Lecture notes: ML for BIO...

Gilles Richard

Abstract

This is a short summary of what we will discuss during the sessions. Feedback more than welcome!

I suggest to follow another order than the one suggested in these notes. We should start with an introductory session where we devise around the association rules concept. Simple to understand, simple implementation and it gives a clear picture of what we are looking for. Then we go for the logical view of machine learning... with Inductive Logic Programming. Then we carry on with decision tree more decision (diagnostic?) oriented than association rules. Bayesian networks should follow, bringing on the forefront the probabilistic view of data mining (or more generally machine learning). Then obviously, it will be a must to get a global view of machine learning and why not to introduce the Valiant model (PAC) and the VC-dimension of Vapnik... let us see if we have enough time. Investigation around diverse techniques linked to data analysis (like Principal Component Analysis for instance) is optional. I will try to provide at the end of these lecture a brief overview of:

- Kolmogorov complexity
- formal concept analysis (FCA as it is usually abbreviated) which is an alternative to more classical views on matrix data analysis. I feel this method could play an increasing role in business analytics !

For now, some notes are in French but this should change very soon. SVM should come very soon as these notes are still work in progress!

In term of references, these notes are a collection of information gathered from research papers, research reports or tutorials from all around the globe, ultimately mixed with my personal knowledge...

So please surf on the web if you want more or if you need more details. All and everything is on the web !

Brief introduction to machine learning hypothesis

Abstract

In this introduction, we restrict the meaning of machine learning to classification... but obviously machine learning cannot be reduced to this simplistic view!

1 Brief introduction to machine learning hypothesis

Supervised learning is simply a formalization of the idea of learning from examples. Unsupervised... is the opposite situation: we do not have example! In supervised learning, the learner (typically, a computer program) is provided with two sets of data, a training set and a test set. The idea is for the learner to learn from a set of labeled examples in the training set so that it can identify unlabeled examples in the test set with the highest possible accuracy. For example, a training set might consist of images of different types of fruit (say, peaches and nectarines), where the identity of the fruit in each image is given to the learner. The test set would then consist of more unidentified pieces of fruit, but from the same classes. The goal is for the learner to identify the elements in the test set. There are many different approaches which attempt to build the best possible method of classifying examples of the test set by using the data given in the training set. We will discuss a few of these in the following, after defining supervised learning in a more formal way.

In supervised learning, the training set consists of a set of n ordered pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where each x_i is some measurement or set of measurements of a single example data, and y_i is the label (or class) for that data. For example, an x_i might be a vector of 5 measurements for a patient in a hospital including height, weight, temperature, blood sugar level, and blood pressure. The corresponding y_i might be a classification of the patient as healthy or not healthy. The test data in supervised learning is another set of m measurements with- out labels: $(x_{m+1}, x_{m+2}, \ldots, x_{n+m})$. As described above, the goal is to make educated guesses about the labels for the test set (such as healthy or not healthy) by drawing inferences from the training set.

2 Common assumptions about training and test sets

To have any utility, the training data and test data for a given supervised learning problem should have some relationship to each other. Let us assume, for the moment, that the test data are drawn from a distribution $p(x, y)$. If the training data are drawn from some probability distribution $q(x, y)$ that is very different from the distribution of the test data, then we should have no expectation that inferences we make from the training data will help us classify elements of the test set. In other words, it is as if we get very marginal examples, quite different from one would commonly observe for instance, receiving people with normal parameters but being all unhealthy for another unknown reason: this is the worst case scenario where a given class does not get representative in the training set (we receive examples of vegetables only, so the best we can do when we observe a fruit is to say "This is not a vegetable"!

That is why, in most cases, people only apply supervised learning methods if they have some expectation that there is some useful relationship between the distribution of the training data and the distribution of the test data. We now discuss two common scenarios in which data sets are generated for supervised learning. In both cases, we shall assume that the test data are drawn from a distribution $p(x, y)$ i.e. we have no control on the new data to be classified. If this is a case of an hospital, we do not choose our patients (training set) , but if we have to predict the output of an electoral process, to a certain extend, we can have a bit of control on the training set by sampling relevant (or considered as relevant) classes of citizen.

2.1 Sampling using the join distribution

In this case, the training data are also drawn from $p(x, y)$. That is, the training and test data are drawn from the same distribution. Theoretically, with enough data, we should be able to estimate from the training data any conditional or marginal distribution of the training distribution, and hence of the test distribution, that is derivable from the distribution $p(x, y)$. That is, we should be able to estimate $p(y|x)$ directly for any y and x. If we estimate these values perfectly, we should be able to build a Bayes optimal classifier with minimum expected error. Alternatively, we could estimate the likelihoods $p(x|y)$ and the priors $p(y)$, and then use Bayes rule to obtain the posteriors. In the limit of infinite data, both of these methods work perfectly. With finite data, it is not always clear which method is preferable.

2.2 Sampling by classes

In this case, the training data are not drawn directly from $p(x, y)$. Rather, we draw a separate sample of each class by drawing samples from $p(x|y)$. That is, we fix y to be the first class and draw some samples of x for the first class. Then we sample some values of x for the second class, and so on. Why would we ever do this instead of drawing from $p(x, y)$?

Consider trying to build a classifier for a rare disease. We would like to have a sample of patients with the disease and a sample of healthy individuals so that we could build a classifier to distinguish the two. At test time, we will be picking random samples from the general population and trying to decide if they have the disease. Hence, the test data $p(x, y)$ are represented by the general population. Now suppose that patients with the disease only occur in 1 out of 100,000 people in the general population. If we sample people randomly from the join probability $p(x, y)$, we will have to sample a million people before we expect to have even 10 examples of subjects with the rare disease. Hence, it is very impractical to sample from the joint distribution. Instead, we would like to visit a specialist physician who treats the disease and obtain a large sample of the disease population, measuring their symptoms. Sampling this population $p(x|y = hasraredisease)$ separately from the healthy population is clearly far more efficient than sampling from the join distribution $p(x, y)$. Note that with the first sampling method, we may estimate the priors $p(y)$ directly from the samples, but with the second one, we must have a separate method for estimating the priors, since our data do not reflect $p(y)$ in any way. Often, people use another data source for the prior, or simply make an educated guess. In the worse case scenario, when there is no information at all about the classes, we assume they are equiprobable.

3 About Bayes formula

No need to provide a formal introduction to probability theory. The reader is supposed to have it in his/her background. Just to recall the Bayes's rule:

$$p(x|y) = \frac{p(y|x) \times p(x)}{p(y)}$$

How can we read this w.r.t. the real life? $p(x)$ is the probability of the event x to occur. This probability has to be estimated, calculated, with whatever method you want. The same is true for the event y.

$p(x|y)$ represents the probability for x to occur knowing that y has occurred. So $p(x)$ is the raw belief that we have in x to occur, but obviously, when the information that y occurred is available, our belief $p(x)$ that x could occur will change and the probability $p(x)$ does not represent accurately our belief. There is a adjustment factor which is $\frac{p(y|x)}{p(y)}$. A simple example is that you have a new patient a and you have to decide if this guy has a cancer or not. Without any other information, you can estimate the probability of this event $x = hasCancer(a)$ as the proportion of individual having cancer in your country. But if you get the information that the guy is a smoker $y = isSmoker(a)$, then your belief will change and become $p(hasCancer(a)|isSmoker(a))$.

In this particular case, it is likely that $p(x|y) \geq p(x)$ but it is not always the case. The adjustment factor $\frac{p(y|x)}{p(y)}$ can be ≤ 1 or ≥ 1. That's it for now.

4 Useful notions for the remaining of this course

An important concept of the probability theory is the notion of independence of events. It is the formalization of the fact that the probability of event x to occur without any prior knowledge, is exactly the same as its probability to occur having observed the occurrence of event y before i.e $p(x|y) = p(x)$ where $p(x|y) = \frac{p(x,y)}{p(y)}$. It comes that, in that situation, $p(x, y) = p(x)p(y)$ which is generally the accepted definition for x independent from y, sometimes denoted $x \perp y$. This is an interesting property since it allows to compute the joint probability of x and y as a product of their respective probabilities. This is a kind of compositional semantics principle.

As a direct generalization, we can define *conditional independence* of x and y w.r.t. z as $p(x, y|z) = p(x|z)p(y|z)$ denoted $x \perp y|z$. $x \perp y$ is just a particular case of conditional independence where z is just always true (or satisfied). Conditional independence of x and y w.r.t z means that, when z occurs, the occurrence of y does not provide any information about the occurrence of x and vice versa. If we are happy with the measure-theoretic interpretation, $p(x, y|z)$ is the measure of the surface $(x \cap y) \cap z$. It is then easy to understand that conditional independence is equivalent to $p(x|y, z) = p(x|z)$.

Some properties of $x \perp y|z$ can be proved as exercises (they correspond to more or less intuitive facts):

1. $(x \perp y)|z \rightarrow (y \perp x)|z$ (symmetry)
2. $(x \perp y)|z \rightarrow (x \perp \overline{y})|z$
3. $(x \perp y)|z \nrightarrow (x \perp y)|\overline{z}$
4. $(x \perp y, z)|t \rightarrow (x \perp y)|t \wedge (x \perp z)|t$
5. $(x \perp y, z)|t \rightarrow (x \perp y)|z, t$
6. $(x \perp y)|z, t \wedge (x \perp z)|y, t \rightarrow (x \perp y, z)|t$

These properties have to be known in order to understand what are the simplification hypothesis in the field of Bayesian inference and what are their exact consequences.

Data mining introduction - association rules

Abstract

What is data mining and a basic understanding of association rules... Just a basic introduction to a very diverse topic where a lot of money and grey matter are (and will be) invested just because everybody wants to know what tomorrow looks like! We start with basic rules (namely association rules) which are more or less statistical rules (we count!) then we move toward a more sophisticated approach, namely Inductive Logic Programming: we reason!

1 General introduction

Originally, "data mining" was a statisticians term for overusing data to draw invalid (i.e. not sound) inference. Example: we observe 10000 birds and they fly. We infer: "all the birds fly" .... which is wrong!
Really? YES. Just because the pinguin, considered as a bird, is definitely not able to fly (please phone to me if you see a flying pinguin!).
As data mining has become recognized as a powerful tool, diverse communities have worked on the subject, using different approaches:

- Statistics
- Artificial intelligence
- Machine learning
- Databases

Something is clear: with the emergence of the Information Society, a lot of information about people, products, events, etc.. are stored on cheaper and cheaper computers hard drives. Everybody agrees that the stored data can reveal much more than they seem to say. And this hidden information could probably be very helpful for giving new business rules. So data mining is the process of automatic extraction useful hidden information or patterns or rules from data: in other words, going from **implicit** to **explicit** knowledge (from information to knowledge?). Then we are looking for automatic tools to mine these huge amount of data and today, we have a lot of such tools and data mining has a lot of success stories. To say but a few:

- Decision trees issued from bank-loan history for producing algorithms to decide whether to grant a loan.
- Patterns of traveller behavior mined to manage the sale of discounted seats on a plane, rooms in hotel,
- Observations that customers who buy diapers are more likely to buy beers than average, allowed supermarkets to place beers and diapers nearby, knowing many customers would walk between them. Placing chips between increased sales of all three items.

OK... We understand the list will never end! Let us recap the basic steps involved in a data mining process:

1. Data gathering: date warehousing, web crawling... we get a maximum of data...
2. Data cleansing: we need to eliminate errors, irrelevant values (patient fever= 135C), to deal with incomplete data
3. Feature extraction: for instance, the socks colour or blood pressure is not useful for detecting a cancer. We need to extract only the relevant feature.
4. Pattern extraction (knowledge discovery): this is the step often called data mining. We shall concentrate on that part in the following.
5. Results visualization: very useful for decision makers and executives.... What You See Is What You Get;-)

2 Some applications among a never ending list...

This is an overview of current applications of data mining in solving problems across different industries and business sectors. This is not an exhaustive list and you can find a lot of other applications simply by searching over the internet. No doubt that tomorrow we shall see other applications!

- Banking predictive and risk assessment models for the financial services industry, including credit and insurance scoring algorithms
- Biotechnology and pharmaceutical industry building special data mining and visualization tools for pharmaceutical and biotech companies, focused on genomics/functional genomics and drug discovery tools for the analysis of genetic sequence data.

- Customer Relationship Management (CRM) integrating and analyzing customer information from Internet and traditional business channels, enabling businesses to improve their customer acquisition, retention and profitability.

- e-Commerce tools for understanding, targeting, and interacting with customers browser-based Web visitor analysis, campaign management, revenue forecasting, analyzing online shopping behavior and delivering targeted personalized marketing messages to different market segments.

- Fraud detection (Telecom, credit card usage) detecting fraud and predicting typical card usage at merchant location internet credit card fraud detection and risk management service for online merchants. uncovering network intrusions detecting bad debt and application fraud

- Health care knowledge management solutions for creating, managing, disseminating and archiving protocols, procedures and care plans. data warehousing solutions for healthcare industry

- Human resources matching employers needs and job applicant's references. Allows employers to select job applicants who are best suited to the company's needs.

- Marketing creating, optimizing and deploying marketing campaigns integrating direct marketing and Web commerce. Learning from every consumer interaction and applying the knowledge in real-time to deliver the right offer to the customer

- Real time decision making integration of algorithms in real-time transaction systems automated customer interaction facilities detection of retention patterns and trends, triggering appropriate actions.

- Retail businesss support for the future demand planning, seasonal sales, size and channel profiling, customer behavior analysis, price optimization marketing promotion modelling allow retailers to better understand and predict their customers' buying habits.

- Stock and investment analysis and prediction optimizing trading strategies, predicting stocks changes, investment analysis for investment in real estate and residential property.

- WEB analysis providing browser-based Web visitor analysis (finding patterns in web log data), marketing campaign management, , visitor and buyer segmentation and qualification, analyzing online shopping behavior, automated delivery of targeted personalized marketing messages to different market segments, content aggregation/web automation for continuous content acquisition, through "driving" the browser to visit web sites and extracting meaningful information, measuring, tracking and acting upon visitor Internet activity, Web server log analysis tools.

3 Association rules and frequent itemsets: informally first!

In this section, we will adopt a database viewpoint, meaning that we do not consider any statistical methods (which could be used in this case). We will focus on a simple technique whose advantage is to be a good introduction to data mining. Other more sophisticated techniques can be studied but the one we present gives a real flavour of the topic. The best way to understand is to consider a seller-customer logic where the seller wants to optimise his sells. The seller has a stock of items, very large when considering a supermarket. Customers fill in their baskets with diverse items and we want to know what items customers buy together: for instance if customers buy together bread and butter, marketers can position more cleverly the bread and butter items in the store, and then, control in some sense, the way customers travel through the store. Of course, starting from the datasheet we get every night at the end of the working day, we could extract statistics like means, variances, standard deviations, correlations, etc... But they are just numbers which still have to be properly interpreted. Which is not a simple task for the shop manager. That is why a slightly more human-understandable approach and results is very welcome. Our problem is in fact very general: we have two concepts (basket and items) and a relationship between the 2 concepts (here the relationship is "is-in": item i is in basket b)

- $basket = web$ document and $item = word$ and $is - in = appear - in$: then if some words appear frequently in documents, they can represent a same concept or linked concept. Very useful for intelligence gathering.

- $basket = sentence$ and $item = document$ and $is - in = contains$: then documents with the same sentences can represent plagiarism or mirror site on the web.

It simply means that the concept of association rule is quite general and can be applied to a large variety of available data. What are we looking for ? In fact, we would like to get simple rule like: if bread is in the basket, then butter is in the basket, or if bread and butter are in the basket then marmalade is in the basket. Such a rule is called an association rule and denoted:

$$bread, butter \Rightarrow marmalade$$

Thus we are looking for rules of the form:

$$X1, X2, \ldots, Xn \Rightarrow Y$$

The first part of the rule is the **antecedent** and the second part is the **conclusion** or **consequent**... simple.
But we understand that we can have some baskets with bread and butter and without marmalade. A rule $bread, butter \rightarrow marmalade$ simply expresses the fact that if we have bread and butter in a given basket, then there is only a good chance to have marmalade as well. The probability of finding marmalade, provided that we have bread and butter is called the **confidence** level of the rule. This is a real number between 0 and 1. From a mathematical viewpoint, this number is the conditional probability to have marmalade in the basket knowing that bread and butter are in the basket. Since we need to get robust rules, representing the reality, we normally look for rules having a confidence above a certain threshold (for instance 0.9 or 0.8). This threshold has to be significantly greater that it would be if the items were placed randomly into baskets. If this confidence is around 0.5, it simply means that having beard and butter in the bag, does not give much information about the fact that there is marmalade in the given basket. We can throw a coin and get an answer with the same rate of success;-)
Now, we get a complete rule with a confidence level associated to the rule:

$$X1, X2, \ldots, Xn \Rightarrow Y(c)$$

From a marketing viewpoint, we can set up a sale on bread and butter, and increase the price of marmalade since the loss on bread and butter, could be covered by the marmalade. And, at the same time, we put bread and butter at the entrance of the store, attracting a lot of people since the prices are low, then we put the marmalade at the other end of the store, displaying on the way from butter to marmalade all the hard-to-sell items;-) And this is done: a lot of people, with their trolley, will see these items on display... and why not, they will buy one of them.... The conclusion is: despite the store seems to be a big mess, it has been very carefully designed ;-)
But we need to be careful, because if all the baskets have marmalade, even if there is no bread and butter, we will probably extract the previous rule but this is not very significant. Or if we have only one person buying bread and butter, and this person buys marmalade, we will certainly extract the rule but this is still not very significant because the items we are focusing on are not frequently sold together. So we are looking for something more clever. In fact, if we get association rules involving items not frequently sold, we cannot run a good marketing policy since almost nobody buys the items. The consequence is that we need to work on items which appear frequently in the baskets. So we care about set of items with high **support**, i.e. appearing frequently together in the baskets: those set of items are called **Frequent ItemSets** or FIS. Normally, we impose a support threshold to insure we deal with relevant data i.e. we care only about itemsets which appear in at least a certain percentage in the customer's baskets. A frequent itemset is for instance an itemset appearing at least in 20% of the baskets (the threshold is 0.2): the support is then a frequency i.e. a number in $[0, 1]$ (like the confidence level). Our work is then to look for association rules regarding **frequent itemsets** and then we have two tasks to achieve:

1. Given a support threshold s, find the itemsets with support greater than s.
2. Given a confidence threshold c, find the association rules for the previous itemsets, with confidence level greater than c.

Before moving on, let us give the formal definitions.

4 Formal definitions

Here we provide some definitions which are more or less standard. Given a set of items $I = \{i_1, \ldots, i_n\}$ (what we sell) and a set $T = \{t_1, \ldots, t_m\}$ of transactions (the customer's basket) where each transaction is ultimately a subset of X. An association rule is a pair (X, Y) where $X \subseteq I$ and $Y \subseteq I$ i.e. X and Y are just itemsets: it is generally denoted $X \Rightarrow Y$, X is the antecedent and Y the consequent.
The support of an itemset X is the proportion of transactions including X i.e.

$$supp(X) = \frac{\#\{t_i | X \subseteq t_i\}}{m}$$

Simple concept to understand and easy to compute with a given database system. If there is no transaction including X, the support of X is 0 and if all the transactions include X, the support of X is 1. We could consider $supp(X)$ as the probability that a given customer buys X.
The confidence of a rule $X \Rightarrow Y$ is just:

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)}$$

This is easy to compute with a database system and it could be considered as the conditional probability to have Y in your basket knowing that you already have X in your basket. More precisely, this is the frequency of the customers having Y among those having X (and it can be interpreted as a probability if we decide that a probability can be estimated by a frequency).
There is another interesting concept easy to understand in terms of probability. The probability that a customer buy Y is $supp(Y)$: this is my initial confidence or belief. But, if I observe in the basket of the customer the itemset X, then my belief that the customer will buy Y increases to $conf(X \Rightarrow Y)$. Then the ratio

$$lift(X \Rightarrow Y) = \frac{conf(X \Rightarrow Y)}{supp(Y)} = \frac{supp(X \cup Y)}{supp(X) \times supp(Y)}$$

measure how much my belief increases. This ratio is called the *lift* of the rule $X \Rightarrow Y$.
Now we come back to our initial problem: extract the association rules from the available data. This is the aim of the next section to explain how to proceed.

5 Frequent itemsets extraction algorithm

One has to devise an algorithm to give us the association rules, starting from a set of observable data (i.e. an Excel sheet or a huge Oracle/MySQL database). One of the key principles for developing an algorithm is the obvious *downward-closure* property:

If X is a Frequent ItemSet (FIS) then every subset of X is also a frequent itemset.

Unfortunately, the reverse statement is definitely wrong and we cannot avoid a great complexity of the algorithm. It is quite clear that if $\{bread, butter\}$ is a FIS, nothing insures that $\{bread, butter, razor\}$ is still a FIS!

We deduce that a FIS of cardinality k is made of k FIS of cardinality $k-1$. But more importantly, a FIS of cardinality k is made of 2 FIS of cardinality $k-1$ differing from 1 element only. This is very useful to design a complete algorithm giving all the FIS related to a given threshold s.
These properties give us the main key to build an algorithm allowing to proceed level-wise. Starting from a threshold $s \in [0, 1]$ for our support level:

1. Among the whole database denoted $L0$, the FIS of cardinality 1 (i.e. the frequent items) are those items whose count reaches our initial threshold s: we get a set of items $L1$ i.e. the set of FIS of cardinality 1.

2. Bringing together 2 distinct elements a and b of $L1$, we get a candidate pair (a, b) and $C2 = (L1 \times L1) \setminus D$ where D denotes the diagonal of the cartesian product, is just the whole set of candidate pairs. In $C2$, the elements whose frequency reaches s constitute $L2$, subset of $C2$: $L2$ is the set of FIS of cardinality 2.

3. The candidate triples $C3$ are those sets (a, b, c) such that (a, b), (b, c) and (a, c) belong to $L2$ (following the monotony principle). So $C3$ is a subset of $L1 \times L1 \times L1$. For each triple in $C3$, we count the occurrences in the baskets. Those ones having a frequency of at least s constitute $L3$.

4. We proceed like that as far as possible (if a set Li is empty, we stop): when we get Li the frequent sets of size i, $Ci + 1$, the set of candidate FIS, is just the set of sets of size $i + 1$ such that each subset of size i is in Li.

A simple algorithmic scheme is below which summarize what has been said:

```
Init given s
 i=0;
 C0=whole-set-of-item;
 While (Li not empty){
    Compute Ci from (set of candidate n-uples)
    Extract Li from Ci
    i=i+1;
    }
```

Obviously, as soon as we have a database for implementing Basket (for instance, we have a relation Basket(BID, item)), this algorithm for getting the pair of frequent items can be implemented as a single SQL query looking like:

```
SELECT b1.item, b2.item, COUNT(*) FROM Basket b1, Basket b2
WHERE b1.BID = b2.BID AND b1.item < b2.item
GROUP BY b1.item, b2.item
HAVING COUNT(*) >= s
```

The term $b1.item < b2.item$ is just for preventing pairs of items which are really a single one (we want to get pair of distinct items). This algorithm can be improved using hash tabling but the main idea remains.
An other different approach is to represent the model basket-item as a boolean matrix (rows=baskets, columns=items). We can assume the matrix is very sparse (almost all 0s) which is the case for the supermarket example. In that case, we can use specific algorithms for matrix manipulation.
We can as well use a Prolog-like program (without any consideration of efficiency) :

$$answer(b) : -basket(b, X), basket(b, Y), X! = Y$$

Then we define an answer, meaning a basket containing the distinct items X and Y. Some more lines allow to count the number of answers and to check if we get the required threshold. Obviously, this is not the best programming language to do the job w.r.t. the structural complexity of the problem. A language like C is much more appropriate.

6 Association rules extraction algorithm

Starting from the previous algorithm, it is not a big deal to finally get the association rules, starting again from a confidence level c and a support level s. Basically, the technique is as follows:

- given a FIS of suitable support (i.e. $\geq s$), generate all the candidate rules
- among the candidates rules, eliminate those ones having a confidence level less than the treshold c.

This has to be done for all FIS and we immediatly understand that the whole process is quite expensive! Especially the first part, finding the FIS. In fact, initially at least we have 2^n (where n is the number of items) candidate FIS... i.e. starting from a brute force approach, the algorithm is at least exponential in the size of FIS. Thanks to the downward closure property, this can be seriously improved. There are diverse efficient algorithms in the litterature. Apriori algorithm, initially designed by Rakesh Agrawal and Ramakrishnan Srikant, (Frank Borgelt provided a very efficient implementation), is quite well known but there are other competitors.

Today, any serious database system (like Oracle for instance) includes an association rules extraction engine in its additional tools ;-) Weka is also a good open source free tool for association rules extraction.

7 Some applications and concluding remarks

Data mining can be viewed as an extension of statistics, with a few artificial intelligence and machine learning twists thrown in. Today, data mining has a lot of practical applications in diverse areas.

Text mining: Consider then that 80 percent or more of the average organization's information is in unstructured, or textual form, compared to 20 percent or less in the structured tables and databases used in traditional data mining. That means that customer e-mails, call center notes, open-ended survey responses, Web forms, and other textrich sources of

valuable dataoften remain unused. This is thus a technical challenge to mine such data but recent advances allow to extract key concepts, sentiments, and relationships from this unstructured data, and convert it to structured format for predictive modelling. Dont forget that mining the textual log files of a server is very useful to detect security holes or to improve server performance. From a practical viewpoint, some software can analyze approximately 1 gigabyte of text (or 250,000 pages) per hour, with 90 percent or better accuracy. In addition, they can process

- all common document types, including plain text, HTML, XML, PDF, and MS Office document formats,
- all types of languages (Dutch, English, French, German, Italian, Japanese, Spanish, Arabic,)

Web mining: Web Intelligence is opening a new chapter in the development of e-Business. A capability to learn interests and preferences of each visitor by observing their behavior at the website, and to have the site interact with its visitors intelligently one-on-one, is a crucial competitive advantage in term of marketing. Which visitors are the best potential customers? Can we learn the interests of a prospect quickly and precisely? What is the best resource to show to the visitor next? A website needs to deliver answers to these questions on the fly, acquiring data that originates from navigating a website, uncovering visitor-specific knowledge hidden in this data, and utilizing this knowledge to increase the quality and value of future interactions with the best prospects. Intelligent mining of the web pages (HTML documents, log files), of the sequence of hypertext links provide deep insight about the customer behaviour.

Scientific applications: one of the most obvious application of data mining is related to the genome analysis. Genomics is the study of an organism's genome and the use of the genes. It deals with the systematic use of genome information, associated with other data, to provide answers in biology, medicine, and industry. Genomics has the potential of offering new therapeutic methods for the treatment of some diseases, as well as new diagnostic methods. Other applications are in the food and agriculture sectors. The major tools and methods related to genomics are bio-informatics, genetic analysis, measurement of gene expression, and determination of gene function. Genomic sequencing and mapping efforts have produced a number of databases which are now accessible over the web (see EBI web site for instance). In addition, there are also a wide variety of other on-line databases, including those containing information about diseases, cellular function, and drugs. Finding relationships between these data sources, which are largely unexplored, is another fundamental data mining challenge. In that field, data mining (and machine learning in the large) plays a very important role.

Of course, there is some technical requirements for effective data mining. Today, data mining applications are available on all size systems for mainframe, client/server, and PC platforms. System prices range from several thousand euros for the smallest applications up to several millions for the largest. Enterprise-wide applications generally range in size from 10 gigabytes to over 11 terabytes (Some companies have the capacity to deliver applications exceeding 100 terabytes).

We understand that there are at least two critical technology drivers:

- the size of the database: the more data being processed and maintained, the more powerful the system required.
- the query complexity: the more complex the queries and the greater the number of queries being processed, the more powerful the system required.

The current infrastructure needs to be enhanced to support future applications:

- by adding extensive indexing capabilities to improve query performance.
- by using new hardware architectures such as Massively Parallel Processors (MPP) to achieve order-of-magnitude improvements in query time.
- by quantum computing when the technology is ready!

Inductive Logic Programming

Abstract

We are still trying to extract implicit knowledge from raw data... but now we are not interested in just counting the number of occurences of blablabla... We want to highlight hidden relations without any count, just relying on a declarative way to describe our universe.

1 General introduction

Since the beginning (Aristote?), logic is mainly related to inference i.e. to derive new conclusions from observations or from rules assumed to be true. Logic tries to formalized the human process of reasoning. Obviously, there is a lot of ways to reason and it is not always based on logic ! We can consider 2 modes of inference:

- *deductive inference* where we have general rules, the premises, and we try to draw other general or particular rules, the conclusion. Logic mainly focus on this mode of inference
- *inductive inference* where we have particular observations (the conclusion) and we try to get the premises (or the causes).

One of the most famous example is the "Socrates" example.

deduction: "all men are mortal" and "Socrates is a man" then "Socrates is mortal. We go from general to particular in that case.

induction: "Socrates is a man" and "Socrates is mortal" then "all men are mortal". We go from particular to general.

We immediately understand that induction is not 100% safe which is not the case for deduction (which is safe as soon as we follow logical rules). Unfortunately, induction is what we need for machine learning or data mining... There is an awful lot of literature about formal logic and it is not our aim to investigated this amzing fiels... just to get some points useful for our business.

2 Deduction (and Prolog)

2.1 Deduction in logic

This introduction is volontarily brief and relatively informal. Formal logic is a whole topic in itself which is out of the scope of this lecture. Different type of logics (a lot !). Let us stick to classical logics:

- propositional logic (order 0):
- predicate logic (order 1)
- higher order logic (mainly order 2)

Language: a set of standard symbols including connectors $\wedge, \vee, \rightarrow, \neg$.
In the case of 1st order and higher order, 2 quantifiers $\forall$ and $\exists$.
A set of propositional variables usually denoted $a, b, \ldots$ for propositional logic A set of individual variables usually denoted $x, y, \ldots$ for 1st order logic and finally a set of predicate variables $X, Y, \ldots$ for 2nd order logic. We do not have propositional variables for 1st and 2nd order.
In the case of 1st order and 2nd order, we need function symbols $f, g, \ldots$ and predicate symbols $p, q, \ldots$.

For instance:

1. $\neg(p \wedge q) \rightarrow (r \vee q)$ is a propositional formula. A propositional formula represents a statement like "it rains" which can be true or false. That's it.
2. $\neg(\forall x(p(x) \wedge (\exists y, q(x, y)) \rightarrow (r(x, y, z) \vee (q(x, z)$ is a 1st order formula. It represents a statement involving variables like $rainyDay(x)$ which can be true or false depending of the day x! For instance $rainyDay(Monday)$ can be true and $rainyDay(Tuesday)$ can be false. Same story for $isSorted(x)$ or $isSorted([1, 2, 3, 1]$.

3. $\forall X, \exists x, X(x)$ is a 2nd order formula whose meaning is "whatever the property X that we consider, there exists and element x satisfying this property". This statement can be true or false (it is probably false since if $X = p \wedge \neg p$, we do not know an x satisfying $p(x) \wedge \neg p(x)$!

Let us have a look on our formula. $rainyDay(x)$ can be made identical to $rainyDay(Tuesday)$ just by substituing the value $Tuesday$ to the variable x. We say we have **unified** the 2 formula (when we have a simple formula like that, we call it an atomic formula or an atom, a literal is just an atom or the negation of an atom, one of the simplest formula on earth!). Obviously, it can be a little bit more tricky as in the case of $p(x, f(g(z))$ and $p(3, f(y))$, but in that case, we can still unify with the **substitution** σ such that: $\sigma(x) = 3, \sigma(y) = g(z), \sigma(z) = z$. We understand that unification is not a process linked to formal logic but it is a general process that we can perform when dealing with a language and terms over this language.

From time to time, we cannot unified 2 terms (for instance $p(x, 3)$ and $p(1, g(y)$. When we can unified 2 terms t_1 and t_2, there exists a more general unifier σ such that $\sigma(t_1) = \sigma(t_2)$. In some sense, when applying this mgu, we make t_1 and t_2 identical.6

Axioms: the same ones in each case BUT some specific rules to deal with quantifiers in 1st order and 2nd order. An axiom is just a particular formula supposed to be true. For instance $a \rightarrow (b \rightarrow a)$.

Deduction rules: Modus ponens - etc...

For instance

$$\frac{a \quad a \rightarrow b}{b}$$

which is known as Modus Ponens, a kind of common sense rule. We have to be careful with the common sense when we do formal logic... The same rules are valid in each case BUT some specific rules to deal with quantifiers in 1st order and 2nd order have to be added.

Resolution: a fantastic deduction rule covering all the other ones when we restrict our business to specific formula... In propositional logic, it can be expressed as:

$$\frac{a \vee b \quad c \vee \neg b}{a \vee b}$$

This can be read as removing the symbol b from the story. In 1st order logic, it can be expressed in a more general way:

$$\frac{a \vee b \quad c \vee \neg b'}{\sigma(a) \vee \sigma(b)}$$

where σ is the most general unifier between b and b'.

Deductive chain or sequence: a finite list of formula which are either axioms, or obtained from a previous formula using a deduction rule.

Our aim is to get something new from what is known. Something is new if we can get it at the end of a deduction. It is known as a theorem.

Notion of truth and falsity: no need to insist for now.

Notion of completeness and soundness (or correction): philosophical problem first, then can be investigated formally (Godel's incompleteness theorems). This is really out of the scope of this introduction.

2.2 Deduction implemented with Prolog

We have at our disposal the standard set of axioms of 1st order logic. We add some more axioms to describe our universe: this is our program. The axioms we add are specific 1st order formula known as "Horn clause" or simply "clauses". A clause is just a disjunction of literals. For instance $a \vee \neg b$ is a clause, equivalent to $b \rightarrow a$. Or $a \vee b \vee \neg c \vee d \vee \neg e$ is also a clause equivalent to $c \wedge e \rightarrow a \vee b \vee d$. If there is at most 1 positive literal, we say that this is a Horn clause. A horn clause $a \vee \neg b \vee \neg c \vee \neg d$ is better understood as an implication $b \wedge c \wedge d \rightarrow a$. The left hand side of the symbol $\rightarrow$ is the body of the clause and the right hand side is the head of the clause. The body is a conjunction of atoms and the head is reduced to 1 atom only. Such a set of Horn clauses is a program (or a logic program). We informally understand that the resolution rule can be easily applied to Horn clauses since we have only one choice for the positive literal.

Unfortunately the standard notation for $b \wedge c \wedge d \rightarrow a$ is $a :- b, c, d....$ I know this looks like a joke but think about your keyboard... and you will understand why this is the notation. The final dot is important as it tells to Prolog interpreter that this is the end for a given clause.

Let us start with a simple example with the following propositional program: $a :- b, c, d.,c :- b., d :- e, b.,e.,b.$ It is an easy game, applying the resolution rule to prove a. So asking for $a?$ (b or c), Prolog answers "Yes". Asking for f, Prolog will answer "No".

When asking a query $Q(x)?$, the Prolog engine will try to find the values x_0 of x making $Q(x_0)$ deducible from the initial program. In other words, if P is a Prolog program i.e. a finite set of Horn clauses, then giving the request $q(x)$ to the Prolog interpreter, will result in the interpreter to try to find out the substitution

$$\sigma \text{ such that } P \models \sigma(q(x))$$

Such a substitution is a solution. Prolog will try to give all the solutions or answers "no" if there is no solution.
It is time to go for our first program decribing the addition of integers.

$add(0, x, x).$

$add(s(x), y, s(z)) :- add(x, y, z)$

And we work about it now... And we do some more exercises live ! And we do some more exercises (homework). When we are comfortable with Prolog, we have to go a step forward. Remember we do data mining or machine learning. So we have examples describing a given situation and we want to automatically extract a Prolog program from which we could derive our observations.

3 Induction

In the previous section, we have learned how to deduce new formula from existing ones. In other words, we tried to extract the implicit knowledge hidden in the initial formula. As we understand, it is not a matter of guessing, it is a matter of formal logic. We are looking to prove something. If we can prove the fact $\sigma(q(x))$, we are sure this is true: this is a logical consequence of our program. From time to time, we cannot prove it because either it is false or our Prolog interpretor loops somewhere. But now, we are in the reverse process where we try to find the cause of our observations. It is essentially a non deterministic problem and unsafe as well. Whatever our effort, we will just get a candidate cause but we will never be sure this is the right one.

3.1 Induction in logic

Our examples are generally recorded in a table or Excel file...This can be seen as an (attribute-value) description and this is definitely not suitable for logic! We have to describe our examples using a proper 1st order logic syntax. Back to our store dataset, selling item $i_1, \ldots, i_n$, where we have a line per customer: for instance for customer 1000 we have the line i_1, i_3 and $i8$ telling us what is in the customer 1000 basket. This line of the Excel sheet or of the Oracle/MySQL database has to be translated into:

$$inBasket(1000, i_1).inBasket(1000, i_3).inBasket(1000, i_8)$$

It is very boring task but a simple C program can do the job in one shot! Back to our association rule story, instead of looking for an association rule, we could get a 1st order formula like, for instance:

$$inBasket(x, i_{15}); -inBasket(x, i_1), inBasket(x, i_3)$$

which is the translation of the association rule:

$$i_{15} \leftarrow i_1, i_3$$

But, be careful, in the cas eof the association rules, there is an associated confidence level: this is not the case for our clause which is supposed to be universaly quantified... i.e. with NO exception. This is one of the main issue within ILP: to deal with approximate truth ! Now that we have transformed our Excel table into a set E of 1st order logic formula (facts), we can start to work. We are looking for a program H, $H \neq E$ such that

$$H \models E$$

Generally, we want H to be simpler than E... simpler ??? For instance, less rules than E, more compact... H can be seen as a compression of E. In fact, very often, it appears that we can have a background knowledge B related or not to the current observations. For instance, we know that

$$isConsistentMenu(m) : -hasStarter(m), hasMainDish(m), hasDesert(m).$$

So we can consider this formula as a background knowledge useful to deal with our problem. Obviously, if $B \models E$, we have an explanation for E and there is no need to look for other formula. So the basic assumption is

$$B \not\models E$$

and our problem is to find the missing hypothesis or explanation H such that $B \wedge H$ is consistent and

$$B \wedge H \models E$$

It looks like the same story as before (a story of deduction) EXCEPT that we reverse the process: we are looking for something on the left hand side of the symbol $\models$ (in the previous section, we were looking for something on the right hand side). In ordinary words, we observe something and we want to know the cause!

3.2 Induction implemented with Progol

To reverse the deduction process is not as simple as it could appear and there is not a unique powerful method. Different inverse deduction rules lead to diverse ILP engines. Progol for instance implements what is known as *inverse entailment*. It is relatively simple to understand why the induction process cannot be deterministic. Let us consider Modus Ponens

$$\frac{a \quad a \rightarrow b}{b},$$

we see that the consequence b appears in the antecedents (or the causes). But if we observe b, how to imagine that its cause is a? a does not appear in the consequent of MP not is hidden in b). We could have c and $c \rightarrow a$ as well: so MP applies and we still get b. It means a and c are candidate causes for b.
More or less, the process is similar:

1. $E_{toCover} = E$
2. Choose an example $e \in E_{toCover}$

3. Using the reverse deduction implemented process, build up a clause c such that

$$B \wedge c \models E$$

4. Remove all the examples $f \in E_{toCover}$ such that $B \wedge c \models f$

5. Stop when $E_{toCover}$ is empty

The set of new build clauses constitutes a Prolog program H such that

$$B \wedge H \models E$$

which is exactly what we want. It just remains to practice... using Progol (freely available on Imperial College website - homepage of Stephen Muggleton - be careful about the setup - seems to run on Unix machine only!) or Weka with ID3 or FOIL (and C4.5) from Quinlan.

4 Conclusion

The main issues coming from the logical approach to data mining and machine learning in the large are

- the lack of flexibilty due to universal quantifiers. There are some attempts to introduce probability to get weighted rules (a kind of confidence level as for association rules). See statistical relational learning for instance.

- the difficulty to deal with real numbers. To do that, we have to introduce constraints like $3.2 \leq x \leq 8.5$ for instance and to manage these constraints with an external constraints solver.

Despite these 2 drawbacks, ILP has been successfully applied to genetics (see for instance http://pages.cs.wisc.edu/ dpage/).
Progol from Stephen Muggleton (see the web) is an easy to use tool for ILP practice.
Syntax: pathToProgol/progol dataToBeMined
The file dataToBeMined should follow the Progol rules which are quite similar to Prolog but with some added features to guide the engine search (mode declaration). See also Aleph as an ILP tool. See also the system of Katsumi Inoue (Tokyo - NII)

Decision trees

Abstract

The approach developped in this chapter is quite different from the association rule approach but the aim is the same: developing prediction and classification rules for various problem domains... an aim which has been pursued by the statistical and machine learning communities for many years.

1 General introduction

As with any data mining problem, we start with an initial set of data which consists of various observations for which a known property or class has been assigned. Further, for each of these observations, a number of measurable features exists which describe various details of the observations. The ultimate problem involves developing a system by which the properties can be predicted from the features. An extension of this objective would be to use the classifier system to predict what we would expect the class to be for an observation which is not in our original data set. This can be conceptualized by considering a data matrix in which the rows represent the observations and the columns represent the features associated with those observations. John Ross Quinlan is a scientist who has extensively contributed to the development of decision tree algorithms: he is the author of the well-known C4.5 algorithm, and its descendant, ID3 algorithm. This is why we will start from examples taken from his litterature. In Table 1, taken from [Quinlan 1989], the observations (rows) represent various days of the week in which we may choose to go outside or stay in for some "unspecified activity." The columns each represent various features associated with the observations, or in this case the various conditions of the day in question. While this simple example represents a choice on whether to go outside or stay inside, the same representation can be

Figure 1: Our data

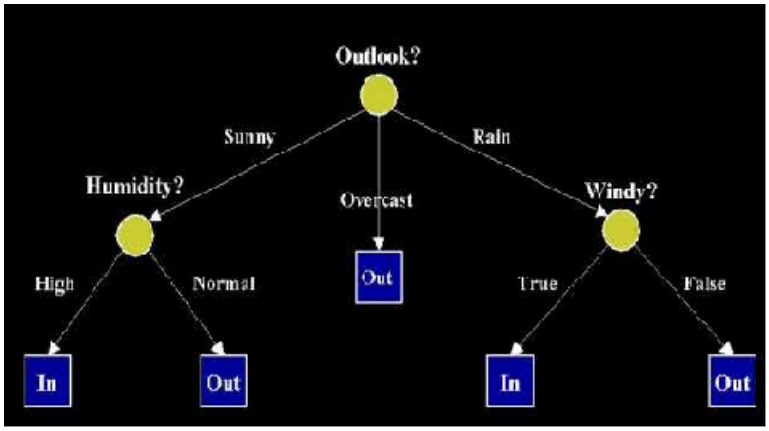
Day #	Outlook	Temperature	Humidity	Windy?	Out or In?
1	Sunny	Hot	High	False	In
2	Sunny	Hot	High	True	In
3	Overcast	Hot	High	False	Out
4	Rain	Mild	High	False	Out
5	Rain	Cool	High	False	Out
6	Rain	Cool	Normal	True	In
7	Overcast	Cool	Normal	True	Out
8	Sunny	Mild	High	False	In
9	Sunny	Cool	Normal	False	Out
10	Rain	Mild	Normal	False	Out
11	Sunny	Mild	Normal	True	Out
12	Overcast	Mild	High	True	Out
13	Overcast	Hot	Normal	False	Out
14	Rain	Mild	High	True	In

applied to virtually any field and domain. For example, consider classifying glass collected at a crime scene as automobile headlight or window pane glass based upon the chemical content of the samples; proteins into various classes based upon their amino acid content; chemical compounds into the categories toxic or non-toxic based upon their chemical structures; tissue samples as cancerous or non-cancerous based upon various measurements of the tissue itself such as their expressed genes. Each of these, and too many others to mention, are actual examples of classification projects which have been the subject of classification rule development. To simplify, a decision tree is a classifier with the form of a tree structure, where each node is either:

- a leaf node - indicates the value of the target attribute (class) of examples,
- a decision node - specifies some test to be carried out on a single attribute-value, with one branch and sub-tree for each possible outcome of the test.

A decision tree can be used to classify an example by starting at the root of the tree and moving through it until a leaf node, which provides the classification of the instance. We can consider a decision tree as another graphical representation of a decision flowchart: each decision node implements a Boolean test (if - then - else) whose answer determines the remaining of the control flow. Figure 2 represents a perfect decision tree for the data of our previous table. We start at the topmost point in the tree and ask the question, "What is the outlook for the day?" This point, and all those in which a question based upon the data is asked, is called a node with the first decision node referred to as the root of the tree. The answer to the question determines the path we take through the tree. For example, if we respond, "Overcast, we move down the middle path to a position which provides a class value for the observation in question. This final destination which has no other paths leading away from it is called a leaf, and each leaf has a classification attached to it. We start at the root and ask the question, "What

Figure 2: Our tree



is the Outlook?" From Table 1 we see that the outlook is sunny which leads us to the left path and another decision node asking the question , "What is the Humidity?" We see that the humidity is High, so the left path is taken and we arrive at a leaf which predicts that we should stay inside. Looking at the category column of Table 1 (In or Out?) we see that, indeed, this is classified as in inside day. As mentioned earlier, one of the more common uses of decision trees such as this one is to predict the class of an observation which was not in our original data set. In this situation, we refer to the original data as the Training Set, and consider it to be a highly accurate and reasonable basis to use for future predictions. A decision tree is generated from this Training Set and used to predict the class for new observations. The actual class of these observations may not be known, and therefore the decision tree is playing the role of a predictive classifier. For example, suppose we developed a decision tree based upon a data matrix which was related to classifying cells as cancerous or non-cancerous. Once the decision tree is generated, a physician can use it to assist in the diagnosis of new patients. The main question now is how to build an effective decision tree. This is the object of the next section to answer this question.

2 Decision tree: formal procedure

To address the problem of decision tree construction, an algorithm known as Recursive Partitioning was developed. While there are many different versions of Recursive Partitioning (RP) available, each with its own unique details, the overall methodology is consistently the same regardless of the exact implementation.

2.1 Recursive partitioning

Figure 3 illustrates the general methodology involved in Recursive Partitioning.

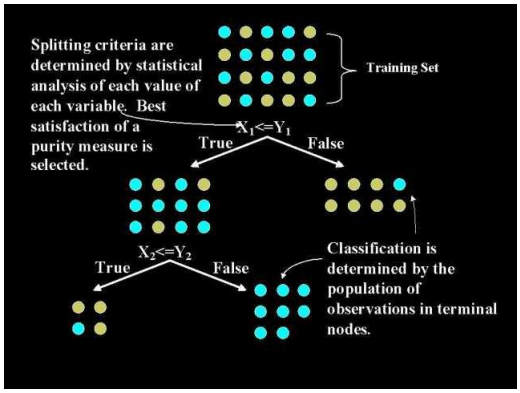
To understand the RP process, a few basic concepts must first be understood. The first of these is the concept of splitting or partitioning the training set. During the process of decision tree induction, we have to consider what questions we will ask in order to direct the user down the appropriate path. For simplicity, lets consider that each potential question can have a true or false answer, thus, any particular node will have at most two paths leading from it to the next node(s) in the path. Every possible value of every possible feature within the training set represents a potential split that could be done. The result is that we will be able to go down the right path or the left path based upon the data and we will effectively split the data at each node into two independent groups this is partitioning. For example, let us consider a training set which has purely numerical data. The features will be called X_n and the possible values for those features will be called Y_m. As a result, every question that could be asked can take the form

"Is X_n less than or equal to Y_m?"

The resulting answer will direct us down the appropriate path, e.g., if X_n is less than or equal to Y_m, then go left, otherwise, go right. Once we have two new nodes (children nodes) linked to a previous node (the parent node), we can repeat the process for each child node independently using only the observations present in that child the recursive step.

The next central concept is related to how we choose which question to ask i.e. which attribute to test at each node in the tree. A new very general notion is now necessary.

Figure 3: Partitioning the dataset



2.2 Entropy and information

Remember that we could ask a question of the above form for every possible value of every possible feature within our training set. For the purpose of choosing the most *appropriate* question i.e. the most appropriate attribute, we need an measure of "appropriateness". This estimation criterion will allow to select the most useful attribute to test at each decision node in the tree. A good quantitative measure of the worth of an attribute is a statistical property called **information gain** that measures how well a given attribute separates the training examples according to their target classification. This measure is used to select among the candidate attributes at each step while growing the tree. In order to define information gain precisely, we need to define a measure commonly used in information theory, called **entropy**, that characterizes the *(im)purity* or *disorder* of an arbitrary collection of examples. Given a set S, containing only positive and negative examples of some target concept (a 2 class problem), the entropy of set S relative to this simple, binary classification is defined as:

Definition 1 $Entropy(S) = -p_p log_2(p_p) - p_n log_2(p_n)$

where $p_p$is the proportion of positive examples in S and p_n is the proportion of negative examples in S. It means $p_n + p_p = 1$. In all calculations involving entropy, we adopt the convention $0log_2(0) = 0$ which is acceptable since we have $lim_{x\to 0} x log_2(x) = 0$ To illustrate, suppose S is a collection of 25 examples, including 15 positive and 10 negative examples $[15+, 10-]$. Then the entropy of S relative to this partitioning is:

$$Entropy(S) = -(15/25)log2(15/25) - (10/25)log2(10/25) = 0.970.$$

- Due to our convention, the entropy is 0 if all members of S belong to the same class. In that case, S is very homogeneous and the *disorder* is minimal. If we have to pick up an element of S, we are certain of the class it belongs to.
- The entropy is 1 (at its maximum!) when the collection contains an equal number of positive and negative examples. In that case, the disorder is maximum since if we have to pick up an element of S, we have no clue to guess its class. The process is entirely random.
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1.

Another interpretation of entropy from information theory is that it specifies the minimum number of bits of information needed to encode the classification of an arbitrary member of S (i.e., a member of S drawn at random with uniform probability). For example, if p_p is 1, the receiver knows the drawn example will be positive, so no message need be sent, and the entropy is 0. On the other hand, if p_p is 0.5, one bit is required to indicate whether the drawn example is positive or negative. If p_p is 0.8, then a collection of messages can be encoded using on average less than 1 bit per message by assigning shorter codes to collections of positive examples and longer codes to less likely negative examples. Thus far, we have discussed entropy in the special case where the target classification is binary. If the label of an example can takes on c different values $l_1, \ldots, l_c$, then the **entropy of** S is defined as a generalization of the previous definition:

Definition 2 $Entropy(S) = -\Sigma_{i=1}^{c} p_i log_2(p_i)$

where p_i is just the proportion of elements in S whose label is l_i. As in physics, entropy is a measure of the *disorder* of the set S with regard to the setsof labels $\mathcal{L} = l_1, \ldots, l_c$.

If we can partition thanks to the value of a given attribute a set S in to 2 subsets $S1$ and $S2$ with lower entropy than the initial set S, it means we have extracted an interesting attribute and corresponding value: in each subset, the disorder has decreased or the homogeneity has increased. Note the logarithm is still base 2 because entropy is a measure of the expected encoding length measured in bits. Note also that if the target label/class can take on c possible values, the maximum possible entropy is $log_2(c)$.

2.3 Information gain

Given entropy as a measure of the impurity in a collection of training examples, we can now define a measure of the effectiveness of an attribute in classifying the training data. The measure we will use, called **information gain**, is simply the expected reduction in entropy caused by partitioning the examples according to this attribute. In fact, we want to choose an attribute such that testing on its values will partition the remaining subset into lower entropy (more homogeneous sets). More precisely, the information gain, $Gain(S, a)$ of an attribute a taking its value in a finite set $value(a)$, relative to a collection of examples S, is defined as

$$Gain(S, a) = Entropy(S) - \Sigma_{v \in value(a)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where $S_v = \{x \in S | a(x) = v\}$ i.e the elements of S whose attribute a has value v. The first term in the equation for $Gain$ is just the entropy of the original collection S and the second term is the expected (or average) value of the entropy after S is partitioned using attribute a. The expected entropy described by this second term is simply the sum of the entropies of each subset S_v, weighted by the fraction of examples $\frac{|S_v|}{|S|}$ that belong to S_v. $Gain(S, a)$ is therefore the expected reduction in entropy caused by knowing the value of attribute a. Put another way, $Gain(S, a)$ is the information provided about the target attribute value, given the value of some other attribute a. The value of $Gain(S, a)$ is the number of bits saved when encoding the target value of an arbitrary member of S, by knowing the value of attribute a.

Ultimately, a decision tree will have as top nodes the *most important* attributes for reducing the entropy (in other words the more discriminating ones). When we build a decision tree, we start from the whole training set S to find the first attribute a to be used. Then this training set is breakdown into disjoint subsets $S_1, \ldots, S_k$, each one corresponding to a value of the first attribute a. Now when looking for the next attribute, we do not deal anymore with the whole S but with the particular subsets S_i associated to the values of a. The entropy of each S_I w.r.t. the initial set of classes has still to be computed (in fact it could have been memorized from the previous step) then the information gain for each remaining attribute and each S_i has to be computed. Of course, it is not always the same attribute which will do the job for 2 distinct S_i's... We immediately understand that this is a computationally expensive process!!!

2.4 Algorithm

The process of selecting a new attribute and partitioning the training examples is now repeated for each non-terminal descendant node, this time using only the training examples associated with that node. Attributes that have been incorporated higher in the tree are excluded, so that any given attribute can appear at most once along any path through the tree. This process continues for each new leaf node until either of two conditions is met: every attribute has already been included along this path through the tree, or the training examples associated with this leaf node all have the same target attribute value (i.e., their entropy is zero).

Of course, these theoretical criteria could not always been got in a reasonable time, that is why we have to add some type of practical stopping criteria. If we were to allow the splitting process to continue until each leaf only had 1 observation, we would have a perfect tree! This situation, however, is very dangerous if we want to use our resulting decision tree for the purpose of prediction. Most likely a tree of this sort is over fit for the training data and will not perform well on new data. To avoid this process, we introduce a stopping criteria to halt the recursive partitioning process. Stopping criteria come in many different forms, including: A maximum number of nodes in the tree. Once this maximum is reached, the process is halted. A minimum number of observations in a particular node can be set such that if the number of observations in a node is less than or equal to a minimum value, partitioning of that node will not be attempted, and it becomes a leaf. Once we have determined that we will stop the procedure, we have effectively reached a leaf in our tree. Based upon the observations that have made it through the tree to that leaf, we can assign it a class value. One common way to do this is to determine which class is in higher numbers and use that class as the leafs class value. This is merely a majority rules voting method and, like all other aspects of tree induction, many different methods exist by which to determine a leafs class label. While the above procedure for decision tree induction can be modified and fine-tuned for the task at hand, the general procedure remains the same regardless of the choice of purity measures or stopping criteria. Additionally, while this is a method that is used extensively throughout the field of pattern recognition, a potential flaw exists. At each node which we have decided needs to be split, we look at the immediate effects on the children. As mentioned above, we look through each and every possible way to partition this nodes observations and choose the one which gives us the best purity in the immediate children. It is conceivable, however, that if we choose a splitting criteria which is not optimal, we might have better choices much later in the tree which we cannot see in the immediate children. In other words, a sub-optimal split may lead to an increase in optimality of choices for later splits. We can summarize this discussion by giving the standard scheme. Most algorithms that have been developed for learning decision trees are variations on a core algorithm that employs a top-down, greedy search through the space of possible decision trees. Decision tree programs construct a decision tree T from a set of training cases (J. Ross Quinlan originally developed ID3 at the University of Sydney):

```
function ID3
Input:   R: a set of non-target attributes,
         C: the target attribute,
         S: a training set
Output:  a decision tree

begin
   If S is empty, return a single node with value Failure;
   If S consists of records all with the same value for the target attribute,
      return a single leaf node with that value;
   If R is empty, then return a single node
       with the value of the most frequent of the values of the target attribute that are
      found in records of S; [in that case there may be be errors, examples improperly classified];
```

```
   Let A be the attribute with largest Gain(A,S) among attributes in R;
   Let {aj| j=1,2, .., m} be the values of attribute A;
   Let {Sj| j=1,2, .., m} be the subsets of S consisting respectively of records
      with value aj for A;
   Return a tree with root labeled A and arcs labeled a1, a2, .., am going respectively
      to the trees (ID3(R-{A}, C, S1), ID3(R-{A}, C, S2),...,ID3(R-{A}, C, Sm);
   Recursively apply ID3 to subsets {Sj| j=1,2, .., m} until they are empty
end
```

ID3 searches through the attributes of the training instances and extracts the attribute that best separates the given examples. If the attribute perfectly classifies the training sets then ID3 stops; otherwise it recursively operates on the m (where m = number of possible values of an attribute) partitioned subsets to get their "best" attribute. The algorithm uses a greedy search, that is, it picks the best attribute and never looks back to reconsider earlier choices. Note that ID3 may misclassify data.

3 Some improvements

Practical issues in learning decision trees include determining how deeply to grow the decision tree, handling continuous attributes, choosing an appropriate attribute selection measure, handling training data with missing attribute values, handing attributes with differing costs, improving computational efficiency, etc.. Some of them are addressed by the ID3 algorithm: we discuss in the following sections.

3.1 Avoiding over-fitting the data

In principle, decision tree algorithm described above can grow each branch of the tree just deeply enough to perfectly classify the training examples. While this is sometimes a reasonable strategy, in fact it can lead to difficulties when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. In either of these cases, this simple algorithm can produce trees that *over-fit* the training examples. Over-fitting is a significant practical difficulty for decision tree learning and many other learning methods. There are several approaches in order to avoid over-fitting in decision tree learning. These can be grouped into two classes:

- approaches that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data,
- approaches that allow the tree to over-fit the data, and then post prune the tree.

Although the first of these approaches might seem more direct, the second approach of post-pruning over-fit trees has been found to be more successful in practice. This is due to the difficulty in the first approach of estimating precisely when to stop growing the tree. Regardless of whether the correct tree size is found by stopping early or by post-pruning, a key question is what criterion is to be used to determine the correct final tree size. Approaches include:

- Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree.
- Use all the available data for training, but apply a statistical test to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set.
- Use an explicit measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized. This approach is based on a very deep theoretical principle called the *Minimum Description Length principle*.

The first of the above approaches is the most common and is often referred to as a training and validation set approach. In this approach, the available data are separated into two sets of examples: a **training set**, which is used to form the learned hypothesis, and a separate **validation set**, which is used to evaluate the accuracy of this hypothesis over subsequent data and, in particular, to evaluate the impact of pruning this hypothesis.

3.2 Incorporating Continuous-Valued Attributes

The initial definition of ID3 is restricted to attributes that take on a discrete set of values. First, the target attribute whose value is predicted by the learned tree must be discrete valued. Second, the attributes tested in the decision nodes of the tree must also be discrete valued. This second restriction can easily be removed so that continuous-valued decision attributes can be incorporated into the learned tree. This can be accomplished by dynamically defining new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals. In particular, for an attribute a that is continuous-valued, the algorithm can dynamically create a new Boolean attribute a_c that is true if $a < c$ and false otherwise. The only question is how to select the best value for the threshold c. Clearly, we would like to pick a threshold, c, that produces the greatest information gain. By sorting the examples according to the continuous attribute a, then identifying adjacent examples that differ in their target classification, we can generate a set of candidate thresholds midway between the corresponding values of a. It can be shown that the value of c that maximizes information gain must always lie at such a boundary. These candidate thresholds can then be evaluated by computing the information gain associated with each. The information gain can then be computed for each of the candidate attributes, and the best can be selected. This dynamically created Boolean attribute can then compete with the other discrete-valued candidate attributes available for growing the decision tree.

3.3 Handling Training Examples with Missing Attribute Values

In certain cases, the available data may be missing values for some attributes. For example, in a medical domain in which we wish to predict patient outcome based on various laboratory tests, it may be that the lab test Blood-Test-Result is available only for a subset of the patients. In such cases, it is common to estimate the missing attribute value based on other examples for which this attribute has a known value. Consider the situation in which $Gain(S, A)$ is to be calculated at node n in the decision tree to evaluate whether the attribute a is the best attribute to test at this decision node. Suppose that $< x, c(x) >$ is one of the training examples in S and that the value $a(x)$ is unknown, where $c(x)$ is the class label of x. One strategy for dealing with the missing attribute value is to assign it the value that is most common among training examples at node n. Alternatively, we might assign it the most common value among examples at node n that have the classification $c(x)$. The elaborated training example using this estimated value for $a(x)$ can then be used directly by the existing decision tree learning algorithm. A second, more complex procedure is to assign a probability to each of the possible values of a rather than simply assigning the most common value to $a(x)$. These probabilities can be estimated again based on the observed frequencies of the various values for a among the examples at node n. For example, given a Boolean attribute a, if node n contains six known examples with $a = 1$ and four with $a = 0$, then we would say the probability that $a(x) = 1$ is 0.6, and the probability that $a(x) = 0$ is 0.4. A fractional 0.6 of instance x is now distributed down the branch for a = 1 and a fractional 0.4 of x down the other tree branch. These fractional examples are used for the purpose of computing information $Gain$ and can be further subdivided at subsequent branches of the tree if a second missing attribute value must be tested. This same fractioning of examples can also be applied after learning, to classify new instances whose attribute values are unknown. In this case, the classification of the new instance is simply the most probable classification, computed by summing the weights of the instance fragments classified in different ways at the leaf nodes of the tree. This method for handling missing attribute values is used in C4.5.

4 Decision trees: pros and cons

Decision trees are a very popular method for data mining, and as such, they have advantages and drawbacks. The strengths are:

- Decision trees are able to generate understandable rules.
- Decision trees perform classification of a new data without requiring much computation.
- Decision trees are able to handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

In term of complexity, if we assume to have only binary branching nodes leading to the choice of 1 label among n, the complexity of the process to classify a new data is in $log_2(n)$ which is quite good. But obviously, this is not the case that we usually get this lower bound of complexity. On the other hand, the weaknesses are:

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many class and relatively small number of training examples.
- Decision tree are computationally expensive to train (high complexity)
- The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights.
- Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.
- Decision trees do not treat well non-rectangular regions. Most decision-tree algorithms only examine a single field at a time. This leads to rectangular classification boxes that may not correspond well with the actual distribution of records in the decision space.

5 Conclusion

Decision trees are powerful and popular tools for classification and prediction. The attractiveness of decision trees is due to the fact that, in contrast to neural networks, decision trees represent decision rules. Rules can readily be expressed so that humans can understand the meaning or even directly used in a database query language like SQL so that records falling into a particular category may be retrieved. In addition to classifier development, there are works implementing control systems in a similar fashion. In this case, the decision trees are used to evaluate the current status of the environment and make decision upon what course of action to take. So, decision tree are not only useful for classification but as **decision tools** as well which is one of their main power. Obviously, a lot of variations around decision trees as presented in this document have been investigated, leading to better results and/or more efficient algorithms.

Weka is an open source free software implementing diverse decision tree strategies.

K-nearest neighbours

Abstract

K-nearest neighbours algorithms are simply based on common sense, without any sophisticated theoretical background. Despite their theoretical simplicity. k-nn-like algorithms still remain among the most effective classification tools at the moment.

1 k-nn in a nutshell

As with any data mining problem, we start with an initial set of data S which consists of various observations x of a given universe $\mathcal{U}$, for which a known property or class has been assigned. For instance, you are a smoker and your class or label is $HasCancer$. Further, for each of these observations x, a number of measurable features exists which describe various details of the observations: in other words, x is a vector of values $smoker$, age, $gender$, $weight$, etc (i.e. $x = (x_1, \ldots, x_i, \ldots, x_n)$).

The basic principle underlying k-nn is to consider that an individual is similar to the other individuals "close to" it. The main issue is then to define what is the meaning of "close to".

An idea would be to consider a distance which would give us a numerical measure of the closedness of 2 individuals. Let us assume that on our representation set X we have a distance d. The k-nn algorithm proceed as follows when a new individual x has to be classified:

- compute the distance $d(x, y)$ for each $y \in S$
- retain the k nearest neigbours of x
- allocate to x the label which is the most frequent among the k nearest neigbours of x

2 Distances

There is a huge amount of literature to define a distance and to choose the most well-suited to the current problem: Euclidean, Hamming, Manhattan, etc.... Have a look on the literature to get an idea.

3 k-nn and weights

With the above classification procedure, every neigbour among the k-nearest ones, has the same voting power, whatever its distance to x. It could be fair to consider that the accuracy of a neighbour y to classify x is inversely proportional to it distance from x, $d(x, y)$. That is why it can improve accuracy to introduce a weighted k-nn procedure where each vote is done along with a weight (generally $\frac{1}{d(x,y)}$. The classification procedure is then slightly modified to take into account the weights. For each candidate class c, we compute the associated weight as the sum of the weights of every example labeled c. The winning class is the one having the highest weight. That's it...

It is amazing that this simple common sense reasoning procedure leads to surprisingly good results for a large panel of datasets. K-nn is often used as a benchmark to compare with other more sophisticated classification methods.

Bayesian classification

Abstract

The general framework of Bayesian classification is introduced in this chapter. This type of classifier is one of the most effective ones at the moment despite its theoretical simplicity.

1 Prediction and Bayesian inference

As with any data mining problem, we start with an initial set of data S which consists of various observations x of a given universe $\mathcal{U}$, for which a known property or class has been assigned. For instance, you are a smoker and your class or label is $HasCancer$. Further, for each of these observations x, a number of measurable features exists which describe various details of the observations: in other words, x is a vector of values $smoker$, age, $gender$, $weight$, etc (i.e. $x = (x_1, \ldots, x_i, \ldots, x_n)$).

From a practical viewpoint, the set S could be considered as a simple Excel table or a database, The ultimate problem involves developing a system by which the properties can be predicted from the features. A nice extension of this objective would be to use the classifier system to predict what we would expect the class to be for an observation which is not in our original data set S: in that case, our initial data set X is the one from which we learn and is often called **the training set** (or sampling set) and it is assumed that an example belongs to one class only (or that the classes are disjoint). We use the word training because we will use it to "train" an algorithm (as input for an algorithm). S consists of the current observations that we have about our universe $\mathcal{U}$. One has to understand that a data x i.e. a vector of features, can be associated to a lot of different individuals of $\mathcal{U}$: for instance, there is a lot of smokers having the same gender, the same age and the same weigth. On the other hand, S is just a finite part of the whole representation set X where every element of $\mathcal{U}$ is represented. Let us consider for instance a representation of a human being with only 2 real parameters $(weight, height)$. Then the real plan $\mathbb{R}^2$ is the representation of the mankind $\mathcal{U}$. It is quite clear that we have a lot of individuals with weigth between 40Kg and 100Kg and size between 1m and 2m. So the best way to have a mathematical representation of this fact is to add a probability measure on $\mathbb{R}^2$ i.e. on our representation set. The probability to get a patient whose weight is between 40 and 100 and height is between 1m and 2m is very very high... but the probablity to have a patient with size greater than 2m is not null! We understand that we have not a clear picture of this probability measure but this probability exists and is fixed.

Let the set of k possible class labels be denoted $Y = \{c1, c2, \ldots, ck\}$. Suppose that, for each class in our classification problem, we are given the conditional probability of the class given an observation. Let us note that this assumption is ENTIRELY irrealistic since we are not supposed to have a complete knowledge of our working universe (in the opposite case, there would be no need to predict !).

That is, we are given $p(y|x)$ for all possible values of y and x in S. Recall that in this scenario, the Bayes optimal classifier, i.e. the classifier that minimizes the expected probability of error for an observation x is the one assigning to x the label

$$y = argmax_{y \in Y} p(y|x) \text{ (MAP hypothesis)}$$

i.e. we allocate to x the most probable label.

Another way to put it, is to consider the set of labels Y as a set of potential cause for the observation x. For instance, if x is a vector of medical parameters related to a patient, we are looking for an explanation or hypothesis in Y to explain these values which could be $hascancer$, $isDrunk$, $isTooOld$, etc. Then the previous rule just tells us that we look for the most probable a posteriori hypothesis: this hypothesis is called the **Maximum A Postiori (MAP) hypothesis**.

What exactly does it mean to *minimize the probability of error*? Assume that a classifier implements a deterministic function $f(x)$ which returns a class label for any data vector x. An error on x means that the predicted label $f(x)$ is not equal to the real label y of x: $y \neq f(x)$. Obviously, we can check if we have an error on the training set, but we cannot check it on the non observed data. For instance, if we have in our record an individual x who is a smoker, then we can apply to x the function f predicting the label $hasCancer$ (yes/no) and check if we get the right answer. But for a new individual $z \notin S$, we apply f to z and we get a predicted label $hasCancer$ (yes/no) ... and we have no way to check if it is the right answer... we just pray !

- The true error set e_T is then $\{x \in X | f(x) \neq y\}$
- while the empirical error set is just $e_E = \{x \in S | f(x) \neq y\} = e_T \cap S$ (i.e. restricted to the errors made on the training set).

Then given a joint distribution of data vectors and labels, $p(x, y)$, there is a way to measure the error: this is the probability of error as

$$Prob(e_E) = \Sigma_{(x,y) \in S \times Y, f(x) \neq y} p(x, y)$$

If we have a continuous distribution, this probability is null. So it only makes sense to compute it with a discrete distribution. Following the above discussion, it is clear that we cannot compute $Prob(e_T)$ (can be estimated? can be borned ?). **No function $f(x)$ has lower probability of error than the Bayes optimal classifier**.

2 Probability estimation

Now if we are not given the values of the so-called posterior probabilities $p(y|x)$, we may estimate these values from the training data. There are many ways to do this, and there is continuing debate among statisticians and machine learning practitioners about what methods are best under various circumstances for estimating posteriors. For the moment, we will avoid the discussion of which method is best, and simply present some commonly used methods of posterior estimation.

First of all, a vocabulary detail: both $p(x|y)$ and $p(y|x)$ are conditional probabilities and, from a purely mathematical viewpoint, they are exactly the same type of object. Nevertheless, in this machine learning approach, x is considered as an input and y as the output (the thing we have to predict). So given x, $p(y|x)$ is the probability that the item represented with x belongs to the class y, but $p(x|y)$ has a different meaning and is called the likehood of x knowing y... This number answers the question "How realistic it is to have the value x when you belong to the class y".

One method for estimating the posteriors is to first estimate the class likelihoods $p(x|y)$ as $p'(x|y)$ and the priors $p(y)$ as $p'(y)$ from the training data, and then to use Bayes rule to compute an estimated value of the posterior with

$$p'(y|x) = \frac{p'(x|y) \times p'(y)}{p'(x)} =$$

$$\frac{p'(x|y) \times p'(y)}{\Sigma_{z \in Y} p'(x|z) \times p'(z)}$$

The simplest way to estimate is to count and to consider that the frequencies are just the probabilities which is generally wrong especially if the training set is small. For instance, $p'(y)$ is just the proportion of elements with label y in S. Doing that way, we are able for a given new data x to allocate the MAP label as

$$y = argmax_{y \in Y} p'(y|x) = argmax_{y \in Y} (\frac{p'(x|y) \times p'(y)}{p'(x)} =$$

$$argmax_{y \in Y} (p'(x|y) \times p'(y)) \text{ since } p'(x) \text{ is constant in that context}$$

As mentioned above, when our training data is drawn from conditional distributions of the form $p(x|y)$ (Case 2) rather than from the same distribution $p(x, y)$ as the test data (Case 1), we cannot estimate the priors from the training data, but need to obtain a prior from somewhere else. A commonly used method is to assume the prior probabilities $p(y)$ are all equal, and hence equal to $\frac{1}{k}$, where k is just the number of classes. Because we have no other knowledge of the distribution, we assume the classes are equiprobable! For instance, if we have 3 classes A, B and C, we consider the prior for each class as being $1/3$.

It is interesting to note, that in that case, the above formula can be simplified to

$$y = argmax_{y \in Y} p'(x/y) \text{ since the } p'(y) \text{ are identical (MLP hypothesis)}$$

leading to what is known as the Maximum Likehood Principle (MLP) since we have only likehood in the formula. We have moved from the MAP hypothesis to the MLP hypothesis.

3 Estimating likelihoods from training data

Just as there are many ways of estimating posteriors, there are many ways of estimating likelihoods from training data. For a full discussion, see any statistical textbook on the topic of estimation. We shall start with one of the simplest estimators of the likelihood, which is just the frequency of x in the training set given a particular value of the label y. That is,

$$p'(x|y) = \frac{\#(z \in S | z = x, z \in y)}{\#\{z \in S | z \in y\}}$$

For instance, we estimate the likehood to be a smoker (parameter=x) when we have a cancer (label=y) as the number of smokers having cancer divided by the total number of people having cancer among the observable data S. This is a very raw approximation which works relatively well in general.

Let us consider a simple standard example. Suppose we wish to have a procedure to decide whether a patient has cancer based upon the outcome of a certain procedure which tests for the presence of a particular antibody in the blood stream. For training data, suppose we perform the test on 100 cancer patients and 200 healthy subjects, having a total of 300patients (training set S). Assume that 90 of the cancer patients tested positive for the antibody and 40 of the healthy subjects tested positive. Then we could estimate the following likelihoods.

1. $p(test = positive|cancer) = 0.9$ (true positive)
2. $p(test = negative|cancer) = 0.1$ (false negative)
3. $p(test = positive|nocancer) = 40/200 = 0.2$ (false positive)

4. $p(test = negative|nocancer) = 160/200 = 0.8$ (true negative)

Should we assume that the subjects were drawn from the general population and that the prior probability of having cancer is $100/300 = 1/3$? Probably not. More likely, it makes sense to establish the prior from another study in which the rate of cancer in the general population is estimated. Lets say this rate is estimated at $p(cancer) = 0.001$, i.e., about 1 in a thousand people have cancer.

Then Bayes rule allows us to combine the likelihoods and the priors to obtain

$$p(cancer|test = positive) = \frac{0.9 \times 0.001}{0.9 \times 0.001 + 0.2 \times 0.999} = 0.005$$

Notice the result, which some people find quite surprising. Even testing positive, while the chance of having cancer has approximately doubled, the chance of having cancer is still small (0.005). This, of course, is because of the extremely low prior probability of having cancer, and the fact that the test is not very discriminative. Strangely, enough, this means we would never classify a patient as having cancer using Bayes rule and this particular test, if our goal was to minimize error.

It is very important to note that in medical diagnosis, the goal may not be to strictly minimize the number of errors, since one type of error (declaring a sick patient healthy) may be far more costly than another type of error (declaring a healthy patient sick). Since errors do not have the same cost, the doctor may want to skew the decision to reduce the number of costly errors even if the total error rate goes up. Instead of the above notion of error, we should go for more general notion known as risks or costs. This is another subject!

4 Naïve Bayesian classification

4.1 Conditional independence hypothesis

The algorithm to classify a new data is more or less a consequence of what has been explained above and a (more or less) direct translation of the Bayes formula, as soon as we have computed (offline) all the conditional probabilities (i.e. we get a table of conditional probabilities). Nevertheless, there is an implementation of Naive Bayes leading to what is know as a Bayesion network.

To understand the picture, let us start with a classical example where a shop has to consider if a new customer x is likely to buy a computer and so if we have to send a sale assistant to help him. Obviously, the shop got records of previous customers and loans which are summarized in Table 1. Now, the question is to consider this new customer with the following

Table 1: Bank data

Age	Income	Student	Credit_rating	Buys_computer
<=30	High	No	Fair	No
<=30	High	No	Excellent	No
31...40	High	No	Fair	Yes
>40	Medium	No	Fair	Yes
>40	Low	Yes	Fair	Yes
>40	Low	Yes	Excellent	No
31...40	Low	Yes	Excellent	Yes
<=30	Medium	No	Fair	No
<=30	Low	Yes	Fair	Yes
>40	Medium	Yes	Fair	Yes
<=30	Medium	Yes	Excellent	Yes
31...40	Medium	No	Excellent	Yes
31...40	High	Yes	Fair	Yes

characteristics (where "youth" means $age \leq 30$):

$$x = (age = youth, income = medium, student = yes, credit\_rating = fair)$$

We apply the MAP principle trying to estimate (it is a binary classification problem) $p(buy\_computer = yes|x)$ and $p(buy\_computer = no|x)$, then predict and decide. There are 2 options now:

1. either we have in our database individuals with the same characteristics as x . In that case, it is an easy game to compute the 2 probabilities above and to apply MAP. We just consider the total number of individual similar to x, and among them we compute the frequency of individuals buying computer and individuals who do not buy computer. Then we get our 2 probabilities.

2. or we do not have any similar individual in our database. And this is the general case where we have no clue how to estimate the 2 above probabilities. In fact, using Bayes formula, we have to maximize $p(y|x) = p(x|y) \times p(y)$. If $p(y)$ is easy to estimate as a frequency starting from our records, it is not the same for the likehood $p(x|y)$ which is formally equals to $p(age = youth, income = medium, student = yes, credit\_rating = fair)|y)$. Since we have not observed x, we can just compute a probability, which is a sensible way to proceed.

One idea is then to breakdown this big expression using the conditional probability chain rule i.e.

$$p(age = youth|y) \times p(income = medium|age = youth, y) \times p(student = yes|age = youth, income = medium, y)$$

$$\times p(credit\_rating = fair|age = youth, income = medium, student = yes, y)$$

In that case, we rely on simpler probabilities over events which have been likely observed. The boring thing is that what we have suppressed from the left hand side of the bar | is now on the right hand side! A simpler estimation would be to consider that the parameters are **class conditional independent** because in that case we would just have:

$$p((age = youth, income = medium, student = yes, credit\_rating = fair)|y) =$$

$$p(age = youth|y) \times p(income = medium|y) \times p(student = yes|y) \times p(credit\_rating = fair|y)$$

and it is an easy game to estimate all these likehood using our records.

If one specific value of a parameter has not been observed in our data, the corresponding probability will be 0 (for instance $p(color = yellow)$ if we have no item with attribute color equals to yellow). This point can be addressed in different ways (allocate a small value to this probabability, remove this factor from the equation, etc.)
More formally, class conditional independence means

$$p(x|y) = \Pi_{i \in [1,n]} p(x_i|y)$$

Then we are done since everything can be estimated and we can predict (in that case) that this new customer will buy a computer! Let us investigate this simple idea a little bit more.

4.2 Bayesian networks

We can adopt a more general viewpoint where the attribute representing an individual of our universe $\mathcal{U}$ are random variables as it is the case for the label $y \in Y$. In that case, there is no need for now to distinguish the label from the other attributes and we have at our disposal n random variables $x_1, \ldots, x_n$. A Bayesian network is a graphic representation (direct acyclic graph) of the link between variables or attributes, together with tables of conditional probabilities. If we consider that $smoker$ is a cause for (or influences) $hasCancer$, then we create a link from $smoker$ to $hasCancer$. Then we have a directed graph where the nodes are the variables $x_1, ..., x_n$ appearing in our problem description. Consider the example (coming from the Internet) in figure 1 where we have 4 binary parameters (or variables) with obvious meaning. $Cloudy$ can cause $Rain$ or in the opposite case, we have to sprinkle the lawn. Ultimately, whatever the reason (rain or sprinkling), the lawn is Wet. But, it is not an obligation to rain when the sky is cloudy... there is only a probability of rain. The assumption made by a Bayesian network is that the value of a variable is independent of the values of its ancestors, given the values of its parents. This view is less simplistic than the one in the previous section, but still is a way to simplify our calculations. For instance, in our case, $p(W|R, S, C) = p(W|R, S)$ because W is independent of C given R and S. $p(C) = p(C|R) = p(C|R, W)$ just because C has no parent: this is the prior probablity of C.

Remember, that we have a data set S where, as soon as we have defined the structure of the graph, will help to estimate all the conditional probabilities needed in the graph. Then, to each node, is associated a conditional probability distribution that we estimate via the observable data set S and is represented as a table when we have for each variable a finite number of options (in our case, we have 2 options per variable). When variables can take a lot of distinct values (still discrete), we need a computer to build up the tables.

In any case, the sum of the values of a given row should be equals to 1 since we have probability distributions.

These tables allow to compute the complete join distribution since:

$$p(C, S, R, W) = p(C) \times p(S|C) \times p(R|C, S) \times p(W|C, S, R)$$

and due to the independance assumption:

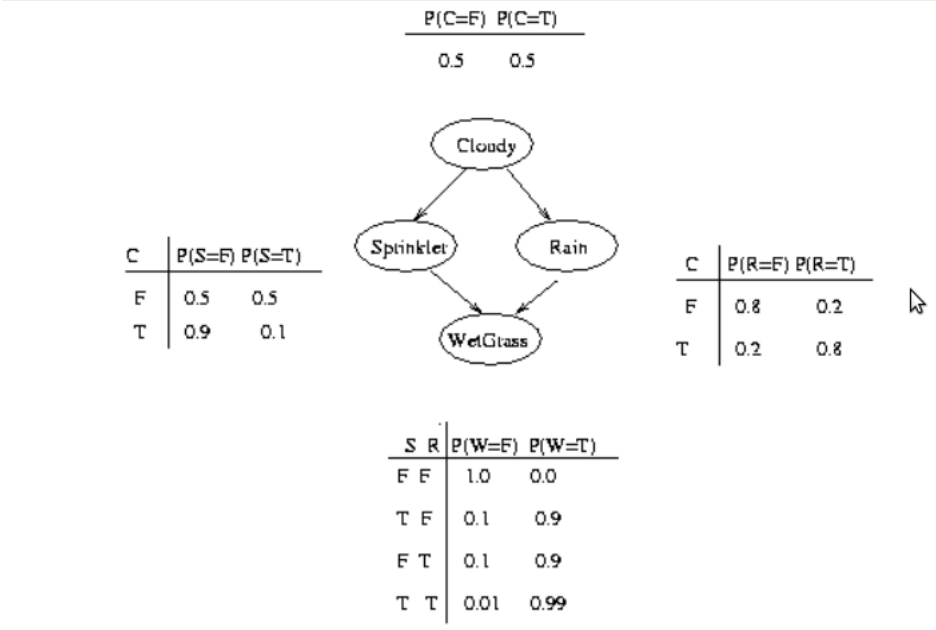
$$= p(C) \times p(S|C) \times p(R|C) \times p(W|S, R) (1)$$

It is intuitively clear but let us put it formally with $p(R|C, S)$:

$$p(R|C, S) = \frac{p(R, C, S)}{p(C, S)} = \frac{p(R, S|C) \times p(C)}{p(C, S)} = \frac{p(R|C) \times p(S|C) \times p(C)}{p(S/C) \times p(C)} = p(R|C)$$

In the right hand side of the equality, we just have the conditional distributions appearing in the nodes of the graph: so any option is computable via the tables. This approach allows to apply the MAP principle for any event we would like to predict. And we can go bottom-up (diagnostic) or top-down (prediction).

Figure 1: A simple Bayesian network with 4 binary attributes/variables



- *bottom-up*: observation: the grass is wet - query: what is the cause?

  This is called **diagnostic** since it goes from effects to causes (like a doctor). So we have to compute $p(R = 1|W = 1)$ and $p(S = 1|W = 1)$, to compare the 2 probabilities and choose the bigger one to predict the cause.

  $$p(R = 1|W = 1) = \frac{p(R = 1, W = 1)}{p(W = 1)}$$

  where the numerator is equals to $\Sigma_{c,s} p(C = c, S = s, R = 1, W = 1)$, the denominator is equals to $\Sigma_{c,s,r} p(C = c, S = s, R = r, W = 1)$ and c, s, r are Boolean values $\{0, 1\}$. As previously explained, formula (1) allows to compute these 2 numbers thanks to the tables associated to the node.

- *top-down*: observation : it is cloudy - query: will the grass be wet?

  It is more of a **prediction** now. We have to compute $p(W = 1|C = 1)$ and $p(W = 0|C = 1)$ and this is the same story...

5 Naive Bayes classifier: pros and cons

The strengths are:

- In spite over-simplified assumptions, they often perform better in many complex real-world situations.
- They require a small amount of training data to estimate the parameters.
- They can facilitate learning about causal relationships between variables.
- It is easier to understand (a sparse set of) direct dependencies and local distributions than a complete joint distribution!

On the other hand, the weaknesses are:

- It can be difficult to reach an agreement on the structure with different experts
- It is not that easy to deal with continuous data. We have to discretize.

- They are prone to what is known as Berkson'paradox. When 2 independent causes like S et R compete to explain an observation W, they become conditionaly dependent given their common child leading to some paradox like: $p(S = 1|W = 1, R = 1) \leq p(S = 1|W = 1)$ despite the fact that the 2 probabilities should be the same due to the independence of S and R. In fact

$$p(S = 1|W = 1, R = 1) = \Sigma_c p(C = c, S = 1, W = 1, R = 1) \leq$$

$$\Sigma_{c,r} p(C = c, S = 1, W = 1, R = r) = p(S = 1|W = 1)$$

  Roughly speaking, if 2 causes are sufficient to explain an event, and we observe the event and one of the 2 causes, then the probability of the other cause decreases...

6 Conclusion

A classifier that assumes variables are independent is called a **Naive Bayes classifier**, and the impressive performance of these classifiers shows that, in many cases, the loss from assuming independence may be overcome by the gain that comes from estimating probabilities separately. In real life applications, it is quite rare that attributes are independent but it does not affect the performance of Naive Bayes which is, despite its theoretical simplicity, among the best classification tool nowadays.

Weka is an open source free software implementing diverse Naive Bayesian strategies. Bayesian networks allow long discussion about the difference between correlation and causation x is correlated to y does not mean "if x occurs then y occurs" but simply $p(y|x) \geq p(y)$...

La complexite de Kolmogorov et quelques applications

G. Richard

September 18, 2014

1 Introduction

Kolmogorov.. tout le monde connait... Le type qui a donne les fondements axiomatiques de la theorie des probabilites. Mais il s'est aussi interesse a la notion d'information .... DE quoi parle t on ? disons de chaines de caracteres ou meme plus simplement de chaines de 0 et 1, c'est finalement la meme chose.

Etant donnee une chaine $s = 010101010101...$ avec 10000 fois 01, quelle est l'information qu'elle contient? Ou plutot quelle quantite d'information contient elle? Pas grand chose finalement meme si elle est tres longue. Et puis surtout, il est tres facile de deviner la suite (et deviner la suite simplement en regardant le debut... c'est aussi une definition d'apprendre!) Regardons celle la: $s = 11100000110010101000001$..... pas facile de deviner la suite.... On doit développer un peu.
On part d'une machine de Turing à 3 rubans (1 clavier Input, 1 ecran Output et de la mémoire pour mettre le programme). Appelons cela un calculateur et notons le U. Facile à imaginer. Quand on lance la machine U avec un programme p et une chaîne d'entrée y sur le ruban Input, deux comportements possibles: soit U ne s'arrête jamais et alors là, on est cuit... soit U s'arrête. Dans le cas où U s'arrête, on peut alors regarder l'écran Output et voir ce qu'il y a. Il y a une chaîne disons x: on notera $U(p, y) = x$. On peut dire que le programme p a permis de (re)construire la chaîne x avec l'aide de y. Bien sûr, comme on ne manipule que des chaînes de 0 et 1, distinguer programme, input, output est juste une manière de parler qui permet de comprendre mais finalement on pourrait tout aussi bien parler simplement de chaînes. Bon, on y est presque:

Definition 1 *Etant donnée deux chaînes finies (de 0 et 1) x et y, on note $K(x/y)$ le nombre entier $min\{|p|$ such that $U(y, p) = x\}$.*

En gros $K(x/y)$ est la taille du (d'un) plus petit programme qui affiche x puis s'arrête en ayant y en entrée. On peut dire "qui produit x avec l'aide de y". Et si on ne s'aide de rien i.e. $y = \epsilon$, alors $K(x/\epsilon)$ est une quantité intrinsèque à x: on note $K(x)$ et on parle de complexité de Kolmogorov ou complexité algorithmique de la chaîne x. Intuitivement, $K(x)$ est la longueur minimale de la chaîne nécessaire et suffisante pour produire la chaîne x: en quelque sorte, c'est une mesure du contenu en information de x et aussi une mesure de la compression minimale que l'on peut obtenir de x. On

pourrait presque dire qu'un tel programme p de longueur $K(x)$ est une cause de x et que c'est la courte cause possible.

Autre manière de dire, la taille de $gzip(x)$ est une borne supérieure de $K(x)$... Est ce une borne précise?... c'est une autre histoire;-)

Intuitivement encore, on comprend que la longueur d'une chaîne est sans rapport avec sa complexité puisque un programme court $for\ i = 1\ to\ 1000\ do\ \{print0; print1\}$ produit la chaine du début $s = 010101....$ Finalement $K(x) \equiv |x|$ est une assez bonne définition de l'aléatoire: x est aléatoire si x n'est pas compressible! on ne retrouve aucun pattern, on ne peut pas prévoir ses digits... même si on en connait les 10^9 premiers...on ne peut pas raccourcir sa définition. Evidemment si ce nombre n'existe pas, alors la complexité de Kolmogorov de x reste indéfinie. Regardons quelques propriétés simples de ce nombre quand il existe.

Propriétés 1

- *Premièrement, comme $K(x)$ est une sorte de limite inférieure idéale, c'est un nombre non calculable au sens de Turing. Autre manière de dire: quand on a un programme produisant x, on ne peut jamais être sûr que c'est le plus court! Mais on pourra peut être approximer $K(x)$. Si on considère l'ensemble de programmes de longueur $K(x)$ produisant x, on peut l'ordonner lexicographiquement par exemple, noter p^* le plus petit dans cet ordre: dans ce cas on a exactement $K(x) = |p^*|$.*

- *Et si on change de machine de Turing universelle U' au lieu de U... alors $K_U(x)$ est probablement different de $K'_U(x)$. Oui bien sûr MAIS la différence entre les 2 complexité est simplement une constante qui ne dépend que de U et U'. Donc quand on travaille sur des chaînes très complexes, c'est négligeable.*

  ...voir cours et discours sur la thèse de Church et les machines de Turing universelles.

- *Evidemment, le programme $print(x)$ produit la chaîne x sans aucune aide... donc sa taille $|print(x)|$ est une borne supérieure de $K(x)$. Quelle est la taille de $|print(x)|$? c'est quelque chose comme $|print|$ + $|x|$ où $|print|$ est la taille du programme $print$ sur la machine de Turing universelle que nous considerons. Ceci nous dit que $K(x) \leq |x| + K_U$ où K_U est une constante dépendant uniquement de la machine U.*

- *Soit n un entier, on a 2^n chaînes de longueur n. Or on ne dispose que de $2^n - 1$ programmes de longueur inferieure strictement a n (c'est simplement la somme des $2^p, p \in [0, n-1]$). Donc parmi ces chaînes de longueur n, il y en a au moins 1 dont la complexité de Kolmogorov est au moins n.*

- *Continuons un peu sur cette voie... Si on s'interesse aux programmes de longueur inférieure strictement a $n-1$, on en a $2^{n-1} - 1$, donc parmi nos chaînes de longueur n, il y en a au moins $2^n - (2^{n-1} - 1) = 2^{n-1} + 1$ qui ont une complexité supérieure a $n-1$...soit en gros la moitié de nos 2^n chaînes (si n est grand)*

- *Allons encore plus loin, et remarquons qu'il y a au plus 2^{n-10} programmes de longueur strictement inférieure a $n-10$... OK, donc il y a au plus 2^{n-10} chaines*

de longueur n et dont la complexité est inférieure strictement a $n - 10$... soit à peine 0.1% de nos chaînes...

- *Conclusion1: il n'y a pas assez de programmes;-) Conclusion2: ya pas tellement de chaînes vraiment compressibles Conclusion3: l'univers des chaînes est une vraie jungle...*

Un petit raisonnement intuitif encore une fois: si on considère une machine de Turing U_{zip} qui implémente le programme $unzip$. Alors si on considère comme programme p d'entrée pour cette machine la chaîne $p = zip(x)$, ce programme va s'exécuter, s'arrêter et afficher x à l'écran (ruban de sortie). DONC... la taille de p c'est à dire la taille de $zip(x)$ est une (bonne?) approximation de $K(x)$ sur la machine de Turing en question. Amazing! Le même raisonnement s'applique pour tout algorithme de compression/décompression (rar, jpeg pour les images, mp3 pour le son,etc.).

2 Distance informationnelle

Naturellement on se pose la question: "on fait quoi avec ca?" Pour l'instant, rien de concluant. Si on etait capable de construire quelque chose d'utile pour machine learning, on serait heureux. Et il y a un concept tres simple qui est extremement utile: la notion de distance. Peut on, à partir de K, construire une distance? OUI absolument. Regardons comment.

Premiere idee: finalement $p*$ permet de passer de y a x: en d'autres termes, il mesure l'effort que j'ai a faire (l'energie à depenser) pour aller de y a x. Pas mal mais est ce que je dois dépenser la même énergie pour le retour... en d'autres termes est ce que $K(x/y) = K(y/x)$ ... absolutely not except if... le même programme $p*$ satisfait la propriété $U(p^*, y) = x$ et $U(p^*, x) = y$ et que ca reste le plus petit (on parle de réversibilité). Mais alors, supposons qu'il existe un tel programme reversible, on pourrait considerer sa longueur comme la distance entre x et y:

Definition 2

$$E_0(x, y) = min\{|p| \textit{ such that } U(p, x) = y \textit{ and } U(p, y) = x\}$$

Elle est bien positive, symetrique par definition, $E_0(x, x) = 0$ (rien a faire) et $E_0(x, y) = 0$ implique $x = y$. Reste l'inegalite triangulaire: $E_0(x, y) \leq E_0(x, z) + E_0(z, y)$. On verra un peu plus tard.

Prenons un petit exemple en considerant 2 chaines de longueur n incompressibles (i.e. aleatoires). On a $K(x/y) = K(y/x) = n$. Mais alors si on considere la chaine $x \oplus y$ de longueur n, elle permet de passer de x a y et reciproquement simplement en faisant $x \oplus (x \oplus y) = y$ et $y \oplus (x \oplus y) = x$.

Deuxième idée: Pourquoi pas revenir a nos $K(x/y)$... le problème est que cela n'est évidemment pas symétrique puisque par exemple $K(\epsilon/x)$ est probablement très petit quel que soit x, mais pour un x incompressible, $K(x/\epsilon) = K(x)$ est de l'ordre de la taille de x. On peut alors décider, pour symétriser de choisir $K(x/y) + K(y/x)$ mais cela n'est pas très satisfaisant. En effet, si par miracle un programme reversible transforme x en y et minimise la taille, alors on compte 2 fois sa longueur comme étant la distance...Une autre idée est de prendre

Definition 3

$$E_1(x,y) = max\{K(x/y), K(y/x)\}$$

C'est intuitivement satisfaisant, c'est positif, c'est symétrique, ca vaut 0 seulement quand $x = y$ et l'inégalité triangulaire a voir... Evidemment par definition

$$E_0(x,y) \geq E_1(x,y)$$

Ya un truc marrant c'est que en fait on a (presque) l'inégalité inverse et donc l'égalité.... voir le resultat suivant:

Theorème 1 *(theoreme de conversion). Soit $K(x/y) = k_1, K(y/x) = k_2 \geq k_1, l = k_2 - k_1 \geq 0$. Il existe une chaine d de longueur l et une chaine q de longueur $k_1 + K(k_1, k_2) + \mathcal{O}(1)$ telles que: $U(q, xd) = y$ et $U(q, y) = xd$.*

Preuve: a bit technical...□

Cela veut dire que q est un programme réversible pour y et xd donc on aura $|q| \geq E_0(xd, y)$. Mais comme par construction $|q| = K(y/x) = max\{K(x/y), K(y/x)\} = E_1(xd, y)(\sim log)$ d'où

$$E_0(xd, y) = E_1(xd, y)(\sim log).$$

Il est tentant de considérer le programme qd comme permettant de passer de x a y dans les 2 directions. En gros ce prog, quand il part de x, il concatène d à x et applique q ensuite pour obtenir y. Et quand il part de y... en fait il faut rajouter au programme qd une sorte de conditionnelle au début permettant de choisir ce que l'on fait selon que l'on a x en entrée ou autre chose que x.

Le même raisonnement qu'avant s'applique prouvant que $E_0(x,y) = E_1(x,y)(\sim log)$.

Troisième idée: En fait, c'est pas facile de calculer cette distance ou de l'estimer à partir de cette définition. Regardons si on ne peut pas trouver quelque chose d'approchant mais qui utilise $K(x)$ et $K(y)$ directement. Considerons par exemple $K(x,y)$ et le programme p^* associe. On peut imaginer que p^* qui doit produire x et y est grosso modo de taille $K(x)$ + $K(y)$. Dans tous les cas, il ne peut pas etre de taille inferieure a $min(K(x), K(y)$ cela contredirait la def de K. En fait il est surement un peu plus petit car pour produire disons x suivi de y, il suffit de concatener un programme produisant y a un programme produisant x. Si on prend les 2 plus petits possible, on obtient un programme de longueur exactement $K(x) + K(y)$ qui produit le couple (x, y). Ce qui prouve que $K(x,y) \leq K(x) + K(y)$ tout simplement.

- Donc le nombre $m(x,y) = K(x) + K(y) - K(x,y)$ est positif.
- Que vaut $m(x,x)$? en gros $K(x)$ car pour produire (x,x) il suffit de faire tourner 2 fois le programme produisant x, donc $K(x,x)$ est a peu pres egal a $K(x)$.
- $m(x,y) = m(y,x)$ evidemment.
- Encore un petit raisonnement intuitif: en gros $K(x,y) = K(x) + K(y/x)(= K(y) + K(x/y))$ i.e. le programme doit produire x puis s'aider de x comme

auxilaire pour produire y... Supposons que x et y ne partagent aucune information commune, x n'apporte aucune aide pour produire y et donc $K(y/x)$ vaut à peu près $K(y)$. On a presque prouvé que $K(x,y) = K(x) + K(y)$ quand x et y n'ont pas d'information commune (de pattern commun) et donc $m(x,y) = 0$.

- Ceci nous permet de conclure que le nombre $m(x,y)$ est une bonne mesure du contenu commun en information de x et y. On l'appelle "information mutuelle" entre x et y.

3 Application

Voir mes slides et le cours. En gros avec la distance informationnelle, on peut appliquer toutes les methodes de classification qui utilisent des distances (k-plus proches voisins, SVM et vector spaces,...) et autres encore.

4 Mesure de Solomonoff

On peut dire que l'idee de Solomonof etait de s'attaquer au probleme des probabilites a priori souvent difficiles a cerner et pourtant necessaires pour appliquer la formule de Bayes qui relie probabilites a priori avec probabilite a posteriori. Le resultat de ses reflexions est simple: etant donnee une chaine de caracteres (ou bits 0 et 1), sa probabilite d'apparaitre ou d'etre generee par une machine de Turing universelle est:

Definition 4

$$p(x) = 2^{-K(x)}$$

Oh la, mais est on sur que l'on puisse parler de probabilite.... non en fait et on doit revenir un peu en arriere dans nos definitions. Si on considere un programme p generant x en un temps fini et s'arretant ensuite, alors quand il s'arrete, la machine de Turing a lu seulement une partie de ce programme, peut etre la totalite mais pas sur... Notons $pr(p,x)$ ce bout de programme et appelons le "programme reduit associe a p et x". On voit que finalement $K(x)$ n'est autre que la taille du plus petit progamme reduit produisant x.

Theorème 2 *p est une mesure de probabilite sur l'ensemble des chaines*

Preuve: Considerons l'ensemble des programmes reduits pour toutes les chaines finies x imaginables de X. Cet ensemble infini de chaines finies possede une propriete essentielle: il est prefix-free ... ca veut dire qu'il n'y a aucun element $p1$ de cet ensemble qui soit un prefixe d'un autre element $p2$ de cet ensemble... pourquoi? tout simplement ce que si $p2$ est un programme reduit produisant x et que $p2$ est un prefixe strict de $p1$, $p2$ ne peut en aucun cas etre un programme reduit lui meme. OK, bonne nouvelle et alors on fait quoi?

On peut associer a tout programme reduit $p = 11101100110$ par exemple le nombre reel en base 2: 0.11101100110. Ce nombre se trouve dans l'intervalle [0,1]. Completons ce nombre de toutes les manieres possibles (i.e. on colle derriere 0.11101100110

toutes les chaines finies ou infinies possibles). Alors on obtient un intervalle de [0,1] bien sur. Lequel? l'intervalle qui debute a 0.11101100110 et qui finit a 0.1110110011011111.... ou on complete a l'infini avec des 1. Et bien sur ce nombre n'est autre que 0.11101100111. Quelle est la taille de cet intervalle? tout simplement la difference entre ses 2 bornes qui n'est autre que $2^{-11} = 2^{-|p|}$.

Je note $[p]$ cet intervalle associe au programme reduit p. Ya un truc incroyable: si on prend 2 intervalles associes $[p1]$ et $[p2]$ à 2 programmes reduits $p1$ et $p2$, ils sont necessairement disjoints... why ??? tout simplement parce que s'ils avaient une intersection commune, cela signifierait qu'un des 2 p serait prefixe de l'autre.... voila la beaute de la chose...

Donc si on ajoute les longueurs de tous ces intervalles c'est à dire tous les $2^{|p|}$ pour p decrivant l'ensemble des programmes reduits, on obtient necessairement un nombre plus petit que 1 tout simplement parce que ces intervalles sont disjoints et inclus dans [0,1] ... dont la longueur est justement 1. Quel rapport avec le problème du début? Si on considere uniquement les $2^{-K(x)}$ pour x decrivant l'ensemble des chaines finies, cela revient a ne considerer que les programmes reduits p pour lesquels il existe x telle que $|p| = K(x)$ et donc la somme des $2^{-K(x)}$ sera évidemment encore plus petite;-) On parle de l'inégalité de Kraft (en gros cette inégalité nous dit quand est ce que l'on est sûr de pouvoir construire un ensemble prefix free pour coder des chaînes par exemple en donnant une contrainte sur la longueur de ces codes).□

5 Applications

Voir mes slides et le cours. En gros, la section precedente montre que l'on peut passer de K a une distribution de probabilite... donc si on a une distribution de probabilite, on peut calculer K simplement avec le log inverse $K(x) = -log_2(p(x))$... et ensuite on recommence comme avant puisque l'on a une distance...

6 Conclusion

La complexité de Kolmogorov a été utilisé en finaces, en biologie, spam detection, intrusion detection systems, etc... C'est un outil extremement puissant. You know what? Il y a encore plein de choses a dire, a lire aussi. L'idee initiale est non seulement geniale mais aussi avec des kilos d'applications pratiques. Par exemple, de détecter si 2 personnes ont copié le même texte (plagiat!). Cherchez sur le net le logiciel findFraud (sur le site www.complearn.com). Vladimir Vapnik a dit un jour quelque chose comme "ya rien de plus pratique qu'une bonne theorie"... Il avait raison finalement...