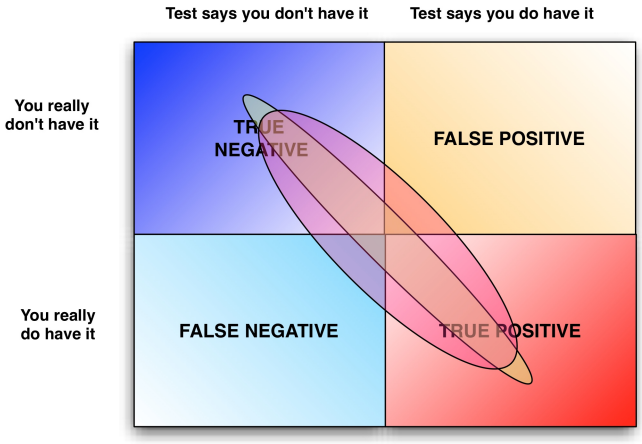
- Categorical data: ordinal, nominal and binary.
- Generative classifiers. Steps in modeling: model selection; density estimation of the data; classification of new data.
- **Bayesian Theorem**. $p(c|x) = \frac{p(c,x)}{p(x)} = \frac{p(x|c)p(c)}{p(x)}$. c: the model to be inferred. x: the observations. $p(x|c)$: the likelihood. $p(c)$ the prior. $p(c|x)$ the posterior. $p(x)$ the evidence.
- Something more about Bayes. $p(c|x)$ means the probability of c given x. Take a concrete example. $p(L|W) = 0.75$ (from Wikipedia)—the probability of a woman with long hair is 75%. Or rather, the probability of event L given event W is 0.75.
- Even something more. $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$. $P(A)$ the prior, the initial degree of belief in A. $P(A|B)$ the posterior, the degree of belief having accounted for B.
- **Bayes decision rule**. $p(x)$ is independent of the class and $p(c)$ is frequently assumed to be the *same* for all classes.
- Standard *k-NN*. When we use big k, then we could have a better noise resistance. At the same time we have worse resolution.
- Kernel density estimation. When we increase the number of kernels, better smoothness is obtained. But we will have a much higher dimension. (You know what this implies.)
- Calculate the joint probability distribution is normally very difficult, but Naive Bayesian method solves this by assuming that class conditional distributions are all independent. $p(c_i|x) = \frac{p(x_1,x_2,\ldots,x_n|c_i)p(c_i)}{p(x)} = \frac{p(x_1|c_i)p(x_2|c_i)\ldots p(x_n|c_i)p(c_i)}{p(x)}$.
- Kind note about Naive Bayesian classifier. The evidence $p(x)$ is class-independent, which means it has nothing to do with, the variable. Aka, the stuff that we *should* really look into.
- **Multivariate Bernoulli** model. For this model, one weakness is that whether a word (in Anti-Spamming case) appears 1 or 100 times, the final probability representation is the same. (Let's do it in Chinese. It's NOT scientific.)
- **Multinomial** event model. For this model, each "word" (in spamming example) is an event. One word appears, then it's probability is calculated. Appears many times, then times as many.
- **Naive Bayesian Classifier for Continuous Data**. When we try to extend NBC to classify data with continuous data, there is one "best practice" that we should think about. The basic idea is to divide each feature into *two or more* bins.
- Equal Frequency Discretization. In this method, each bin contains the same amount of points. (If you do want to "imagine" what point is like, in hyper space.) It's analogous to *k-NN* approach.
- **Fuzzy Discretization**. Form k equally spaced bins; the corresponding likelihood includes contributions from *every training instance*.
- **Lazy Discretization**. The determination of $p(x|c)$ is postponed. When query x is presented, place it at *center* of the bin. The bin scale thus equals to $[x-\delta, x+\delta]$. $p(x|c)$ is then proportional to the number of instances within the specific scale.
- Non-disjoint discretization. Bins are set in advance, but are overlapping. The actual bin for a query point is $i-1$, i and $i+1$. **Query point is always close to the center**.
- To avoid overfitting, as top scientists and engineers, we always prefer simpler models than complex ones.
- Type 1 and Type 2 errors. A type 1 error is the incorrect rejection of a true null hypothesis. It is a false positive. Usually a type 1 error leads one to conclude that a supposed effect or relationship exists *when in fact it doesn't*.



- $(1,1)$ True positive. $(1,0)$ False negative. $(0,1)$ False positive. $(0,0)$ True negative. The first element 1 means *you do have it*.
- False positive. A result indicating that a given condition is present when it actually it not.
- False negative. It's failing to assert what is present. It could also be regarded where a test result indicates that a condition failed, *while actually it was successful*.
- One word to describe False Negative? Miss Jessie.
- Precision: $\frac{TP}{TP+FP}$. Recall: $\frac{TP}{TP+FN}$.
- When we say that we are encouraging *overfitting*, we are saying the "generalization" of the models we build, sucks.
- Another explanation. Precision: $\frac{\sum True\ Positive}{\sum Test\ Outcome\ Positive}$ Recall: $\frac{\sum True\ Positive}{\sum Condition\ Positive}$
- Data Mining methods are always intrinsic probabilistic, why? One explanation given by Dr. Woon is that real world data *is always noisy*.
- **Top down inductive tree**. Select a feature to split on; Sort examples into subsets based on the values of feature, one for each *value*; Branch the tree by creating new nodes (aka, new subtrees) foe each subset; *recurse* until a complete tree is obtained.

- **Entropy**. In Information Theory, we define entropy as follows: $H(x) = -\sum_{i=1}^{n} p(x_i) \ln p(x_i)$
- Information Gain. In general terms, the expected information gain is the *change* in information entropy from **a prior state** to a state that takes some information as given: $IG(T, a) = H(T) - H(T|a)$.
- Gain Ratio. Normalize information gain with respect to *the number of values* that a feature can take.
- Gini impurity index. It's the probability that a randomly selected instance is wrongly classified based on a label randomly sampled from that subset. $I_G(f) = \sum_1^m f_i(1 - f_i)$.
- Post pruning. Each node to which *only leaves are attached* is considered for pruning. The idea is to evaluate combined (*weighted*) error rate and compare that of the father node. Core idea is that a compact tree with *good* prediction.
- Decision trees with continuous variables. Treat N values in the training set as N separate features, and choose the split point with highest *Information Gain* or any other criterion that's reasonable. Split between points w. different labels.
- In decision tree classification, there are usually two ways to avoid overfitting: limit tree depth; reduced error pruning.
- Decision Tree. Disadvantages: learning process is heuristic thus often results in overfitting; closeness to boundary, confidence intervals... are ignored.
- **K-means Clustering**. Generate k initial cluster centroids; Assign each point to the closest centroid; recompute the location of each centroid using existing class memberships; iterate untill convergence criterion is met.
- Dunn Index. $DI(c) = \min_{i,j \in c: i \neq j} \{\frac{\delta(A_i, A_j)}{\max_{k \in c}(\triangle(A_k))}\}$. We want to see a larger DI when build clusters.
- Cluster Quality metrics. As with clustering algorithms, we want to see **compact** clusters and long **distances** between each cluster. For groups of the same interest, get as close as possible. And vice versa. Eg: $C = \frac{S - S_{\min}}{S_{\max} - S_{\min}}$. Smaller!
- K-centers clustering has a better outlier resistance.
- **K-means** uses a two-phase iterative algorithm. #1, *batch updates*. Reassigning each point to it's nearest cluster centroid; recalculation of cluster centroids. #2, *online updates*. Points are individually reassigned if doing so will reduce the sum of distances and cluster centroids are recomputed after each assignment. Also called *K-center*.
- **PCA** generates a new set of components, called *principle components*. Each is a linear combination of original variables. All are orthogonal to each other, so there is no redundant information. The first principle component is *a single axis* in space, when you project each observation on that axis, the resulting values form a new variable. And the variance of this variable is the maximum among choices of the first axis.
- Self-Organizing Map. Strategy: Embed a string of markers or nodes along the axis; optimize the position of nodes so that each approximates position of nearby nodes; projection, points attached to closest node.
- **SOM**. Weight update equation: $V^{t+1} = V^t + \eta^t \Theta^t (X^t - V^t)$. V^t, position of the node at time t. η^t, learning rate. X^t, input vector selected at time t. In SOM, it first learn the topological structure of the data; smaller η^t and Θ^t values allow "fine-tuning" so that nodes match data distribution.
- UPGMA. Input: Distance matrix between all pairs of node. *In fact this is yet another "greedy" algorithm.* Steps: combine nearest pairs of nodes; recalculate new distances between all nodes and clustering ($\frac{1}{|A||B|} \cdot \sum_{x \in A} \sum_{y \in B} d(x, y)$). The formula requires to calculate average distance between all pairs of points in two groups (clusters); generate new distance matrix. In another algorithm called neighbor joining, it tries to minimize the *total branch length in the tree.*
- Unsupervised algorithms tend to be more descriptive in nature rather than prescriptive, also more "exploratory".
- A simple neuron model. $Output = f(a) = f(\sum_{i=1}^{n} w_i \cdot x_i + b)$. x_i, input vector. w_i, weight vector. b, bias term to adjust threshold.
- **Perceptron**. In this simplest class of feedforward network, function $f()$ is a *step function.* Input weights $w_1, w_2, \ldots, w_n$ defines a direction in the input space. The primitive perceptron **only works** with data that are *linearly separable.*
- **Gradient Descent** is an iterative parameter optimization technique. Steps: Initial value x_0; update x_n at each iteration so the gradient *is* descending. Update is the form: $x(n+1) = x(n) - \frac{\eta dE(x)}{dx}$. Latter is called **update term**.
- Gradient descent with momentum. Modified update equation: $w(n+1) = w(n) - \triangle'_w E$, and the error equation is $\triangle'_w E = \frac{\eta dE(x)}{dx} - \mu(w(n) - w(n-1))$. Trans: a portion of previous update term is added to the current one at *each step.* This technique helps "overshoot" local minimal locations, and also helps enforce consistent update directions.
- Non-disjoint discretization would possibly help most if you insist on sticking to histogram approximation of some data.
- Deleting nodes where information gain fails to exceed a particular threshold **will not help prune the tree** built.
- **Perceptron Classification**. In Dr. Woon's tricks, x axis is usually named x_1, and y axis named x_2. Remembering this, the "transfer function", i.e. the linear expression of $f(x)$ using x_1, x_2 and w_1, w_2 would *never* be difficult to understand. One more thing, this $f(x)$ function is used to "predict" the output, whereas the **decision boundary** is perpendicular to $f(x)$. Mathematically, *slope of the decision boundary* is $(-1 \cdot \frac{w_1}{w_2})$. Too young; Too simple.
- Hebbian Learning. A form of learning where only parameters related to large inputs and erroneous outputs are changed.
- K-means does *not* provide visualization capabilities.
- LM algorithm works by trying to find zeros in the gradient space. One more thing: the second order gradient of error function *defines the "shape" of* the corresponding function.