IAML Condensed Summary Notes For Quick In-Exam Strategic Fact Deployment

Maksymilian Mozolewski

December 10, 2020

Contents

IAML

Introduction

Machine Learning

A machine learning model **takes in** data, **outputs** predictions. It's a function of data really together with a set of training data.
Learning = Representation + Evaluation + Optimisation

Thinking about data

Classification

Sort data points into discrete buckets based on training data

Regression

Output a continuous/real value for each data point based on training data.

Clustering

Detect which data points are related to which other data points, find outliers.

Data representation

What format do we feed the data in ? Most likely as a **bag of features**. I.e. collection of attribute-value pairs, every data point must have an attribute-value pair for each property (in most cases)
Data representation has more impact on the performance of your ML algorithm than anything.

Types of attributes

- **Categorical**
  - e.g. red/blue/brown
  - a set of possible **mutually exclusive** values
  - meaningful operators: equality comparison
  - usually represented as numbers
  - problems: synnonymy is a major challenge e.g. some values might mean the same thing to a human but not to the machine (country == folk?)
- **Ordinal**
  - e.g. poor < satisfactory < good < excellent
  - a set of possible **mutually exclusive** values, but with a **natural ordering**
  - meaningful operators: equality comparison, sorting
  - problems: sometimes hard to differentiate from categorical (single < divorced)?
- **Numeric**
  - e.g. 3.1/5
  - meaningful operators: arithmetic, distance metrics, equality, sorting
  - problems:
    - sensitive to extreme outliers (handle these **before normalization**)
    - skewed distributions (assymetric) - outliers might actually be real data (e.g. personal wealth data)
    - Non-monotonic effect of attributes - e.g. predicting someone is going to win a marathon, here the relationship is not monotonic i.e. norect correlatio din, might be a curve with a "sweet spot"
  - solutions:
    - *Deal with outliers, maybe trim them for training phase only?*
    - *use a log/atan scale to make data more linear*
    - *discretize data into buckets*

Normalisation

Normalization is the process of converting all the data such that each different attribute is roughly in the same range, and comparable.
Normalization is mostly necessary for linear methods.

Picking attributes

We want:

- all our attributes to have similar values if the data points that posses them are similar themselves!
- small change in input $\rightarrow$ small change in values

Images

For images, can we use pixel data as attributes directly ? it depends, if the pixel 20,20 always corresponds to the middle of a letter then yes, this is a meaningful attribute which might help us discern whether digits given have strokes going through the middle of the image! What if the pixel is always something random ? This could happen whenever we are looking for an object which can be anywhere in the image, in which case the attribute would be gibberish! In the case of image classification, we want attributes which are:

- invariant to size
- invariant to rotation
- invariant to illumination

How can we classify whether an image contains a desired object ? In general we do the following:

- **Segment** the image, into regions of pixels which we believe are related (by colour, texture, position etc..)
- Pick a number of attributes which you will use to describe each of the regions
- Then use those features in your machine learning algorithm!

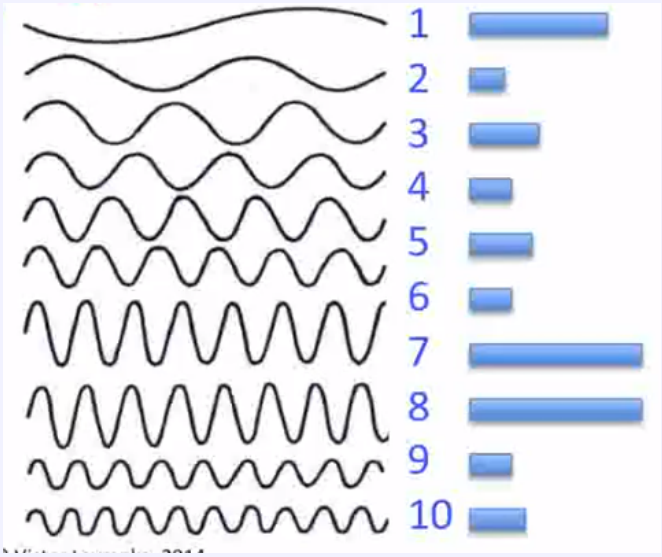
Keep in mind though, the segmentation **will** make errors, we can hope that these will be consistent across all images (all images will have their legs chopped off). Sometimes we can segment using a grid

Text

For textual data, often we can use the **bag-of-words** approach. I.e. we can form an attribute vector which counts the amount of occurrences of each word, regardless of position. This is invariant to word shuffling for example.

Sound

For sound, the data is sound waves. How can we select attributes here? We can count the number of different frequencies occurring in the piece, (using Fourier's transform) and treat this as a feature vector!



Supervised Learning

Supervised learning algorithms have some sort of "performance" metric they can use, i.e. test labels they can validate their guesses on. When the algorithm can measure accuracy directly it's a supervised algorithm.

Unsupervised Learning

Learning without a specific accuracy measure available. Algorithms in this area usually look for structure/patterns/information in the data which can be helpful in other ways. There is nothing specific the algorithm is looking for. Can be **direct** when the algorithm helps to make sense of the data directly, or **indirect** when it is "plugged" into another machine learning aglorithm as an attribute itself.

Semi-supervised Learning

Using unsupervised methods to improve supervised algorithms. Usually have a few labelled examples but lots more unlabelled.

Multi-class classification

Classification with multiple mutually exclusive labels/classes.
Might be hard to tell when something belongs to none of the available classes.

Binary classification

Classification with 2 mutually exclusive labels/classes in each "run". This way of classification can be applied to multiple-classes classification but with a "One-vs-Rest" meta-strategy (a vs not a, b vs not b). In this way a sample may belong to multiple classes but never to two sides of the one-vs-rest structure simultaneously in each run.
In this classification method we can actually tell when something doesn't belong to any class!

Analysing data

We have to check for a number of things in our data sets:

- Are there any dominating classes ? what would the best "dummy" model do ? always predicting no ?
- What should we use as the appropriate error metric ? How important are the false positives vs the false negatives ?

Generative model

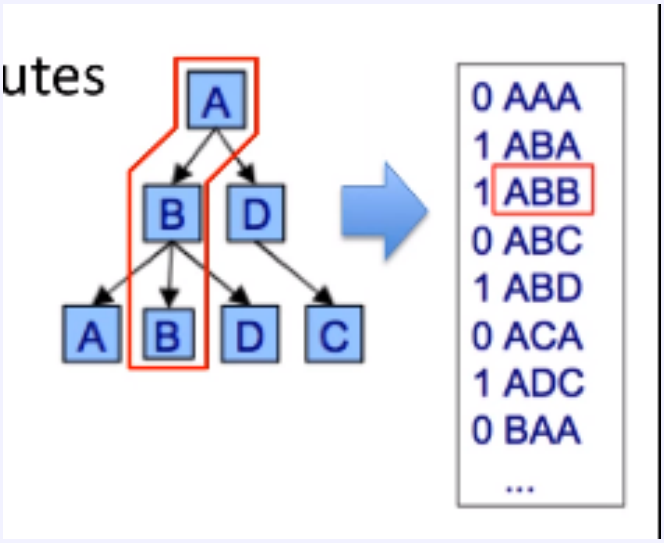
A generative model, develops a probabilistic model of each class, i.e. tries to "model" the underlying probability distribution directly. The decision boundary becomes implicitly defined by the probabilities of each input being in each class.

Discriminative model

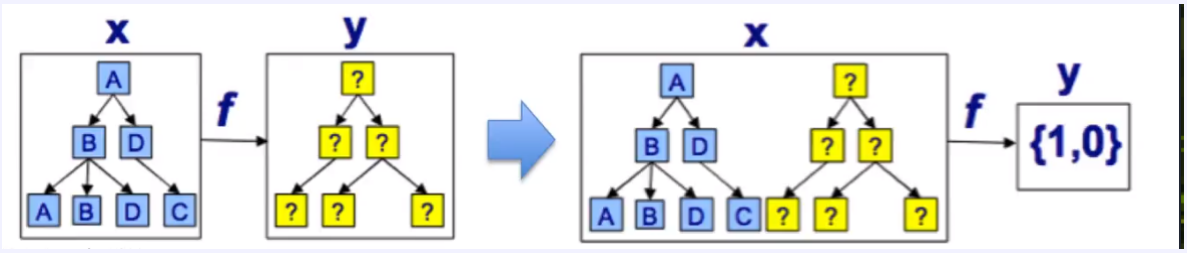
A discriminative model ignores the underlying model and tries to "separate" the data, i.e. it tries to model the boundaries that divide the classes. **Not designed to use unlabeled data** so cannot be used for unsupervised tasks.

Dealing with data structure

What do we do if the data input has some sort of hierarchical structure ? Where the position of occurrence of a node affects its meaning? We can encode as attributes the existence of root-to-leaf paths in the entire tree, and use this bag-of-words approach to perform machine learning



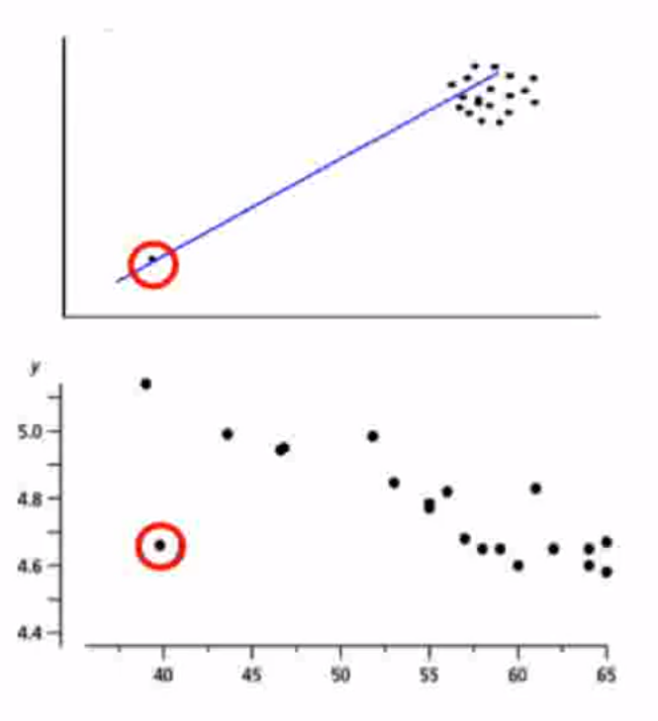
What if we need to predict the output structure from the input structure ? This is very difficult, but we can "trick" our classifier and turn this more into a search problem by embedding the possible outputs with each input and classifying on that instead:



This of course means we have to search for all possible output structures!

Dealing with outliers

Outliers are isolated instances of a class that are unlike any other instance of that class. These affect all learning models to some degree.
There are some ways we can deal with outliers. One method is to remove the outliers just before we perform any sort of normalisation on the data, (ONLY FOR THE PURPOSES OF TRAINING!!) We can also put a confidence interval around our data, and removing values outside of those intervals (with x,y values outside of a normal range). Some data points might still be outliers even though they are within expected x,y ranges! (second figure)



Best way to deal with outliers ? **VISUALISE YOUR DATA**

Naive Bayes

Bayes rule

$$P(y|x) = \frac{P(x|y)P(y)}{\sum_{y'} P(x|y')P(y')} \tag{1}$$

where:

- y : one of the classes
- x : the input data, feature vector
- P(y) : **prior**, the probability of seeing elements of class y in general prior to making any observations
- $P(x|y)$: **class model/likelihood**, given the data is in class y, how likely are the features that i am seeing
- P(x) $= \sum_{y'} P(x|y')P(y')$: **normalizer**, does not affect the probabilities, but without this term we cannot compare data in terms of probability of being in each class (i.e. for confidence values)

Note: this works with probability densities too (which we're using almost exclusively)

Naive bayes

Bayesian classifier, which assumes **conditional independence** between different attributes. It attempts to model the underlying probability distribution so it's a **generative** model.
I.e.:

$$P(\mathbf{x}|y) = \prod_{i=1}^{d} P(x_i|x_1 \dots x_{i-1}, y) = P(x_1|y) \cdot P(x_2|y) \cdot P(x_3|y) \dots \tag{2}$$

It assumes that there are no correlations between the variables themselves, and any correlations are explained by the fact they belong to the same class.

Example

The probability of going to the beach and getting a heat stroke are not independent: $P(B, S) > P(B)P(S)$. The may be independent if we know the weather is hot (i.e. external factor is what actually causes the dependence, which we can factor into the equation): $P(B, S|H) = P(B|H)P(S|H)$ This hidden factor in naive bayes is **the class**

Gaussian Naive bayes

A Naive Bayes classifier which uses the gaussian distribution as a class model for each class.

$$p(x|y) = \frac{1}{\sqrt{2\pi\sigma_{x,y}^2}} \exp^{-\frac{1}{2}\frac{(x-\mu x,y)^2}{\sigma_{x,y}^2}} \tag{3}$$

With:

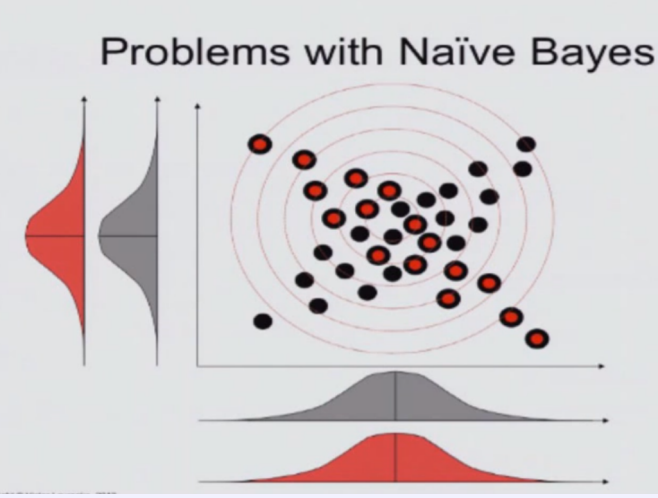
- $\mu_{x,y}$: the mean of the gaussian modelling class y's attribute x
- $\sigma_{x,y}^2$: variance of the gaussian modelling class y's attribute x
- x : the value of the attribute
- y : the class

Note: this is not actually a probability per-se, this is a p.d.f function which can be higher than 1

Problems with Naive Bayes

Covariance

Naive bayes cannot model covariance of the attributes, i.e.:



in the above case the only thing differentiating the classes is their relationship between x and y, i.e. the two attributes. Naive bayes can only model **spherical** distributions

Zero frequency problem

Should any data never appear under any of the classes the probability of it belonging to that class is always going to be zero. So say we are classifying emails as spam or non-spam, and the word "now" never occurred in non-spam emails, but had occurred in some spam emails. Any sentence containing the word "now" would always be classified as spam no matter the actual probability of it being spam!
Solution, add a small epsilon to all "feature counts", or perform laplace smoothing:

$$P(w,c) = \frac{num(w,c) + \epsilon}{num(c) + 2\epsilon} \tag{4}$$

Zipf's law: 50% of words occur once

Independence assumption

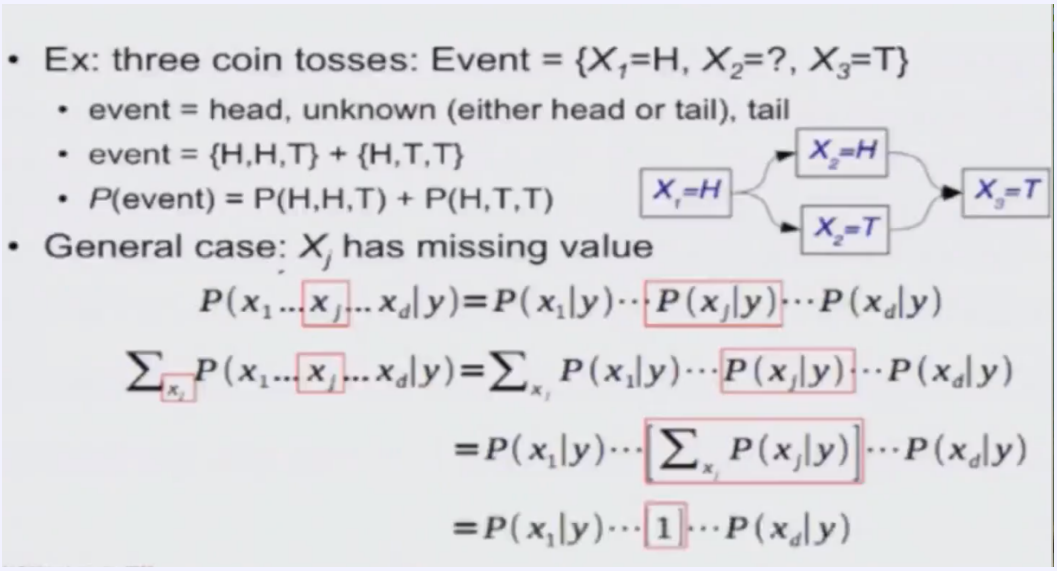
Continuing with the spam example, every word contributes to the total probability independently, easy to fool. Simply stuff lots of non-spammy words into email

Deaing with missing data

Suppose we have missing data for some attribute i of a sample x. How can we compute the likelihood of this sample belonging to any class ?
We simply ignore it:

$$P(x_1 \dots x_i \dots x_d|y) = \prod_{k \neq i}^{d} P(x_i|y) \tag{5}$$

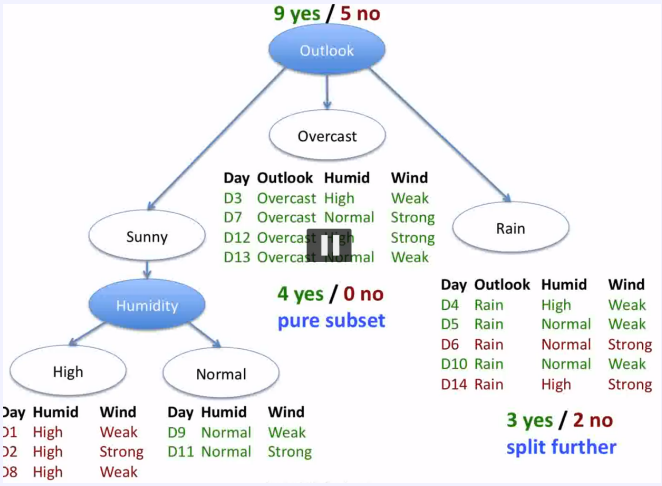
There is no need to fill in the gaps artificially because missing attribute data essentially has a probability of 1:



Decision Trees

Method

Decision trees split the data based on the label of each data point, in a way which maximizes the entropy of each split. This means we are splitting on attributes which have the most "discriminatory" power at each step



We classify each new sample by following the splits in the tree untill we hit a pure subset (or non-pure), and select the label based on the most frequent label in that leaf.

Quinlan's ID3 Algorithm

- A ¡- best attribute for splitting (**needs a heuristic**)
- Decision attribute for this node ¡- A
- For each value of A, create new child node
- Split training data at this node into child nodes
- For each child node/subset
  if subset is **pure**, stop
  otherwise repeat on this subnode

Purity measures

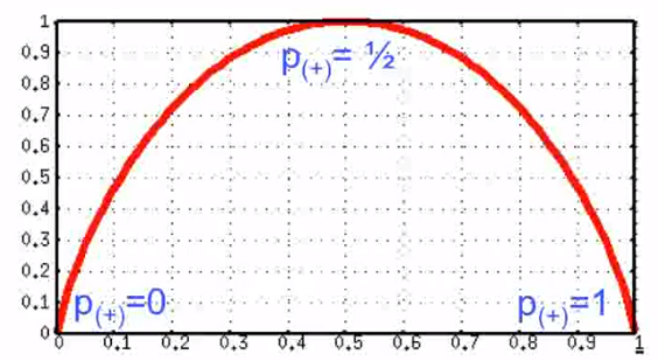
In order to pick the best attribute we need a purity measure
Even splits do not give us any information, we want ones biased towards the +'ve or -'ve labels
i.e. the purity for a pure set (4y/0n) is 100% but the purity for an impure set (3y/3n) is 50% (uncertain)

Entropy

$$H(S) = -p_+ \log_2 p_+ - p_- \log_2 p_- \tag{6}$$

where:

- S is the subset of training examples
- p's refer to the proportion of positive and negatives in the subset S



we interpret this measure as the number of bits of information needed to tell if a random item in S is a + or a -, we dont need any information at all if the subset is entirely composed of +'s or -'s, and all the information if it's evenly split

Information gain

once we have a measure of purity, we can establish the heuristic which will define which split will be the best:

$$Gain(S, A) = H(S) - \sum_{V \in Values(A)} \frac{|S_v|}{|S|} H(S_v) \tag{7}$$

where:

- **S** : subset of data points
- **A** : attribute we are splitting on
- **V** : possible values of attribute A
- S_v : subset of S with data points which have label v

This measures the "information" we gain by performing the split. Notice how we take a weighted average (weighted by the number of data points) of the entropy of each new subset formed in the split. If the entropy in the new subsets is on average higher than the entropy in the original subset, we have lost information (note this is not possible in this scenario) and if it's lower, we have indeed gained information. The split which maximises this quantity is the best.

Gain ratio

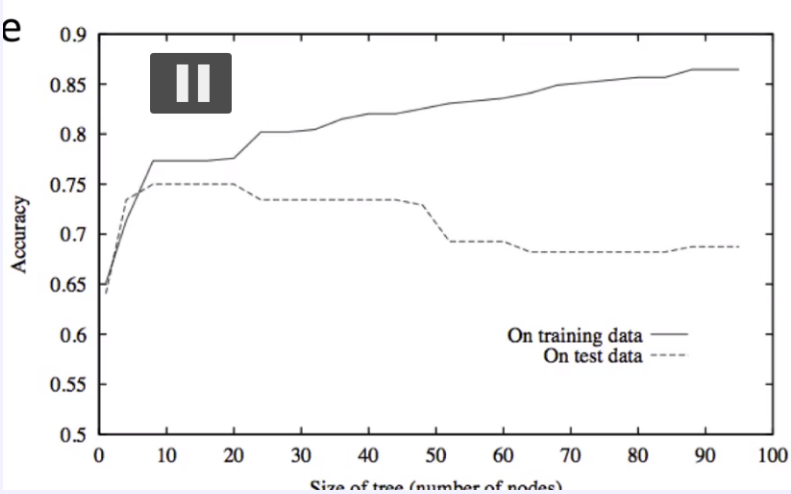
Using this heuristic will lead the algorithm to pick splits which lead to more subsets naturally (if allowed by shape of attributes and data) We can use the entropy of the split itself to counteract this instead:

$$\begin{aligned} SplitEntropy(S, A) &= -\sum_{V \in Values(A)} \frac{|S_v|}{|S|} log \frac{|S_v|}{|S|} \\ GainRatio(S, A) &= \frac{Gain(S, A)}{SplitEntropy(S, A)} \end{aligned} \tag{8}$$

not how the gain will be lower when we have too many or too few subsets, the split ratio will be the highest with higher information gains which result in 2 subsets that each take half of the data.

Overfitting

Overfitting happens when the model learns the "noise" in the training data, and becomes too specific.
Decision trees will always classify training examples perfectly if we make them big enough (because eventually we will split the data into singletons, and those instances will always be labelled as the correct instances)
This is an example of **overfitting**, we need to limit the size of the tree in order to counteract this, i.e. force some subsets to end up being non-pure in order to stop the model from becoming too specific to our training data.



Solutions

- Stop splitting when not statistically significant
- Grow the tree fully and then prune in a way which maximises the accuracy (using a validation set)

  - **Sub-tree replacement pruning**: pretend you remove a node and all its children, measure the performance on validation set. Remove node with highest performance gain, repeat untill removal is harmful to performance(greedy)

Numeric attributes

How can we apply this to numeric attributes? Split based on **thresholds**, threshold can be picked via some process. Easy solution is to take all the values for the attribute we are evaluating a split on, sort them and go through the split on each of those one-by-one.

Multi-class classification

We can apply decision trees to multi-class classification by predicting the most frequent class in the subset we land on. The entropy calculation becomes:

$$H(S) = -\sum_{c} p_c \log_2 p_c \tag{9}$$

where p_c is the fraction of samples of class c in the subset

Regression

We can also apply this algorithm to regression by taking the average of the training examples in the subset we land on as the predicted output value. In which case instead of maximising gain we grow the tree by minimising variance in the subsets. (pick split which reduces variance in the output the most!) We can also replace the average by linear regression at the leaves.

Random decision forest

- Pick a random subset S_r of training examples
- grow a full tree T_r (without pruning)
- when splitting pick from $d << D$ random attributes (i.e. hide some attributes for some trees)
- compute gain based on S_r instead of full set
- repeat K times to make K trees

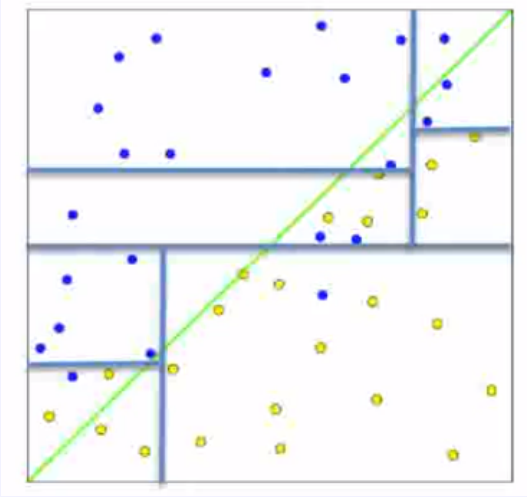
Given a new data point X we can then:

- classify X using each of the trees $T_1 \dots T_k$
- use majority vote: class predicted most often as the label!

State-of-the-art performance in many domains, very good stuff

Evaluation of decision trees

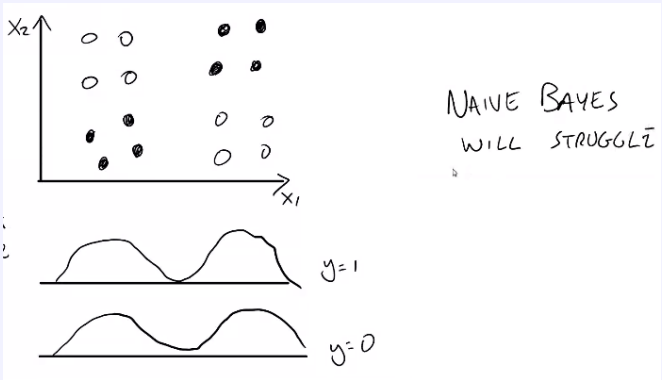
+ **interpretable** - humans can understand the underlying decisions! (ONLY ALGORITHM IN THIS COURSE TO HAVE THIS PROPERTY)
+ easilly handles irrelevant attributes (gain = 0)
+ can handle missing data
+ very compact: nodes << D after pruning
+ very fast at testing time O(depth)
- only axis-aligned splits are possible (because we split only taking into account one dimension at a step)



- greedy (may not find best tree globally)
- exponentially many possible trees

Why trees?

Can model non-linear, weirdly shaped data which does not realy follow a nice probability distribution (i.e. **non-monotonic** data!)



Generalisation & Evaluation

Linear regression

Logistic regression

Optimisation & Regularisation

Support Vector Machines

Ethics

Nearest Neighbours

K-Means

Gaussian mixture models

Principal components analysis

Hierarchical Clustering

Perceptrons

Neural networks