) = 3$ by

$$\text{kitten} \to \textbf{s}\text{itten} \to \text{sitt}\textbf{i}\text{n} \to \text{sittin}\textbf{g}$$

Transformation Invariant Metrics[3]

If you wanted to recognise numbers that can be rotated then you would want the distance between the following images to be zero.

One way would be to build a special metric[2], another idea is to add rotations (and other transformations) to your training data. With k-nearest neighbour you can run into memory problems, but with other learning algorithms this might not be too much of a problem.

[2] `https://ieeexplore.ieee.org/document/576916` Memory-based character recognition using a transformation invariant metric

[3] Image from Wikimedia

Problems with k-NN

- As the size of the data set grows, and the more dimensions of the input data the computational complexity explodes. This is sometimes referred to as the curse of dimensionality.
- With reasonable clever data structures and algorithms you can speed things up.

Even with these problems, k-NN can be a very effective classifier.

The problem of labels

- Good labels are hard to come by. Sometimes you will need a human to classify the data, and there is a limit to how much data you can get.

k-nearest neighbours for regression

You can also use it for regression (predicting a value).

You data is something like a collection of points $(\vec{x_i}, y_i)$, where $\vec{x_i}$ is a vector and y_i is the value.

Given a new point $\vec{x}$, look at the k-nearest neighbours

$$(\vec{x_1}, y_1), \ldots (\vec{x_k}, y_k)$$

Then predict the mean $\frac{1}{k}(y_1 + \cdots + y_k)$. Variations include, predicting the median value.

You can use this to fill in missing values.

Advantages of k-NN

- Easy to implement.
- Adapts easily to new data.
- Few hyper parameters.
- Apart from the metric it does not make any assumptions about the model.

Disadvantages of k-NN

- Does not scale well. You have to store all the data.
- Curse of dimensionality. Does not perform well with high-dimensional data.
- Prone to overfitting.

This slide is intentionally left blank. Although it is not clear if I succeeded.

Unsupervised learning

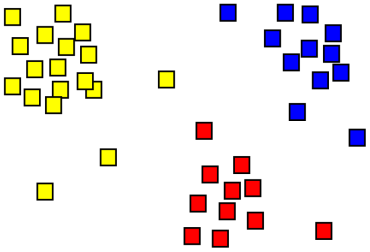
Simply put: how do we learn on data without labels? There are lots of motivations.

- Lack of good labels[4].
- Even with labels, there might be other patterns in the data that are not captured by the labels.
- Historical motivation, a big part of human learning is unsupervised. What can learn about biology and cognition from un-supervised learning algorithms. For example look at Hebbian Learning.
- Recommender systems. How does Amazon tell you which book to buy? Or how does Netflix tell you which film to watch? There are some specialised algorithms, but a lot of it is machine learning.
- Data mining. I have this data, what does it tell me?

[4] `https://www.theregister.co.uk/2020/02/17/self_driving_car_dataset/` Please check your data: A self-driving car dataset failed to label hundreds of pedestrians, thousands of vehicles.

Clustering[5]

General idea. Take the input data and group it into clusters of similar data points.

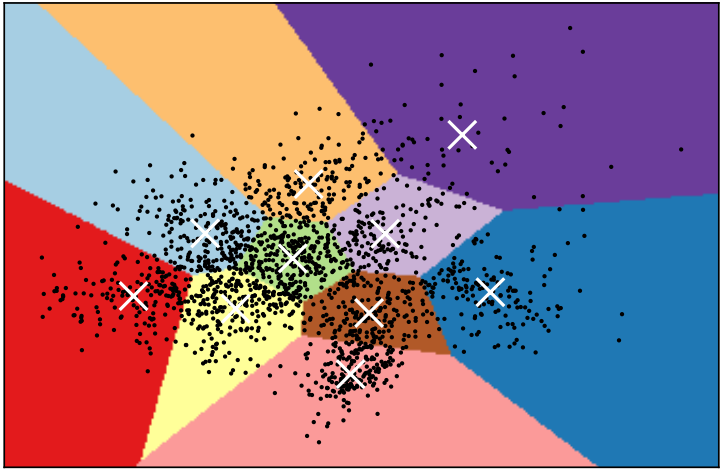


Each cluster becomes a class.

[5] Picture taken from `https://commons.wikimedia.org/wiki/File:Cluster-2.svg`

Clustering

K-means clustering on the digits dataset (PCA-reduced data)
Centroids are marked with white cross



Applications of clustering

- Market segmentation, divide your customers into similar groups.
- Image segmentation
- In Astronomy, clustering similar observations.
- Clustering gene expression data from micro arrays.
- Clustering texts with various word2vec models.

You could even use it as a preprocessing step for a learning algorithm. Find the clusters and use that as an inferred parameter.

Clustering

Questions

- How do we decide how similar items are?
- How do we determine how many cluster there should be?
- What is the computational complexity of clustering?

Clustering Algorithms

Finding the best clustering of a data set is computationally hard, and you do not always know what the best clustering of your data set should be.
We will look at 3 algorithms that take a data set and produce a clustering:

- Hierarchical clustering
- K-Means
- Density-based spatial clustering of applications with noise (DBSCAN)

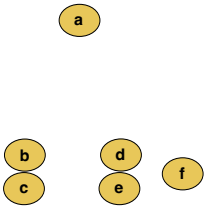
Hierarchical Clustering

Basic idea:

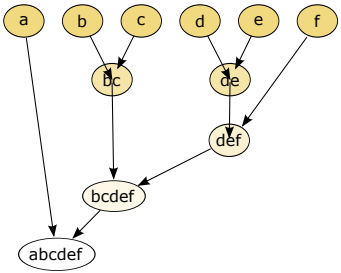
- Start with the same number of clusters as you have data points.
- At each stage cluster together close together cluster.
- Stop when you have enough clusters or everything is clustered together.

You get a tree out of the data that gives you information on clustering. You can use this to work the number of clusters that you should use.

Example



Raw Data



A possible hierarchy

Clustering Clusters

- Given your metric it is easy to cluster together single points. Two points are clustered if they are close together. For example b and c or d and e in the previous example.
- Given two clusters say $\{b, c\}$ and $\{d, e\}$ how do you decide to cluster them?

Linkage Criteria

There are lots of different criteria, each having their own advantages and disadvantages. Two common ones are:

Maximum or complete linkage clustering: $\max\{d(a,b) : a \in A, b \in B\}$

Minimum of single-linkage clustering: $\min\{d(a,b) : a \in A, b \in B\}$

Where $d(a,b)$ is the distance between two points, and A and B are the clusters. Cluster together the points with the smallest linkage between them.

Hierarchical Clustering

- The algorithm is not deterministic. Two clusters could be the same distance apart. You have to make a choice. As you move up the tree, it does not happen so much.
- You have to decide when to stop. Obviously clustering everything into a single point is not a good idea.
- It is not always clear what the best metric to choose for clustering.

k-means clustering

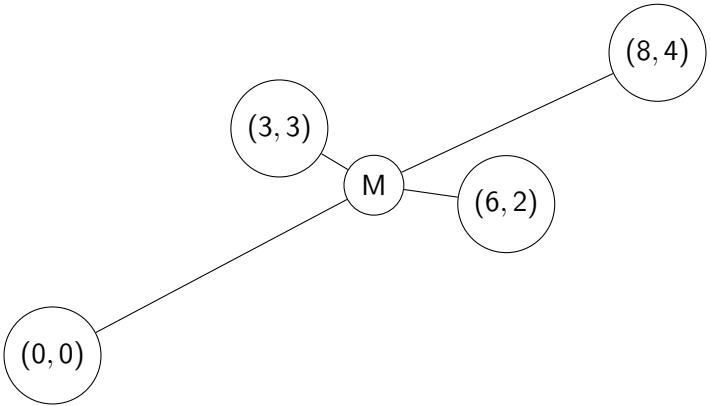
Given vectors $x_1, \ldots, x_n$ define the average as

$$\mu = \frac{1}{N} \sum_{i=1}^{n} x_i$$

This is also called the centroid or the barycentre.
You can prove that the point μ minimises the sum of the squared euclidean distances between itself and all the points in the set.
That is, it minimises

$$\sum_{i=1}^{n} ||x_i - \mu||^2$$

Centroid



$$\frac{1}{4}((0+3+6+8, 0+3+2+4)) = (17/4, 9/4) = (4.25, 2.25)$$

k-means clustering

Given a set of points $x_1, \ldots, x_n$ find a partition of the n-points into k sets $S = S_1, \ldots, S_k$ that minimises the objective

$$\sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2$$

Where

$$\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} x$$

So we want to find k centres $\mu_1, \ldots, \mu_k$ that minimises the spread or max distance in each cluster.

This is a declarative description of what we want, not an algorithm to find the clusters.

k-means clustering — Complexity

- NP-hard even for two clusters.

This means that you could in theory code up a SAT or a graph colouring problem as a clustering problem.
This also means that none of the heuristic algorithms are guaranteed to find the best clustering.

k-means clustering — Greedy Algorithm

Initialise $\mu_1, \ldots, \mu_k$ with random values.
Repeat the following two steps until convergence:

Clustering For each data point x_i assign its cluster to be th point μ_j that it is closest to.

Recompute μ For each cluster j recompute the cluster recompute μ_j to be

$$\frac{1}{|S_j|} \sum_{x \in S_j} x$$

Note that the sets $S_1, \ldots, S_j$ can change as the algorithm runs.

1 dimensional example

Random assignment $k_1 = 2$ and $k_2 = 3$.

$$0 \quad 1 \quad k_1 \quad k_2 \quad 5 \quad 6$$

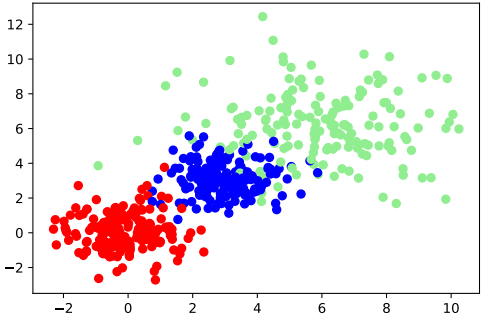
Points 0 and 1 are closest to k_1 and points 5 and 6 are closest to k_2.
New means $k_1 = (0 + 1)/2$ and $k_2 = (5 + 6)/2 = 5.5$

Clustering Example

$0 \quad k_1 \quad 1 \qquad\qquad 5 \quad k_2 \quad 6$

Again 0 and 1 are closets to k_1 and 5 and 6 are closets to k_2. So we stop as k_1 and k_2 don't change.
Note that the algorithm is a type of gradient descent.

What is the correct value of k?



How many clusters 1,2 or 3?

What is the correct value of k?

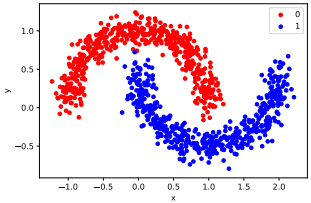
You can look at how objective changes

$$\sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2$$

With different numbers of clusters, and stop increasing the number of clusters when you don't get any more big drops in the objective (the elbow method).
Often the number of clusters is application dependent and depends on what you doing with your clustering algorithm. There might be an application dependent method of evaluating the number of clusters.

Limitations of K-means Clustering

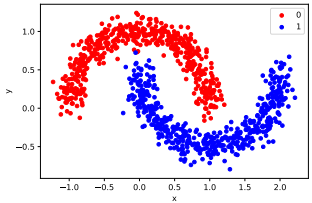
- Clusters can only be blobs. You could not learn clusters for a data set like this:



- You have decide what value of k you should pick.

DBSCAN

- K-means clustering cannot deal with nested non-convex blobs.



In clusters the data points are more dense, and clusters are separated by regions of low intensity.

DBSCAN

Intuitive idea:

- For each point, try to estimate if it is in a high density region, on the border of a region or an outlier.
- Pick an unclassified point in a high density region, then use a flood fill algorithm and add the neighbours to the same cluster.

DBSCAN — Estimating the density

Classify points into

Core points Points in high density regions

Reachable points Points less than a certain distance from each other. You can reach non core points.

Outliers or noise points Points that cannot be reached from core points.

DBSCAN — Estimating the density

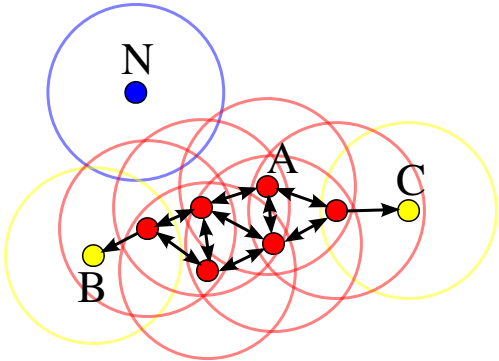
Pick some radius ϵ and draw a sphere around each point in the data set. Pick some value m (often called `minPts` or `minSamples`).

Core point: A point is a *core point* if there are at least m points with a distance of ϵ of it.

Reachable: A point q is *reachable* from a core point p if $d(p, q) < \epsilon$.

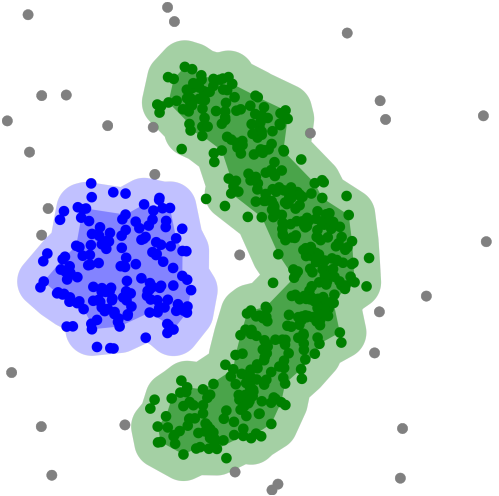
Outliers or noise points Points that cannot be reached from core points.

Note that you can reach a non core point from a core point.



The red points are core points, the yellow points are reachable non-core points and the blue point is an outlier.

DBSCAN works with Nonlinear Regions



Why are there so many clustering Algorithms

Scikit-learn has 11 different clustering algorithms not counting the various levers and knobs you can set for each algorithm.

Are we just too stupid to come up with the best clustering algorithm?

- Even simple k-means clustering is NP-hard, so we should expect that all efficient algorithms are some sort of approximation to the best result that sometimes work well.[6]
- In fact it is mathematically impossible to come up with a perfect clustering algorithm.[7]

This is what makes machine learning fun. You have to get familiar with the strengths and weaknesses of the different algorithms.

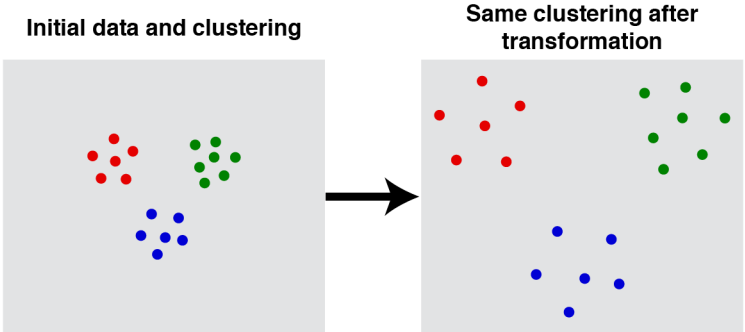
[6] This is why you should learn some complexity theory.

[7] "An impossibility Theorem for Clustering". Jon Kleinberg
https://proceedings.neurips.cc/paper/2002/hash/43e4e6a6f341e00671e123714de019a8-Abstract.html

Impossibility of Perfect Clustering[8]

Three properties:

- *Scale invariance:* If we have a dataset and multiply all the distances between points by a constant factor, then we should get the same clusters.

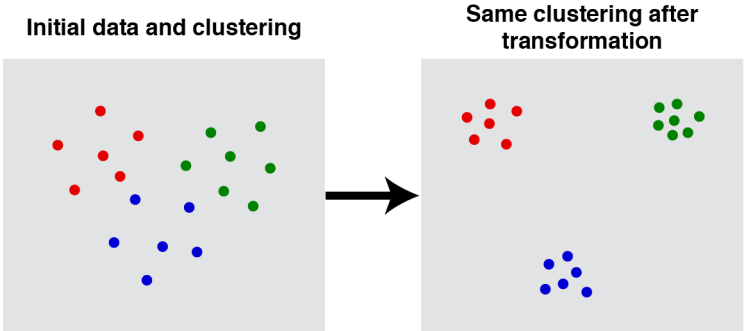


[8] Picture taken from `https://alexhwilliams.info/itsneuronalblog/2015/10/01/clustering2/`

Impossibility of Perfect Clustering[9]

Three properties:

- *Consistency:* If you move the clusters apart and stretch or contract the individual clusters the you should get the same clustering.

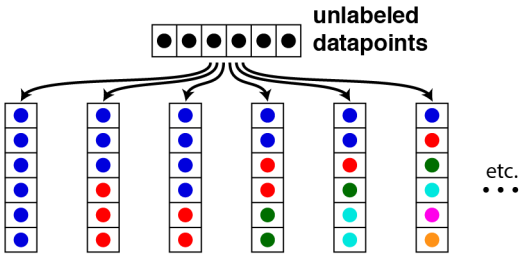


[9] Picture taken from `https://alexhwilliams.info/itsneuronalblog/2015/10/01/clustering2/`

Impossibility of Perfect Clustering[10]

Three properties:

- *Richness:* All possible clustering are possible. A bit hard to explain without too much mathematics. But certain natural clustering should not be possible. You algorithm should not be blind to some clustering.



[10] Picture taken from `https://alexhwilliams.info/itsneuronalblog/2015/10/01/clustering2/`

Impossibility of Perfect Clustering

It is impossible to have a clustering algorithm that is Rich, Scale invariant and Consistent. At best you can have two properties at the same time.

Clustering Overview and things to think about

- Lots of parameters and distance measures to pick.
- Clustering high dimensional data is quite hard, picking the correct value of ϵ can be difficult and your intuitions about 2 and 3 dimensions does not always help in high dimension. There is a lot more space in high-dimensional space (what ever that might mean).
- There are lots more clustering algorithms that you can try, some are better adapted to different types of data.
- Clustering is a hard computational problem, so all the known algorithms are fast algorithms that don't always give the perfect answer or in some cases the will run slow. Most people prefer fast.
- Mathematically there is no perfect clustering algorithm. It is a bit hard to understand the implications of this, but it is one reason that there are a lot of different algorithms out there.