

3/38

5:09:51



Data Mining


(search(ing) for PATTERNS)



What is data mining?

Data mining is the science of **extracting useful information from large datasets.**

 At the heart of data mining is the process of **discovering RELATIONSHIPS** between parts of a dataset.

“Data mining is the analysis of (often large) observational data sets to **find unsuspected relationships** and to summarize the data in novel ways that are both understandable and useful to the **data owner**. The relationships and summaries derived through a data mining exercise are often referred to as models or **patterns**”. 

 The term 'trend' is used to describe patterns that occur or change over time.

So then what is machine learning?

How is ML different from DM?



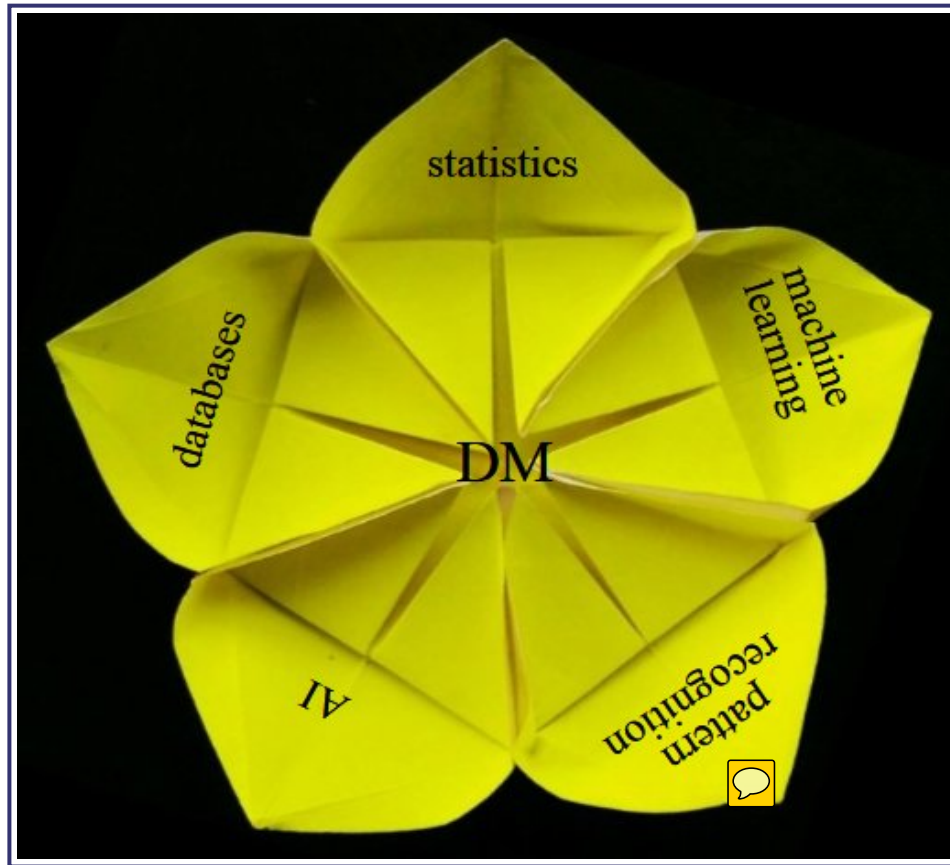
Machine learning is the **process of TRAINING** an algorithm on an EXISTING dataset in order to have it discover relationships (so as to create a model/pattern/trend), and USING the result to analyze NEW data.

[Here](#) is Andrew Ng's 'classic' Stanford course on ML that is hosted at Coursera, which he co-founded (Andrew is now with [Baidu](#)).



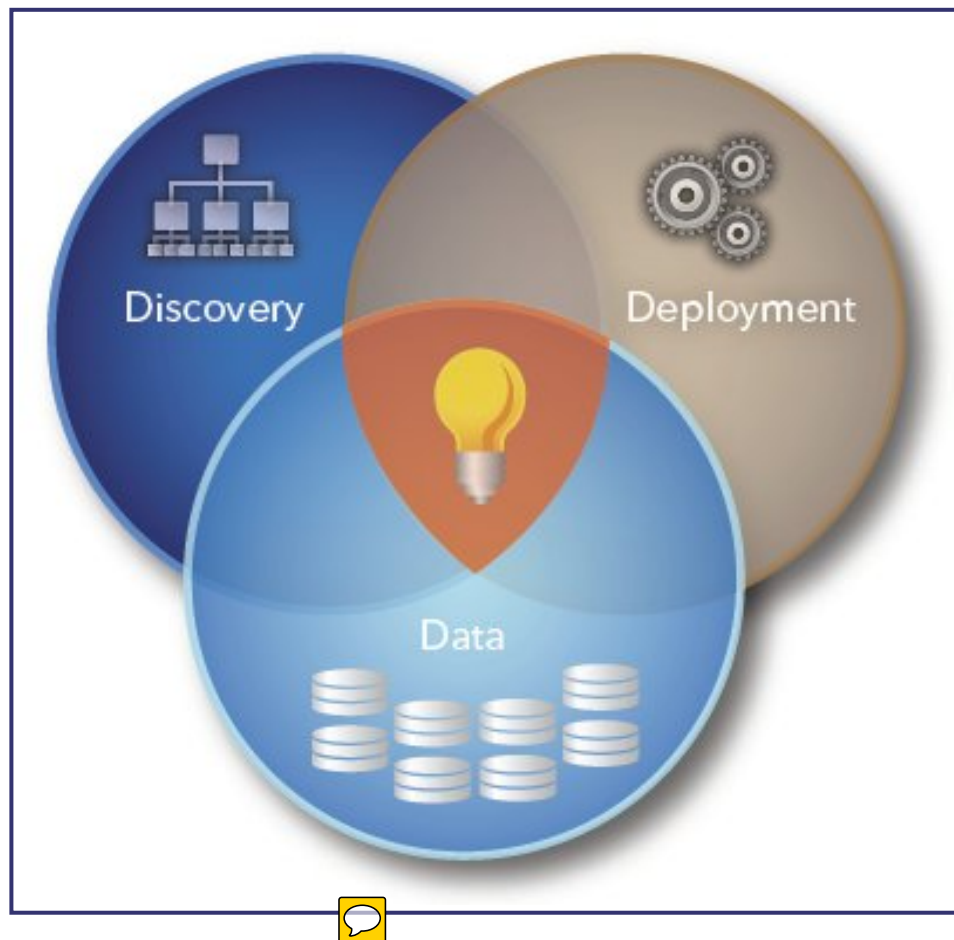
DM is inter-disciplinary

Here is how data mining relates to existing fields:

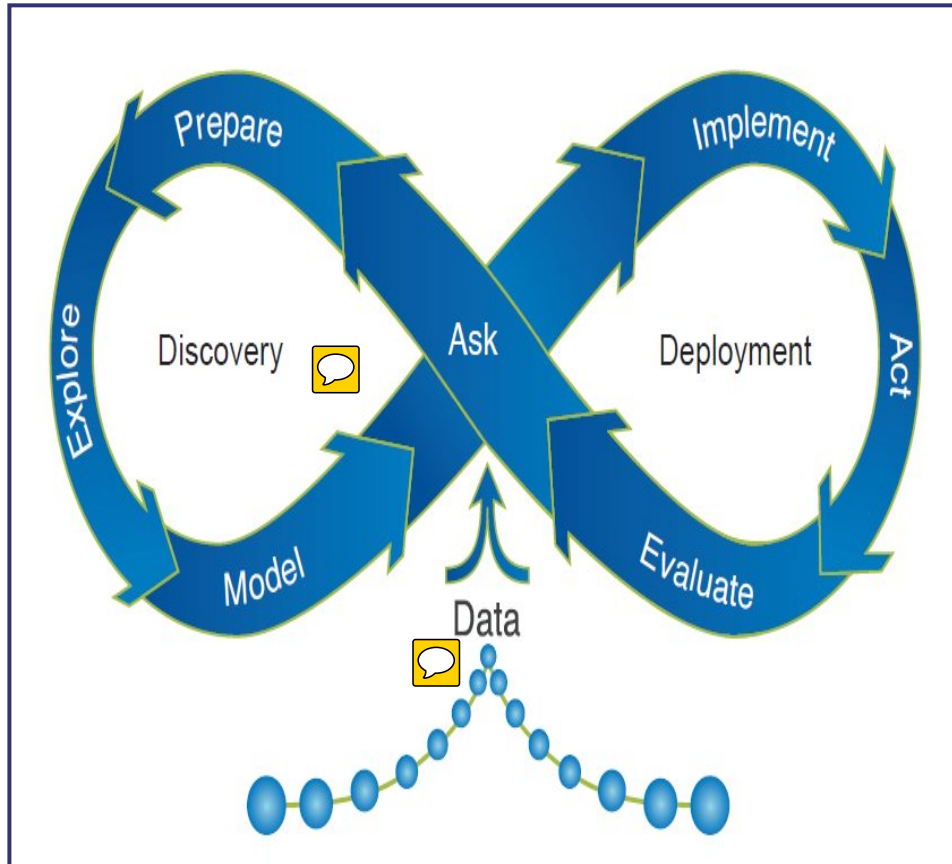


The data mining cycle

By nature, most data mining is cyclical. Starting with data, mining leads to discovery, which leads to action ("deployment"), which in turn leads to new data – the cycle continues.







A more nuanced depiction of the cycle:



Some uses of data mining

As you can imagine, data mining is useful in dozens (hundreds!) of fields!! Almost ANY type of data can be mined, and results put to use. Following are typical uses:

- predicting which customers will purchase what products and when 
- deciding should insurance rates be set to ensure profitability 
- predicting equipment failures, reducing unnecessary maintenance and increasing uptime to optimize asset performance 
- anticipating resource demands
- predicting which customers are likely to leave and what can be done to retain them
- detecting fraud 
- minimizing financial risk
- increasing response rates for marketing campaigns

Uses, cont'd



Common Applications for Data Mining Across Industries

Business Question	Application	What Is Predicted?
How to better target product/service offers?	Profiling and segmentation.	Customer behaviors and needs by segment.
Which product/service to recommend?	Cross-sell and up-sell.	Probable customer purchases.
How to grow and maintain valuable customers?	Acquisition and retention.	Customer preferences and purchase patterns.
How to direct the right offer to the right person at the right time?	Campaign management.	The success of customer communications.
Which customers to invest in and how to best appeal to them?	Profitability and lifetime value.	Drivers of future value (margin and retention).



Industry-Specific Data Mining Applications





Business Question	Application	What Is Predicted?
How to assess and control risk within existing (or new) consumer portfolios?	Credit scoring (banking).	Creditworthiness of new and existing sets of customers.
How to increase sales with cross-sell/up-sell, loyalty programs and promotions?	Recommendation systems (online retail).	Products that are likely to be purchased next.
How to minimize operational disruptions and maintenance costs?	Asset maintenance (utilities, manufacturing, oil and gas).	The real drivers of asset or equipment failure.
How to reduce health care costs and satisfy patients?	Health and condition management (health insurance).	Patients at risk of chronic, treatable/preventable illness.
How to decrease fraud losses and lower false positives?	Fraud management and cybersecurity (government, insurance, banks).	Unknown fraud cases and future risks.
How to bring drugs to the marketplace quickly and effectively?	Drug discovery (life sciences).	Compounds that have desirable effects.

Data mining algorithms: categories

Practically all **data mining algorithms** neatly fit into one of these 4 categories:

-  • **Classification:** involves **LABELING** data 
-  • **Clustering:** involves **GROUPING** data, based on similarity 
- **Regression:** involves **COUPLING** data  
- **Rule extraction:** involves **RELATING** data  

We'll look at examples in each category, that will provide you a concrete understanding of the above summarization. 

Algorithms can also be classified as being a **'supervised' learning method** (where **we need to provide categories** for, ie. train, using known outcomes), or an **'unsupervised' method** (where we **provide just the data** to the algorithm, leaving it to learn on its own).  



Algorithms!

Now we can start discussing specific algorithms (where each one belongs to the four categories we just outlined).

Challenge/fun – can each algorithm be talked about, without using any math at all? You get the 'big picture' (a clear, intuitive understanding) that way.. After that, we can look at equations/code for the algorithms, to learn the details (God/the devil **is** in the details :)).





Algorithm: **Decision trees** (eg. C4.5, C5.0 etc.)



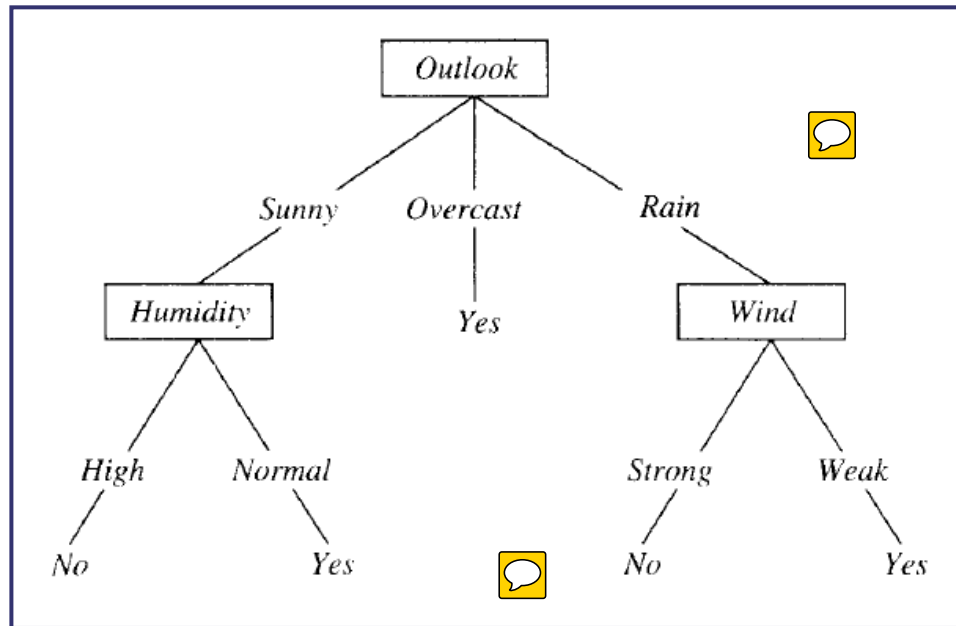
Classification and regression trees (aka decision trees) are **machine-learning methods** for constructing **prediction** models from data. The models are obtained by recursively partitioning the data space and fitting a simple prediction model within each partition.



The decision tree algorithm works like this:

- **user provides a set of input (training) data**, which consists of features (independent parameters) **for each piece of data, AND an outcome** (a 'label', ie. a class name)
- the algorithm uses the data to build a **'decision tree'** (binary tree, with feature-based conditionals at each non-leaf node), leading to the outcomes (known labels) at the terminals
- the user makes use of the tree by providing it new data (just the feature values) – the **algorithm uses the tree to 'classify' the new item into one of the known outcomes** (classes)

Should we play tennis? Depends (on the weather) :)

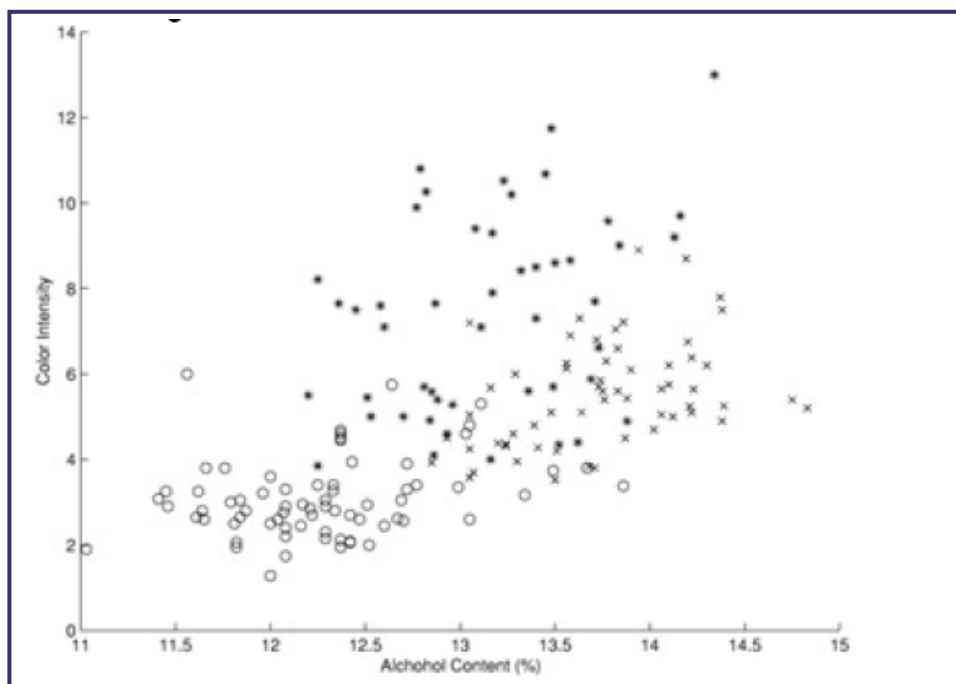


This is a VERY simple algorithm! The entire tree is a disjunction (where the branches are) of conjunctions (that lead down from root to leaves).

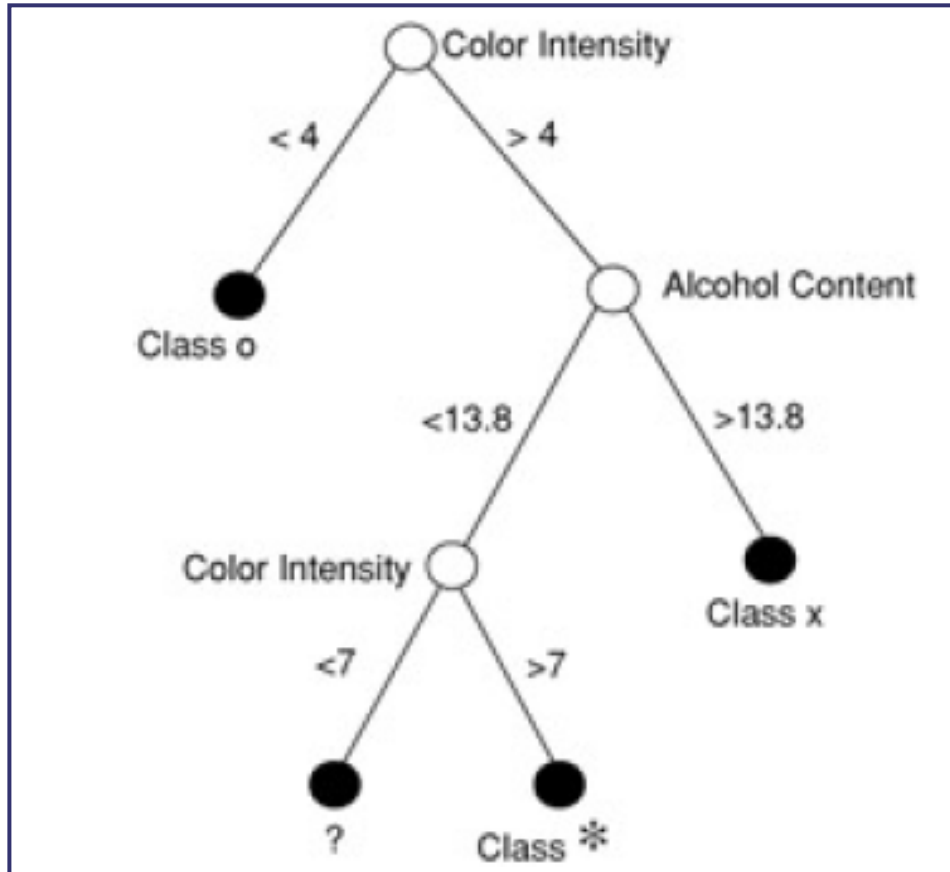
If the **outcome** (dependent, or 'target' or 'response' variable) **consists of classes** (ie. it is 'categorical'), the tree we build is called a **classification tree**. On the other hand if the target variable is continuous (a **numerical quantity**), we build a **regression tree**. **Numerical quantities can be ordered**, so such data is called '**ordinal**'; **categories are names, they provide 'nominal' data**. Given that,

nominal data results in classification trees, and ordinal data results in regression trees.

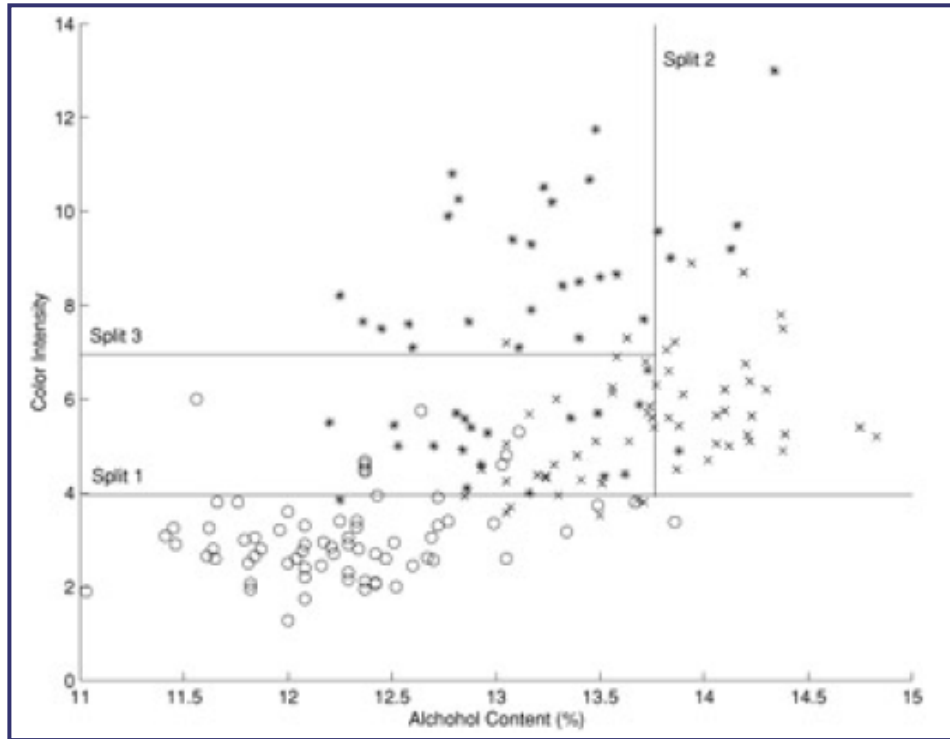
Here is another example (from 'Principles of Data Mining' by David Hand et. al.). Shown is a 'scatter plot' of a set of wines – color intensity vs alcohol content:



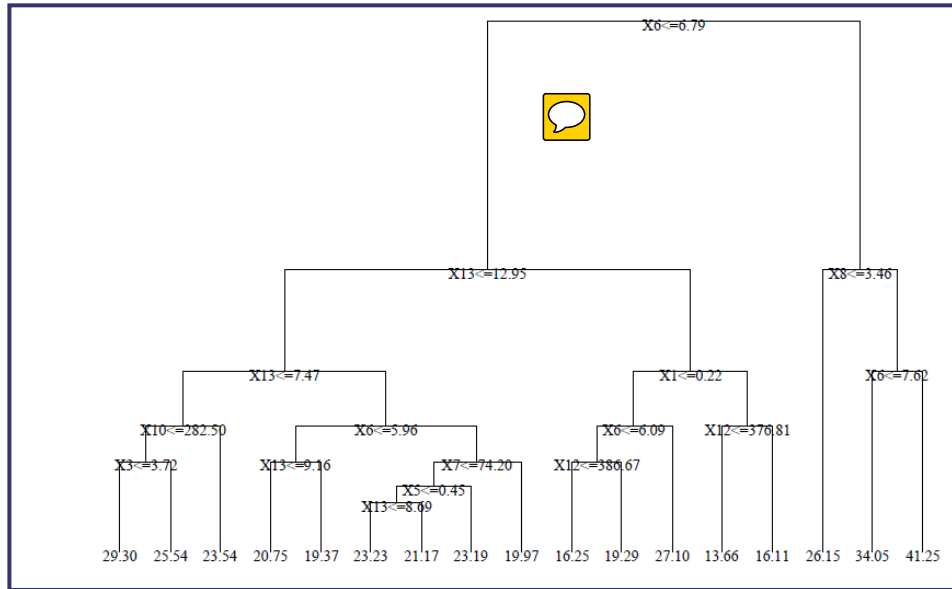
The classification tree for the above data is this:



When we superpose the 'decision boundaries' from the classification tree on to the data, we see this:



A sample regression tree is shown below – note that the predictions at the leaves are numerical (not categorical):






Note – algorithms that create classification trees and regression trees are referred to as **CART algorithms** (guess what CART stands for :)).

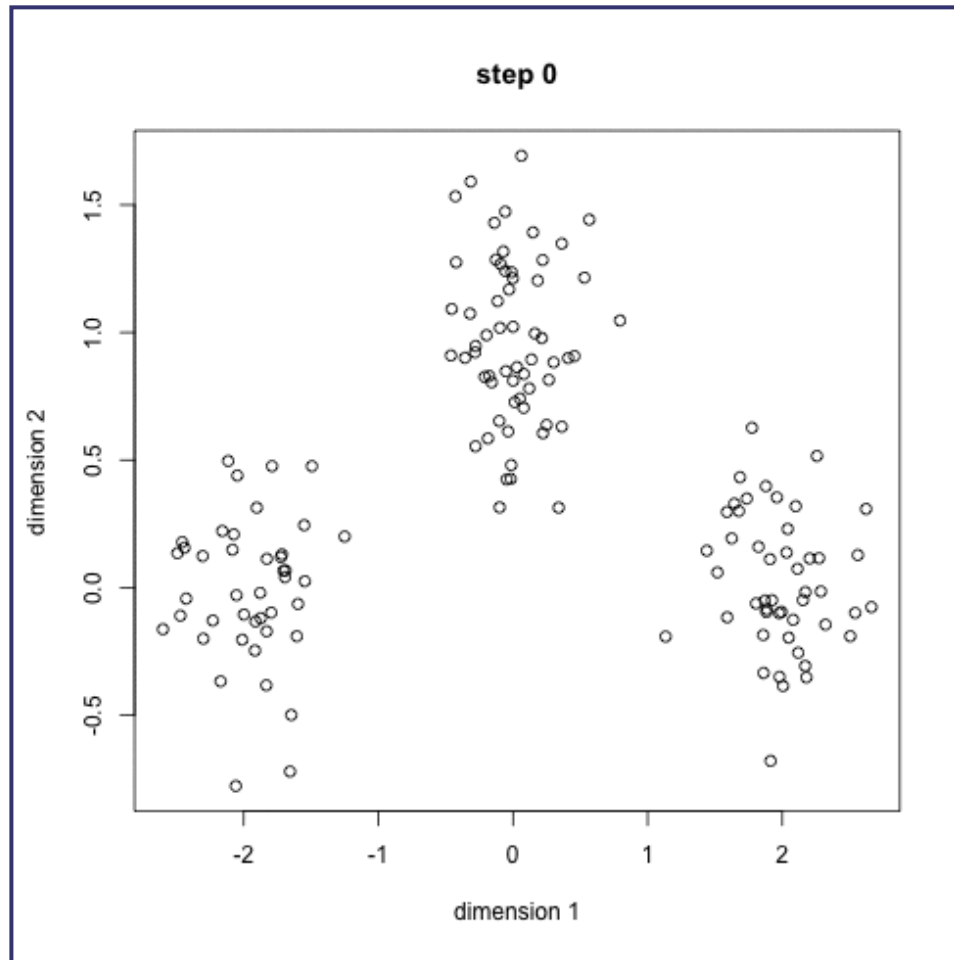


Algorithm: k-means clustering

This algorithm creates 'k' number of "clusters" (sets, groups, aggregates..) from the input data, using some measure of closeness (items in a cluster are closer to each other than any other item in any other cluster). This is an example of an unsupervised algorithm – we don't need to provide training/sample clusters, the algorithm comes up with them on its own.

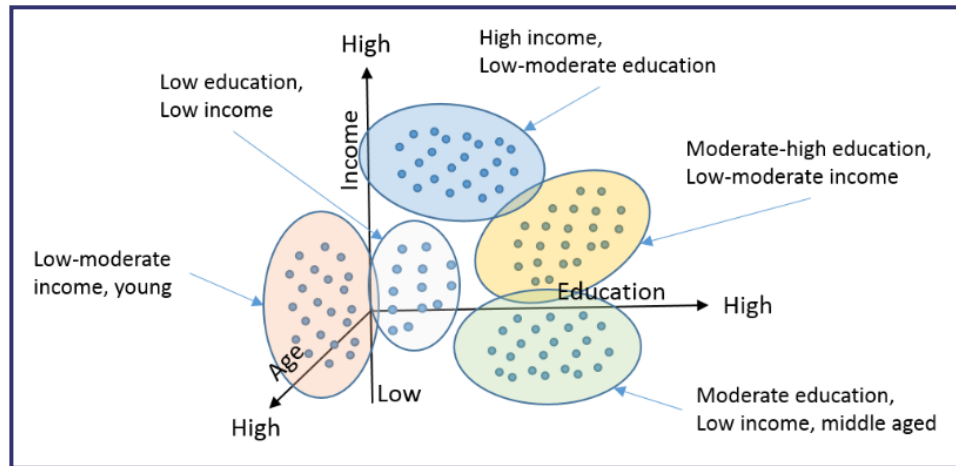
If each item in our (un-clustered) dataset has 'n' attributes, it is eqvt to a point in n-dimensional space (eg. n can be 120, for Amazon!). We are now looking to form clusters in n-D space! 

Approach: start with 'n' random locations ('centroids') in/near the dataset; assign each input point (our data) to the closest centroid; compute new centroids (from our data); iterate (till convergence is reached). 




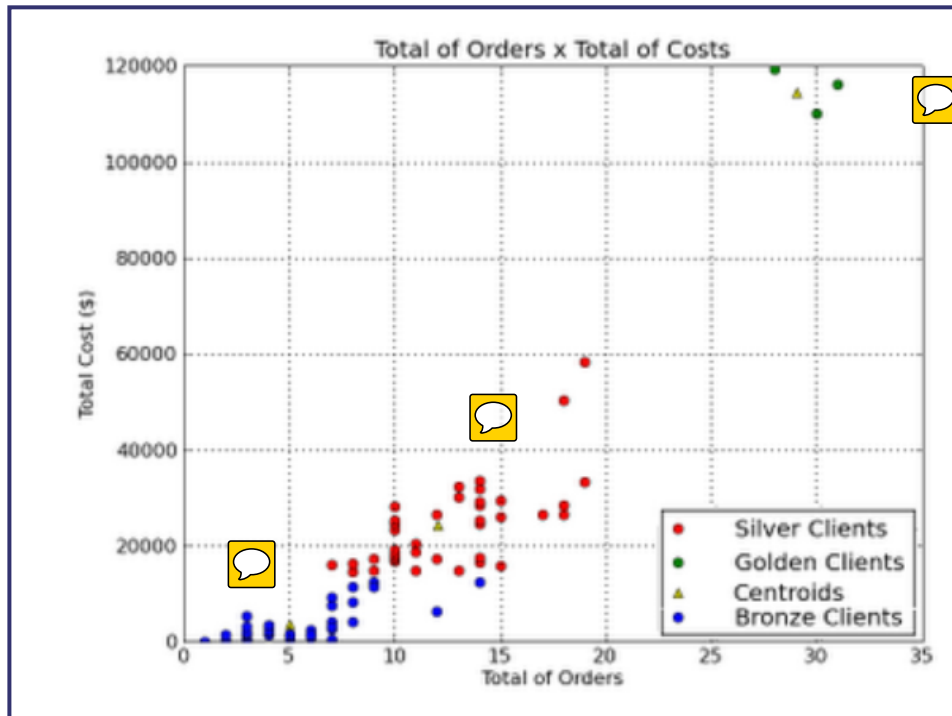
Here is a use case for doing clustering:





Again, very simple to understand/code!

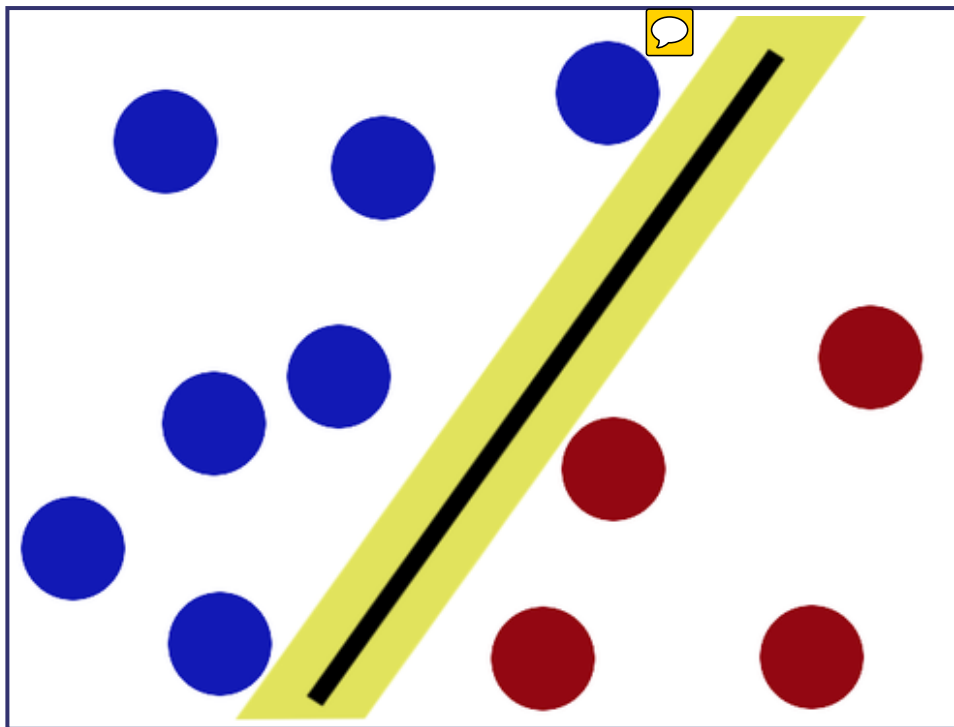
Another example – here, we classify clients of a company, into 3 categories, based on how many orders they placed, and what the orders cost:



Algorithm: Support Vector Machine (SVM)

-  An SVM always partitions data (classifies) them into TWO sets – uses a 'slicing' hyperplane (multi-dimensional equivalent of a line), not a decision tree.  

The hyperplane maximizes the gap on either side (between itself and features on either side). This is to minimize chances of misclassifying new data.





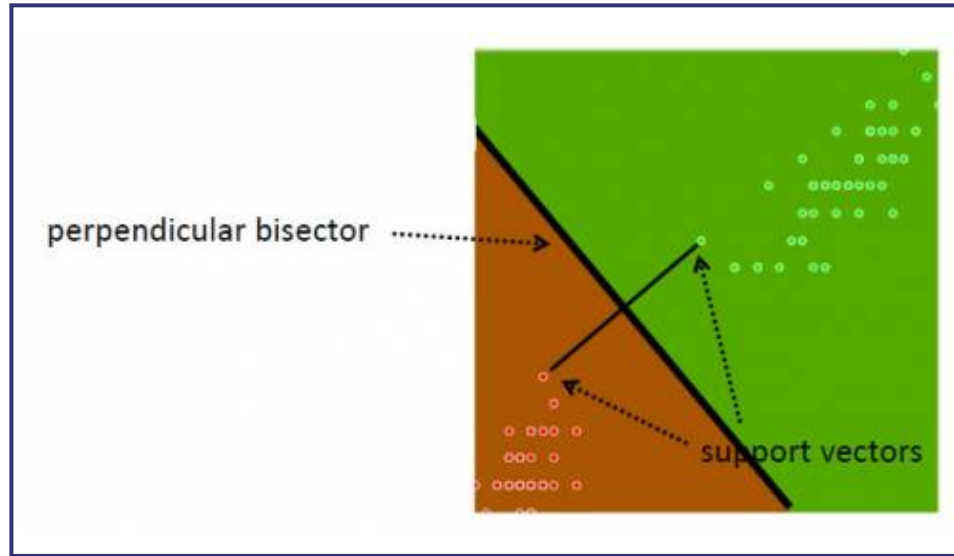
On either side, the **equidistant data points closest to the hyperplane** are the '**support vectors**' (note that there can be several of these, eg. 3 on one side, 2 on the other). Special case – if there is a single support (closest point) on either side, in 2D, the separator is the perpendicular bisector of the line segment joining the supports; if not, the separating line/plane/hyperplane needs to be calculated by finding two parallel hyperplanes with no data in between them, and maximizing their gap. The goal is to achieve "margin maximization".



How do we know that a support data point is indeed a **support vector**? If we move the data point and therefore the boundary moves, that is a support vector :) In other words, non-support data can be moved around a bit, that will not change the boundary (unless the movement brings it to be inside the current margin).



Note – in our simplistic setup, we make two assumptions: that our two classes of data are indeed separable (not inter-mingled), and that they are **linearly separable**. FYI, it is possible to create SVMs even if both the assumptions aren't true.




Here's some geek humor for ya:





Algorithm: A priori



Looking for hidden relationships in large datasets is known as association analysis or association rule learning. 

 The A priori algorithm comes up with association rules (relationships between existing data, as mentioned above). Here is an example: outputting "items purchased together" (aka 'itemsets'), eg. chip, dip, soda, from grocery transaction records.

 Inputs: data, size of the desired itemsets (eg. 2, 3, 4..), the **support count** (number of times the itemset occurs, divided by the total # of data items) and the **confidence** (conditional probability of an item being in a datum, given another item). 

Transaction number	Items
0	soy milk, lettuce
1	lettuce, diapers, wine, chard
2	soy milk, diapers, wine, orange juice
3	lettuce, soy milk, diapers, wine
4	lettuce, soy milk, diapers, orange juice



In the above, support for {soy milk,diapers} is 3/5; the confidence for {diapers} \rightarrow {wine} is $\text{support}(\{\text{diapers}, \text{wine}\}) / \text{support}(\{\text{diapers}\})$, which is 3/5 over 4/5, which is 0.75.



What is specified to the algorithm as input, is a (support,confidence) pair, and the algorithm outputs all matching associations – we would then make appropriate use of the results (eg. co-locate associated items in a store, put up related ads in the search page, print out discount coupons for associated products while the customer is paying their bill nearby, etc.).



Below is another 'shopping basket' example, with products in columns, customers in rows.

basket-id	A	B	C	D	E		
t_1	1	0	0	0	0		
t_2	1	1	1	1	0		
t_3	1	0	1	0	1		
t_4	0	0	1	0	0		
t_5	0	1	1	1	0		
t_6	1	1	1	0	0		
t_7	1	0	1	1	0		
t_8	0	1	1	0	1		
t_9	1	0	0	1	0		
t_{10}	0	1	1	0	1		

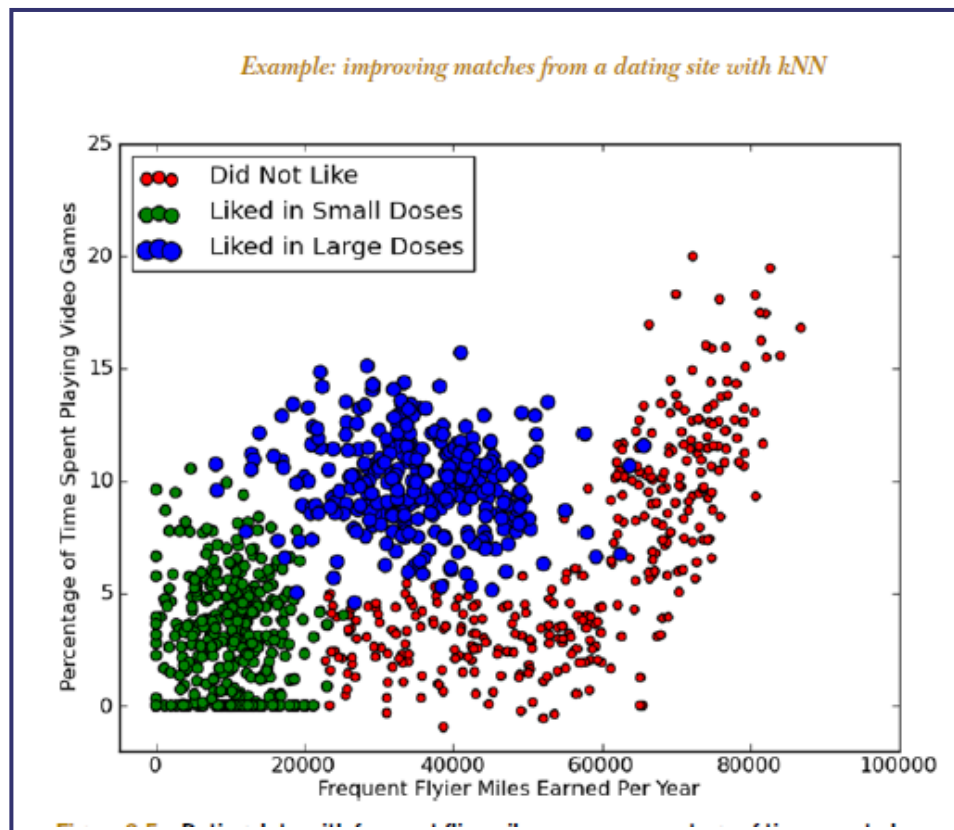
If we set a frequency (ie support) threshold of 0.4, we can see that the matching items are {A}, {B}, {C}, {D}, {AC}, {BC} (each of these occur ≥ 4 times). We can see that {AC} has an accuracy (ie. confidence) of $4/6=2/3$, and {B,C} has an accuracy of $5/5=1$ (EVERY time B is purchased, C is also purchased!).



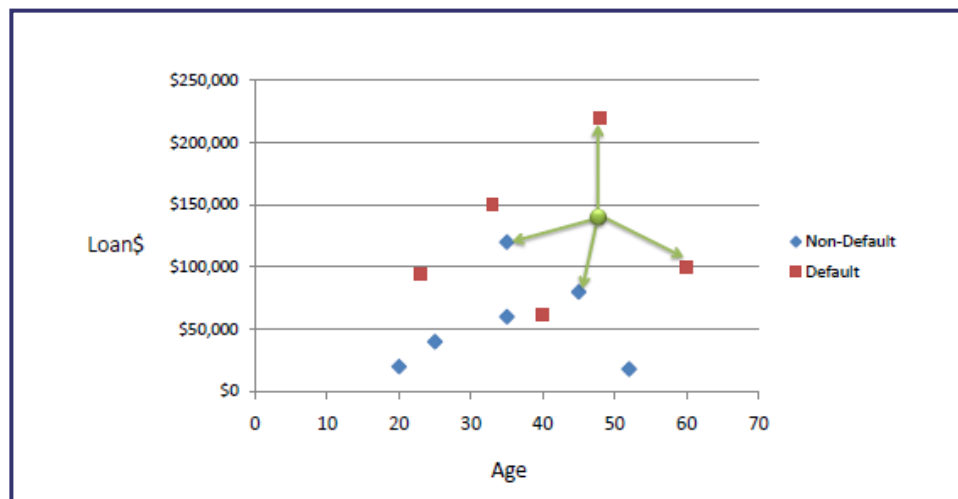
Algorithm: kNN



kNN (k Nearest Neighbors) algorithm picks 'k' nearest neighbors, closest to our unclassified (new) point, considers the 'k' neighbors' types (classes), and attempts to classify the unlabeled point using the neighbors' type data – majority wins (the new point's type will be the type of the majority of its 'k' neighbors).



Here is an example of 'loan defaulting' prediction. With $k=3$, and a new point of (42,142000), our neighbors have targets of Non Default, Non Default, Default – so we assign 'Non Default' as our target.




Note that if $k=1$, we assign as our target, that of our closest point.

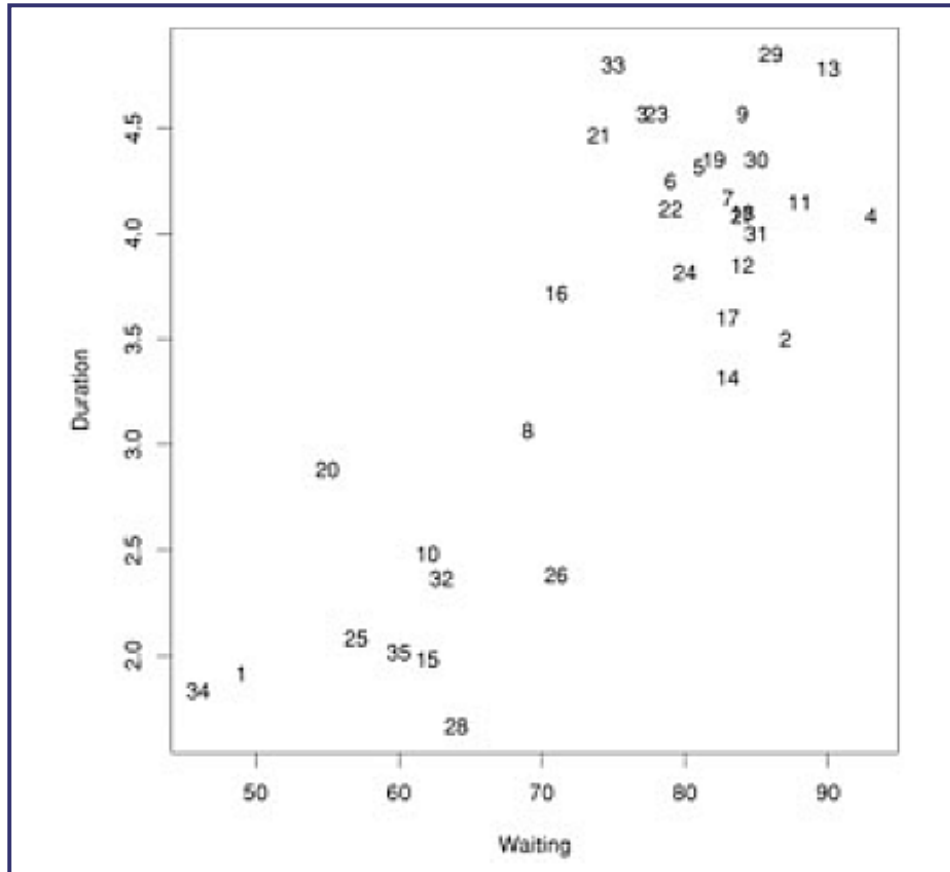
Also, to eliminate excessive influence of large numerical values, we usually **normalize our data** so that all values are 0..1.


kNN is a 'lazy learner' – just stores input data, uses it only when classifying an unlabeled (new) input. VERY easy to understand, and implement!

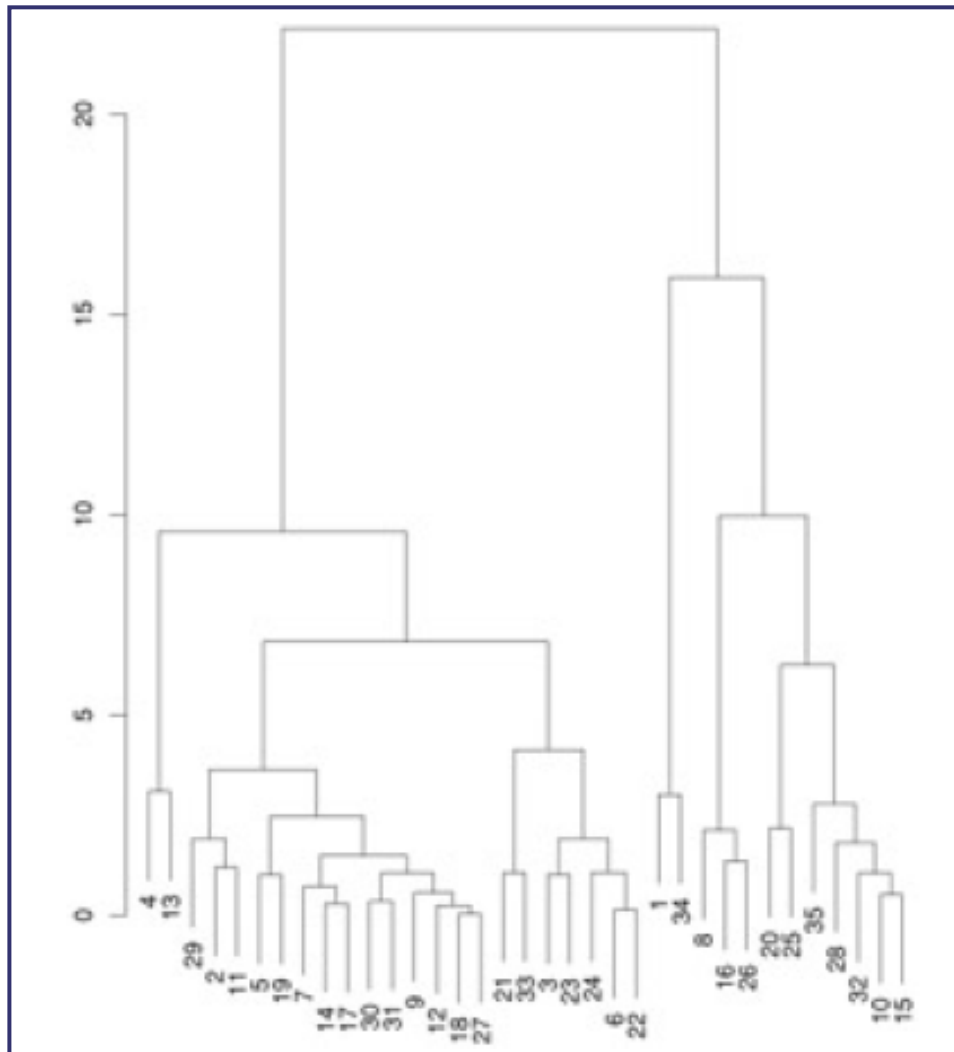
Algorithm: Hierarchical clustering

In some situations (where it is meaningful to do so), it is helpful to separate items in a dataset into hierarchical clusters (clusters of clusters of..). There are two ways to look at this – as the merging of smaller clusters into bigger superclusters, or dividing of larger clusters into finer scale ones. 

Below is a dataset that plots, for Yellowstone National Park, the waiting time between geyser eruptions, and length of eruptions. The data points are numbered, just to be able to identify them in our cluster diagram.



Given this data, we **run** a hierarchical clustering algorithm, whose output is a **'dendrogram'** (tree-like structure) that **shows the merging of clusters:** 



How do we decide what to merge? A popular strategy – pick clusters that lead to the smallest increase in sum of squared distances (in the above diagram, that is what the vertical bar lengths signify). The dendrogram shows that we need to start by merging

#18 and #27, which we can verify by looking at the scatter plot (those points almost coincide!).

Algorithm: Neural nets!



A form of 'AI' – use neuron-like connected units that store learning (training) data that has known outcomes, use it to be able to gracefully respond to new situations (non-training, 'live' data) – like how humans/animals learn!



Neural networks (NNs) can be used to:

- recognize/classify features – traffic, terrorists, expressions, plants, words..
- detect anomalies – unusual CC activity, unusual machine states, gene sequences, brain waves..
- predict exchange rates, 'likes'..

As you can imagine, 'Big Data' can help in all of the above! The bigger the training set, the better the learning (more conditions, more features), better the result.





Algorithm: NN (cont'd)



Neural nets are excellent for speech recognition tasks.



A speech recognition system has several stages (ref. [Geoff Hinton](#)) :

- * **Pre-processing: Convert the sound wave into a vector** of acoustic coefficients. Extract a new vector about every 10 mille seconds. 
- * **The acoustic model: Use a few adjacent vectors of** acoustic coefficients to place bets on which part of which phoneme is being spoken. 
- * **Decoding: Find the sequence of bets that does** the best job of fitting the acoustic data and also fitting a model of the kinds of things people say.

Results of NN modeling of speech:

Word error rates from MSR, IBM, & Google
(Hinton et. al. IEEE Signal Processing Magazine, Nov 2012)

The task	Hours of training data	Deep neural network	Gaussian Mixture Model	GMM with more data
Switchboard (Microsoft Research)	309	18.5%	27.4%	18.6% (2000 hrs)
English broadcast news (IBM)	50	17.5%	18.8%	
Google voice search (android 4.1)	5,870	12.3% (and falling)		16.0% (>>5,870 hrs)

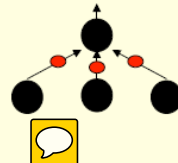
Algorithm: NN (cont'd)

Here is a barebones overview of how NNs work..

The brain (specifically, learning/training) is modeled after strengthening **relevant neuron connections** – neurons communicate (through axons and dendrites) dataflow-style (neurons send output signals to other neurons):

How the brain works on one slide!

- Each neuron receives inputs from other neurons
 - A few neurons also connect to receptors.
 - Cortical neurons use spikes to communicate.
- The effect of each input line on the neuron is controlled by a synaptic weight
 - The weights can be positive or negative.
- The synaptic weights **adapt** so that the whole network learns to perform useful computations
 - Recognizing objects, understanding language, making plans, controlling the body.
- You have about 10^{11} neurons each with about 10^4 weights.
 - A huge number of weights can affect the computation in a very short time. Much better bandwidth than a workstation.

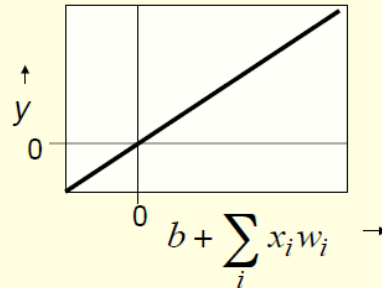


Linear output: input values get passed through 'verbatim' (not very useful to us, does not happen in real brains!):

Linear neurons

- These are simple but computationally limited
 - If we can make them learn we may get insight into more complicated neurons.

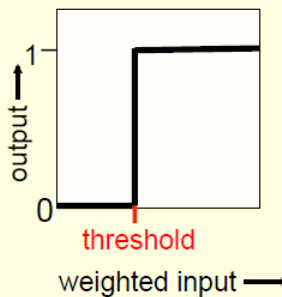
$$y = b + \sum_i x_i w_i$$



A better model is when a neuron outputs a 1 (stays 0 to start with) ("fires") if and when its combined inputs exceed a threshold value:

Binary threshold neurons

- McCulloch-Pitts (1943): influenced Von Neumann.
 - First compute a weighted sum of the inputs.
 - Then send out a fixed size spike of activity if the weighted sum exceeds a threshold.
 - McCulloch and Pitts thought that each spike is like the truth value of a proposition and each neuron combines truth values to compute the truth value of another proposition!



Another option is to convert the 'step' pulse to a ramp:



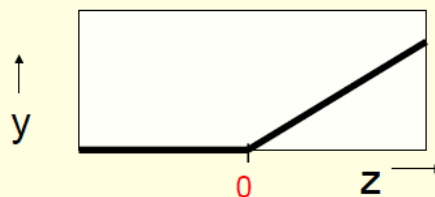


Rectified Linear Neurons (sometimes called linear threshold neurons)

They compute a **linear** weighted sum of their inputs.
The output is a **non-linear** function of the total input.

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$



Even better – use a smoother buildup of output:

Sigmoid neurons

- These give a real-valued output that is a smooth and bounded function of their total input.

- Typically they use the logistic function
- They have nice derivatives which make learning easy (see lecture 3).

$$z = b + \sum_i x_i w_i \quad y = \frac{1}{1 + e^{-z}}$$



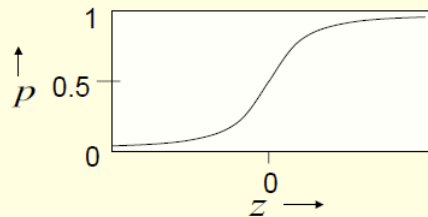
***Even* better – use a sigmoidal probability distribution for the output:**



Stochastic binary neurons

- These use the same equations as logistic units.
 - But they treat the output of the logistic as the **probability** of producing a spike in a short time window.
- We can do a similar trick for rectified linear units:
 - The output is treated as the Poisson rate for spikes.

$$z = b + \sum_i x_i w_i \quad p(s=1) = \frac{1}{1 + e^{-z}}$$



With the above info, we can start to **build our neural networks!**



* we create LAYER upon LAYER of neurons – each layer is a set (eg. column) of neurons, which feed their (stochastic) outputs



downstream, to neurons in the next (eg. column to the right) layer, and so on

* **each layer is responsible for 'learning' some aspect of our target – usually the layers operate in a hierarchical (eg. raw pixels to curves to regions to shapes to FEATURES) fashion**



* **a layer 'learns' like so: its input weights are adjusted (modified iteratively) so that the weights make the neurons fire when they are**

given only 'good' inputs.



Here is how to visualize the layers.



The above steps can be summarized this way:

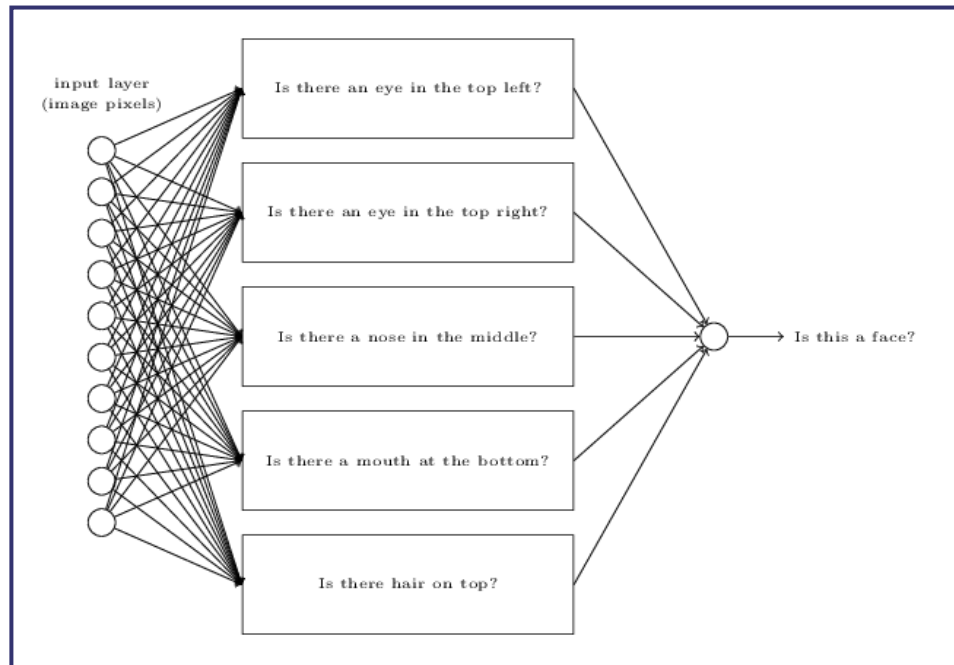
- We start by choosing a **model-class**: $y = f(\mathbf{x}; \mathbf{W})$
 - A model-class, f , is a way of using some numerical parameters, \mathbf{W} , to map each input vector, \mathbf{x} , into a predicted output y .
- Learning usually means adjusting the parameters to reduce the discrepancy between the target output, t , on each training case and the actual output, y , produced by the model.
 - For regression, $\frac{1}{2}(y - t)^2$ is often a sensible measure of the discrepancy.
 - For classification there are other measures that are generally more sensible (they also work better).



Note that learning (ie. iterative weights modification/adjustment) works via 'backpropagation', with weight adjustments starting from the last hidden layer (closest to the output layer) to the first hidden layer (closest to the input layer). Backpropagation aims to reduce the ERROR between the expected and the actual output, for a given training input. Two parameters to guide convergence: learning rate, momentum.



As per the above, here is a schematic showing how we could look for a face:






NN-based learning has started to REVOLUTIONIZE AI, thanks in no small part to Big Data (BILLIONS of images, tens of thousands of hours of video/audio, to use for training), better algorithms, faster computing platforms (MR, GPUs, MR on GPUs..) – more on this ("Deep Learning") later!

Algorithm: EM (Expectation Maximization)



EM is a rather 'magical' algorithm that can solve for a Catch-22 (circular reference) set of values!

 Imagine we have a statistical model that has some parameters, and  some 'latent' (hidden) variables. The model can operate on new data (predict/classify), given training data. Sounds like a straightforward DM algorithm (eg. k means clustering), except that it is not! 

We do not know the values for the parameters or latent variables!
We DO have observed/collected data, which the model should be able to act on.

The algorithm works as follows:

- start with random (!) values for the parameters
- use these to compute probabilities for all possible values for each hidden var, then do a weighted average to compute the best value (this is the 'E' step)
- use the hidden vars' values to improve the parameters ('M' step)
- iterate the above two steps till values converge



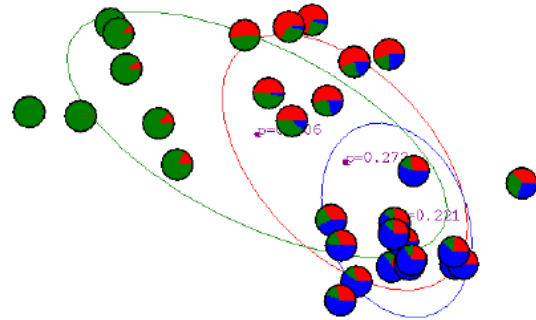
As counter-intuitive as it sounds, this does work!

Algorithm: EM (cont'd)

The next 4 slides show an example of EM being used to compute clustering on a dataset:

Algorithm: EM (cont'd)

After first
iteration

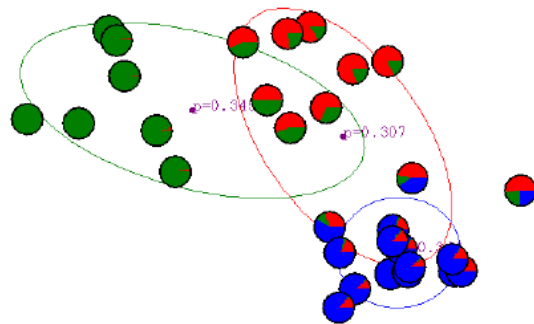


Copyright © 2001, 2004, Andrew W. Moore

Clustering with Gaussian Mixtures: Slide 41

Algorithm: EM (cont'd)

After 3rd
iteration

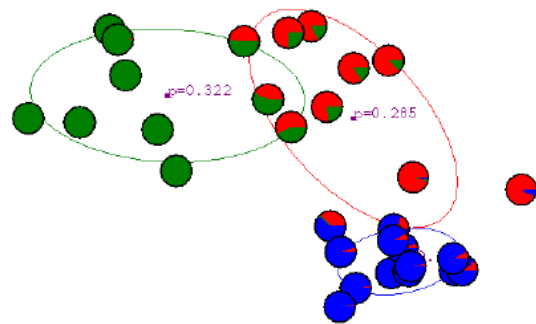


Copyright © 2001, 2004, Andrew W. Moore

Clustering with Gaussian Mixtures: Slide 43

Algorithm: EM (cont'd)

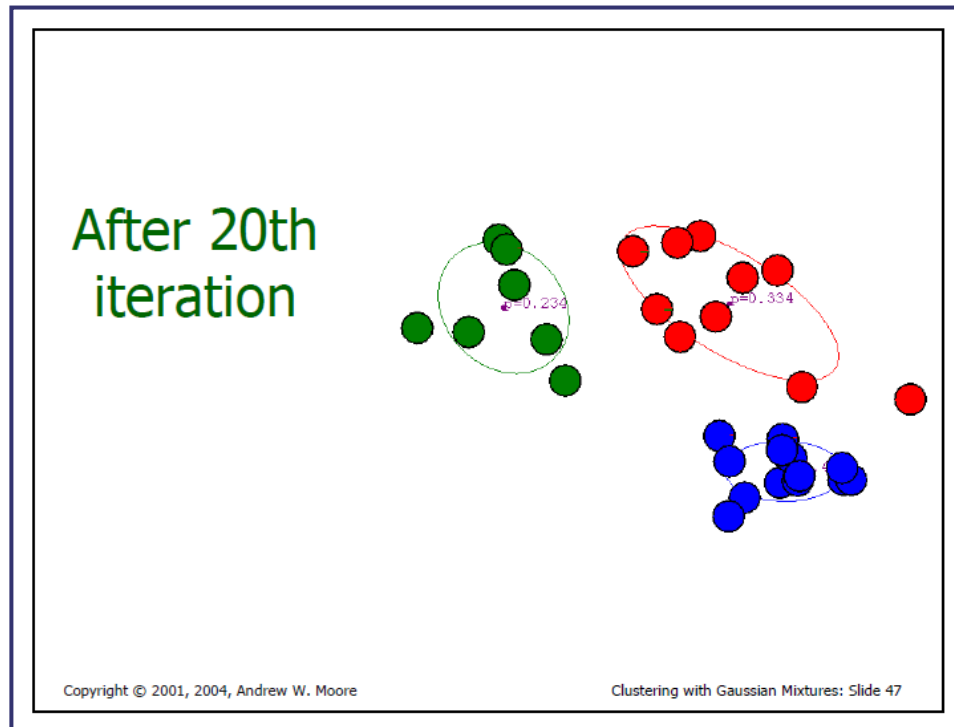
After 5th
iteration



Copyright © 2001, 2004, Andrew W. Moore

Clustering with Gaussian Mixtures: Slide 45

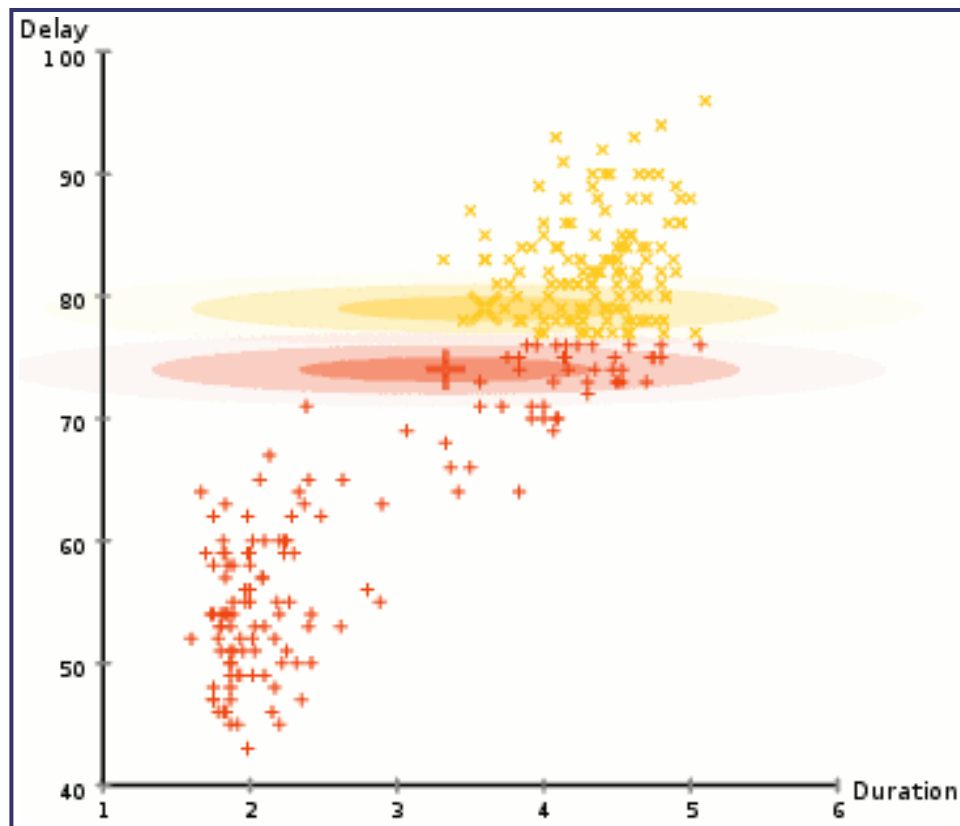
Algorithm: EM (cont'd)



Algorithm: EM (cont'd)


EM is an example of a family of **maximum likelihood estimator algorithms** – others include gradient descent, and conjugate gradient algorithms.

One more example: EM clustering of Yellowstone's Old Faithful eruption data:



Algorithm(s): a clarification

What is the difference between **classification and clustering**? Aren't they the same?



Classification algorithms (when used as learning algorithms) have one ultimate purpose: **given a piece of new data, to place it into one of several pre-existing, LABELED "buckets"** – these labels could be just names/ordinal (eg. ShortPerson, Yes, OakTree..) or value ranges/ordinal (eg. 2.5–3.9, 100,000–250,000).. 




Clustering algorithms on the other hand (again, when used as learning algorithms) **take a new piece of data, and place it into a pre-existing group** – **these groups are UN-LABELED**, ie. don't have names or ranges.




Also, in parameter (feature/attribute) space, each cluster would be distinct from all other clusters, by definition; with classification, just 'gaps' don't need to exist.

(Meta) Algorithm: Ensemble Learning

What if we used a training dataset to train several different algorithms – eg. decision tree, kNN, neural net(s)..? You'll most likely get (slightly!) different results for target prediction.

We could use a voting  scheme, and use the result as the overall output. Eg. for a yes/no classification, we'd return a 'yes' ('no') if we got a 'yes' ('no') majority. 

 This method of combining learners' results is called 'boosting', and resulting combo learner is called an 'ensemble learner'. Why do this? "Wisdom of the crowds"  We do this to minimize/eliminate variances between the learners. 

 FYI – 'AdaBoost' (Adaptive Boosting) is an algorithm for doing ensemble learning  here, the individual learners' weights are  iteratively and adaptively tweaked so as too minimize overall classification errors (starting from a larger number of features used by participating learners to predict outcomes, the iterative training

steps select only those features known to improve the predictive power of the overall model).



FYI – 'Bagging' (bootstrap aggregating) is a data-conditioning-related ensemble algorithm (where we employ 'bootstrap



resampling' of data – divide the data into smaller subsets, use all the subsets for training different 'variations' of a model, use all the resulting models to predict outcomes, transform these into a single 'ensemble' outcome).



Algorithm: Linear regression

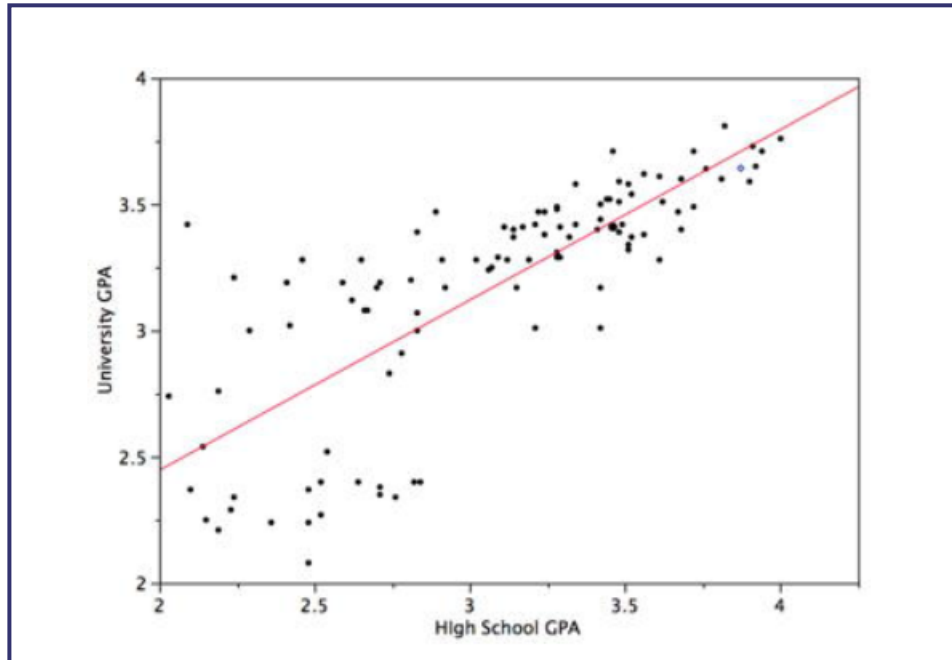


This (mining) technique is straight out of statistics – given a set of training pairs for a feature x and outcome y , fit the best line describing the relationship between x, y . The line describes the relationship/pattern.



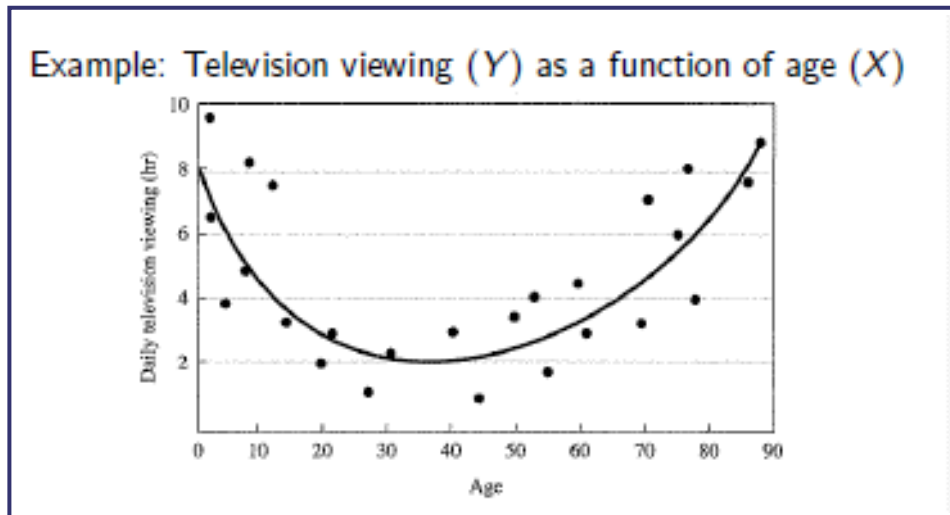
Given that we have Y_i | which depends on $X_{i1}, X_{i2}, X_{i3} \dots X_{ip}$ | (i is the sample index, $1, 2, 3 \dots p$ are the variable indices (dimensions)), model the dependency relationship as $Y_i = f(X_i) + \varepsilon_i$ | where f is the unknown function (that we seek), and ε is the error with mean=0.

Here is an example:



Algorithm: Non-linear regression

Here we fit a higher order polynomial equation (parabola, ie. $ax^2 + bx + c$) to the observed data:



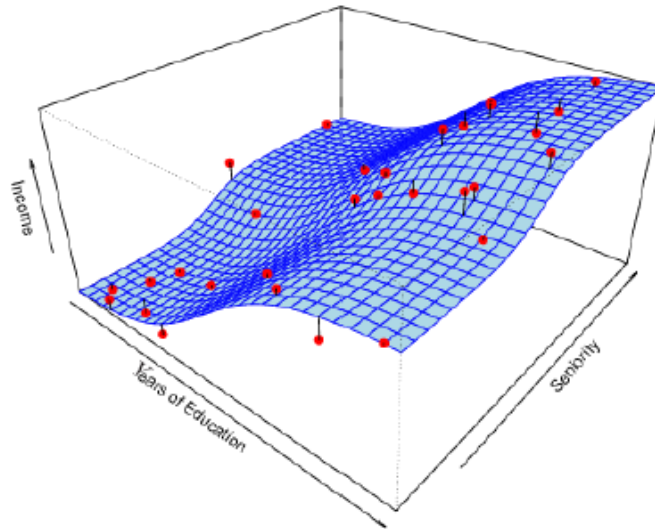


Algorithm: Two parameter, non-linear regression

Here we need to fit a higher order, non-linear surface (ie. non-planar) to the observed data:




Income vs. Education Seniority



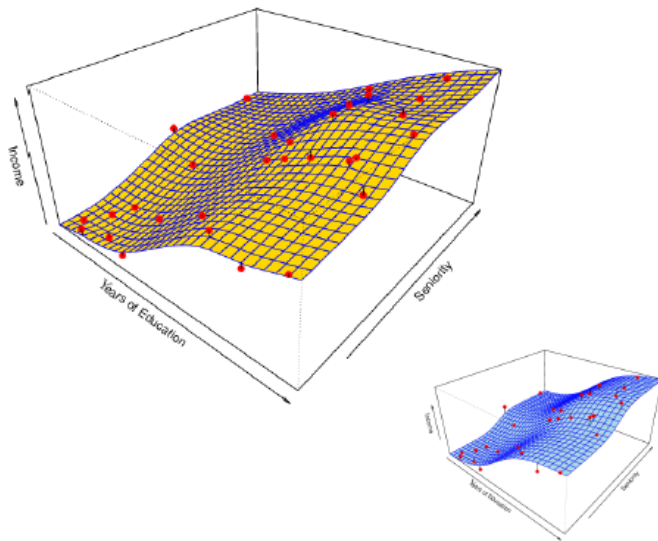
Algorithm: Non-parametric modeling

Parametric methods (eg. regression analysis which we just looked at) involve parameter estimation (regression coefficients).

Non-parametric methods – no assumptions on what our surface (the dependent variable) would look like; we would need much  more data to do the surface fitting, but we don't need the surface to be parameter-based!

Example: A Thin-Plate Spline Estimate

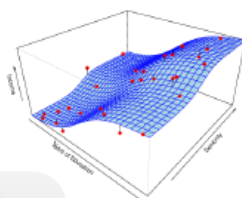
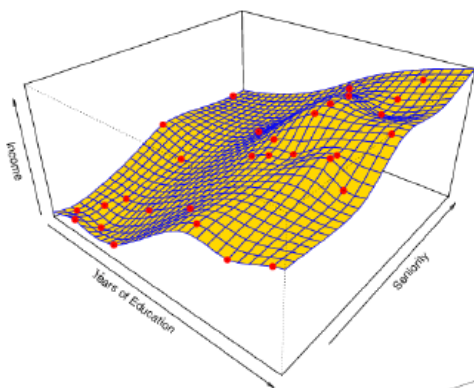
- Non-linear regression methods are more flexible and can potentially provide more accurate estimates.



While non-parametric models might be better suited to certain data distributions, they could lead to a poor estimate as well (if there is over-fit)..



A Poor Estimate

- Non-linear regression methods can also be too flexible and produce poor estimates for f .



Algorithm: Logistic regression

Logistic regression is a **classification** (usually binary) algorithm:

- compute regression coeffs (linear) corresponding to a 'decision boundary' (line dividing two classes of training data) 
- use the derived regression coeffs to compute outcome for new data
- transform the outcome to a logistic regression value, and use the 0..1 result to predict the binary outcome (class A or class B) 

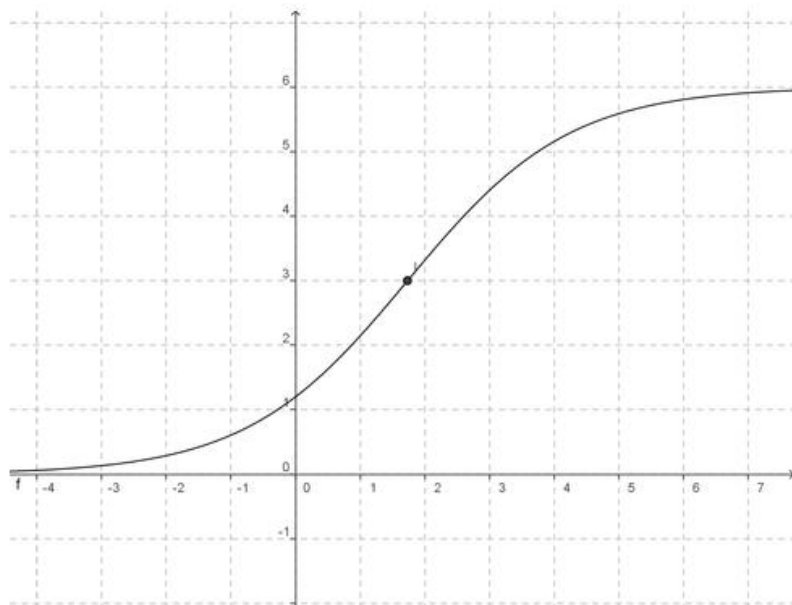
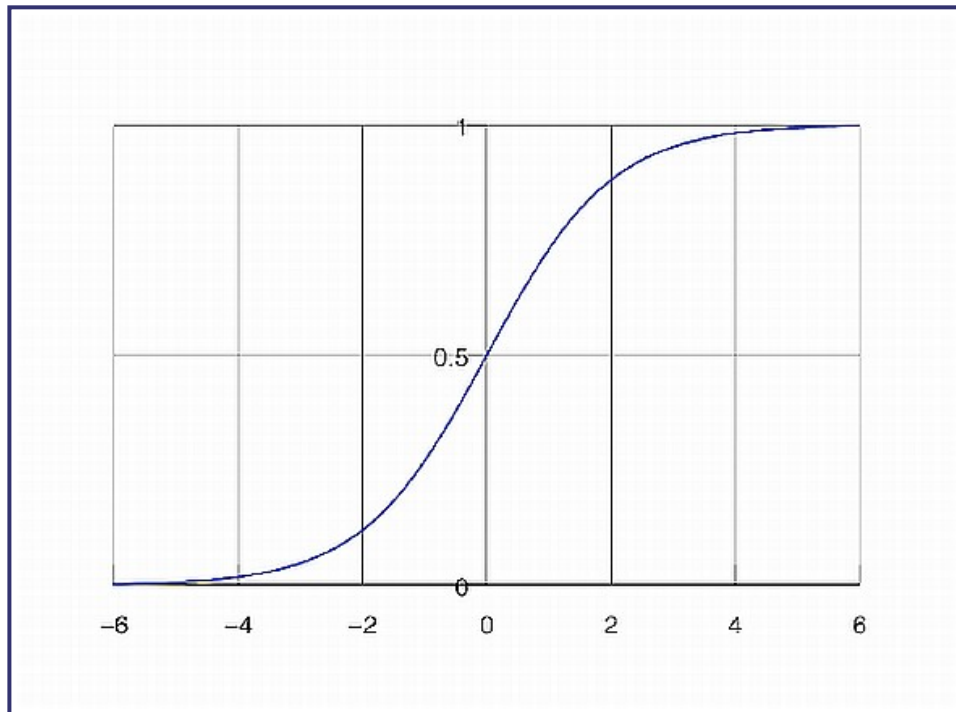


Figure: Logistic function $y = \frac{1}{a + b e^{-x}}$



Points that lie *on* the (linear) decision boundary would yield a value of 0.5, and points on either side would yield values <0.5 (eg. "class A") or >0.5 (eg. "class B").

Result – we are transforming a continuous, regression-derived value into a class (eg. yes or no) (similar to using an SVM or creating two clusters via k Means).

Uses?

Algorithm: Naive Bayes

The 'naive' Bayes probability-based, supervised, classifier algorithm (given $x_1, x_2, x_3 \dots x_n$ features, classify the data point to be one of 1,2,3...k classes).

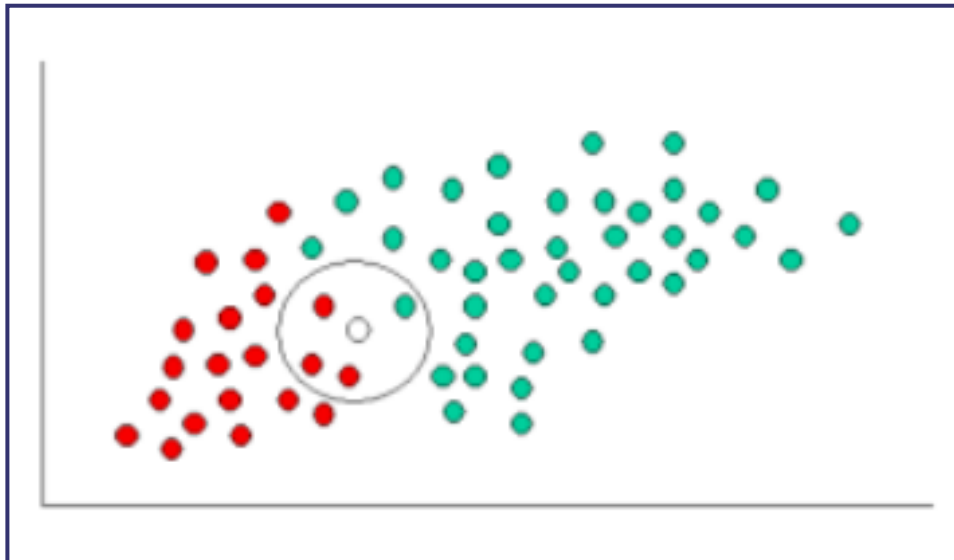
The algorithm is so called because of its strong ('naive') assumption: the $x_1 \mid \dots x_n$ features are statistically independent – eg. a fruit is an apple if it is red, round and ~4 in. dia.

Other names for this algorithm: simple Bayes, independence Bayes, idiot Bayes (!).

For each training feature set, probabilities are assigned for each possible outcome (class). Given a new feature, the algorithm outputs a classification corresponding to the max of the most probable value of each class (which the algorithm calculates, using the 'maximum a posteriori', or 'MAP' decision rule).

Here is an example (from <http://www.statsoft.com/textbook/naive-bayes-classifier>).

Given the following distribution of 20 red and 40 green balls (ie. 60 samples, 2 classes), how to classify the white one ('X')?



Prior probability of green, of red:

$$\text{Prior probability for GREEN} \propto \frac{40}{60}$$

$$\text{Prior probability for RED} \propto \frac{20}{60}$$



Probability of X **given** green, X **given** red: 

$$\text{Probability of } X \text{ given GREEN} \propto \frac{1}{40}$$

$$\text{Probability of } X \text{ given RED} \propto \frac{3}{20}$$

Probability of X being green, X being red:



Posterior probability of X being GREEN \propto

Prior probability of GREEN \times Likelihood of X given GREEN

$$= \frac{4}{6} \times \frac{1}{40} = \frac{1}{60}$$

Posterior probability of X being RED \propto

Prior probability of RED \times Likelihood of X given RED

$$= \frac{2}{6} \times \frac{3}{20} = \frac{1}{20}$$



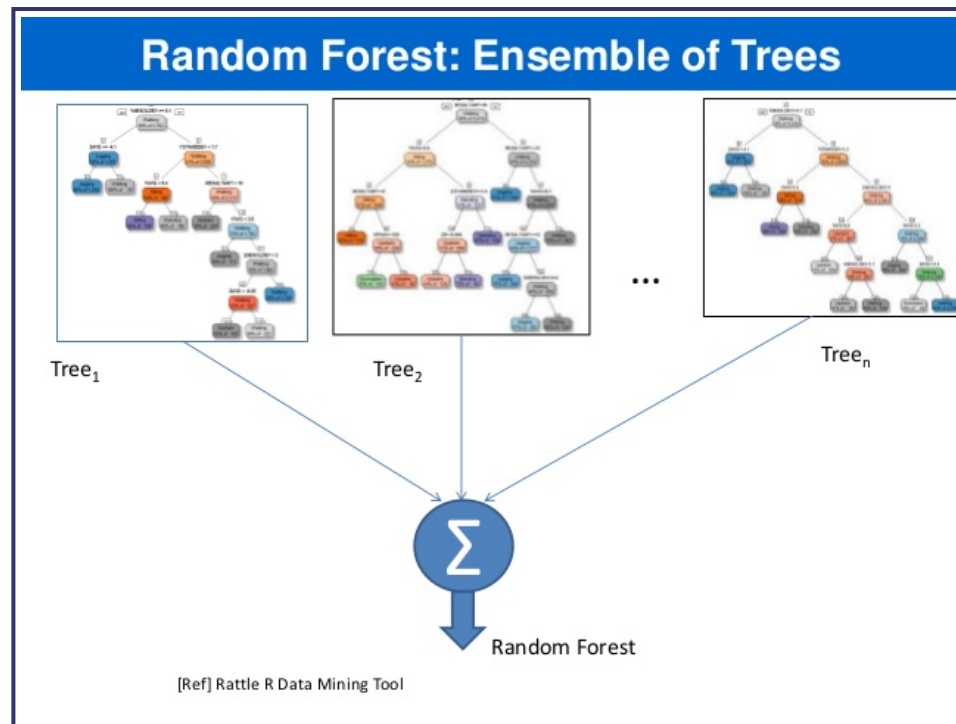
Classify, using the max of the two above: X is classified as 'red'.

Algorithm: RandomForest (TM)




RandomForest(TM) is an **ensemble** method where we:


- grow a 'forest' (eg. with count=500) decision trees, run our new feature through all of them, then use a voting or averaging scheme to derive an ensemble classification result
- keep each tree small – use \sqrt{k} features for it, chosen randomly from the overall 'k' samples

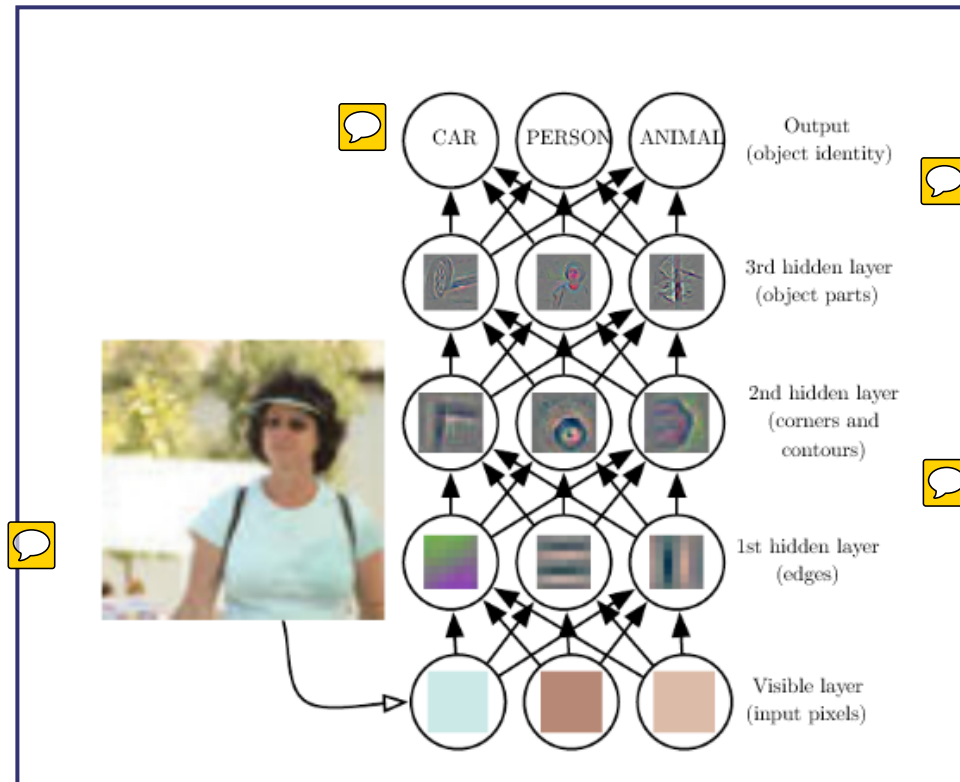


NN++ : Deep Learning!

 As mentioned in the previous class Deep Learning is starting to yield spectacular results, to what were once considered intractable problems.. 

Why now? Massive amounts of learnable data, massive storage, massive computing power, advances in ML.. [Here](#) is NVIDIA's response (to 'why now').. 

In Deep Learning, we have large numbers (even 1000!) of hidden  layers, each of which learns/processes a single feature. Eg. here is a (non-so-deep) NN:



DNN: who's doing it??

- Google: self-driving cars, TensorFlow, DeepMind
- IBM: Watson, AlchemyAPI, Watson Analytics
- Microsoft: ImageNet entry, CNTK
- Facebook: DeepFace can search 800M faces in <5 sec! Also, Facebook is planning to open source its hardware setup.

CNN

Convolutional Neural Networks (CNNs, aka ConvoNets) are biologically inspired – (convo) filters are used across a whole layer, to enable the entire layer as a whole to detect a feature. Detection regions are overlapped, like with cells in the eye.

[Here](#) is an **excellent** talk on CNNs/DNNs, by Facebook's LeCun.

[Here](#) is a **great** page, with plenty of posts on NNs – with lots of explanatory diagrams.

DNN: on GPUs!

GPUs (multi-core, high-performance graphics chips made by NVIDIA etc.) and DNNs seem to be a match made in heaven! 

NVIDIA has made available a **LOT** of resources related to DNNs using GPUs, including a framework called DIGITS (Deep Learning GPU Training System). NVIDIA's **DGX-1** is a deep learning platform built atop their Tesla P100 GPUs. **Here** is an excellent intro' to deep learning – a series of posts. **Here** is a GPU-powered self-driving car (with 'only' 37 million neurons) :)

Microsoft has created a GPU-based network for doing face recognition, speech recognition, etc.

IBM has its SyNAPSE chip, and **TrueNorth** NN chip.

FPGAs also offer a **custom path** to DNN creation.

Also: TeraDeep, CEVA, Synopsis, Alluviate..

